Durham University

Durham E-Theses

Digital mixing consoles: parallel architectures and taskforce scheduling strategies

Ken N. Linton

How to cite:

Linton, Ken N. (1995) Digital mixing consoles: parallel architectures and taskforce scheduling strategies. Doctoral thesis, Durham University.

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a https://etheses.durham.ac.uk/id/eprint/5371/ is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full Durham E-Theses policy for further details.

Academic Support Office, The Palatine Centre, Durham University, Stockton Road, Durham, DH1 3LE
e-mail: e-theses.admin@durham.ac.uk Tel: +44 0191 334 6107
https://etheses.durham.ac.uk

Digital Mixing Consoles: Parallel Architectures and Taskforce Scheduling Strategies

Ken N Linton, BSc

The copyright of this thesis rests with the author.
No quotation from it should be published without
his prior written consent and information derived
from it should be acknowledged.

Submitted in partial fulfilment of the degree
of Doctor of Philosophy

University of Durham, 1995

UNIV DUNELM

17 JAN 1996

Declaration

I, the undersigned, hereby declare that the work reported in this thesis, unless otherwise stated, has been undertaken solely by the candidate and that it has not previously been submitted by the candidate for any other degree in this or any other university.

Ken N Linton

Statement of Copyright

The copyright in this thesis rests with the author. No quotation from it should be published without his prior written consent and information derived from it should be acknowledged.

Acknowledgements

Many people have played a part in the production of this thesis. I would like to thank Dr Peter D Manning (Music School), Dr Alan Purvis (School of Engineering), and Peter Eastty (SSL) for initiating this interdisciplinary research project. For sound academic advice and nurturing my ideas, I am especially grateful to my industrial supervisor, Dr Stephen Terepin (SSL).

My thanks are also due to industry experts including Chris Jenkins (*Product Director*, SSL Ltd.), Mike Scuffham (Digital Developments, JMP Ltd.), and David Pope (previously of Neve Electronics Intl. Ltd.). Valuable discourse regarding both technological and conceptual issues has helped maintain a propitious commercial perspective throughout this work.

The research presented here would not have been possible without the efforts of University personnel including Peter Friend, Ian Hutchinson and Trevor Nancarrow. Colleagues from the *Centre for Music Technology* at the University of Glasgow and Banque Paribas' *Swaps and Options Research Team* also deserve special mention in this regard.

For agreeable camaraderie, I am most grateful to my colleagues Dr G Lee Gould and Dr Norman Powell. Dr Graham H Cross, Dr David Gray and Michael Webster deserve recognition for fine hospitality and *quare craic*. Finally, I would like to acknowledge my gratitude to the examiners for their recommendations upon reading the original manuscript.

Research detailed here was funded by the Department of Education for Northern Ireland, Solid State Logic Ltd. and the University of Durham. The author gratefully acknowledges additional grants from the Fellowship of Engineering and the Audio Engineering Society Educational Foundation.

Abstract

This thesis is concerned specifically with the implementation of large-scale professional DMCs. The design of such multi-DSP audio products is extremely challenging: one cannot simply lash together *n* DSPs and obtain *n*-times the performance of a sole device. M-P models developed here show that topology and IPC mechanisms have critical design implications.

Alternative processor technologies are investigated with respect to the requirements of DMC architectures. An extensive analysis of M-P topologies is undertaken using the metrics provided by the *TPG* tool. Novel methods supporting DSP message-passing connectivity lead to the development of a hybrid audio M-P (*HYMIPS*) employing these techniques.

A DMC model demonstrates the impact of task allocation on ASP M-P architectures. Five application-specific heuristics and four static-labelling schemes are developed for scheduling console taskforces on M-Ps. An integrated research framework and *DCS* engine enable scheduling strategies to be analysed with regard to the DMC problem domain.

Three scheduling algorithms — CPM, DYN and AST — and three IPC mechanisms — FWE, NSL and NML — are investigated. Dynamic-labelling strategies and mix-bus granularity issues are further studied in detail. To summarise, this thesis elucidates those topologies, construction techniques and scheduling algorithms appropriate to professional DMC systems.

Contents

List of Figures xv

List of Tables xx

Chapter 1 Introduction **1-1**

1.1 Background 1-1

1.1.1 Historical Overview 1-1
1.1.2 The Recording Studio 1-1
1.1.3 Production Techniques 1-2

1.2 Digital Mixing Consoles 1-2

1.2.1 Design Goals 1-2
1.2.2 Audio Performance 1-2
1.2.3 Control Surface Ergonomics 1-3

1.3 Parallel Architectures 1-3

1.3.1 Multi-Processors 1-3
1.3.2 Processing Technology 1-4
1.3.3 IPC Topologies 1-4
1.3.4 M-Ps and DMCs 1-4

1.4 Task Scheduling Strategies 1-5

1.4.1 Introduction 1-5
1.4.2 Partitioning 1-5
1.4.3 Complexity 1-5
1.4.4 Opportunities 1-6

1.5 Thesis Overview 1-6

1.5.1 Digital Mixing Consoles 1-6
1.5.2 Parallel Architectures 1-7
1.5.3 Taskforce Scheduling Strategies 1-7
1.5.4 Contribution & Significance 1-7

1.6 Conclusions 1-8

1.7 References 1-8

Chapter 2 Analogue Mixing Consoles **2-1**

2.1 Audio Signal Processing 2-1

2.1.1 Frequency Spectrum 2-1

2.1.2 Dynamic Range 2-3
2.1.3 Time-Domain Processing 2-4
2.1.4 Spatial Localisation 2-5

2.2 Console Terminology 2-8

2.2.1 Multitrack Recording Techniques 2-8
2.2.2 Two-Channel Console 2-10
2.2.3 Simple Multi-Channel Console 2-11
2.2.4 Metering 2-12
2.2.5 Free-Grouping 2-12

2.3 Professional Console Features 2-13

2.3.1 Input/Output Module 2-13
2.3.2 Centre Section 2-15
2.3.3 Master Routing and Signal Flow 2-16
2.3.4 VCA Grouping 2-17

2.4 Console Automation 2-17

2.4.1 Automated Mix-Down 2-18
2.4.2 Beyond Fader Automation 2-19
2.4.3 *Total Recall* 2-20

2.5 Conclusions 2-20

2.6 References 2-21

Chapter 3 Console Digitalisation 3-1

3.1 Discrete-Time Signal Processing 3-1

3.1.1 Sampling 3-1
3.1.2 Quantisation 3-2
3.1.3 Data-Domain Conversion 3-4
3.1.4 Comparison of Processing Domains 3-5

3.2 Digital Audio Processing 3-6

3.2.1 Matrix Representation 3-7
3.2.2 Processing Redundancy 3-10
3.2.3 Audio Performance 3-11

3.3 Control and Display Processing 3-12

3.3.1 Control Rate vs. Audio Rate 3-12
3.3.2 Control & Display Response 3-13
3.3.3 Resolution & Psychoacoustics 3-13
3.3.4 Automation 3-13

3.4 Surface Ergonomics 3-14

3.4.1 Overview 3-14
3.4.2 Control Layout 3-15
3.4.3 Display Layout 3-16
3.4.4 Long vs. Virtual Surfaces 3-17

3.5 Assignable Mixing 3-17

3.5.1 Overview 3-17
3.5.2 Implementation Issues 3-18
3.5.3 Disadvantages of Assignable Faders 3-19
3.5.4 The Virtual Console 3-19

3.6 Conclusions 3-21

3.7 References 3-22

Chapter 4 Programmable DSP Devices 4-1

4.1 Introduction 4-1

4.1.1 History of DSP Devices 4-1
4.1.2 General-Purpose vs. DSP Processors 4-1
4.1.3 Benchmarks 4-2

4.2 Arithmetic 4-2

4.2.1 Fixed-Point 4-2
4.2.2 Floating-Point 4-3

4.3 Memory 4-5

4.3.1 Parallel Memories 4-5
4.3.2 Demand Ratio 4-7
4.3.3 Internal and External Memory 4-7
4.3.4 Addressing Modes 4-8

4.4 Pipelining 4-10

4.4.1 Interlocking 4-10
4.4.2 Time-Stationary Coding 4-11
4.4.3 Data-Stationary Coding 4-11
4.4.4 Comparison of Coding Methods 4-12
4.4.5 Branching 4-12

4.5 Software Development Tools 4-12

4.5.1 Optimising C Compilers 4-12
4.5.2 Block-Diagram Environments 4-13
4.5.3 Software Simulators 4-13
4.5.4 Real-Time Operating Systems 4-14
4.5.5 West vs. East 4-14

4.6 Processing Technology and DMCs 4-14
4.6.1 Parallel-Bus DSPs 4-14
4.6.2 Serial-Bus DSPs 4-15
4.6.3 Application-Specific Integrated Circuits 4-16
4.6.4 Vector Processors 4-17
4.6.5 Parallel DSP Devices 4-19
4.7 Conclusions 4-21
4.8 References 4-23

Chapter 5 Architectural Issues 5-1
5.1 Background 5-1
5.1.1 Classification of Parallel Architectures 5-1
5.1.2 Processing Budget 5-2
5.1.3 Implementing Parallelism 5-3
5.1.4 Processor Pipelining 5-4
5.2 Multi-Processor Topologies 5-5
5.2.1 Single-Stage Networks 5-5
5.2.2 Analysing M-P Topologies 5-9
5.3 Topology Metrics 5-11
5.3.1 Overview 5-11
5.3.2 Expansion Metrics 5-11
5.3.3 Network Connectivity 5-12
5.3.4 Topology Diameter 5-13
5.3.5 Node Valency 5-14
5.3.6 Diameter.Valency Product 5-15
5.4 Comparison of M-P Topologies 5-16
5.4.1 Simple Topologies 5-16
5.4.2 Tree Topologies 5-17
5.4.3 Hypercubes 5-19
5.4.4 Preferred Topologies 5-21
5.5 Summary 5-23
5.6 References 5-24

Chapter 6 Audio M-P Implementation 6-1
6.1 Multi-DSP Configurations 6-1
6.1.1 Shared-Bus Design 6-1
6.1.2 Alternative Synchronization Mechanisms 6-2

6.2 Message-Passing IPC 6-4

6.2.1 Overview 6-4
6.2.2 Routing Schemes 6-5
6.2.3 Comparing Message-Passing M-P 6-6

6.3 Message-Passing Realisation 6-8

6.3.1 First-In First-Out Buffers 6-8
6.3.2 Dual-Ported RAM 6-9
6.3.3 DPR and FIFOs Compared 6-11
6.3.4 Novel Devices 6-12

6.4 *HYMIPS* — A Hybrid Audio M-P 6-13

6.4.1 Design Decisions 6-13
6.4.2 Message-Passing 6-15
6.4.3 Performance 6-16
6.4.4 SP-DIF Interfaces 6-17

6.5 Conclusions 6-21

6.6 References 6-24

Chapter 7 Multi-Processor Models 7-1

7.1 DMC Task Scheduling 7-1

7.1.1 Introduction 7-1
7.1.2 Scheduling Algorithms 7-1
7.1.3 Algorithm Constraints 7-2
7.1.4 Static vs. Dynamic Scheduling 7-4
7.1.5 Model Operation 7-5
7.1.6 Extensions 7-5

7.2 Modelling M-P Speed-Up 7-7

7.2.1 Introduction 7-7
7.2.2 Preliminary Conditions 7-7
7.2.3 Opportunities for SU Speed-Up 7-8
7.2.4 Scaled Speed-Up 7-9

7.3 Linear Processor Pipeline 7-10

7.3.1 Background 7-11
7.3.2 Speed-up 7-11
7.3.3 Efficiency 7-12
7.3.4 Throughput 7-13

7.4 Task Granularity 7-14

7.4.1 Introduction 7-14
7.4.2 Non-Overlapped IPC 7-14

7.4.3 Fully Overlapped IPC 7-17
7.4.4 Linear IPC Overhead 7-18
7.4.5 Multiple Communication Links 7-19

7.5 Conclusions 7-21

7.6 References 7-23

Chapter 8 Scheduling Strategies 8-1

8.1 Combinatorial Optimisation 8-1

8.1.1 Overview 8-1
8.1.2 Definitions 8-1
8.1.3 Encoding Schemes 8-2
8.1.4 Time Complexity Functions 8-3
8.1.5 Complexity of Task Scheduling 8-5

8.2 Stochastic Techniques 8-6

8.2.1 Simulated Annealing 8-6
8.2.2 Genetic Algorithms 8-8
8.2.3 Application to DCS 8-9

8.3 Deterministic Strategies 8-10

8.3.1 Graph Theory 8-10
8.3.2 Integer Programming 8-11
8.3.3 Clustering Techniques 8-11
8.3.4 Dynamic Programming 8-12

8.4 List Scheduling 8-14

8.4.1 Critical Path Method 8-14
8.4.2 Hu-Level Labelling 8-15
8.4.3 Alternative Labelling Schemes 8-16
8.4.4 Upper Bounds on Schedule Quality 8-16
8.4.5 Application to DCS 8-18

8.5 M-P Scheduling Anomalies 8-19

8.5.1 Idle Periods 8-19
8.5.2 List Scheduling 8-19

8.6 Conclusions 8-21

8.7 References 8-23

Chapter 9 DCS Research Framework 9-1

9.1 Software Environment 9-1

9.1.1 Hierarchical Console Design 9-2
9.1.2 Graphics Capture 9-2
9.1.3 Behavioural Models 9-3
9.1.4 Taskforce Simulation 9-3
9.1.5 Preliminary Results 9-4

9.2 Digital Console Scheduler 9-5

9.2.1 Overview 9-5
9.2.2 Scheduling Strategies 9-8
9.2.3 Control Parameters 9-9
9.2.4 Scheduler Outputs 9-12

9.3 Performance of AST 9-13

9.3.1 Time & Space Complexity 9-13
9.3.2 Algorithm Run-Time 9-13
9.3.3 State-Space Complexity 9-15

9.4 Performance of CPM 9-16

9.4.1 Algorithm Run-Time 9-16
9.4.2 Schedule Length 9-19
9.4.3 Speed-Up 9-20
9.4.4 Scaled Speed-Up 9-20

9.5 Conclusions 9-21

9.6 References 9-23

Chapter 10 Dynamic Labelling Schemes 10-1

10.1 Motivation 10-1

10.1.1 Alternative Labelling Schemes 10-1
10.1.2 Upper Bounds on Schedule Quality 10-2
10.1.3 Processor Thrashing 10-3

10.2 Performance with FWE 10-5

10.2.1 Algorithm Run-Time 10-5
10.2.2 Schedule Length 10-8
10.2.3 Speed-Up 10-8
10.2.4 Scaled Speed-Up 10-8

10.3 Performance with NSL 10-10

10.3.1 Algorithm Run-Time 10-10
10.3.2 Schedule Length 10-11
10.3.3 Speed-Up 10-12
10.3.4 Scaled Speed-Up 10-13

10.4 Performance with NML 10-13

10.4.1 Algorithm Run-Time 10-13
10.4.2 Schedule Length 10-14
10.4.3 Speed-Up 10-15
10.4.4 Scaled Speed-Up 10-15

10.5 Task Granularity 10-16

10.5.1 DMC Implementation 10-17
10.5.2 Algorithm Run-Time 10-18
10.5.3 Schedule Length 10-19
10.5.4 Speed-Up 10-21
10.5.5 Scaled Speed-Up 10-21

10.6 Conclusions 10-22

10.7 References 10-24

Chapter 11 Conclusions & Further Work 11-1

11.1 Digital Mixing Consoles 11-1

11.1.1 Analogue Mixing Consoles 11-1
11.1.2 Console Digitalisation 11-2

11.2 Parallel Architectures 11-2

11.2.1 Programmable DSP Devices 11-2
11.2.2 Architectural Issues 11-3
11.2.3 Audio M-P Implementation 11-3

11.3 Task Scheduling Strategies 11-4

11.3.1 Multi-Processor Models 11-4
11.3.2 Scheduling Strategies 11-5
11.3.3 DCS Research Framework 11-5
11.3.4 Dynamic Labelling Schemes 11-6

11.4 Further Work 11-6

11.4.1 Digital Mixing Consoles 11-6
11.4.2 Parallel Architectures 11-7
11.4.3 Task Scheduling Strategies 11-7
11.4.4 Manufacturing Considerations 11-8

11.5 Summary 11-9

11.6 References 11-10

Appendix A Survey of DSP Processors A-1

Appendix B Topology Expansion Metrics B-1

Appendix C Topology Analysis Tool C-1

C.1 Overview C-1

C.2 Definition Files C-2

C.2.1 tpg.def C-2

C.3 Include Files C-3

C.3.1 tpg_alg.h C-3
C.3.2 tpg_inf.h C-16
C.3.3 tpg_ini.h C-20
C.3.4 tpg_utl.h C-25
C.3.5 tpg_win.h C-27
C.3.6 tpg_xlo.h C-29

C.4 Source files C-31

C.4.1 tpg_alg.c C-31
C.4.2 tpg_inf.c C-65
C.4.3 tpg_ini.c C-74
C.4.4 tpg_utl.c C-80
C.4.5 tpg_win.c C-82
C.4.6 tpg_xlo.c C-85

Appendix D Digital Console Scheduler D-1

D.1 Overview D-1

D.2 Public Definition Files D-2

D.2.1 dcs_ast.pub D-2
D.2.2 dcs_clc.pub D-3
D.2.3 dcs_cpm.pub D-7
D.2.4 dcs_dyn.pub D-8
D.2.5 dcs_edb.pub D-9
D.2.6 dcs_glb.pub D-10
D.2.7 dcs_idb.pub D-12
D.2.8 dcs_lst.pub D-14
D.2.9 dcs_sch.pub D-24
D.2.10 dcs_trl.pub D-26
D.2.11 dcs_utl.pub D-28
D.2.12 dcs_wri.pub D-30

D.3 Private Definition Files D-33

D.3.1 dcs_ast.prv D-33
D.3.2 dcs_clc.prv D-37
D.3.3 dcs_cpm.prv D-39
D.3.4 dcs_dyn.prv D-40
D.3.5 dcs_edb.prv D-43
D.3.6 dcs_inf.prv D-46
D.3.7 dcs_lst.prv D-55
D.3.8 dcs_sch.prv D-56
D.3.9 dcs_utl.prv D-58
D.3.10 dcs_wri.prv D-62

D.4 Prolog Source Files D-63

D.4.1 dcs_ast.pro D-63
D.4.2 dcs_clc.pro D-69
D.4.3 dcs_cpm.pro D-74
D.4.4 dcs_dyn.pro D-76
D.4.5 dcs_edb.pro D-80
D.4.6 dcs_idb.pro D-84
D.4.7 dcs_inf.pro D-86
D.4.8 dcs_lst.pro D-97
D.4.9 dcs_sch.pro D-102
D.4.10 dcs_trl.pro D-107
D.4.11 dcs_utl.pro D-109
D.4.12 dcs_wri.pro D-114

Appendix E Original Publications E-1

E.1 Conference Papers E-1
E.2 Invited Papers E-2

Appendix F *ICASSP '91* Paper Reprint F-1

Appendix G AES UK DSP Paper Reprint G-1

Appendix H Glossary of Abbreviations H-1

Appendix I Bibliography I-1

List of Figures

Figure 2-1:	Classification of audio signal processing techniques.	2-1
Figure 2-2:	Range of control provided by one section of parametric equalisation.	2-2
Figure 2-3:	Example of effect of flanging on flat frequency spectrum.	2-5
Figure 2-4:	Effect of artificial reverberation unit on sound impulse.	2-6
Figure 2-5:	Example of typical console pan-pot characteristic.	2-7
Figure 2-6:	Outline arrangement for sum-and-difference panning.	2-8
Figure 2-7:	Typical arrangement of microphones around drum-kit.	2-9
Figure 2-8:	Signal flow diagram of a typical 2-channel mixing console.	2-10
Figure 2-9:	Signal paths in a typical *m*-channel console.	2-11
Figure 2-10:	Illustration of the free-grouping principle.	2-13
Figure 2-11:	SSL *G* Series I/O module Filters section.	2-14
Figure 2-12:	SSL *G* Series I/O module EQ section.	2-14
Figure 2-13:	SSL *G* Series I/O module Dynamics section.	2-15
Figure 2-14:	SSL *G* Series I/O module Cue and Aux Send section.	2-16
Figure 2-15:	SSL *G* Series Master Facilities Module and Centre Section.	2-17
Figure 2-16:	SSL *G* Series I/O module Large Fader section.	2-18
Figure 3-1:	Block schematic representation of generic DMC.	3-1
Figure 3-2:	Illustration of the aliasing phenomenon in the time domain.	3-2
Figure 3-3:	Illustration of the sampling process in the frequency domain.	3-3
Figure 3-4:	Quantisation noise shown as errors relative to original analogue...	3-4
Figure 3-5:	Schematic of the processes involved in analogue-to-digital...	3-5
Figure 3-6:	Schematic of the processes involved in digital-to-analogue...	3-6
Figure 3-7:	Time Division Multiplexing on a word-wide parallel bus.	3-7
Figure 3-8:	Simplified schematic of a mixing console during mixdown to stereo.	3-7
Figure 3-9:	Transposed direct form II, or transposed canonical, bi-quad section.	3-9
Figure 3-10:	Outline diagram of the assignable console concept.	3-18
Figure 3-11:	Outline drawing of the control layout of the *VSC60*.	3-20
Figure 4-1:	Peak performance of commercially available DSP devices.	4-2
Figure 4-2:	SQNR and dynamic range, shown as a function of signal level.	4-4

Figure 4-3: Basic Harvard architecture, as used in many early DSPs. 4-5

Figure 4-4: First modification — data stored in program memory. 4-5

Figure 4-5: Second modification — multi-ported data memory. 4-6

Figure 4-6: Third modification — instruction cache. 4-6

Figure 4-7: Fourth modification — third memory bank. 4-6

Figure 4-8: Fifth modification — several memory banks. 4-7

Figure 4-9: Visualisation of a circular buffer of length *m*. 4-9

Figure 4-10: Model of the pipelining within a programmable DSP. 4-10

Figure 4-11: Example MAC instruction, in Motorola DSP56001 assembly code. 4-11

Figure 4-12: Block diagram of the Motorola DSP56001. 4-15

Figure 4-13: Block diagram of the Intel i860. 4-18

Figure 4-14: Block diagram of the Inmos T801. 4-19

Figure 4-15: Block diagram of the TI TMS320C40. 4-20

Figure 5-1: Example speed-up curves for M-P DMC implementations. 5-3

Figure 5-2: Exploiting spatial parallelism. 5-3

Figure 5-3: Exploiting temporal parallelism. 5-4

Figure 5-4: Detail of overlapped processing in a *n*-stage linear pipeline. 5-4

Figure 5-5: Illustration of chordal ring with 6 nodes. 5-5

Figure 5-6: Illustration of 2-ary x-tree with 4 levels. 5-6

Figure 5-7: Illustration of 2-D nearest-neighbour mesh of width 4. 5-6

Figure 5-8: Illustration of 3-D spanning bus hypercube of width 3. 5-7

Figure 5-9: Illustration of 3-D dual bus hypercube of width 3. 5-7

Figure 5-10: Illustration of 2-D torus of width 4. 5-8

Figure 5-11: Illustration of 2-ary 3-cube. 5-8

Figure 5-12: Recursive creation of binary hypercubes of dimension *d*. 5-9

Figure 5-13: Illustration of 3-D cube-connected cycles. 5-9

Figure 5-14: $\delta \cdot \upsilon$ characteristics for simple topologies. 5-17

Figure 5-15: Link increments for tree topologies. 5-18

Figure 5-16: $\delta \cdot \upsilon$ characteristics of tree topologies. 5-18

Figure 5-17: Number of links for hypercube topologies. 5-19

Figure 5-18: Node connectivities for hypercube topologies. 5-20

Figure 5-19: $\delta \cdot \upsilon$ characteristics for hypercube topologies. 5-21

Figure 5-20: Node increments for preferred M-P topologies. 5-22

Figure 5-21: Node connectivities for preferred M-P topologies. 5-22

Figure 5-22: $\delta \cdot \upsilon$ characteristics for preferred M-P topologies. 5-23

Figure 6-1: Example shared-memory M-P architecture. 6-2

Figure 6-2: Flow chart of test-and-set and spin-lock semaphore operations. 6-4

Figure 6-3: Analysis of Equation (6-3), for various message head/size ratios. 6-6

Figure 6-4: IPC latency ratios for several topologies given constant node valency. 6-8

Figure 6-5: DSP-to-DSP communication using width-expanded DPR. 6-12

Figure 6-6: Dual-bus hybrid M-P architecture. 6-14

Figure 6-7: Multi-bus hybrid M-P architecture. 6-15

Figure 6-8: Timing diagram for *HYMIPS* SP-DIF interfaces. 6-17

Figure 6-9: Initialisation macro for *HYMIPS* SP-DIF peripherals. 6-18

Figure 6-10: Interrupt service routine for *HYMIPS* SP-DIF peripherals. 6-18

Figure 6-11: Schematic diagram of *HSR* SP-DIF peripheral. 6-20

Figure 6-12: Schematic diagram of *HST* SP-DIF peripheral. 6-22

Figure 7-1: Illustration of the DMC task scheduling problem. 7-2

Figure 7-2: Illustration of loop removal by sample period bisection. 7-4

Figure 7-3: Illustration of conventional speed-up for various values of w_p. 7-8

Figure 7-4: Illustration of conventional and scaled speed-up curves. 7-11

Figure 7-5: Speed-up characteristics for LPP M-Ps. 7-12

Figure 7-6: Efficiency characteristics for LPP M-Ps. 7-13

Figure 7-7: Non-overlapped IPC with 50 tasks, where $/c = 10 : k_{opt} = 0$ or m. 7-15

Figure 7-8: Non-overlapped IPC with 50 tasks, where $r/c = 40 : k_{opt} = m/2$. 7-16

Figure 7-9: Fully overlapped IPC execution of 50 tasks, where $r/c = 30 : \ldots$ 7-17

Figure 7-10: Speed-up characteristics with a single IPC link: $r/c = 200, \ldots$ 7-22

Figure 7-11: Speed-up characteristics with multiple IPC links: $r/c = 200, \ldots$ 7-23

Figure 8-1: Prolog source to generate $N(m, n)$, assuming no precedence. 8-4

Figure 8-2: Illustration of the *NP* class of optimisation problems. 8-5

Figure 8-3: Pseudo-code listing of the Metropolis Algorithm. 8-6

Figure 8-4: Sketch demonstrating the need for uphill moves. 8-7

Figure 8-5: Outline of GA procedure. 8-8

Figure 8-6: Task scheduling in terms of state-space searching. 8-12

Figure 8-7: Hu-level labelling of a taskforce consisting of unit-length tasks. 8-15

Figure 8-8: Upper bounds for CPM with unit task times. 8-18

Figure 8-9: Upper bounds for CPM with mutually commensurate task times. 8-19

Figure 8-10: Example taskforce to illustrate the idle time anomaly. 8-20

Figure 8-11: Schedules of the taskforce of Figure 8-8 to illustrate the idle time... 8-20

Figure 9-1: Block diagram of DCS research framework. 9-1

Figure 9-2: Symbol representing a 2-input bus-configured mix task. 9-2

Figure 9-3: Symbol representing *HYMIPS* Motorola DSP56001 module. 9-3

Figure 9-4: Module of HELIX behavioural models for DMC taskforce primitives. 9-4

Figure 9-5: DSP56001 assembly code for DMC EQ biquad kernel. 9-5

Figure 9-6: Example 4-input:1-output console taskforce. 9-6

Figure 9-7: Optimal allocation of example taskforce of Figure 9-6. 9-6

Figure 9-8: Pseudo-code of *DCS* CPM algorithm, using the HLE labelling scheme. 9-8

Figure 9-9: AST–FWE run-time for bus-type taskforces on a 2-DSP M-P. 9-14

Figure 9-10: AST–FWE run-time for tree-type taskforces on a 2-DSP M-P. 9-15

Figure 9-11: AST–FWE state-space for bus-type taskforces on a 2-DSP M-P. 9-16

Figure 9-12: AST–FWE state-space for tree-type taskforces on a 2-DSP M-P. 9-17

Figure 9-13: CPM–FWE run-time for bus-type taskforces on an 8-DSP M-P. 9-17

Figure 9-14: CPM–FWE $\log t$ for bus-type taskforces on an 8-DSP M-P. 9-18

Figure 9-15: CPM–FWE run-time for tree-type taskforces on an 8-DSP M-P. 9-19

Figure 9-16: CPM–FWE $\log t$ for tree-type taskforces on an 8-DSP M-P. 9-19

Figure 9-17: CPM–FWE speed-up for tree-type taskforces. 9-20

Figure 9-18: CPM–FWE scaled speed-up for tree-type taskforces. 9-21

Figure 10-1: Pseudo-code of *DCS* DYN algorithm, using HLE & TSK labelling... 10-3

Figure 10-2: DYN–FWE–HLE–NON schedule for 4-stage EQ section on a 2-DSP... 10-4

Figure 10-3: DYN–NSL–HLE–NON schedule for 4-stage EQ section on a 2-DSP M-P. 10-4

Figure 10-4: DYN–NSL–HLE–TSK schedule for 4-stage EQ section on a 2-DSP M-P. 10-5

Figure 10-5: DYN–FWE–HLE run-time for bus-type taskforces on a 16-DSP M-P. 10-6

Figure 10-6: DYN–FWE–HLE $\log t$ for bus-type taskforces on a 16-DSP M-P. 10-6

Figure 10-7: DYN–FWE–HLE run-time for tree-type taskforces on a 16-DSP M-P. 10-7

Figure 10-8: DYN–FWE–HLE $\log t$ for tree-type taskforces on a 16-DSP M-P. 10-7

Figure 10-9: DYN–FWE–HLE ω_{sch} for tree-type taskforces on a 16-DSP M-P. 10-8

List of Tables

Table 4-1:	DSPs representative of the most important architectural techniques.	4-3
Table 4-2:	Summary of memory systems for preferred DSPs.	4-8
Table 4-3:	Summary of addressing modes for preferred DSPs.	4-9
Table 4-4:	Outline of reservation table for DSPs that use time-stationary coding.	4-11
Table 5-1:	M-P topology physical characteristics.	5-12
Table 5-2:	M-P topology network connectivities.	5-13
Table 5-3:	M-P topology diameters.	5-14
Table 5-4:	M-P topology node valencies.	5-15
Table 5-5:	M-P topology diameter.valency products.	5-16
Table 6-1:	Selection of dual-ported RAMs available from Integrated Device...	6-10
Table 6-2:	Component list for *HSR* schematic.	6-19
Table 6-3:	Component list for *HST* schematic.	6-21
Table 8-1:	Analysis of alternative encoding schemes for DMC taskforces.	8-2
Table 8-2:	Example behaviour of GA crossover operator.	8-9
Table 9-1:	*DCS* internal database representation of DMC taskforces.	9-7
Table 9-2:	*DCS* internal database representation of M-P architectures.	9-7
Table 9-3:	*DCS* run-time using AST–FWE to schedule bus-type taskforces.	9-14
Table 10-1:	Assessments for the schedule of Figure 10-2.	10-3
Table 10-2:	Selected CPM schedules on an 8-processor architecture.	10-3
Table A-1:	Summary of important DSP devices from VLSI manufacturers.	A-1
Table B-1:	M-P topology node expansion metrics.	B-1
Table B-2:	M-P topology link expansion metrics.	B-2
Table C-1:	Worksheet functions supported by *TPG*.	C-1
Table D-1:	Options supported by the *DCS* scheduling engine.	D-1

Chapter 1

Introduction

Trends in audio engineering and VLSI suggest the feasibility of an integrated all-digital music production chain. Clearly, a digital mixing console (DMC) is required to complete this chain.

1.1 Background

There has been a requirement to mix musical sound sources from the very early days of recording. Even in the acoustic era vocalists were provided with their own recording 'horn'.

1.1.1 Historical Overview

The advent of multi-track tape machines and audio processing 'effects' in the early 1960s marked a significant turning point in recording and production techniques.[1] Recording to 'multi-track' meant that there was no longer a need to take irrevocable creative decisions until mix-down. Today, 24-track analogue and 32-track digital machines are commonplace with machine transports and MIDI sequencers synchronised via SMPTE timecode.

1.1.2 The Recording Studio

Close-miking and multi-track recording techniques are now an integral part of the creative process. To this end, the professional recording studio is divided into two distinct two areas.

1.1.2.1 The Sound Room

The sound room is the actual artist performance area and is normally constructed to provide both acoustically 'live' and 'dead' ends: the distinction being the amount of sound absorption in each. Generally, the only technical equipment found in the sound room, apart from electronic instruments and amplification, is monitoring headphones and microphones.

1. A *track* is a path on a magnetic tape which stores a single channel of audio information.

1.1.2.2 The Control Room

The control room is the nerve centre of the recording studio, where all recording information is routed to be processed and enhanced towards a marketable end product. The control room also houses most of the technical equipment available in the recording studio including the mixing console, multi-track tape machine, outboard effects and monitoring loudspeakers.

1.1.3 Production Techniques

With the consumer base increasingly aware of the subtleties of sound balance, studio production techniques are becoming more sophisticated as technical requirements become ever-more stringent. Computer-assisted mixing can be used to manipulate controls set up on previous passes and simultaneously record new control movements. Engineers need only concentrate on a few controls at a time in order to optimise the mix-down process.

1.2 Digital Mixing Consoles

Console design is limited both by available technology and conceptual limitations. Not withstanding, DMC design must at least maintain the standards set by analogue counterparts.

1.2.1 Design Goals

Goals which the industry has set for console manufacturers include perfect audio response, increased numbers of inputs and outputs, extended creative capabilities, and all at an optimised cost/performance ratio [White, 1988]. It is relatively easy to implement advances in any one, or perhaps a few, of these areas but only at the expense of other performance aspects.

Since standard analogue consoles require mechanically linked control sets, the requirements of more I/O channels together with more creative signal processing capability translate directly into larger desks. Beyond a certain point operational problems, particularly audio quality, parallax and channel strip length cannot be solved by traditional means.

1.2.2 Audio Performance

The acuity of the human ear is astonishing, detecting tiny amounts of distortion and yet accepting a dynamic range in excess of 110 dB. While most listeners cannot distinguish an

upper frequency limit, a response down to 20 Hz improves reality and ambience [Muraoka, 1978]. Digital ASP provides the dynamic range and frequency response required.

DSP has replaced and enhanced many ASP functions that were previously accomplished by analogue circuitry: However, the professional audio industry has not succumbed to the apparent digital panacea so readily. Concerns about cost, reliability and audio performance have, to date, limited the acceptance of digital console technology.

1.2.3 Control Surface Ergonomics

In analogue consoles, controls have to be positioned close to the actual processing circuitry for performance reasons. One control knob is needed for every variable with the result that the control panel is physically large. Remote control is difficult with such construction techniques and the order of processing stages is determined at the time of design.

Programmability eliminates the need for mechanical links between controls and ASP hardware. The programmability inherent in digital ASP promises an unprecedented degree of freedom for the console designer. Control and display types, and layouts can be optimised strictly for ease of operation, without concern for behind-the-panel physical restriction.

1.3 Parallel Architectures

Interest in this area can be traced to the construction of the *Cosmic Cube* at CalTech. Originally intended for work on planetary dynamics, this M-P found much wider application.

1.3.1 Multi-Processors

There are three demarcation points in the spectrum of architectures for parallel computation. At one extreme massively-parallel architectures, such as Thinking Machines *Connection Machine*, use thousands of simple bit-serial elements. At the opposite extreme, machines such as the Cray *X-MP* and Fujitsu *VP-200* rely on several customised vector processing units.

M-P architectures lie in between, offering potentially greater performance in applications with a high degree of intrinsic parallelism. Applications include DSP,

computational fluid dynamics and, more recently, financial modelling. Commercial M-Ps increasingly use 'off-the-shelf' work-station RISC processors, such as DEC's α chip.

1.3.2 Processing Technology

Two fundamental decisions are, firstly, how powerful each DSP should be, and, secondly, how many processors should be supported. For a system of required performance P, and an ideal architecture of n processors, each with an individual processing capability of p, the hyperbolic relationship between n and p defines a span of possible architectures satisfying P.

System architects may choose a small number of powerful, expensive processors; or many relatively slow, cheap parts. This latter option is made possible by the emergence of programmable DSP devices suitable for professional ASP. The advantages of using low-cost technology is balanced by the degradation in efficiency that inevitably occurs as n increases.

1.3.3 IPC Topologies

Many issues arise from the need for parallel architectures: firstly, some form of inter-processor communications (IPC) mechanism is necessary; secondly, access to shared data must be coordinated. Repeated studies have shown that optimising M-P performance requires a judicious combination of computation and communication performance.

A M-P consists of tens of nodes connected in some fixed *topology*. Ideally, each DSP would be directly connected to all others. Unfortunately, packaging constraints and hardware costs limit node degree. While many interconnection schemes are feasible, topology metrics enable console designers to quantitatively compare alternative architectures.

1.3.4 M-Ps and DMCs

A number of M-P architectures have been built in response to an ever-growing need for supporting computationally-intensive applications. Dedicated architectures aim at maximising performance for a particular taskforce. Within the realm of digital mixing, this approach simply corresponds to direct digital implementation of an analogue console design.

On the other hand, general-purpose architectures are designed to provide good performance for a range of processing configurations. While non-real-time editing is

commonplace in post-production situations, flexible real-time audio mixing is essential for the recording studio environment. It is within this context that M-P topologies are analysed.

1.4 Task Scheduling Strategies

Regardless of processing technology and IPC topology, task scheduling strategies are necessary to allocate hardware resources to digital console audio processing algorithms.

1.4.1 Introduction

With the availability of fast DSPs with sufficient precision to accommodate the word-growth produced by 16-bit audio [Eastty, 1986], the all-digital successor to today's analogue consoles is now realisable. Several issues pertinent to all large parallel computer architectures are relevant here including code partitioning and task scheduling strategies.

Many of the DMC scheduling techniques discussed here are developed from problems in management science and operations research originally concerned with the utilisation of people, equipment and raw materials. If raw materials are equated with ASP tasks, and people and equipment with processors, then the rationale behind these techniques becomes clear.

1.4.2 Partitioning

Partitioning of a console functional description occurs naturally at a level of granularity, appropriate to the large grain dataflow (LGDF) paradigm [Gaudiot, 1988]. To efficiently utilise a DSP computation engine bench-marked at several hundred MIPS and maximise throughput, the DMC functionality must be distributed across available system resources.

Analogue implementations are free from this overhead as the logical structure of the console control surface is fixed by the analogue ASP hardware 'beneath'. In order to shed some light on the complexity of this concept, a systematic study has been undertaken of techniques relevant to the mapping of digital ASP tasks onto multiple DSP configurations.

1.4.3 Complexity

Since its introduction in the early 1970's, a wide variety of problems from computer science and operations research are now known to be *NP*-complete. Such problems are now so

pervasive that it is important for anyone concerned with the computational aspects of ASP task scheduling algorithms to be familiar with the meaning and implications of this concept.

Because of the intractability of the task assignment problem, research effort has focused on finding heuristic algorithms that produce sub-optimal assignments and efficient optimal algorithms for restricted cases. Here, these approaches are extended within the context of scheduling mixing console taskforces on large-scale multi-DSP architectures.

1.4.4 Opportunities

Such console architectures are inherently software-defined, designed from a pre-defined library of audio DSP kernels. The studio engineer will have the freedom to rebuild the console functionality and re-allocate processing power as appropriate for each session. It is the availability rather than type of processing power which becomes the limiting factor.

Since the audio processing in such a DMC is accomplished through software, the configuration of the desk can be changed at will by running the processing routines for the various functions in a different order. The engineer can configure the DMC to his own requirements by, say, entering symbols on a block diagram displayed on a video screen.

1.5 Thesis Overview

Following the discussion of the previous section, this thesis is effectively divided into 3 parts: dealing with digital mixing consoles, parallel architectures and task scheduling strategies.

1.5.1 Digital Mixing Consoles

As an introduction to the field of audio engineering, Chapter 2 considers the various forms of ASP algorithm employed in the professional recording studio. An example of 'pop' recording practice and console operation is given together with a review of console terminology and the facilities now considered 'standard' on *professional* mixing consoles.

This treatment is extended in Chapter 3 to discuss the implications of implementing the mixing function entirely in the digital domain. Automation features are described with the

foundations of control layout and ergonomics developed from the studio engineer's perspective. An indication is given of the form of possible next-generation control surfaces.

1.5.2 Parallel Architectures

Chapter 4 discusses the architectural features of programmable DSP devices, serial-bus, ASIC and vector processors with respect to the requirements of digital audio products. Twelve topology metrics are defined in Chapter 5, and an extensive analysis of alternative M-P architectures is undertaken using the *Topology Analysis Tool* (*TPG*) developed by the author.

Novel schemes supporting message-passing connectivity for programmable DSPs are compared in Chapter 6. Design decisions taken by the author, leading to the development of a hybrid M-P (*HYMIPS*) employing these techniques are documented. *HSR* and *HST* digital audio peripherals, including schematic diagrams, are also reported in this chapter.

1.5.3 Taskforce Scheduling Strategies

A model of DMC scheduling is developed in Chapter 7 and used to explain a variety of speed-up characteristics. LPP architectures and IPC mechanisms are then investigated within the same paradigm. A brief survey of deterministic scheduling theory in Chapter 8 leads to the derivation of five application-specific heuristics and four static-labelling schemes.

In Chapter 9, the *DCS* research framework and predicate hierarchy are described in detail. AST heuristics and CPM labelling schemes are compared using FWE. Dynamic-labelling strategies and mix-bus granularity issues are studied in Chapter 10. Results of this research and further work proposed in the light of these findings are summarised in Chapter 11.

1.5.4 Contribution & Significance

HYMIPS, a hybrid audio M-P targeted at real-time multi-channel ASP, has been developed in accordance with the author's detailed design notes, complete with *HSR* and *HST* SP-DIF interfaces. *TPG* enables direct comparison of 12 M-P topologies within a framework of 13 topology metrics defined by the author in the course of this research. In *DCS* the author has developed three algorithms tailored to scheduling DMC taskforces on M-P architectures.

Several IPC mechanisms originally investigated from a theoretical perspective are supported including FWE, NSL and NML. CPM, DYN and AST control parameters select from 5 static-labelling schemes, 4 dynamic strategies and 5 admissible heuristics. Together, *TPG*, *HYMIPS* and *DCS* contribute directly to the problem domain by advancing fundamental analytical and architectural techniques vital for commercial DMC implementation.

1.6 Conclusions

Recent commercial developments have lead to considerable interest in the M-P paradigm for large-scale DMCs. This thesis focuses on salient issues in processing hardware, system architecture and software allocation for such products. The goal is to provide insight into the major design decisions which must be addressed during the development of such a console.

The development of M-P topologies clearly depends on the recent appearance of powerful single-chip DSPs and inexpensive memory. Although M-Ps offer potentially enormous computational power at modest cost, many important questions must be resolved before DMCs can exceed the cost/performance ratio provided by today's analogue consoles.

As yet, there is no expedient solution. Although DSP processors are available with the required level of processing power, audio processing and operating system software, and the IPC to couple it must all be developed by the console manufacturer. This is a massive undertaking, but there is no other way to fully realise the future of audio console design.

1.7 References

[Eastty, 1986] Eastty P: "Digital Audio Processing on a Grand Scale", *81st Audio Engineering Society Convention*, Los Angeles, California, USA, September 1986.

[Gaudiot, 1988] Gaudiot J L, Pi J I and Campbell M L: "Program Graph Allocation in Distributed Multicomputers", *Parallel Computing*, Vol. 7, No. 3, pp. 227-247, May 1988.

[Muraoka, 1978] Muraoka T (et al.): "Sampling Frequency Considerations in Digital Audio", *Journal of the Audio Engineering Society*, Vol. 26, No. 3, pp. 252-256, March 1978.

[White, 1988] White P: "Console Ideology", *Sound Engineer and Producer*, pp. 29-35, August 1988.

Chapter 2

Analogue Mixing Consoles

The mixing console — also called a 'desk' or 'board' — is the heart of the recording studio. Once an audio signal enters the console, everything that happens to it before reaching the final medium, including level control, can be described as audio signal processing (ASP).

2.1 Audio Signal Processing

The primary ASP techniques are those which modify the frequency spectrum, dynamics, time domain, or spatial location. Figure 2-1 shows how they may be classified.

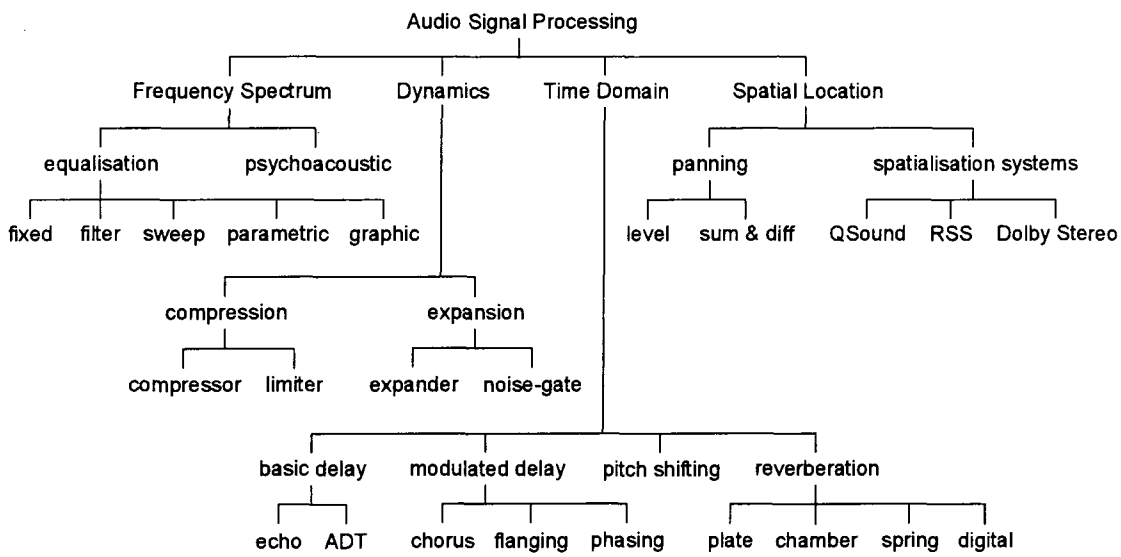

Figure 2-1: Classification of audio signal processing techniques.

2.1.1 Frequency Spectrum

In the infancy of sound recording the goal was to achieve an essentially 'flat' frequency characteristic. Equalisation (EQ) arose as a means of correcting deficiencies in the response.

2.1.1.1 Filters

Sharply attenuating frequencies above and below the spectral range of a musical instrument reduces 'leakage'.[1] For example, a 10 kHz low-pass filter attenuates hiss-type circuit noise without affecting tone quality as much as gradual treble roll-off.

2.1.1.2 Parametric Equalisation

The most typical EQ employed in mixing consoles is the parametric equaliser. Parametrics allow continuous adjustment of centre frequency, gain, and tightness (or *Q*) of the response curve. As illustrated in Figure 2-2, *Q* can be adjusted to act on one narrow frequency band or to have a very broad effect. Overlapping bands make this type of equaliser very flexible.

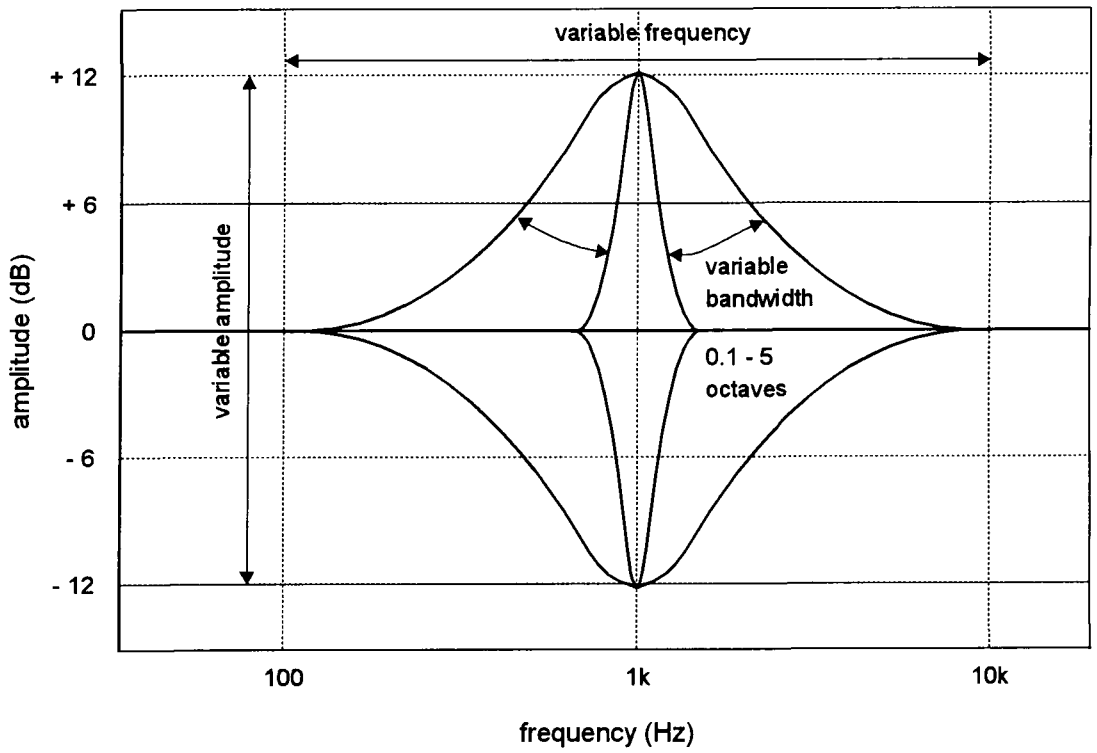

Figure 2-2: Range of control provided by one section of parametric equalisation.

2.1.1.3 Graphic Equalisation

A row of slide potentiometers divides the audible spectrum into bands centred at frequencies determined by ISO recommendations — a third-octave unit has 31 separate bands. A large

1. In Audio Engineering, 'leakage' refers to microphones pick-up from sources other than that being recorded. Conversely, in signal processing the term refers to the spectral smearing introduced by windowing functions.

number of bands can result in considerable phase response degradation. Rarely found within professional consoles, graphic EQ is often used as outboard processing.

2.1.1.4 Psychoacoustic Processing

Psychoacoustic processors increase presence without the side-effects, such as increased noise, inherent in conventional EQ. Units such as the Aphex *Aural Exciter* add a low-level signal of harmonics not present in the original, generated via distortion with high-pass filtering.

2.1.1.5 Studio Applications

EQ is used in the studio for two distinct but overlapping purposes. In the first, EQ is used to compensate for frequency response aberrations, particularly those which result from cardioid microphone placement. On an orchestral session, this kind of manipulation may be quite sufficient. Conversely, on a pop session, the engineer will make an unconscious transition from using EQ in a 'traditional' sense to using it for purely creative purposes.

2.1.2 Dynamic Range

Dynamic range control is a vital part of the recording process. Whilst riding a fader is still one of the most effective methods, engineers have come to rely on automated ASP techniques.

2.1.2.1 Compression and Limiting

Compressors reduce the gain applied to an input when the signal level exceeds a preset 'threshold'. Dynamic range is reduced while transient peaks are passed unaltered. Musically useful ratio settings range from 1.5:1 to 4:1. In comparison, limiters apply extreme compression to transient peaks to retain 'musical' events without altering dynamic range.[1]

2.1.2.2 Expansion and Noise-Gating

In ASP terms, expanders and noise-gates are simply the inverse of compressors and limiters, A gentle characteristic will gradually increase the attenuation as the levels falls; a harsh ratio, close to ∞, as in a noise-gate, will sharply reduce the apparent noise level. Noise gates usually allow access to the side chain to enable the gate to be controlled by a separate 'key' signal.

1. It should be noted that sufficiently slow attack times may lead to harmonic distortion in low-frequency tones.

2.1.2.3 Studio Applications

Again, applications can be divided into two main areas: control of dynamics to enable 'difficult' material to be recorded, and dynamics control for creative effect. For example, a slow compressor attack time can be used to 'tighten-up' a bass guitar as the deliberate overshoot on initial transients adds punch. During mixdown, a gate is sometimes used on each output of an analogue equipment to reduce tape hiss without removing program material.

2.1.3 Time-Domain Processing

Modern music production requires numerous time-domain techniques. Central to these effects is the ability to delay audio signals, and then recombine original and delayed waveforms.

2.1.3.1 Basic Delay Effects

If the original signal is delayed by 50 ms to 1 s two distinct sounds are heard. A short delay of around 200 ms results in 'slap echo', as originally produced by tape-loops in the 1950s for rock 'n' roll records. If the delay time is reduced to around 15 to 35 ms, delays can be used to 'double' a sound, creating an effect known as automatic double tracking (ADT).[1]

2.1.3.2 Modulated Delay Effects

If the delay time used in ADT is modulated by a low-frequency oscillator (LFO), slight pitch detunings are produced like those which occur in a chorus of unison voices. At delay times less than 20 ms, the ear is unable to resolve direct and delayed signals. Phase cancellations result in a series of dips in the net frequency response known as flanging (see Figure 2-3).

2.1.3.3 Pitch-Shifting

A pitch-shifter is a delay device which can change the pitch of a signal in real-time without changing its duration. In professional units with well-designed algorithms, such as the *Eventide Harmonizer*, shifts of up to 2 octaves are possible [Bogdanowicz, 1989]. Pitch-shifting allows the creation of harmony parts or the correction of out-of-tune notes.

1. Due to arrival time localisation, known as the *Hass Effect*, 'spatial' and level balance can differ significantly.

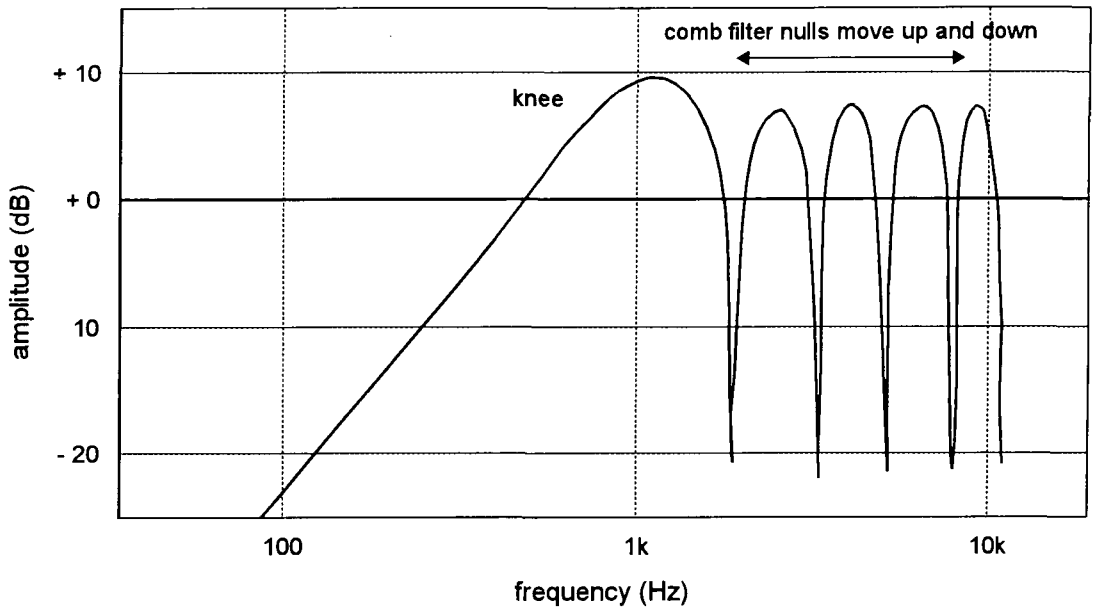

Figure 2-3: Example of effect of flanging on flat frequency spectrum.

2.1.3.4 Digital Reverberation

To simulate the highly diffuse reverberant tail of natural reverberation (see Figure 2-4), a comb and all-pass filters can be used together with recirculating delays.[1] Another method is to trace the paths of individual rays emanating from an isotropic source. Unnatural effects, such as non-linear and reverse reverb, are now fully integrated into the studio engineers' repertoire.

2.1.3.5 Studio Applications

A wide variety of musical effects can be created by simply delaying an audio signal. For example, several repetitions can be evenly spaced creating an echo rhythm in sync with the original track. ADT imparts a sense of ambience to close-miked instruments not unlike early reflections. Chorusing can be used to split a mono sound into pseudo-stereo. Digital reverberation is employed to give an authentic sense of space to acoustically dry recordings.

2.1.4 Spatial Localisation

Classical psychoacoustics predicts that interaural intensity difference (IID) and interaural time delay (ITD) are the primary cues used by the human auditory system [Rossing, 1990].

1. Advances in DSP have made other synthetic reverberation techniques obsolete.

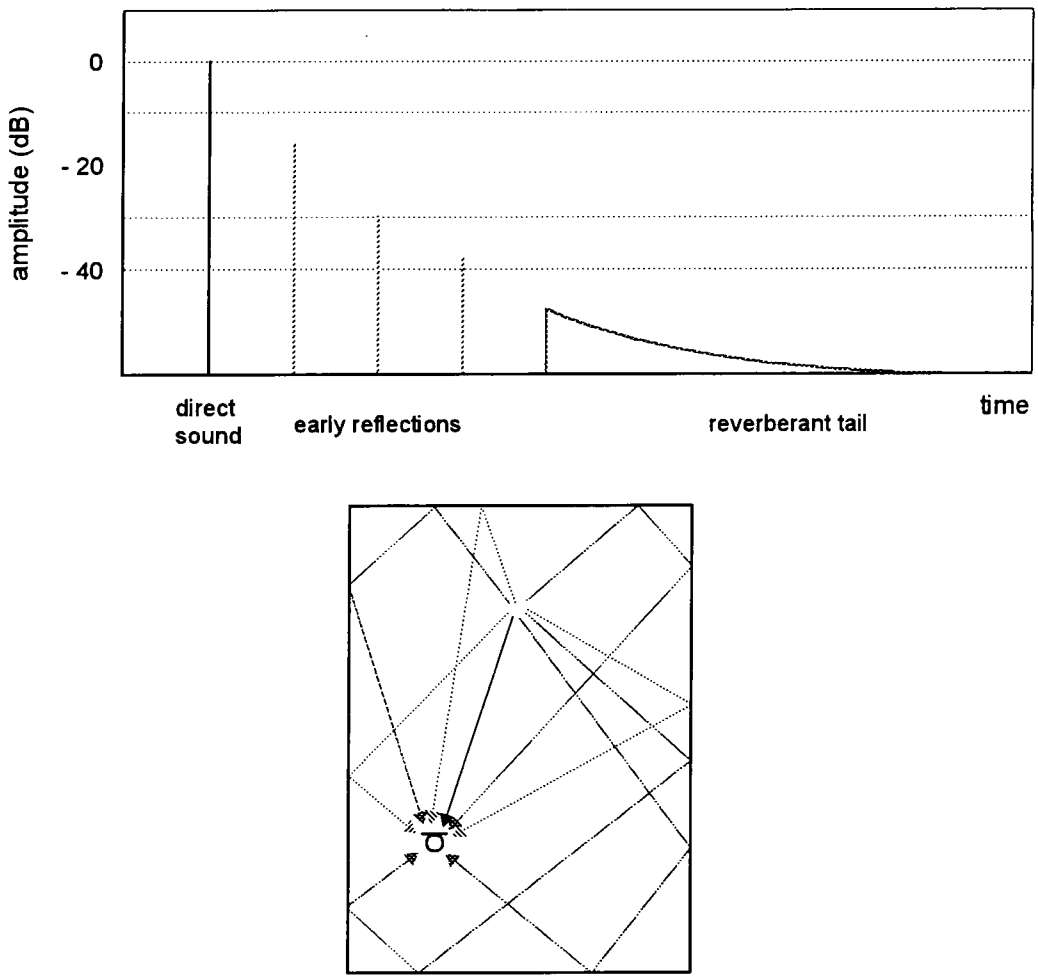

Figure 2-4: Effect of artificial reverberation unit on sound impulse.

2.1.4.1 Localisation by Level

The panoramic potentiometer or *pan-pot* gives constant loudness for a single source between stereo channels, as Figure 2-5. This type of spatial manipulation is more accurately referred to as panned-mono, as sound localisation is determined solely by IID. Pan-pot design is a surprisingly complex area, primarily, because it is necessary to retain mono-compatibility.

2.1.4.2 Sum-and-Difference Panning

An extension of level-based panning, this method relies on a sum-and-difference matrix encoder and decoder, as shown in Figure 2-6. This effect is often found in consumer portable stereo systems to generate a 'wide stereo' effect. Although capable of placing a sound beyond the width of the loudspeakers, compatibility with mono is severely limited.

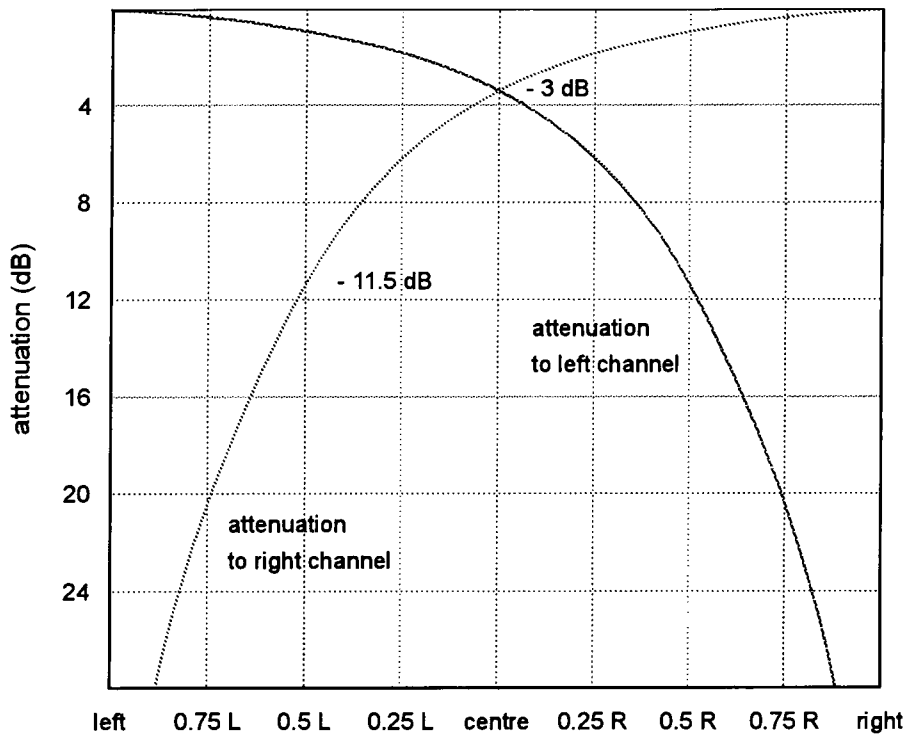

Figure 2-5: Example of typical console pan-pot characteristic.

2.1.4.3 Commercial Spatialisation Systems

Spatialisation techniques emulate the directional cues used by the human auditory system, combining level and phase cues together with head-related transfer-functions (HRTF).[1]

2.1.4.3.1 QSound

This system combines delay with selective manipulation of the signal in the frequency domain — 6-8 dB of boost/cut in narrow 11 Hz bands. A total of 8 inputs leads to either 4 placement positions on each side or continuous panning across the soundfield. Sound sources can be placed with high resolution in a 180° arc in front of the listener [White, 1992].

2.1.4.3.2 Roland Sound Space — *RSS*

RSS provides optimum loudspeaker reproduction via transaural processing. Crosstalk is eliminated by subtracting left-right and right-left crosstalk components from the originals. Left and right signals are then processed using a set of FIR filters and weighting coefficients. In practice RSS works best between 500 Hz and 2 kHz as it can occasionally disturb timbre.

1. Coincident microphone systems, such as the Calrec *SoundField*, are not strictly sound processing techniques.

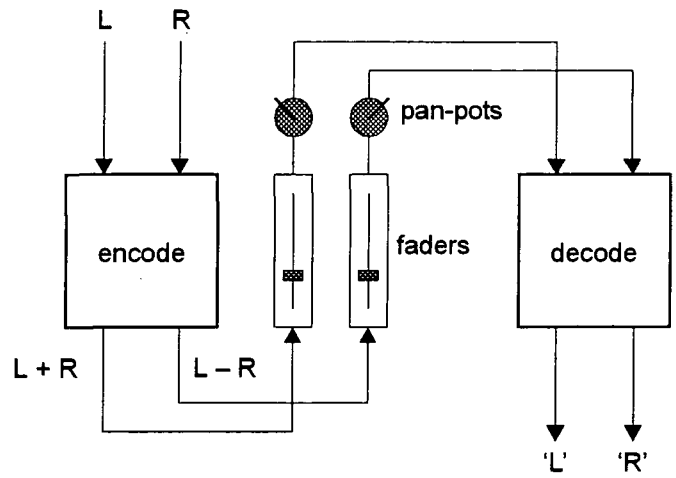

Figure 2-6: Outline arrangement for sum-and-difference panning.

2.1.4.3.3 Dolby Stereo

Both *Dolby Stereo*, and the *Pro-Logic* surround system split the signal into 4 channels: LEFT, RIGHT, CENTRE and SURROUND. The CENTRE channel consists of dialogue only, whereas ambience and special effect information are assigned to the SURROUND channel. It should be noted that SURROUND channel is limited to a bandwidth of 100 Hz to 7 kHz.

2.2 Console Terminology

This section describes the elements found in a typical recording console. While every manufacturer's terminology varies to some degree, most include the features described here.

2.2.1 Multitrack Recording Techniques

Over the years fundamental differences have developed between multitrack recording techniques used for 'classical' and 'popular' repertoire. As popular music is unquestionably the most demanding of console facilities, a method of 'pop' recording is described below.

2.2.1.1 Track-Laying

For the purposes of tracking, instruments are not all normally recorded simultaneously. To allow maximum flexibility in the later stages, the multitrack master is built up a few tracks at a time.[1] On the first 'take' the basic rhythm tracks including drum kit, bass guitar, rhythm

guitar and guide vocal will be laid down using a minimum of processing. Figure 2-7 shows the complexity of microphone arrangement typical for a studio drum kit.

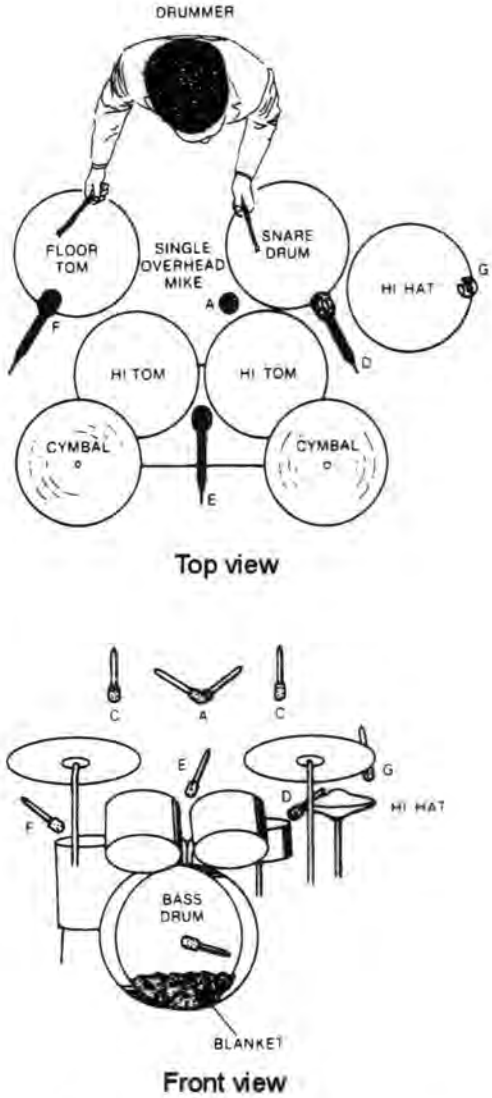

Figure 2-7: Typical arrangement of microphones around drum-kit (courtesy [Bartlett, 1988]).

2.2.1.2 Overdubbing

Successive takes are used to 'overdub' further instrumental parts in sync with those already on tape until the lead vocal is re-recorded. The signal path is the same as that used in tracking, except that microphone signals are mixed with tracks already on tape. In order to avoid microphone 'spillage', it is usual to acoustically isolate performers from one another. Studio 'cue' headphones provide a balanced mix of live and recorded tracks.

1. To enable synchronisation to MIDI equipment, the tape is first *striped* with SMPTE timecode along its length.

2.2.1.3 Mixdown

At some later date the multitrack master is mixed down to stereo. The engineer will add various effects processing and change, often dynamically, the relative balance and position of instruments. As a result, the final sound is often very different to that of the monitor mix heard during previous stages. Input channels are often used as channel processing facilities are usually more elaborate than those provided in the track monitor paths [Bartlett, 1988].

2.2.2 Two-Channel Console

Figure 2-8 shows the signal paths in a basic two-channel console.[1] After the microphone signal is properly pre-amplified it passes through EQ and, perhaps, other dedicated processing sections. A 'fader' — usually a linear[2] potentiometer — provides level adjustment, while a pan-pot simulates IID in the panned-mono sound-stage of multitrack recording.

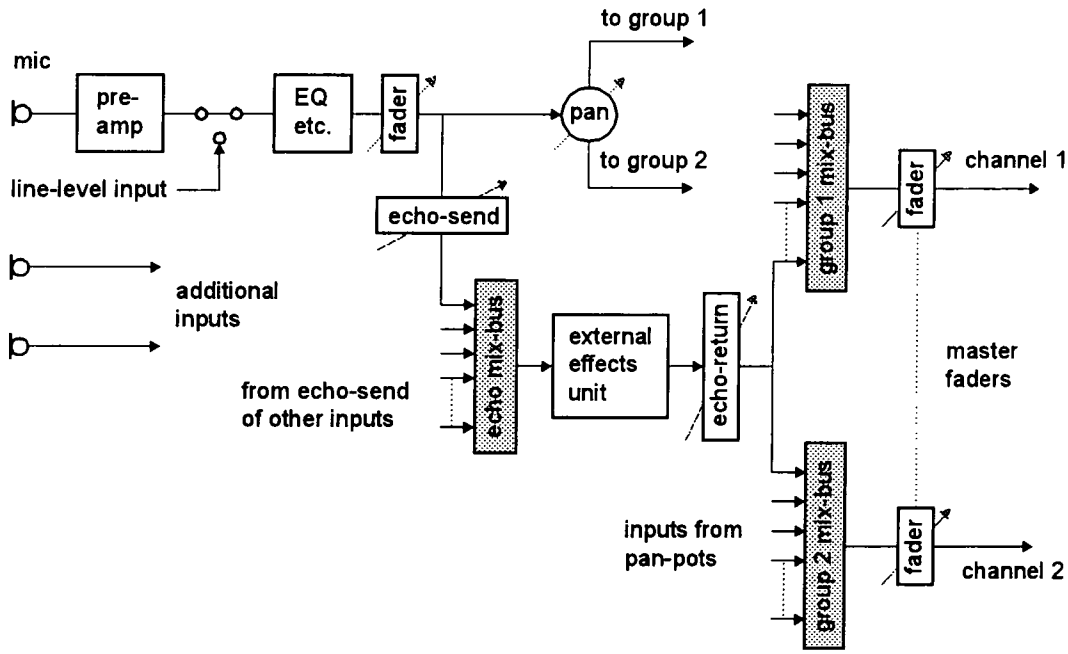

Figure 2-8: Signal flow diagram of a typical 2-channel mixing console.

Signal may also be sent to external processing, such as a reverberation unit, by using the 'echo-send' or 'aux-send' control. The 'echo-return' adjusts the overall level of

1. For clarity, only one microphone input channel is shown.
2. Of course, any linear audio potentiometer is still *logarithmic* in scale.

reverberation added to the mix as a whole. Note that the signal path illustrated in Figure 2-8, with inputs from tape tracks, is essentially that of a multitrack console during mixdown.

2.2.3 Simple Multi-Channel Console

Figure 2-9 shows the signal paths in a typical multi-channel console. The design is similar to the 2-channel console described above, but with the addition of channel assignment routing.

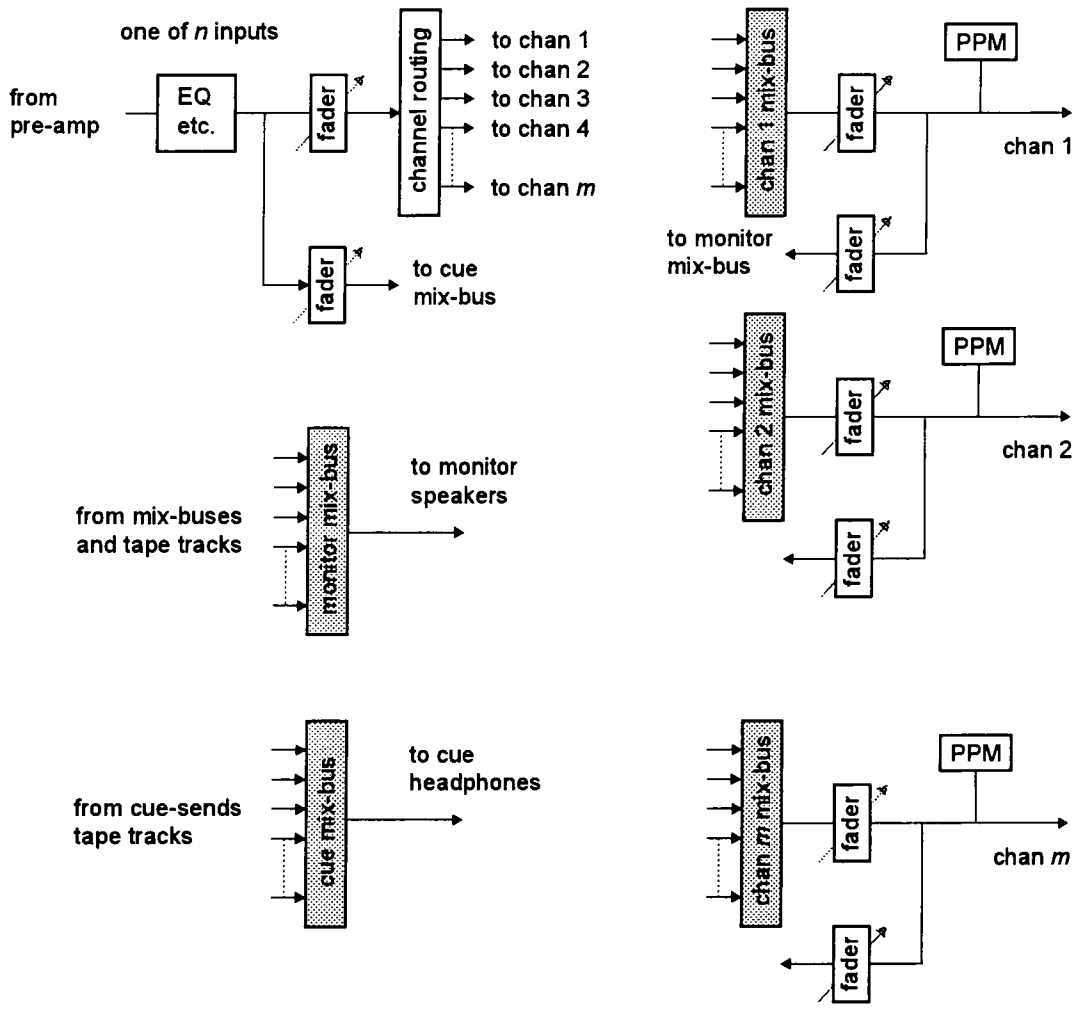

Figure 2-9: Signal paths in a typical *m*-channel console (during recording).[a]

a. For clarity, all level controls are shown as linear faders: however, many will be rotary. PPM represents peak programme meter, one type of professional console metering ballistic (see Section 2.2.4 below).

2.2.3.1 Signal Routing

Controls that affect a single input are termed a 'module' or 'channel strip'. In conventional designs this is generally a vertical panel in the control surface. A typical desk consists of *n*

such modules, output groups and some form of metering.[1] Several microphones (say, for a brass section) can be grouped and panned across a number of outputs, to create a 'sub-mix'.

2.2.3.2 Solo and Cut

The module 'solo' button allows the engineer to monitor this input only, without affecting other console functions. As the main monitor signal is momentarily muted, soloed inputs are still heard at their proper levels and stereo positions. The 'cut' switch mutes the input by disconnecting the module output from the channel fader, routing and post-fader aux sends.

2.2.3.3 Mix Buses

As shown in Figure 2-9, there are also two subsidiary mixing circuits in any console. During recording, the 'monitor' mixer allows the engineer to preview the sound of the final master without affecting signals going onto tape. The 'cue' mixer combines pre-recorded tape tracks and live microphone signals onto the cue mix-bus sent to the musicians' headphones.

2.2.4 Metering

Console level meters are of two types: volume unit (VU) and peak programme meters (PPMs). PPMs are almost linear in dB and able to detect momentary transients, whereas VUs give low readings on music programme. Mechanical meter movements have been superseded by neon plasma or LCD columns, with electrically simulated ballistics. Plasma meter systems often display stereo spectrum analysis or channel VCA display during automated mix-down.

2.2.5 Free-Grouping

With free-grouping, any channel may be designated as a sub-group. Several channels, 1 & 2 in Figure 2-10, are directed to mixbus 1 which feeds the 'group' channel 3. This output, and other inputs such as channel 4, are combined on mixbus 2 with the last channel controlling the group output. This concept is common in 'in-line' multitrack consoles to permit varying programme use. Switching is arranged so that a channel cannot pick up its own group.

1. In most consoles, *n* is a multiple of 4.

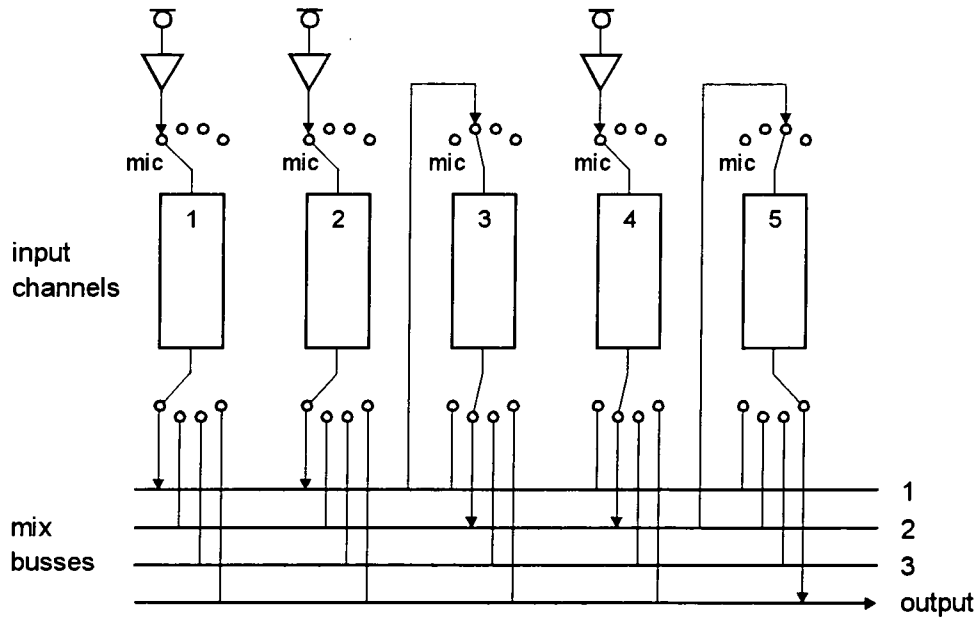

Figure 2-10: Illustration of the free-grouping principle.

2.3 Professional Console Features

Most professional consoles use an I/O type of construction, also termed 'in-line'. The following sections discuss those features found on such professional recording consoles.

2.3.1 Input/Output Module

For example, the SSL *G Series* SL 611G I/O module has two completely independent audio signal paths — CHANNEL and MONITOR. ASP sections may be switched into either path.

2.3.1.1 Filters and Equalisation

This section combines high-pass (H-P) and L-P filters, Figure 2-11, with four-band EQ, Figure 2-12. High-frequency (H-F) and L-F sections comprise 12 dB/octave shelves with variable cut-off frequency and boost/cut. Continuously variable Q (0.5 to 3.0), gain (±15 dB) and centre frequency (12:1) are provided in the high-mid-frequency (HMF) & LMF sections.

×3 and ÷3 buttons enable the HMF and LMF sections to work with or against the H-F and L-F sections. The EQ and Filters can be routed separately to the signal paths within the module using SPLIT. This approach allows the Filters to be used in the CHANNEL, which feeds the multitrack while in RECORD, with the Equaliser used on the MONITOR path only.

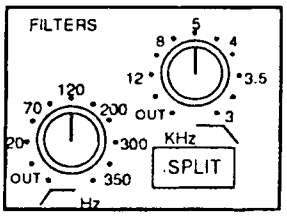

Figure 2-11: SSL G Series I/O module Filters section (courtesy SSL Ltd.).

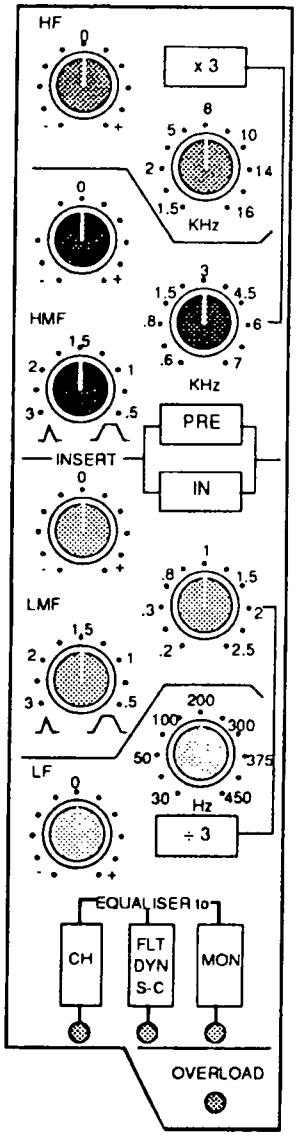

Figure 2-12: SSL G Series I/O module EQ section (courtesy SSL Ltd.).

2.3.1.2 Dynamics

The dynamics section of an SSL *G Series*, shown in Figure 2-13, comprises a compressor/ limiter and an expander/gate using the same VCA. Both sections work independently but can

be operational at the same time to provide dynamics control of either signal path. Switching links the side-chain signal to that of the next Dynamics section for stereo operation.

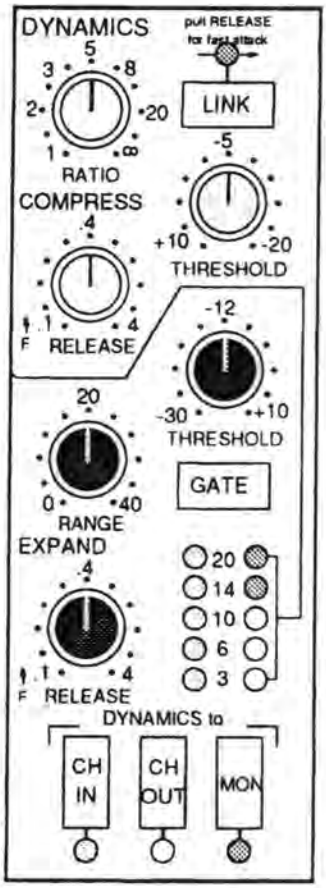

Figure 2-13: SSL G Series I/O module Dynamics section (courtesy SSL Ltd.).

2.3.1.3 Cue and Aux Sends

As shown in Figure 2-14, the SSL Cue and Aux send system is based around one stereo and four mono sends per module. Each send can be derived pre- or post-fader (PRE) and from the MONITOR signal path, instead of the CHANNEL signal path, with the SMALL FADER button. The stereo send, normally used for musicians' cue headphones, also has a pan control.

2.3.2 Centre Section

Typically, the centre section is designated for machine control, automation management and log-keeping. The centre section of a *G Series* is fitted with a full-sized QWERTY keyboard, as Figure 2-15, including an in-built TV monitor to display automation and other housekeeping

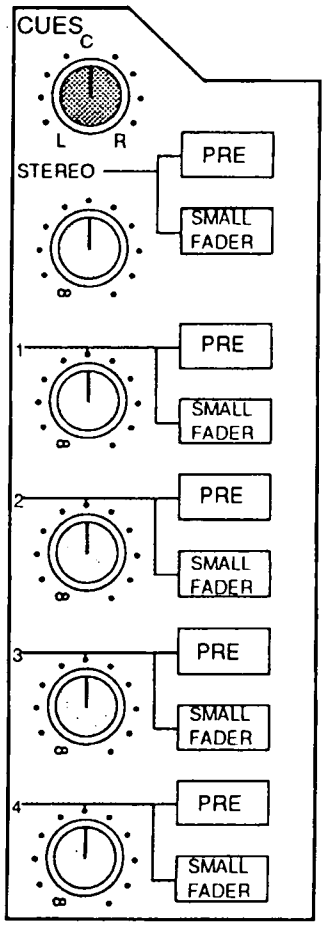

Figure 2-14: SSL G Series I/O module Cue and Aux Send section (courtesy SSL Ltd.).

information. The SL 651G Master Facilities Module houses master logic, monitoring and metering control; aux send and return masters; and talkback and listen microphone systems.

2.3.3 Master Routing and Signal Flow

Six key points in each I/O module define two separate audio paths: CHANNEL and MONITOR input sections, SMALL and LARGE faders, and two outputs. One path, the CHANNEL, is normally controlled by the LARGE (VCA) fader. Global status buttons determine the default signal paths of all I/O modules, supporting configurations from track laying to final mixdown.

There are four basic desk statuses: RECORD, REPLAY[1], MIX and RECORD+MIX (overdub). In RECORD or REPLAY, all inputs switch to MIC, whereas in MIX all inputs switch to

1. REPLAY is the same as RECORD except that off-tape tracks come from the 'replay' head instead of the 'sync' head of an analogue multitrack machine, if connected.

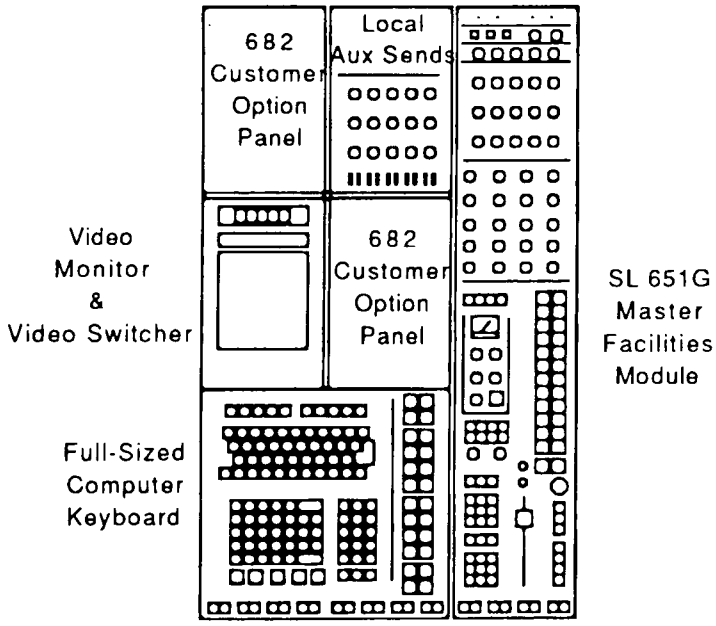

Figure 2-15: SSL G Series Master Facilities Module and Centre Section (courtesy SSL Ltd.).

LINE for final mixdown, allowing SMALL faders to be used for additional inputs. RECORD+MIX reverses the normal MIC/LINE selection for selected channels for overdubbing purposes.

2.3.4 VCA Grouping

The LARGE fader sends a control voltage, via the studio computer, to a VCA in the I/O module. As a result, the automation system can automate the level of the LARGE fader, Figure 2-16, usually fed from the CHANNEL signal path. Control of the LARGE fader is assigned to one of the eight VCA group faders using the channel VCA thumbwheel.

VCA grouping allows several channels to be grouped at the control, rather than audio, interface. For example, several independent tracks in a backing vocal mix can be grouped while the individual audio signals remain intact. These channels may be controlled by the same fader, so easing problems of physical reach and simultaneous operation.

2.4 Console Automation

According to Jones & Jubb, "The mix-down process is a search for perfection" [Jones, 1987]. Every recording engineer has encountered the frustration, usually experienced in the early hours of the morning, of searching for 'that mix' achieved 8 hours previously.

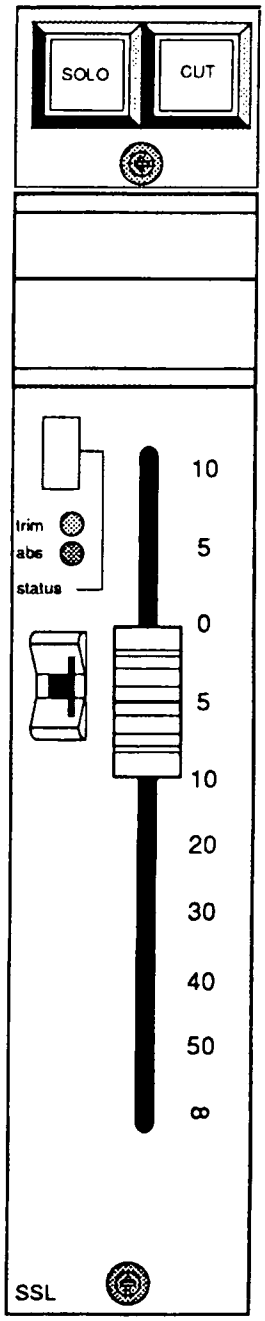

Figure 2-16: SSL G Series I/O module Large Fader section (courtesy SSL Ltd.).

2.4.1 Automated Mix-Down

Console automation reduces the effort required to perform routine studio operations. Virtually all high-end music production consoles have extensive computer-assisted mixing facilities.

2.4.1.1 Comparison

An inherent disadvantage of VCA automation is that the physical fader position bears no direct relation to absolute gain. Despite visual aids, the loss of this direct relationship presents

a serious perceptual difficulty. An alternative technique, employed in moving fader systems such as Neve's *Flying Faders* and SSL's *Ultimation*, is to drive the fader itself. This approach recognises that the fader knob is simultaneously a means of control and an indicator.[1]

2.4.1.2 Synchronisation

SMPTE time-code is recorded onto one tape track and used to relate mix data to tape position. Computer-assisted mixing manipulates the controls set up on previous passed and simultaneously records any new movements. Such mix automation systems, as exemplified by the Neve *VR* and SSL *G Series* desks, enable the engineer to recall previous combinations at will, and use the best of previous mixes together with just a touch of final adjustment.

2.4.2 Beyond Fader Automation

In order to achieve complete automation of the professional analogue console, programmable equivalents must be found for both switching and variable functions.

2.4.2.1 Switching Functions

FET switching is sufficiently reliable and cost-effective for professional audio use. Using non-locking buttons controlling FETs, it is possible to build analogue consoles in which all simple switch functions are automated. However, FET switches can inject 'charge transfer' clicks into the signal path, presenting considerable problems when full programmability is sought.

2.4.2.2 Variable Functions

In order to automatically reset potentiometers to within 2% error, it is necessary to find some means of either remotely controlling potentiometers or developing a programmable replacement. To a limited extent, both approaches have been successfully implemented in moving-fader or VCA schemes: both have gained widespread acceptance within the industry.

Many variable functions are achieved by rotating knobs and variable potentiometers. The costs involved in replacing all such standard control sets with VCA or servo-motor

1. *Ultimation* is something of a hybrid, allowing the motorised fader to be used purely as a level indicator, with the audio signal able to be switched silently and transparently between VCA and fader track as required.

technology are, however, staggering [Saunders, 1985]. Control range stability in VCAs and repeat-accuracy tolerances in servo-motors restrict the usefulness of either substitute.

2.4.3 *Total Recall*

Memory and automation can today extend further than the faders alone. The *Total Recall* system, pioneered by SSL, employs an additional track on each potentiometer and an extra pole on each switch, to provide positional information.[1] As the controls are manual, there is of course no automatic reset, but a colour VDU facilitates manual resetting by the operator.

To go beyond *Total Recall*, requires automatically resetting over 1,500 potentiometers and possibly double that number of push button switches. While digitally-controlled analogue designs — such as the Euphonix *CS* — do exist, the desire for greater automation capability poses a virtually insurmountable challenge to analogue mixing technology.

2.5 Conclusions

ASP has defined the characteristic recorded sound of each musical era. The 1950s had valve-based compressors and slap echo; the 1960s used 'fuzz' distortion, wah-wah, and flanging; the 1970s popularised delay and psychoacoustic processors; in the 1980s recording techniques were revolutionised by digital reverberation. Today, spatialisation systems are stressed.

Professional mixing consoles contain considerable processing facilities, notably EQ. Several high-end products also offer control of dynamics and some have offered a degree of time-domain control. Basic control of spatial parameters is provided by the humble pan-pot. At the present time more comprehensive processing requires outboard equipment.

DSP has added to the repertoire of sounds available in the recording studio, while making effects that were originally difficult to obtain now relatively easy. Digital ASP has taken full advantage of decades of evolution in the computer industry and is fortunate in benefiting from contributions from such otherwise unrelated fields as seismology and radar.

Console operation would be simple — and totally inflexible — if the signal paths were fixed. As it is, there are many permutations which give professional consoles the flexibility

1. Based on enhanced Z80 processor technology.

accepted as an essential part of audio mixing. These permutations allow the engineer to configure the desk to conform to any task presented during a recording session.

The industry demands a greater numbers of inputs and outputs with more creative signal processing. Size, number of controls, and component density are constant design problems as analogue consoles have kept pace with operational requirements. It is unlikely that these problems can be resolved while maintaining standard analogue performance levels.

Rather than developments in ASP as previous decades, the author predicts that advances in professional console technology will define recorded sound in the coming epoch. While some digital consoles exist in the current marketplace, none provide the degree of functionality beyond programmable analogue originally anticipated by the industry.

This issue depends very much on the design of the control surface itself, and the power and reliability of control and automation software. These factors are especially critical in the evaluation of any digital console. Consequently, it is appropriate to investigate the issues of digital implementation as they affect the console designer and, ultimately, the DMC operator.

2.6 References

[Bartlett, 1987] Bartlett B: *Introduction to Professional Recording Techniques*, Howard W. Sams & Co., 1987, ISBN 0-672-22574-3.

[Bogdanowicz, 1989] Bogdanowicz K and Belcher R: "Using Multiple Processors for Real-Time Audio Effects", *Proceedings of the AES 7th International Conference*, Toronto, Canada, May 1989.

[Jones, 1987] Jones M and Jubb A: "Console Automation and Digitalisation", from *Sound Recording Practice*, Oxford University Press, 1987, ISBN 0-19-311927-7.

[Rossing, 1990] Rossing T D: *The Science of Sound*, Addison-Wesley, 1990.

[Saunders, 1985] Saunders C, Dickey D, and Jenkins C: *The Future of Audio Console Design*, Solid State Logic Limited, Oxford, UK, 1985.

[SSL Ltd., 1988] SSL Ltd.: *G Series Master Studio System Console Operator's Manual*, Solid State Logic Limited, Oxford, UK, 1988.

[White, 1992] White P: "Spatialisation Systems", *Electronic Musician*, October 1992.

Chapter 3

Console Digitalisation

In any DMC, three sub-systems can be identified as shown in Figure 3-1. The control/display surface sends commands to — and displays data from — the control/display processing, which determines the control parameters to be conveyed to the audio signal processing.

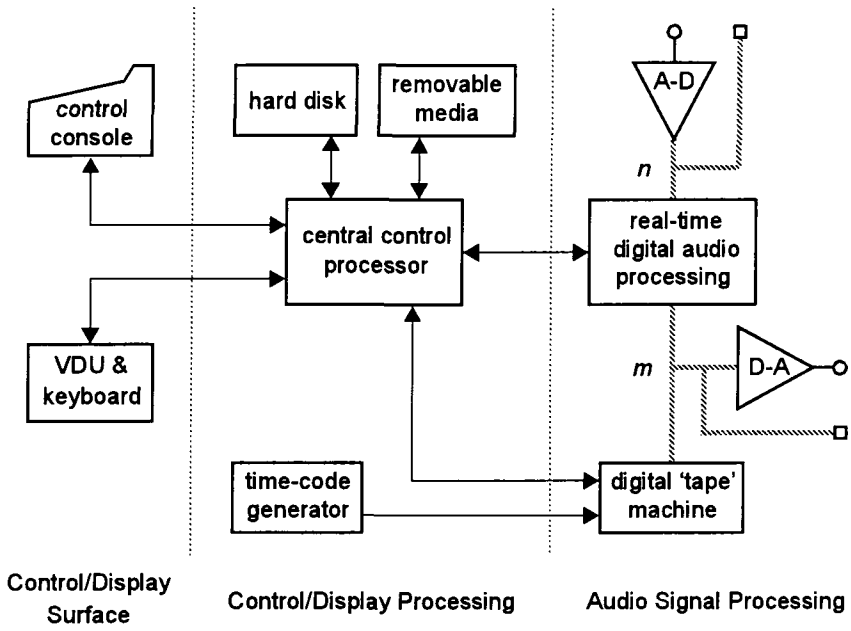

Figure 3-1: Block schematic representation of generic DMC.

3.1 Discrete-Time Signal Processing

In order to manipulate an audio signal in the digital domain, it must be first broken into evenly spaced time elements — *sampling* — and then expressed in binary form — *quantisation.*

3.1.1 Sampling

In Figure 3-2(a), a high sampling rate is intuitively adequate whereas in Figure 3-2(b) the output waveform (shaded) is at a new, lower, frequency. To explain this phenomenon fully the sampled signal must be studied in the frequency domain.

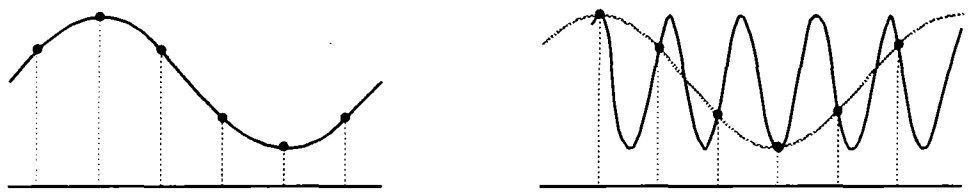

Figure 3-2: Illustration of the aliasing phenomenon in the time domain.

3.1.1.1 Periodic Sampling

Since the sampling impulses, f_s, are vanishingly short in duration, their spectrum is infinite as shown in Figure 3-3(a). The input amplitude-modulates these impulses, producing upper and lower side-bands as Figure 3-3(b). In Figure 3-3(c), f_s is too low for the input bandwidth, and there is a region of overlap (shaded) between the base-band and mirror image about f_s.

3.1.1.2 Aliasing

In Figure 3-3(d), reconstruction can no longer separate the resulting ambiguities. This phenomenon, termed *aliasing*, is similar to stroboscopic effects observed in rotating objects in motion pictures. Aliasing can be avoided only if f_s is greater than twice the original signal bandwidth, f_b. Typically the input is band-limited with a brick-wall low-pass filter.

3.1.1.3 Choice of Sampling Rate

The slope of available L-P filters forces designers to raise f_s above the theoretical minimum. In consumer products, rates close to twice 20 kHz are used. For vari-speed operation a higher rate is required since the first image frequency could pass the reconstruction filter as off-tape sampling rate falls. 48 kHz has been adopted for professional use as it has a simple relationship (3/2) to 32 kHz FM landlines and is far enough above 40 kHz for vari-speed.

3.1.2 Quantisation

Uniform quantisation divides the continuous range up into equal intervals of width Q. Since n bits may represent at most 2^n different values, the closest n-bit value must be determined.

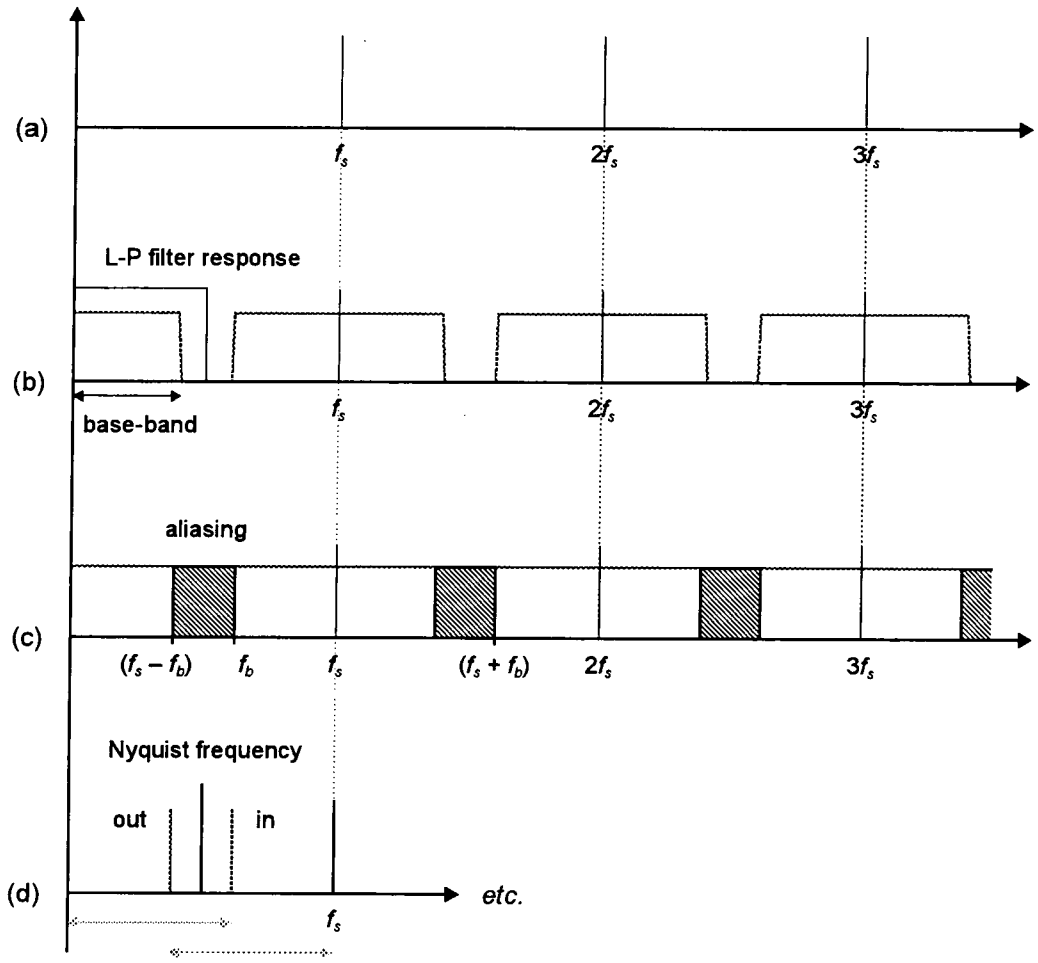

Figure 3-3: Illustration of the sampling process in the frequency domain.

3.1.2.1 Quantisation Levels

Whatever the exact analogue input value, it will be represented by the number at the centre of the interval in which it falls. The true continuous analogue voltage differs from its binary representation by an error which cannot exceed $\pm 0.5Q$, as detailed in Figure 3-4. This additive random noise is termed *quantisation noise*, the digital equivalent of analogue thermal noise.

3.1.2.2 Quantisation Noise

If the probability density function of the quantisation noise[1] is assumed to be uniform, a simple connection can be made between n bits and the signal-to-quantisation noise ratio

1. Then the mean squared error due to uniform quantisation is given by $\int_{-\frac{1}{2}}^{\frac{1}{2}} Q^2 dQ = \frac{1}{12}$.

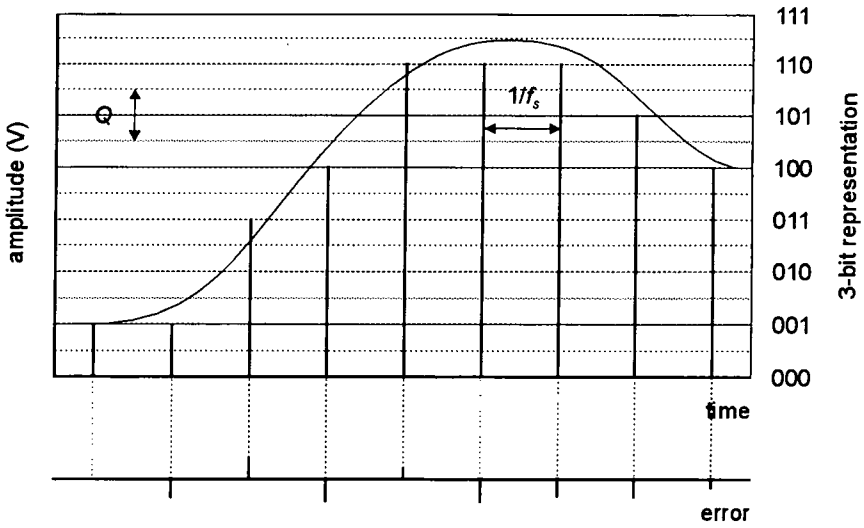

Figure 3-4: Quantisation noise shown as errors relative to original analogue waveform.

(SQNR). For example, if a sinusoid is sampled with a peak amplitude of $V = 2^{n-1}$, then $V^2 = 2^{2n-2} = 2^{2n}/8$, giving the SQNR for a full-scale sinusoid as:

$$\begin{aligned} SQNR &= 10\log_{10}\left(\frac{\frac{2^{2n}}{8}}{\frac{1}{12}}\right) \\ &= 10\log_{10}(2^{2n} \times 1.5) \\ &= 6.02n + 1.76 \text{ dB}. \end{aligned} \qquad \text{Eq. (3-1)}$$

Under these assumptions the maximum SQNR is approximately $6n$ dB: however, two caveats must be considered. First, it is often assumed, incorrectly, that quantisation noise may be treated as random uncorrelated noise. Second, if the analogue signal is not at maximum amplitude, the noise level still remains the same, rendering quantisation noise more apparent.

3.1.3 Data-Domain Conversion

Now that the essentially separate processes of sampling and quantisation have been described, it is instructive to examine the complete digital signal processing chain.

3.1.3.1 A-D Conversion

An analogue source, $x(t)$, is digitised by the process depicted in the schematic of Figure 3-5. Prior to sampling, $x(t)$ is first band-limited by a low-pass (L-P) anti-aliasing filter, attenuating

components greater than $f_s/2$. A sample-and-hold produces a pulse amplitude modulated (PAM) signal, having continuous amplitude at regular time intervals determined by f_s. The PAM pulses are then quantised to produce a pulse code modulated (PCM) signal, $x(n)$.

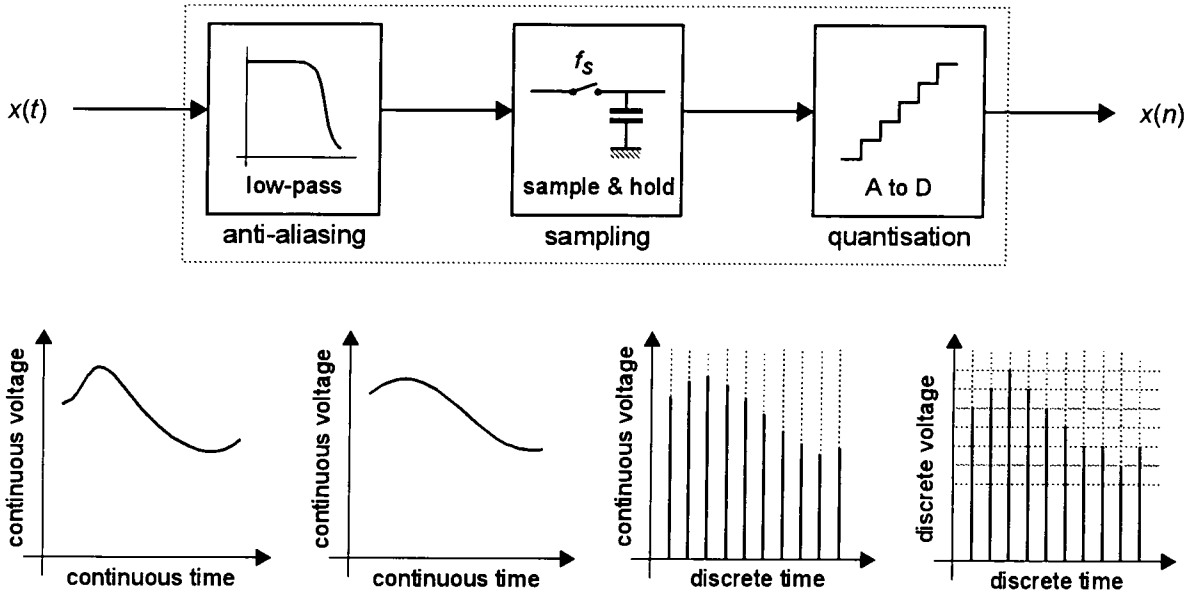

Figure 3-5: Schematic of the processes involved in analogue-to-digital conversion.

3.1.3.2 D-A Conversion

$x(n)$ may then be manipulated by DSP algorithms implementing ASP techniques to produce a discrete output signal, $y(n)$. Digital-to-analogue conversion (DAC) reverses the digitisation process, as Figure 3-6. The sample train, $y(n)$, is converted to analogue form, $y(t)$, by first transforming $y(n)$ to stepped voltage levels. Resampling produces another PAM waveform and a low-pass reconstruction filter then returns the time-domain signal $y(t)$.

3.1.4 Comparison of Processing Domains

Any DMC must be, firstly, economically viable and, secondly, operationally useful. As DSP is invariably more costly, it is useful to study the advantages presented by the digital approach.

3.1.4.1 Disadvantages of Analogue

In an analogue system, a signal can take on any one of a continuous set of values at any point in time. Degradations introduced in each processing section inevitably accumulate, setting an upper limit on the number of sections that can be cascaded. Impairments can be considered in

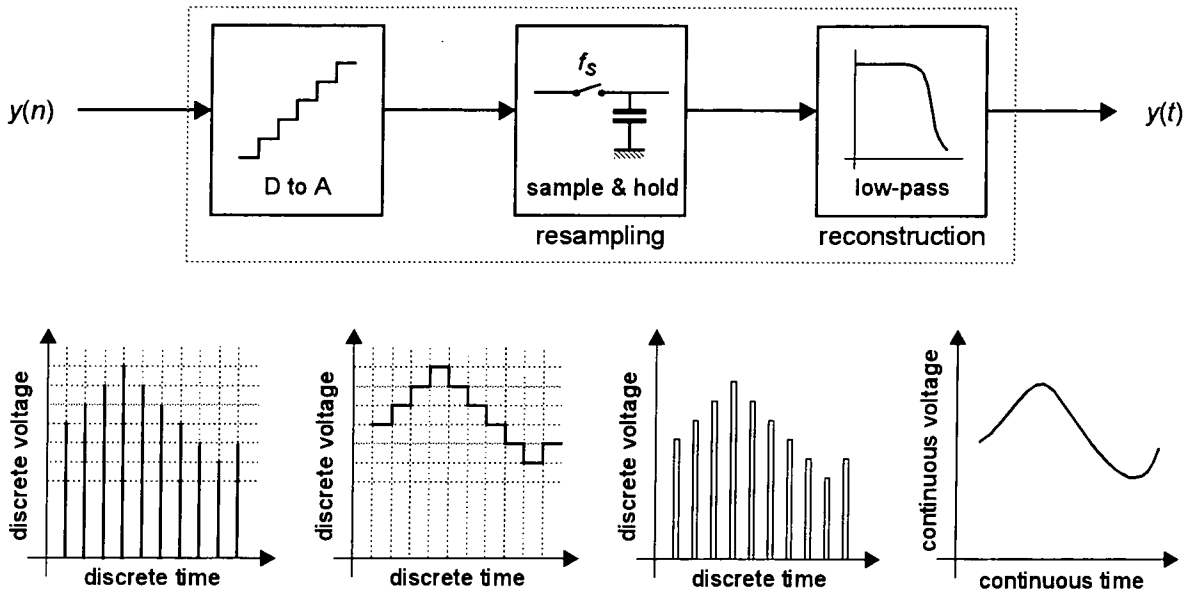

Figure 3-6: Schematic of the processes involved in digital-to-analogue conversion.

terms of additive noise or distortion, and timing instabilities such as group-delay effects and jitter. These imperfections can never be separated from the original signal.

3.1.4.2 Advantages of Digital

A digital signal can have only discrete values at discrete times. Once a signal has been digitised it can, theoretically, be processed indefinitely without further corruption. As DSP only uses multiplies, additions and delays, audio processing can be constructed with mathematical precision. Problems such as component drift can be virtually eliminated.

Time-division multiplexing (TDM) enables many audio signals on different time-slots to share one word-wide data bus as Figure 3-7. Delay elements may be used creatively or for the more subtle restoration of phase coherence in multi-microphone situations. Finally, direct interfacing at each machine boundary preserves signal fidelity until the final D-A conversion.

3.2 Digital Audio Processing

A mixing console can be specified by the number of input channels, *n*, and output groups, *m*. Figure 3-8 shows one possible configuration representative of multitrack mix-down.

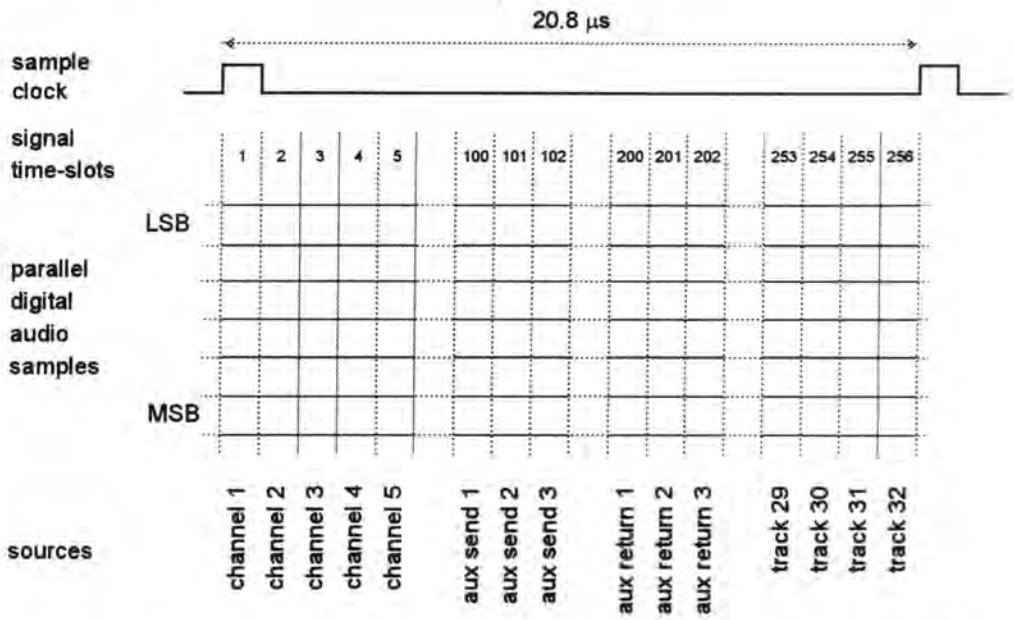

Figure 3-7: Time Division Multiplexing on a word-wide parallel bus.[a]

a. LSB represents the least significant bit, and MSB the most significant bit of the sample word.

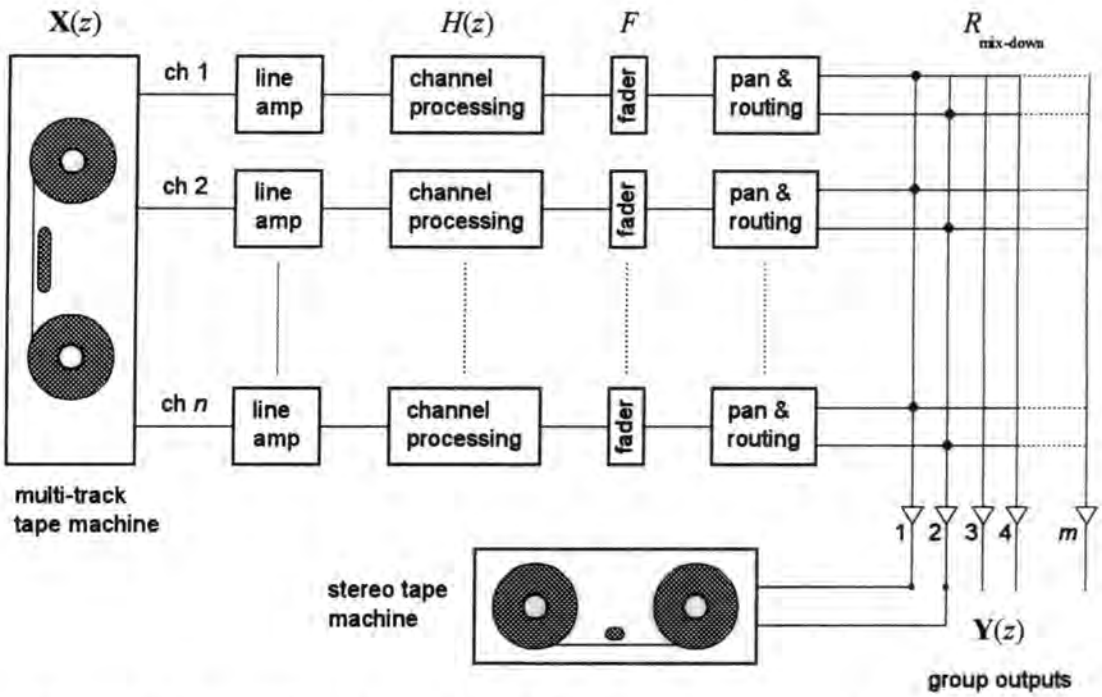

Figure 3-8: Simplified schematic of a mixing console during mixdown to stereo.

3.2.1 Matrix Representation

The group outputs of a DMC can be represented in simplified form (i.e. partially ignoring channel pan) by the following equation. Note, that neither R or F are dependent on z:

$$\mathbf{Y}(z) = RFH(z)\mathbf{X}(z) \qquad \text{Eq. (3-2)}$$

3.2.1.1 Group Output Vector — Y(z)

Equation (3-3) shows the group output vector, Y(z), where $y_t(z)$ is the output from track t of m:

$$\mathbf{Y}(z) = \begin{bmatrix} y_1(z) \\ y_2(z) \\ \cdot \\ y_t(z) \\ \cdot \\ y_m(z) \end{bmatrix} \qquad \text{Eq. (3-3)}$$

3.2.1.2 Routing Matrix — R

R, the routing matrix, is shown in Equation (3-4). If p_c is the pan coefficient for channel c, then $r_{t,c}$ is either p_c, $t = 1, 3, \ldots$ or $(1 - p_c)$, $t = 2, 4, \ldots$ if c is routed to track t:

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdot & r_{1,c} & \cdot & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdot & r_{2,c} & \cdot & r_{2,n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ r_{t,1} & r_{t,2} & \cdot & r_{t,c} & \cdot & r_{t,n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ r_{m,1} & r_{m,2} & \cdot & r_{m,c} & \cdot & r_{m,n} \end{bmatrix} \qquad \text{Eq. (3-4)}$$

3.2.1.3 Fader Coefficient Matrix — F

The fader coefficient matrix, F, of Equation (3-5), denotes the values of the fader coefficients, f_c, for each channel c:

$$F = \begin{bmatrix} f_1 & 0 & \cdot & 0 & \cdot & 0 \\ 0 & f_2 & \cdot & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & f_c & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & 0 & \cdot & f_n \end{bmatrix} \qquad \text{Eq. (3-5)}$$

3.2.1.4 Channel Processing Matrix — $H(z)$

Another diagonal matrix, $H(z)$, specifies the processing within each channel:

$$H(z) = \begin{bmatrix} h_1(z) & 0 & \cdot & 0 & \cdot & 0 \\ 0 & h_2(z) & \cdot & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & h_c(z) & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & 0 & \cdot & h_n(z) \end{bmatrix} \qquad \text{Eq. (3-6)}$$

where $h_c(z)$ is the characteristic of j cascaded second-order filter sections in channel c:

$$h_c(z) = \prod_{i=1}^{j} \frac{a_{0_{i,c}} + a_{1_{i,c}} z^{-1} + a_{2_{i,c}} z^{-2}}{1 + b_{1_{i,c}} z^{-1} + b_{2_{i,c}} z^{-2}} \qquad \text{Eq. (3-7)}$$

Such bi-quadratic (or bi-quad) sections are usually implemented in the transposed direct form II of Figure 3-9 [McNally, 1981]. Advantages of this second-order topology include less tendency to overflow and minimal limit cycles effects.[1] Although IIR filters are sensitive to coefficient quantisation, this ordering of DSP primitives is minimally effected.

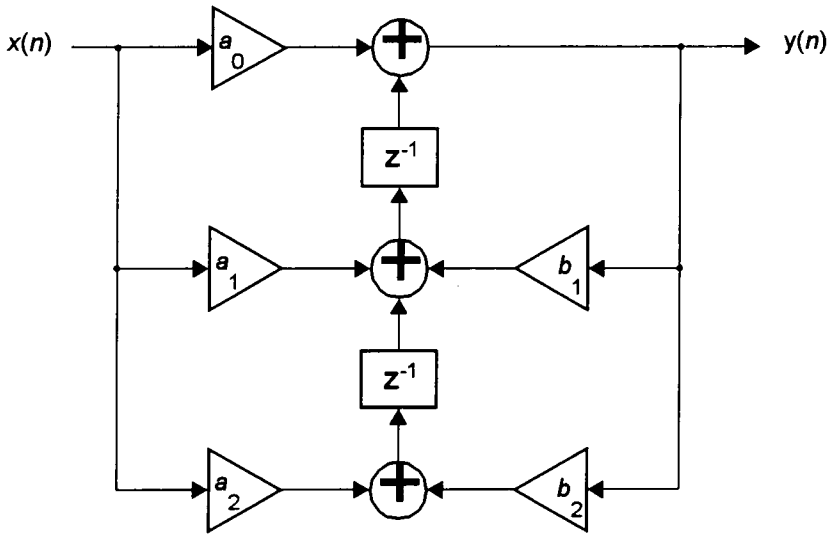

Figure 3-9: Transposed direct form II, or transposed canonical, bi-quad section.

3.2.1.5 Input Vector — X(z)

To complete the matrix representation the input vector, **X**(z), contains the input, x_c, to each console channel c of n:

1. As the dominant truncation takes place at one node.

$$\mathbf{X}(z) = \begin{bmatrix} x_1(z) \\ x_2(z) \\ \cdot \\ x_c(z) \\ \cdot \\ \cdot \\ x_n(z) \end{bmatrix} \qquad \text{Eq. (3-8)}$$

3.2.1.6 Summary

If the dimensions of a matrix, $X = (x_{ij})$, $1 \le i \le r$, $1 \le j \le c$, are denoted by X_{rc}, then, from matrix algebra, the product $AB = C_{mn}$ is defined only if A_{mp} and B_{pn}. $A_{mp}B_{pn}$ requires mpn multiplications ($\times$) and $m(1-p)n$ additions (Σ), or alternatively $m(1-p)n$ MAC and mn $\times$. If, however, A or B is a diagonal matrix then mn $\times$ alone is sufficient.

From this analysis, implementing $\mathbf{Y}(z)$ in Equation (3-2) requires $n(m+2)$ $\times$ and $m(n-1)$ Σ, with an additional $2n$ $\times$ and n Σ for R and a further $5jn$ $\times$ and $4jn$ Σ for $H(z)$. Consequently, a DMC configured with $n = 96$, $j = 6$, $m = 24$, and $f_s = 48$ kHz needs a minimum of 2.24×10^8 MAC/s and 4.26×10^7 $\times$/s, equivalent to a nominal 267 MIPS.[1]

3.2.2 Processing Redundancy

Recognising that the routing matrix R in Equation (3-4) is normally sparse achieves a major reduction in hardware: every possible input is never routed to every possible output.

3.2.2.1 Tracking

When used for tracking, the aim is normally to keep the mix uncommitted by preserving discrete microphone signals on tape. As a result there is a one-to-one correspondence between inputs and outputs, so $n = m$. R now has dimension R_{mm} and the number of DSP operations required to implement this configuration is reduced by a factor of m/n to 67 MIPS.

1. Assuming, not unreasonably, both single-cycle MAC and multiply instructions.

$$R_{\text{tracking}} = \begin{bmatrix} r_{1,1} & 0 & \cdot & 0 & \cdot & 0 \\ 0 & r_{2,2} & \cdot & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & r_{t,c} & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & 0 & \cdot & r_{m,n} \end{bmatrix}, n = m \qquad \text{Eq. (3-9)}$$

3.2.2.2 Mixdown

During mixdown, all channels are routed to two groups as shown in Figure 3-8.[1] R is sparse with only 2 rows populated and its effective dimension is reduced to R_{2n}. Processing resources required drops to 166 MIPS, some 60% of nominal. Experience suggests that channel processing is sparsely used, from 0% on 'classical' to around 30% for 'pop' work.

$$R_{\text{mix-down}} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdot & r_{1,c} & \cdot & r_{1,n} \\ r_{2,1} & r_{2,2} & r_{2,3} & \cdot & r_{2,c} & \cdot & r_{2,n} \\ 0 & 0 & 0 & \cdot & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & 0 & \cdot & 0 \end{bmatrix} \qquad \text{Eq. (3-10)}$$

3.2.3 Audio Performance

As both consumer media and professional digital production equipment have displayed audible imperfections, it is important to consider the main sources of such problems.

3.2.3.1 Sampling Rate & Frequency Response

The highest fixed sampling rate specified by international standards is 48 kHz. Even at this rate, the construction of low-pass analogue reconstruction filters providing over 100 dB attenuation is extremely difficult. Given that 48 kHz is only 8 kHz higher than a single octave above the extremes of human hearing, the extent of the problem becomes clear.

In *oversampling*, the DAC input is sampled at 4 or 8 times f_S, allowing the radical slope to be pulled back into the digital domain. Remaining filter requirements are reduced to

1. The switch matrix for the track monitor and cue mixes are invariably similar to $[S]_{\text{mix-down}}$

that easily delivered by standard analogue design. As the market has grown, quality convertors have become available from Crystal and Motorola among other manufacturers.

3.2.3.2 Word-Length & Dynamic Range

The 96 dB SQNR range provided by true 16-bit conversion serves well in a single pass situation. Due to noise addition from multiple inputs an absolute minimum of 18-bits resolution is required for mix functions. In DSP, the major problem is ALU rounding error: results longer than the supported word-length are simply rounded off when stored.

Multiplication of any word-wide coefficient will always yield such a result. Mix data may be multiplied 20 or more times, accumulating rounding degradation with each multiply. There is a general acceptance that the minimum mantissa for professional ASP is 24-bits, giving a theoretical dynamic range of 144 dB for fixed point implementations [Eastty, 1986].

3.2.3.3 Real-Time Operation

Opportunities for innovation are irretrievably lost if DMCs merely mimic analogue consoles. It is a simple matter to change ASP coefficients in order to tailor frequency characteristics to orchestral or popular music recording. Consider the channel EQ in Equation (3-7): the response of each bi-quad section is entirely determined by the coefficient sets a and b.

Processing delays are frustrating: in professional audio, such latency is completely unacceptable. An ASP throughput lag of only 1 ms can make ordinary studio procedures very difficult [Deutsch, 1982]. If the lag should increase beyond this limit, or vary depending on the type of simultaneous operations, overdubbing and track-bouncing become impossible.

3.3 Control and Display Processing

As all housekeeping tasks including mix-down automation are related to the same time reference, normally SMPTE timecode, one central computer system should manage all.

3.3.1 Control Rate vs. Audio Rate

A major design choice is how to apportion the decoding of control movements and switch closures between control data and audio sample domains. It is generally held that, for audio

consoles, panel scanning need only be performed at a rate of 100-200 Hz. Consequently, the majority of control data should be processed in the control domain.

This approach leaves only coefficient interpolation for the more costly audio processing environment. The command structure should be able to control up to 128 audio channels, including groups, masters and auxiliaries as found in comprehensive music recording consoles, plus the return of all necessary status and display data.

3.3.2 Control & Display Response

The entire control system must produce apparently instantaneous results: ASP must follow control movement in a natural fashion, with no *perceptible* lag. Acceptable delay between command initiation and ASP response is in the region of 4 ms [Saunders, 1985]. For mutes and other push-button controls, the impression of immediate response is even more critical.

As the brain superimposes any visual lag, display update must also appear instantaneous [Gulick, 1989]. Initial display response has somewhat more latitude, owing to differences in auditory and visual acuity. Investigations at SSL have determined the maximum acceptable display lag to be in the order of 20 ms [Saunders, 1985].

3.3.3 Resolution & Psychoacoustics

Two kinds of resolution must be considered for a given audio control — *static* resolution and *dynamic* resolution. For example, most researchers agree that a listener cannot pinpoint more than 15 discrete stereo positions between hard left and hard right: many put this number at 9. The higher number safely determines the static resolution requirements of a stereo pan-pot.

If an audio signal is dynamically panned using 15 steps, the jumps between discrete positions are clearly audible. Dynamic resolution requirements of stereo pan are, therefore, much greater than the static requirements. Sufficient dynamic resolution of 2^6 to 2^8 steps must be provided to emulate continuous variability for many functions [Saunders, 1985].

3.3.4 Automation

Console automation systems perform many different functions apparently simultaneously. Here, that old chestnut of how much automation is actually useful is reconsidered.

3.3.4.1 Reset Automation

It has been normal practice to have a 'lock-out' between studio sessions to ensure that console controls are not inadvertently adjusted. Automation SAVE and RESTORE commands allow sessions to be interlaced with the knowledge that previous settings can be exactly reproduced. To be operationally useful, such commands should take no more than a few seconds. Reset automation thus helps offset production cost by allowing more efficient use of studio time.

3.3.4.2 Dynamic Automation

Dynamic automation allows control movements during mix-down to be refined incrementally. Current systems operate at roughly 0.25 to 1.0 SMPTE frame accuracy, storing information for approximately 100 dynamic and 200 momentary events. Total dynamic automation (TDA) extends this approach to embrace *all* control movements across the console.

An advanced console has 30 variable functions in addition to faders and 70 switch functions in addition to mutes in each channel. TDA achieving control resolution and repeat accuracies similar to 'fader only' dynamic automation will require somewhere in the region of 50 times the processing power and data storage of a standard reset system.

3.3.4.3 Control Implications

Beyond the economics of this, the designer must also tackle the control implications. Automation systems have evolved over the years to give the engineer many data editing modes such as RELATIVE, UPDATE, and AUTOTAKEOVER, which depend on dedicated status switches and displays. It is left to the reader to envision a suitably self-explanatory and efficient means of providing such facilities for all control functions on a large DMC.

3.4 Surface Ergonomics

Ergonomics, or *biotechnology*, is the study of the man-machine interface (MMI) especially in terms of physiological, psychological and technological requirements.

3.4.1 Overview

As control and display conventions have survived and evolved, analogue console design has achieved a high degree of ergonomic integrity. Two factors are particularly important here.

3.4.1.1 Patterns

Pattern recognition allows the operator to gaze across a properly designed control surface and acquire a quick grasp of the general situation. On a standard analogue console, clusters of controls serving the same purpose on different channels are easily identified by their *location* and *physical characteristics* [Kvalseth, 1983].

For example, all of the equaliser controls are usually positioned in the same row. The spacing of the controls within the equaliser, the colour-coding of their knobs, and the position and status indication of their switches is consistent. It is these patterns that make it easy to differentiate between an equaliser and a compressor, even on a high density control surface.

3.4.1.2 Anomalies

Anomalies are the complement of patterns: the anomaly is that which stands out from the pattern. Anomalies serve as visual hooks, directing the operator's focus to a control that may deserve attention [Pheasant, 1988]. Glancing across the control surface effortlessly detects anomalies such as the presence of an EQ IN lamp.

When the operator focuses on that equaliser, anomalies within its local pattern, such as a mid-range boost control set fully clockwise, instantly convey enough information for the human brain to determine if a closer look is warranted. On closer examination, the scale surrounding the knob provides precise definition of the control's value.

3.4.2 Control Layout

If all console controls are within reach, then each is just one action away once relative positions are memorised. Controls that lie beyond reach are quite a different story, however.

3.4.2.1 Ideal

For ideal monitoring during mix-down, the engineer will sit centrally between the studio monitors. Controls constantly being adjusted should be placed under his hands and the most important visual indicators straight ahead. Moving out from the centre line are controls requiring occasional adjustment and second priority meters and lights. Everything other than re-adjustment of individual faders can be reached without moving up and down the console.

3.4.2.2 In-Line

The vast majority of analogue production consoles follow the 'in-line' philosophy. Perhaps the main drawback of this compromise solution to ergonomic problems has been the exchange of console length for console depth. As the demand for functionality has steadily increased, as many control set sets as can be physically accommodated are provided in the channel strip. In most current layouts, it is possible to reach routing and some EQ controls only by standing up.

3.4.3 Display Layout

Display information may be provided around or above the control on some kind of legend, on a VDU screen or both. Every indication must be obvious in meaning and easily interpreted.

3.4.3.1 Resolution & Psychoacoustics

The degree of variability provided in a particular processor must be matched by the display. This is an inherent property of mechanically linked control sets but often overlooked in DMC. If control and audio processing emulate continuous variability and the corresponding display only supports a limited resolution, psychoacoustic phenomena will convince even the most experienced listener that they are hearing 'jumps' where none exist [Pheasant, 1988].

3.4.3.2 Definition & Pattern Recognition

While numeric displays provide excellent discrete definition, pattern recognition is extremely low. Differences between one bar-graph column and another instantly convey more useful information than read-outs of '30' and '80'. The height of a vertical column meaning quantity is intuitively clear but the best way to convey, say, EQ parameters is not immediately obvious. Except for channel and timecode indicators, the worst option is to display numbers directly.

3.4.3.3 VDU Displays

The VDU screen is capable of representing virtually any display and has long been used in fader automation. Two screens will fit within the width of a compact console, Figure 3-10, displaying a huge amount of information — combined or superimposed. For example, the left screen may carry time of day, timecode and machine status, channel levels, and mute flags.

The right screen may show assignable control settings with channel processing shown as a response curve or knob pictorial. During mix-down, all track levels or automated fader

levels could be displayed across both screens together with PPM ballistics. Recent products from SSL Ltd., such as *Senaria*, and *Optimix*, use two screens in such a fashion.

3.4.4 Long vs. Virtual Surfaces

Ergonomic testing has confirmed that operating a conventional 'long' console requires many eye movements. Very few actions are needed as all controls are effectively dedicated. Assessment of identical tasks on a variety of virtual systems by local expert operators shows that the number of eye movements is surprisingly similar in each case [Stavrou, 1993].

Given sufficient familiarity, the time required for operations is comparable, although observers cannot see the mental gymnastics involved. If the number of controls is reduced the number of actions and keystrokes required is inevitably increased. Ergonomic testing of surface demands confirms that the weak link between operator and DMC is an ergonomic one.

3.5 Assignable Mixing

Recognising that the operator can examine and operate only a limited number of controls at one time has brought about the concept of *assignability*.[1]

3.5.1 Overview

The standard way of providing free-grouping in professional consoles is by latching push-buttons. Besides occupying considerable space, this vast number of switches is a serious cost element. In many designs channel-to-track routing buttons have been centralised into a single assignable panel. Figure 3-10 shows how this concept can be extended across the console.

Processing functions are also centralised: access to a particular channel is made by pressing the 'access' button adjacent to each fader. One set of controls and displays is moved from channel to channel, enabling the engineer to remain in the central listening position. This is clearly better than dodging back and forth to confirm that the stereo image is correct.

1. Note that this is the sharing of controls and displays only — not the underlying processing resource.

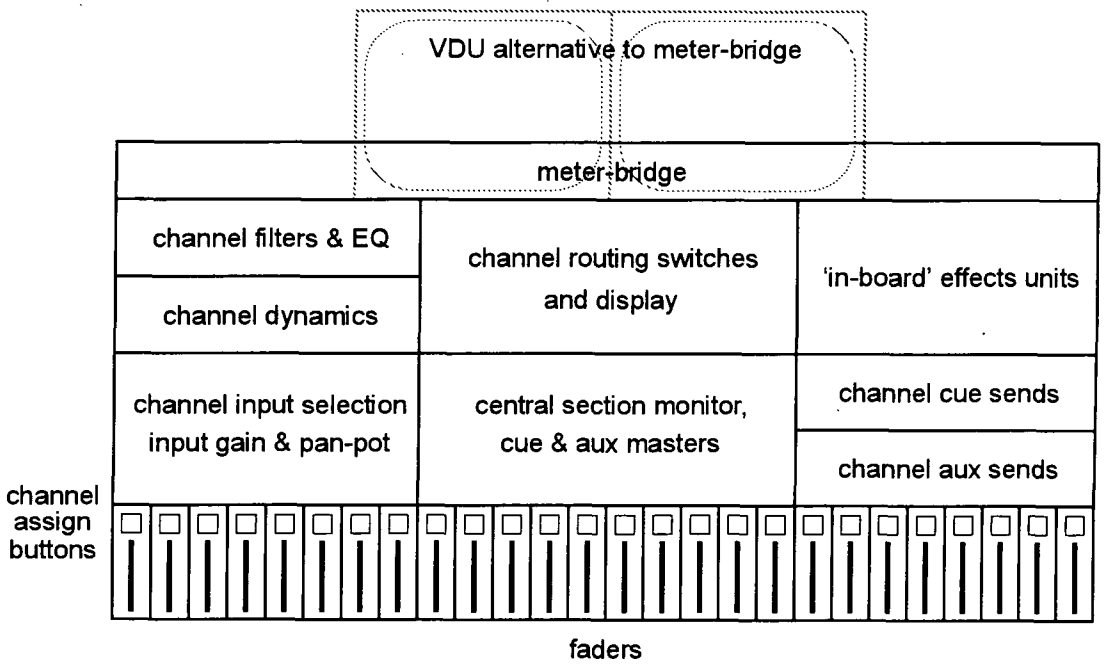

Figure 3-10: Outline diagram of the assignable console concept.

3.5.2 Implementation Issues

A control set may perform the same function for a range of channels or different functions for the same channel. Conventional console design weighs heavily in favour of the later option.

3.5.2.1 Per-Channel

Sharing local control sets between different functions provides a means of temporarily 'locking' any desired function. The only requirement is that a sufficient number of parameters be provided to accomplish all necessary control functions. However, this arrangement denies the operator the primary pattern recognition clues normally used to differentiate functions.

3.5.2.2 Full-Function Master

An alternative scheme is to provide two or more assignable full-function control sets. Each one has a fixed location and readily identifiable physical characteristics, providing Simultaneous access to one channel's processing. This integration of controls and displays is less physically and mentally taxing than a separated system which splits the operator's focus.

3.5.2.3 Control Layout

The proximity of 'per channel' assignable control sets to a channel's 'permanently assigned' controls is a definite plus. On the other hand, the removal of assignable functions from the

channel frees channel real estate for more 'permanently assigned' controls. Control spacing can also be increased to allow the use of displays with greater clarity and definition.

3.5.2.4 Comparison

It is not clear which feature is more valuable: the ability to adjust all interacting parameters on a given channel, or certain values of specific functions on different channels simultaneously. One needs to consider how many parameters can be simultaneously controlled and how many 'full-function master' control sets are provided. Substantial differences of opinion exist between mixing engineers with considerable experience and superb credentials [Jones, 1985].

3.5.3 Disadvantages of Assignable Faders

The main drawback of assignability is betrayed by its name: access to ASP functions is no longer immediate. This applies not only to *control* access, but *display* access as well. While it is not desirable for a DMC control surface to map directly onto a conventional in-line configuration, ergonomic problems in the MMI must still be minimised.

It is generally held that, irrespective of whatever else is assignable, there still has to be a fader dedicated to every input channel.[1] The more that automated mixdown is used, the less weight this argument holds. Now the channels to be manipulated in one pass of the multi-track can be assigned to a limited number of faders: that number is unlikely to exceed 10.

3.5.4 The Virtual Console

The term 'virtual console' has been loosely applied to any console that separates control and ASP functions. Strictly, the term refers to a DMC architecture that does not, in fact, exist.

3.5.4.1 Screen-Based Digital Mixing

One approach to the virtual console is to replace the real control surface by a picture on a screen manipulated by a mouse or similar pointing device. Whilst appearing most ingenious, this approach forces the operator to access controls in a purely serial fashion. Such designs are not usable in real-time and are essentially editing rather than true mixing systems.

1. Opponents of assignable mixing argue that one might want to touch any control instantly.

Edit stations have been proposed on which the engineer may prepare control sequences off-line. Unlike video, where still frames have meaning, one can hardly perform audio mixing without auditioning ASP operations in real-time. Clearly, a far more advanced MMI is necessary than that offered by purely screen-based digital console designs.

3.5.4.2 A Very Small Console

In the *VSC60* virtual surface prototype Stavrou and his colleagues redesigned a tactile control surface from first-principles, discarding many axioms of surface design. To perform the same tasks as conventional consoles needs a radical shift in allocation of human resources. For example, eye movements are reduced to just 0.39% of operating time [Stavrou, 1993].

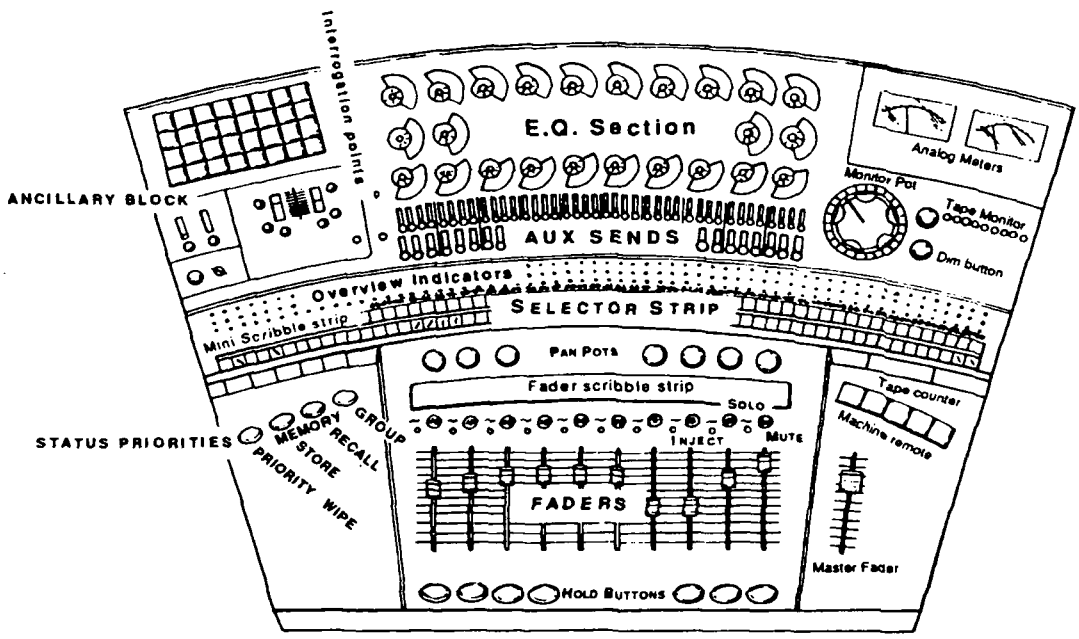

Figure 3-11: Outline drawing of the control layout of the *VSC60* (courtesy AES Inc.).

At first glance the *VSC60* in Figure 3-11 looks like a relatively conventional 10-channel desk. Secondary controls such as routing, filters, and dynamics are placed in a self-assigning Ancillary Block. Note the lack of multi-function controls: even from across the control room, traditional visual pattern and anomaly clues are preserved intact.

3.6 Conclusions

Recent trends suggest the real possibility of an integrated digital sound production chain between studio microphones and PWM-powered loudspeakers. AES/EBU Serial Transmission Formats (ANSI S4.40-1985), co-axial and fibre-optic consumer standards, and MADI are major steps towards this goal. What is still required is an all-digital studio console.

A vital stage in DMC implementation is the ASP sub-system itself. In this chapter, the author investigates this part from a novel perspective. Not only does this approach determine the level of real-time DSP necessary for a 96:24 DMC (267 MIPS), but it also reveals how to exploit inherent redundancy during tracking (67 MIPS) and mix-down (166 MIPS).

Much effort has been invested in advancing digital ASP techniques. In contrast, the layout of the control surface, or man-machine interface, is not generally regarded as greatly affecting usability. Freedom from the constraints imposed by mechanical linkage, however, creates a number of potential opportunities to improve the man-machine interface (MMI).

Assignable mixing reduces the graphic density of the control surface and the total number of control sets required for a given configuration. Digital implementation makes TDA a technical reality although at significant cost uplift. In a mature digital architecture, the only restrictions should be the amount of processing resource and the imagination of the engineer.

On a virtual, or fully-assignable, console out-of-reach controls are physically removed: only the correct sequence of keystrokes brings them within 'reach'. Each virtual console uses different sequences of keystrokes to access out-of-reach control sets. It would appear that the MMI is becoming more maladroit with each advance in surface technology.

In Stavrou's *VSC60*, a selector strip adds to the engineer's orientation, altering this 10-channel desk (220 knobs) as though the engineer moved to that position. Some multi-function designs require 400 keystrokes to bring all these parameters within reach. Notably, such improvements are solely as a result of surface design and not advanced technology.

At the start of this chapter, Figure 3-1 showed the DMC as consisting of three major sub-systems. The next part of this thesis on digital mixing consoles will concentrate on the third of these. As the logical starting point for any M-P design is the choice of processing element, the next chapter focuses on this specific aspect of the ASP computation engine.

3.7 References

[Deutsch, 1982] Deutsch D: *The Psychology of Music*, Academic Press, 1982, ISBN 0-12-213562-8

[Eastty, 1986] Eastty P: "Digital Audio Processing on a Grand Scale", *81st Audio Engineering Society Convention*, Los Angeles, California, USA, September 1986.

[Gulick, 1989] Gulick W L, Gescheider G A, and Frisina R D: *Hearing: physiological acoustics, neural coding, and psychoacoustics*, Oxford University Press, 1989, ISBN 01-95043-073.

[Jones, 1985] Jones R: "The Virtual Console", *Recording Engineer/Producer*, pp. 49-53, October 1985.

[Kvalseth, 1983] Kvalseth T O: *Ergonomics of Workstation Design*, Butterworths, 1983, ISBN 0-408-01253-6.

[McNally, 1981] McNally G W: *Digital Audio: Recursive digital filtering for high-quality audio signals*, BBC Research Department, Internal Report 23/1981, 1981.

[Pheasant, 1988] Pheasant S: *Bodyspace Anthropometry, Ergonomics and Design*, Taylor & Francis, 1988.

[Saunders, 1985] Saunders C, Dickey D, and Jenkins C: *The Future of Audio Console Design*, Solid State Logic Limited, Oxford, UK, 1985.

[Stavrou, 1993] Stavrou M P: "A New Approach to Assignable Control Surface Design", *Journal of the Audio Engineering Society*, Vol. 41, No. 7/8, pp. 556-563, July/August 1993.

[Strawn, 1985] Strawn J (ed.): *Digital Audio Signal Processing: An Anthology*, Kaufmann, 1985, ISBN 0-865-76082-9.

[Swettenham, 1991] Swettenham R: "Digitally Controlled Analogue Mixing: 10 years on", *Studio Sound*, pp. 23-27, April 1991.

[Watkinson, 1988] Watkinson J: *The Art of Digital Audio*, Focal Press, 1988, ISBN 0-240-51270-7.

Chapter 4

Programmable DSP Devices

Many semiconductor manufacturers have developed DSP VLSI suitable for digital ASP. Here, the principle features of available processing technologies are analysed and compared.

4.1 Introduction

Differences between reduced-instruction-set computers (RISCs), superscalar[1] μPs, and programmable DSPs result mainly as a consequence of real-time design decisions.

4.1.1 History of DSP Devices

True programmable DSPs began to appear 25 years ago with the Bell Labs DSP1. In 1984, Texas Instruments (TI) introduced the popular TMS32010 with a full suite of development software. Motorola, a relative newcomer, shipped its first DSP in 1987. As shown in Figure 4-1, peak performance has doubled approximately every 3 years. Recent developments have included several devices directly supporting multi-DSP topologies (see Appendix A).

4.1.2 General-Purpose vs. DSP Processors

DSPs are, in effect, RISCs optimised for the execution of DSP instructions. A relatively large area of silicon is devoted to an array multiplier and a barrel shifter [Nishitani, 1986]. In contrast, these operations are statistically unimportant for G-P programs. Most G-P processors have a limited number of internal registers and lack a fast multiply instruction.

Such restrictions led to the development, firstly, of specialised real-time DSP sub-systems, such as the BBC's bit-slice COPAS-2[2] [McNally, 1982], and then programmable VLSI DSPs. Differences to G-P μPs include: dual-bus architectures, DSP-related addressing modes, and serial interfaces to serve AES/EBU and ADC/DAC peripherals.

1. Superscalar architecture improve performance by exploiting additional instruction-level parallelism.
2. COPAS-2 technology led to the development of the first commercial DMC, the Neve *DSP1*.

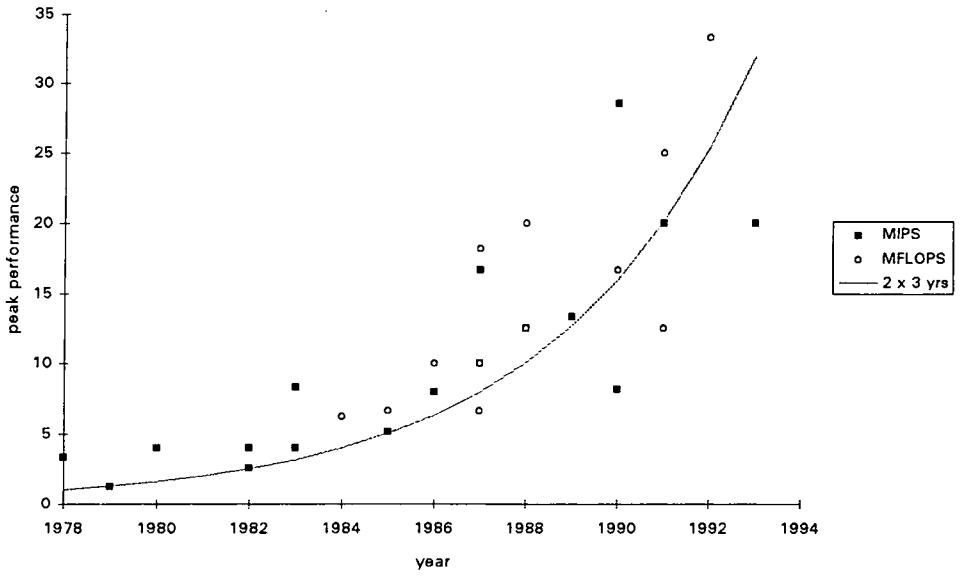

Figure 4-1: Peak performance of commercially available DSP devices.

4.1.3 Benchmarks

MIPS comparisons of DSPs have two disadvantages. Firstly, MIPS is dependent on the actual instruction set. Secondly, MIPS can vary widely between different code on the same device. One of the most basic metrics is the multiply and accumulate (MAC) operation, listed in Table 4-1. While small ASP routines of non-trivial program complexity, such as the biquad filter, can be more representative, the MAC determines the maximum rate of DSP algorithms.

4.2 Arithmetic

When evaluating programmable DSPs, one of the first features to consider is numeric format. In this section, the characteristics of fixed-point and floating-point arithmetic are compared.

4.2.1 Fixed-Point

Considerable attention must be given to the limitations of fixed-point arithmetic. Precision is lost through quantisation arising from both data-domain conversion and multiplication.

4.2.1.1 Multiplication and Quantisation

Quantisation errors occur during multiplication since the number of bits required to specify the product is equal to the sum of the number of bits in each operand. If operands have n bits,

Company	Part No.	Release Date	MAC (ns)	Precision[a]	
				Fixed-Point	Floating-Point
Analog Devices	ADSP21020	1991	80	32/40	32 or 40/40
AT&T	DSP32C	1988	80	16 or 24/40	32/40
	DSP16A	1988	33	16/36	—
	DSP3210	1992	30	16 or 24/40	32/40
Motorola	DSP56001	1987	60	24/56	—
	DSP96002	1990	60	32/64	44/96
	DSP56116	1991	50	16/40	—
TI	TMS320C30	1988	50	16/32	32/40
	TMS320C50	1990	35	16/32	—
	TMS320C40	1991	40	24/32	32/40

Table 4-1: DSPs representative of the most important architectural techniques.

a. x/y: where x is the number of bits in the multiplier operands, and y is the number of bits in ALU operands.

both ALU and accumulators must be $2n$ bits wide to preserve the full product. As only n bits are ultimately stored, the programmer must determine which bits to keep. Storing the lowest order n bits possible without causing overflow maintains maximum operand precision.

4.2.1.2 Overflow

Overflow can occur in two ways in a fixed-point DSP. Either an accumulator overflows, producing large magnitude/sign errors, or high-order bits are discarded when it is stored. These effects can be ameliorated by greater precision for operands, additional accumulator headroom, and saturation limiters. There is a direct trade-off between probability of overflow and number precision as variables must be scaled so that overflow is sufficiently unlikely.

4.2.2 Floating-Point

Compared with fixed-point, the role of floating-point arithmetic is to identify the best bits in the result to store and at the same time keep track of scaling changes.

4.2.2.1 Overview

A floating-point number, x, is made up of a mantissa, $M(x)$, and an exponent, $E(x)$, such that:

$$x = M(x) \times 2^{E(x)} \qquad \text{Eq. (4-1)}$$

Multiplying two floating-point numbers, x and y, gives the product, z, as:

$$z = M(x) \times M(y) \times 2^{E(x)+E(y)} \qquad \text{Eq. (4-2)}$$

A floating-point multiplier must contain both a multiplier and an adder for exponents. In the Motorola DSP96002, for example, the floating-point multiplier takes operands with up to 44 bits (32/11), with the full precision result (64/11) stored in 96-bit registers.

4.2.2.2 Renormalisation

When two mantissas are multiplied together, several of the most significant bits in the result may be identical and, hence, redundant. A shift can dispose of these extra bits and the exponent adjusted to track scaling. In virtually all floating-point DSPs, such 'renormalisation' is done automatically in hardware after each arithmetic operation [Motorola, 1989b].

4.2.2.3 Dynamic Range

Figure 4-2 shows that the dynamic range of floating-point arithmetic is much greater than that of fixed-point. 24-bit fixed-point approaches an upper bound SQNR of +144 dB whereas floating-point with a 24-bit mantissa achieves the best performance possible with 24-bits. An 8-bit exponent is sufficient to maintain this SQNR over a dynamic range of ≈1500 dB.

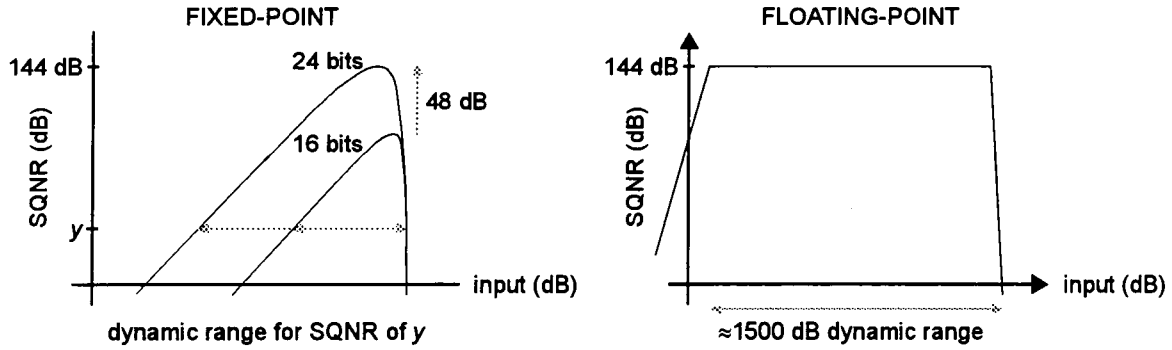

Figure 4-2: SQNR and dynamic range, shown as a function of signal level.

4.3 Memory

Simultaneous memory accesses are also crucial to the performance of DSPs. This requirement is accomplished through a combination of parallel memory banks and rich addressing modes.

4.3.1 Parallel Memories

In conventional von Neumann architectures, an instruction and its operands have to be fetched sequentially. Consequently, many DSPs have employed the Harvard architecture shown in Figure 4-3. Two memories and two busses permit instruction and operand fetch to coincide.

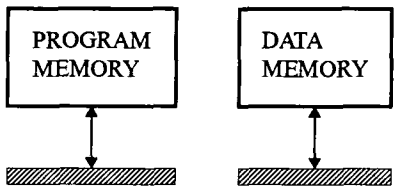

Figure 4-3: Basic Harvard architecture, as used in many early DSPs.

4.3.1.1 First Modification — Data Stored in Program Memory

The first modification to the basic Harvard architecture (see Figure 4-4) permits data stored in program memory to be used in arithmetic instructions (e.g. AT&T DSP32C [Fuccio, 1988]). Instructions with two or more operands (i.e. MAC) still require two memory cycles.

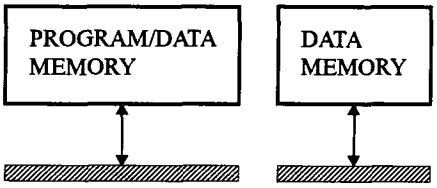

Figure 4-4: First modification — data stored in program memory.

4.3.1.2 Second Modification — Multi-Ported Data Memory

The second variation, illustrated in Figure 4-5, is to use multi-ported data memory as utilised in the Fujitsu MB86232. Although often perceived as a costly solution, multi-operand instructions do not require operands to be separated into multiple memory banks.

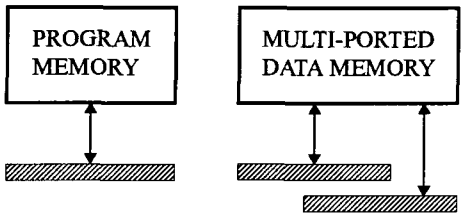

Figure 4-5: Second modification — multi-ported data memory.

4.3.1.3 Third Modification — Instruction Cache

In Figure 4-6, an instruction cache supplements program/data memory. When cached instructions are executed, no instruction-fetch is required and so a memory access is freed for a data fetch (e.g. Hitachi 61810). Such low-overhead looping is a common feature today.

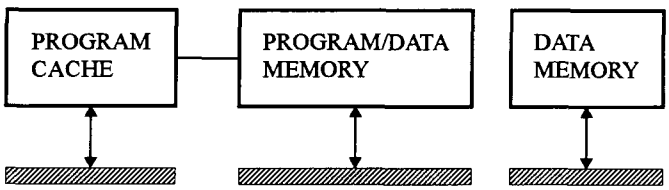

Figure 4-6: Third modification — instruction cache.

4.3.1.4 Fourth Modification — Third Memory Bank

An obvious next step is to expand the instruction cache until it becomes a third memory bank, as Figure 4-7. This approach is used in the Motorola DSP56001 and DSP96002 to provide simultaneous on-chip fetches of an instruction and two operands.[1] In contrast, the TI TMS320C30/C40 use an instruction cache to conserve the external memory cycle [TI, 1987].[2]

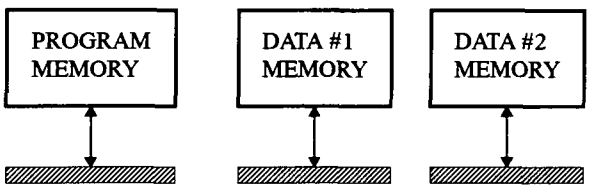

Figure 4-7: Fourth modification — third memory bank.

1. On-chip RAM can be accessed *twice* in each cycle, ensuring that DMA does not interfere with execution.
2. In these TI DSPs, the second access can be used by some multi-operand instructions.

4.3.1.5 Fifth Modification — Several Memory Banks

Several DSPs achieve high memory bandwidth by accessing parallel memories twice in each instruction cycle. The Hitachi DSPi, on the other hand, uses six memory banks as Figure 4-8. Instructions with three memory operands (e.g. 2 reads, 1 write) use three of the four data memories and program memory. Simultaneous input/output (I/O) can access the fourth bank.

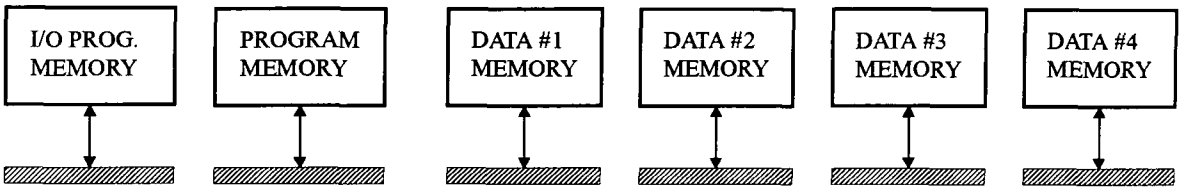

Figure 4-8: Fifth modification — several memory banks.

4.3.2 Demand Ratio

Demand ratio is defined as the total number of memory cycles per instruction cycle, summarised in Table 4-2. Devices with the smallest ratio of two require instruction caches in order to execute instructions with two operands in a single cycle. Circular buffer addressing modes are required in order to implement FIR filters in one instruction cycle per tap.

DSPs with a demand ratio of three do not require an instruction cache, but do require circular buffer addressing for delay lines. When the value is four or higher neither circular buffer addressing nor instruction caches are needed. Note, however, that the demand ratios given here are for internal memory: values are usually much lower when memory is off-chip.

4.3.3 Internal and External Memory

As DSPs are intended for stand-alone use, most have memory on-chip: there are more subtle reasons, however. Multiple independent memory banks and multi-ported memories are difficult to implement off-chip as the pin count is excessive. Furthermore, the speed penalty of going off-chip increases the system cost by requiring faster memories.

Motorola's DSP96001 has a 32-bit word-length and three memory banks, each with a 32-bit[1] address, requiring 192 pins for bussing alone if implemented externally. Multiplexing

1. For a 4G word address space per memory bank, the largest of all the DSPs

Part No.	No. of Banks	Demand Ratio	Cache	Internal Memory[a]	External Memory
ADSP-21020	0	(2)	32 W	—	4 GW P 16 MW P/D
DSP32	2	2	no	2 × 512 W D RAM 512 W P ROM	12 KW
DSP32C/3210	3	4	no	2 × 512 W D RAM 512 W P RAM or 1 KW P ROM	4 MW
DSP56001	3	3	no	2 × 256 W D RAM 2 × 256 W D ROM 512 W P RAM	3 × 64 KW
DSP96002	3	4	no	2 × 512 W D RAM 2 × 512 W D ROM 512 W P RAM	4 GW P 2 × 4 GW D
TMS320C30/C40	3	4	64 W	2 × 1 KW P/D RAM 4 KW P/D ROM	16 MW

Table 4-2: Summary of memory systems for preferred DSPs.[b]

a. P and D represent program and data memory, respectively.
b. W represents 'words', where word-length is dependent on individual DSP numeric formats.

is not an option due to fast cycle times and costly external demultiplexing. Only the more expensive 200-pin version, the DSP96002, carries two physical busses off-chip.

4.3.4 Addressing Modes

While parallel memory banks can increase memory bandwidth, one fundamental problem remains. Instructions must specify as many as three simultaneous memory accesses.

4.3.4.1 Register-Indirect Addressing

In register-indirect addressing, a set of *address* registers permits the specification of several addresses in a one-word instruction. Loaded with the address of one word in a data structure (e.g. the first sample in a delay line), all instructions accessing the delay line then reference this register. Since the register bank is small, few bits are required to specify each register.

4.3.4.2 Modulo-Mode Addressing

An efficient alternative to shifting data in memory is the circular buffer (Figure 4-9): $m + 1$ contiguous memory locations are allocated beginning at l. Using *modulo-mode* addressing, a register reverts to m when it is incremented beyond $l + m$. Motorola DSPs have 24 registers arranged in triplets: the address is held in *rx*, the increment in *nx*, and the buffer length in *mx*.

Two address ALUs perform post-auto-increment and modulo arithmetic, supporting simultaneous modulo addressing for data and coefficients. In order to simplify address ALUs, circular buffers must begin on power-of-two boundaries. If $(m + 1) \leq 2^k$ but $(m + 1) > 2^{k-1}$, then the low order k-bits of l must be zero. Consequently, l must fall on a multiple of 2^k.

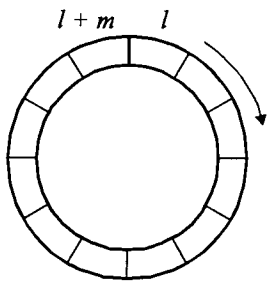

Figure 4-9: Visualisation of a circular buffer of length m.

4.3.4.3 Other Addressing Modes

Register-direct and modulo addressing modes cannot be used exclusively in ASP. Most DSPs have some form of *immediate* addressing (operand is part of instruction) and *direct* addressing (address is part of instruction) as shown in Table 4-3. Other modes include *indexed* addressing for non-sequential array access, and *bit-reversed* for re-ordering FFT output vectors.

Part No.	Immediate	Direct	Indirect			Bit Reverse
			No. of registers	Modulo	Index	
ADSP-21020	yes	yes	16	yes	yes	yes
DSP32/32C/3210	int. only	int. only	15	no	no	no/yes/yes
DSP56001	yes	yes	8	yes	yes	yes
DSP96002	yes	yes	8	yes	yes	yes
TMS320C30/C40	yes	64 KW	8	yes	yes	yes

Table 4-3: Summary of addressing modes for preferred DSPs.

4.4 Pipelining

To keep the instruction cycle times of programmable DSPs short, operations are *pipelining*. DSP internal pipelines and the different programming styles required are examined below.

4.4.1 Interlocking

As shown in Figure 4-10, one model of DSP pipelining distinguishes between instruction fetch, decode, operand fetch, and execution stages.[1] Instruction fetch occurs at the same time as the previous instruction decode and the operand fetch for the instruction before. Overlapping instructions gives the impression that each one finishes before the next begins.

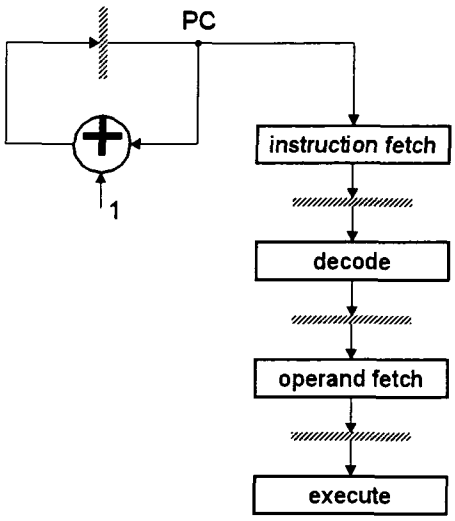

Figure 4-10: Model of the pipelining within a programmable DSP.[a]

a. Cross-hatched boxes indicate *latches*, which latch signals once per instruction cycle. PC represents the program counter, controlling which address is fetched from program memory.

With this scheme, contention can occur for internal resources: e.g. fetching two operands from the same memory bank. In *interlocking* control logic delays ALU execution to resolve the contention. Of the DSPs discussed here, TI processors make the most use of interlocking [TI, 1987]. Accordingly, the internal timing of these devices is rather elaborate.

1. To which the TI TMS320C30 and TMS320C40 conform well.

4.4.2 Time-Stationary Coding

In time-stationary coding, an instruction specifies the operations that occur simultaneously. Several DSPs, including the Motorola DSP56001, are built around the outline reservation table of Table 4-4 [Lee, 1989]. Each instruction specifies simultaneous operand fetch and execute operations. In essence, the program model is one of *parallelism* rather than pipelining.

Hardware Resource	Instruction Cycle			
	n	$n+1$	$n+2$	$n+3$
Program memory				
X-data memory				
Y-data memory				
ALU				

Table 4-4: Outline of reservation table for DSPs that use time-stationary coding.

Consider the DSP56001 MAC instruction of Figure 4-11: registers $x0$ and $y0$ were loaded from memory in a *previous* instruction; the product of $x0$ and $y0$ is added to the a accumulator in the *same* cycle. Unlike any other device, the DSP56001 has an adder integrated into the multiplier, so that the MAC actually occurs together in the same hardware.

```
mac x0,y0,a     x:(r0)+,x0     y:(r4)-,y0
```

Figure 4-11: Example MAC instruction, in Motorola DSP56001 assembly code.

4.4.3 Data-Stationary Coding

In data-stationary coding, a single instruction specifies all operations performed on a set of operands. This approach is more natural since the instruction specifies what happens to given data. The main upshot is that the results of an instruction may not be immediately available. AT&T floating-point DSPs are the most dramatic examples of data-stationary coding.

Fast interrupts are more difficult with data-stationary than with time-stationary coding. For example, the AT&T floating-point DSPs have a three-cycle quick interrupt. To accomplish this, a second set of pipeline registers 'shadow' the main set, storing the processor state when an interrupt occurs.[1] Nested interrupts are, of course, not possible [AT&T, 1991].

4.4.4 Comparison of Coding Methods

Time-stationary coding has a number of advantages. Firstly, the timing of a program is clearer. Secondly, interrupts can be much more efficient, since the programmer has explicit control over the pipeline. The Motorola parts have a very fast interrupt, taking exactly two instruction cycles to pull audio samples from an ADC or AES/EBU peripheral into a circular buffer.

Data-stationary is no less efficient than time-stationary coding. However, to specify a multiply, accumulate and store with time-stationary coding requires several instructions. Other operations proceed in parallel specified by the unused operand fields. In data-stationary coding other operations proceed in parallel due to neighbouring instructions.

4.4.5 Branching

One difficulty with pipelining is branching: several problems conspire to make it difficult to achieve efficient branching. First, there may not be sufficient time between instruction fetches to decode a branch instruction before the next fetch. Secondly, in conditional branching, the next instruction fetch cannot occur before the ALU condition codes can be tested.

For reasons of efficiency in ASP, it is necessary to use low-overhead looping rather than branch instructions. In Motorola DSPs, any number of instructions may be included inside a loop and loops are interruptible. Another technique used to avoid the inefficiencies associated with conditional branches are *conditional instructions* [Motorola, 1989a].

4.5 Software Development Tools

DSP software development tools have evolved from simple assemblers to the integrated software tools necessary to cope with ever-increasing DSP complexity.

4.5.1 Optimising C Compilers

One of the main problems with programmable DSPs still persists: code must be tuned by hand to meet hard real-time constraints. Optimising parallel C compilers exist for the latest generation of floating-point DSPs. Floating-point arithmetic in conjunction with this compiler

1. Shadowing is also used by Analog Devices in the ADSP-21020 to achieve single-cycle context switching.

technology removes the need for much programmer-intensive scaling. Regardless of efficiency, these compilers will inevitably be used for large-scale digital ASP applications.

4.5.2 Block-Diagram Environments

It is doubtful that good C compilers alone are a complete solution. While not entirely new in the field of DSP, various very high-level 'iconic' development environments have emerged.

4.5.2.1 Gabriel

At the University of Berkeley, Lee et al. have developed a block-diagram environment called *Gabriel*. Real-time constraints, sample rates, recurrences, conditionals, and iteration are managed systematically. Also capable of generating code for multiple processors, *Gabriel's* basic facilities have been demonstrated for both the DSP56001 and DSP32 [Lee, 1987a].

4.5.2.2 Comdisco Signal Processing Worksystem

Comdisco's Signal Processing Worksystem (SPW) allows ASP algorithms to be quickly implemented with minimal software coding. The designer 'wires-up' DSP algorithms chosen from a library of processing blocks, and then compiles and downloads the resultant assembly code. Processing blocks can be manually allocated for multi-DSP platforms.

4.5.2.3 IRCAM Signal Processing Workstation

In addition to developing the dual-i860 ISPW, the IRCAM research organisation in Paris has also developed custom applications for this 93.5 MIPS NeXTcube add-in. *Animal* provides a real-time object-oriented programming environment, while the graphical programming language *MAX* is a suited to distributed real-time ASP applications such as musical synthesis.

4.5.3 Software Simulators

An essential capability almost totally lacking in software simulators is the simulation and debugging of multiple-DSP systems. Motorola is a notable exception: simulator sub-routines can be called from user-written code. Each call emulates the state change of a processor in one clock cycle. A DMC designer planning to use several DSPx600x can write C code that emulates the interconnection of DSPs, shared memory, and other peripheral hardware.

4.5.4 Real-Time Operating Systems

SPOX-OS is a real-time C-based operating system for programmable DSPs providing multitasking, memory-management and device-independent I/O. SPOX-MATH, a library of optimised assembly-language routines, minimises the effort required to code a wide-range of DSP applications. While the inevitable consequence of using SPOX is increased overhead, a designer can develop an entire application in C and then hand-optimise time-critical sections.

4.5.5 West vs. East

Japanese manufacturers specifically target their DSP devices at a few large manufacturers with large-volume sales. While their designs provide good support for application-specific routines, development tools suffer as a result. In contrast, North American manufacturers target their products at the signal processing community in general. Consequently, these manufacturers must provide high-level language compilers and integrated development tools.

4.6 Processing Technology and DMCs

DMCs must be implemented using the technology of the day. In the 1970s, only discrete logic or bit-slice technology could be tailored to the processing demands of professional ASP.

4.6.1 Parallel-Bus DSPs

By the late 1970s, many manufacturers had produced 16-bit fixed-point DSPs. Although used for ASP, audio performance was simply not adequate for professional DMCs [Eastty, 1986].

4.6.1.1 Fixed-Point

In 1987, Motorola introduced the first 24-bit fixed-point DSP shown in Figure 5-11. The DSP56001 offers a 24 × 24-bit multiplier/adder, 2 × 56-bit accumulators and a dual Harvard architecture internally. Problems with operand scaling and limit-cycles were greatly reduced. Some products based on bit-slice parts, such as the SSL *01*, were superseded overnight.

The SCI port interfaces to serial protocols such as MIDI, and the SSI to data-domain and digital audio protocol converters. An 8-bit wide HOST port is also provided. Time-

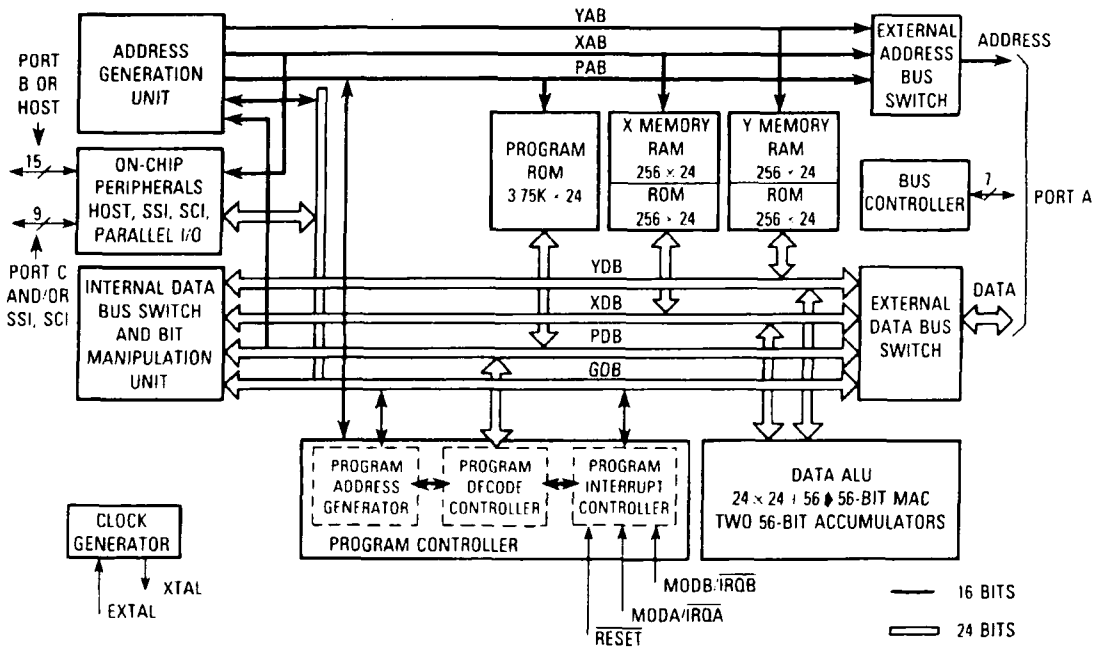

Figure 4-12: Block diagram of the Motorola DSP56001 (courtesy [Motorola, 1989a]).

stationary coding coupled with operational concurrency results in efficient coding of stock ASP algorithms. This DSP has proved popular with professional audio manufacturers.[1]

4.6.1.2 Floating-Point

Some console designers have opted for IEEE-format floating-point DSPs. From Figure 4-2, 32-bit floating-point arithmetic does not offer increased resolution over 24-bit fixed-point but does remove the need for scaling. Limit cycles are less problematic through the much increased dynamic range of both signal and coefficients alike. However, these advantages are gained at the expense of increased cost and instruction cycle times (see Appendix A).

4.6.2 Serial-Bus DSPs

Sony developed the CXD-xxxx serial-bus DSPs for professional audio use. Dedicated devices, such as the CXD-8307 8:1 mix-bus, are complemented by programmable ASP parts.

4.6.2.1 Sony CXD-1160

The CXD-1160 has two stereo inputs, one stereo output and contains a multiplier, ALU, on-board control program and serial host port. Processing speed and instruction cycle run at 128-

1. Products include the Peavey *DPM* range of synthesizers and AMS/Neve *Logic* consoles.

times the word clock. This first-generation part can be programmed to perform a wide variety of ASP tasks including: phase inversion, high and low pass filtering, or 3-bands of sweep EQ.

4.6.2.2 Sony CXD-2705

The CXD-2705 is a enhanced version of the CXD-1160. A faster processing speed (256-times word-clock), 52-bit internal word-length and two additional stereo outputs support fader level, muting, output routing and stereo 4:2 sub-mix functions. Addition hardware is also integrated into the multiplier unit to realise a true MAC instruction [Hingley, 1992].

4.6.2.3 Comparison with Parallel-Bus DSPs

Serial-bus devices offer flexible processing and AES/EBU-transparent audio quality at low cost. A DMC signal path can be created by simply cascading devices.[1] However, chip count rises as a product of the number of inputs, outputs and processing functions. Furthermore, the cumulative pipeline delay inherent in serial-bus DSPs becomes a serious design problem. Conventional parallel-bus processors are more suitable for large-scale DMC implementations.

4.6.3 Application-Specific Integrated Circuits

Unlike DSPs, which are fabricated in runs of 100s of thousands at a time, application-specific integrated circuits (ASICs) are generally made in batches of 10-100 thousand.

4.6.3.1 Overview

Only the top layers of metal interconnect define circuit function in semi-custom ASICs. In full-custom ASICs, traditionally the exclusive preserve of IC manufacturers, all layers are defined. Silicon compilers accept an ASIC specification as a CAD logic schematic or VHDL description, which is transformed with minimal intervention into a silicon layout. As a result, design engineers with little knowledge of IC manufacture can produce DMC-specific VLSI.

4.6.3.2 Design Methodologies

Static-synchronous is the most straightforward, using edge-sensitive, single-phase clocking schemes. Multi-phase static designs use multi-phase level-sensitive clocking, an approach ideally suited to audio ASIC production. In contrast, multi-phase dynamic designs require a

1. Sony's stereo-format 8:2 DMX-E3000 is based on the CXD serial-bus chip set discussed here.

clock to maintain their state. ASIC vendors rarely have libraries to support this advanced methodology and few, if any, will accept responsibility for such designs [Naish, 1988].

4.6.3.3 Advantages

ASICs release DMC designers from the constraints imposed by DSP VLSI manufacturers. Many analogue *and* digital functions can be implemented on one wafer, with the number format optimised throughout. Testibility features, such as scan paths, can be built-in as an integral part of the design [Trontelj, 1989]. Neve have implemented such a chip-set, using TI libraries, which are now employed in the recently-released *Capricorn* DMC.

4.6.4 Vector Processors

While the Intel i860 was originally developed for graphics processing, its high-speed vector capabilities are being exploited in a wide range of DSP-intensive applications.

4.6.4.1 Overview

The Intel i860, Figure 4-13, brings super-computer design concepts to floating-point μPs. Dual-operation instructions use both FPUs increasing the efficiency of IIR filters and FFTs. In the dual-instruction mode, a RISC integer instruction is executed simultaneously. On-chip, the i860 has both instruction and data caches, with a TLB. Direct MIPS comparisons with DSPs, however, can be misleading: other performance factors are just as critical.

4.6.4.2 Peak Performance

Intel quotes 100 MFLOPS (peak) for the 50 MHz part: 2-3 times that of most DSP devices [Intel, 1990]. As the i860 relies heavily on pipelining, this rate can only be achieved by fully utilising dual-operation instructions. In non-vector ASP, cache-misses and pipeline-flushes reduce actual performance to a fraction of peak. Most DSPs, by contrast, do not implement pipelining within the ALU, simplifying programming, particularly hand-coding.

4.6.4.3 Data Movement and I/O

To keep an ALU continuously supplied with new data, many DSPs support two EMIs with separate address ALUs. In contrast, the i860 provides only one external bus and can only support concurrent accesses when operating out of on-chip cache. Most DSPs provide one or

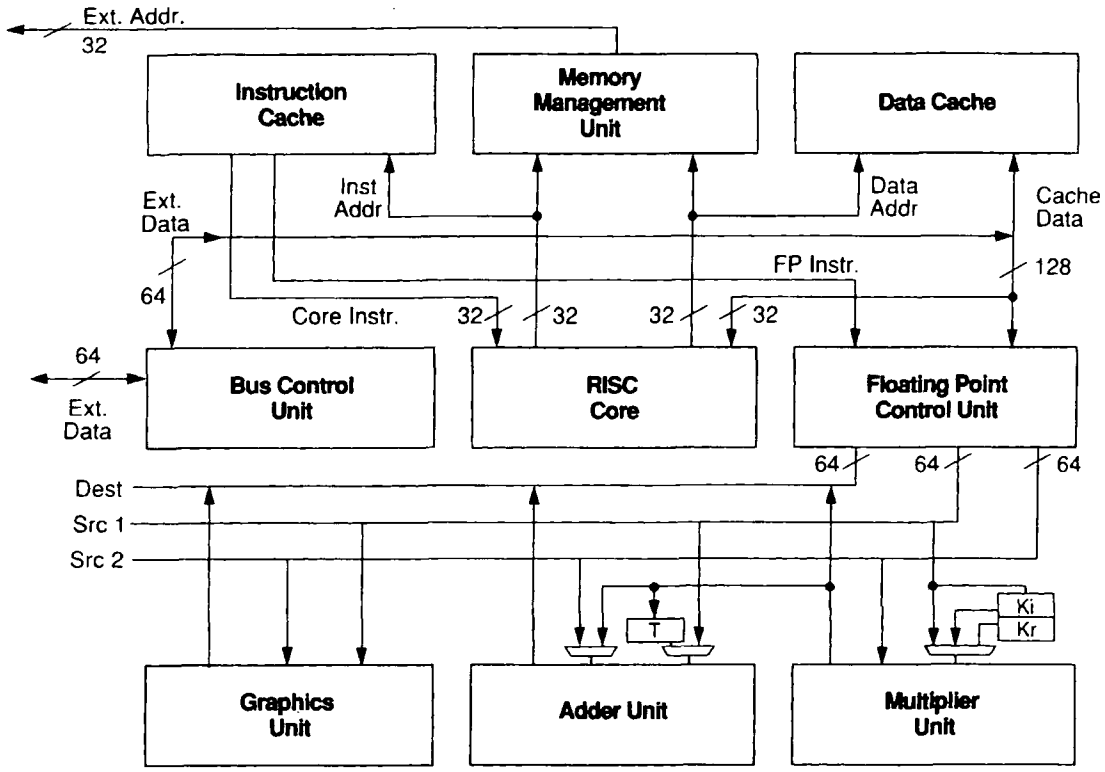

Figure 4-13: Block diagram of the Intel i860 (courtesy [Intel, 1990b]).

more I/O port, each with a independent DMA controller. The i860 provides no I/O ports and no DMA controller, making it impossible to access and process data concurrently.

4.6.4.4 Context-Switching and Determinism

Large register files and multi-stage pipelines limit context-switch response. The i860 requires 81 instruction cycles to save registers before each switch: DSPs can take as few as 18. Many DSPs enable fast interrupts to be serviced in 2 instructions: the i860 requires 24. Real-time performance of vector processors depends on cache hit-rate. While many DSPs include cache memory, portions can be locked independently to ensure a 100% cache hit-rate.

4.6.4.5 Application to Real-Time ASP

One way to alleviate I/O limitations is to use a DSP front-end processor to handle real-time I/O, and i860s as back-end compute servers.[1] Vector processors may then be successfully employed in DSP requiring limited real-time I/O — e.g. synthesis. For I/O-intensive

1. Ariel's ISPW integrates two i860s with a Motorola DSP56001 on one NeXTcube expansion card.

applications such as digital mixing, however, programmable DSPs offer many benefits. The i860's single EMI and limited I/O also make it cumbersome to use multiple units together.

4.6.5 Parallel DSP Devices

Manufacturers continue to increase DSP performance as silicon processing technologies allow. Historically, very little support has been provided for parallel computation.

4.6.5.1 Inmos T801 Transputer

Specifically designed for parallelism, the Inmos transputer can perform DSP operations especially when non-time critical.[1] As shown in Figure 5-12, a 32-bit time-slicing CPU is integrated with 4K bytes of RAM and four serial communication links. CPU performance degradation is ≈5% when all four links are saturated. As the T801 has a non-multiplexed EMI, memory access is 1.5-times that of an equivalent T800 [Inmos, 1989].

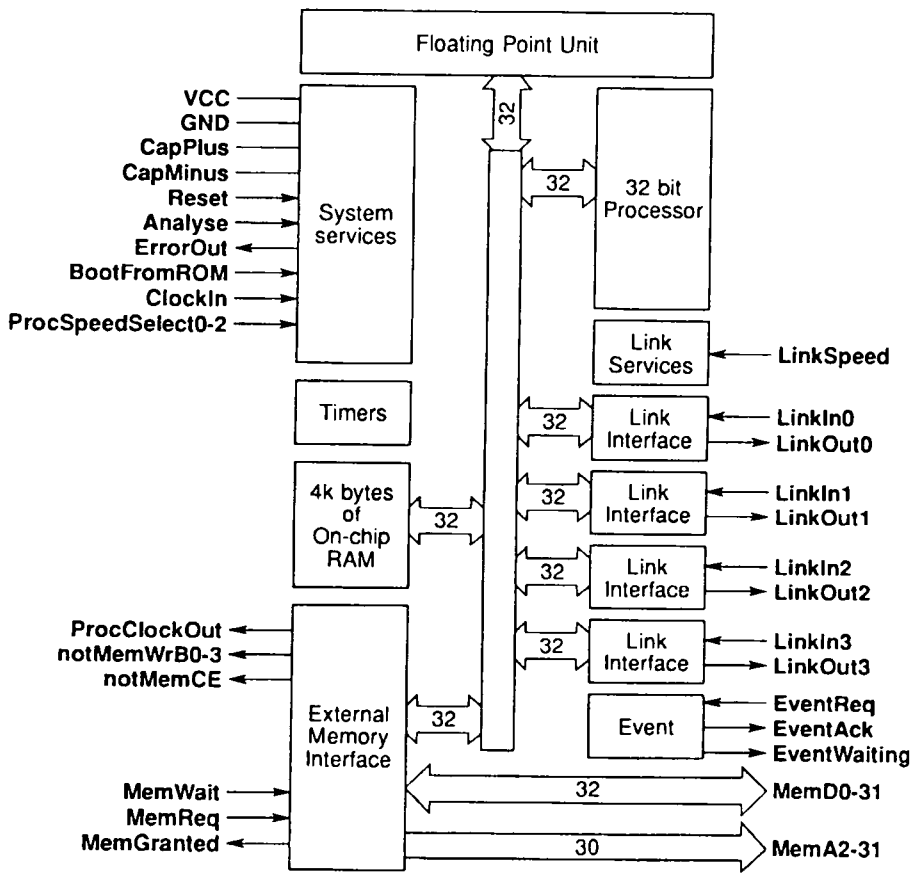

Figure 4-14: Block diagram of the Inmos T801 (courtesy [Inmos, 1989]).

1. With no dedicated hardware, the T8xx has a multiply time of 13 instruction cycles, or 433 ns at 30 MHz!

4.6.5.2 Texas Instruments TMS320C40

With conventional DSPs inter-processor communications (IPC) causes device I/O to quickly saturate [Snell, 1989]. DSP manufacturers addressed this problem by providing two EMIs. The DSP96002 and TMS320C30 extend the dual Harvard architecture beyond the silicon. Even with support for dual-channel DMA, IPC still consumes considerable I/O bandwidth.

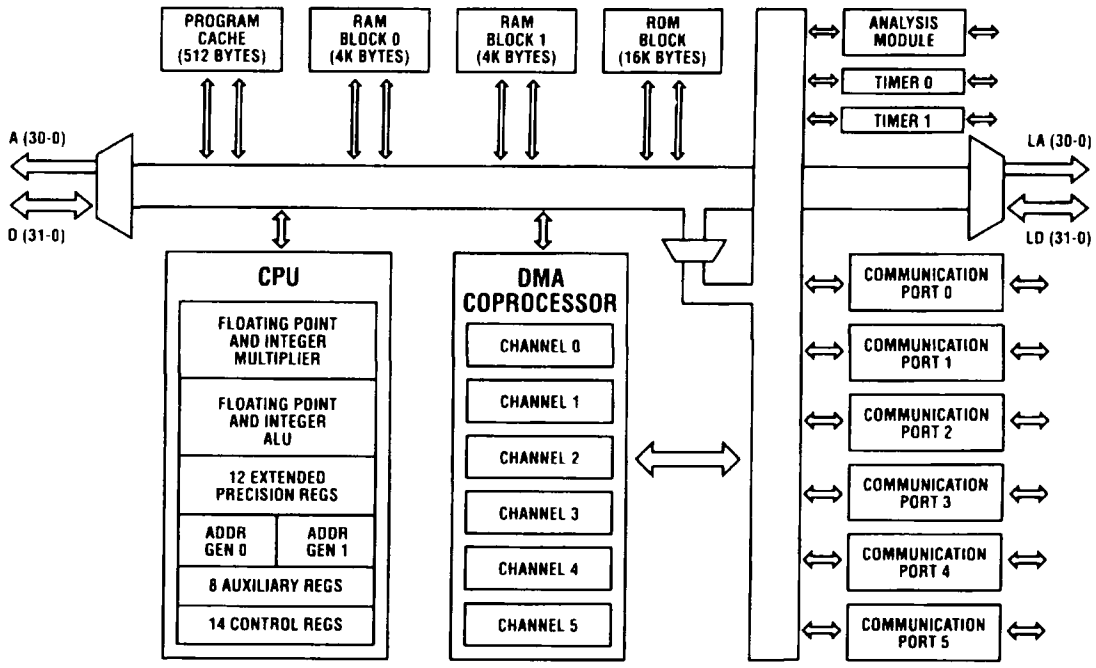

Figure 4-15: Block diagram of the TI TMS320C40 (courtesy [TI, 1991]).

TI's TMS320C40 (Figure 4-15) goes further: six bidirectional communication ports support glueless IPC. Each port has a dedicated DMA permitting 20 Mbytes/s of concurrent I/O. Without these ports, the corresponding IPC would have to be squeezed onto the EMIs. An on-chip analysis module provides efficient parallel debugging facilities [TI, 1991].

4.6.5.3 Inmos T9000

The first member of a new generation of transputers, the superscalar 50 MHz T9000 integrates a 200 MIPS 32-bit CPU with a 25 MFLOPS 64-bit FPU, 16 Kbytes of cache/conventional memory, and four communications links [Inmos, 1993]. Inmos provides a powerful development toolset, several optimised parallel compiler and real-time kernels.

Four bidirectional serial links and a dedicated virtual channel processor (VCP) support 80 Mbytes/s. The hardware kernel provides an instruction grouper, multiple interrupts and

sub-microsecond context switching. Up to 8 Mbytes of DRAM can be interfaced with zero external logic. T8xx-T9xxx networks can be built using the C100 system protocol convertor.

4.6.5.4 Texas Instruments TMS320C80

With a unique MIMD micro-architecture supporting 200 MIPS, the TI TMS320C80, or multimedia video processor (MVP), brings an unprecedented level of performance to programmable DSP. Four 64-bit DSPs are integrated on a single IC together with a 32-bit RISC CPU, a DMA controller, and 50 Kbytes of static RAM [Andrews, 1994].

With 400 Mbytes/s IPC bandwidth, the MVP can support real-time applications which currently require large M-P configurations. Although targeted at video processing, the MVP is ideally suited to other high-end applications including large-scale digital audio mixing. TI expects full production to begin by mid 1995, at a unit cost of $250 in 10,000-unit quantities.

4.6.5.5 Philips Semiconductor TriMedia

With patents including the CD, Philips has an impressive pedigree in audio innovation. Now this company has released the TriMedia, a 100 MHz DSP targeted at multimedia [Hars, 1995]. This device employs a VLIW architecture to execute five operations in a single clock cycle. Each operation can contain 11 RISC instructions, making for some impressive benchmarks.

Unlike RISC where the CPU reveals parallelism, VLIW relies heavily on compiler technology. By incorporating serial audio DMA, Philips envisage that the TriMedia will find use in applications requiring multiple DSPs, i.e. audio mixing. With sampling beginning in early 1996, full production of the TriMedia is scheduled for the middle of the year.

4.7 Conclusions

Programmable DSPs come in two basic types, fixed- and floating-point. Floating-point DSPs are more expensive and generally at least 50% slower than fixed-point devices of comparable technology. The main advantage of floating-point devices is that they free the DMC designer from concerns about scaling: the chief disadvantages are reduced speed and increased cost.

The dominant trend in DSPs is towards complexity: each new device includes μP facilities lacking in the previous product. Conversely, a market still exists for simple and fast

DSPs. High-speed low-power fixed-point DSPs, such as the AT&T DSP16A and TI TMS320C50, are targeted specifically at mobile communications applications.

An interesting development is the emergence of DSP cores: programmable DSPs surrounded by customer-specified circuitry. For example, Motorola has released the DSP56116, a 16-bit core version of the original 24-bit DSP56001. Devices such as these, and the recent TASP family from Texas Instruments, provide strong competition for ASICs.

A fourth approach to pipelining, not yet implemented in any commercial DSP, is *pipeline interleaving*. Instead of a single in-line instruction stream, instructions from several independent streams are interleaved. Pipeline interleaving transforms a single DSP with data-stationary code into multiple processor slices sharing the same hardware.

With the newest versions of G-P μPs already incorporating DSP-like dual-bus architectures, the obvious next step is the integration of an array multiplier and a barrel shifter. New VLSI technologies such as gallium arsenide (GaAs) suggest that another order-of-magnitude speed improvement is possible through processing technology alone.

Concepts borrowed from parallel computation have moved across into the DSP arena. Since VLSI devices are subject to stringent packaging restrictions, byte-wide communication links providing highly efficient parallel processing capabilities. Devices, such as the TI TMS320C40, integrate multiple IPC ports directly on-chip with high-speed DSP cores.

While recent advances support M-Ps, few devices have specific architectural features for DMC systems on-chip. For example, the time-domain effects of Section 2.1.3 require dedicated delay processing resources. Although many DSPs include sine-wave tables, none support the additional modulation waveforms necessary for chorus and pitch-shift algorithms.

In order to implement the mix function described in Section 3.2.1, many instruction cycles are consumed by interpolating real-time control parameters in order to remove zipper noise. Incidentally, built-in data-domain converters are not desirable as manufacturers prefer to choose parts from a variety of sources or implement proprietry solutions.

As device technology advances in to the multimedia era, the distinctions between DSP devices and G-P processors are being swiftly eroded. Novel devices like the TMS320C80 are

emerging with multiple DSP cores integrated on a single die. Although currently expensive, their cost compares favourably with the original pricing structure of the Motorola DSP56001.

4.8 References

[Andrews, 1994] Andrews D: "Digital Signal Processing: The Engines to Make Multimedia Mainstream", *Byte*, pp. 22-24, May 1994.

[AT&T, 1991] AT&T: *WE DSP32C Digital Signal Processor Information Manual*, AT&T Document Management Organisation, MN89-010DMOS, January 1990.

[Eastty, 1986] Eastty P: "Digital Audio Processing on a Grand Scale", *81st Audio Engineering Society Convention*, Los Angeles, California, USA, September 1986.

[Fuccio, 1988] Fuccio M L (et al.): "The DSP32C: AT&T's Second-Generation Floating-Point Digital Signal Processor", *IEEE Micro*, pp. 30-48, December 1988.

[Hars, 1995] Hars A: "Hot Chips, Cool Media", *Byte*, pp. 5-10, August 1995.

[Hingley, 1992] Hingley, A: "Cost Effective DSP for Audio Mixing", *Proceedings of the AES UK DSP Conference*, London, UK, pp. 226-234, September 1992.

[Inmos, 1989] Inmos: *The Transputer Databook*, 2nd edition, Inmos Ltd., Document No. 72 TRN 203 01, 1989.

[Inmos, 1993] Inmos: *The Inmos T9000 Transputer: Product Overview*, Inmos Ltd., March 1993.

[Intel, 1990] Intel: *i860 64-bit Microprocessor Performance Brief*, Intel Corp., Order No. 240588-004, 1990.

[Lee, 1987a] Lee E A and Messerschmitt D G: "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", *IEEE Transactions on Computers*, Vol. C-36, No. 1, pp. 24-35, January 1987.

[Lee, 1989] Lee E A: "Programmable DSP Architectures: Part II", *IEEE ASSP Magazine*, pp. 4-14, January 1989.

[McNally, 1982] McNally G W: *Digital Audio: COPAS-2, A Modular Digital Audio Signal Processor for Use in a Mixing Desk*, BBC Research Department, Report No. RD 1982/13, 1982.

[Motorola, 1989a] Motorola: *DSP56000/DSP56001 Digital Signal Processor User's Manual*, Revision 2.0, Motorola Inc., 1989.

[Motorola, 1989b] Motorola: *DSP96002 IEEE Floating-Point Dual-Port Processor User's Manual*, Motorola Inc., 1989.

[Naish, 1988] Naish P and Bishop P: *Designing ASICS*, Halstead Press, 1988, ISBN 0-470-21375-2.

[Nishitani, 1986] Nishitani T: *Signal Processor Design Methodology*, North-Holland, 1986, ISBN 0-13-14625-03.

[Snell, 1989] Snell J M: "Multiprocessor DSP Architectures and Implications for Software", *Proceedings of the 7th Audio Engineering Society International Conference*, pp. 318-336, Toronto, Canada, 1989.

[TI, 1987] TI: *TMS320C30 User's Guide*, Rev. B, Texas Instruments Inc., Document No. 2562587-8437, 1987.

[TI, 1991] TI: *TMS320C40 User's Guide*, Rev. A, Texas Instruments Inc., Document No. 2564090-9721, 1991.

[Trontelj, 1989] Trontelj J, Trontelj L and Shenton G: *Analog-Digital ASIC Design*, McGraw-Hill, 1989, ISBN 0-0770-7300-2.

Chapter 5

Architectural Issues

Due to fundamental physical limits on processor technology, any DMC design demands multi-processor (M-P) implementation. Some form of inter-processor communications (IPC) network is necessary to exchange audio samples and mix-function control parameters.

5.1 Terminology

In this section the terminology of parallel processing is developed in the context of supporting computationally intensive real-time ASP on multi-DSP architectures.

5.1.1 Classification of Parallel Architectures

Four general classes of parallel-machine architectures are distinguished in the most widely known classification, given by Flynn [Flynn, 1966]. His taxonomy categorises computer systems according to the relationship between instruction and data streams.

5.1.1.1 Single Instruction, Single Data Stream (SISD)

In SISD machines there is a single instruction stream and, hence, a single processing unit. Each instruction results in one arithmetic operation on one set of data at a time. This category includes conventional sequential uni-processors such as the Motorola MC68040 μP.

5.1.1.2 Single Instruction, Multiple Data Stream (SIMD)

SIMD machines have a single master unit, which makes decisions on the flow of control, and a large number of slave processors. As they are designed to exploit data parallelism, SIMD M-Ps retain the single instruction stream but support instructions which execute on many data streams. Process parallelism cannot be utilised here as it demands multiple control threads.[1]

1. In *autonomous* SIMD systems, individual processors can be masked during certain operations.

As a result, SIMD machines perform poorly in applications which contain substantial proportions of sequential code. The most significant machines in this class are array processors such as Active Memory Technology's DAP, and the CM-1 and CM-2 from Thinking Machines which connect 64K 1-bit PEs in pseudo-hypercube topologies.

5.1.1.3 Multiple Instruction, Single Data Stream (MISD)

Though Flynn's taxonomy reveals that MISD architectures might exist, there are currently no examples of machines which apply multiple instructions at once to a single data stream.

5.1.1.4 Multiple Instruction, Multiple Data Stream (MIMD)

The multiple instruction streams of MIMD machines lead to a number of devices each with one or more data streams. Processors are either synchronous, performing successive instructions in lock-step, or achieve synchronisation via messages-passing. Note that the interaction of instruction streams can lead to dead-lock and other such software ailments.

Two broad sub-categories of MIMD exist: homogeneous machines constructed from identical PEs, and heterogeneous designs. A large number of commercial M-Ps adhere to the MIMD model, including the Ametek 2010 2-D mesh and Intel iPSC hypercubes. MIMDs using programmable DSP devices are particularly attractive for DMC implementations.

5.1.2 Processing Budget

Each channel in a SSL *G Series* console (see Section 2.3.1) has 78 switch and 31 continuously variable functions, or over 10,000 functions on a 96-channel frame. In an ideal M-P with n processors, MIPS is increased by a factor of $n/(n-1)$ for each additional PE — linear *speed-up*. If each additional DSP contributes only x-times current performance, then:

$$S(1) = 1$$
$$S(n) = \left(\frac{n-1+x}{n-1}\right)S(n-1), \quad 0 \le x \le 1 \qquad \text{Eq. (5-1)}$$

where $S(n)$ is the speed-up of n processors. Such diminishing returns, shown in Figure 5-1, make successive DSP less and less cost effective. From Section 3.2.1, the core ASP engine of a 96:24 DMC requires 267 MIPS. Allowing 30 MIPS for additional mix circuits and assuming $x = 0.75$, Equation (5-1) implies 80 DSPs with a nominal 10 MIPS.

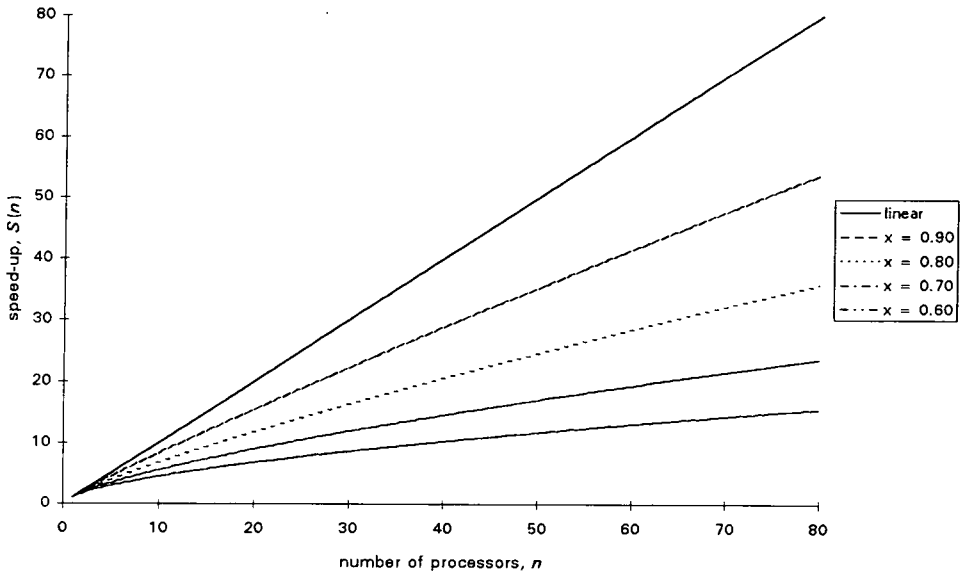

Figure 5-1: Example speed-up curves for M-P DMC implementations.

5.1.3 Implementing Parallelism

There are two fundamental ways in which DSPs can be composed to create M-P architectures. Perhaps the simplest way to introduce parallelism is to replicate a component *n* times, as shown in Figure 5-2. To exploit this form of parallelism, digital audio streams must be separated in space. For this reason this approach is known as *spatial* or *replicated* parallelism.

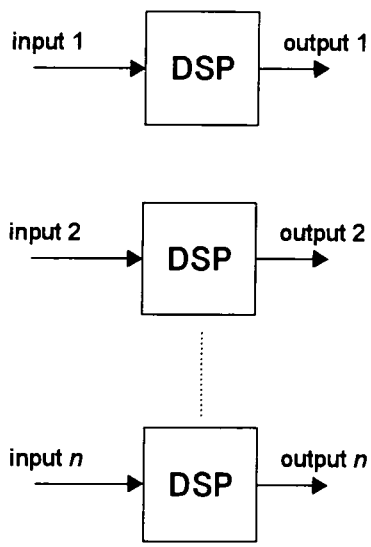

Figure 5-2: Exploiting spatial parallelism.

Alternatively, ASP can be partitioned into a number of steps, as shown in Figure 5-3. Each step is applied sequentially to restore the original ASP algorithm. Accordingly, this form of parallelism is also known as *temporal* parallelism. Although the distinction between these forms of parallelism is clear, an increasing number of M-Ps employ both techniques.

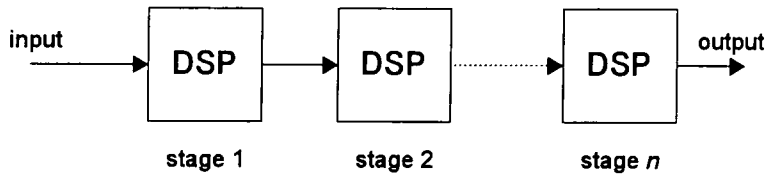

Figure 5-3: Exploiting temporal parallelism.

5.1.4 Processor Pipelining

Pipelining is an implementation of temporal parallelism whereby multiple tasks are overlapped in time. In a processor pipeline the ASP required to complete an application is broken into smaller tasks, which are then connected one to the next to form a pipe. Samples enter at one end, are processed through the pipe, and exit at the other end, as Figure 5-4.

In DMCs response is perceived as immediate if the system operates with an accumulated processing lag of less than 1 ms. To meet the frequency response requirements of Section 3.1.1.3, each DSP must capture input samples every 20.8 μs. In a DMC, this corresponds to a maximum processor pipeline of 48 stages between inputs and outputs.[1]

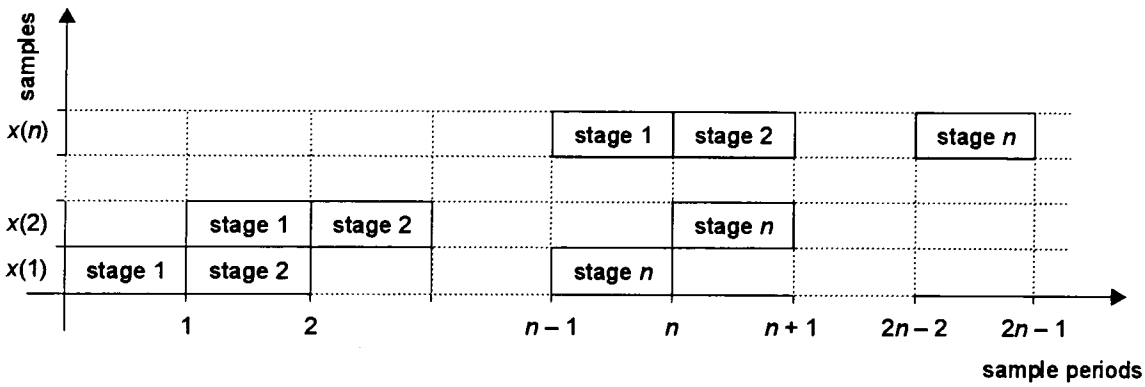

Figure 5-4: Detail of overlapped processing in a *n*-stage linear pipeline.

1. See Section 4.2.3, *Processing Lag and Real-Time Operation.*

5.2 Multi-Processor Topologies

Over the years a plethora of interconnection networks have appeared in the research literature [Bhuyan, 1989]. Here, the differences among the more exotic topologies are investigated.

5.2.1 Single-Stage Networks

Single-stage topologies have particular intrinsic value as they present a uniform methodology to compare and contrast possible M-P configurations for a DMC implementation.

5.2.1.1 Chordal Ring

The chordal ring, Figure 5-5, introduces cross-links between nodes on opposite sides of a simple ring. These chordal links reduce the maximum number of links that must be traversed to reach a destination node, and increase tolerance to node and link failure. Optimal choice of choral link placement reduces the diameter of an n node ring from $O(n)$ to $O(n^{1/2})$.

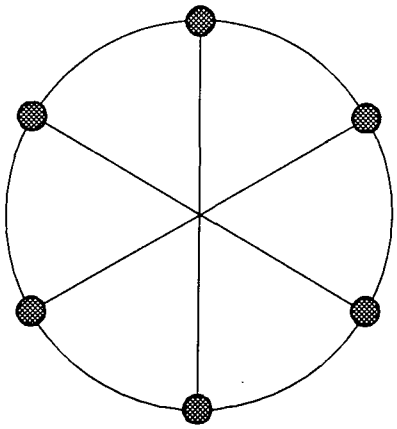

Figure 5-5: Illustration of chordal ring with 6 nodes.

5.2.1.2 X-Tree

The x-tree or full-ring tree is an adaptation of the simple b-ary tree, where b is the fanout at each level. All nodes at each level are also connected in a ring, as shown in Figure 5-6. Increasing resilience to faults, this approach also reduces the communication bottleneck near the tree root. Topology diameter is also reduced for binary and ternary configurations.

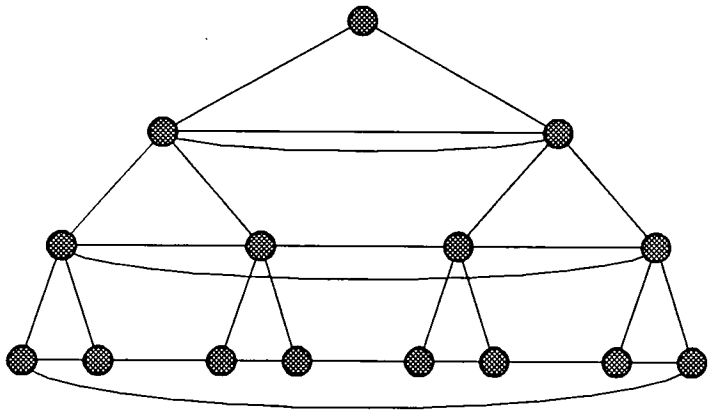

Figure 5-6: Illustration of 2-ary x-tree with 4 levels.

5.2.1.3 Nearest-Neighbour Mesh

With w nodes in each of d dimensions the nearest-neighbour mesh, shown in Figure 5-7 adheres to the generalised d-dimensional hypercube topology. Each of the w^d nodes, except peripheral nodes, is connected to 2 nodes in each dimension by point-to-point links. This reduces the number of links by w^{d-1} giving the total number of links as $(w-1)\,dw^{d-1}$.

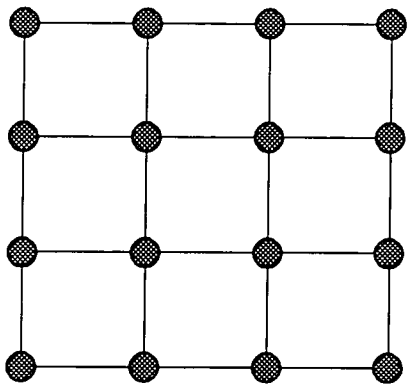

Figure 5-7: Illustration of 2-D nearest-neighbour mesh of width 4.

5.2.1.4 Spanning-Bus Hypercube

The spanning bus hypercube is a d-dimensional lattice of width w in each dimension. Each node of Figure 5-7 is connected to d buses, one in each orthogonal dimension: w nodes share each bus. Note that the binary hypercube used in many of the current generation of commercial M-Ps is a degenerate case of this topology, arising when $w = 2$.[1]

1. Spanning-bus hypercubes should not be confused with the binary hypercube. The former, as the name suggests, uses buses connecting w nodes, the latter uses point-to-point connections.

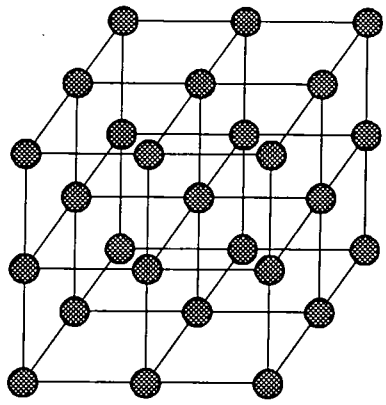

Figure 5-8: Illustration of 3-D spanning bus hypercube of width 3.

5.2.1.5 Dual-Bus Hypercube

Pruning $d-2$ connections from each node gives the dual bus hypercube of Figure 5-9. All nodes are connected to a '0-th' dimension, with a second connection in one of the $d-1$ orthogonal hyperplanes. The direction of this bus differs from hyperplane to hyperplane, but repeats if the width w of that hypercube dimension exceeds $d-2$.

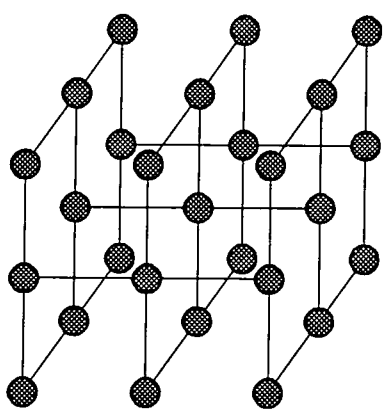

Figure 5-9: Illustration of 3-D dual bus hypercube of width 3.

5.2.1.6 Torus

Identical to the spanning bus hypercube, the torus replaces the bus connecting each group of w nodes with a ring of point-to-point connections, as shown in Figure 5-10. As a special case, the torus includes the closed nearest-neighbour mesh. Lattices, and hence tori, of higher degree can be obtained by further connections to nearest-neighbour nodes.

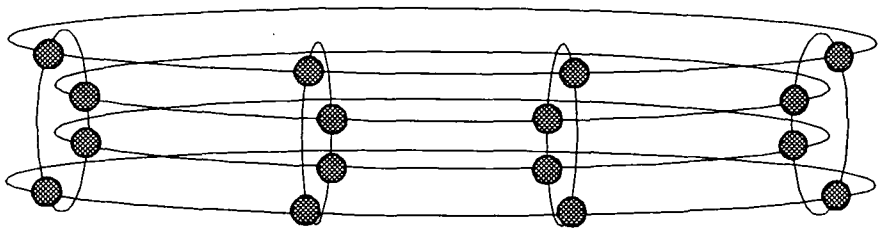

Figure 5-10: Illustration of 2-D torus of width 4.

5.2.1.7 *R*-ary *N*-cube

The R-ary N-cube is a generalisation of the indirect binary n-cube or butterfly network. As there are N levels and R^N nodes on each level, the network contains NR^N nodes each connected to $2R$ other nodes and therefore NR^{N+1} links. Unlike Pease's original proposal, the cube is closed: the top and bottom rows shown in Figure 5-11 are, in fact, the same.

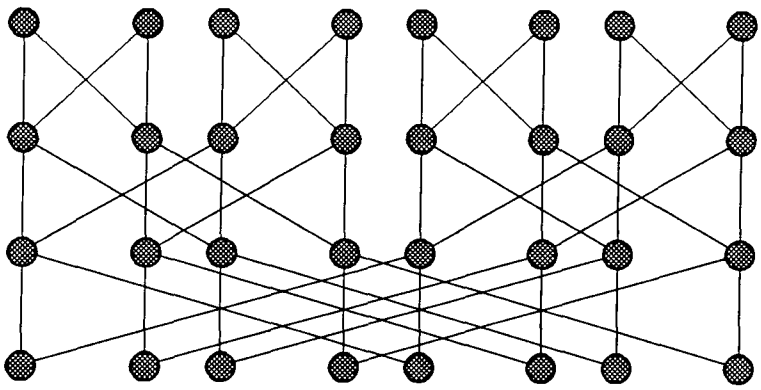

Figure 5-11: Illustration of 2-ary 3-cube.

5.2.1.8 Binary Hypercube

A binary hypercube of dimension d has 2^d nodes, each connected to one node in each d orthogonal dimensions: a total of $d2^{d-1}$ links. Figure 5-12 shows the recursive creation of hypercubes from a single node which is, by definition, a 0-dimensional hypercube. Mapped onto an n-dimensional co-ordinate system, the label of a node is then a tuple of n binary digits.

5.2.1.9 Cube-Connected Cycles

Also proposed to limit the number of connections, cube-connected cycles replace each 2^d nodes of a binary hypercube with a ring of d nodes. Each ring of Figure 5-13 connects to one of d links incident on the vertex. A fixed node valency of three gives a total of $3d2^{d-1}$ links. Expanding the dimension d requires $(d+2)\,2^d$ additional nodes and $3\,(d+2)\,2^{d-1}$ links.

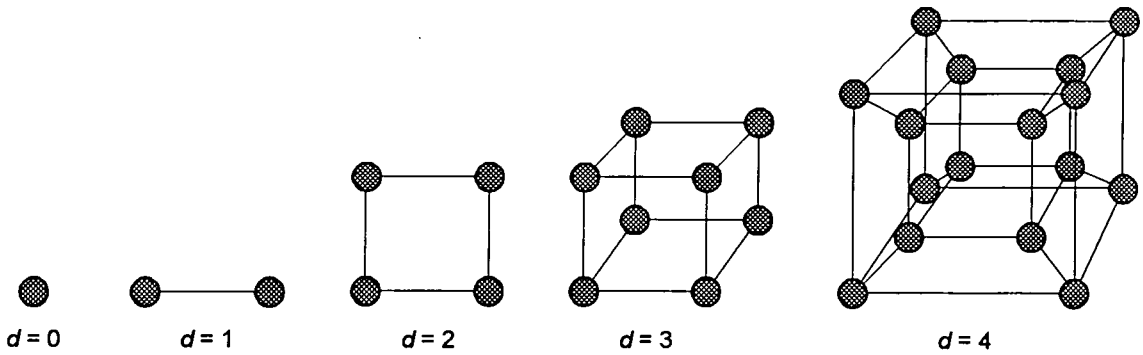

Figure 5-12: Recursive creation of binary hypercubes of dimension *d*.

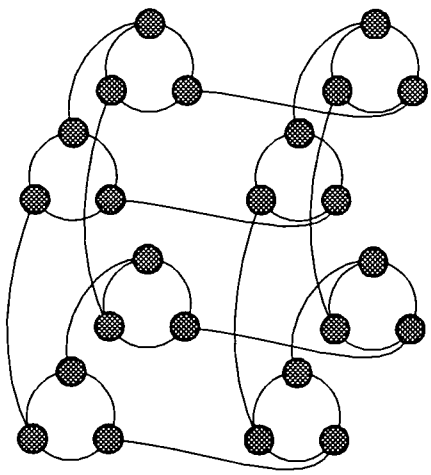

Figure 5-13: Illustration of 3-D cube-connected cycles.

5.2.2 Analysing M-P Topologies

In the following sections x-tree, mesh, torus and binary hypercubes are used to illustrate the analytic techniques used to construct the *Topology Analysis Tool* detailed in Appendix C.

5.2.2.1 X-Tree

If b is the fanout at each level l in a b-ary tree, then the number of nodes is a geometric series:

$$n = \sum_{x=1}^{l} b^{x-1} = \sum_{x=0}^{l-1} b^{x} = \frac{b^{l}-1}{b-1} \qquad \text{Eq. (5-2)}$$

Each node is connected to the level above, to b nodes in the level below, and all nodes at each level are connected in a ring, as Figure 5-6. With the node valency fixed at $b+3$, the number of connections is $(b+3)\,n$ less $b^{l}-3$ missing for peripheral nodes, giving $2\,(n-1)$ links.

Expanding the x-tree topology by incrementing b requires $((b+1)^l - b^l - n)/b$ nodes whereas incrementing l requires b^l. Since the number of links is directly proportional to n, it increments by $2n$ in both cases. Topology diameter is max $(2(l-2), l-1)$ for $b = 2$, max $(2l-3, l-1)$ for $b = 3$ and then $2(l-1)$ for $b \geq 4$ as in a conventional b-ary tree.

5.2.2.2 Nearest-Neighbour Mesh

If all w^d nodes in a mesh were connected to 2 nodes in each dimension there would be a total of $2dw^d$ connections to dw^d links as in the torus below. As noted in Section 5.2.1.3, opening the nearest-neighbour mesh reduces the number of links to $(w-1)dw^{d-1}$. With a uniform node valency of $2d$, unused connections are then available for peripheral I/O devices.

Incrementing the width of the mesh w increases the number of nodes by $(w+1)^d - w^d$, whereas incrementing the dimension d increases n by $(w-1)w^d$. The corresponding increases in the number of links are $wd(w+1)^{d-1} - (w-1)dw^{d-1}$ and $(w-1)w^{d-1}[w(d+1) - d]$ respectively. The maximum inter-node distance is $(w-1)d$.

5.2.2.3 Torus

The d-dimensional torus, a d-dimensional mesh of width w, connects each of w^d nodes to a ring of size w in each of d orthogonal dimensions (see Figure 5-10). As each node is connected to d rings, there are a total of $2dw^d$ connections to dw^d links. Extension of results for the binary hypercube leads to link and node connectivities of $2d$ for arbitrary w.

Topology diameter is $d\lfloor w/2 \rfloor$, that is d times the maximum inter-node distance in any dimension using that dimension's ring. The torus, with its lattice structure, has node expansion increments of $(w+1)^d - w^d$ increasing w, and $(w-1)w^d$ increasing d. Since expanding d requires an increase in valency, the later usually requires the M-P to be re-wired.

5.2.2.4 Binary Hypercube

The binary hypercube, Figure 5-12, contains 2^d nodes each connected to a single node in each of d orthogonal dimensions. A special case of the spanning-bus hypercube, this topology is also a degenerate torus, obtained when $w = 2$ and the 2-node ring replaced with a single link. Hence, there are $d2^d$ link connections and $d2^{d-1}$ communication links.

As adjacent node labels are identical except in one bit position, the topology diameter is thus d. As was noted in Section 5.2.1.8, a $d+1$ dimensional hypercube is constructed recursively from interconnecting two identical d-dimensional hypercubes. Consequently, the node expansion increment of a d-dimensional binary hypercube is simply 2^d.

5.3 Topology Metrics

Many configurations have been proposed for M-P architectures. Many topologies have resulted in actual practical M-P designs while the rest serve as theoretical models.

5.3.1 Overview

Graphs are probably the most natural model to describe M-P INs. Each graph consists of n nodes and a set of point-to-point links connecting pairs of nodes, as Table 5-1. Physically, each node corresponds to one compute processing element (PE). It is possible to define a set of metrics which provide a framework to systematically compare alternative INs. In the following sections, these metrics are defined and then determined for various M-P topologies.

5.3.2 Expansion Metrics

A fundamental consideration is the effect of altering the degree of replication, usually upward as the end-user runs out of compute power. Ideally, a DMC should be incrementally expandible: however, scalability may be restricted by topology-dependent factors. Table B-1 shows that the smallest increments for many M-Ps are dependent on the current configuration.

As M-P topologies are not necessarily incrementally expandible, so the link expansion metrics detailed in Table B-2 are dependent on both n and the specific topology. As the degree of replication is large then a careful analysis must be performed. However, the ability to extend a particular multi-DSP console is clearly only important within a given range.[1]

1. Due to the size of these tables, they have been repositioned as Appendix B.

Topology	Number of Nodes	Number of Connections	Number of Links
full interconnection	n	$n(n-1)$	$\frac{n(n-1)}{2}$
shared bus	n	n	1
simple ring	n	$2n$	n
chordal ring	n	$2\left\lfloor \frac{3n}{2} \right\rfloor$	$\left\lfloor \frac{3n}{2} \right\rfloor$
b-ary tree	$\frac{b^l - 1}{b-1}$	$2(n-1)$	$n-1$
b-ary x-tree	$\frac{b^l - 1}{b-1}$	$4(n-1)$	$2(n-1)$
nearest-neighbour mesh	w^d	$2(w-1)dw^{d-1}$	$(w-1)dw^{d-1}$
spanning-bus hypercube	w^d	dw^d	dw^{d-1}
dual-bus hypercube	w^d	$2w^d$	$2w^{d-1}$
torus	w^d	$2dw^d$	dw^d
R-ary N-cube	NR^N	$2NR^{N+1}$	NR^{N+1}
binary hypercube	2^d	$d2^d$	$d2^{d-1}$
cube-connected cycles	$d2^d$	$3d2^d$	$3d2^{d-1}$

Table 5-1: M-P topology physical characteristics.

5.3.3 Network Connectivity

A DMC should continue to function with reduced capacity when components fail. This property is particularly important for large M-Ps as the probability of flawless operation decreases as $n \to \infty$, for constant MTBF. The link connectivity $lc(i,j)$ between two nodes i and j is defined as the minimum number of links that must be removed to disconnect i and j.

Similarly, node connectivity $nc(i,j)$ is defined as the minimum number of nodes other than i or j that must be removed to disconnect i and j. Network connectivity metrics therefore

measure the resiliency of a M-P to faults and subsequent ability to continue operation. As Table 5-2 illustrates, there is wide variation in the connectivities of M-P topologies.

Topology	Number of Nodes	Link Connectivity	Node Connectivity
full interconnection	n	$n-1$	$n-1$
shared bus	n	1	n
simple ring	n	2	2
chordal ring	n	3	3
b-ary tree	$\frac{b^l-1}{b-1}$	1	1
b-ary x-tree	$\frac{b^l-1}{b-1}$	3	3
nearest-neighbour mesh	w^d	d	d
spanning-bus hypercube	w^d	d	$d(w-1)$
dual-bus hypercube	w^d	2	$2(w+1)$
torus	w^d	$2d$	$2d$
R-ary N-cube	NR^N	$2R$	$2R$
binary hypercube	2^d	d	d
cube-connected cycles	$d2^d$	3	3

Table 5-2: M-P topology network connectivities.

5.3.4 Topology Diameter

The maximum internode distance, or topology diameter δ, places a lower bound on the delay required to propagate data. It is defined as the maximum number of IPC links that must be traversed to pass a message to any node along a shortest path [Fleckenstein, 1992]. Topologies with relatively large n which still maintain small δ are often referred to as *dense* networks.

From Table 5-3, the diameter of most M-P topologies is $O(\log_p n)$ or $O(n^{1/p})$ where p is a topology specific parameter. In isolation, the diameter alone is not a good general measure

of a network's communication capacity as it is biased towards topologies constructed using buses: a shared bus has the same maximum internode distance as full interconnection.

Topology	Number of Nodes	Topology Diameter, δ	Order of δ
full interconnection	n	1	1
shared bus	n	1	1
simple ring	n	$\left\lfloor \frac{n}{2} \right\rfloor$	n
chordal ring	n	$\left\lceil \frac{n}{4} \right\rceil$	$\sqrt{n}$
b-ary tree	$\frac{b^l - 1}{b - 1}$	$2(l-1)$	$\log_b n$
b-ary x-tree	$\frac{b^l - 1}{b - 1}$	$\max(2(l - \max(3 - b/2, 1)), l-1)$	$\log_b n$
nearest-neighbour mesh	w^d	$(w-1)d$	$\log_w n$
spanning-bus hypercube	w^d	d	$\log_w n$
dual-bus hypercube	w^d	$2(d-1)$	$\log_w n$
torus	w^d	$d\left\lfloor \frac{w}{2} \right\rfloor$	$w\log_w n$
R-ary N-cube	NR^N	$N + \left\lfloor \frac{N}{2} \right\rfloor$	$\log_R n$
binary hypercube	2^d	d	$\log_2 n$
cube-connected cycles	$d2^d$	$2d + \left\lfloor \frac{d}{2} \right\rfloor - 1$	$\log_2 n$

Table 5-3: M-P topology diameters.

5.3.5 Node Valency

Table 5-4 details node valencies υ for the M-P topologies considered here. Unfortunately, technological constraints such as pin-out typically limit $\upsilon \leq 6$. While some topologies scale independent of υ, many rely on υ scaling with a topology-specific parameter. Often VLSIs

support a constant off-chip bandwidth β which must support the υ IPC ports. From Table 5-1 it is clear that node valency has a direct bearing on the connectivity metrics of M-P topologies.

Topology	Number of Nodes	Valency, υ
full interconnection	n	$n-1$
shared bus	n	1
simple ring	n	2
chordal ring	n	3
b-ary tree	$\frac{b^l-1}{b-1}$	$b+1$
b-ary x-tree	$\frac{b^l-1}{b-1}$	$b+3$
nearest-neighbour mesh	w^d	$2d$
spanning-bus hypercube	w^d	d
dual-bus hypercube	w^d	d
torus	w^d	$2d$
R-ary N-cube	NR^N	$2R$
binary hypercube	2^d	d
cube-connected cycles	$d2^d$	3

Table 5-4: M-P topology node valencies.

5.3.6 Diameter.Valency Product

As can be seen from Table 5-4 and 5-5, topology diameter δ is mitigated by node valency υ. In open networks, edge processors require fewer links and dangling links are available for I/O. In homogeneous closed M-Ps, all PEs have equal υ and I/O must be accomplished by breaking a link or through the use of supplementary hardware. To quantify this inter-dependency, $\delta \cdot \upsilon$ describes the system costs incurred as a consequence of minimising δ.

Topology	Number of Nodes	Diameter.Valency Product, $\delta \cdot \upsilon$
full interconnection	n	$n-1$
shared bus	n	1
simple ring	n	$2\left\lfloor \frac{n}{2} \right\rfloor$
chordal ring	n	$3\left\lceil \frac{n}{4} \right\rceil$
b-ary tree	$\frac{b^l-1}{b-1}$	$2(b+1)(l-1)$
b-ary x-tree	$\frac{b^l-1}{b-1}$	$(b+3)\max(2(l-\max(3-b/2,1)),l-1)$
nearest-neighbour mesh	w^d	$2(w-1)d^2$
spanning-bus hypercube	w^d	d^2
dual-bus hypercube	w^d	$2d(d-1)$
torus	w^d	$2\left\lfloor \frac{w}{2} \right\rfloor d^2$
R-ary N-cube	NR^N	$2R\left(N+\left\lfloor \frac{N}{2} \right\rfloor\right)$
binary hypercube	2^d	d^2
cube-connected cycles	$d2^d$	$3\left(2d+\left\lfloor \frac{d}{2} \right\rfloor-1\right)$

Table 5-5: M-P topology diameter.valency products.

5.4 Comparison of M-P Topologies

Here, the metrics defined in the previous section and the *Topology Analysis Tool* are applied to various M-P topologies in order to determine the optimum connectivity subset for DMCs.

5.4.1 Simple Topologies

In the context of this comparison, 'simple' includes shared-bus and ring topologies. The number of connections and links are linearly dependent on n, except that a shared-bus has, by

construction, only one IPC 'link'. All three topologies have a node increment of 1, though a chordal-ring with odd n is degenerate as one PE cannot connect to a cross-topology link.

One additional link is required in a simple-ring, whereas chordal-rings expand by 3 links for every 2 PEs. Network connectivities equal υ, except the node connectivity of a shared-bus which must be n. Due to cross-topology links, δ for a chordal-ring is half that of the simple equivalent. Valencies of 1, 2, and 3 give the $\delta \cdot \upsilon$ characteristics of Figure 5-14.

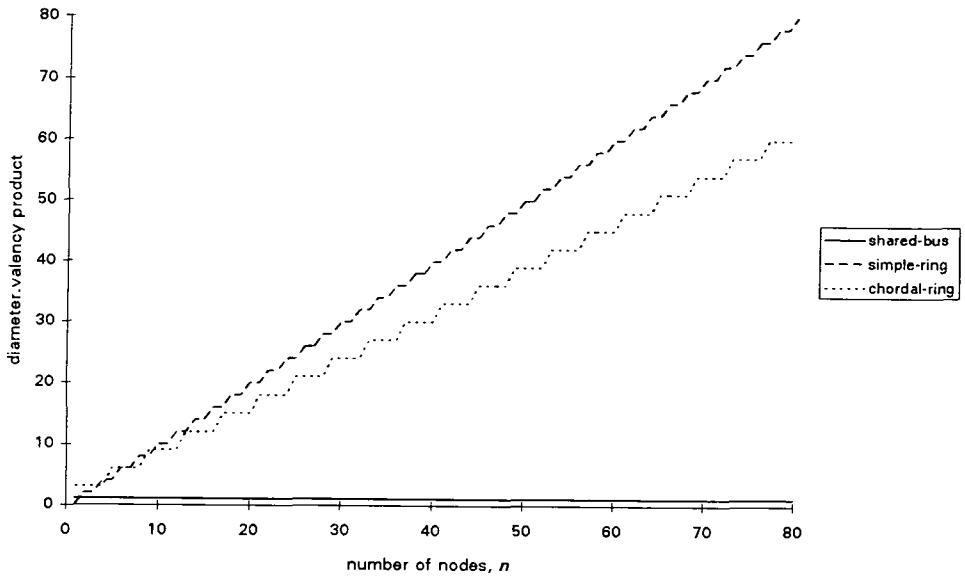

Figure 5-14: $\delta \cdot \upsilon$ characteristics for simple topologies.

5.4.2 Tree Topologies

From Table 5-1, the number of point-to-point links required in a simple b-ary tree is $n-1$. Due to the ring at each level, the number required in an x-tree is twice that of the simple tree.

5.4.2.1 Network Connectivity

From a practical stand-point, it is poor DMC design to expand a tree-based M-P by incrementing b as the PE valency does not remain fixed. Incrementing l, the node increment for expanding any b-ary tree is linear with respect to n, i.e. $n(b-1)+1$. From Figure 5-15, the same is true for link increments for simple b-ary topologies but is 2-times this factor for similar x-trees. As a result, expansion factors for both topologies converge to b as $n \rightarrow \infty$.

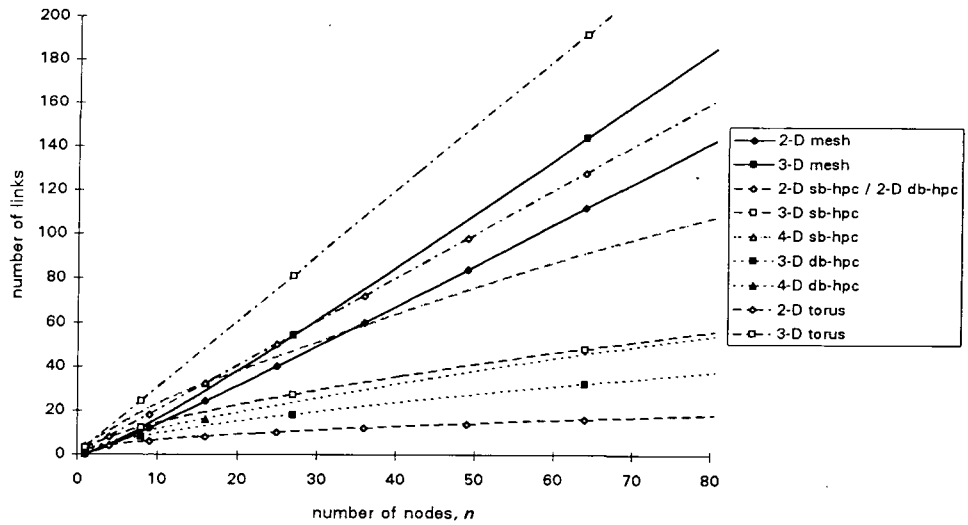

Figure 5-15: Link increments for tree topologies.

5.4.2.2 Diameter & Valency

Since the valency of b-ary topologies is $b + 1$ for simple trees and $b + 3$ for x-trees, link and node connectivities are simply 1 and 3, respectively. As noted in Section 5.2.1.2, 2-ary and 3-ary x-trees have a definite δ advantage over their simple b-ary counter-parts, giving the $\delta \cdot \upsilon$ characteristics of Figure 5-16 below. Because of the practical limits on PE valency discussed in Section 5.3.5, the $b + 3$ -valent 4-ary x-tree is not considered further in this thesis.

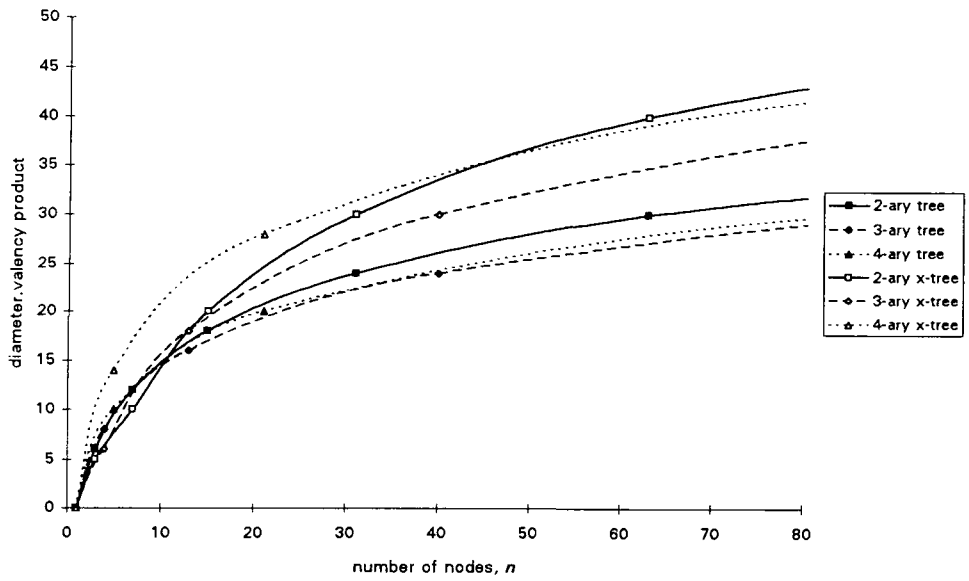

Figure 5-16: $\delta \cdot \upsilon$ characteristics of tree topologies.

5.4.3 Hypercubes

In this section, the hypercube topologies discussed previously are evaluated. 4-D mesh and 4-D torus implementations are excluded from this analysis as they have a node valency of 8.

5.4.3.1 Construction Techniques

While a 2-D mesh requires as few connections as other hypercubes for small n, this is not the case for the size of M-P required for a DMC system. Dual-bus hypercubes require least connections, with 3-D mesh and 3-D tori needing 2 and 3 times this number. Figure 5-17 illustrates the wide variation in the number of links required to construct these M-Ps.

Naturally, bus-based hypercubes have the lowest link requirements with less than 20 buses required for a 81-PE 2-D dual-bus hypercube. With only 1 bus in each orthogonal dimension, this scheme is $d/2$-times cheaper than a spanning-bus implementation. Hypercubes constructed with point-to-point links require over 140 for a M-P of the same size.

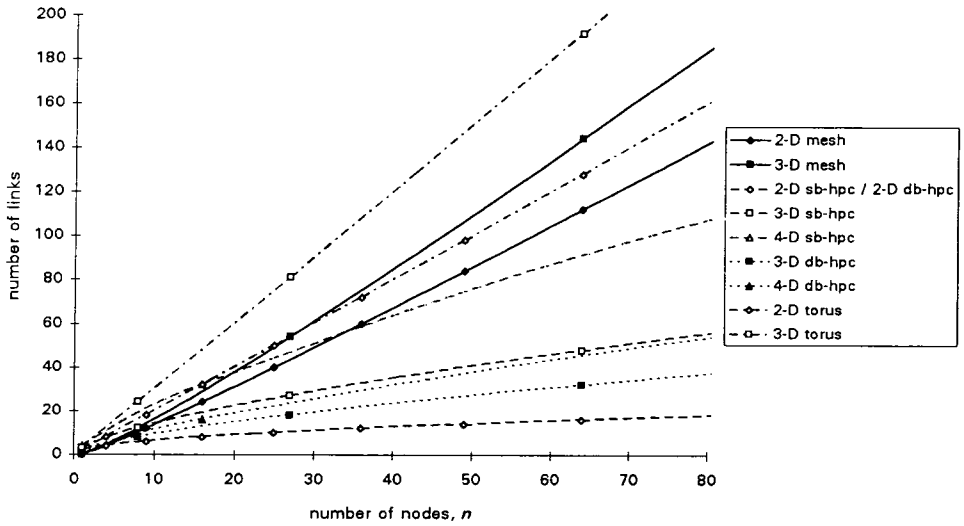

Figure 5-17: Number of links for hypercube topologies.

5.4.3.2 Expansion Metrics

Since the number of nodes in a hypercube is w^d, all hypercube topologies of a given width w and dimension d have the same n and, as a result, identical node expansion metrics. From a manufacturing perspective, lower dimension hypercubes have a definite advantage as lower node increments mean that 8 intact DMC system configurations exist with less than 81-PEs.

Due to the wide variation in the number of links noted in Section 5.4.3.1, link increments also vary widely. By construction, 2-D spanning-bus and 2-D dual-bus hypercubes have a constant link increment of 2. Hypercube topologies based on point-to-point links expand very rapidly since in the order of d links are required for each additional PE.

5.4.3.3 Network Connectivity

Link connectivities are constant and vary between 2 and 6 for the hypercube topologies considered here. The 2-D mesh, 2-D spanning-bus and all dual-bus hypercubes have a metric value of 2. Due to 6- and 3-valent construction, the 3-D mesh and 3-D spanning-bus have a link connectivity of 3. Incrementing d gives a metric of 4, the same as for the 2-D torus.

Under this metric, the best topology is the 3-D torus which can withstand 6 link failures before a PE is isolated. Figure 5-18 illustrates that the node connectivities for the point-to-point hypercubes (mesh and torus) are fixed by the node valency. Due to the construction of the bus-based hypercubes, node connectivity scales with n.

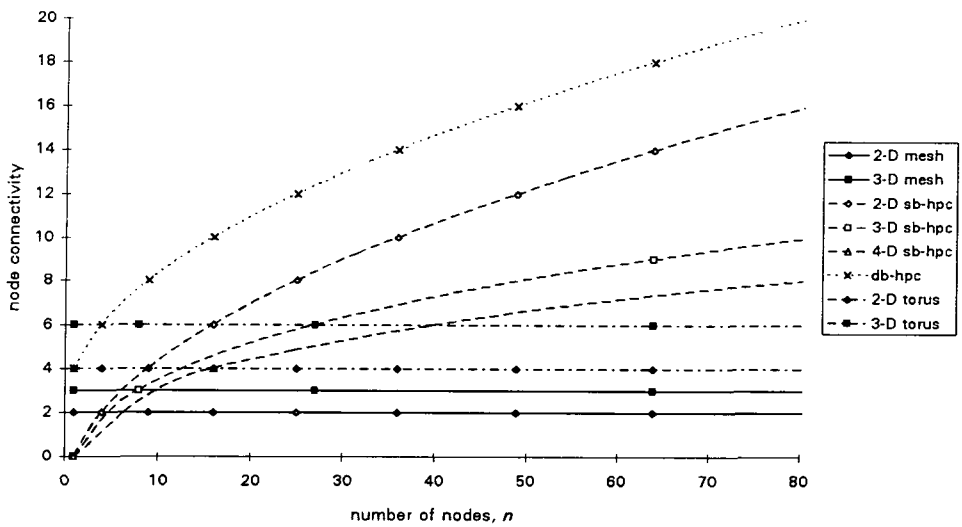

Figure 5-18: Node connectivities for hypercube topologies.

5.4.3.4 Diameter & Valency

Bus-based hypercubes have a definite advantage over their point-to-point counterparts as $\delta \propto d$. Mesh diameter increases with n giving $\delta = 16$ for a 81-node 2-D M-P. Comparable tori reduce this value by half as the topology is closed. Node valency for mesh and tori hypercubes is simply $2d$, whereas dual-bus hypercubes only require a valency of 2.

By construction, shared-bus hypercubes require PEs with $\upsilon = d$. From Figure 5-19, the $\delta \cdot \upsilon$ metric for hypercubes varies depending on the construction method employed. As the bus-based hypercubes have constant υ and a diameter proportional to d, $\delta \cdot \upsilon$ is constant. Consequently, both 2-D spanning-bus and 2-D dual-bus structures have a metric value of 4.

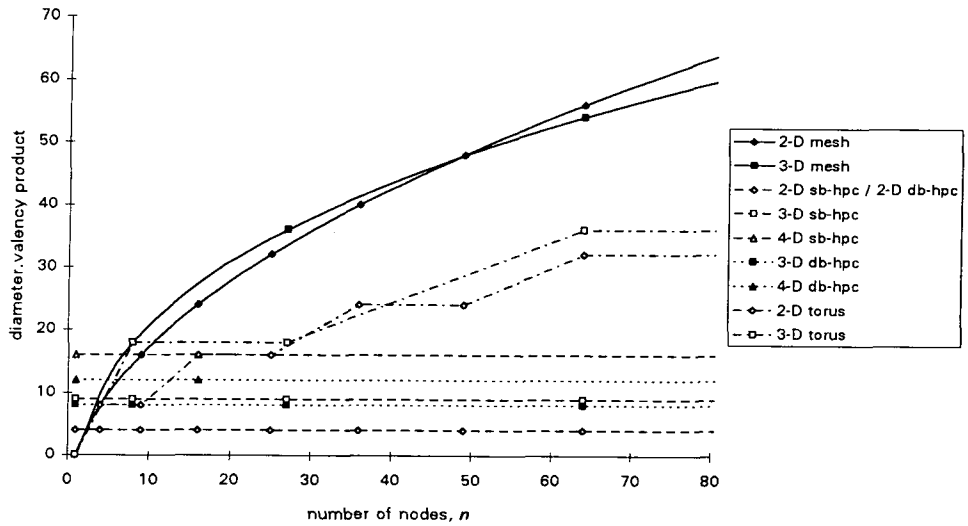

Figure 5-19: $\delta \cdot \upsilon$ characteristics for hypercube topologies.

5.4.4 Preferred Topologies

Although bus-based hypercubes are attractive from a theoretical perspective, bus contention rapidly becomes a critical factor. Here, practical DMC M-P topologies are analysed together.

5.4.4.1 Expansion Metrics

Ideally, a DMC M-P should be incrementally expandible to afford greater design flexibility and a structured upgrade path. From Figure 5-20 it is clear that the scalability of many M-P architectures is restricted by topology-dependent factors if other characteristics are to be preserved. For example, the size of a binary hypercube can only be increased by doubling n.

Whereas shared-bus, chordal-ring and, to a lesser degree, 2-D hypercubes scale in modest increments, 3-ary trees and cube-connected cycles expand rapidly. Other factors are at work here: for example IPC can quickly saturate the nodes near the root of a simple b-ary tree as n increases. Of course, extending a DMC is only important within commercial constraints.

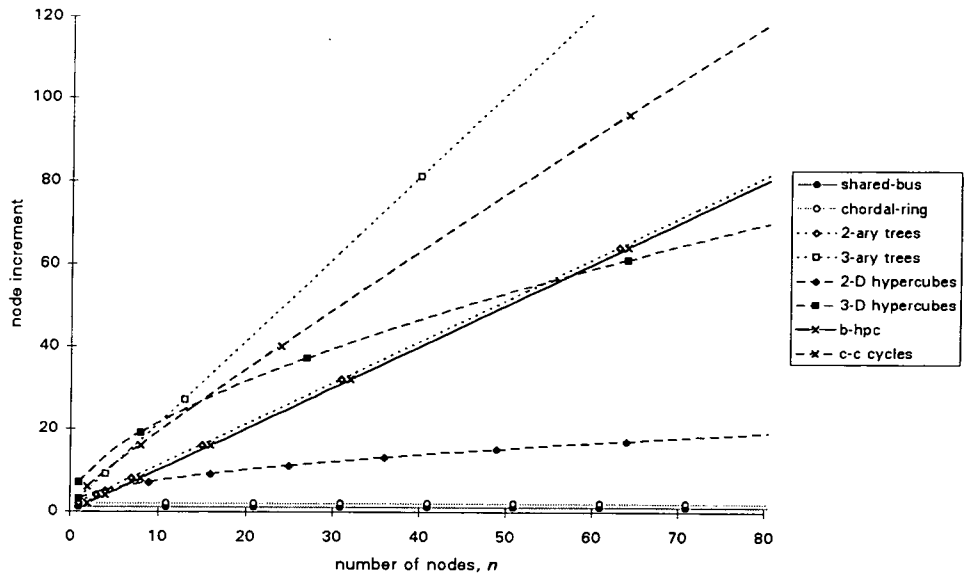

Figure 5-20: Node increments for preferred M-P topologies.

5.4.4.2 Network Connectivity

Alternate paths ensure that the DMC cannot be partitioned by the failure of a single component. Unfortunately, technological constraints limit the number of connections per node. Table 5-2 shows that the tori and binary hypercube are most resistant to link failures though the chordal-ring, x-tree, 3-D mesh, and cube-connected cycles maintain a metric of 3.

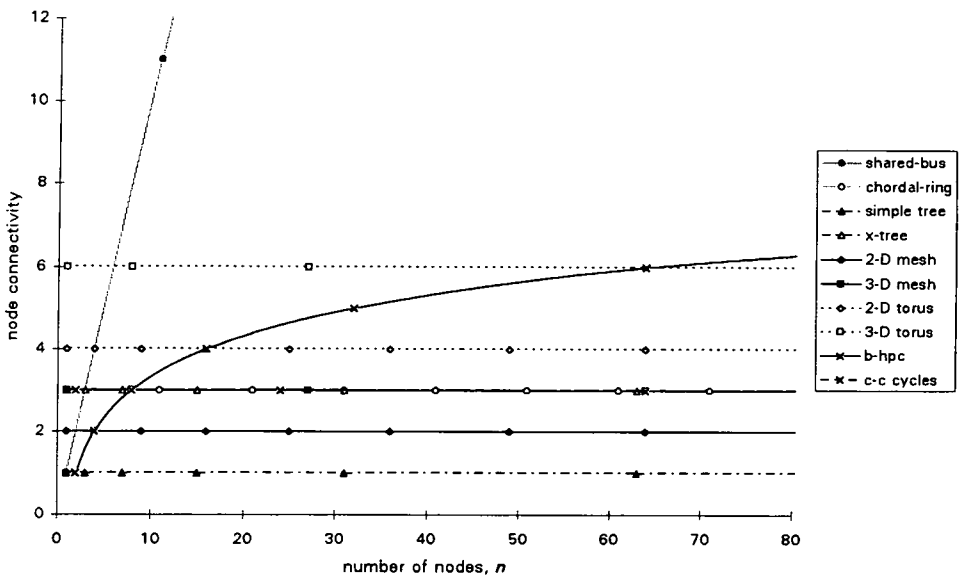

Figure 5-21: Node connectivities for preferred M-P topologies.

Link connectivity for the shared-bus is fixed at 1 as this topology is based on a single IPC channel, whereas Figure 5-21 indicates that the node connectivity is directly proportional

to n. Tori and the binary hypercube again perform well under this metric though, all things being equal, the most unreliable topology with regard to node failures is the simple b-ary tree.

5.4.4.3 Diameter & Valency

Topology diameters vary over a range of 1 to 20 for the size of M-P architectures required for a DMC computation engine. Perhaps surprisingly, the chordal-ring is the worst topology with regards to this diameter since $\delta = 20$, with the 2-D mesh having $\delta = 16$. All other prospective connectivity subsets, apart from the shared-bus, lie within the range 6 to 12.

All topologies discussed in this section have a constant valency except for the binary hypercube where $\upsilon = d$. Figure 5-22 illustrates the $\delta \cdot \upsilon$ characteristics for these topologies. With the valencies required for mesh and chordal-ring configurations, these topologies are particularly expensive in terms of architectural costs incurred in order to minimise δ.

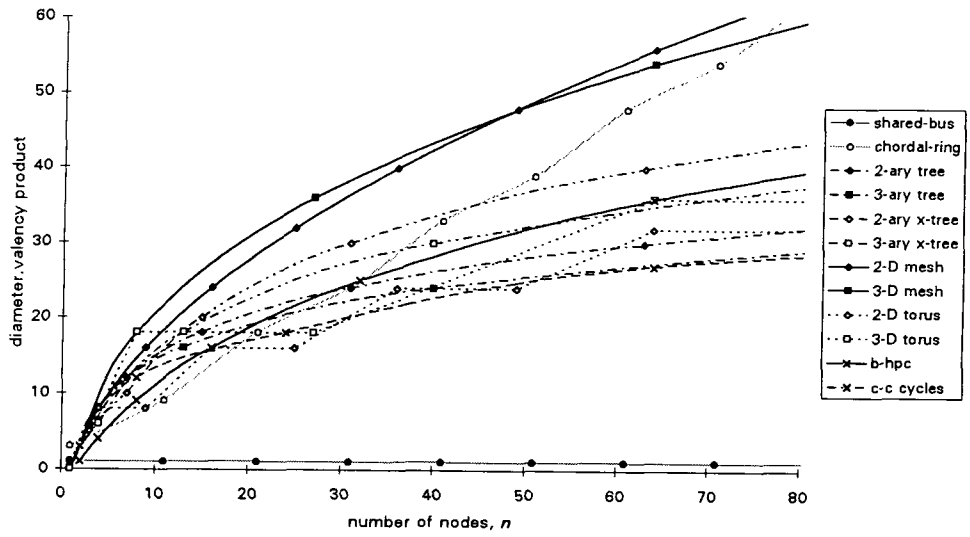

Figure 5-22: $\delta \cdot \upsilon$ characteristics for preferred M-P topologies.

5.5 Conclusions

In this chapter, architectural issues pertinent to the construction of M-Ps for DMC use are developed. Equation (5-1) encapsulates a novel technique for estimating n, starting from the processing budget evaluated in the analysis of Section 3.2.1. The concept of processor pipelining and the fundamental real-time constraints on this scheme are established.

Thirteen single-stage M-P topologies are discussed and four of the more unfamiliar are analysed from first principles to determine their characteristics. A consistent framework of twelve metrics has been subsequently defined for these configurations. This schema has been used to construct the *Topology Analysis Tool* (*TPG*) deliverable described in Appendix C.

As demonstrated in Section 5.4, *TPG* allows the console designer to make a quantitative comparison of prospective DMC system architectures. In simple topologies, the number of links is linearly dependent on n. With a link expansion factor of one, the shared-bus is well suited to small-scale DMCs: however, contention quickly limits extensibility.

Ring architectures, particularly the tri-valent chordal ring, deserve attention from the DMC designer. Cross-topology links reduce the maximum inter-node distance δ to half that of the simple case. Point-to-point IPC channels avoid contention problems, while wormwhole routing, studied in Section 6.2.2, can minimise the effect of $\delta = 20$ when $n = 80$.

Expanding tree-based M-Ps by incrementing fanout b cannot be recommended as the node valency is not fixed. Incrementing the number of levels l, however, fixes the number of IPC ports on each PE. Expansion factors ultimately converge to b, though the number of links required to extend an x-tree architecture is double that required for an equi-valent tree.

If valency is constant, PEs need not be changed to build larger DMCs. 2-D mesh, 2-D torus, and simple trees are the most attractive constant-valency M-Ps with small υ. Less familiar topologies such as cube-connected cycles are also attractive for this reason. Despite being able to embed mesh and tree topologies, υ for a binary hypercube increases with d.

Although the simple b-ary tree appears attractive, it suffers greatly from an imbalance of inter-node traffic near the root node. Ring connections in the x-tree solve this problem while still corresponding closely to converging precedence constraints. Irrespective of these advantages, a PE with a valency of 5 is required to assemble a 2-ary x-tree.

If a primary design goal is a wide product range, lower dimensional hypercubes and trees with restricted b provide a structured upgrade path. While dual-bus hypercubes are extremely undesirable from a manufacturing perspective, hypercube topologies based on point-to-point links expand rapidly as $O(d)$ links are required for each additional PE.

In terms of reliability, the torus and binary hypercube are most resistant to link failures as $nc \propto d$ though at least 3 links must fail in the chordal-ring, x-tree, 3-D mesh and cube-connected cycles before a processor is isolated. Unfortunately, the shared-bus is least reliable from this point of view as no IPC traffic is possible if the single IPC channel should fault.

Dense networks offer short inter-node distances but are more difficult to construct in practice as n becomes large. However, alternative message-passing schemes can minimise the effect of large δ. With this theoretical analysis complete, it is time to consider the question of how to assemble M-Ps suitable for DMCs with commercially available DSP technology.

5.6 References

[Bhuyan, 1989] Bhuyan L N, Yang Q, and Agrawal D P: "Performance of Multiprocessor Interconnection Networks", *IEEE Computer*, pp. 25-37, February 1989.

[Fleckenstein, 1992] Fleckenstein C J (et al.): "Multiprocessing", from *Advances in Computers*, Academic Press, 1992, ISBN 0-12-012135-2.

[Flynn, 1966] Flynn M J: "Very High-Speed Computing Systems", *Proceedings of the IEEE*, Vol. 54, No. 5, pp. 1901-1909, 1966.

[Reed, 1987] Reed D A and Fujimoto R M: *Multicomputer Networks: Message-Based Parallel Processing*, MIT Press, 1987, ISBN 0-262-18129-0.

Chapter 6

Audio M-P Implementation

While the previous chapter investigates architectural issues from a theoretical view-point, this chapter focuses on the implementation of audio M-Ps using programmable DSPs. Message-passing schemes are analysed and design notes presented for a hybrid M-P architecture.

6.1 Multi-DSP Configurations

A limited zero-chip interface may be implemented by connecting the serial ports of DSPs. For audio-rate IPC, the external memory interface (EMI) must be used to connect multiple DSPs.

6.1.1 Shared-Bus Design

Shared-bus architectures are probably the most common form of M-P, since they are the natural extension of conventional uni-processor DSP systems.

6.1.1.1 Introduction

The simplest way to construct an audio M-P is to connect DSPs on a shared-bus, as shown in Figure 6-1. In order to isolate DSPs so that they can simultaneously access local DSP RAM, bi-directional bus transceivers are required: the bus is then active only during IPC. Without local DSP RAM, bus contention would result in severe arbitration delay [Snell, 1989].

Devices such as the DSP56001 have no provision to force the EMI to quickly tri-state in a deterministic manner. Since the address bus is always driven, one DSP's local RAM cannot be addressed by another DSP while the first accesses internal memory. Additional bus transceivers stop the DSP56001 shorting its main bus, but increase board real-estate.

6.1.1.2 Arbitration Schemes

Arbitration schemes enable multiple processors to access the same piece of data at the same time. Most DSP devices have provision for asynchronous sharing of their EMI, with the use of

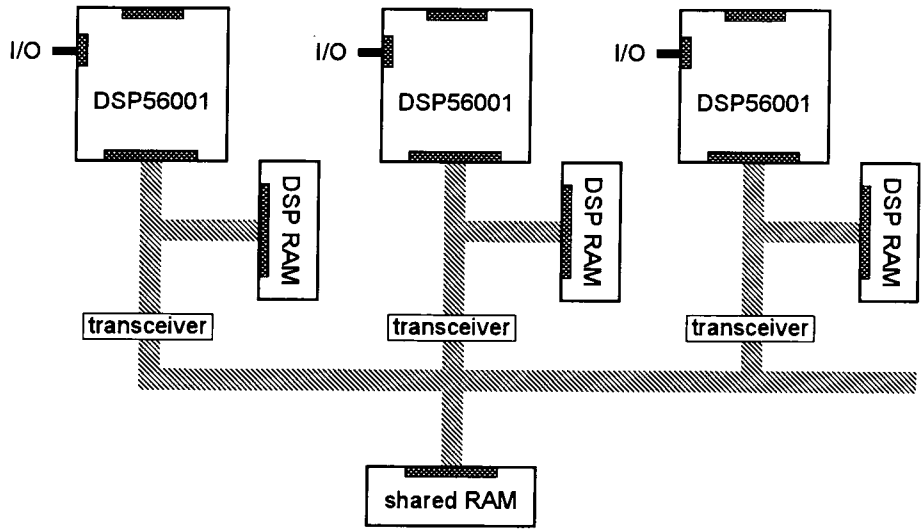

Figure 6-1: Example shared-memory M-P architecture.

bus-request (BR) and bus-grant (BG) signals. Architectures which use BR and BG signals directly typically require a hard-wired master-slave relationship between the DSPs.

There is little space in a 48 kHz sample period for BR/BG arbitration. In synchronous designs, the most serious problem is the indeterminate timing of BG. For example, the BR to BG delay for a 20.5 MHz DSP56001 is between 73 and 240 ns. Since the upper bound is greater than one instruction cycle, synchronous DSPs cannot be kept in lock-step.

6.1.1.3 Disadvantages of Bus Systems

An n-processor system requires an IPC media whose bandwidth is of the order of n-times that of a uni-processor. Consequently, the major shortcoming of shared-bus M-Ps is self-evident: IPC bandwidth is constant. As n increases, the shared bus between DSP modules becomes a serious bottle-neck, fixing an upper limit on the extensibility of the design.

For limited n, bus-oriented systems can yield high performance/cost ratios without any pretensions of scalability. Beyond this range, delays due to contention and subsequent arbitration lead to considerable performance degradation. Such systems cannot support DMCs unless a technological breakthrough provides a very high bandwidth bus at relatively low cost.

6.1.2 Alternative Synchronization Mechanisms

Clock skew is a considerable problem in large synchronous M-P architectures. In asynchronous designs, synchronization mechanisms ensure that shared data is always valid.

6.1.2.1 Semaphores

Semaphore flags, first proposed by Dijkstra in 1965, are non-negative integer variables used as locks to shared resources [Dijkstra, 1965]. Two different operations ensure that only one PE gains access at once: a request operation which attempts to gain access and a release operation which signals access termination. Together these two processes guarantee mutual exclusion.

At one extreme a shared memory can be controlled by a single semaphore, whereas at the other every data item could be individually locked. The former precludes any parallel operations on shared data, while in the latter the overhead of locking operations is prohibitive. Typically, this overhead is distributed with one semaphore locking the corresponding data-set.

6.1.2.2 Test-and-Set

One popular semaphore technique is the *test-and-set* primitive. Firstly, the DSP requiring exclusive access reads the corresponding flag. It then tests this prior value to determine if it was successful (i.e. a '0') and, finally, forces the semaphore to '1'. If the flag was not already set, then this DSP has permission to access the shared data: all other processors are locked out.

The test-and-set must be uninterruptible, otherwise synchronization could fail. One PE may test the semaphore and, before it can be rewritten, another PE may also initiate a test. Both processors then believe they have exclusive access. Most DSPs support an indivisible READ/MODIFY/WRITE (RMW) instruction, ensuring that the address bus remains active.

6.1.2.3 Spin-Lock

Test-and-set is only one half of this mechanism: the other is the action taken depending on whether or not access has been granted. One technique which gives fast resource entry is *spin-lock*. If the lock is granted, the PE continues on to use the shared resource. If not, then a spin-lock occurs and the PE spins backwards to re-execute the test-and-set, as Figure 6-2.

There are, however, two drawbacks to this scheme. Firstly, a PE polling a semaphore does no useful ASP. Secondly, it consumes cycles in the memory containing the semaphore. If several DSPs are waiting at the same flag, contention rapidly consumes IPC bandwidth. Spin-locks can be wasteful, dedicating instruction cycles to repeatedly testing semaphores.

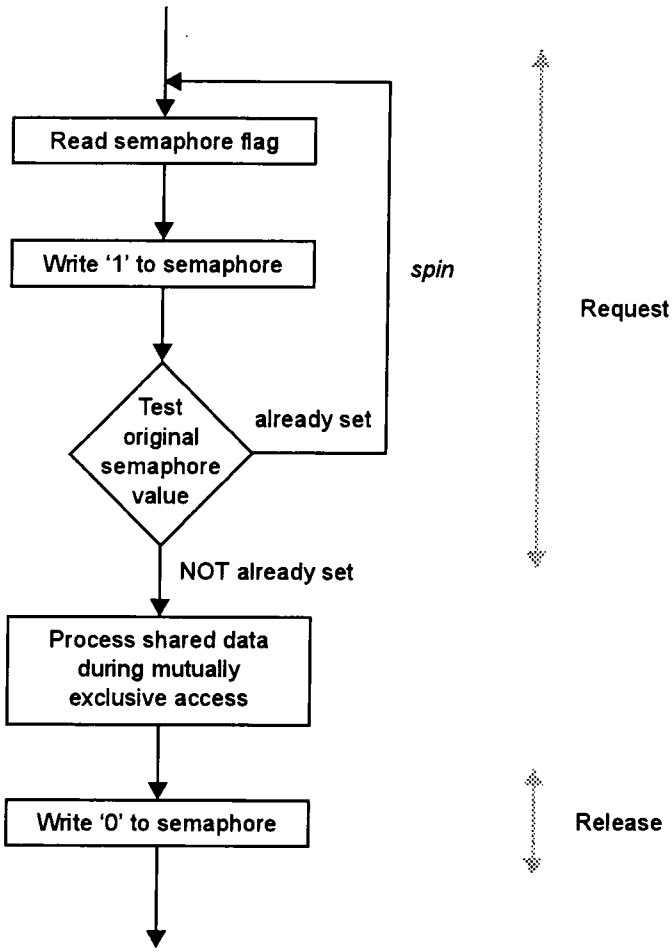

Figure 6-2: Flow chart of *test-and-set* and *spin-lock* semaphore operations.

6.2 Message-Passing IPC

Communication efficiency, one of the most important factors to be considered when designing a M-P architectures, often becomes one of the main obstacles to computation performance.

6.2.1 Overview

The simplest way to interconnect multiple DSPs for message-passing is to provide full connectivity. Another option is to employ some form of partial connection topology as discussed in the previous chapter. When a message passes between a pair of DSPs in a DMC, it may be routed through the M-P in a number of hops. Thus the communication speed of the architecture depends not only on the M-P topology but also the method of IPC used.

6.2.2 Routing Schemes

One method used to route messages between PEs requires intermediate PEs to store the entire message. Alternatively, the message may be directly transferred to the destination processor.

6.2.2.1 Store-and-Forward

Many first-generation M-Ps such as the Intel iPSC/1 binary hypercube use a store-and-forward mechanism [Tanenbaum, 1981]. As messages pass indirectly between source and destination, each intermediate PE must temporarily store the message. While messages move between nodes, memory bandwidth and instruction cycles are consumed in intermediate PEs.

Communication latency is sensitive to the distance across the IN a message must be passed. As latency is linearly proportional to the number of hops, it can be expressed as:

$$T_s = T_1 H \qquad \text{Eq. (6-1)}$$

where $T_1 = K/B_1$ is the time for a message of K bytes to pass through a link of bandwidth B_1 bytes/s. If H is the number of hops, then T_1 is the routing delay of each node.

6.2.2.2 Wormhole

Wormhole routing is used in many second-generation M-Ps, like the Intel iPSC/2, to reduce message latency caused by topology diameter [Athas, 1988]. Instead of storing a complete message, only a few flow-control digits (*flits*) need be buffered at each node. Flits at the head of the message govern the route, while the remaining bits follow in pipeline fashion.

The communication latency of the wormhole routing scheme can be defined as:

$$T_w = T_2 H + \frac{K}{B_2} \qquad \text{Eq. (6-2)}$$

where $T_2 = k/B_2$ is the time required for the message head k to pass through a channel of bandwidth B_2 bytes/s. H is the number of hops as before, and K/B_2 is the time required to transmit the message of K bytes continuously through the B_2 bytes/s wormhole channel.

6.2.2.3 Comparison

The ratio between Equations (6-1) and (6-2) is a quantitative comparison of the two schemes:

$$\frac{T_s}{T_w} = \left(\frac{B_2}{B_1}\right)\frac{KH}{kH + K} \qquad \text{Eq. (6-3)}$$

Typically, $k \ll K$ and, as shown in Figure 6-3, Equation (6-3) can be approximated by:

$$\frac{T_s}{T_w} \approx \left(\frac{B_2}{B_1}\right) H \qquad \text{Eq. (6-4)}$$

Equation (6-4) indicates that wormhole routing reduces communication latency by up to $(B_2/B_1) \times H$ times over store-and-forward for similar message size K and distance H.

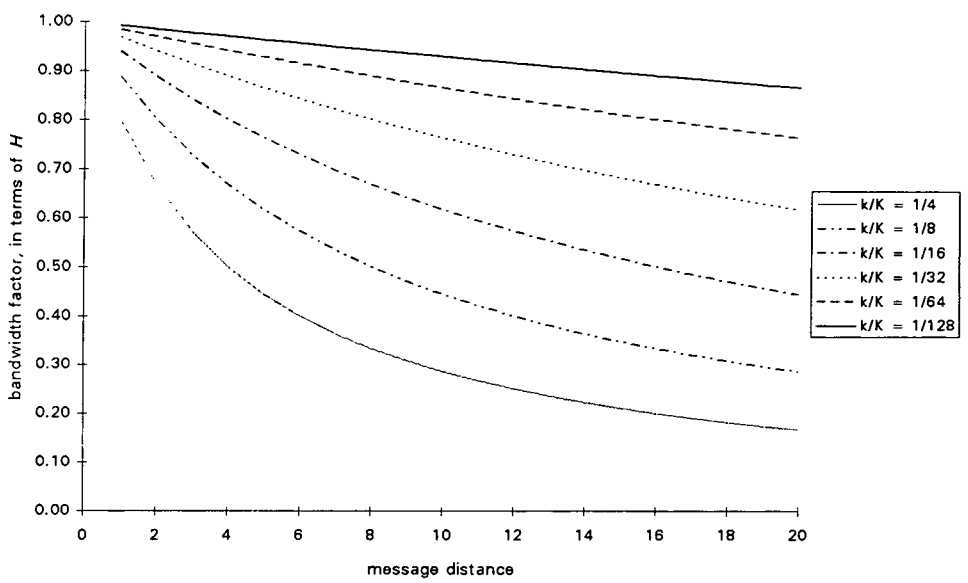

Figure 6-3: Analysis of Equation (6-3), for various message head/size ratios.

For example, the first-generation iPSC/1 uses store-and-forward with a bandwidth of 1.25 Mbytes/s, whereas the iPSC/2 uses the wormhole routing and 4 Mbytes/s. For a 32-node hypercube $H = 5$, indicating that a 32-node iPSC/2 reduces communications latency by 16 times over a 32-node iPSC/1. Even if $B_1 = B_2$, T_w is still reduced by a factor of H.

6.2.3 Comparing Message-Passing M-P

Intel iPSC systems are based on the binary hypercube, while many other commercial M-Ps use a 2-D mesh. In this section wormhole topologies are compared in terms of IPC latency.

6.2.3.1 Constant Bisection Bandwidth

As noted in Section 5.2.1.8, a d-D binary hypercube is created by duplicating the bisection, or $(d-1)$-D hypercube, and inter-connecting corresponding nodes from each. This approach

needs $n/2$ links across the bisection: using the same method to construct a 2-D mesh requires $\sqrt{n}$ links. The diameter of a binary hypercube is $\log_2 n$, and $\sqrt{n}$ for a 2-D mesh.

Assuming a constant bisection bandwidth, the 2-D mesh has $\sqrt{n}$ times more bandwidth per link than an n-node hypercube. For the maximum H, Equation (6-2) becomes:

$$T_{\text{b-hpc}} = T_d \log_2 n + \frac{K}{B} \qquad \text{Eq. (6-5)}$$

Since the link bandwidth in a 2-D mesh is $\sqrt{n}$ times wider than for a binary hypercube:

$$T_{\text{mesh}} = T_d \sqrt{n} + \frac{K}{B\sqrt{n}} \qquad \text{Eq. (6-6)}$$

$$\frac{T_{\text{b-hpc}}}{T_{\text{mesh}}} = \sqrt{n}\frac{T_d \log_2 n + T_c}{T_d N + T_c} \qquad \text{Eq. (6-7)}$$

When K is reasonably large, T_d may be ignored, and the ratio in Equation (6-7) is $O(\sqrt{n})$.

6.2.3.2 Constant Node Bandwidth

In many M-Ps, the I/O pins of each DMC PE support a constant bandwidth β. Under this assumption, if the valency of a node is υ, the bandwidth of each link is β/υ. For a d-D binary hypercube $n = 2^d$, the diameter $\delta = \log_2 n$, and the valency $\upsilon = \log_2 n$. From Equation (6-3), the maximum latency of binary hypercube using wormhole routing is:

$$T_{\text{b-hpc}} = T_d \log_2 n + \frac{K \log_2 n}{\beta} \qquad \text{Eq. (6-8)}$$

where $T_d = k\upsilon/\beta$ is the delay of individual nodes on the wormhole path. Similarly, the metrics from Chapter 5 for a 2-D mesh, 2-ary x-tree, and chordal ring yield, respectively:

$$T_{\text{mesh}} = 2T_d(\sqrt{n} - 1) + \frac{4K}{\beta} \qquad \text{Eq. (6-9)}$$

$$T_{\text{x-tree}} = 2T_d(l - 2) + \frac{5K}{\beta} \qquad \text{Eq. (6-10)}$$

$$T_{\text{c-ring}} = T_d\left\lceil \frac{n}{4} \right\rceil + \frac{3K}{\beta} \qquad \text{Eq. (6-11)}$$

The second term K/B_2 of Equation (6-2) dominates latency for all but very short messages. When $K \gg k$, $T_d \rightarrow 0$ giving the latency ratios in terms of valency as illustrated in Figure 6-4.

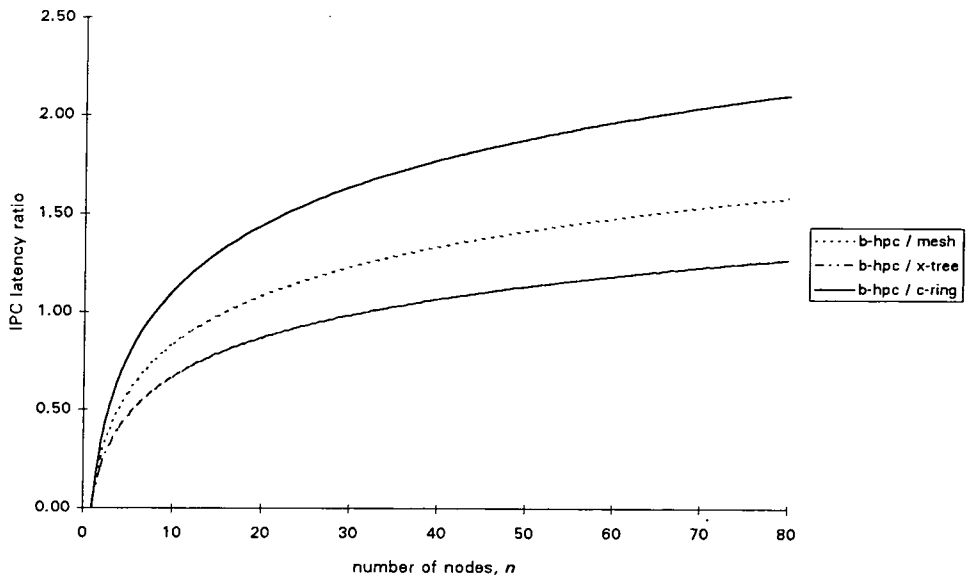

Figure 6-4: IPC latency ratios for several topologies given constant node valency.

6.3 Message-Passing Realisation

Various schemes have been suggested for message-passing between conventional DSP processors. Here, approaches which utilise widely available integrated devices are developed.

6.3.1 First-In First-Out Buffers

One common technique for providing an elastic data buffer between two asynchronous or asynchronous systems is the first-in first-out (FIFO) buffer.

6.3.1.1 Integrated FIFOs

Integrated products, such as the devices available from IDT, provide FIFO buffers at very fast speeds (50 ns). Two self-incrementing pointers locate where reading (READ) and writing (WRITE) take place in a dual-ported circular-buffer. This approach results in very low fall-through times[1] compared to first-generation designs based on ganged shift-registers.

In second-generation designs, separate READ and WRITE ports allow for simultaneous asynchronous accesses without hand-shaking or arbitration. Additional logic provides empty

1. The fall-through time of a FIFO is the time elapsing between the end of the first write to the FIFO and the time the first read may begin.

(EF), half-full (HF), and full (FF) queue flags. Two additional pins, one input (XI) and one output (XO), provide for unlimited expansion while maintaining the 50 ns fall-through time.

6.3.1.2 Status Flags

Status flags allow the boundary conditions of integrated FIFOs to be monitored. As soon as a WRITE occurs, EF is de-asserted. When WRITE operations fill the buffer, FF flags that there are no more empty locations. If a READ reduces data to just below half-way, then HF is deactivated. EF and FF are also fed back internally, to eliminate data over- or under-flow.

Flagged FIFOs offer the features discussed above plus two additional programmable flags: ALMOST-EMPTY (AE) and ALMOST-FULL (AF). These flags can be used as early warning flags in real-time applications such as pipeline DSP. In a multi-tasking environment, these enhanced flags can set interrupt requests in advance to minimise IPC latency.

6.3.1.3 FIFO Memory Expansion

If the maximum FIFO requirement is 1024 locations of 9-bits, a single IDT7202 will suffice. Wider word widths can be achieved by connecting several devices in parallel. Status flags can be detected from any device because each device is working in lock-step parallel. Classical architectures require more external logic to account for differences in internal clocks.

In older architectures, fall-through time increased in direct proportion to the number of devices connected in series. A parallel architecture results with the two-pointer approach retaining the single-chip fall-through time. Since FIFOs do not have chip selects and external decoding mechanisms, XO and XI pins allow the pointer enable to cross device boundaries.

6.3.2 Dual-Ported RAM

In dual-ported RAM (DPR) two sets buses and control signals access one block of RAM. As a result, two DSPs can share the same physical memory in their respective address spaces.

6.3.2.1 Integrated DPR

Several manufacturers produce integrated DPR operating at static RAM speeds, shown in Table 6-2. Previously, few true DPRs were commercially available: designers configured external logic and single-ported RAM to simulate DPR operation. Integrated DPR devices

out-perform discrete parts designs as bus-access arbitration is no longer required. Nevertheless, synchronization must be performed at other levels to ensure data integrity.

Width	Size	Part No.	Support Logic			
			Interrupt	Busy Logic		Semaphore
				MASTER	SLAVE	
× 8	1K	IDT7130	■	■		
		IDT7140	■		■	
	2K	IDT7132		■		
		IDT7142			■	
		IDT71321	■	■		
		IDT71421	■		■	
		IDT71322				■
	4K	IDT71342				■
× 16	2K	IDT7133		■		
		IDT7143			■	

Table 6-1: Selection of dual-ported RAMs available from Integrated Device Technology.

6.3.2.2 Interrupt Logic

A common problem in DPR systems is signalling between processors. For example, DSP-1 needs to signal DSP-2 to request a task to be performed, and when DSP-2 has completed, it needs to signal DSP-1 that the task is done. Note that the signalling must occur in both directions. A common form of signalling is for one DSP to cause an interrupt on the other.

On certain DPR devices, the top two memory addresses also as dedicated interrupt generators. If DSP-1 writes into one of these addresses, an interrupt latch is set and the interrupt line to the other port activated. The interrupt latch is cleared when DSP-2 reads from this address. A similar set of logic is provided to allow DSP-2 to cause an interrupt on DSP-1.

6.3.2.3 Busy Logic

There are two significant cases: if one port is reading while the other is writing, data on the read side will change; if both ports attempt to write at the same time, the result can be random.

Busy logic solves this problem by detecting when both sides access the same location. The BUSY pins of DPR devices are suitable for attachment to the WAIT inputs of most DSPs.

Although one or other processor may have to wait occasionally, throughput loss is minimal. For example, in a 1K word DPR with a relatively uniform and random access profile, the probability of any given location being accessed by one side is $O(10^{-3})$. Average throughput can be reduced by only one part in a million due to DPR access contention.

6.3.2.4 Semaphore Support

Software constraints may require mutual exclusion of data structures rather than memory locations. Instead of comparing addresses on every cycle and occasionally asserting BUSY, other DPR devices support semaphores. Conventional test-and-set semaphores require that the two memory accesses are indivisible: some DPR devices employ a twist by using set-and-test.

The *set* corresponds to a request and the *test* checks to see if the request was granted. As the indivisible double access requirement is avoided, semaphores can be supported for DSPs which do not provide an indivisible RMW. Since there is no hardware relationship between semaphores and DPR locations, variable channel widths can be defined in software.

6.3.2.5 DPR Memory Expansion

Integrated DPRs can be combined to form larger dual-port memories. Expansion in depth is similar to that used for conventional RAM, where the address MSBs enable individual memory blocks. DPRs can also be expanded in width. If addresses for both ports arrive simultaneously it is possible for both sides to assert BUSY, causing both DSPs to live-lock.

Expanded DPRs used for DSP-to-DSP IPC is shown in Figure 6-5. Two DSP56001s communicate using three 8-bit wide DPR chips in a MASTER/SLAVE word-wide configuration. If the MASTER asserts BUSY, the BUSY input on the SLAVE devices disables their write enables. Note that the BUSY signals from the MASTER are also fed to the WAIT pins of the DSPs.

6.3.3 DPR and FIFOs Compared

A 2K × 24-bit DPR block requires three 48-pin ICs, plus address decode components. Using uni-directional devices, a bi-directional 2K × 24-bit FIFO channel requires a total of 6 × 28-pin ICs. With each doubling in size, 2 address pins must be added to the DPR package. Larger

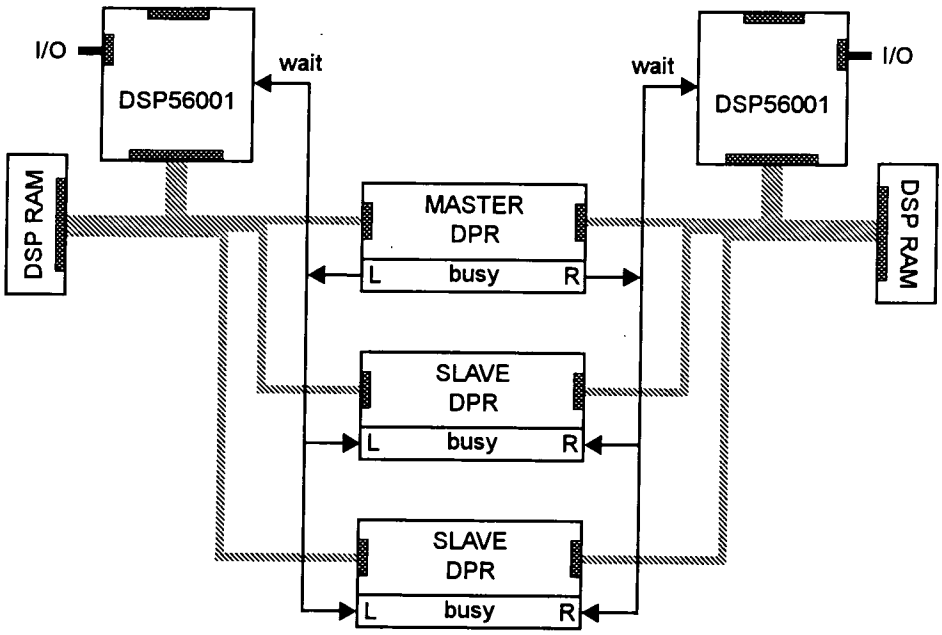

Figure 6-5: DSP-to-DSP communication using width-expanded dual-ported RAMs.

FIFOs use the same package size, since memory locations are not directly addressed. While FIFOs scales better than DPRs, the latter provide considerably more IPC flexibility.

6.3.4 Novel Devices

Recent technological developments in VLSI design have resulted in the emergence of novel integrated devices which have direct application in multi-DSP architectures.

6.3.4.1 Synchronous FIFOs

High-speed integrated FIFO may recognise noise-induced glitches as valid control pulses. Third-generation synchronous integrated FIFO designs reduce sensitivity to glitches via input and output registers. Independent external clocks force READ and WRITE operations to occur on clock edges. IDT's *SyncFIFO* products allow the depth of status flags to be programmed.

Many high-speed digital systems rely on a synchronous approach, since the complexity of asynchronous control circuitry increases with system clock frequency. In a synchronous system, the amount of control logic is minimal. If a design is made synchronous at the outset, later clocking improvements do not require redesign of control logic.

6.3.4.2 Quad-Ported RAM

Integrated quad-ported RAMs (QPRs) simplify the task of creating generalised multi-DSP systems for real-time ASP, all but preventing IPC contention. Although typically three times as expensive as comparable DPR devices, QPRs have a considerable advantage over DPRs. In order to fully interconnect four DSPs requires 1 bank of QPR as opposed to 6 banks of DPR.

As in DPR devices there are only two constraints on access patterns (see Section 6.3.2, *Dual-Ported RAM*). Hardware methods of controlling access — interrupts, semaphores and busy logic — and software-only token-passing schemes are possible. Separate locations dedicated to the status of each processor can also guarantee clean inter-task communications.

6.3.4.3 CalTech Torus Routing Chip

General purpose M-P routing components have been studied by several researchers. One such component is the Torus Routing Chip (TRC) developed at CalTech. This trivalent design was primarily designed to support efficient message-passing in 2-dimensional tori of width *w*. Two bi-directional links carry message traffic between distinct nodes: the third allows several chips to be cascaded permitting the construction of higher dimension tori.

6.4 *HYMIPS* — A Hybrid Audio M-P

Architectural issues studied in this and the previous chapter have led to the construction of a scalable hybrid M-P — *HYMIPS* — designed specifically for real-time, multi-channel ASP.

6.4.1 Design Decisions

In this section, design decisions surrounding the choice of routing and compute processors, interconnection components and memory maps for *HYMIPS* are considered.

6.4.1.1 Processors

The Inmos transputer is capable of parallel communication and computation, providing an ideal building block for scalable M-P architectures. IPC is accomplished by asynchronous point-to-point links which incur minimum CPU overhead [Inmos Ltd., 1989]. Unfortunately, as noted in Section 4.6.5, the transputer ALU is not efficient at executing ASP algorithms.

In contrast, DSPs such as the Motorola DSP56001 are specifically designed to implement such algorithms (see Section 4.6.1, *Parallel-Bus DSPs*). Although supporting a high degree of operational concurrency, M-Ps are not easily implemented with these devices alone. A hybrid M-P offers the required properties of scalability and efficient execution.

6.4.1.2 Interconnection

As discussed in Section 6.1.1, the fixed IPC bandwidth of the shared bus topology is a severe constraint on extensibility: bus contention results in substantial IPC latency. A variation of this architecture is to use DPR as the shared resource, as shown in Figure 6-6. Although the T801 and one DSP may simultaneously access DPR, DSPs still experience bus contention.

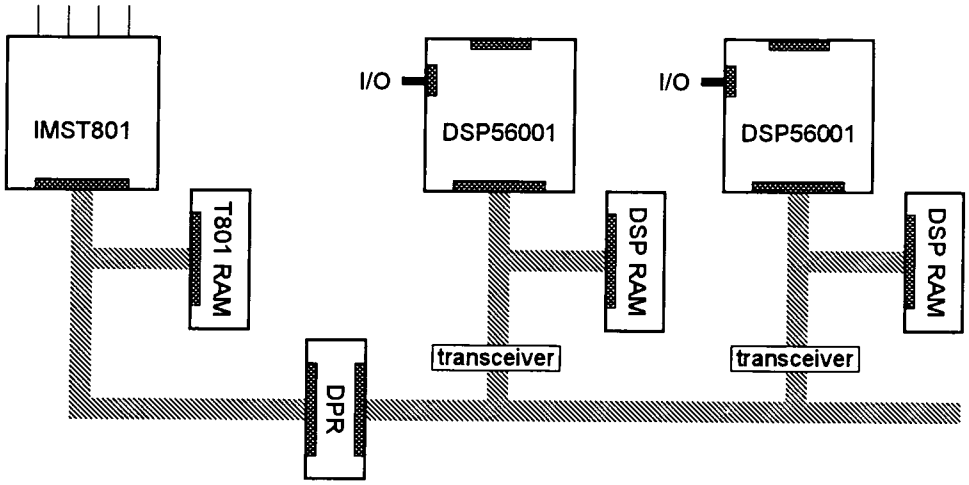

Figure 6-6: Dual-bus hybrid M-P architecture.

If this scheme is extended to provide each compute PE with a DPR channel to the transputer EMI as Figure 6-7, bus access arbitration is no longer required. IPC data may be mapped into the relevant DPR using the Occam PLACE instruction. A DSP processes data-set n in parallel with the routing processor transferring $n - 1$ and $n + 1$ via DPR channels.

6.4.1.3 Memory Maps

While the DSP56001 has 3 independent memory spaces, only one can be accessed via the EMI in any cycle. 2 Kwords of DPR is mapped to a sufficiently high base address to allow

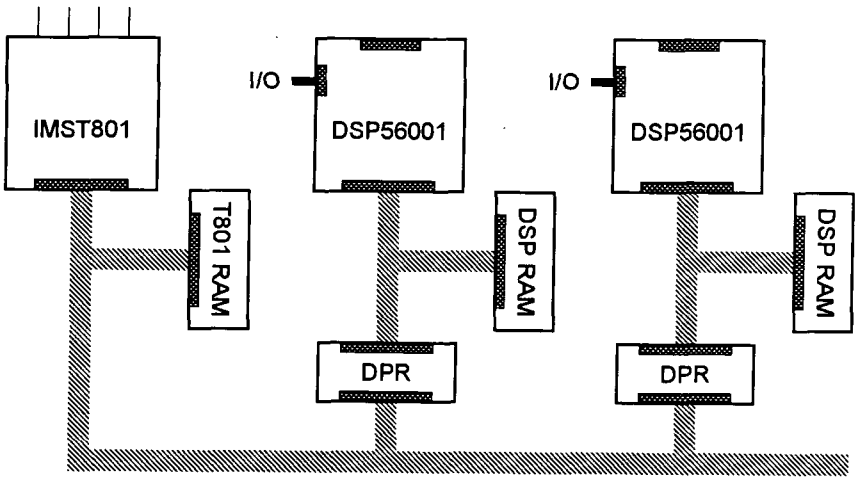

Figure 6-7: Multi-bus hybrid M-P architecture.

modulo-addressing across the whole DPR.[1] An address-decode PAL allows half of the DPR to be mapped into P-space, so that the T801 can place object code directly into P.

The T801 can access a single address space of 1 Gwords, with 1 Kword placed on-chip. 8 Kwords of local memory is contiguous with the on-chip RAM, allowing internal program and workspace areas to overflow into external memory. Alternative DPR memory-maps may be implemented by replacing the transputer address-decode daughter-board.

6.4.2 Message-Passing

Although the T801 itself implements store-and-forward IPC, the compute PEs in Figure 6-7 can utilise a pseudo-wormhole scheme as they do not handle messages for other PEs.

6.4.2.1 Semaphore Protocol

As the transputer is designed to perform IPC over serial links in a CSP context, the EMI does not support an indivisible RWM. Consequently, a modified semaphore scheme is required to avoid the problems discussed in Section 6.1.1. Previously, the semaphore indicated if the associated channel is locked by one of, possibly, several processors.

In *HYMIPS*, DPR channels are shared between two processors allowing a different semaphore protocol to be implemented. Rather than flagging whether or not the channel is

1. See the analysis of programmable DSP addressing modes in Section 4.3.4.

locked, the token determines which processor has exclusive access rights. Together with the DPR on-chip arbitration logic, this modified protocol ensures data-set integrity.

6.4.2.2 Inter-Node Communication

Inter-node communication is achieved using transputer links. With a bi-directional bandwidth of 2.35 Mbytes/s, each link supports 12 digital audio channels. One link can be reserved for broadcasting control data to the DSPs via a IMSC011 link adaptor [Inmos Ltd., 1989]. In this configuration each transputer can support up to 36 inter-node audio channels.

Intra-node communication is achieved through the non-multiplexed EMI of the T801. Each DSP possesses an exclusive block of DPR: the other port of each DPR is mapped into the transputer's EMI. As a 25 MHz T801 transputer can transfer data over its non-multiplexed EMI at a rate of 12.5 Mwords/s, providing 130 digital audio paths within the *HYMIPS* node.

6.4.2.3 Intra-Node Reconfiguration

Although the physical topology of the *HYMIPS* node is fixed the logical topology is not, however. As the T801 code determines intra-node IPC, the organisation of compute PEs may be altered dynamically to give a variety of *virtual* topologies. An aliased memory-map is included in the transputer address-decode PAL to enhance broadcast IPC performance.

One DSP56001 memory mapping mode addresses half of the DPR as P-space, allowing the T801 to down-load ASP tasks to the DSP. Object kernels may be stored in local transputer memory or read from the host file-system via a link. For efficiency reasons, the DSP must then move the object code into local DSP RAM or, preferably, internal P-space.

6.4.3 Performance

The computation performance of *HYMIPS* is limited by intra- and inter-node IPC bandwidth. Two message-passing schemes were proposed: orthogonal and pipelined. Using an orthogonal IPC harness, IPC channels are transferred over transputer links. As the T801 has four bi-directional links, this limits the number of.DSPs that may be serviced in any IPC cycle.

In the pipelined harness, only one I/P channel and one O/P channel are serviced directly by hardware links: all other IPC is intra-node. With 4 bi-directional links and an IPC

channel of 256 words, Gould has shown that *HYMIPS* node can support 17 DSP56001s per node with zero IPC latency, giving a total node performance of 175 MIPS [Gould, 1992].

6.4.4 SP-DIF Interfaces

Motorola's ADS56016 EVB demonstrates the ease with which data-domain converters can be assembled. To use *HYMIPS* in an all-digital audio path requires direct digital audio I/O.

6.4.4.1 Motorola DSP56001 PORT C

The nine I/O pins (PC0-PC8) of the DSP56001 PORT C support serial communications (SCI) and synchronous serial (SSI) interfaces. PC0-PC2 can be configured as specific SCI pins or for general-purpose I/O. To interface with Motorola SPI peripherals, the SSI has three pins dedicated to transmit data (STD), receive data (SRD) and a bi-directional serial clock (SCK).

Three other pins — SC0, SC1 and SC2 — may also be used to control bits in the SSI status register. In synchronous clock modes, SC0 and SC1 can addresses multiple peripheral devices. SC2 supports frame sync for both transmitter and receiver [Motorola Corp., 1989]. In *HYMIPS*, SC0 and SC2 have been mapped onto LR and WC as shown in Figure 6-6.

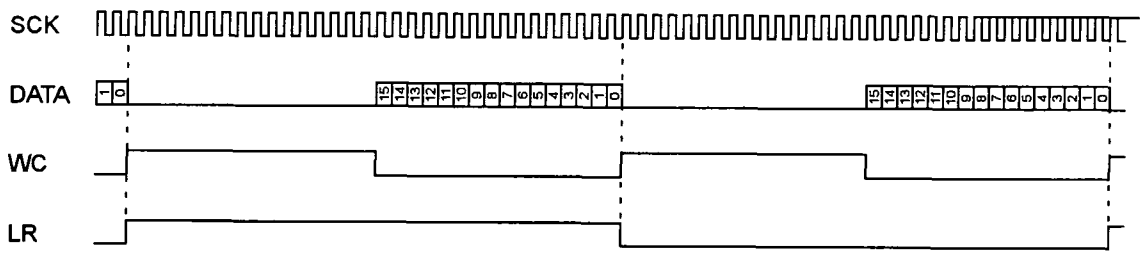

Figure 6-8: Timing diagram for *HYMIPS* SP-DIF interfaces.

6.4.4.2 SSI Programming Model

From a coding perspective, the SSI can be viewed as two 16-bit control registers (CRA, CRB), one read-only status register (SSISR), a read-only receive register (RX), and a write-only transmit register (TX). CRA controls the clock generator and frame sync rates and word length. CRB controls the multifunction pins, interrupt enables, and SSI mode (see Figure 6-9).

SSISR is an 8-bit read-only status register used to interrogate the SSI status and serial input flags. RX register accepts 24-bit data from SRD when the receive shift register becomes

```
; ---------------------------------------------------------------------------
; Initialisation macro for HSR/HST SP-DIF peripherals
; ---------------------------------------------------------------------------
SPDIF_INI \
      MACRO
      MSG    'spdif_ini...'

      ; clear bus control register, zero wait-states
      movep  #$000000,x:BCR

      ; set SSI RX interrupt priority level
      movep  #$003000,x:IPR

      ; set continuous external clock, 16 bits word-length, word frame sync
      movep  #$004000,x:SSI_CRA

      ; enable SSI RX interrupt, RX and TX, synchronous mode
      movep  #$00B200,x:SSI_CRB

      ; enable port C pins PC3-PC8 as SSI
      movep  #$0001FF,x:PCC

      MSG    'spdif_ini OK'
      ENDM
; ---------------------------------------------------------------------------
```

Figure 6-9: Initialisation macro for *HYMIPS* SP-DIF peripherals.

full. If the associated interrupt is enabled, the DSP56001 CPU is interrupted as Figure 6-7. Data written to TX is automatically transferred to STD via the transmit shift register.

```
; ---------------------------------------------------------------------------
; Interrupt service for HSR/HST SP-DIF peripherals: normally mapped to p:$000C
; ---------------------------------------------------------------------------
SPDIF_ISR \
      MACRO  lin,rin,acc,lout,rout
      MSG    'spdif_isr...'

      ; move sample from SSI receive register
      movep  x:SSI_RX,acc

      ; test SC0 for left/right channel
      btst   #0,x:SSI_SR

      .IF <CS>
            ; transfer left channel
            move   acc,x:lin
            movep  x:lout,x:SSI_TX
      .ELSE
            ; transfer right channel
            move   acc,y:rin
            movep  y:rout,x:SSI_TX
      .ENDI

      ; return from interrupt service
      rti

      MSG    'spdif_isr OK'
      ENDM
; ---------------------------------------------------------------------------
```

Figure 6-10: Interrupt service routine for *HYMIPS* SP-DIF peripherals.

6.4.4.3 *HYMIPS* SP-DIF Receiver

The YM3623B, developed by Nippon Gakki, was used in conjunction with TTL logic to implement a SP-DIF receiver (HSR) for *HYMIPS*. In the absence of optic fibre or coaxial SP-DIF, the internal PPL self-oscillates using the external crystal oscillator. LR indicates to which channel the current sample belongs, in sync with the MSB-first audio data on SRD.

If a parity error is detected in this design, audio data preceding the error is re-output. Copy disabled (DIS/CPY), error status (ERR), de-emphasis (DEF), source (CD/DAT) and sampling frequency are displayed on front-panel LEDs. As illustrated in Figure 6-8, the error status is also output on SC1. Table 6-4 lists the components identified in the HSR schematic.

Component Type	Schematic ID	Value
capacitor	C1	4.7 nF
	C2-C3	10 pF
	C4	220 nF
	C5	47 μF
	C6	100 μF
diode	D1	IN4140
inductor	L1	101 μH
resistor	R1-R2	4.7 kΩ
	R3	150 Ω
	R4	1 MΩ
	R5	1 kΩ
	R6	300 Ω
	R7-R14	470 Ω
	R15	21 kΩ
crystal oscillator	X1	18.43 MHz
zenner diode	Z1	5.0 V

Table 6-2: Component list for HSR schematic.

6.4.4.4 *HYMIPS* SP-DIF Transmitter

As shown in the schematic of Figure 6-9, the HST board is based on the Yamaha YM3613B. A built-in PLL synchronises with the sampling frequency of the digital audio input to this

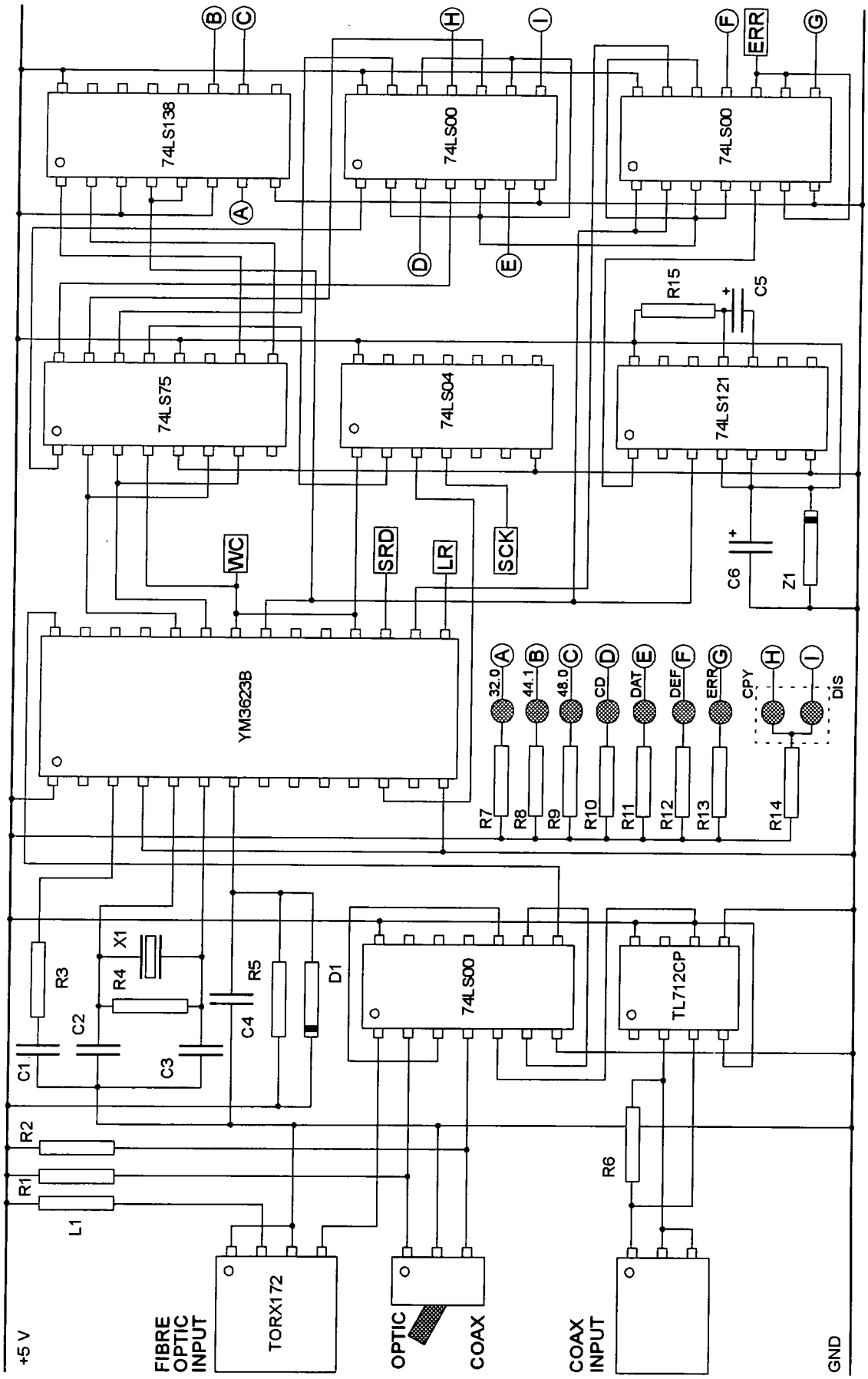

Figure 6-11: Schematic diagram of HSR SP-DIF peripheral.

24-pin CMOS DIP. Internal timing signals are generated by an external 16.93 MHz crystal connected across pins 3 (XI) and 4 (XO). Serial audio data on STD is assumed to be MSB first.

Although this design permits 'P-W' sequence data to be inserted, the overhead of sub-code information was discarded in *HYMIPS*. Both optic fibre and coaxial outputs are driven at the same time, the latter via a VC54820 line transformer. A front panel LED indicates the presence of data errors. Table 6-4 lists the values of components identified in Figure 6-9.

Component Type	Schematic ID	Value
capacitor	C1-C2	10 pF
	C3	100 μF
	C4	10 μF
	C5	220 pF
	C6	100 nF
	C7	10 nF
	C8	47 μF
resistor	R1	1 MΩ
	R2	470 Ω
	R3	2.2 kΩ
	R4	75 Ω
	R5	21 kΩ
crystal oscillator	X1	16.93 MHz
Zenner diode	Z1	5.0 V

Table 6-3: Component list for HST schematic.

6.5 Conclusions

Equations (6-1) and (6-2) show that IPC latency for message-passing schemes is dependent on the channel bandwidth, distance and message size. Compared to store-and-forward, wormhole routing greatly reduces IPC latency as delay is no longer sensitive to the distance involved in message transport. From Equation (6-7), the IPC latency in a 2-D mesh may be reduced up to $O(\sqrt{n})$ times over a binary hypercube if constant bisection bandwidth can be maintained.

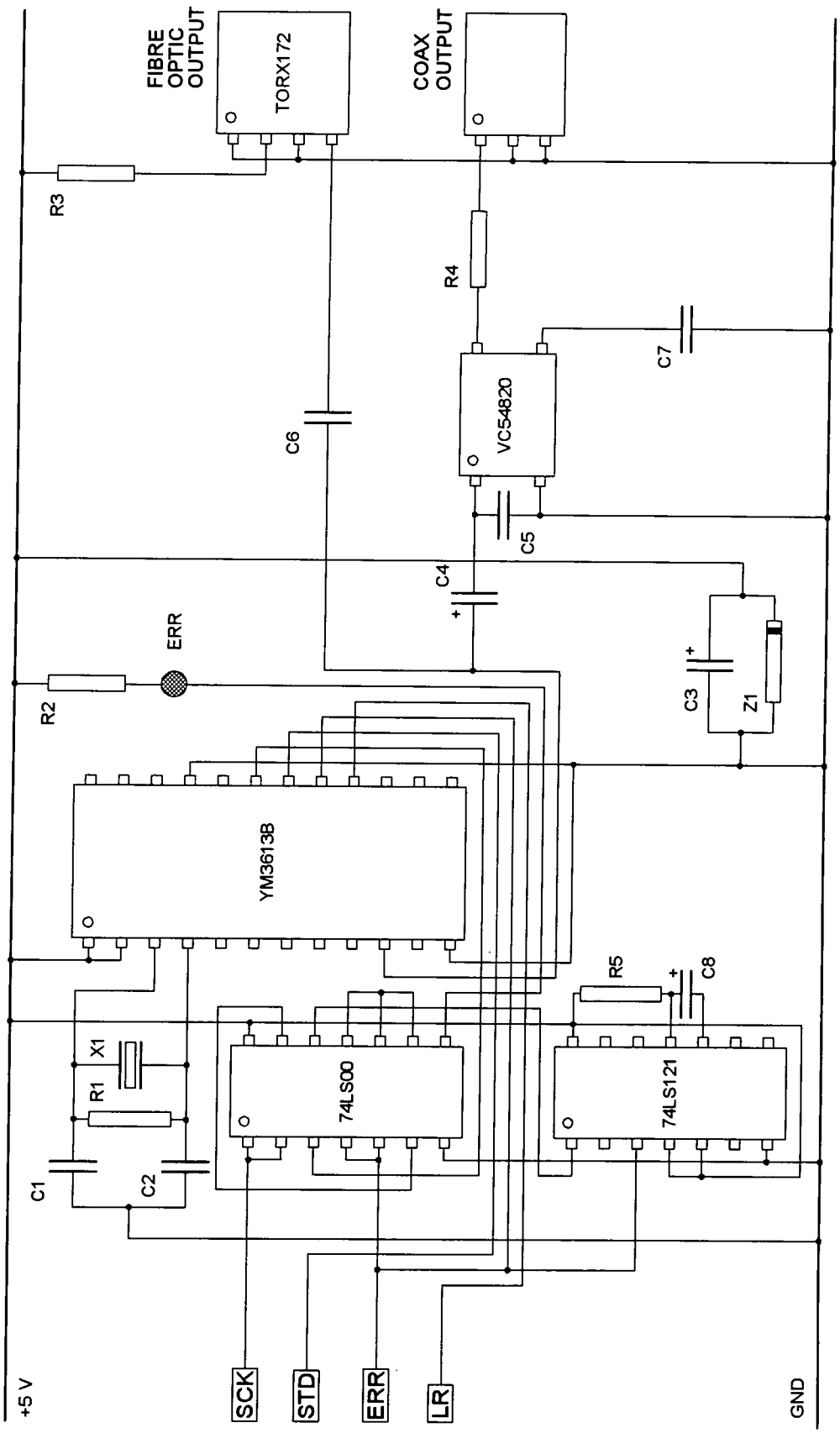

Figure 6-12: Schematic diagram of HST SP-DIF peripheral.

Fewer links contribute to the bisection, which permits each channel to be made wider. Throughput is bounded, however, by requiring more channels to cross the bisection for higher dimensions. Figure 6-4 reveals that there is a 21.26% advantage in reconfiguring an 80-processor binary hypercube as a 2-ary x-tree when node bandwidth is constant. For chordal-ring and 2-D mesh latency is reduced by 53.67% and 37.41% respectively for constant n.

Assuming constant PE bandwidth, IPC latency can be reduced by a up to a factor of 2 if a higher-dimensional n-processor hypercube is reconfigured as a 2-D mesh. X-tree topologies do not perform especially well under these conditions since 2 connections are required per processor to interconnect the ring at each level. A simple 3-ary tree gives the same wormhole latency improvement over a binary hypercube as a chordal-ring when $k \ll K$.

Each *HYMIPS* PE occupies its own PCB with boards connected via a backplane bus. An IMSC011 link adaptor is provided on the T801 card to drive the DSP HOST ports with control parameters via link 0. All links and the C011 are accessible via the front panel. DSP56001 boards incorporate a self-overwriting boot EPROM, DPR and a RS232/TTL converter for host access to the SCI. SSI and HOST ports are also available on the front panel.

As this is not a shared-bus M-P in the classical sense, performance is not limited by bus contention. Shared DPR and a modified semaphore protocol allow pseudo-wormhole messaging to overlap with computation. A *HYMIPS* node has been constructed with 2 compute PEs, together with HSR and HST SP-DIF peripherals. Independent testing confirms that computation performance aligns closely with theoretical predictions in Section 6.4.3.[1]

As the comprehensive design notes intimate, *HYMIPS* offers scalability both in terms of the number of compute PEs per node and the number of nodes. Topologies which require a node valency greater than 4 are possible with the addition of hardware switches such as the IMSC004 link crossbar. For further information, the interested reader should consult the paper presented by the author at ICASSP[2] and included in this thesis as Appendix F [Linton, 1991].

1. In that each node can support up to 17 DSP56001s with zero latency.
2. IEEE *International Conference on Acoustics Speech and Signal Processing.*

6.6 References

[Athas, 1988] Athas W C and Seitz C L: "Multicomputers: Message-Passing Concurrent Computers", *IEEE Computer*, Vol. 21, No. 8, pp. 9-24, August 1988.

[Dijkstra, 1965] Dijkstra E W: *Selected Writings in Computing*, Addison Wesley, 1982, ISBN 0-387-90652-5.

[Fleckenstein, 1992] Fleckenstein C J (et al.): "Multiprocessing", from *Advances in Computers*, Academic Press, 1992, ISBN 0-12-012135-2.

[Gould, 1992] Gould G L: *Digital Signal Conditioning on Multiprocessor Systems*, PhD Thesis, University of Durham, BLDSC No. D172306, 1992.

[IDT Inc., 1991] IDT: *Specialised Memories Databook*, Integrated Device Technology Incorporated, 1991.

[Inmos Ltd., 1989] Inmos Ltd.: *The Transputer Databook*, 2nd edition, Inmos Limited, 1989.

[Linton, 1991] Linton K N (et al.): "Real-Time Multi-Channel Digital Audio Processing: Scalable Parallel Architectures and Taskforce Scheduling Strategies", *IEEE ICASSP '91*, Toronto, Canada, May 1991.

[Motorola Corp., 1989] Motorola Corp.: *DSP56000/DSP56001 Digital Signal Processor User's Manual*, Rev. 2, Motorola Corporation, 1989.

[Snell, 1989] Snell J M: "Multiprocessor DSP Architectures and Implications for Software", *Proceedings of the AES 7th International Conference*, Toronto, Canada, May 1989.

[Tanenbaum, 1981] Tanenbaum, A S: *Computer Networks*, 2nd Edition, Prentice-Hall Inc., Englewood Cliffs, NJ, 1981, ISBN 013-166836-6.

Chapter 7

Multi-Processor Models

On M-P architectures like *HYMIPS*, load balancing tries to distribute tasks evenly whereas minimising IPC tends to assign the whole taskforce to one PE. As a conflict exists between these two aims, a compromise must be made to obtain good performance for a given DMC.

7.1 DMC Task Scheduling

A critical aspect of DMC design, on which the third part of this thesis focuses, is the development of algorithms for scheduling partially ordered tasks on multi-DSP topologies.

7.1.1 Introduction

The basic digital console scheduling (DCS) model consists of a *taskforce* of m tasks, $T = \{T_1, \ldots, T_m\}$, and a M-P of n identical PEs, $P = \{P_1, \ldots, P_n\}$. With each task T_i is associated the number of instruction cycles to execute it. A *scheduling algorithm* allocates T onto P, ordering the execution of tasks. No PE can execute more than one task at any time.

A DMC can be represented by a *directed acyclic graph* (DAG) where each open circle represents a processing task T_i and each arc a *precedence constraint.*[1] As shown in Figure 7-1, DCS can be visualised as a graph-matching exercise. If $m > n$, at least two tasks will be assigned to the same PE. However, multiple tasks may be co-resident even if $m < n$.

7.1.2 Scheduling Algorithms

A scheduling algorithm is said to solve a problem if it can be applied to any instance and is guaranteed to always produce a *feasible* solution. A feasible schedule is an assignment which does not violate any resource, precedence or preemption constraint. The degree of optimality, or cost, may be expressed formally in terms of an *objective function* [Hu, 1961].

1. As arcs are always assumed to be directed toward the exit node, they are not explicitly drawn.

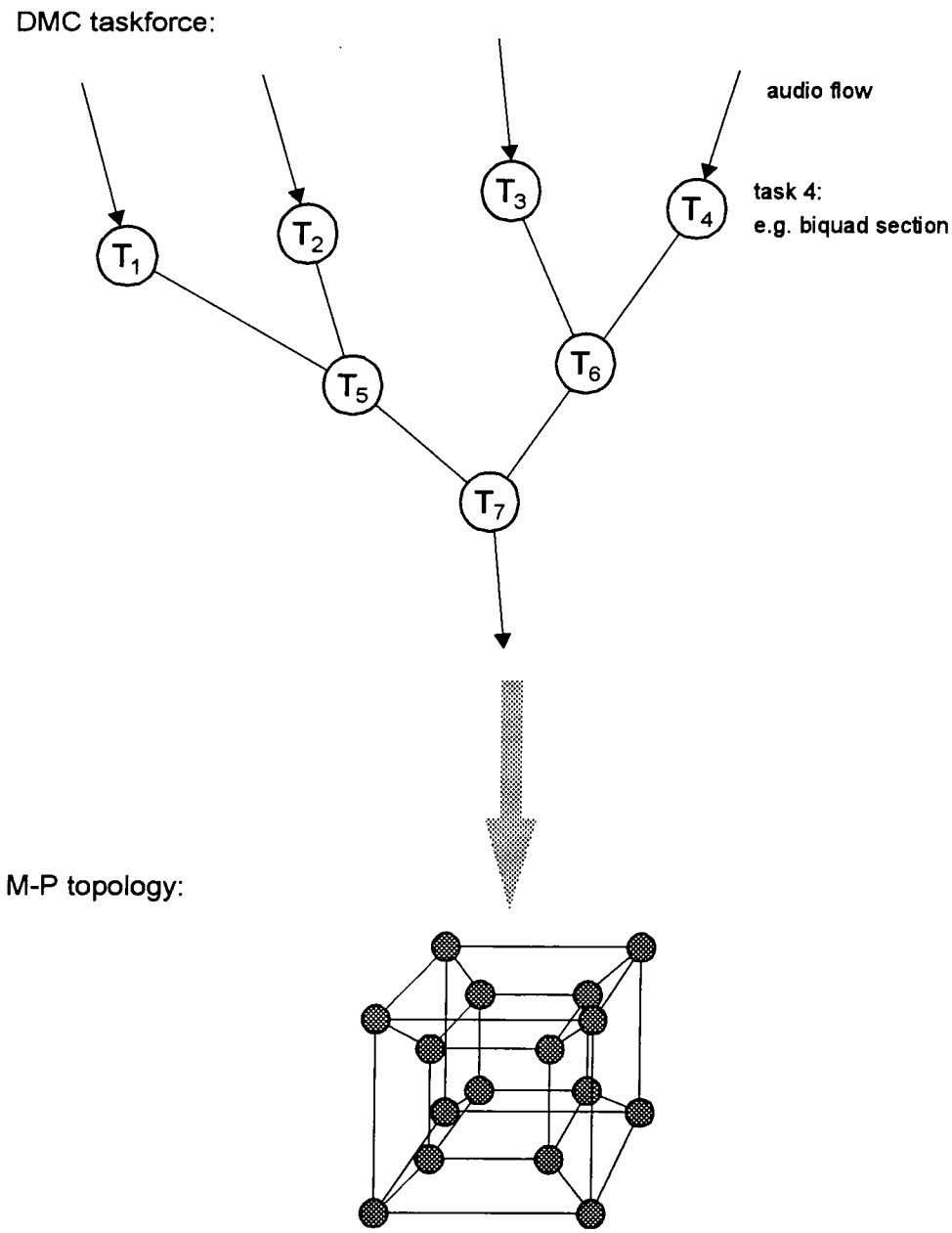

Figure 7-1: Illustration of the DMC task scheduling problem.

Given that a DMC taskforce is periodic, processing real-time sample trains, schedule length and accumulated latency are chosen as objective functions. In general, the most 'efficient' algorithm is required. In the context of DCS, the notion of efficiency primarily involves execution time as this factor determines if an algorithm is useful in practice.

7.1.3 Algorithm Constraints

By inspecting Figure 7-1, several pertinent observations can be made. Most notably, it is apparent that a scheduling algorithm is restricted by diverse technological constraints.

7.1.3.1 Precedence

Individual tasks within a taskforce can be related to each other in a number of different ways. In some instances, it is possible for all tasks to be independent; in others, ordering restrictions may be represented by a tree. Assuming that tasks can be executed in arbitrary order is not justified for DCS. Audio mixing requires a general precedence structure — 'a forest of trees'.

In DMCs, one task must be completed before another can begin execution. This type of constraint is incorporated into the DCS model as follows. Directed arcs between tasks in Figure 7-1 imply that a partial ordering, or precedence relation <, exists between tasks. Thus, if $T_i < T_j$, the execution of task T_i must be completed before T_j can be initiated.

7.1.3.2 Task Preemption

In non-preemptive schedules, a processor assigned to a task is dedicated to that task until it is completed. In general, preemptive disciplines generate better schedules than those generated by non-preemptive strategies.[1] Nevertheless, certain penalties such as task switching overhead are incurred in preemptive schemes. As, by construction, atomic ASP tasks cannot be further partitioned, preemptive schedules are not considered further in this thesis.

7.1.3.3 Cycles and Conditional Tasks

The taskforce shown in Figure 7-1 is *acyclic*. A loop would prevent static scheduling, as the loop condition cannot be resolved until run-time. Most published work on scheduling ignore the difficulties presented by loops. However, cycles can be removed by either loop unrolling or by bisecting cycles across adjacent sample periods as depicted in Figure 7-2.

Also, the taskforce of Figure 7-1 contains no *decision* tasks. A decision task is a data-dependent branch whose run-time outcome can affect the flow of control. While Lee (et al.) have developed techniques to deal with conditionals in general DSP, the requirement for such constructs is of minimal importance in real-time DMC computation engines [Lee, 1987].

7.1.3.4 Processor Idleness & Processing Lag

Performance metrics can often be improved by deliberately idling a PE, at the expense of computational complexity. In fact, enumerative algorithms are only guaranteed to be optimal

1. For example, consider 2 processors and a taskforce of 3 independent tasks each requiring 2 instruction cycles.

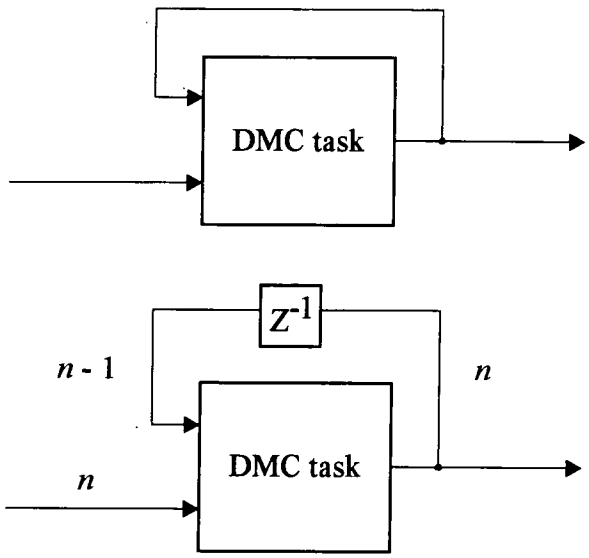

Figure 7-2: Illustration of loop removal by sample period bisection.

if the option of idling is permitted instead of scheduling a ready task. In 'greedy' DCS, no idle periods are inserted and a pending task is begun as soon as a PE is available.

Most important for real-time ASP applications, the DCS algorithm must ensure that accumulated processing lag — the result of processor pipelining — is not apparent. Experience indicates that all input-to-output delays through the DMC should be less than 1 ms (see Section 4.2.3) to facilitate real-time procedures such as overdubbing and track-bouncing.

7.1.4 Static vs. Dynamic Scheduling

There are two primary approaches to parallel processing and task scheduling on a M-P architecture. In the *static* approach, all parallel tasks in the problem taskforce are known before-hand. Consequently the taskforce can be statically mapped onto the processors at compile-time and remain fixed throughout run-time. This paradigm corresponds naturally to an DMC software design consisting of a network of communicating tasks.

In the *dynamic* approach, ASP computation is defined by a dynamically created taskforce and allocation decisions must be postponed until run-time, incurring substantial overheads. In DMC systems, the set of DSP algorithms and their resource requirements are known in advance. Consequently, the taskforce can be efficiently distributed beforehand, and the overheads associated with dynamic allocation eliminated.

7.1.5 Model Operation

With the foundation laid for DCS in terms of constraints and relationship to compile- and run-time, it is appropriate to consider how a DCS algorithm may fulfil the allocation role.

7.1.5.1 Input Processing

The operation of the DCS model begins at time $t = 0$ with all n PEs scanning the taskforce in search of *ready* tasks. A task is said to be ready for execution if all its predecessors have been completed and no processor has begun to execute it. If two or more processors compete for a task T_i; $1 \le i \le m$, T_i is assigned to the processor P_j; $1 \le j \le n$ with the lowest index j.

When a processor cannot find a ready task, it stops scanning and becomes idle until a task is completed. As this task might be a predecessor of other tasks, whenever a task is completed, all idle processors instantaneously begin to scan the taskforce. The schedule ends when the final task is completed: following convention, the finishing time is denoted by ω.

7.1.5.2 Output Processing

The run-time of the scheduling algorithm employed provides a second important performance criteria. In order to fully compare schedules obtained for various problem instances, a further performance metric is defined. The efficiency of a processor P_j can be defined as total active processing time of P_j divided by the schedule finishing time ω [Coffman, 1976].

Gantt charts are used to represent the timing of DMC taskforce schedules. Each task is represented by a time-space span whose length corresponds to its execution time. A horizontal row of time-space spans represents the sequence of tasks executed by a processor. Cross-hatched regions indicate periods during which the corresponding processor is idle.

7.1.6 Extensions

In this section, some extensions to the basic constraints discussed above are considered. Both additional resource constraints and heterogeneous M-Ps are directly applicable to DCS.

7.1.6.1 Additional Resource Constraints

Most effort to date has assumed the unlimited availability of secondary resources. ASP tasks require more resources than just a compute PE, however. Most references assume that

sufficient memory is available to contain the program and data of assigned tasks. Of course memory is not the only secondary resource: another is the availability of SP-DIF I/O ports.

As might be expected, worst-case behaviour degrades further as the number of resources is increased. Interestingly, CPM still guarantees the same worse-case bound for one additional resource and $m > n$ [Garey, 1978].[1] A related observation is that investing further effort in the preparation of priority lists can lead to better results [Yao, 1974].

7.1.6.2 Heterogeneous Multi-Processors

Up till now, it has been assumed that all processors are identical. In *heterogeneous* M-P systems, a DSP can be replaced with a higher clock rate part or even a different processor altogether. This enhanced model has been studied from a theoretical performance-guarantee viewpoint by Gonzalez [Gonzalez, 1978]. The case where task times may vary arbitrarily from processor to processor has been treated, for independent tasks, by Bruno [Bruno, 1974].

7.1.6.3 Deadline-Driven Schedules

Manacher has considered the case in which terminal and non-terminal tasks require different completion times [Manacher, 1967]. Tasks with unequal execution times and precedence constraints are executed in a non-preemptive manner. Manacher's heuristic solution is a variation of the longest-path schedules considered in the next chapter. Multiple critical paths are defined for those tasks contained in paths that contain tasks with deadlines.

7.1.6.4 Job Periodicity

The majority of investigations deal with only a single execution of taskforce. Although the analysis required to generate near-optimal schedules can be significant, this is justified by the time saved during each of these executions. Consideration has also been given to the use of M-P in a *control environment*. Kafura has characterised an environment of this type by a set of tasks each of which has a known periodicity and processing time [Kafura, 1974].

1. Enough processors so that all tasks could be executed simultaneously if it were not for resource constraints.

7.2 Modelling M-P Speed-Up

A model of m tasks executing on n PEs is presented which is capable of explaining various speed-up characteristics. The model is then considered in the context of DMC and DCS.

7.2.1 Introduction

'Superlinear' speed-up has been used to describe a computation using an n-PE M-P which is more than n-times faster than the uni-processor (U-P) case. Often this term is inappropriate as $S(n)$ may be linear and yet still greater than n. In this thesis, the terms *superunitary*, *unitary*, and *subunitary* speed-up are used when $S(n) > n$, $S(n) = n$, and $S(n) < n$, respectively.

Superunitary (SU) speed-up is only possible when the total amount of work performed by n processors is *strictly less* than that performed by 1. Hypothesised upper bounds on $S(n)$ range from Minsky's rather pessimistic $O(\log n)$ to unbounded [Minsky, 1970]. Conversely, there have been reports of SU speed-up in real applications [Sanuinetti, 1986].

7.2.2 Preliminary Conditions

The total number of instructions executed when a taskforce is processed can be represented by w. Each w_i then indicates the number of times tasks of type i are executed. To determine ω, $r_i(n)$ weights the time required to execute one type-i task on a homogeneous n-PE M-P. If one PE executes while the other $n-1$ are idle in sequential mode, then $r_s(n) = nr_s(1)$.

For notational convenience, $r_s(1) = 1$. Separating the DMC taskforce into tasks which must execute sequentially and those with at least n-fold parallelism gives:

$$w = w_s + \sum_{i=1}^{m'} w_i \qquad \text{Eq. (7-1)}$$

where m' is the number of tasks that exhibit n-fold parallelism and w_s is the number of operations that must be performed sequentially. Then, $S(n) > n$ if and only if:

$$\sum_{i=1}^{m'} w_i (r_i(1) - r_i(n)) > (n-1) w_s r_s(1) \qquad \text{Eq. (7-2)}$$

7.2.3 Opportunities for SU Speed-Up

In this section, the model of Section 7.3.2 is used to investigate previously reported occurrences of SU speed-up behaviour in a variety of M-P environments.

7.2.3.1 Conventional Speed-Up

Equation (7-7) clearly states the conditions necessary for SU speed-up. If w_p and $c_p(n)$ represent the amount and run-time cost of work that exhibits n-fold parallelism then the argument against SU speed-up characteristics can be stated as follows:

$$w = w_s + w_p \qquad \text{Eq. (7-3)}$$

$$r_s(n) = nr_s(1) = n \qquad \text{Eq. (7-4)}$$

$$r_p(n) = 1 \qquad \text{Eq. (7-5)}$$

Equation (7-7) becomes $w_p(r_p(1) - r_p(n)) > (n-1)w_s$, and since the left-hand side is 0, SU speed-up is impossible. In fact, if w_s is positive, then the speed-up becomes:

$$S(n) = \frac{w_s + w_p}{w_s + \frac{w_p}{n}} \leq 1 + \frac{w_p}{w_s} \qquad \text{Eq. (7-6)}$$

and Amdahl's law is obtained, as illustrated in Equation (7-6) [Amdahl, 1967].

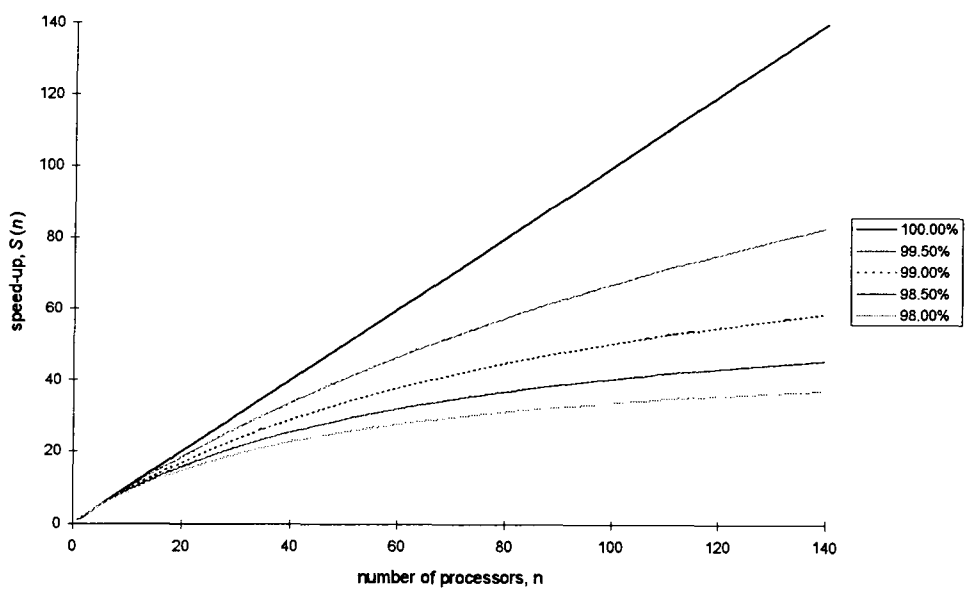

Figure 7-3: Illustration of conventional speed-up for various values of w_p.

7.2.3.2 Increasing Cache Size

SU speed-up can occur when each PE contains a local program or data cache. In this case, not only does the number of DSP ALU cores increase with n, but so does the total amount of cache memory. Whether or not a processor's local cache should be considered part of the processor depends on what is being measured. However, it is unreasonable to suppose that the cache size of a single DMC compute PE can be increased to match that of the equivalent M-P.

7.2.3.3 Hiding Latency

From Section 6.4, several methods can be used to hide IPC latency. Excess process-level parallelism can be achieved via time-slicing, reducing idle time at the expense of additional context-switching overhead. Alternatively, if the PE has DMA IPC capability, latency is concealed by utilising additional parallelism beyond that explicitly described. Since both schemes can be used on a U-P, they are optimisations rather than sources of true SU speed-up.

7.2.4 Scaled Speed-Up

In the previous sections, a given taskforce was executed on an extensible M-P architecture. A revised speed-up metric reflects cases where both n and m are increased [Helmbold, 1990].

7.2.4.1 Overview

Let $w_s(m, n)$ and $w_p(m, n)$ represent the number of sequential and parallel operations required to process a taskforce of size m on n-processors. If $r_s(n)$ and $r_p(n)$ are defined as before, the time taken to process a taskforce of size m on n PEs, $\omega(m, n)$, is then:

$$\omega(m, n) = \frac{w_s(m, n)r_s(n) + w_p(m, n)r_p(n)}{n} \qquad \text{Eq. (7-7)}$$

A reasonable assumption is that the number of instruction cycles required by the taskforce is independent of the number of processors, n. Therefore, $w_s(m, 1) = w_s(m, n)$ and $w_p(m, 1) = w_p(m, n)$, which can be abbreviated to $w_s(m)$ and $w_p(m)$, respectively.

7.2.4.2 Definition

Following convention, the scaled speed-up (SSU) on an n-PE system $S'(n)$ is defined as the ratio of the execution time of a taskforce of size n on a U-P to that on an n-processor M-P:

$$S'(n) = \frac{\omega(n, 1)}{\omega(n, n)} = \frac{w_s(n, 1)r_s(n) + w_p(n, 1)r_p(n)}{\frac{1}{n}(w_s(n, n)r_s(n) + w_p(n, n)r_p(n))} \qquad \text{Eq. (7-8)}$$

Assuming $c_p(n) = 1$ and $c_s(n) = n$ as before, Equation (7-13) simplifies to:

$$S'(n) = \frac{n(w_s(n) + w_p(n))}{nw_s(n) + w_p(n)} \qquad \text{Eq. (7-9)}$$

Since $w_s(n) + w_p(n)$ is always less than $nw_s(n) + w_p(n)$, provided $w_s(n) \neq 0$, unitary SSU is impossible. However, unitary SSU does occur in the limit when:

$$\frac{w_s(n) + w_p(n)}{nw_s(n) + w_p(n)} \geq \varepsilon \qquad \text{Eq. (7-10)}$$

for some positive constant ε. From above, the following must hold for linear subunitary SSU:

$$\frac{w_p(n)}{w_s(n)} \geq \frac{\varepsilon n - 1}{1 - \varepsilon} \qquad \text{Eq. (7-11)}$$

7.2.4.3 Problem Size & Execution Speed

One reason for considering scaled speed-up is the question of how large a problem can be solved within some fixed time, rather than how quickly a given problem can be solved. It is reasonable to assume that the total work grows no faster than $O(n)$, i.e. $w_s(n) + w_p(n) \leq \alpha n$. Under this assumption, $w_s(n)$ must be bounded by a constant to as shown below:

$$w_s(n) \leq \alpha n - w_p(n) = \alpha n - w_s(n)\frac{\varepsilon n - 1}{1 - \varepsilon} = \frac{(1 - \varepsilon)\alpha n}{\varepsilon(n - 1)} \qquad \text{Eq. (7-12)}$$

This analysis contrasts sharply with the corresponding non-scaled case, where $S(n)$ is always bounded by a constant. Once $w_p(n)$ and $w_s(n)$ are known, the problem size m can be fixed for (unscaled) speed-up and curves plotted for Equation (7-6) and Equation (7-9). As illustrated in Figure 7-4, the curves intersect where $S(n) = S'(n)$ or:

$$\frac{w_p}{w_s} = \frac{w_p(n)}{w_s(n)} \qquad \text{Eq. (7-13)}$$

7.3 Linear Processor Pipeline

Here, the linear processor pipeline (LPP) of Section 5.1.4 is re-examined in detail. Several performance metrics are defined in terms of this topology, which supports a linear DAG.

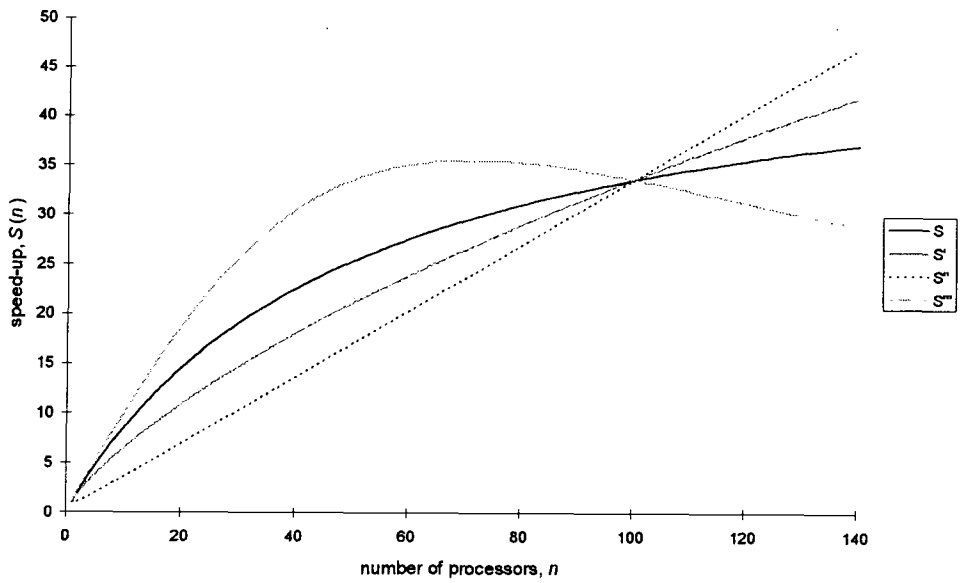

Figure 7-4: Illustration of conventional and scaled speed-up curves.[a]

a. For the conventional speed-up curve, S, $w_p = 98$ and $w_s = 2$. In scaled curves, S', S'', and S''', $w_s(n) = \sqrt{n}/5$, $w_s(n) = 2$, and $w_s(n) = n^2/(n+4900)$, respectively, with $w_p(n) = n - w_s(n)$ in each case.

7.3.1 Background

The sample period τ of a homogeneous processor pipeline is the reciprocal of the sampling frequency, i.e. $\tau = 1/f_s$. As shown in Figure 5-4, a space-time diagram illustrates the overlapped tasks in an LPP M-P architecture. Once the pipeline is full, maximum performance gives one output for each sample period independent of the number of stages k.

Ideally, an LPP with k stages can process n audio samples in T_k sample periods:

$$T_k = k + (n-1) \qquad \text{Eq. (7-14)}$$

given that k sample periods are required to fill the pipeline and process the first sample, and $n-1$ to complete the remaining $n-1$ samples. Processing the same number of samples on a non-pipelined architecture with identical f_s gives the number of sample periods T_1 as:

$$T_1 = nk \qquad \text{Eq. (7-15)}$$

7.3.2 Speed-up

Consequently, the speed-up of a k-stage LPP over an equivalent non-pipelined M-P is:

$$S(k) = \frac{T_1}{T_k} = \frac{nk}{k + (n-1)} \qquad \text{Eq. (7-16)}$$

It should be noted that unitary speed-up, $S(k) \rightarrow k$, can be achieved by a processor pipeline if $n \gg k$, as illustrated in Figure 7-3. This maximum is never fully achieved due to data dependencies between tasks, interrupt handlers, control branches and other factors.

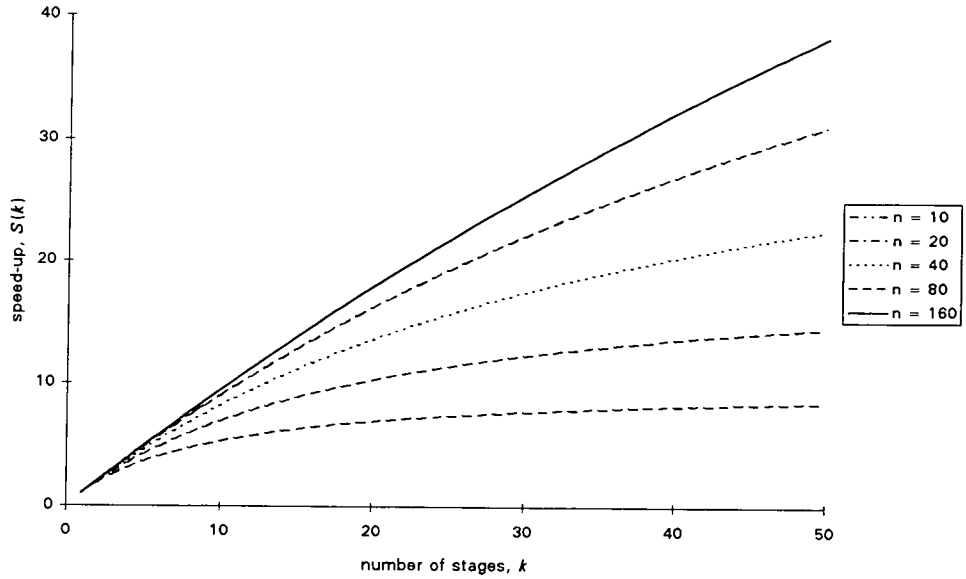

Figure 7-5: Speed-up characteristics for LPP M-Ps.

After describing speed-up in Equation (7-16), two related performance measures can be developed. First, the product of a time interval and a PE space in the space-time diagram of Figure 5-4 is defined as a time-space span. From the constraints of Section 7.1.3, a time-space span must be either busy or idle, but not both. This concept can be used to measure efficiency.

7.3.3 Efficiency

The efficiency of a linear processor pipeline can be defined as the ratio of time-space span required by the taskforce against the total time-space span, which equals the sum of all busy and idle time-space spans. If n, k, and τ are the number of samples, number of pipeline stages and sample period, respectively, the efficiency of a LPP architecture μ is:

$$\mu = \frac{nk\tau}{k\,[k\tau + (n-1)\,\tau]} = \frac{n}{k + (n-1)} \qquad \text{Eq. (7-17)}$$

Note that $\mu \rightarrow 1$ as $n \rightarrow \infty$, implying that the larger the number of samples flowing through the pipeline, the higher its efficiency, as Figure 7-4. From Equation (7-16) it is apparent that $\mu = S(k)/k$. Consequently, μ can be interpreted as the ratio of actual speed-up to the ideal unitary speed-up k. In steady-state with $n \gg k$, the efficiency tends to 1.

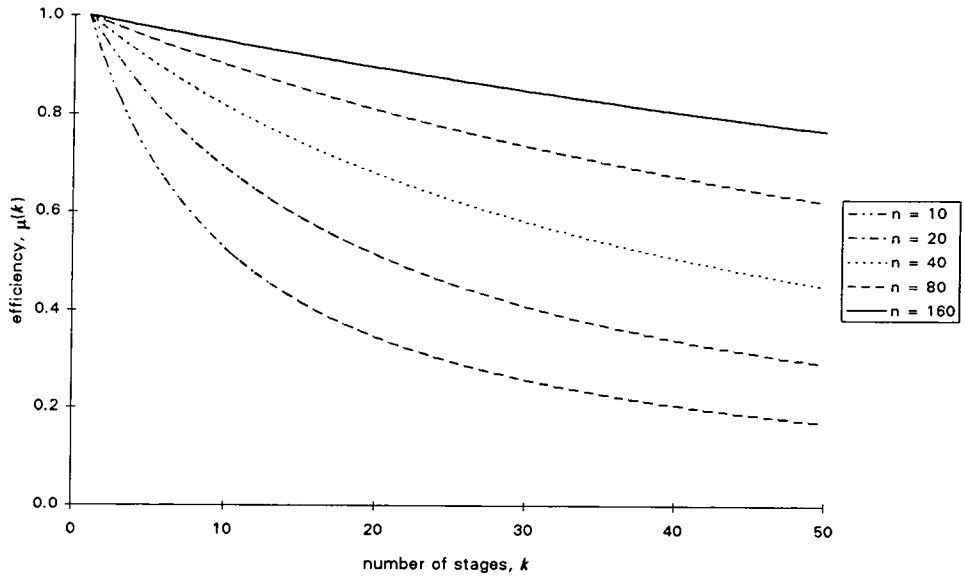

Figure 7-6: Efficiency characteristics for LPP M-Ps.

7.3.4 Throughput

Throughput is defined as the number of tasks that can be completed per unit time. This rate reflects the compute power of a given M-P architecture. In terms of efficiency μ and sample period τ the throughput for a linear processor pipeline can be defined as:

$$w = \frac{n}{k\tau + (n-1)\tau} = \frac{\mu}{\tau} \qquad \text{Eq. (7-18)}$$

where n is equals the total number of samples being processed during the observation period $k\tau + (n-1)\tau$. In the ideal case, $w = 1/\tau = f_s$ when $\mu \rightarrow 1$. Maximum throughput is therefore equal to the sampling frequency, corresponding to one output per sample period.

7.4 Task Granularity

M-P system architects have long debated the relative merits of fine and coarse granularity. Accordingly, several M-P performance models are developed with respect to this metric.

7.4.1 Introduction

In this section, it is assumed that each task executes in r instruction cycles, and that IPC takes c cycles when tasks are *not* co-resident. Various studies have shown that M-P performance depends strongly on r/c. This ratio expresses how much IPC overhead is incurred per unit of computation and, as such, is a measure of computation task granularity [Stone, 1989].

In coarse-grained parallelism, r/c is high, tasks are relatively large and IPC overhead can be amortised over many instruction cycles. Conversely, in fine-grained taskforces the overhead per unit computation r/c is relatively low. Many more PEs may be efficiently utilised than in coarse-grained schemes, but not without significant uplift in IPC overhead.

7.4.2 Non-Overlapped IPC

In DMCs, the central aim is to execute a taskforce consisting of m tasks efficiently on n processors. In this section, M-P architectures with non-overlapped IPC are considered.

7.4.2.1 Two-Processor Case

In order to minimise IPC overhead when using a dual-DSP M-P, all m tasks can be assigned to one DSP and the other ignored. Alternatively, tasks can be allocated across both in any combination. Total processing time $\omega\,(m, n, k)$ must then include activities such as IPC which incur time overhead. When k tasks are assigned to one PE, and $m - k$ to the other, then:

$$\omega(m, n, k) = r\max(m-k, k) + c\,(m-k)\,k \qquad \text{Eq. (7-19)}$$

Note that the second term models IPC transfers. One optimum solution, illustrated in Figure 7-7, is to assign all tasks to one processor if r/c is less than or equal to $m/2$. If the computation per unit communication exceeds this threshold, then the tasks should be split evenly as Figure 7-8. That is, either $k_{opt} = 0$ or m for $r/c \le m/2$, or $m/2$ otherwise.

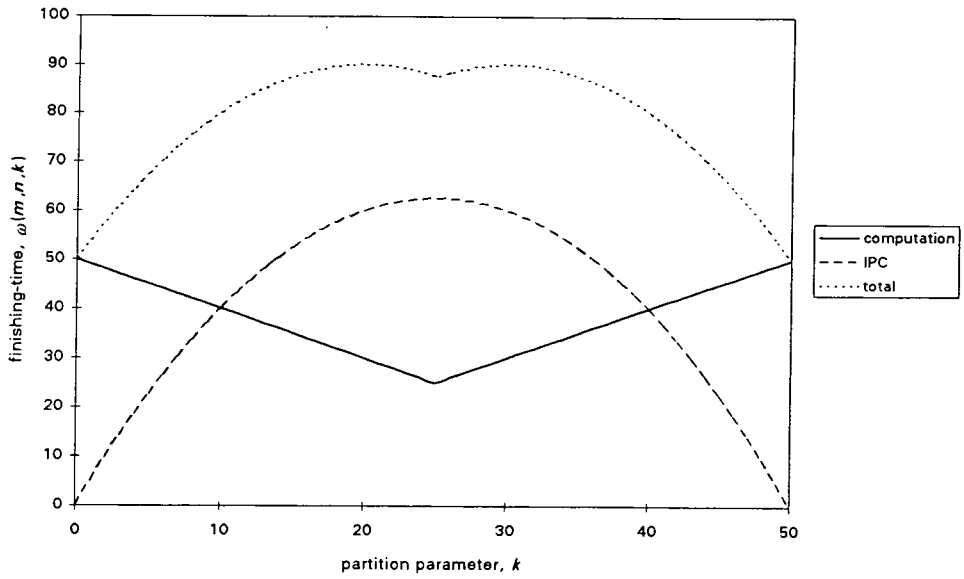

Figure 7-7: Non-overlapped IPC with 50 tasks, where $r/c = 10$: $k_{opt} = 0$ or m.

7.4.2.2 Extension to *n* Processors

In a M-P with n processors available for taskforce execution, Equation (7-19) becomes:

$$\begin{aligned}\omega(m, n, k) &= r\max(k_i) + \frac{c}{2}\sum_{i=1}^{n} k_i(m - k_i) \\ &= r\max(k_i) + \frac{c}{2}\left(m^2 - \sum_{i=1}^{n} k_i^2\right)\end{aligned} \qquad \text{Eq. (7-20)}$$

where k_i is the number of tasks assigned to the i-th processor.

From Section 7.4.2.1, the difference in processing costs of 'even'[1] distribution and uni-processor execution, $\Delta\omega$, is given by:

$$\Delta\omega(m, n, k) = \frac{rm}{n} + \frac{cm^2}{2} - \frac{cm^2}{2n} - rm \qquad \text{Eq. (7-21)}$$

where the first three terms are the cost of even allocation.

Ignoring values of m that are not exact multiples of n, Equation (7-21) solves for r/c:

$$\frac{r}{c} = \frac{m}{2} \qquad \text{Eq. (7-22)}$$

1. In this context, the term 'even' means that if m is a multiple of n, then each PE executes m/n tasks. Otherwise, all but one executes $\lceil m/n \rceil$, with one PE executing the remaining tasks.

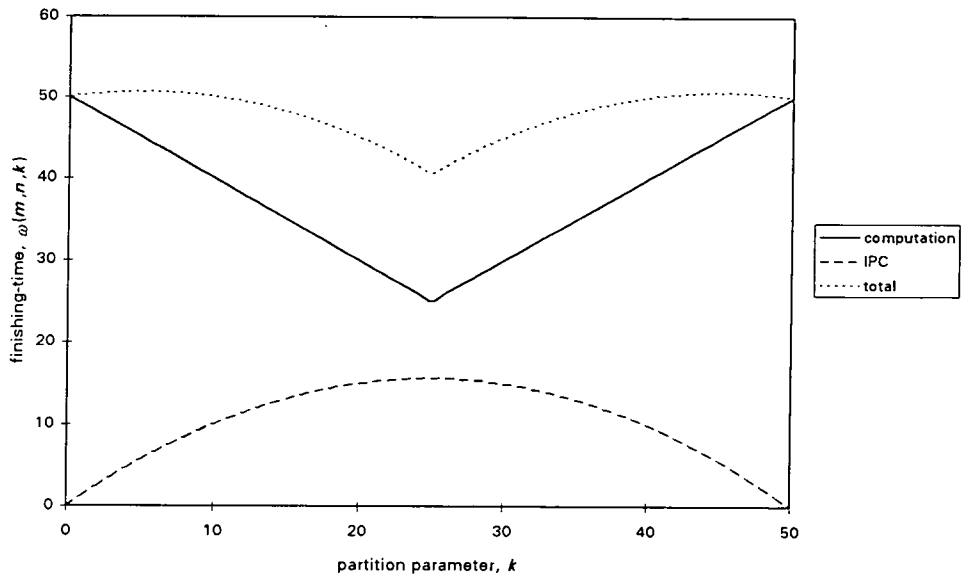

Figure 7-8: Non-overlapped IPC with 50 tasks, where $r/c = 40 : k_{opt} = m/2$.

If $r/c > m/2$ then even distribution will produce the best schedule. If, however, r/c is below or equals this threshold then as $n \rightarrow \infty$ no assignment produces a faster schedule than n.

7.4.2.3 Speed-Up

Control of overhead costs is absolutely essential for DCS to be successful. The overhead per unit computation r/c determines the point at which parallelism is cost-effective. Even when r/c is sufficiently high to warrant parallelism, the performance gain is diminished by the second term in Equation (7-20). The speed-up $S(n)$ is then:

$$S(n) = \frac{rm}{\left(\frac{rm}{n} + \frac{cm^2}{2} - \frac{cm^2}{2n}\right)} = \frac{\frac{rn}{c}}{\left(\frac{r}{c} + \frac{m(n-1)}{2}\right)} \qquad \text{Eq. (7-23)}$$

If m and n are small and r/c large, then $S(n) \propto n$. However, if the number of processors is increased such that $n \rightarrow \infty$, then $S(n) = 2r/cm$. Hence, $S(n)$ does not depend on n and approaches a constant asymptote as $n \rightarrow \infty$. Even though performance can be improved incrementally as n increases, diminishing returns defeat this M-P strategy.

7.4.3 Fully Overlapped IPC

In practice, IPC can be successfully overlapped with computation to minimise overhead penalties as described in the previous chapter. Such a M-P model is developed here.

7.4.3.1 Two-Processor Case

If overhead can be overlapped fully with execution costs then Equation (7-20) becomes:

$$\begin{aligned}\omega(m,n,k) &= \max\left(r\max(k_i), \frac{c}{2}\sum_{i=1}^{n} k_i(m-k_i)\right) \\ &= \max\left(r\max(k_i), \frac{c}{2}\left(m^2 - \sum_{i=1}^{n} k_i^2\right)\right)\end{aligned} \qquad \text{Eq. (7-24)}$$

While this model is perhaps over-optimistic, the design of *HYMIPS* (see Chapter 6) shows that it is in fact possible to fully overlap computation with IPC and other overhead.

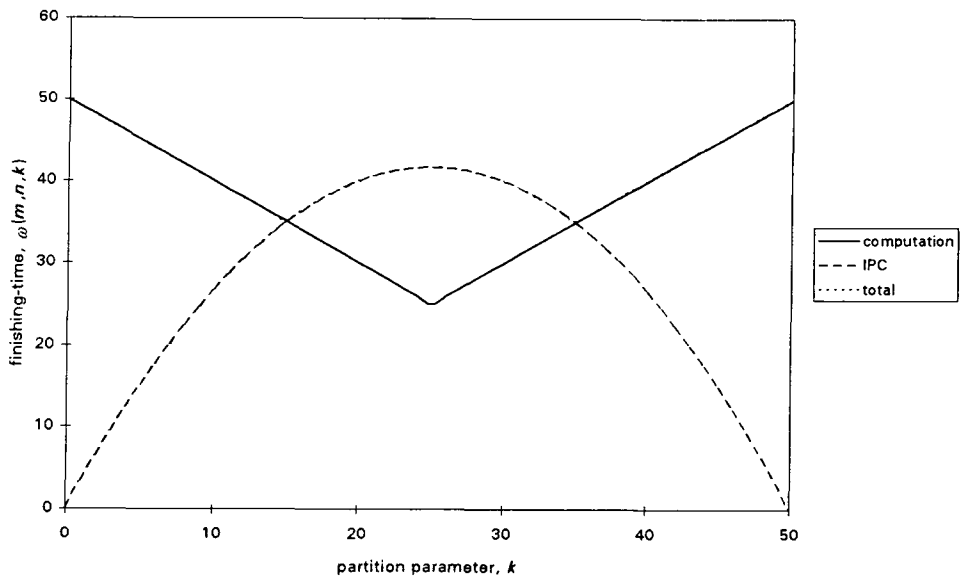

Figure 7-9: Fully overlapped IPC execution of 50 tasks, where $r/c = 15 : k_{opt} = 3m/10$.

The threshold occurs when the execution time is just long enough to completely mask the overhead incurred by IPC, as illustrated in Figure 7-9, i.e. $r(m-k) = c(m-k)k$, or:

$$k_{opt} = \frac{r}{c} \qquad \text{Eq. (7-25)}$$

where k is restricted to $1 \le k \le m/2$. $\omega(m,n,k)$ becomes $r(m-r/c)$, and the speed-up:

$$S(n) = \frac{1}{1 - \frac{r}{cm}} \qquad \text{Eq. (7-26)}$$

$S(n)$ lies between 1 and n and is again maximised when $r/c = m/2$. As $r/c \to 1$, $S(n) \to 1$ and the optimum distribution becomes more unbalanced. This M-P model also depends on r/c, but gives rather more optimistic performance predictions.

7.4.3.2 Extension to *n* Processors

For n processors and for any given maximum value of k_i, even distribution of tasks produces minimum IPC overhead. Hence, the best possible ω for fully overlapped IPC occurs when:

$$\frac{rm}{n} = \frac{cm^2}{2}\left(1 - \frac{1}{n}\right) \qquad \text{Eq. (7-27)}$$

which simplifies to:

$$\frac{r}{c} = \frac{m(n-1)}{2} \qquad \text{Eq. (7-28)}$$

For larger n, Equation (7-28) shows that n for minimum ω is given by:

$$n = \frac{2r}{cm} \qquad \text{Eq. (7-29)}$$

As m increases, the best strategy is to use increasingly fewer processors. For $n = 2$, $r/c = m/2$ which is consistent with previous findings. That n decreases with m is the result of IPC overhead increasing m-times faster than the total execution cost, ω.

7.4.4 Linear IPC Overhead

Assuming linear IPC overhead, IPC costs are proportional to n rather than the number of tasks assigned remotely. If each task communicates α-times with a task on each of the n PEs:

$$\omega(m, n, k) = r\max(k_i) + \alpha cn \qquad \text{Eq. (7-30)}$$

As only the first term depends on k, ω is mimimised by distributing the m tasks evenly.

As $n \to \infty$, the increase in αcn eventually becomes larger than the decrease in $r\max(k_i)$. Correspondingly, there is a maximum value of n for which performance increases:

$$\frac{d\omega(m, n, k)}{dn} = -\frac{rm}{n^2} + \alpha c \qquad \text{Eq. (7-31)}$$

which becomes zero when:

$$n = \sqrt{\frac{rm}{\alpha c}} \qquad \text{Eq. (7-32)}$$

This square-root ensures that the cost of incrementing n cannot be justified as $n \to \infty$.

7.4.5 Multiple Communication Links

A common assumption in the previous models is that IPC proceeds sequentially. It is perfectly possible, as discussed in Chapters 5 & 6, to replicate architectural features for IPC.[1]

7.4.5.1 Non-Overlapped IPC

If the number of communication links is allowed to increase with n to provide full interconnection, then communication operations can be overlapped. In so doing, c is no longer a constant but itself becomes a function of the number of processors, n. However, even with $O(n^2)$ links installed, no more than $O(n)$ concurrent IPC messages can be supported:

$$\begin{aligned} \omega(m, n, k) &= r\max(k_i) + \frac{c}{2n}\sum_{i=1}^{n} k_i(m - k_i) \\ &= r\max(k_i) + \frac{c}{2n}\left(m^2 - \sum_{i=1}^{n} k_i^2\right) \end{aligned} \qquad \text{Eq. (7-33)}$$

In Equation (7-33), it is assumed that a processor is either computing, communicating, or idle, and that the total cost of IPC decreases inversely with n. From Section 7.4.2, it is known that the previous equation is minimised by assigning tasks as evenly as possible. Under such a schedule, Equation (7-33) becomes:

$$\omega(m, n, k) = \frac{rm}{n} + \frac{cm^2}{2n}\left(1 - \frac{1}{n}\right) \qquad \text{Eq. (7-34)}$$

1. As found in architectures with a single common IPC channel, such as shared bus and simple ring topologies.

7.4.5.2 Threshold Analysis

Parallel execution is beneficial until $\omega(m, n, k)$ fails to decrease as $n \to \infty$, i.e. $\delta\omega = 0$.

$$\delta\omega(m, n, k) = \frac{rm + \frac{cm^2}{2}}{n(n+1)} - \frac{\frac{cm^2}{2}(2n+1)}{[n(n+1)]^2} = \left[r + \frac{cm}{2}\left(1 - \frac{2}{n}\right)\right]\frac{m}{n(n+1)} \qquad \text{Eq. (7-35)}$$

However, Equation (7-35) is positive for $n \geq 2$, and so $\omega(m, n, k)$ improves for nearly all n. Comparing U-P processing cost with that for an n-PE architecture gives the threshold:

$$\frac{r}{c} = \frac{m}{2n} \qquad \text{Eq. (7-36)}$$

In this case, the task granularity factor r/c and the number of processors, n, are inversely related: the larger n becomes, the smaller the task granularity required. As a result, this M-P is a failure at break-even. The performance provided by n processors is identical to that of one, yet real cost is increased by a factor of $O(n)$ processors and $O(n^2)$ links.

7.4.5.3 Fully Overlapped IPC

If IPC overhead can be fully overlapped with execution and ω minimised by assigning tasks as evenly as possible as before, then the finishing-time ω becomes:

$$\omega(m, n, k) = \max\left(\frac{rm}{n}, \frac{cm^2}{2n}\left(1 - \frac{1}{n}\right)\right) \qquad \text{Eq. (7-37)}$$

This minimum value of ω occurs when computation fully overlaps IPC overheads:

$$\frac{rm}{n} = \frac{cm^2}{2n^2}(n-1) \qquad \text{Eq. (7-38)}$$

corresponding to a computation to communication ratio of:

$$\frac{r}{c} = \frac{m}{2}\left(\frac{n-1}{n}\right) \approx \frac{m}{2} \qquad \text{Eq. (7-39)}$$

As r/c is virtually independent of n, it is only contingent on the size of DMC taskforce. Providing that the task granularity exceeds the threshold shown in Equation (7-39), this configuration is a success as $S(n) = n$. If, however, $r/c \leq m/2$, computation does not overlap IPC, so M-P performance is reduced as speed-up is constrained by m:

$$S(n) = \frac{2r}{cm}n \qquad \text{Eq. (7-40)}$$

7.5 Conclusions

The model developed here has sufficient structure to exhibit many of the difficulties encountered in DCS. Console taskforces are described by the number of tasks m and their resource requirements, with ordering restrictions represented by precedence relations. The M-P is specified by the number of processors n, their resources and the capacity of IPC links.

In addition, a DCS algorithm must observe all precedence relations, ensuring that no task can be started until all its predecessors are finished. Due to the indivisible nature of ASP kernels it must also be non-preemptive: a task cannot be interrupted until completion. Since parameters are known in advance, the DMC can be efficiently scheduled at compile-time.

Returning to the DMC context, the conventional speed-up metric described in Section 7.3.3.1 assumes that the amount of parallel and sequential work in the taskforce are fixed. If, however, this ratio changes (see Figure 7-4) as the underlying M-P *and* taskforce are scaled together SSU indicates how performance will vary for alternative DMC configurations.

If $w_p(n)/w_s(n)$ is constant, then conventional speed-up and SSU curves are identical. If, however, the ratio of n-fold parallel to sequential work increases with n, then $S(n) > S'(n)$ for $1 \le n \le m$ and $S(n) < S'(n)$ for $n > m$. Conversely, if $w_p(n)/w_s(n)$ decreases as n increases, then $S'(n)$ lies above $S(n)$ for $1 \le n \le m$ but below it when $n > m$.[1]

Conventional speed-up quantifies the relative performance of a DMC taskforce with a fixed ratio of parallel to serial code on a n-PE M-P to that of an identical U-P. In situations were the taskforce and underlying M-P are to be scaled, SSU provides a more accurate insight. Depending on how the ratio changes with n, SSU can exceed conventional speed-up.

As noted in Section 3.2.3.3, $k_{max} = 48$ for a LPP-based DMC. Equation (7-16) implies that as $n \to \infty$, $S(k) = k_{max}$, i.e. unitary, provided that no more than k_{max} PEs are required. In real-time $n \gg k$, and Equation (7-4) shows that efficiency $\mu \to 100\%$, provided that τ instructions are allocated to each PE. In this scenario, throughput w equals f_s.

1. In any case, the curves intersect at $n = 1$ and $n = m$.

Analysing the LPP shows that this approach to parallelising a DMC taskforce provides opportunities to achieve unitary speed-up and maximum processing efficiency with a throughput of one sample per sample period. This technique has been demonstrated within the *HYMIPS* architecture described in Chapter 6 and will appear again in the context of DCS.

In Figure 7-10, speed-up behaviour is plotted for M-Ps with a single communications channel. With non-overlapping IPC, $r/c > m/2$ ensures that even distribution is more effective than allocating all tasks to a single DSP. As the DMC is scaled speed-up improves incrementally, but reaches a constant asymptote of $2r/cm$ irrespective of other factors.

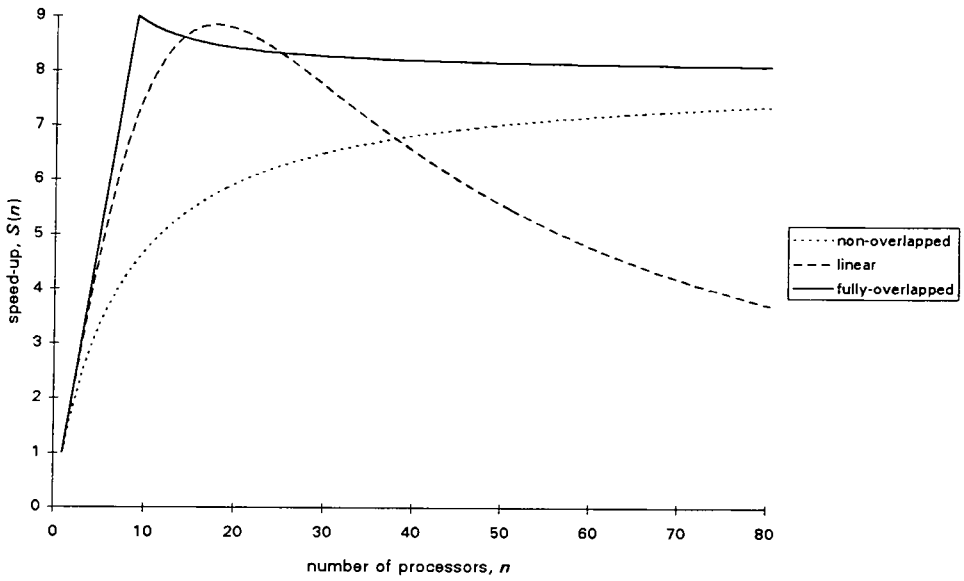

Figure 7-10: Speed-up characteristics with a single IPC link: $r/c = 200$, $m = 50$.

Fully-overlapping IPC with task execution gives unitary speed-up, $S(n) = n$, until IPC overhead overwhelms execution costs. Once the threshold of $n = 2r/cm + 1$ is reached, however, $S(n)$ tends towards the same constant asymptote. A DMC with linear IPC overhead performs well until the linear increase in IPC costs progressively limits this approach.

The effect of multiple IPC links on M-P performance are shown in Figure 7-11. With non-overlapped execution, Equation (7-36) confirms that for this case even distribution for any n is more effective than U-P allocation. Overlapping task execution with IPC is the ideal M-P configuration, giving unitary speed-up providing that task granularity exceeds $m/2$.

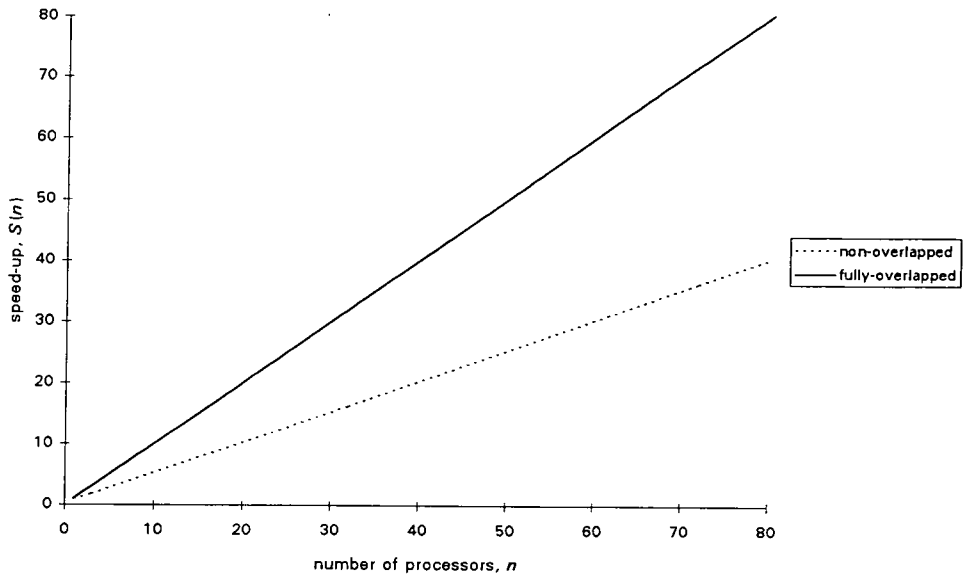

Figure 7-11: Speed-up characteristics with multiple IPC links: $r/c = 200$, $m = 400$.

A number of different M-P models have been investigated with respect to task granularity. In every case the role of r/c is the same: small ratios lead to higher IPC overhead. For maximum performance, granularity must be offset against this overhead. Unitary speed-up can be achieved by overlapping computation with multiple IPC links.

ω is totally dependent on IPC when run-time is smaller than IPC overhead. Consequently, it is essential that IPC overhead is restricted to be no greater than execution time. For each model, there is some maximum n that is cost-effective: this number depends on the M-P topology, on the underlying IPC mechanisms and the particular DMC taskforce.

7.6 References

[Amdahl, 1967] Amdahl G: "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", *AFIPS Computing Conference*, 1967.

[Bruno, 1974] Bruno J, Coffman E G and Sethi R: "Scheduling Independent Tasks to Reduce Mean Finishing Time", *Communications of the ACM*, Vol. 17, No. 7, pp. 382-387, July 1974.

[Coffman, 1976] Coffman E G (ed.): *Computer and Job-Shop Scheduling Theory*, Wiley & Sons, 1976, ISBN 0-471-16319-8.

[Garey, 1978] Garey M R, Graham R L and Johnson D S: "Performance Guarantees for Scheduling Algorithms", *Operations Research*, Vol. 26, No. 1, pp. 3-21, January 1978.

[Gonzalez, 1978] Gonzalez T and Sahni S: "Flowshop and Jobshop Schedules: Complexity and Approximation", *Operations Research*, Vol. 26, No. 1, pp. 36-52, January-February 1978.

[Helmbold, 1990] Helmbold D P and McDowell C E: "Modelling Speedup (n) Greater than n", *IEEE Transactions on Parallel and Distributed Computing*, Vol. 1, No. 2, pp. 250-256, April 1990.

[Hu, 1961] Hu T C: "Parallel Sequencing and Assembly Line Problems", *Operations Research*, Vol. 9, No. 6, pp. 841-848, December 1961.

[Kafura, 1974] Kafura D G and Shen V Y: "Scheduling Independent Processors with Different Storage Capacities", *ACM National Conference*, July 1974.

[Lee, 1987] Lee E A and Messerschmitt D G: "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", *IEEE Transactions on Computers*, Vol. C-36, No. 1, pp. 24-35, January 1987.

[Manacher, 1967] Manacher G K: "Production and Stabilisation of Real-Time Task Schedules", *Journal of the ACM*, Vol. 14, No. 3, pp. 439-465, July 1967.

[Minsky, 1972] Minsky M: "Form and Content in Computer Science", *Journal of the ACM*, Vol. 17, No. 2, pp. 197-215, April 1970.

[Sanuinetti, 1986] Sanuinetti J: "Performance of a Message-Based Multiprocessor", *IEEE Computer*, Vol. 19, No. 9, pp. 47-55, September 1986.

[Stone, 1989] Stone H S: *High-Performance Computer Architecture*, Addison-Wesley, 1987, ISBN 0-201-16802-2.

[Yao, 1974] Yao A C: "Scheduling Unit-Time Tasks with Limited Resources", *1974 Sagamore Computer Conference on Parallel Processing*, August 1974.

Chapter 8

Scheduling Strategies

Scheduling is one of the few areas of discrete optimisation that is motivated by practical considerations. Research in this broad area has been ongoing since the early 1960s. In this chapter, the author develops this discipline with specific regard to the DMC problem domain.

8.1 Combinatorial Optimisation

An overview of combinatorial optimisation (CO) is presented in this section. Emphasis is on the fundamental issues defining the intrinsic complexity of the DMC task scheduling problem.

8.1.1 Overview

A wide variety of important CO problems have emerged from such diverse areas as operations research and VLSI design. Solving such problems amounts to finding the *optimal* solution among an extremely large, but finite, number of alternatives. Optimisation algorithms can be used to find globally optimal solutions to a CO problem, often in a prohibitive amount of time.

Approximation algorithms yield solutions in an acceptable amount of time at the risk of sub-optimality. *General* algorithms are applicable to a wide variety of problems, whereas *tailored* algorithms rely on problem-specific information. Performance analysis concentrates on the quality of the final solution and run-time required for average and worst-case scenarios.

8.1.2 Definitions

Algorithm time requirements are conveniently expressed in terms of a single variable — the 'size' — which reflects the amount of input data needed to describe a problem. The *input length*, l, is defined as the number of symbols used to describe a problem instance in the chosen *encoding scheme*.[1] This measurement is used as a formal indication of instance size.

1. An encoding scheme maps problem instances onto strings of symbols which describe them.

Time complexity functions express algorithm time requirements in terms of the largest amount of time needed to solve problem instances of a given size. These functions are not well-defined until the encoding scheme is fixed. However, choices made here have little effect on the broad distinctions made in the theory of computational complexity [Garey, 1978].

8.1.3 Encoding Schemes

A taskforce can, for example, be described by listing all tasks and arcs, or by the taskforce adjacency matrix. Referring to Figure 7-1, it is clear that various schemes can give different l for the same console. Fortunately, any algorithm having polynomial-time complexity under one scheme will have polynomial-time complexity under all others [Coffman, 1976].

Scheme	Encoding	Length	Bound[a]
task list, arc list	$T_1T_2T_3T_4T_5T_6T_7$, $(T_1T_5)\ (T_2T_5)\ (T_3T_6)\ (T_4T_6)\ (T_5T_7)\ (T_6T_7)$	50	$2t + 6a$
neighbour list	$(T_5)\ (T_5)\ (T_6)\ (T_6)\ (T_1T_2T_7)\ (T_3T_4T_7)\ (T_5T_6)$	38	$2t + 4a$
adjacency matrix	$\begin{bmatrix} 0&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0 \\ 1&1&0&0&0&0&1 \\ 0&0&1&1&0&0&1 \\ 0&0&0&0&1&1&0 \end{bmatrix}$	51	$t^2 + 2$

Table 8-1: Analysis of alternative encoding schemes for DMC taskforces.

a. where t represents the number of tasks and a the number of arcs in the console taskforce.

In fact, it would be hard to envisage a 'reasonable' encoding scheme that differs more than polynomially from Table 8-1. Although 'reasonable' is not easily formalised, it is clear that the scheme employed in a DCS algorithm should be concise and not padded with unnecessary symbols. Hence, numbers should be represented in any fixed base other than 1.

8.1.4 Time Complexity Functions

Scheduling algorithms posses a wide range of time complexity functions. However, the distinction between polynomial-time and exponential-time algorithms is clear.

8.1.4.1 Upper and Lower Bounds

Despite some successes, the record for proving lower bounds on the complexity of specific problems is poor [Garey, 1979]. Time complexity as defined is a worst-case measure or upper bound: a time complexity of 2^l means only that at least one problem instance of size l requires $O(2^l)$ operations to complete. For example, the FFT, one of the most frequently used signal processing algorithms, requires a maximum of $O(l \log l)$ arithmetic operations.

8.1.4.2 Polynomial-Time Algorithms

An important concept is the identification of the class of problems, P, solvable in time bounded by a polynomial in the length of the input, l. Correspondingly, a *polynomial-time* algorithm is one whose time complexity function is $O(p(l))$ for some polynomial function p. That polynomial-time roughly corresponds to tractability was first expressed by Edmonds who termed polynomial-time algorithms 'good algorithms' [Edmonds, 1965].

8.1.4.3 Exponential-Time Algorithms

In enumerative disciplines, as $m \rightarrow \infty$, the number of feasible schedules grows explosively. If a taskforce consists of m tasks then, assuming no precedence, the number of U-P schedules is $m!$, an exponential function in m.[1] Extending this argument to a M-P context, the `Prolog` source of Figure 8-1 shows that the number of schedules of m tasks on n PEs, $N(m, n)$, is:

$$N(m, n) = \frac{(m + (n-1))!}{(n-1)!} \qquad \text{Eq. (8-1)}$$

Algorithms with complexity not bounded as Section 8.1.4.2 are termed *exponential-time*. Note, this category includes non-polynomial (NP) functions, such as $l^{\log l}$, which are not normally regarded as exponential. Most exponential-time algorithms are merely variations on exhaustive search, whereas polynomial schemes usually demand some deeper insight.

1. If $m = 20$ and 10 million schedules evaluated per second, it would take almost 8000 years to check all 20! alternatives.

```
/* ------------------------------------------------------------------------ */
domains
/* ------------------------------------------------------------------------ */
    d_num_sch = real
    d_tsk_dom = symbol
    d_pro_nam = symbol
    d_tsk_lst = d_tsk_dom*
    d_pro_dom = pr( d_pro_nam, d_tsk_lst)
    d_pro_lst = d_pro_dom*

/* ------------------------------------------------------------------------ */
database
/* ------------------------------------------------------------------------ */
    sch( d_pro_lst)

/* ------------------------------------------------------------------------ */
predicates
/* ------------------------------------------------------------------------ */
    runalg( d_tsk_lst, d_pro_lst, d_num_sch)
    schedl( d_tsk_lst, d_pro_lst, d_pro_lst)
    delete( d_tsk_dom, d_tsk_lst, d_tsk_lst)
    delete( d_pro_dom, d_pro_lst, d_pro_lst)
    insert( d_pro_dom, d_pro_lst, d_pro_lst)
    chkdba( d_pro_lst)
    getsch( d_num_sch, d_num_sch)

/* ------------------------------------------------------------------------ */
clauses
/* ------------------------------------------------------------------------ */
    runalg( TaskList, ProcList, _):-
        retractall( _),
        schedl( TaskList, ProcList, Schedule),
        assert( sch( Schedule)),
        fail.
    runalg( _, _, NumScheds):-
        getsch( 0, NumScheds).

    schedl( [], Schedule, Schedule):-
        chkdba( Schedule), !.
    schedl( TaskList1, ProcList1, Schedule):-
        delete( Task, TaskList1, TaskList2),
        delete( pr( ProcName, Done), ProcList1, ProcList2),
        insert( pr( ProcName, [Task|Done]), ProcList2, ProcList3),
        schedl( TaskList2, ProcList3, Schedule).

    delete( Head, [Head|Tail], Tail).
    delete( Head1, [Head2|Tail1], [Head2|Tail2]):-
        delete( Head1, Tail1, Tail2).

    insert( pr( PN1, D1), [pr( PN2, D2)|T], [pr( PN1, D1), pr( PN2, D2)|T]):-
        PN1 < PN2, !.
    insert( pr( ProcName, Done), [Proc|Tail1], [Proc|Tail2]):- !,
        insert( pr( ProcName, Done), Tail1, Tail2).
    insert( Proc, [], [Proc]).

    chkdba( Schedule):-
        retract( sch( Schedule)), !,
        assert( sch( Schedule)),
        fail.
    chkdba( _).

    getsch( NumScheds1, NumScheds):-
        retract( sch( _)), !,
        NumScheds2 = NumScheds1 + 1,
        getsch( NumScheds2, NumScheds).
    getsch( NumScheds, NumScheds).
```

Figure 8-1: Prolog source to generate $N(m, n)$, assuming no precedence.

8.1.4.4 NP-Complete Algorithms

Many practical CO problems have been proved to be *NP-complete.*[1] Such problems are unlikely to be solvable by an amount of computation effort which is bound by a polynomial $p(l)$. It is clear that $P \subset NP$, as illustrated in Figure 8-2. Although still an open problem, '$P = NP$' is every unlikely, since NP contains many notorious combinatorial problems.

1. An extensive classification is given in the well-known book by Garey and Johnson [Garey, 1979].

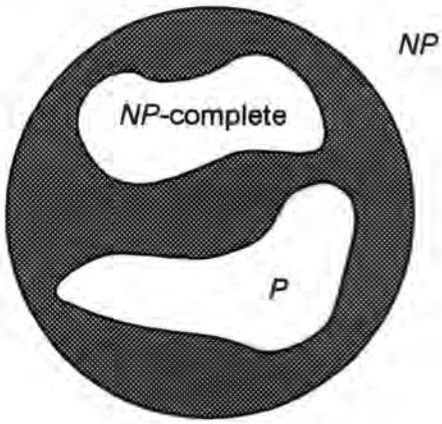

Figure 8-2: Illustration of the *NP* class of optimisation problems.

In spite of intensive research effort, no good algorithms are known for problems in *NP*. Therefore, the practical effect of *NP*-completeness is as a lower bound. A proof that a problem is *NP*-complete is a proof that the problem is not in *P* and therefore *intractable*. In the absence of a rigorous definition, *P* may be safely identified with polynomial-bounded algorithms.

8.1.5 Complexity of Task Scheduling

Most scheduling problems are *NP*-complete. In fact, even the general 2-processor case presents an *NP*-complete problem. In this section, reported theoretical results are summarised.

8.1.5.1 Uni-Processor Bounds

While many U-P problems are *NP*-complete, many can be solved to optimality [Lenstra, 1978]. For example, scheduling with arbitrary precedence constraints is clearly trivial because every feasible schedule is optimal. Problems with precedence relations forming a *tree* can be solved in $O(l \log l)$. Furthermore, scheduling unit-length tasks with arbitrary precedence constraints has been shown to be polynomial solvable [Gonzalez, 1978].

8.1.5.2 Two-Processor Bounds

The general 2-processor scheduling problem is *NP*-complete [Bruno, 1974]. Scheduling with arbitrary precedence constraints to minimise ω for unit-length tasks can be solved in $O(l^2)$ [Coffman, 1973]. Garey and Johnson have developed an $O(l^2)$ algorithm for the case where each task has to meet a given deadline [Garey, 1978]. However, the 2-processor problem for 1- or 2-unit tasks with arbitrary precedence constraints is *NP*-complete [Lenstra, 1978].

8.1.5.3 Multi-Processor Bounds

Scheduling tasks with precedence relations and individual deadlines on an n-processor M-P, $n \geq 3$, is solvable in $O(l \log l)$ [Conway, 1967]. Assuming unit processing times, the n-processor problem for tasks with in-tree precedence can be solved in $O(l)$ due to the often-cited algorithm by Hu [Hu, 1961]. With arbitrary precedence, however, this problem and therefore general DCS become *NP*-complete in the strong sense [Lenstra, 1978].

8.2 Stochastic Techniques

Stochastic search techniques are modelled on processes found in nature. These algorithms are often touted as near-optimal general-purpose strategies for notoriously difficult CO problems.

8.2.1 Simulated Annealing

Simulated Annealing (SA) is the mathematical analogue of cooling: just as in the natural annealing process, SA associates a cooling schedule with the CO problem to be optimised.

8.2.1.1 Overview

First proposed by Kirkpatrick, SA is a stochastic computational technique derived from statistical mechanics [Kirkpatrick, 1983]. The name originates from the strong analogy with the physical annealing of solids: the final 'frozen' configuration being optimal. SA has evolved as a general and robust technique for solving many *NP*-hard optimisation problems.

```
M = number of moves to attempt;
T = current "temperature";

for ( m = 1 to M )
{
    generate a random move i.e. perturb task schedule;
    evaluate the change in energy (cost), ΔE;
    if ( ΔE < 0 )
    {
        // downhill move towards minima: accept it
        accept move, and update schedule;
    }
    else
    {
        // uphill move: accept maybe
        accept with probability p = e^(-ΔE/T);
        update schedule if accepted;
    }
}
```

Figure 8-3: Pseudo-code listing of the Metropolis Algorithm.

To avoid entrapment at local minima, SA employs a stochastic relaxation technique. The Metropolis Algorithm of Figure 8-3 generates a sequence of M configurations for each value of T [Metropolis, 1953]. Without this counter-intuitive feature of accepting 'up-hill' transitions, illustrated in Figure 8-4, SA would revert to a form of iterative improvement.

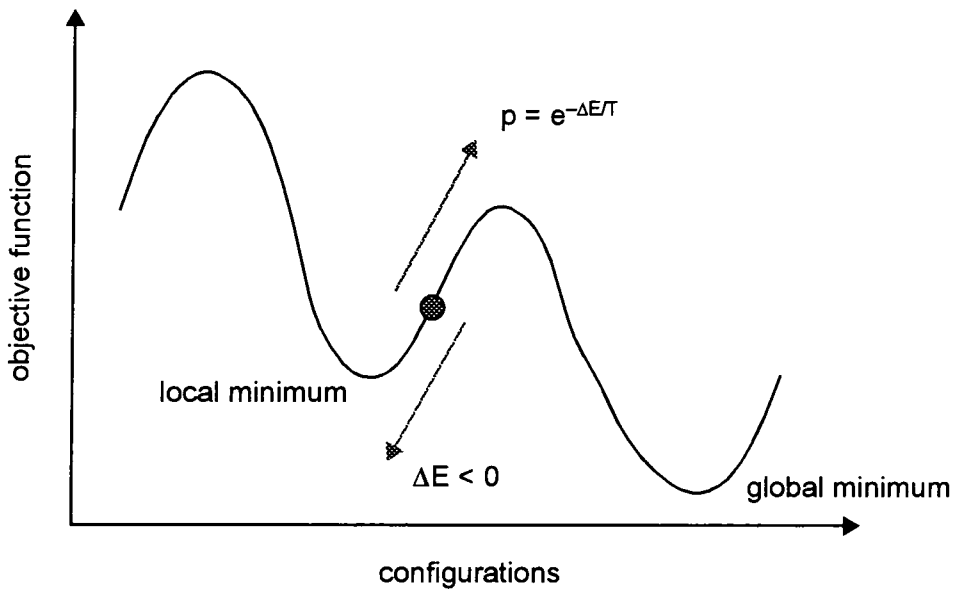

Figure 8-4: Sketch demonstrating the need for uphill moves.

8.2.1.2 Asymptotic Behaviour

It can be shown that, asymptotically, SA behaves as a true optimisation algorithm [von Laarhoven, 1987]. However, optimal configurations are only guaranteed for an infinite number of transitions [Lundy, 1986]. Convergence, and hence efficiency, depend on the so-called 'cooling schedule'. Aarts and Korst have derived a polynomial-time cooling schedule which can approximate asymptotic behaviour for some problem instances [Aarts, 1989].

8.2.1.3 Performance

SA finds near-optimal solutions which do not depend strongly on initial configuration [Aarts, 1989]. Computational effort for a given quality of solution is predominantly dependent on cooling schedule, which in term must be determined by trial and error. Furthermore, average-case run-time is typically close to worst-case [Gaudiot, 1988]. Experience shows that for even basic CO problems, SA is outperformed by tailored heuristics [Johnson, 1987].

8.2.2 Genetic Algorithms

Another popular stochastic search technique, genetic algorithms (GAs) are based on a model of population genetics and the principles of the *Darwinian Theory of Natural Selection.*

8.2.2.1 Overview

Holland has shown that a population of bit-strings can be made to "evolve as populations of animals do" [Holland, 1975]. If effective genetic operators can syntactically manipulate such chromosomal representations, then GA can be exploited as Figure 8-5. Populations constitute a parallel search trajectory through the space of all possible solutions. Chromosomes that survive from generation to generation are those which prove to be most fit.

```
P = population of chromosomal representations;
T = number of generations to attempt;
t = current configuration;

initialise population P(0);
evaluate population P(0);

for ( t = 1; t < T && !solution; t++ )
{
    // get most fit population
    select P(t) from P(t-1);

    // recombine population
    crossover P(t);
    mutate P(t);
    invert P(t);

    // determine fitness
    evaluate P(t);
}
```

Figure 8-5: Outline of GA procedure.

8.2.2.2 Recombination Operators

Recombination operators introduce new genetic material into the population by guiding the generation of new chromosomes through the modification of bit-string structures.

- **Crossover**

The function of crossover is to combine beneficial genetic material from parent chromosomes in the next generation. For example, the operator in Table 8-2 takes two structures, splits the strings at the same randomly determined point, and then creates two new structures by swapping the tail portions. It is this feature of evolution which is the key to the power of GAs.

Old Structures		New Structures	
100	10110	100	00101
111	00101	111	10110

Table 8-2: Example behaviour of GA crossover operator.

- **Mutation**

Mutation randomly changes a bit in a structure so introducing new material into the knowledge base. Assigned a very low percentage of activation, this operator occasionally introduces beneficial material into a problem instance. As a result, mutation operators guarantee that no genetic configuration will be permanently lost from a population.

- **Inversion**

Altering the positional linkages of bits within one structure by inverting the sequence between two randomly assigned points is termed inversion. For instance, '110011101' randomly cut after the 2nd and 6th bits, gives '111100101' when the centre sub-string is inverted. In this way, the location of genes is changed to provide a more fertile ground for recombination.

8.2.2.3 Performance

Like natural genetic systems, GAs quickly locate optimal regions in vast search spaces [Holland, 1975]. Unfortunately, many implementations are prone to premature convergence. Careful implementation of operators is necessary to adjust the balance between exploration and exploitation [Booker, 1987]. Many application-dependent issues, such as the percentage of descendants created by each operator, have considerable effect on optimality.

8.2.3 Application to DCS

While both paradigms have been successfully implemented for a range of difficult CO problems, their application to DCS is mitigated by algorithm- and problem-dependent issues.

8.2.3.1 Simulated Annealing

Although, asymptotically, SA is capable of locating near-optimal solutions, average-case algorithm run-time is typically close to worst-case. Furthermore, both solution and run-time

are largely dependent on cooling schedule. Due to these facts, a DCS based on SA would require a prolonged empirical study in an effort to determine suitable cooling schedules.

While polynomial-time cooling schedules may approximate asymptotic behaviour, a DMC control system would have to rely on pre-determined cooling schedules. No bounds can be determined on the quality of schedules generated in the real production environment. In addition, work in other areas has shown that SA is outperformed by tailored heuristics.

8.2.3.2 Genetic Algorithms

Unfortunately, GA implementations are often prone to premature convergence due to the over-exploitation of individual operators. To increase the ratio of exploration and exploitation requires a judicious balance of the descendants created by each operator. Again algorithmic issues have a direct bearing both on the run-time of the algorithm and quality of the solution.

Opportunities exist for incorporating problem-specific information into GAs: firstly, the initial schedule may be determined by a bootstrap heuristic; secondly, recombination can go beyond simple syntactic processing. However, DCS does not map well onto this numerical scheme and non-probabilistic techniques are better suited to exploiting tailored heuristics.

8.3 Deterministic Strategies

The static scheduling problem is not unique to the DCS problem being considered here. As it also exists in many other guises a variety of deterministic solutions have been proposed.

8.3.1 Graph Theory

As early as the 1960s, Ford and Fulkerson proposed a graph theoretic approach to task scheduling [Ford, 1962]. Subsequently, Stone showed that the problem could be solved by using algorithms constructed for finding maximum flows in commodity networks [Stone, 1973]. Developing this work, Lo included the concept of interference costs [Lo, 1988]. Although helpful for $n \leq 3$, this technique has limited application to DCS.

8.3.2 Integer Programming

Task scheduling can be cast as integer programming by defining a suitable objective function. To this end, Chu suggested a optimisation structure of the following type [Chu, 1980].

8.3.2.1 Problem Construction

$$\omega_{opt} = \sum_{i=1}^{m} \sum_{j=1}^{n} \left[w x_{i,j} s_{i,j} + (1-w) \sum_{k=1}^{m} \sum_{l=1}^{n} d_{j,l} s_{i,j} s_{k,l} \right] \qquad \text{Eq. (8-2)}$$

where w $(0 \le w \le 1)$ is the relative weighting of the two terms. The first term of Equation (8-2) is simply the execution time of T_i on P_j. The second term sums the IPC delays across all tasks that must communicate with T_i: $d_{j,l}$ is the IPC delay from P_j to P_l. Finally, Equations (8-3) and (8-4) model memory and execution time constraints for each PE:

$$\sum_{i=1}^{m} m_i s_{i,k} \le M_k, \text{ where } k = 1, \ldots, n \qquad \text{Eq. (8-3)}$$

$$\sum_{i=1}^{m} x_{i,j} s_{i,k} \le X_k, \text{ where } k = 1, \ldots, n \qquad \text{Eq. (8-4)}$$

8.3.2.2 Application to DCS

This 01 integer-programming approach has several liabilities, notably considerable computational expense [Hillier, 1986]. In terms of DCS, the absence of precedence relations is a critical shortcoming. Another drawback is the difficulty involved in extending this technique to include the finite capacity of IPC channels. Although it is possible to include additional constraints, run-time problems are greatly exacerbated [Thesen, 1978].

8.3.3 Clustering Techniques

Rather than tailor a problem to a mathematical technique, it may be more natural to begin with problem characteristics and develop a mapping algorithm, as shown by Efe [Efe, 1982]. Following this scheme requires developing an algorithm that not only clusters tasks and balances workload, but also selects a number of nodes and assigns clusters to these nodes.

Such clustering and iterative reassignment may arrive at solutions quickly for certain problems. However, the method is not always able to manage all constraints and can become

trapped in local minima [Konstantinides, 1989]. Although preferable to integer programming, this technique shares many of the former's problems and has limited application to DCS.

8.3.4 Dynamic Programming

Dynamic programming is a mathematical technique employed in problems where finding a solution can be reduced to a sequence of interrelated decisions forming a path in a state-space.

8.3.4.1 Overview

Scheduling taskforces on M-P architectures can be naturally cast in terms of state-space searching. As shown in the illustration of Figure 8-6, each alternative decision from a node corresponds to scheduling a different task, or idle period, on a processor. At each step the non-intuitive option of scheduling an idle period, lasting until the next processor has completed its currently assigned task, must be allowed to guarantee optimality [Coffman, 1976].

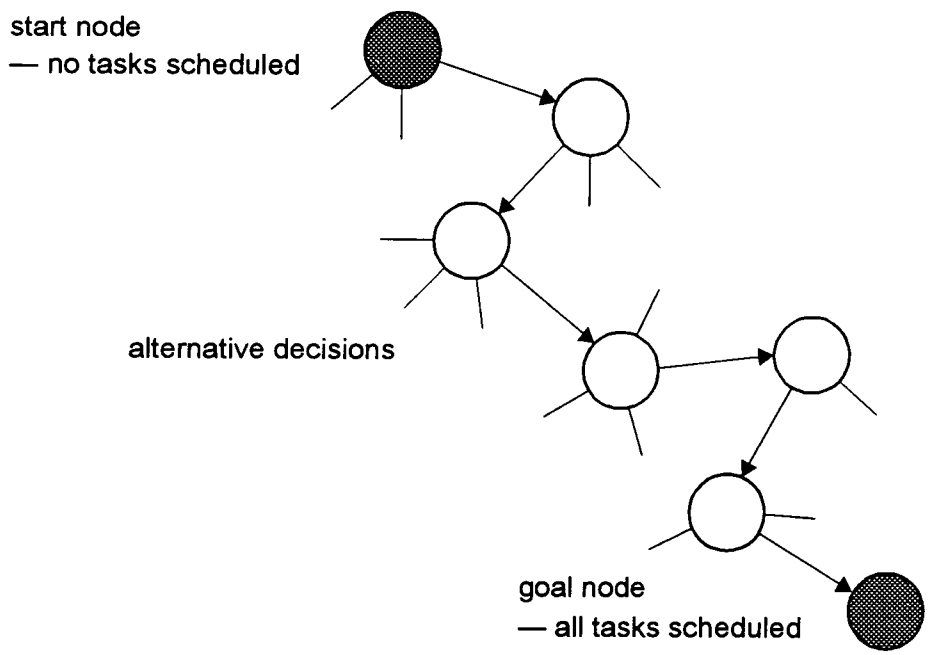

Figure 8-6: Task scheduling in terms of state-space searching.

8.3.4.2 Basic Strategies

Two basic strategies for exploring state-spaces are depth-first and breadth-first search: depth-first leads to long run-times as the search is uninformed; breadth-first disciplines maintain the set of decision nodes on the growing edge of the search tree. With an objective function as the optimisation criteria, *best*-first methods expand the best partial path at each node.

Expanding the best partial path at each decision node reduces algorithm run-times dramatically, but does not reduce the combinatorial explosion exhibited in state-space searching. A more sophisticated scheme is required, using problem-specific information. Heuristic rules avoid aimless paths, drawing the search process quickly towards a goal node.

8.3.4.3 A* Search

As A* is effectively an improved branch-and-bound, it is the preferred paradigm for the task scheduling problem examined here. A* employs the best-first principle with a heuristic underestimate of remaining cost [Nilsson, 1982]. It is based on the intuitive notion that a optimal solution is more likely to be reached by following paths with lower underestimates.

For each decision node n corresponding to a partial schedule, $h^*(n)$ denotes the cost of an optimal path to a goal node. A heuristic $h(n)$ is admissible if for all n if [Pearl, 1985]:

$$h(n) \leq h^*(n) \qquad \text{Eq. (8-5)}$$

A* employs $h(n)$ to guide the search tree expansion. One of the algorithms principle features is that it never expands a node with an underestimate greater than ω_{opt} [Sinclair, 1987].

8.3.4.4 Application to DCS

Although A* can revert to exhaustive search worst-case, alternative heuristics can be investigated by observing their ability to prune the search-tree. In order to realise tailored deterministic algorithms, four admissible application-specific heuristics are defined below.

- **Minimum Processor Cost — MPC**

This heuristic corresponds to the optimistic estimate of the finishing time of the partial schedule, completed with all currently unassigned tasks. If $t = \{t_1, t_2, \ldots, t_m\}$ is the list of un-scheduled tasks and $p = \{p_1, p_2, \ldots, p_n\}$ the list of active processors, then:

$$\text{MPC} = \frac{1}{n}\left(\sum_{i=1}^{m} e(t_i) + \sum_{j=1}^{n} f(p_j) \right) - F(t,p) \qquad \text{Eq. (8-6)}$$

where $e(t_i)$ is the execution time of task t_i, $f(p_j)$ the partial finishing time of processor p_j, and $F(t,p)$ the finishing time of the partial schedule. Note that MPC is computed under the assumption that precedence, preemption and distributed execution constraints are relaxed.

- **Maximum Remaining Critical-Path — MRC**

Alternatively, the finishing time of the remaining critical path for the most recently assigned task on each processor can be evaluated with respect to the current partial schedule. If $sl(t_i)$ is the level of task t_i, $ft(t_i)$ the finishing time of t_i, and $F(t,p)$ the finishing time of the partial schedule as before, then MRC for any node in the A* search-tree can be defined as:

$$\text{MRC} = \max\,(sl(t_i) - e(t_i) + ft(t_i) - F(t,p)), \quad 1 \le i \le m \qquad \text{Eq. (8-7)}$$

- **Minimum Independent Assignment — MIA**

The minimum assignment cost for each of the remaining tasks is computed ignoring all unscheduled tasks, but including IPC with all non-coresident (NCR) tasks already assigned. Here, $i(t_i)$ represents the earliest time that all NCR IPC is available to t_i on the earliest ready processor. MIA is then defined as the sum of minimum assignment costs for all t, or:

$$\text{MIA} = \sum_{i=1}^{m} (i(t_i) + e(t_i) - F(t,p)) \qquad \text{Eq. (8-8)}$$

- **Maximum Heuristic Underestimate — MHU**

By definition, MPC, MRC and MIA lie within the constraints imposed by Equation (8-5). If all underestimates are calculated simultaneously for each node, then MHU can be defined as:

$$\text{MHU} = \max\,(\text{MPC, MRC, MIA}) \qquad \text{Eq. (8-9)}$$

As the scheduler search-tree expands, clearly the individual heuristics approximate $h^*(n)$ with varying degrees of accuracy, particularly when n is in the vicinity of a goal node.

8.4 List Scheduling

In list scheduling strategies, tasks are first ordered according to some external priority scheme and then scheduled according to non-increasing priority.

8.4.1 Critical Path Method

One technique used to form a priority list is the critical path method (CPM), originally investigated by Hu [Hu, 1961]. In what is probably the most frequently cited reference in M-P scheduling, Hu developed two problems for equal-duration tasks. Although only optimal when precedence constraints form a rooted tree, the concept of *Hu-level* is widely applicable.

In CPM, as in PERT analysis, tasks are assigned to PEs according to the length of the precedence chains they head. The longest chains in the unexecuted part of the taskforce are those which have the greatest sum of execution times: they are termed 'critical paths' because the tasks which lie on them are those most likely to retard the execution of the DMC taskforce.

8.4.2 Hu-Level Labelling

A task T_i is given the label $sl(T_i) = lp(T_i) + 1$, where $lp(T_i)$ is the length of the longest path from T_i to the root task, which is labelled '0'. Tasks one unit removed are labelled '1', as Figure 8-7. As DMCs contain many root tasks[1], a 'dummy' root facilitates labelling. Priority is assigned by non-increasing Hu-level and the taskforce scheduled according to this regime.[2]

Ties are broken by assigning T_i to P_j with the smallest index j or, alternatively, by scheduling T_i with the longest execution time. Tasks with $e(T_i) > 1$ can be represented by an indivisible chain of unit-weight tasks whose sum equals $e(T_i)$. It can be shown that the level of the chain-task furthest from the root is equal to that of the multi-unit task [Kohler, 1975].

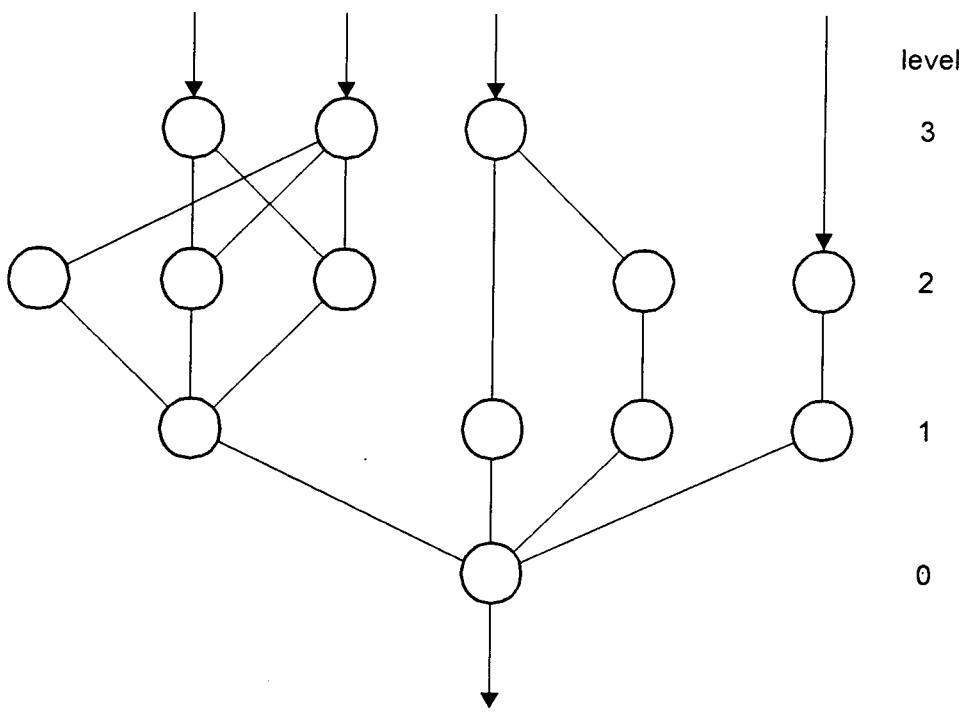

Figure 8-7: Hu-level labelling of a taskforce consisting of unit-length tasks.

1. Analogous to a 'forest of trees'.
2. Consequently, processors are never left idle if they can be assigned.

8.4.3 Alternative Labelling Schemes

The basis of the list scheduling discipline suggests several different labelling techniques. Assigning priorities assuming all tasks have identical $e(T_i)$, corresponds to Highest Levels first with No estimated times (HLN). Under this taxonomy, conventional CPM as described in Section 8.4.1 corresponds to Highest Levels first with Estimated times (HLE).

If a task's *co-level* is measured from the entry rather than exit task, two more labelling schemes are implied. Lowest Co-levels first with Estimated times (LCE) gives a task with the smallest co-level highest priority. If all tasks are assumed to have the same execution time and the LCE concept applied, this gives Lowest Co-levels first with No estimated times (LCN).

8.4.4 Upper Bounds on Schedule Quality

Hu's optimal solutions are limited to rooted trees of tasks with equal execution time. In this section, other known theoretical upper bounds for list schedules are compared and contrasted.

8.4.4.1 Preemptive vs. Non-Preemptive

List schedules may be preemptive or non-preemptive. For arbitrary n ($n \geq 1$), if the length of a non-preemptive schedule is ω_n and the length of a preemptive (optimal) schedule ω_p then:[1]

$$\frac{\omega_n}{\omega_p} \leq 2 - \frac{1}{n} \qquad \text{Eq. (8-10)}$$

The upper bound of Equation (8-10) can be shown to be tight for $n = 2$ [Coffman, 1973].

Kohler has shown that pathological problems for which ω_n / ω_p is arbitrarily close to 3/2 can be constructed for any $n \geq 2$ [Kohler, 1975]. He also conjectured that:

$$\frac{\omega_n}{\omega_p} \leq 2 - \frac{2}{n+1} \qquad \text{Eq. (8-11)}$$

if CPM is used to produce schedules, though this tighter bound has not yet been proven.

8.4.4.2 Precedence Constraints

When there are precedence constraints among the tasks, the worst-case performance bound for CPM with unit-length tasks, that is, the ratio of ω_{cpm} to the optimal finishing-time, ω_{opt}, is:

1. A higher priority 'ready' task will pre-empt a partially processed lower priority task.

$$\frac{\omega_{cpm}}{\omega_{opt}} \leq 2 - \frac{1}{n}. \qquad \text{Eq. (8-12)}$$

When there are no precedence constraints, the tasks are distributed across the processors purely by the order given in the priority-list L [Graham, 1976].

It has been shown that, for a set of independent tasks, specifically:

$$\frac{\omega_{cpm}}{\omega_{opt}} \leq \frac{4}{3} - \frac{1}{3n} \qquad \text{Eq. (8-13)}$$

which is substantially less than Equation (8-12), as $n \rightarrow \infty$. Since there are instances where these bounds are attained, they cannot be improved. As Equation (8-12) crosses the y-axis at the origin, it is better than Kaufman's bound for small values of ω_{opt}. [Kaufman, 1974].

8.4.4.3 Unequal Task Times

Let ω_p be the length of a taskforce scheduled by an algorithm which ignores the task uninterruptibility condition. Let ω_{opt} be the length of an optimal schedule, under the restriction that tasks may not be interrupted. If ω_{cpm} is the length of schedule given by the CPM algorithm (which also may not interrupt tasks) then, after Coffman [Coffman, 1973]:

$$\omega_p \leq \omega_{opt} \qquad \text{Eq. (8-14)}$$

$$\omega_{opt} \leq \omega_{cpm} \qquad \text{Eq. (8-15)}$$

Equation (8-14) follows from the fact that the possible sequence of schedules by any restricted algorithm are a subset of those available to the optimal unrestricted algorithm. Equation (8-15) follows from the definition of an optimal algorithm, since CPM is also restricted its schedules are possible outcomes for the algorithm which found ω_{opt}.

8.4.4.4 Mutually Commensurate Task Times

The results discussed above can be extended to trees providing that task times are mutually commensurate [Kaufman, 1974]. Tasks with weights greater than one can be represented by an indivisible string of unit-weight tasks. As all tasks in the restructured taskforce are of unit length, the optimal non-preemptive solution can be found using Hu's algorithm.

If ω_p is the length of an optimal schedule with arbitrary preemption, and ω_{cpm} that determined by a CPM algorithm prohibiting preemption of indivisible string tasks, then:

$$\omega_{cpm} \leq \omega_P + k - \left\lceil \frac{k}{n} \right\rceil \qquad \text{Eq. (8-16)}$$

where k is an upper bound on task size [Kaufman, 1974]. With ω_{opt} representing the optimal length of any non-preemptive schedule, combining Equation (8-14) to Equation (8-16) gives:

$$\omega_p \leq \omega_{opt} \leq \omega_{cpm} \leq \omega_p + k - \left\lceil \frac{k}{n} \right\rceil \qquad \text{Eq. (8-17)}$$

8.4.5 Application to DCS

In many instances critical-path schedules are extremely efficient. Figure 8-8 illustrates that even for worst case the theoretical finishing time, ω_{cpm} lies within twice the optimal value ω_{opt} for unit task times. Clearly a good strategy for constructing feasible DMC schedules, this heuristic scheduling strategy can be implemented to run in $O(n\log m)$ [Garey, 1978].

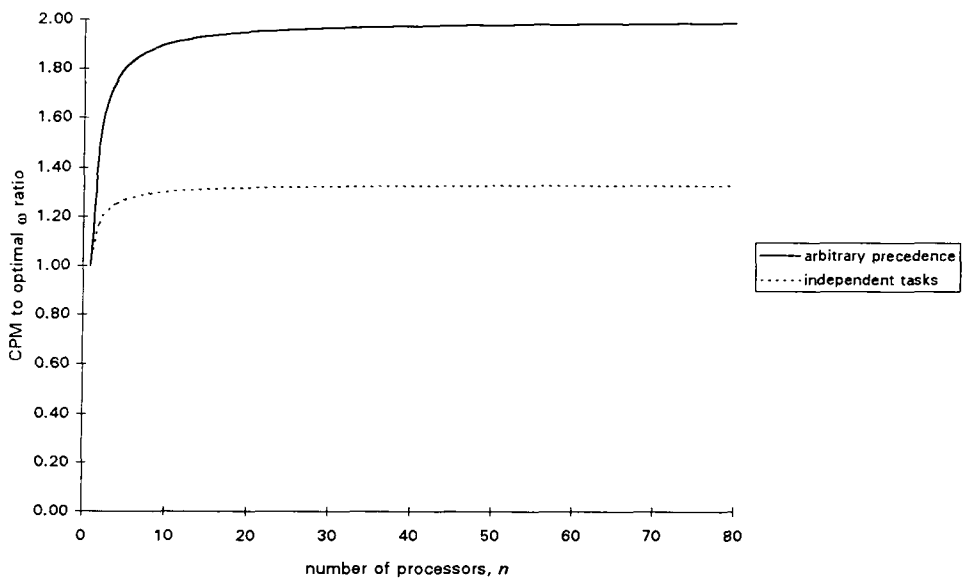

Figure 8-8: Upper bounds for CPM with unit task times.

Taskforces with arbitrary precedence and mutually commensurate $e(T_i)$ can be scheduled efficiently, providing that k is restricted as Figure 8-9. Incidentally, this discipline is more likely to generate an optimal solution as $n \rightarrow \infty$, for fixed m. As ω approaches the lower limit determined by the critical-path, increasing n will not reduce ω further.

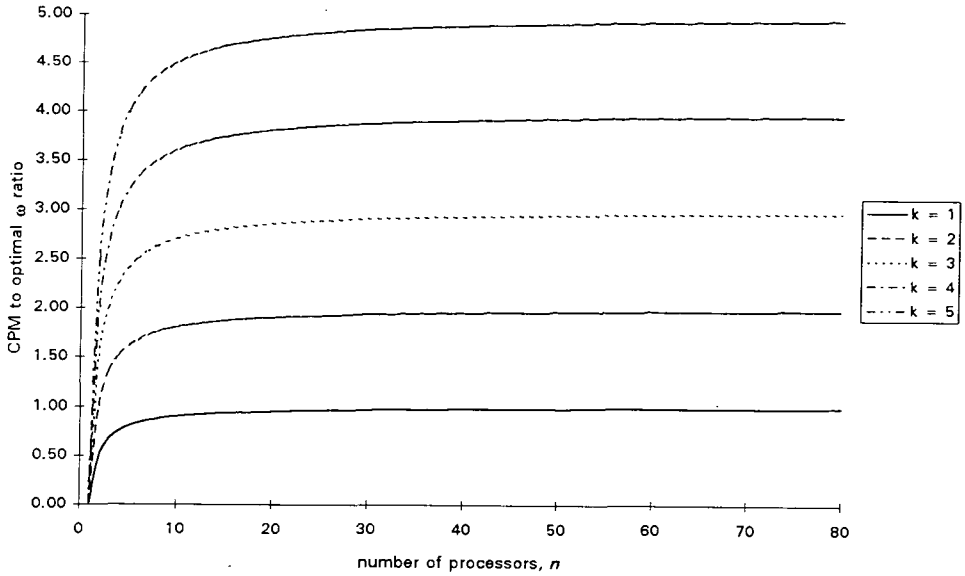

Figure 8-9: Upper bounds for CPM with mutually commensurate task times.

8.5 M-P Scheduling Anomalies

It has been shown, both theoretically and in practice, that scheduling algorithms can often locate feasible schedules which are at odds with intuitive concepts.

8.5.1 Idle Periods

With preemptive disciplines, it is never beneficial to introduce idle periods on processors when there are tasks available for execution. This is not true with non-preemptive disciplines as illustrated by the taskforce of Figure 8-8 and schedules of Figure 8-9. When P_2 is idle in the optimal schedule S_1 from 2 to 4, T_7 is continuously available for assignment. By disallowing idle periods and assigning T_7 to P_2 at time 2, the longer schedule S_2 is obtained.

8.5.2 List Scheduling

In list scheduling strategies, tasks are usually scheduled in order according to some external priority scheme. Schedule length depends not only on n, but also on the priority list used.

8.5.2.1 Overview

The unpredictable or anomalous behaviour of list scheduling refers to those instances when counterintuitive *increases* in schedule length ω are produced by: increasing the number of

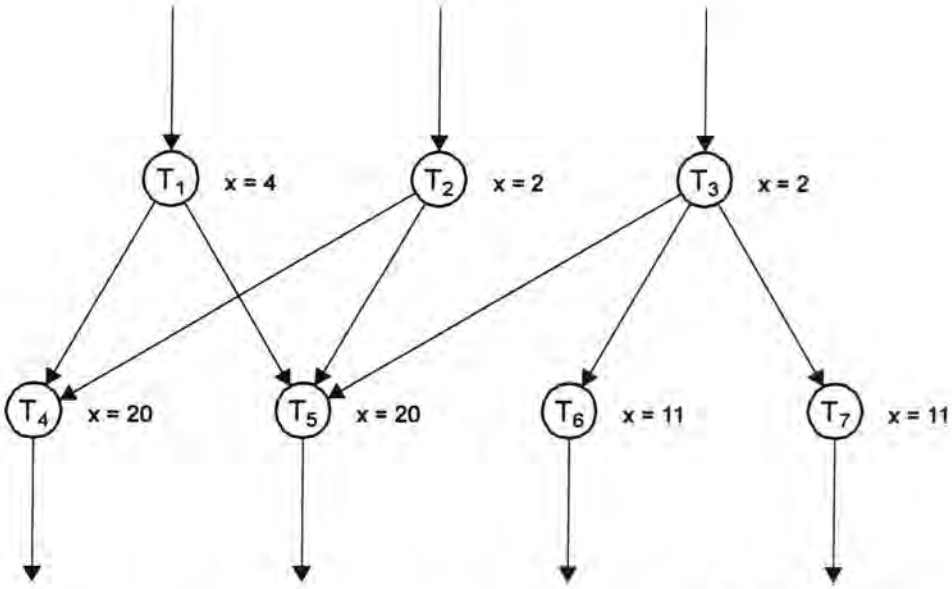

Figure 8-10: Example taskforce to illustrate the idle time anomaly.[a]

a. Here, the symbol 'x' represents the execution time of the adjacent task.

processors in P; decreasing some of the execution times in T; relaxing some of the precedence constraints in $<$; or by using a different priority list, L. That is, efforts intended to improve the values of model parameters can cause an unavoidable increase in ω.

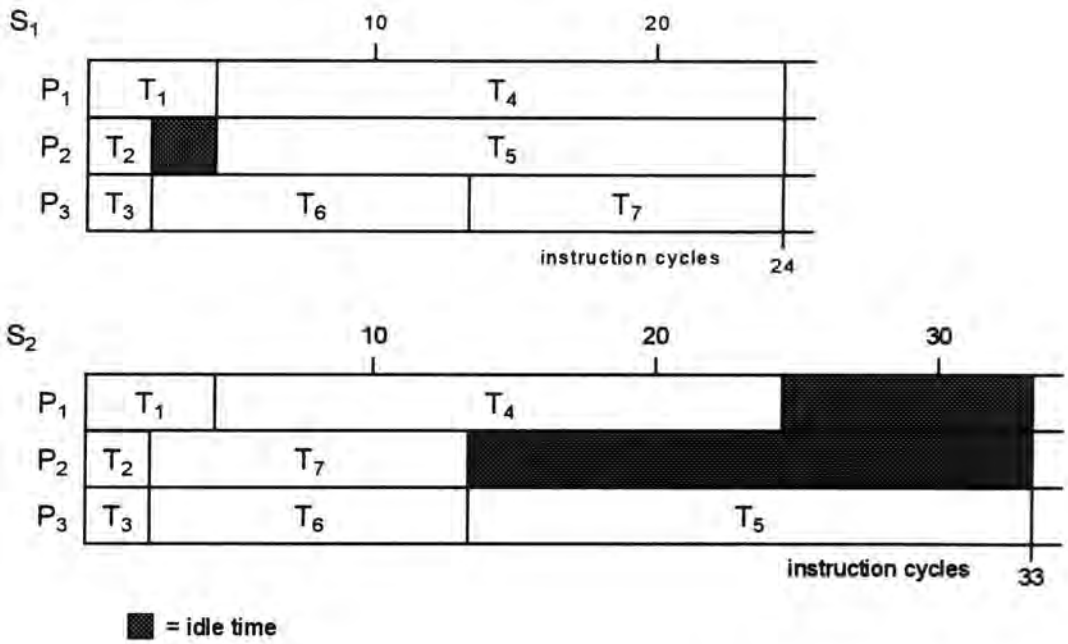

Figure 8-11: Schedules of the taskforce of Figure 8-8 to illustrate the idle time anomaly.

8.5.2.2 Upper Bounds

Graham has developed a general limit on these adverse effects by executing a set of tasks twice [Graham, 1976]. During the first execution tasks are characterised by the set of parameters $\{T, <, L, P, \omega\}$ and during the second execution by $\{T', <', L', P', \omega'\}$ such that every constraint of $<'$ is also in $<$. The result of this general bound is:

$$\frac{\omega'}{\omega} \leq 1 + \frac{n-1}{n'} \qquad \text{Eq. (8-18)}$$

Graham has shown that this bound is the best possible and, for instances where the number of processors remains the same (i.e. $n = n'$), the inequality has the elegant form:

$$\frac{\omega'}{\omega} \leq 2 - \frac{1}{n} \qquad \text{Eq. (8-19)}$$

which is achieved by the variation of any one of L, T, or $<$. As examples exist which match this inequality, the upper bound on the right-hand side cannot be improved [Graham, 1979].

8.5.2.3 Alternative Priority Lists

When 3 processors are used the inequality of Equation (8-20) is never more than 5/3. Even utilising the worst case priority list increases the finishing-time compared with that obtained using the best possible list by no more than 66.67%. This performance guarantee holds irrespective of the complexity of a given taskforce. No matter how carefully or carelessly the priority list is chosen, the ratio of finishing-times is strictly bounded by Equation (8-20).

8.5.2.4 Reduced Execution Times

Anomalies that result when task execution times are reduced were first discussed by Richards [Richards, 1960]. Results of simulations show that approximately 80% of test cases exhibit such irregularities. Manacher has developed an algorithm such that the completion time of a taskforce is not increased by reducing the execution time of any task. This is accomplished by adding a modest number of precedence constraints to the original ordering [Manacher, 1967].

8.6 Conclusions

In this analysis, consideration has been given to some of the more prominent aspects of scheduling deterministic DMC taskforces. It is assumed that taskforce precedence graphs are

acyclic, with no branching and that task execution times are exactly known a priori. It should be noted, though, that in other DSP environments these assumptions are not necessarily valid.

With the complexity of the DCS problem derived in Equation (8-1), the borderline between polynomial and *NP*-complete problems is apparent. Among these are found important examples such as scheduling unit-length tasks with arbitrary precedence for $n \geq 3$, and taskforces with *in-tree* precedence constraints, which have particular relevance here.

Both SA and GA techniques require empirical tuning of implementation-specific factors to guarantee convergence to an optimal solution. Depending on a cooling schedule or recombination operators to generate feasible console schedules within a run-time acceptable in the studio environment makes these stochastic disciplines somewhat unpromising for DCS.

Referring back to Figure 7-1, it is clear that the problem domain under investigation contains rather more inherent structure — e.g. precedence — than those in which general optimisation performs well. A* dynamic programming enables tailored heuristics, such as MPC, MRC, MIA and MHU developed in Section 8.3.4.4 to leverage directly off this structure.

Whereas A* can revert to exhaustive search, list scheduling forms the basis of many non-enumerative disciplines. This is particularly true as guaranteed upper bounds can be derived for many algorithms which fall into this category. Within the bounds discussed here, the choice of method used to generate the priority-list can make a great difference to ω.

Intuition suggests that the best priority lists are those in which tasks on longest critical paths appear near the head. In that case only relatively small additions to processor usage would be made at the end of the schedule. Four such labelling schemes — HLN, HLE, LCN and LCE — are put forward in Section 8.4.3 and will be developed further in the next chapter.

Lack of idle time and non-preemption are responsible for the unpredictable behaviour of CPM scheduling. Ready processors are forced to begin the execution of relatively unimportant tasks: tasks that are short or involved in few precedence constraints. Execution cannot be interrupted to commence more urgent tasks that subsequently become available.

By determining the performance ratio for an approximation algorithm, information is obtained which provides a useful and rigorously defined quantity with which different

approximation algorithms can be compared. In the remainder of this thesis, the author shows how tailored heuristic disciplines capture the salient features of scheduling DMC taskforces.

8.7 References

[Aarts, 1989] Aarts E and Korst J: *Simulated Annealing and Boltzmann Machines — A Stochastic Approach to Combinatorial Optimisation and Neural Computing*, Wiley & Sons, 1989, ISBN 0-471-92146-7.

[Booker, 1987] Booker L: "Improving Search in Genetic Algorithms", from *Genetic Algorithms and Simulated Annealing*, Pitman, 1987, ISBN 0268-7526.

[Bruno, 1974] Bruno J (et al.): "Scheduling Independent Tasks to Reduce Mean Finishing Time", *Communications of the ACM*, Vol. 17, No. 7, pp. 382-387, July 1974.

[Chu, 1980] Chu W W (et al.): "Task Allocation in Distributed Data Processing", *IEEE Computer*, pp. 57-69, November 1980.

[Coffman, 1973] Coffman E G and Denning P J: *Operating Systems Theory*, Prentice-Hall, 1973, ISBN 0-136-3786-8.

[Coffman, 1976] Coffman E G (ed.): *Computer and Job-Shop Scheduling Theory*, Wiley & Sons, 1976, ISBN 0-471-16319-8.

[Conway, 1967] Conway R W, Maxwell W L and Miller L W: *Theory of Scheduling*, Addison-Wesley, 1967, ISBN 0-876-26410-0.

[Edmonds, 1965] Edmonds J: "Paths, trees, and flowers", *Canadian Journal of Mathematics*, Vol. 17, No. 3, pp. 449-467, May 1965.

[Efe, 1982] Efe K: "Heuristic Models of Task Assignment Scheduling in Distributed Systems", *IEEE Computer*, pp. 89-102, April 1982.

[Ford, 1962] Ford L R and Fulkerson D R: *Flows in Networks*, Rand Corporation Research Studies, Princeton University Press, 1962.

[Garey, 1978] Garey M R, Graham R L and Johnson D S: "Performance Guarantees for Scheduling Algorithms", *Operations Research*, Vol. 26, No. 1, pp. 3-21, January 1978.

[Garey, 1979] Garey M R and Johnson D S: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W H Freeman & Company, 1979, ISBN 0-716-71044-7.

[Gaudiot, 1988] Gaudiot J L, Pi J I and Campbell M L: "Program Graph Allocation in Distributed Multicomputers", *Parallel Computing*, Vol. 7, No. 2, pp. 227-247, July 1988.

[Gonzalez, 1978] Gonzalez T and Sahni S: "Flowshop and Jobshop Schedules: Complexity and Approximation", *Operations Research*, Vol. 26, No. 1, pp. 36-52, January-February 1978.

[Graham, 1976] Graham R L: "Bounds on the Performance of Scheduling Algorithms", from *Computer and Job-Shop Scheduling Theory*, Wiley & Sons, 1976, ISBN 0-471-16319-8.

[Graham, 1979] Graham R L (et al.): "Optimisation and Approximation in Deterministic Sequencing and Scheduling: A Survey", from *Annals of Discrete Mathematics*, North-Holland, 1979.

[Hillier, 1986] Hillier F S and Lieberman G J: *Introduction to Operations Research*, fourth edition, Holden-Day, 1986, ISBN 0-816-23871-5.

[Holland, 1975] Holland J H: *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975, ISBN 0-262-08160-1.

[Hu, 1961] Hu T C: "Parallel Sequencing and Assembly Line Problems", *Operations Research*, Vol. 9, No. 6, pp. 841-848, December 1961.

[Johnson, 1987] Johnson D S (et al.): *Optimisation by Simulated Annealing: an experimental evaluation*, internal report, AT&T Bell Laboratories, 1987.

[Kaufman, 1974] Kaufman M T: "An Almost-Optimal Algorithm for the Assembly Line Scheduling Problem", *IEEE Transactions on Computing*, Vol. C-23, No. 11, pp. 1169-1174, November 1974.

[Kohler, 1975] Kohler W H: "A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems", *IEEE Transactions on Computers*, Vol. C-24, No. 12, pp. 1235-1238, December 1975.

[Konstantinides, 1989] Konstantinides K (et al.): "Scheduling and Task Allocation for Parallel Digital Signal Processing Architectures", *ICASSP '89*, September 1989.

[Kirkpatrick, 1983] Kirkpatrick S, Gelatt C D and Vecchi M P: "Optimisation by Simulated Annealing", *Science*, Vol. 220, No. 4598, May 1983.

[von Laarhoven, 1987] Laarhoven P J M von and Aarts E H L: *Simulated Annealing: Theory and Applications*, Reidel Publishing Company, 1987, ISBN 90-277-2513-6.

[Lenstra, 1978] Lenstra J K and Rinnooy Kan A H G: "Complexity of Scheduling under Precedence Constraints", *Operations Research*, Vol. 26, No. 1, pp. 22-35, January-February 1978.

[Lo, 1988] Lo V M: "Heuristic Algorithms for Task Assignment in Distributed Systems", *IEEE Transactions on Computers*, Vol. 37, No. 11, pp. 1156-1161, November 1988.

[Lundy, 1986] Lundy M and Mees A: "Convergence of an Annealing Algorithm", *Mathematical Programming*, Vol. 34, No. 2, pp. 111-124, February 1986.

[Manacher, 1967] Manacher G K: "Production and Stabilisation of Real-Time Task Schedules", *Journal of the ACM*, Vol. 14, No. 3, pp. 439-465, July 1967.

[Metropolis, 1953] Metropolis N (et al.): "Equations of State Calculations by Fast Computing Machines", *Journal of Chemical Physics*, Vol. 21, No. 5, pp. 1087-1091, May 1953.

[Nilsson, 1982] Nilsson N J: *Principles of Artificial Intelligence*, Springer-Verlag, 1982, ISBN 0-387-11340-1.

[Pearl, 1985] Pearl, J: *Heuristics: Intelligent Search Strategies for Computer Problems Solving*, Addison-Wesley, 1985, ISBN 0-201-05594-5.

[Richards, 1960] Richards P: *Timing Properties of Multiprocessor Systems*, Technical Report, TD-B60-27, Technical Operations Inc., Burlington, Mass., USA, 1960

[Sinclair, 1987] Sinclair J B: "Efficient Computation of Optimal Assignments for Distributed Tasks", *Journal of Parallel and Distributed Computing*, Vol. 4, No. 2, pp. 342-362, April 1987.

[Stone, 1973] Stone H S and Fuller S H: "On the Near-Optimality of the Shortest-Latency-First Drum Scheduling Discipline", *Communications of the ACM*, Vol. 16, No. 2, pp. 352-353, March 1973.

[Thesen, 1978] Thesen A: *Computer Methods in Operations Research*, Academic Press, 1978.

Chapter 9

DCS Research Framework

Details are given of the framework developed by the author to support the investigation of taskforce scheduling disciplines. Major features of the *DCS* engine are examined and results reported to illustrate the application of A* and CPM disciplines to digital mixing products.

9.1 Software Environment

In order to research scheduling algorithms in the DMC problem domain, a software environment has been developed specifically to harness the *DCS* deliverable. Incidentally, the methodology used here retains much of the philosophy developed at SSL for the *01* console.

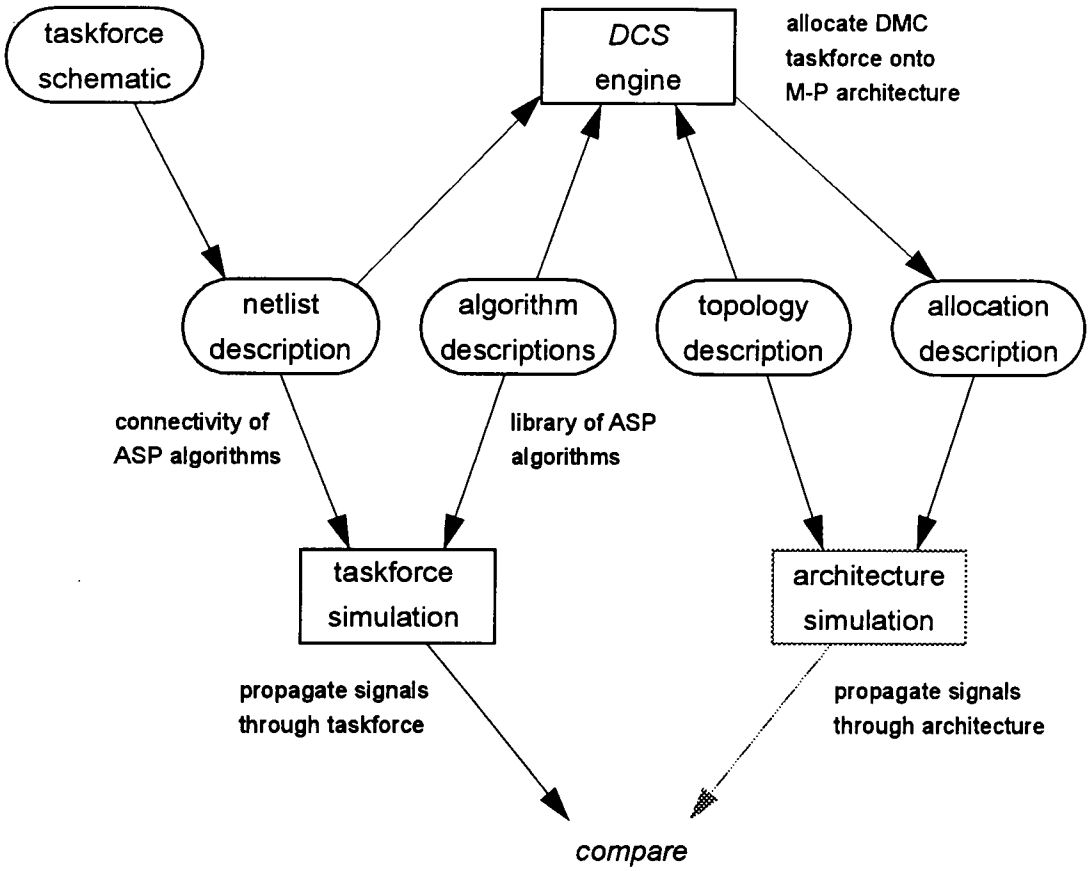

Figure 9-1: Block diagram of *DCS* research framework.

9.1.1 Hierarchical Console Design

A *schematic* is created from *symbols* forming its internal, functional description. The part also has a symbolic description, which represents its external appearance. The symbol is a box, or other appropriate shape, with input and output links as required by the schematic. Symbolic descriptions may be placed, or 'instanced', into any schematic higher up the DMC hierarchy.

Hierarchical design techniques lead to a clear, well-structured constructs. They facilitate 'top-down' design, starting with the overall functional requirement and adding detail as the design develops. A properly structured design ensures that common ASP tasks are debugged in isolation and entered only once, thus simplifying product development.

9.1.2 Graphics Capture

As illustrated in Figure 9-1, a hierarchical graphics-capture package has been employed to enable console taskforces to be entered as schematic diagrams. For example, an EQ section may be treated as a block in a symbol — such as a channel strip, and may itself be a network of ASP tasks — such as low, high, and mid-frequency sections (see Section 2.2.3).

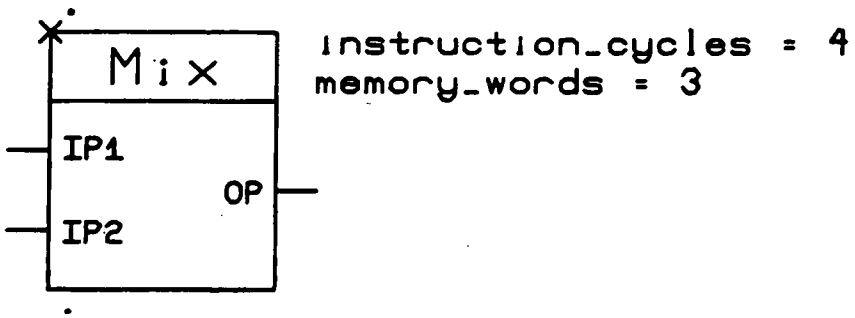

Figure 9-2: Symbol representing a 2-input bus-configured mix task.

In the same manner, the description of the target hardware may be generated via graphics-capture. To this end, a library of schematic primitives has been assembled as illustrated in Figures 9-2 & 9-3. Any DMC taskforce or M-P architecture that can be realised with primitives provided by the console manufacturer may be prepared for scheduling.

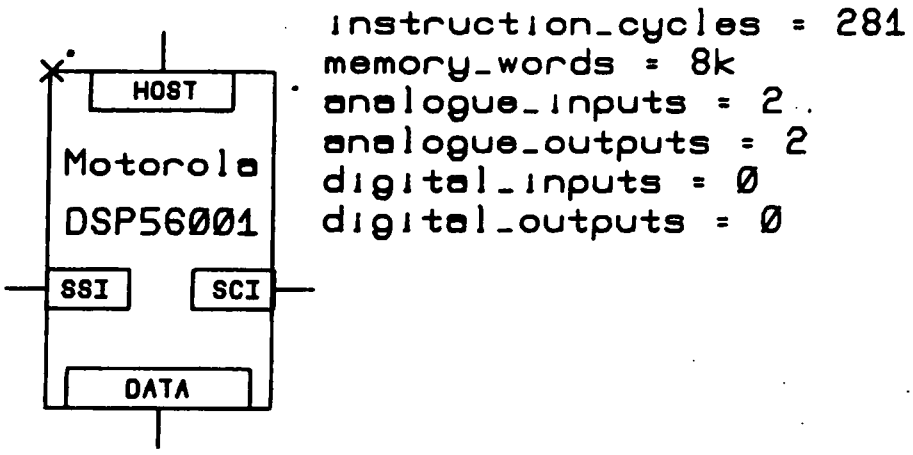

Figure 9-3: Symbol representing *HYMIPS* Motorola DSP56001 module.

9.1.3 Behavioural Models

Behavioural models have been created in the `HELIX` simulation language for each of the primitives in the DMC library. As shown in Figure 9-4, symbol attributes indicating resource requirements can be passed as parameters to these models. Of course, this approach could be extended to ASP in order to replace the trace routines currently used for debugging purposes.

As shown in Figure 9-5, a bi-quad kernel requires 14 instruction cycles, 7 words of P-space for code storage, and a further 10 of D-space for filter coefficients. A 27 MHz DSP56001 provides 281 instruction cycles per 48 kHz sample period, up to 192K words of P and D RAM, and can support 4 analogue or direct digital input-output ports [Motorola, 1989].

9.1.4 Taskforce Simulation

A data-flow simulator can be generated from behavioural models of the console ASP tasks, in accordance with the structure present in the taskforce schematic [Linton, 1991]. Two types of simulation can then be performed: *symbolic*, to verify the connectivity of the ASP algorithms in the DMC taskforce; and *timed*, to obtain an ideal lower bound on the schedule length.

Such taskforce simulation was employed in the early stages of this research to verify the operation of the scheduling algorithms investigated in this thesis. A similar approach has been used to test the three IPC schemes supported by the predicates `sch_non_cor(...)` and `sch_ins_ipc(Ipc, ...)`, described in detail later in this chapter in Section 9.2.3.1.

```
(* ================================================================== *)
MODULE TkfBhv;
(* ================================================================== *)

USE External Modules;

NETTYPE audiobus = RECORD
            audio : bits24;
            trace : PtrToString;
        END;

(* ------------------------------------------------------------------ *)

TASKTYPE Mix( instruction_cycles : integer;  (* attributes passed from..  *)
              memory_words       : integer); (* ..graphics capture symbol *)

   INWARD
        ip1, ip2 : audiobus;
    OUTWARD
        op       : audiobus;
    VAR
        TaskName : string20;
        TaskNum  : integer;
        working  : audiobus;

BEGIN
    IF InitialisationPhase THEN BEGIN          (* Initialisation Code *)
        NEW( op.trace);
        op.audio  := 0;
        op.trace^ := "";
    END
    ELSE BEGIN                                 (* Simulation Code *)
        IdentifyTask( TaskName, TaskNumber);
        EnableInputActivationTest( ip1);
        EnableInputActivationTest( ip2);
        REPEAT
            (* main body of behavioural model *)
            WAITFOR InputActivation( ip1) AND InputActivation( ip2)
                CHECK ip1, ip2;
            working.audio := new sample value;
            UpdateTrace( TaskName, ip1.trace, ip2.trace, working.trace);
            ASSIGN working TO op DELAY instruction_cycles;
        UNTIL FALSE; (* UNTIL simulation ends *)
    END; (* ELSE *)
END; (* Mix *)

(* ------------------------------------------------------------------ *)
.
.
behavioural models for other taskforce primitives
.
.
(* ================================================================== *)
```

Figure 9-4: Module of HELIX behavioural models for DMC taskforce primitives.

9.1.5 Preliminary Results

As an example of inital investigations, the DMC taskforce shown in Figure 9-6 can be executed sequentially such that $\omega_{sch} = 136$. With a critical path length ω_{cp} of 62 cycles, this 4:1 console can be allocated optimally onto 4-DSPs as Figure 9-7. Mean processor efficiency is 54.85% although P_1 remains unallocated throughout. While both A* and CPM disciplines produce an optimal schedule, A* requires 4-times the run-time of the latter method.

```
; ==================================================================
; Function: BIQUAD
;       Five-coefficient second-order canonical biquad filter,
;       as per Equation (3-7), p. 3-9
; Resource requirements:
;       instruction cycles - 15 cycles
;       P       memory      - 25 words
;       X-data memory       -  5 words
;       Y-data memory       -  4 words
; ==================================================================

BIQUAD\
    MACRO input,output,coeffs

        ; ---------------------------------------------------------------
        ; reserve Y-data memory for input and output states
        ; ---------------------------------------------------------------

        org     yh:

_inputs         DS      2               ; 2 previous input states
_outputs        DS      2               ; 2 previous output states

        ; ---------------------------------------------------------------
        ; initialise pointer and modulo registers
        ; ---------------------------------------------------------------

        org     p:

        move    #coeffs,r0
        move    #0004,m0
        move    #_inputs,r4
        move    #0001,m4
        move    #_outputs,r5
        move    #0001,m5

        ; ---------------------------------------------------------------
        ; perform biquad kernel
        ; ---------------------------------------------------------------

biquad  move    input,y1

        mpy     x0,y1,a         x:(r0)+,x0      y:(r4)+,y0
        mac     x0,y0,a         x:(r0)+,x0      y:(r4),y0
        mac     x0,y0,a         x:(r0)+,x0      y:(r5)+,y0
        mac     x0,y0,a         x:(r0)+,x0      y:(r5),y0
        macr    x0,y0,a         x:(r0)+,x0      y1,y:(r4)

        move    a,output,y:(r5)

    ENDM ; BIQUAD
; ==================================================================
```

Figure 9-5: DSP56001 assembly code for DMC EQ biquad kernel.

9.2 Digital Console Scheduler

As shown in Figure 9-1, a digital console scheduler is required in order to allocate a DMC taskforce of *m* tasks onto a M-P hardware architecture comprising *n* processors. Here, the predicate hierarchy of the *DCS* engine is described in relation to supported menu options.[1]

9.2.1 Overview

In this overview, the implementation of *DCS* is described together with the specification of the computer system used in this thesis. Database encoding schemes are also presented.

1. The reader should consult Appendix D for a complete listing of the *DCS* `Prolog` source.

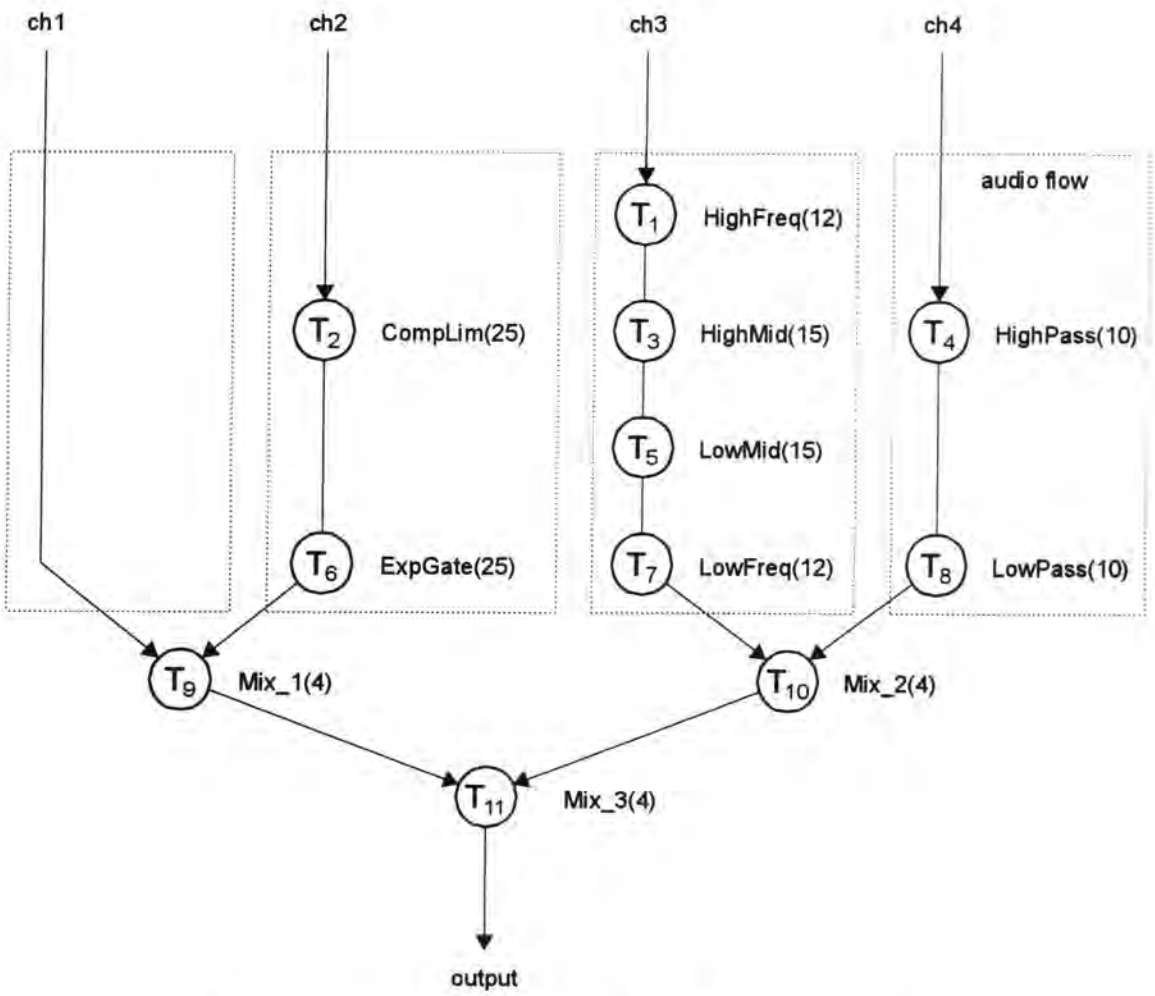

Figure 9-6: Example 4-input: 1-output console taskforce.

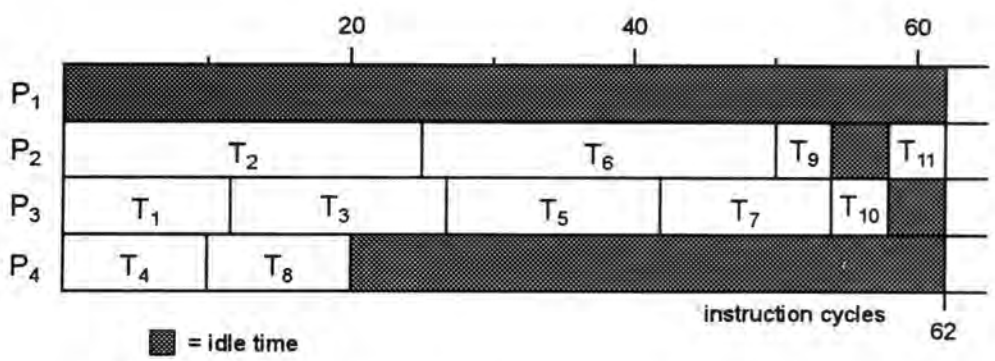

Figure 9-7: Optimal allocation of example taskforce of Figure 9-6.

9.2.1.1 Implementation

The scheduling algorithms outlined in the following pages have been coded in Borland's *Turbo Prolog*, with executables running on a 66 MHz Pentium with 16 Mb RAM. As a

compiled `Prolog`, this implementation proves efficient though it should be noted that the code generator targets the 80286. No advantage is taken of the '586 enhanced instruction set.

As revealed in Section 7.1, the DMC taskforce consists of ASP tasks with execution times known *a priori* and precedence constraints — represented here as *nets* in a schematic. The *Turbo Prolog Toolset*, which provides facilities similar to `UNIX lex` and `yacc` utilities, has been used to parse the BNF netlists produced by the *SilvarLisco* tools into clauses.

9.2.1.2 Database Encoding

As examined in Section 8.1.3, any 'reasonable' encoding scheme ensures that polynomial-time complexity can be maintained. Within the internal `Prolog` database, the taskforce is represented using the data model detailed in Table 9-1. After the database is consulted, these structures are resolved into the internal *DCS* representation described in `dcs_glb.pub`.

Description	Domain[a]
task	`task(d_tsk_nam, d_tsk_typ)`
task type	`tasktype(d_tsk_typ, d_tsk_exe, d_tsk_inp, d_tsk_out)`
net	`net(d_net_nam, d_net_inp, d_net_out)`

Table 9-1: *DCS* internal database representation of DMC taskforces.

a. The prefix `d_` indicates that the corresponding symbol is a user-defined domain name.

For efficiency reasons, non-numeric domains are represented as symbols, where possible. Note that `d_net_inp` is a `Prolog` symbol whereas `d_net_out` is a `d_sym_lst`, as free-grouping enables a DMC channel to operate as a broadcast signal path. Similarly, the homogeneous M-P architectures employed in this thesis are denoted as shown in Table 9-2.

Description	Domain
processor	`proc(d_pro_nam, d_pro_typ)`
processor type	`proctype(d_pro_typ, d_pro_exe)`

Table 9-2: *DCS* internal database representation of M-P architectures.

9.2.2 Scheduling Strategies

The three distinct algorithms developed as part of this research project are reviewed in the following sections. For example, selecting CPM from the scheduling strategy menu causes *DCS* to employ a critical-path list-scheduling discipline to schedule the DMC taskforce.

9.2.2.1 Critical-Path Method

Choosing CPM initiates a list scheduling algorithm developed from the work reported in Section 8.4.1 and detailed as pseudo-code in Figure 9-8. `cpm_bgn_alg(Ipc, Lbl, ...)` calls `utl_lbl_tkf(Lbl, ...)` to label the taskforce using the `Lbl` strategy.[1] Using a quick-sort method, `utl_srt_prl(...)` produces a priority list ordered by non-increasing level. As Table D-1 indicates, this implementation supports five static labelling schemes.

```
// ======================================================================
// DCS CPM scheduling algorithm
// ======================================================================

cpm_bgn_alg(...)
{
    // label taskforce
    initialise levels to zero;
    current_task = root_task;

    while ( !entire taskforce traversed )
    {
        // calculate HLE label
        select next_task;
        new_level = current_task.level + next_task.execution_time;

        // update task level
        if ( new_level > next_task.level )
            next_task.level = new_level;
    }

    // create priority list
    quick-sort task-list by non-increasing level;

    // schedule taskforce
    while ( priority list != [] )
    {
        // select task with no unfinished predecessors
        while ( task has unfinished predecessors )
            select next task from priority list;

        // schedule ready task on available processor
        if ( available processor has sufficient resources )
            schedule task on this processor;
    }
}

// ======================================================================
```

Figure 9-8: Pseudo-code of *DCS* CPM algorithm, using the HLE labelling scheme.

9.2.2.2 Dynamic List Scheduling

`dyn_bgn_alg(Ipc, Lbl, Dyn, ...)` extends CPM by dynamically re-labelling the taskforce after each scheduling step. In order to store the dynamic levels of ready-task,

1. As a callee of `utl_lbl_tfk(...)`, `utl_cnv_lvl(...)` converts co-levels to pseudo-levels via ω_{cp}.

available-processor combinations, `dyn_ini_edb()` initialises a 4-th order B-tree in EMS. `dyn_cre_sch(Ipc, Dyn, ...)` interrogates the B-tree and schedules tasks accordingly. As dynamic labelling schemes are the subject of Chapter 10, they are not detailed further here.

9.2.2.3 A* Search

To initialise the search-tree, `ast_ini_tre(...)` creates a EMS-resident B-tree with a 'null' root node. `ast_exp_tre(Ipc, Hrt, ...)` generates all successors n of the cheapest live node bound by `ast_chp_liv(...)`, computes $h(n)$ and inserts n in order of non-decreasing cost. If `ast_gol_nod(...)` locates a goal node which satisfies Equation (8-5), AST completes. Otherwise `utl_rep_eat()` generates an infinite supply of backtracking points.

9.2.3 Control Parameters

During operation DMC taskforce and M-P architecture are selected via BGI list boxes. *DCS* control parameters are chosen from the modeless menu structure presented in Table D-1.

9.2.3.1 IPC Mechanism

The work in Section 7.4 indicates that several IPC mechanisms are relevant here. `inf_get_ipc(...)` accepts user input and `inf_vld_ipc(...)` validates the menu item against known IPC schemes. The option chosen is passed as `xxx_bgn_alg(Ipc, ...)`.

- **FWE — fully overlapped with execution**

Calling `sch_ins_ipc(Ipc, ...)` with `Ipc` bound to `s_ipc_fwe` schedules multiple IPC messages fully-overlapped with task execution as described in detail in Section 7.4.5.3.[1]

- **NSL — non-overlapped, single link**

Applying `sch_ins_ipc(Ipc, ...)` with `Ipc` bound to `s_ipc_nsl` assumes one IPC link and a transport mechanism that cannot overlap with task execution (see Section 7.4.2.2).

- **NML — non-overlapped, multiple links**

`sch_ins_ipc(s_ipc_nml, ...)` facilitates multiple IPC channels but, as advanced in Section 7.4.5.1, concurrent computation and communication is not supported in this instance.

1. The prefix `s_` denotes the corresponding string as a *DCS* internal symbol.

9.2.3.2 Static Labelling Scheme

DCS supports the four static labelling schemes described in Section 8.4.3, in addition to a random strategy. `inf_get_lbl(...)` accepts user input and `inf_vld_lbl(...)` validates the menu item. The labelling scheme is passed to `CPM` as `cpm_bgn_alg(Ipc, Lbl, ...)`.

- **HLN — highest levels first, no estimated times**

Calling `utl_lbl_tkf(s_lbl_hln, ...)` with `Lbl` bound to `s_lbl_hln` assigns levels using `utl_lbl_all(Lbl, ...)` which assumes all tasks have identical execution time.

- **HLE — highest levels first, estimated times**

Conventional CPM as described in Section 8.4.1 corresponds to `utl_lbl_tkf(Lbl, ...)` with `Lbl` bound to `s_lbl_hle` which is subsequently passed to `utl_lbl_all(Lbl, ...)`.

- **LCN — lowest co-levels first, no estimated times**

Defining task *co-level* as measured from the entry rather than exit task, `utl_lbl_tkf(s_lbl_lcn, ...)` gives a task with small co-level high priority.

- **LCE — lowest co-levels first, estimated times**

`utl_lbl_tkf(s_lbl_lce, ...)` applies the lowest co-level concept using `utl_lbl_all(s_lbl_lce, ...)` which estimates task times using `utl_est_exe(...)`.

- **RND — random**

`utl_lbl_tkf(Lbl, ...)` with `Lbl` bound to `s_lbl_rnd` assigns arbitrary task labels, calling `utl_ini_lvl(s_ini_rnd)` to initialise levels randomly in the range 0 to 100.

9.2.3.3 Dynamic Labelling Scheme

`dyn_bgn_alg(Ipc, Lbl, Dyn, ...)` initiates the `DYN` scheduling algorithm with both static and dynamic taskforce labelling schemes. `inf_get_dyn(...)` accepts user input and `inf_vld_dyn(...)` validates the choice against known *DCS* dynamic-labelling schemes.

- **NON — none**

NON is essentially a conventional statically-labelled CPM scheme, employing `dyn_ins_dls(s_dyn_non, ...)` to insert *static* levels into the `DYN` priority mechanism.

- **PRO — select available processor first**

`dyn_ins_dls(Dyn, ...)` with `Dyn` bound to `s_dyn_pro` inserts dynamic levels for all ready tasks T_r using the available processor P_a with smallest partial finishing-time.

- **TSK — select ready task first**

`dyn_ins_dls(s_dyn_tsk, ...)` generates dynamic levels for the ready task T_r which has the highest static level for all available processors P_a in the partial schedule.

- **ALL — all combinations**

With `Dyn` bound to `s_dyn_all`, `dyn_ins_dls(Dyn, ...)` inserts dynamic levels for all T_r and P_a. `dyn_sch_nxt(...)` schedules T_r on P_a such that dynamic level is maximised.

9.2.3.4 Heuristic Underestimate

Four heuristics admissible in the DMC problem domain have been described in Section 8.3.4. `inf_get_hrt(...)` accepts user input and `inf_vld_hrt(...)` validates the menu item chosen. The heuristic underestimate is passed to AST as `ast_bgn_alg(Ipc, Hrt, ...)`.

- **NON — none**

Calling `ast_hrt_val(Hrt, ...)` with `Hrt` bound to `s_hrt_non` simply returns zero, corresponding to no heuristic underestimate or $O(n^m)$ enumeration of all feasible solutions.

- **MPC — minimum processor cost**

`ast_hrt_val(s_hrt_mpc, Ipc, ...)` estimates the finishing time of the partial schedule via `ast_mpc_hrt(Ipc, ...)` which relaxes all the constraints of Section 7.1.3.

- **MIA — minimum independent assignment**

`ast_hrt_val(s_hrt_mia, Ipc, ...)` calls `ast_mia_hrt(Ipc, ...)` which ignores all unscheduled tasks but includes IPC with all non-coresident (NCR) tasks already assigned.

- **MRC — maximum remaining critical-path**

ω for the remaining critical-path for the most recently assigned task on each PE is evaluated by `ast_hrt_val(s_hrt_mrc, Ipc, ...)` through calling `ast_mrc_hrt(Ipc, ...)`.

- **MHU — maximum heuristic underestimate**

As MPC, MRC and MIA must all satisfy Equation (8-5), `ast_hrt_val(s_hrt_mhu, ...)` gives MHU by calculating all heuristics simultaneously and binding to the maximum value.

9.2.4 Scheduler Outputs

As well as displaying a textual description of the DMC schedule, including idle periods and IPC transfers, *DCS* also supports console taskforce and M-P architecture analysis.

9.2.4.1 Pre-Schedule Statistics

Using `lst_num_elm(...)` on the task-list, precedence-list and processor-list created from internal database facts provides the number of tasks m, precedence relations and processors n, respectively. In order to quantify the distribution of task granularity, `clc_tsk_grn(...)` determines the minimum, maximum and average granularity of tasks in the DMC taskforce.

Sequential schedule length ω_{seq} is calculated in `clc_seq_sch(...)` by simply summing all task execution times. `clc_crt_pth(...)` determines the critical path ω_{cp} by temporarily labelling the taskforce using HLE. *Taskforce parallelism*, defined as ω_{seq}/ω_{cp}, acts as lower bound on n required to minimise the schedule length, ω_{sch}.[1]

9.2.4.2 Pre-Schedule Checks

In the *DCS* implementation presented in this thesis, three separate indicators are used to establish that the DMC taskforce can be scheduled on the selected M-P architecture. Firstly, `inf_grn_tst(...)` recursively determines if all the ASP tasks in the DMC taskforce have execution times strictly less than the processing resource supplied by each of the M-P PEs.

`inf_fit_tst(...)` binds to `s_tst_pas` if the hardware resource capacity given by `clc_hdw_rsc(...)` is greater than the sequential schedule w_{seq} or `s_tst_fai` otherwise. Finally, `inf_ctp_tst(...)` employs `clc_crt_pth(...)` to ascertain if the critical path inherent in the DMC taskforce is less than the processing resource supplied by each PE.

9.2.4.3 Post-Schedule Statistics

Post-schedule statistics are sent to the default output device using the `wri_pst_stt(...)` predicate. `clc_run_tim(...)` calculates algorithm run-time by retracting internal database facts which log system time prior to *DCS* execution and on termination. Schedule length ω_{sch} is obtained from the completed schedule — the processor-list — by `clc_fin_tim(...)`.

---

1. Normalised taskforce parallelism, represented in this thesis by π, must therefore be defined as $\frac{\omega_{seq}}{n\omega_{cp}}$.

To calculate speed-up, `clc_tsk_spd(...)` uses `clc_seq_sch(...)` described in Section 9.2.4.1 to determine ω_{seq} with ω_{sch} obtained as above. `clc_thr_put(...)` evaluates through-put as the number of ASP tasks computed per sample period. Following the work on efficiency, `clc_pro_utl(...)` determines processor utilisation as Section 7.3.3.

9.3 Performance of AST

Here, *DCS* is used to compare the four admissible underestimates defined in Section 8.3.4.4. A homogeneous 2-DSP56001 target and fully-overlapped (FWE) IPC help determine which of these application-specific heuristics may form the basis of efficient approximation algorithms.

9.3.1 Time & Space Complexity

As discussed previously in Section 8.1.4, the DCS problem can be solved through exhaustive enumeration of all feasible schedules. Equation (8-1) indicates that this scheme is $O(n^m)$ and thus prohibitive for relatively small instances. In order to gauge time complexity, the principal performance metric employed here is the run-time t required to produce an optimal schedule.

Comparisons on this basis alone, however, are biased against MHU since this heuristic requires rather more computation than the individual underestimates also considered in this survey. As a result, the number of state-space nodes s generated by each heuristic is recorded since the ability to prune the search tree gives a strong indication of true space complexity.

9.3.2 Algorithm Run-Time

Table 9-3 details AST algorithm run-time for taskforces based on traditional mix-buses. While MRC results in the shortest run-time, relative performance is more visible on log-log axes.

9.3.2.1 Bus-Configured Taskforces

From Figure 9-9, it is apparent that MPC performs little better than no heuristic underestimate (i.e. NON) for bus-oriented taskforces. Referring back to Equation (8-5), this behaviour is explained by the fact that, in calculating MPC, the remaining unscheduled tasks are simply apportioned across the M-P. As all the constraints of Section 7.1.3 are ignored, $h(n) \ll h^*(n)$.

channels	NON	MPC	MIA	MRC	MHU
4	0:00:00.06	0:00:00.06	0:00:00.06	0:00:00.06	0:00:00.06
8	0:02:27.91	0:01:34.27	0:00:00.11	0:00:00.09	0:00:00.08
12	1:54:23.00	0:48:14.10	0:00:00.93	0:00:00.16	0:00:00.22
16	17:59:50.86	6:04:21.00	0:00:11.04	0:00:00.28	0:00:00.50
24	—	—	0:02:15.01	0:00:01.16	0:00:01.70
32	—	—	0:09:54.52	0:00:03.19	0:00:04.50
40	—	—	0:22:36.66	0:00:07.69	0:00:10.38
48	—	—	0:54:17.63	0:00:15.10	0:00:20.16
56	—	—	1:28:47.38	0:00:26.47	0:00:34.88
64	—	—	2:24:00.00	0:00:46.19	0:00:59.98

Table 9-3: *DCS* run-time using AST–FWE to schedule bus-type taskforces.

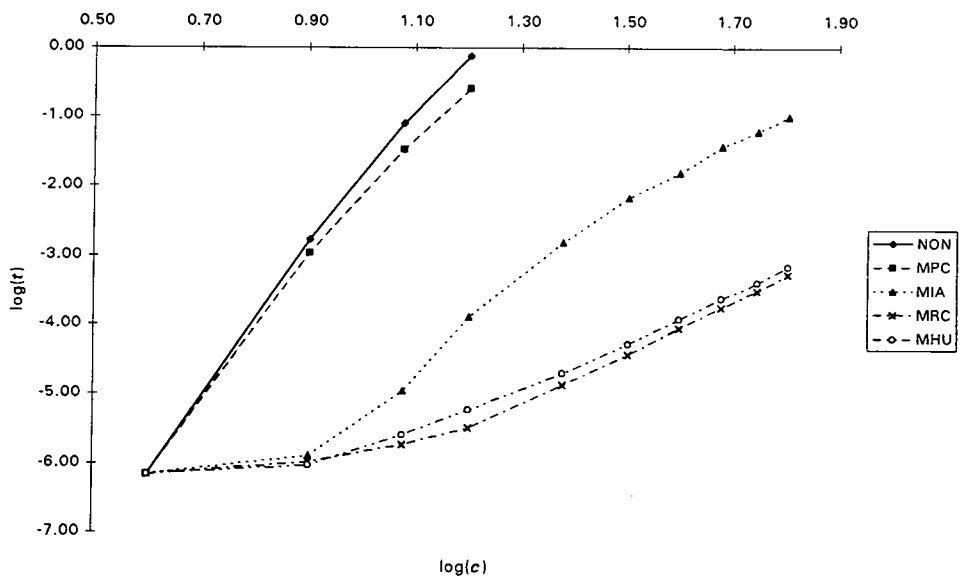

Figure 9-9: AST–FWE run-time for bus-type taskforces on a 2-DSP M-P.

While MIA initially appears attractive, this scheme degenerates towards an enumerative process as the number of channels c is increased. Rather better results are reported for MRC and the combined MHU strategy. MHU lies approximately 3.35% above MRC due to the calculation of MPC, MIA and MRC heuristics for each node in the search-space.

9.3.2.2 Tree-Configured Taskforces

Very few results can actually be produced for taskforces with mix-tree structures as shown in Figure 9-10. In fact, AST without any heuristic underestimate[1] does not complete with 24:00:00.00 for an 8-channel console. While MRC is reasonably efficient in this context, MIA generates optimal schedules in less time for 2 of the 3 cases reported here. With the loose precedence constraints inherent in trees, a combined approach seems to be most effective.

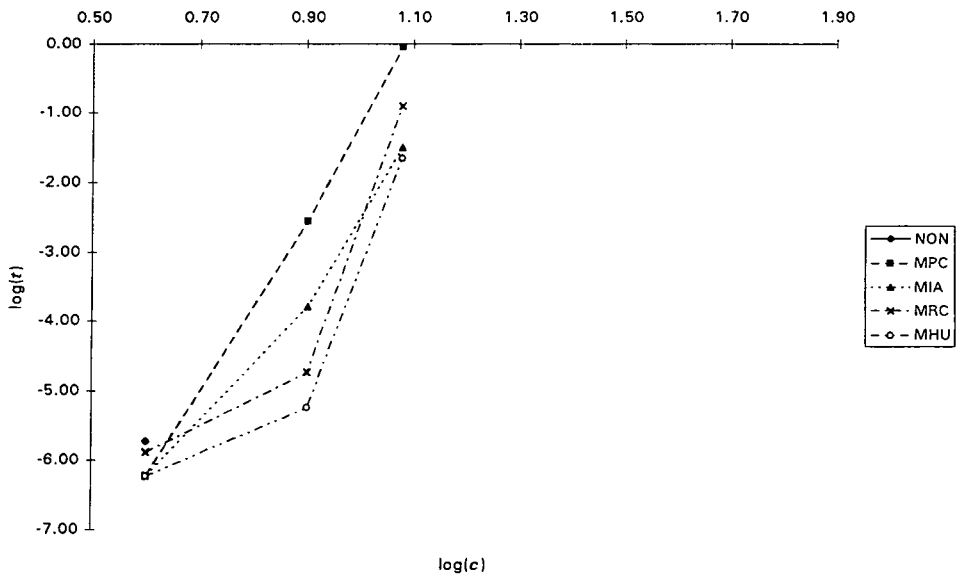

Figure 9-10: AST–FWE run-time for tree-type taskforces on a 2-DSP M-P.

9.3.3 State-Space Complexity

While algorithm run-time is one method of comparison, this approach favours those heuristics which are simple to enumerate. Here, they are compared in terms of state-space complexity.

9.3.3.1 Bus-Configured Taskforces

As the results displayed in Figure 9-11 show, the behaviour of the AST heuristics in *DCS* are markedly different when considered in terms of state-space complexity. Although MIA reveals similar unfavourable behaviour as before, MRC and MHU generate exactly the same number of nodes. Both limit state-space expansion equally well when scheduling bus-type taskforces.

Evidently the computation of `ast_hrt_val(...)` to evaluate Equation (8-9) is directly responsible for the difference in run-times noted in Section 9.3.2.2. NON and MPC

1. In effect, an uninformed search based on dynamic programming.

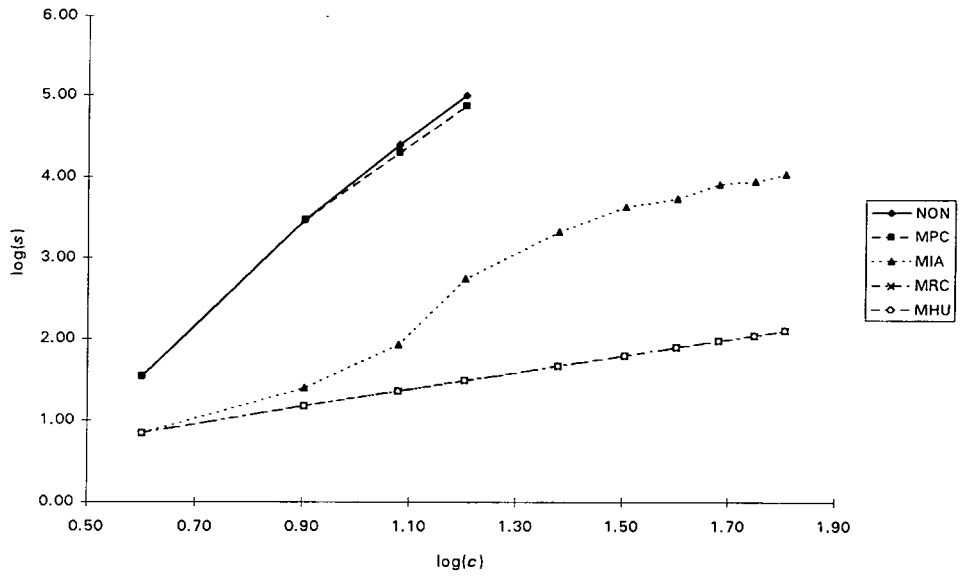

Figure 9-11: AST–FWE state-space for bus-type taskforces on a 2-DSP M-P.

generate in the order of 10^5 nodes to schedule 16-channel consoles on the 2-PE architecture used here, though the rate of increase is notably less than that apparent in Figure 9-9.

9.3.3.2 Tree-Configured Taskforces

Limited results can be produced using AST to schedule taskforces based on tree configurations yet trends similar to those reported in Section 9.3.2.2 are observed. However, instead of MIA lying 9.09% above MHU for $\log c = 1.08$ as shown in Figure 9-12, both heuristics generate the same number of nodes. While not completely conclusive, these results suggest that a judicious combination of MIA and MRC may lead to an effective non-enumerative procedure.

9.4 Performance of CPM

Results are reported and compared for the labelling schemes developed in Section 8.4.3, for instances where all IPC is fully overlapped with task execution — FWE. The objective here is to ascertain which of these static schemes prove most suited to the DMC problem domain.

9.4.1 Algorithm Run-Time

Since console taskforces may vary between linear constructs and tree-based structures with a high degree of inherent parallelism, *DCS* run-time is reported for both ends of this spectrum.

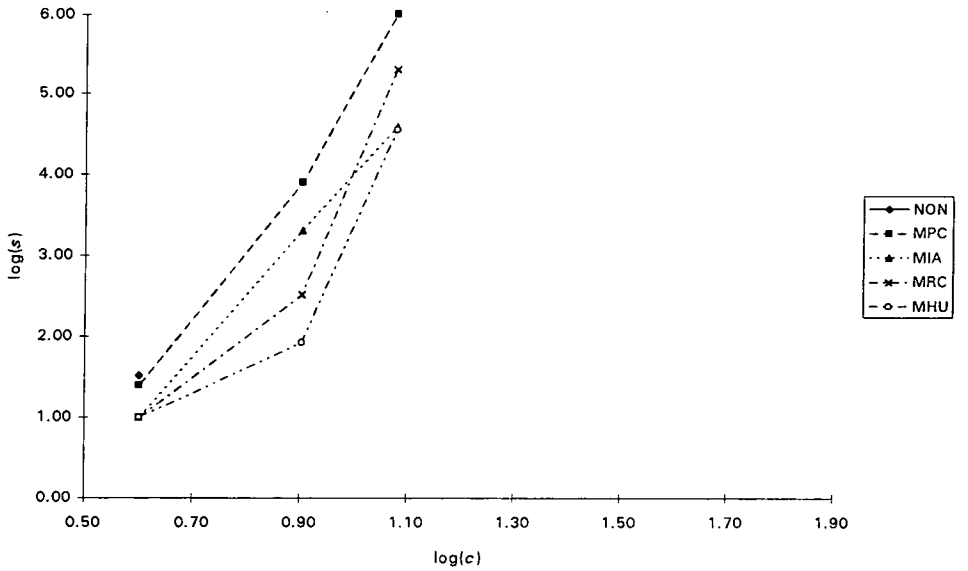

Figure 9-12: AST–FWE state-space for tree-type taskforces on a 2-DSP M-P.

9.4.1.1 Bus-Configured Taskforces

As illustrated in Figure 9-13, random labelling (RND) proves as effective as any of the 'true' labelling schemes. This rather anomalous behaviour is explained by the very nature of the CPM algorithm. Since the conventional mix-bus is heavily constrained by precedence, only one task is ready at each scheduling step. No advantage is gained by ordering the priority list in advance using `utl_srt_prl(...)`: this process simply adds to computation overhead.

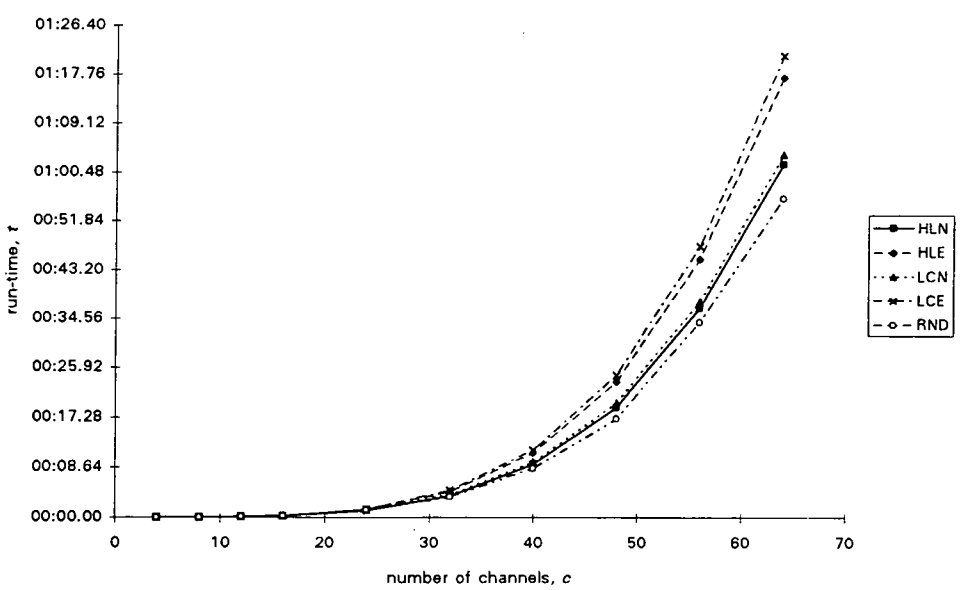

Figure 9-13: CPM–FWE run-time for bus-type taskforces on an 8-DSP M-P.

HLN and LCN are slightly disparate due to the conversion of task co-levels in `utl_cnv_lvl(...)`. HLE and LCE extend run-time further as retrieving $e(T_i)$ requires $O(m)$ database queries. Figure 9-14 re-considers these results from a log-log perspective. Ignoring implementation issues, $\log t$ is linear with product-moment correlation > 0.99925. If c represents the number of channels, linear regression gives: $\log t_{\mathrm{HLE}} = 3.866 \log c - 10.07$.

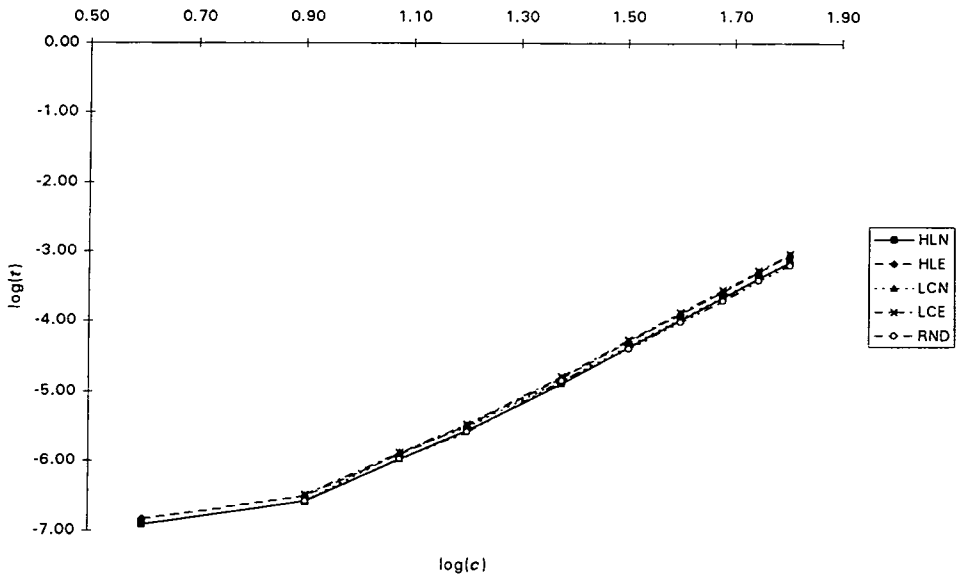

Figure 9-14: CPM–FWE log (t) for bus-type taskforces on an 8-DSP M-P.

9.4.1.2 Tree-Configured Taskforces

Repeating this analysis using taskforces configured with mix-trees results in rather different behaviour, however. From Figure 9-15, it is evident that HLE is extremely effective particularly when compared with HLN and RND schemes. Replotting these experimental results on log-log axes yields piece-wise linear characteristics for the HLE strategy alone.

With LCE and LCN requiring run-times on average 24.15% higher than HLE, the ordering of decreasing performance — HLE, LCN, LCE, HLN, RND — is evident. Clearly, the additional $O(m)$ overhead involved in estimating task execution in `utl_est_exe(...)` returns manifest benefits when task priority is not totally constrained by precedence.

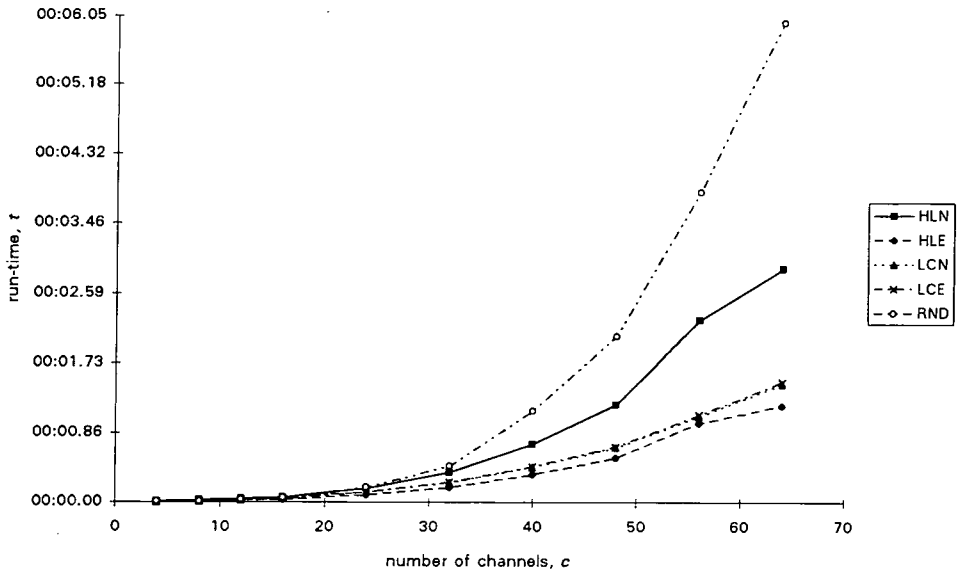

Figure 9-15: CPM–FWE run-time for tree-type taskforces on an 8-DSP M-P.

9.4.2 Schedule Length

Calculating the ratio of schedule ω_{sch} to critical path length ω_{cp} for a range of (normalised) taskforce parallelism π gives the results presented in Figure 9-16. While $\pi \ll 1$, the ideal schedule length ω_{cp} is achieved, but as $\pi \rightarrow 1$, ω_{sch} increases linearly with π. This linear relationship is maintained until $\pi > 1.55$, when the different schemes start to diverge.

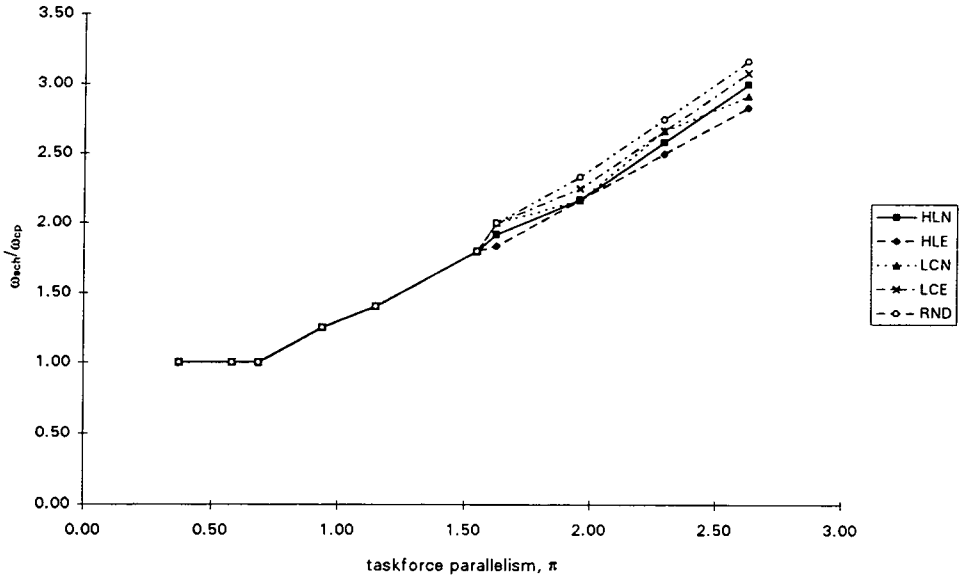

Figure 9-16: CPM–FWE ω_{sch} for tree-type taskforces on an 8-DSP M-P.

HLE consistently returns the shortest ω_{sch} as π increases, surpassing RND by at least 6.86% for $\pi > 1.55$. Interestingly the behaviour of LCN is rather erratic, matching HLE for some values of π and yet identical to LCE for $\pi = 2.29$. Not surprisingly, RND is the worst strategy for minimising ω_{sch} when DMC taskforces contain partial precedence constraints.

9.4.3 Speed-Up

From the speed-up characteristics presented in Figure 9-17, it is plain that HLE returns the best performance under this metric. While HLE is close to linear for small n, it is ultimately constrained by ω_{cp} as n is scaled for the 64-channel taskforce employed here. LCN yields identical performance for $n \leq 12$ and $n \geq 40$, though drops to 93.78% of HLE when $n = 40$.

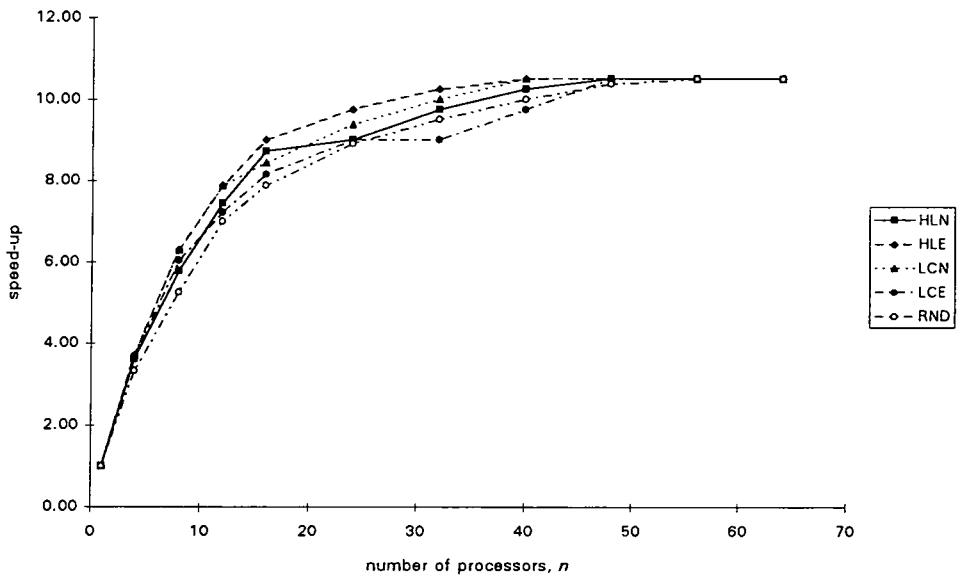

Figure 9-17: CPM–FWE speed-up for tree-type taskforces.

HLN lies below HLE throughout and falls to the level of LCE and RND for $n = 24$. RND is the least suitable labelling scheme for maximising speed-up using CPM, except for $n = 32$ and $n = 40$ when LCE lies 5.26% and 2.50% below RND respectively. Although all five strategies converge when $n = 56$, LCE is as much as 12.20% beneath HLE for $n = 32$.

9.4.4 Scaled Speed-Up

As shown in Figure 9-18, the scaled speed-up measurements fall within one standard deviation for all n. While there is little to separate HLN, LCN and LCE, it is evident that HLE is

again the best technique considered in this investigation. RND is typically the worst strategy, confirming the advantage of tailored labelling techniques. In fact, the characteristic obtained with HLE is up to 21.05% higher than RND for problem instances where $n < 48$.

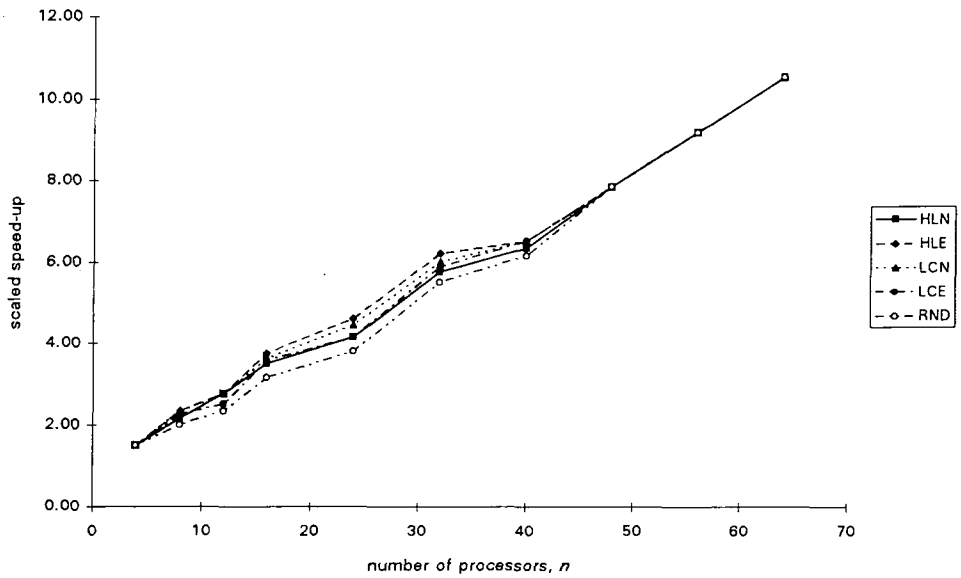

Figure 9-18: CPM–FWE scaled speed-up for tree-type taskforces.

9.5 Conclusions

Aspects of the *DCS* research framework are appropriate to both console design and studio operation. Block-diagram schematics naturally promote the reuse of ASP code and processing hardware, and intuitively describe console layout. Instead of requiring partitioning prior to allocation, LGDF schematics are implicitly partitioned at the level of the ASP task.

The *SilvarLisco* suite has proved convenient for entering data-flow DMC schematics and simulating their behaviour. A parser, translating `HELIX` netlists into `Prolog` clauses, has been written using the *Turbo Prolog Toolbox*.[1] Similar netlist manipulation is employed in the design phase to convert M-P architecture schematics assembled from hardware symbols.

In the production environment, users need not be aware of architectural details or even the number of DSPs. While not a central aspect of this thesis, behaviour models developed in

1. *NTL*, the Netlist Translation Tool.

a regular simulation language have enabled the author to test AST, CPM and DYN algorithms, as well as validate the declarative implementation of NSL, NML and FWE IPC mechanisms.

Internally, the *DCS* engine retains the current state of the partial schedule together with a list of unscheduled tasks. The scheduler predicate hierarchy is described in Section 9.2 from the user's perspective, with reference to the `Prolog` source included as Appendix D. If netlist output is supplied, *DCS* can be interfaced with any schematic-capture package.

Four admissible application-specific heuristics are investigated by comparing the run-time required to obtain an optimal allocation and by observing their ability to prune the A* search-space. Experience shows that the MRC heuristic is most effective at the start of the scheduling process, with MPC & MIA proving more effective as the schedule nears completion.

While MPC may be effective towards the end of the process, this underestimate readily causes AST to revert to exhaustive search. Rather better results are reported for MRC and the combined MHU strategy. Although both generate comparable search-spaces, MHU requires significant run-time uplift as three separate $h(n)$ must be determined for each new node n.

One aim of this study was to determine if the additional complexity of MIA is justified by a significant reduction in the size of the search-space s. With tree-configured taskforces, MIA has a definite run-time advantage over MRC as $\omega_{cp} \ll \omega_{seq}$, and $h(n) \ll h^*(n)$ for MRC as a result. Both heuristics limit state-space expansion equally well as $\omega_{cp} \rightarrow \omega_{seq}$.

MIA reduces the size of the state-space search tree to such a degree when compared to MPC that the additional overhead involved in generating nodes is not significant. Whereas, MIA it is more complex to calculate than MRC, the values of MRC can be effectively pre-computed without prior knowledge of the current state of the partial schedule.

Work in Section 8.1.4 has determined that an efficient DCS algorithm is one which requires an amount of time bounded in the input length l by some polynomial. The essentially $O(n^m)$ requirements of enumerative procedures moves the focus of this research towards heuristic approaches, especially methods derived from MRC & MIA underestimates.

Using the *DCS* research framework with FWE IPC mechanisms, random-labelling (RND) and the critical path length ω_{cp} help determine which, if any, static labelling scheme is

well-suited to the DMC problem. Rather than make this comparison with randomly generated taskforces, the author again extracts precedence DAGs from real DMC layouts.

With bus-configured taskforces, no labelling scheme is more effective than simply assigning labels randomly (RND). Since $\omega_{cp} = \omega_{seq}$, only one ready task exists at any stage in the scheduling process: no improvement is possible by ordering the priority list in advance. Estimating task execution time in HLE and LCE only serves to magnify this effect further.

HLE returns manifest benefits when task priority is not fully constrained by precedence as is the case is a console LGDF displaying *in-tree* precedence relations. When the sequential schedule ω_{seq} is much greater than the length of the critical path, ω_{cp}, the following order of decreasing performance is clearly evident from Figure 9-15 — HCE, LCN, LCE, HLN, RND.

As optimal schedules cannot easily be determined, the ratio of schedule to critical path length, ω_{sch}/ω_{cp}, allows labelling schemes to be compared in terms of ω. π has a direct bearing on ω_{sch} as $n < \pi$ signifies that inherent parallelism will not be fully exploited. With HLE consistently returning the shortest schedule length — HCE, LCN, LCE, HLN, RND.

In terms of speed-up, HLE surpasses other strategies although LCN is as good for small n: HLE, LCN, HLN, RND, LCE prior to convergence. With up to 17.39% separating HLE and RND schemes in Section 9.4.4, the ordering — HLE, LCN, LCE, HLN, RND — emerges. SSU curves intersect those for speed-up at $n = 1$ and $n = m$, as predicted in Section 7.2.4.[1]

CPM–HLE is well inside the theoretical upper bound of Equation (8-17). One flaw in this preferred approach is anomalous behaviour which is a direct result of the static nature of the labelling process. Ready tasks are scheduled onto the earliest available processor, with no regard to the partial schedule. Rectifying this handicap is the main focus of the next chapter.

9.6 References

[Graham, 1969] Graham R L: "Bounds on Certain Multiprocessing Anomalies", *SIAM Journal of Applied Mathematics*, Vol. 17, No. 2, pp. 416-429, March 1969.

[Motorola, 1989] Motorola: *DSP56000/DSP56001 Digital Signal Processor User's Manual*, Revision 2.0, Motorola Inc., 1989.

1. As $S(n) > S'(n)$ for $1 < n < m$, the ratio of n-fold parallel to sequential work must increase with n.

Chapter 10

Dynamic Labelling Schemes

The tendency for AST to revert to an $O(n^m)$ enumerative procedure motivates further research into heuristic approaches. Unfortunately, the static nature of the CPM labelling process is directly responsible for anomalous behaviour. Together, these observations suggest that predetermined task levels should be revised throughout the DMC scheduling process.

10.1 Motivation

Before investigating dynamic schemes within the structure of Section 9.4, the author specifies three alternative strategies and defines an upper bound on schedule quality. Demonstrating 'processor thrashing' illustrates some of the advantages engendered by dynamic techniques.

10.1.1 Alternative Labelling Schemes

Here, the concept of dynamic-level labelling is defined and the four alternative strategies for task and processor selection, previously introduced in Section 9.2.3, are described in detail.

10.1.1.1 Definitions

From Section 8.4.2, $sl(T_i)$ is the HLE static-level of task T_i with $ft(P_j)$ symbolizing the finishing-time of P_j in the current partial schedule. If $ia(T_i, P_j)$ represents the earliest time at which all IPC required by T_i is available on P_j, then dynamic level may be defined as:

$$dl(T_i, P_j) = sl(T_i) - \max(ft(P_j), ia(T_i, P_j)) \qquad \text{Eq. (10-1)}$$

This expression combines aspects of both MIA and MRC AST heuristics in order to assign highest priority to that (T_i, P_j) incurring minimum extension of the current partial schedule.

10.1.1.2 None — NON

NON, effectively a HLE statically-labelled CPM scheme, is included so that implementation-specific issues do not detract from the conclusions presented in this chapter. `dyn_clc_lvl(s_dyn_non, ...)` simply retrieves $sl(T_r)$ from the internal database and deterministic generator predicates insert *static* levels into the DYN priority mechanism.

10.1.1.3 Select Available Processor First — PRO

In this scheme, the available processor P_a with the earliest finishing-time $ft(P_a)$ is first selected by `lst_del_elm(Proc, ...)` and `lst_del_ndt(Task, ...)` generates dynamic levels for all ready tasks T_r. `dyn_ins_dls(s_dyn_pro, ...)` inserts each $dl(T_r, P_a)$ with `dyn_sch_nxt(...)` scheduling (T_r, P_a) with maximal $dl(\ )$.

10.1.1.4 Select Ready Task First — TSK

Alternatively, `lst_elm_del(Task, ...)` selects the ready task T_r with the highest static-level $sl(T_r)$ with the non-deterministic `lst_del_ndt(Task, ...)` generating dynamic levels for all available processors P_a. Again, `dyn_ins_dls(s_dyn_tsk, ...)` inserts each $dl(T_r, P_a)$ and `dyn_sch_nxt(Dyn, ...)` schedules (T_r, P_a) as Figure 10-1 below.

10.1.1.5 All Combinations — ALL

Finally, `dyn_gen_lvl(s_dyn_all, ...)` calls the non-deterministic generator predicates `lst_del_ndt(Task, ...)` and `lst_del_ndt(Proc, ...)` to construct all possible (T_r, P_a). With `dyn_ins_dls(s_dyn_all, ...)` inserting each $dl(T_r, P_a)$ as before, `dyn_sch_nxt(Dyn, ...)` schedules T_r on P_a such that the dynamic level is maximised.

10.1.2 Upper Bounds on Schedule Quality

In the preceding discussions in Section 8.4.4 it was assumed that the priority list remains static during the scheduling process. Dynamic labelling schemes seek to redefine the list every time a PE becomes free. If ω_{dyn} is the finishing time for a taskforce scheduled in this manner and ω_{opt} the optimal finishing time then, using the general bound cited earlier:

$$\frac{\omega_{dyn}}{\omega_{opt}} \leq 2 - \frac{1}{n} \qquad \text{Eq. (10-2)}$$

Graham has developed a slightly better best-possible bound given by [Graham, 1976]:

```
// ==============================================================================
// DCS DYN scheduling algorithm
// ==============================================================================

dyn_bgn_alg(...)
{
    // HLE static labelling
    initialise levels to zero;
    this_task = root_task;

    while ( !entire taskforce traversed )
    {
        // calculate HLE level
        new_level = this_task.level + next_task.execution_time;

        // update task level
        if ( new_level > next_task.level )
            next_task.level = new_level;
    }

    // create priority list
    quick-sort task-list by non-increasing level;

    // schedule taskforce
    while ( priority list != [] )
    {
        // TSK dynamic labelling
        delete all dynamic levels

        for ( this_task = first_task; task <= last_task; i++ )
        {
            if ( !this_task has unfinished predecessors )
            {
                for ( processor = first_processor; processor <= last_processor; j++ )
                {
                    if ( this_processor has sufficient resources )
                    {
                        // calculate TSK level
                        dyn_level = this_task.level - max( this_processor.finish_time,
                                                           all IPC data available    );
                    }
                }
            }
        }

        // schedule ready task on available processor
        schedule task-processor with maximum dyn_level;
    }
}

// ==============================================================================
```

Figure 10-1: Pseudo-code of *DCS* DYN algorithm, using HLE & TSK labelling schemes.

$$\frac{\omega_{dyn}}{\omega_{opt}} \leq 2 - \frac{2}{n-1} \qquad \text{Eq. (10-3)}$$

An alternative to this approach is to assign the task whose execution time plus the execution time of all its successors is maximal. If a set of tasks executed in this manner has a finishing time of ω_m, then ω_m / ω_{opt} is also bounded by the inequality established in Equation (10-2).

10.1.3 Processor Thrashing

Scheduling the EQ section from channel 3 in Figure 9-6 on a 2-DSP M-P illustrates the advantages of dynamic labelling in isolation, particularly with regard to processor thrashing.

10.1.3.1 Static Taskforce Labelling

With DYN–FWE–HLE–NON, T_1 is the first task to be scheduled since $sl(T_1) = 54$. When P_1 is replaced in the processor list, it must be placed after P_2 as $ft(P_2) = 0$. T_3 is selected as the next ready task and scheduled onto P_2. By definition, DYN–FWE–HLE–NON will continue thrashing in this manner until the optimal schedule illustrated in Figure 10-2 is produced.

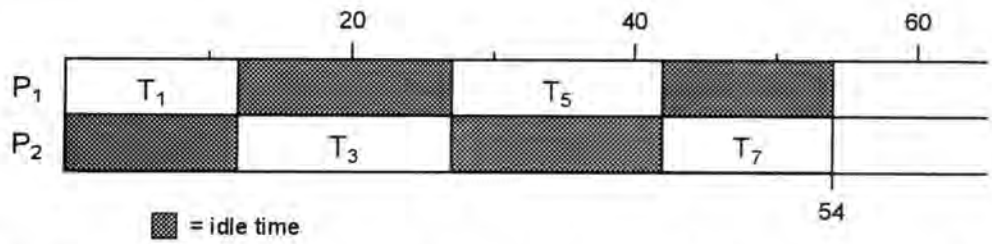

Figure 10-2: DYN–FWE–HLE–NON schedule for 4-stage EQ section on a 2-DSP M-P.

Scheduling the same taskforce using DYN–NSL–HLE–NON produces the schedule of Figure 10-3.[1] The explanation of the thrashing exhibited here is similar to that given above, except that IPC must be allocated before T_3, T_5 and T_7 can be scheduled. Indeed, thrashing is directly responsible for the sub-optimality of this essentially CPM–NSL–HLE strategy.

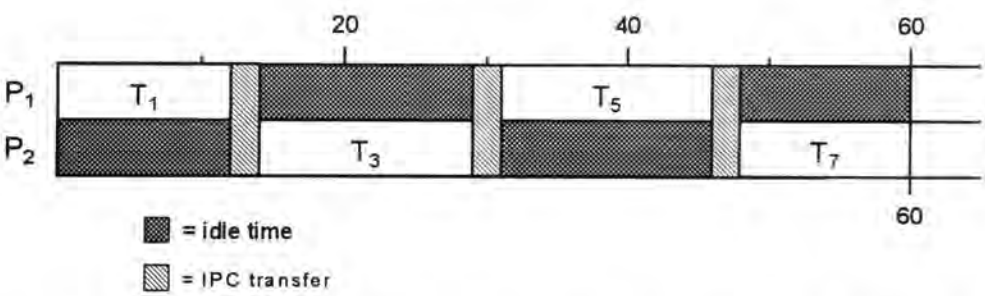

Figure 10-3: DYN–NSL–HLE–NON schedule for 4-stage EQ section on a 2-DSP M-P.

10.1.3.2 Dynamic Taskforce Labelling

Selecting P_a first results in the same schedules as above, whereas DYN–NSL–HLE–TSK is rather better. After T_1 is scheduled on P_1, $sl(T_3) = 42$, $ft(P_1) = 12$ and $ft(P_2) = 0$:

$$dl(T_3, P_1) = 42 - \max(12, 12) = 30 \qquad \text{Eq. (10-4)}$$

$$dl(T_3, P_2) = 42 - \max(12, 14) = 28 \qquad \text{Eq. (10-5)}$$

1. Of course, the same is true of NML IPC since only one concurrent inter-task message is required here.

ensuring that T_3 is also scheduled on P_1 as illustrated in Figure 10-4. From a performance perspective, it is apparent that $O(nm)$ dynamic levels will be created by this procedure.

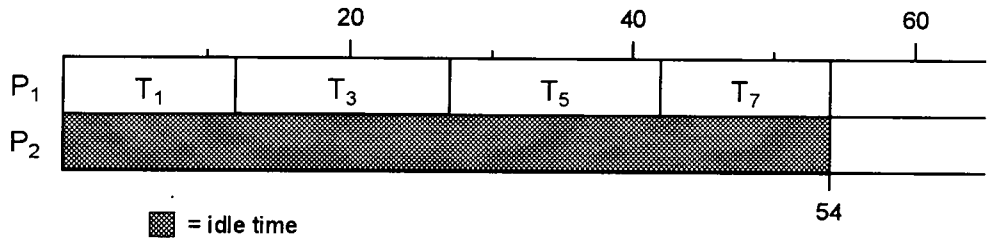

Figure 10-4: DYN–NSL–HLE–TSK schedule for 4-stage EQ section on a 2-DSP M-P.

Now $sl(T_5) = 27$, $ft(P_1) = 27$ and $ft(P_2) = 0$ giving $dl(T_5, P_1) = 0$ and $dl(T_5, P_2) = -2$, so that T_5 is also scheduled on P_1. Finally, T_7 is allocated onto P_1, minimising ω_{sch} and avoiding thrashing. DYN–NSL–HLE–ALL gives the same schedule as above, but not without significant run-time uplift as $O(nm^2)$ levels must be generated.

10.2 Performance with FWE

In the rest of this chapter the aim is to determine which of the dynamic labelling schemes, detailed in Section 10.1.1, minimises the anomalous behaviour of CPM–xxx–HLE. Here, results are analysed for instances where all IPC is fully overlapped with task execution — FWE.

10.2.1 Algorithm Run-Time

DMC taskforces may vary from bus- to tree-based structures with a high degree of inherent parallelism. Consequently, *DCS* run-time is examined at both extremes of this spectrum.

10.2.1.1 Bus-Configured Taskforces

With FWE IPC mechanisms, a rather interesting pattern emerges when algorithm run-time t is plotted as Figure 10-5. PRO and ALL labelling schemes are almost identical with $t > 3{:}00.00$. `dyn_gen_lvl(...)` must generate $O(m^2)$ dynamic levels $dl(T_r, P_a)$ for PRO and $O(nm^2)$ for ALL. When $m \gg n$, as is the case here, these run-time complexities are effectively equal.

By definition, only one T_r is considered in NON and TSK schemes, reducing algorithm run-time to $O(m)$ and $O(nm)$ respectively. Re-plotting Figure 10-5 on log-log axes

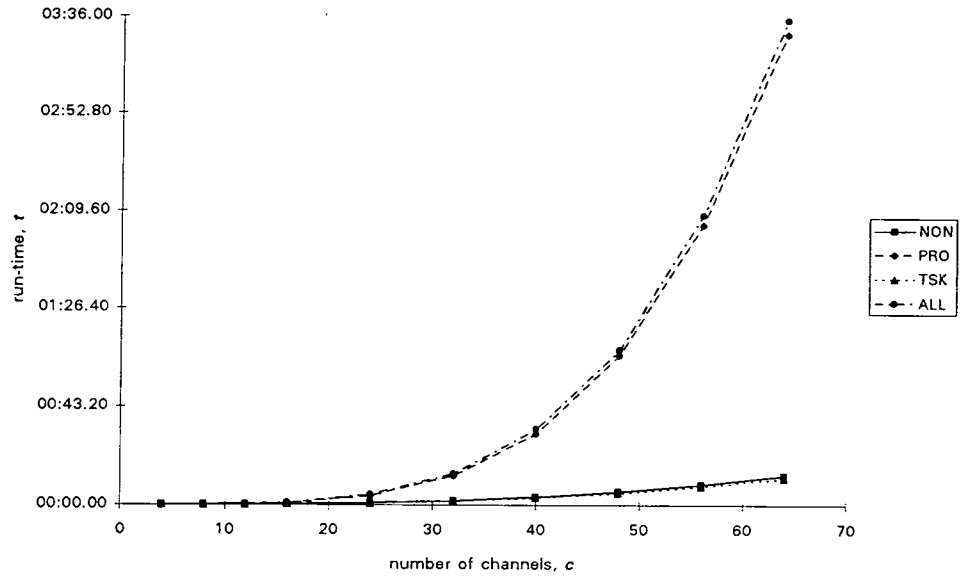

Figure 10-5: DYN–FWE–HLE run-time for bus-type taskforces on a 16-DSP M-P.

demonstrates the relationship previously noted in Section 9.4.1. Linear regression gives run-time for the TSK dynamic strategy as $\log t_{TSK} = 2.597 \log c - 8.597$ with $\rho = 0.9992$.

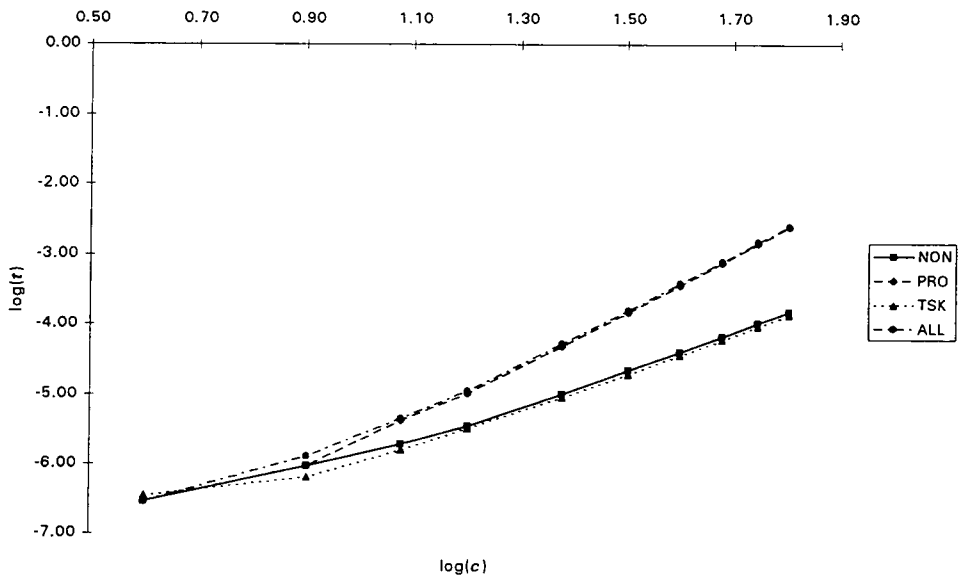

Figure 10-6: DYN–FWE–HLE $\log t$ for bus-type taskforces on a 16-DSP M-P.

10.2.1.2 Tree-Configured Taskforces

With tree-configured mix-buses, NON and TSK labelling schemes prove to be efficient methods for producing DMC schedules, giving run-times in the order of seconds. Although

PRO will generate $O(m)$ ready-task, available-processor pairs for the available processor P_a with lowest index at each of $O(m)$ scheduling steps, run-times are still under 1:00.00.

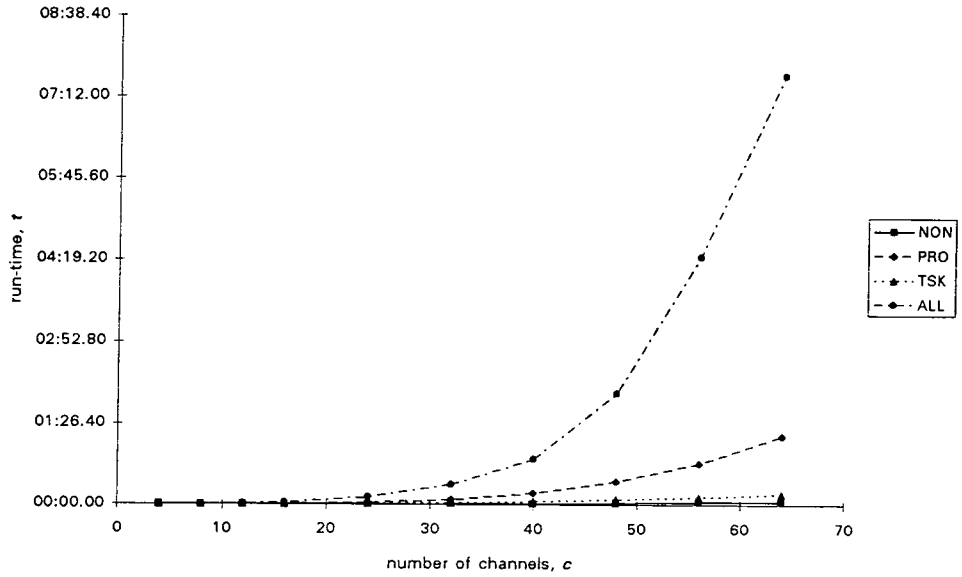

Figure 10-7: DYN–FWE–HLE run-time for tree-type taskforces on a 16-DSP M-P.

This behaviour of PRO is a substantial improvement over the results reported in the previous section. Since all (T_r, P_a) are generated in ALL, this scheme rapidly becomes $O(nm^2)$ with run-times in excess of 8:00.00: over 8-times longer than PRO. Replotting results on log-log axes as shown in Figure 10-8 gives almost 'linear' characteristics for all 4 schemes.

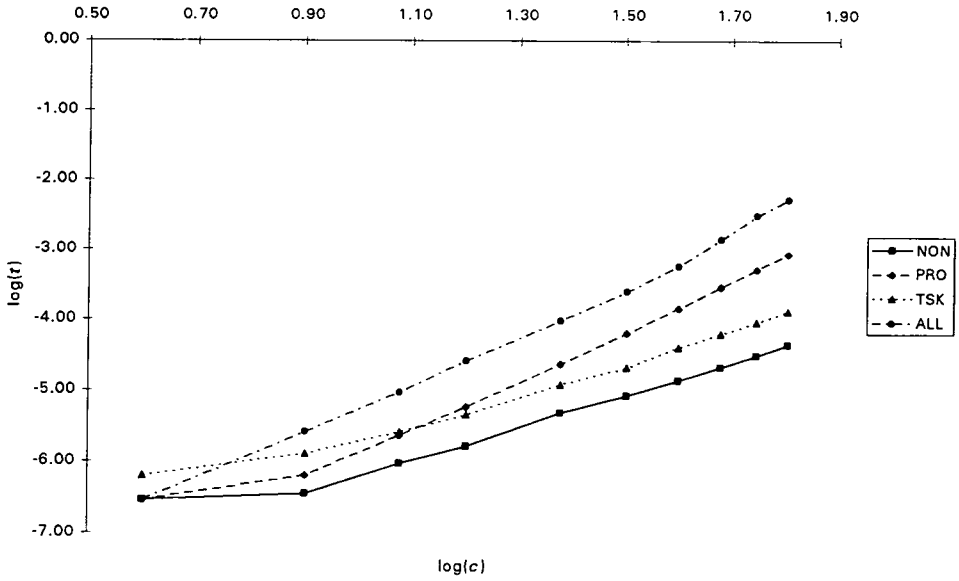

Figure 10-8: DYN–FWE–HLE log$t$ for tree-type taskforces on a 16-DSP M-P.

10.2.2 Schedule Length

Comparing the ω_{sch}/ω_{cp} ratio for console taskforces with varying degrees of taskforce parallelism, as shown in Figure 10-9, confirms that there is nothing to choose between the schemes until $\pi > 1.5$. As π increases further, NON and PRO consistently give the shortest ω_{sch} with schedules produced by ALL at least 9.29% longer. Interestingly, the TSK scheme switches between the NON/PRO and ALL characteristics as π increases beyond 1.5.

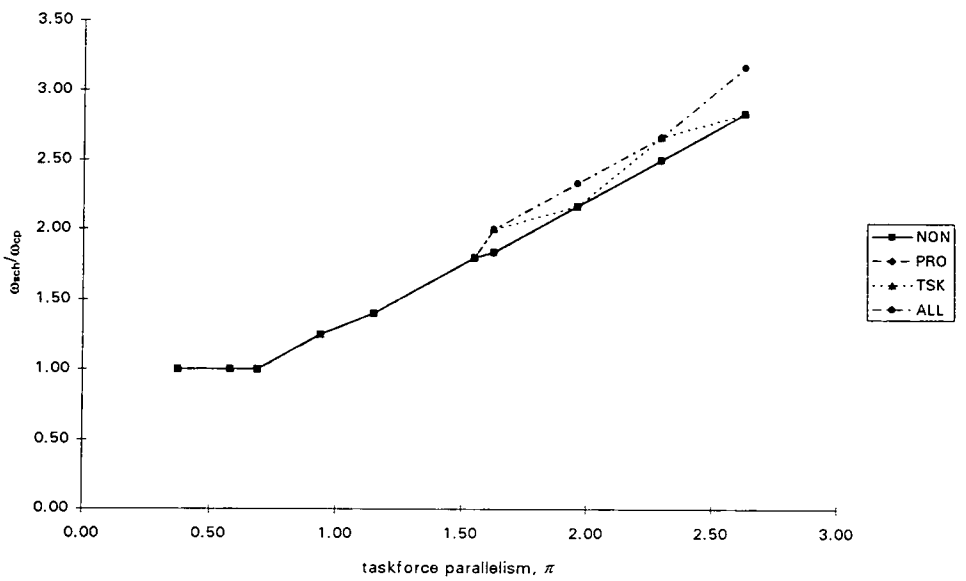

Figure 10-9: DYN–FWE–HLE ω_{sch} for tree-type taskforces on a 16-DSP M-P.

10.2.3 Speed-Up

While different static labelling schemes yield varying performance as regards taskforce speed-up (see Section 9.4.4), Figure 10-10 contains the speed-up characteristics for the four dynamic labelling schemes developed earlier in this thesis. For fully-overlapped (FWE) IPC, all schemes exhibit the same speed-up behaviour such that speed-up is ultimately limited by ω_{cp} when $n \geq 40$ — for the 64-channel tree-configured DMC taskforce considered here.

10.2.4 Scaled Speed-Up

Similarly, different static labelling schemes exhibit varying scaled speed-up performance. As Figure 10-11 shows however, all four dynamic strategies exhibit identical linear, though sub-unitary, scaled speed-up. Note that Figure 10-10 intersects the curve shown below when

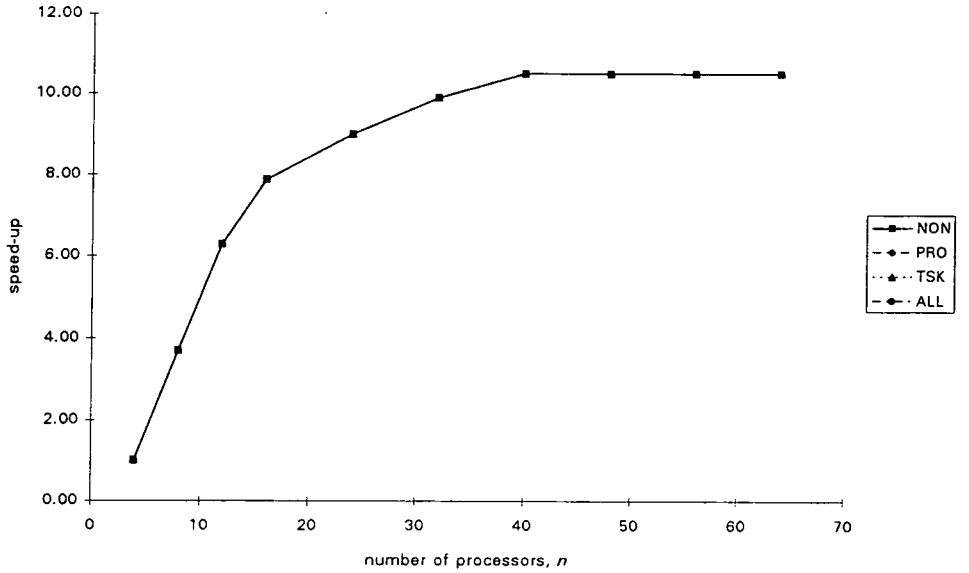

Figure 10-10: DYN–FWE–HLE speed-up for tree-type taskforces.

$n = 1$ and $n = m$ as predicted in the theoretical work presented in Section 7.2.4. This behaviour is rightly attributed to the ratio of parallel to sequential work increasing with n.

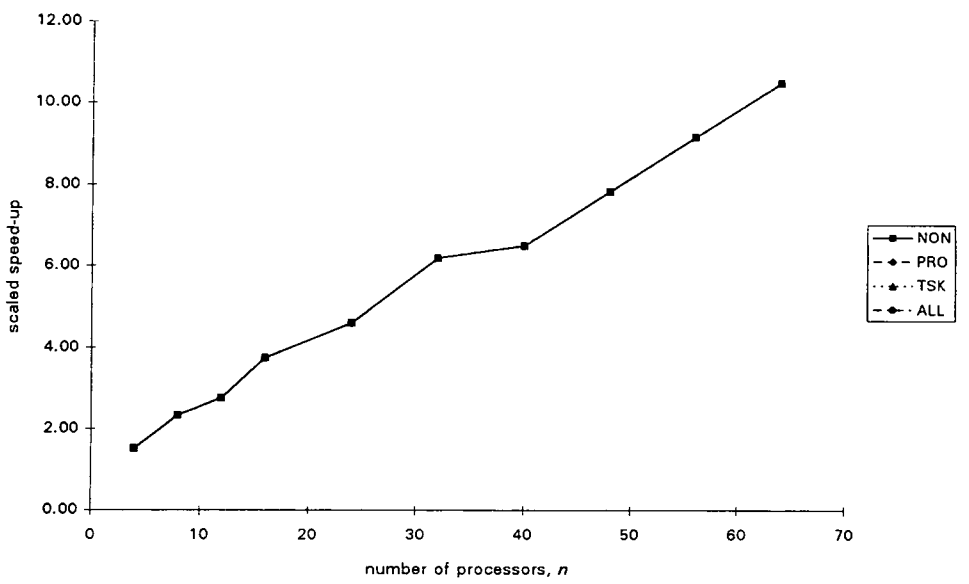

Figure 10-11: DYN–FWE–HLE scaled speed-up for tree-type taskforces.

10.3 Performance with NSL

The performance of a single non-overlapped IPC channel has been studied from a theoretical point of view in Section 7.4.2. In this section, the performance of dynamic labelling schemes is investigated with a individual IPC channel which supports sequential inter-task messaging.

10.3.1 Algorithm Run-Time

With wide variation in run-time characteristics already reported for CPM–FWE–xxx and DYN–FWE–HLE–xxx, here t is investigated for NSL IPC mechanisms with the same DMC taskforces.

10.3.1.1 Bus-Configured Taskforces

DCS run-time for the four dynamic-labelling schemes using NSL is plotted in Figure 10-12. While similar to the results of Section 10.2.1, it is apparent that TSK is preferred with NSL IPC. Although NON gives comparable performance, that TSK selects the first T_r and then generates all (T_r, P_a) pairs ensures that T_r is scheduled on that P_a such that $dl(T_r, P_a)$ is minimal.

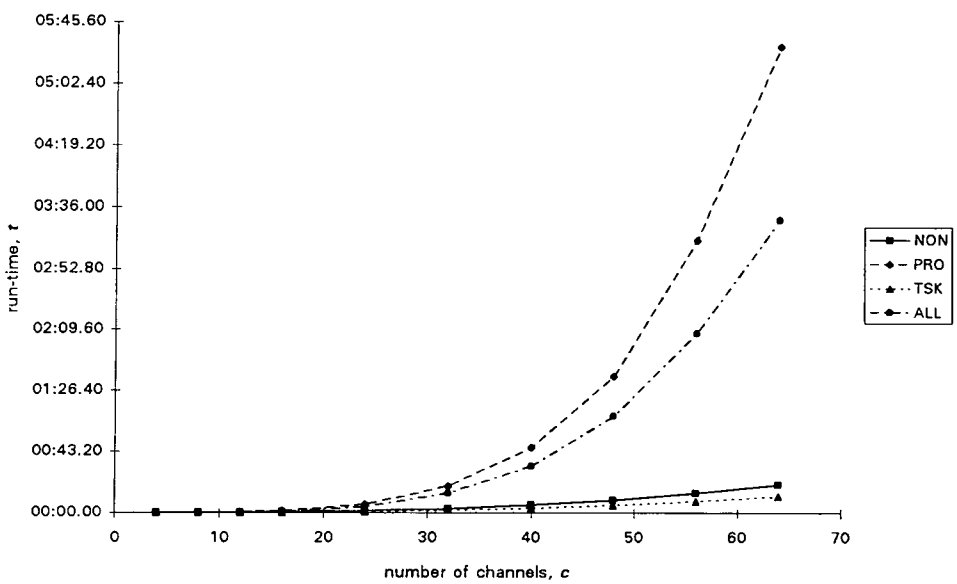

Figure 10-12: DYN–NSL–HLE run-time for bus-type taskforces on a 16-DSP M-P.

Incorporating IPC costs into the *DCS* algorithms evidently improves the choice of processor on which to schedule a given task. The same reasoning also explains why ALL

exhibits similar run-time characteristics for NSL as for FWE IPC detailed in Figure 10-5. Under NSL, PRO is clearly the worst scheme giving run-times well in excess of 5:00.00.

10.3.1.2 Tree-Configured Taskforces

With tree-type console configurations, run-time characteristics are obtained as illustrated in Figure 10-13. These results are almost identical to those presented for FWE IPC mechanisms, except that ALL run-times are reduced by a factor of 0.560. When considered from a log-log perspective, the NSL scheme yields: $\log t_{TSK} = 2.429\log c - 8.240$ with $\rho = 0.9994$, which bears close resemblance to that given previously for fully-overlapping IPC channels.

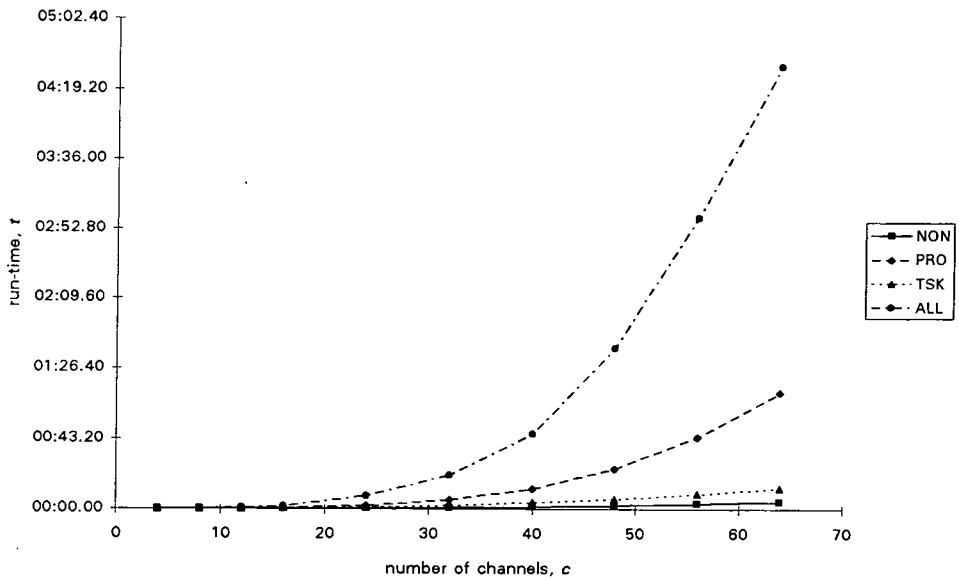

Figure 10-13: DYN–NSL–HLE run-time for tree-type taskforces on a 16-DSP M-P.

10.3.2 Schedule Length

When NSL is considered via a ω_{sch}/ω_{cp} comparison as Figure 10-14, it is clear that IPC directly effects ω_{sch} throughout the entire range of π. Not surprisingly, NON is the worst scheme due to a lack of consideration of the current state of tasks and processors. True dynamic strategies give rather better performance: PRO, TSK and ALL intersect at several points, though TSK and ALL generally give better results for this pessimistic IPC mechanism.

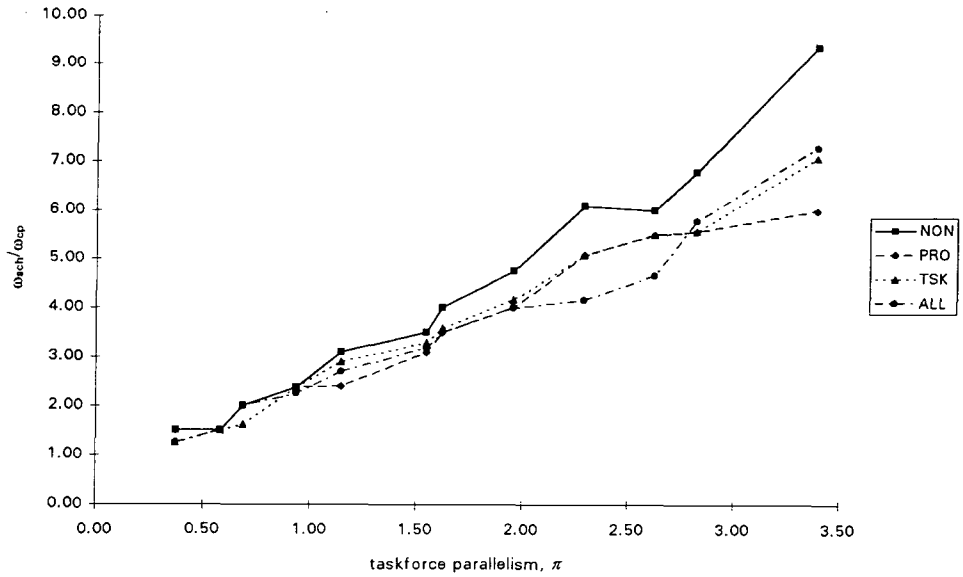

Figure 10-14: DYN–NSL–HLE ω_{sch} for tree-type taskforces on a 16-DSP M-P.

10.3.3 Speed-Up

As Figure 10-15 shows, an unfamiliar pattern emerges when NSL speed-up is plotted for dynamic labelling. Both NON and PRO ultimately converge to a value some 27.04% less than the maximum speed-up determined by ω_{cp}. ALL outstrips all other schemes until it converges at $n = 32$, except for a value of almost 10.00% less than ω_{seq}/ω_{cp} for $n = 24$. In terms of consistent performance, however, TSK is the best strategy to maximise taskforce speed-up.

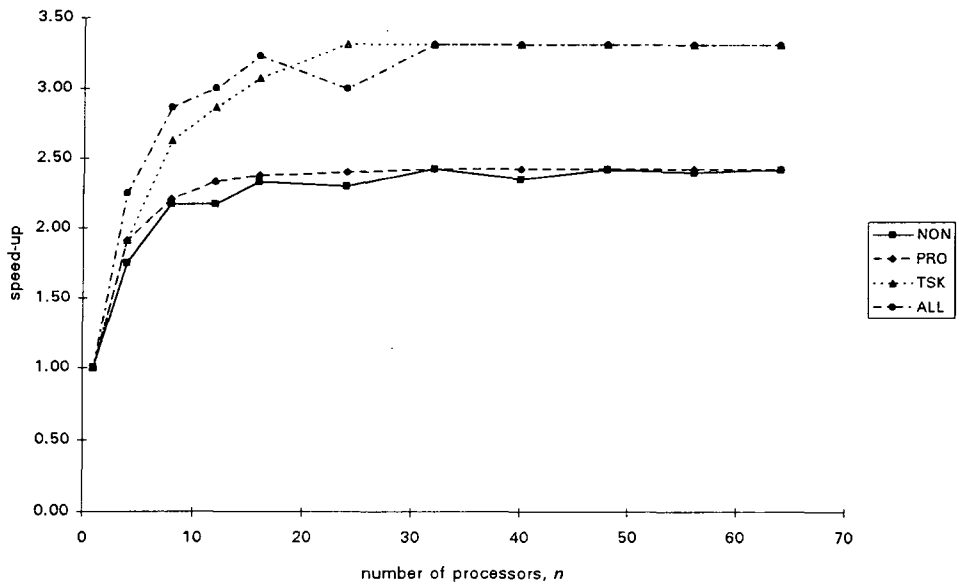

Figure 10-15: DYN–NSL–HLE speed-up for tree-type taskforces.

10.3.4 Scaled Speed-Up

Scaling c as n increases gives the characteristics displayed in Figure 10-16. Both TSK and ALL give results which adhere closely to the theory developed in Section 7.2.4, though TSK never quite converges to the conventional speed-up of 3.24 which ALL reaches when $n = 64$. NON and PRO are very poor from this perspective since NON converges to a value only 57.43% of that achieved by ALL, whereas PRO actually moves away from this asymptote for $n \geq 32$.

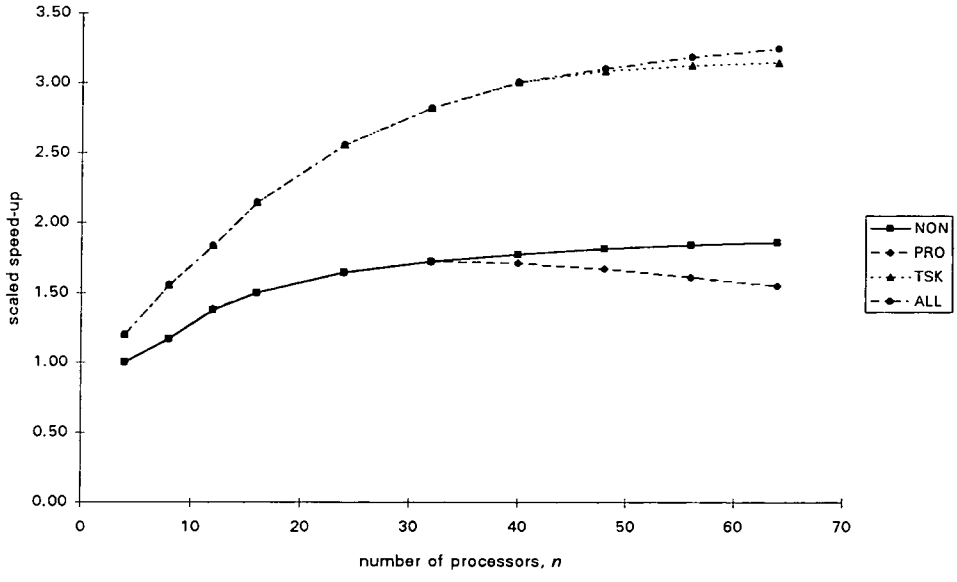

Figure 10-16: DYN–NSL–HLE scaled speed-up for tree-type taskforces.

10.4 Performance with NML

One decisive assumption in NSL is that all IPC proceeds sequentially. It is perfectly possible, as developed for *HYMIPS* in Chapters 5 & 6, to replicate architectural features for IPC. Hence, results are presented for DYN scheduling using multiple non-overlapped IPC channels.

10.4.1 Algorithm Run-Time

As the results in Figure 10-17 show, the run-time performance of dynamic labelling using bus-configured taskforces with NML is virtually identical to that obtained with NSL IPC. When tree-type taskforces are scheduled with NML, Figure 10-18 illustrates that algorithm run-time is again identical to NSL mechanisms. This behaviour can be explained in part by the fact that while reserving IPC is an integral part of DYN, it is ignored in the static labelling process.

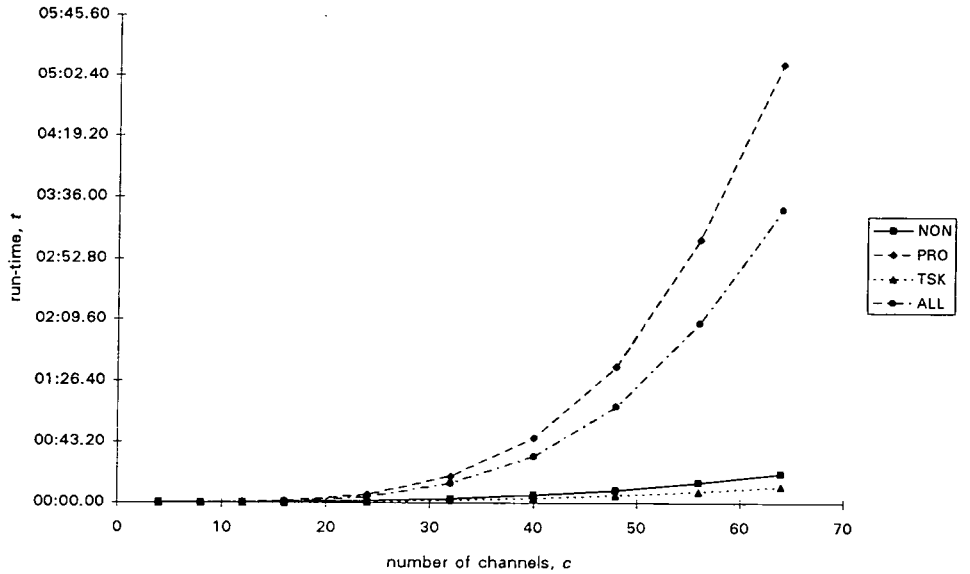

Figure 10-17: DYN–NML–HLE run-time for bus-type taskforces on a 16-DSP M-P.

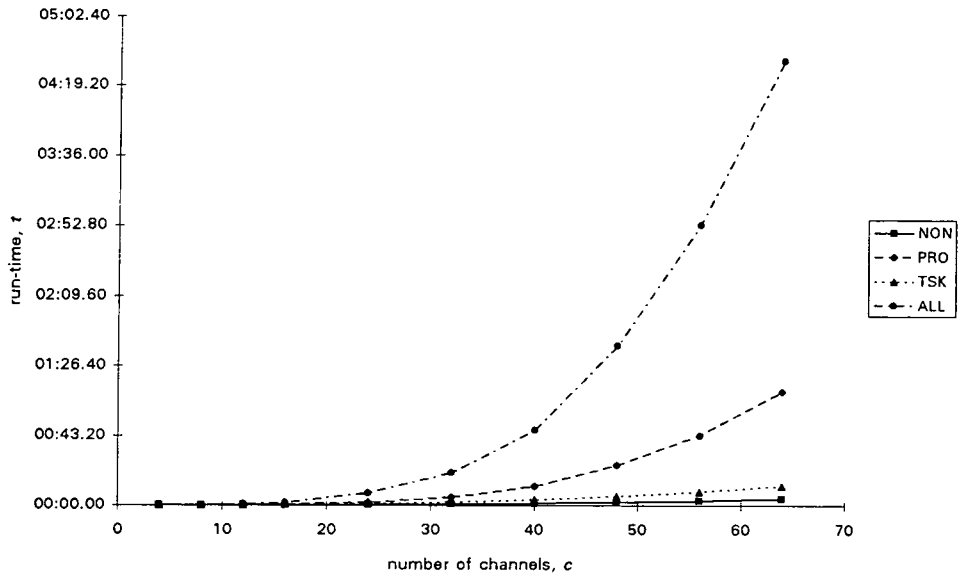

Figure 10-18: DYN–NML–HLE run-time for tree-type taskforces on a 16-DSP M-P.

10.4.2 Schedule Length

Plotting the schedule length to critical path ratio, ω_{sch}/ω_{cp}, against normalised taskforce parallelism, π, gives the graphs for NML included in Figure 10-19. Again the linear trend is apparent especially in NON and PRO labelling schemes. PRO is consistently good at minimising

the ω ratio for the range of π considered here. The performance of TSK and ALL proves to be rather erratic: lying below PRO in a few instances but lying on or above NON for others.

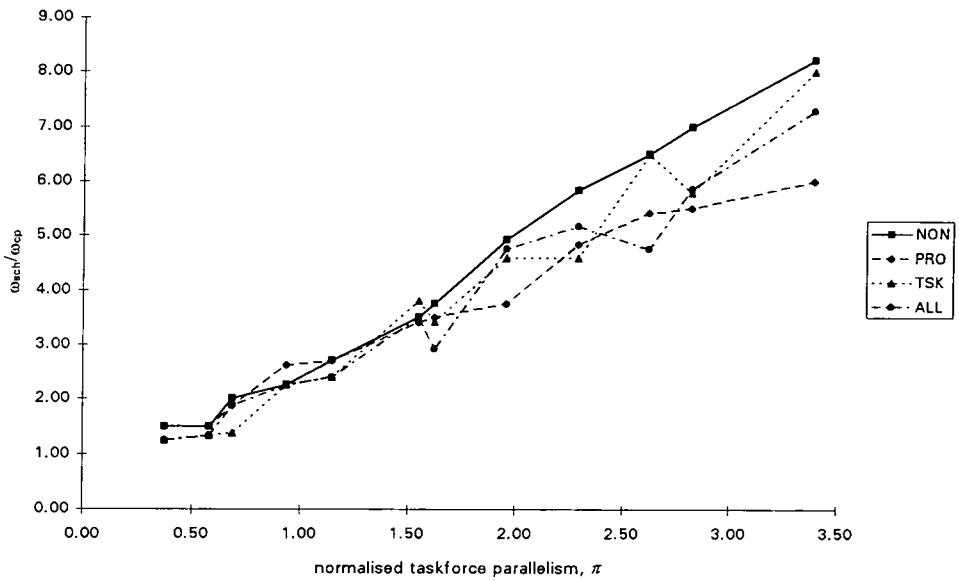

Figure 10-19: DYN–NML–HLE ω_{sch} for bus-type taskforces on a 16-DSP M-P.

10.4.3 Speed-Up

Graphing speed-up for NML gives characteristics not previously seen in this analysis. TSK is precisely linear sub-unitary until speed-up is constrained to 7.41 by ω_{cp}. NON follows TSK for small n, eventually converging to 6.00. This lower value is also met by PRO, which seems to be the poorest labelling scheme as regards speed-up for NML IPC. ALL fluctuates in a similar fashion to TSK, but ultimately converges to the same maximum speed-up as TSK.

10.4.4 Scaled Speed-Up

Figure 10-21 details the scaled speed-up of dynamic strategies with NML. Since no re-labelling is undertaken in NON, the state of the partial schedule is not considered and so gives worst performance. PRO fluctuates about a generally linear trend as n increases, but still lies some 35.52% beneath TSK and ALL when $n = 64$. TSK and ALL arrange IPC overhead in such a way as to give identical linear SSU for the range of M-P architectures considered here.

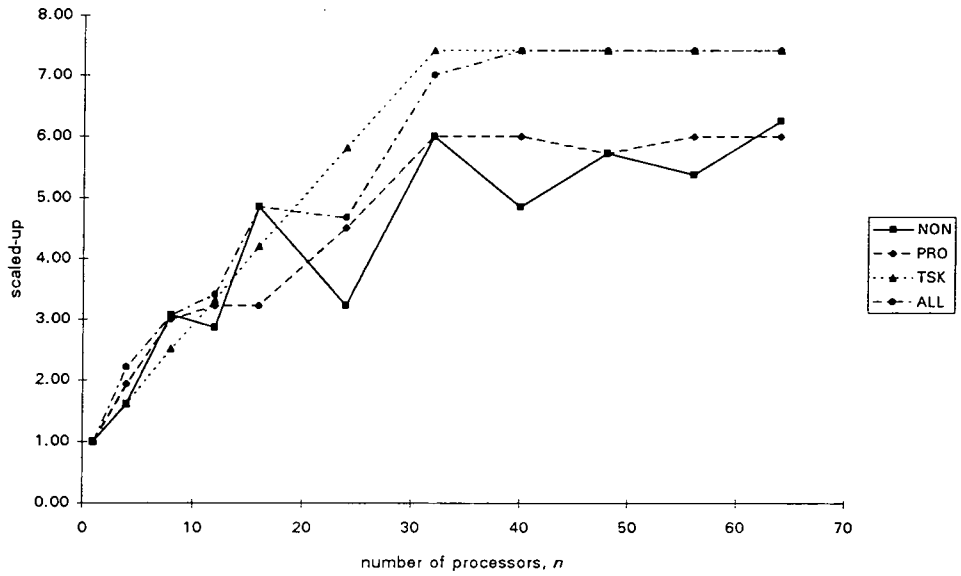

Figure 10-20: DYN–NML–HLE speed-up for bus-type taskforces.

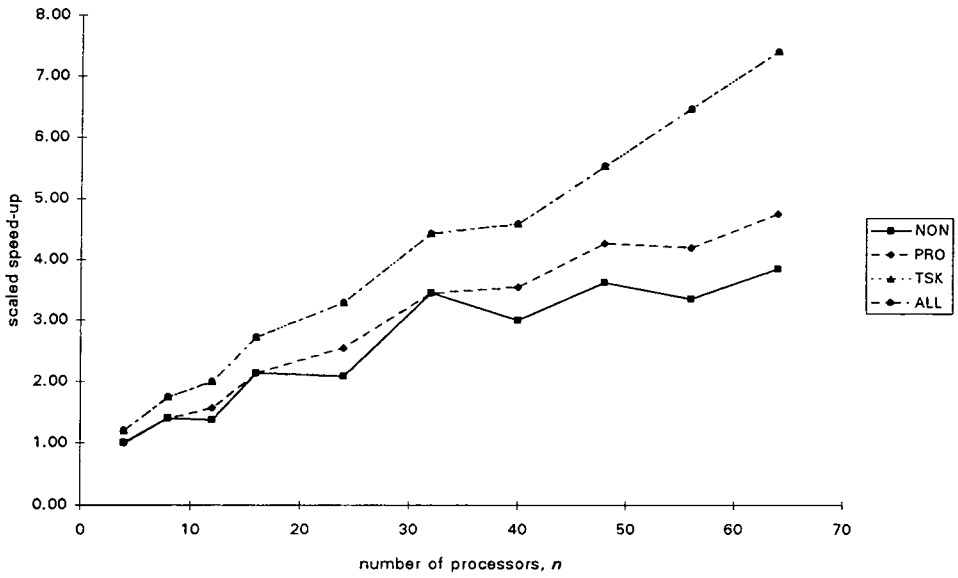

Figure 10-21: DYN–NML–HLE scaled speed-up for bus-type taskforces.

10.5 Task Granularity

M-P system architects have long debated the relative merits of fine and coarse granularity. Accordingly, the performance of DYN–NML–HLE–TSK is measured using DMC taskforces exhibiting a range of task granularity, or r/c, as previously defined in Section 7.4.1.

10.5.1 DMC Implementation

The theoretical investigation in Chapter 7 has shown that DMC depends strongly on r/c. As Figure 10-22 illustrates, r/c is high in coarse-grained parallelism and IPC overhead can be overlapped during many instruction cycles. Conversely, in fine-grained taskforces the overhead per unit computation r/c is relatively low. Many more PEs may be efficiently utilised than in coarse-grained schemes, but not without significant uplift in IPC overhead.

```
; ==================================================================
; Function: MIX_BUS
;       fan_in:1 mix-bus primitive
;       as per Figure 9-2
; Resource Requirements:
;       instruction cycles - (10 + fan_in) cycles
;       P       memory      -  0 words
;       X-data memory       -  0 words
;       Y-date memory       -  0 words
; ==================================================================

MIX_BUS \
    MACRO inputs,params,fan_in,output

        ; ---------------------------------------------------------------
        ; initialise pointer and modulo registers
        ; ---------------------------------------------------------------

        org     p:

        move    #inputs,r0
        move    #fan_in,m0
        move    #params,r4
        move    #fan_in,m4

        ; ---------------------------------------------------------------
        ; perform fan_in:1 mix-bus kernel
        ; ---------------------------------------------------------------

        clr     a
        move                    x:(r0)+,x0      y:(r4)+,y0

        .LOOP   #fan_in-1
                mac             x0,y0,a         x:(r0)+,x0      y:(r4)+,y0
        .ENDL

        macr    x0,y0,a
        move    a,x:output

    ENDM ; MIX_BUS

; ==================================================================
```

Figure 10-22: DSP56001 assembly code for generic DMC mix-bus kernel.[a]

a. Note that audio rate interpolation of control rate data is external to this macro.

The reader's attention is drawn to the allocation of a 16-input DMCs onto a 8-PE M-P detailed in Table 10-1. Dynamics ASP was removed from four of the sixteen channels, allowing this 16:8 taskforce to meet the constraints imposed by f_s [Linton, 1991]. Reducing m and ω_{seq} together by replacing the 2-input mix tasks, $e(\ ) = 12$ [1], with 16-input primitives, $e(\ ) = 26$, can significantly reduce the *DCS* algorithm run-time scheduling overhead.

1. Rather than a generic kernel, tailored code can reduce the cycle count for a 2-input primitive by 41.67%.

Console Taskforce	Sequential Schedule	Parallel Schedule	Algorithm Run-time	Speed-up	Processor Efficiency
16:1	2044	268	0:07.57	7.62	95.31%
16:2	2104	268	0:11.65	7.85	98.13%
16:4	2224	280	0:23.28	7.94	99.30%
16:8[a]	2214	278	1:04.83	7.96	99.48%

Table 10-1: DYN–NML–HLE–TSK schedules for various 16-input consoles on an 8-DSP M-P.
a. Dynamics processing was removed from four channels.

10.5.2 Algorithm Run-Time

Variation in run-time is anticipated due to the number of primitives required to construct a mix-bus. For example, it i is the number of inputs supported and b represents the *fan-in* of each mix primitive (see Figure 9-2), then the number of mix tasks required n_m is given by:[1]

$$n_m = \left\lceil \frac{i-1}{b-1} \right\rceil \qquad \text{Eq. (10-6)}$$

10.5.2.1 Bus-Configured Taskforces

As illustrated in Figure 10-23, there is clear distinction in run-time t when scheduling 64-channel bus-configured consoles constructed from mix primitives of varying granularity. While the 16-input console requires less than 00:01.00 to be scheduled on the 32-DSP topology considered here, almost a minute is required if the taskforce uses 2-input primitives.

Re-plotting these experimental results on logarithmic axes as shown below in Figure 10-24 reveals that all curves are linear in this domain. Performing linear regression on these values gives $\log t_2 = 2.394 \log c - 7.710$ and $\log t_{16} = 0.785 \log c - 7.091$ for 2-input and 16-input primitives, respectively, with product-moment correlation > 0.9878.

10.5.2.2 Tree-Configured Taskforces

Repeating this analysis using taskforces configured with *mix-trees* results in similar characteristics, as Figure 10-25 illustrates. Run-time for 64-channels with 2-input primitives is some 21.78% above that reported in the previous section. Evaluating these results from a

1. Irrespective of whether the i-input mix-bus is bus- or tree-configured.

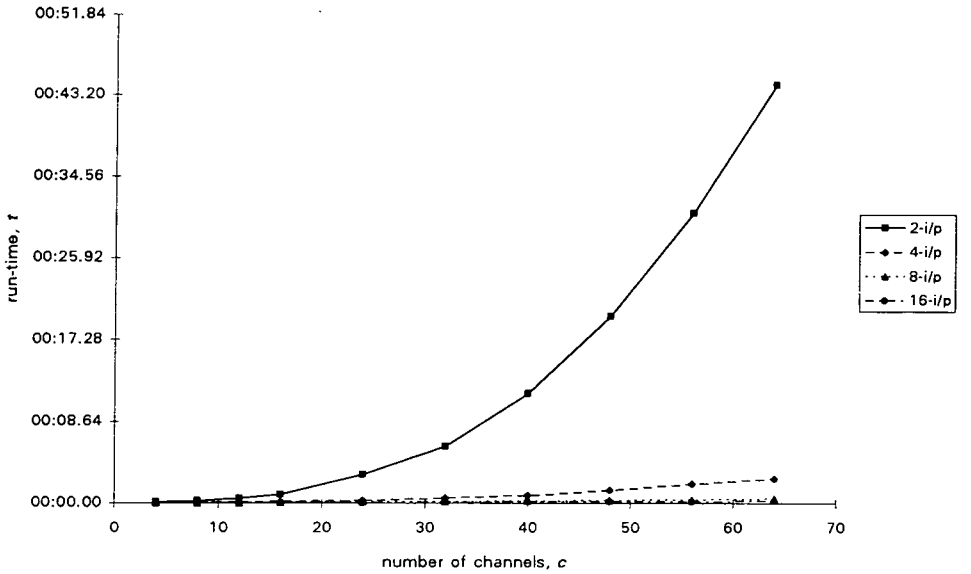

Figure 10-23: DYN–NML–HLE–TSK run-time for bus-type taskforces on a 32-DSP M-P.

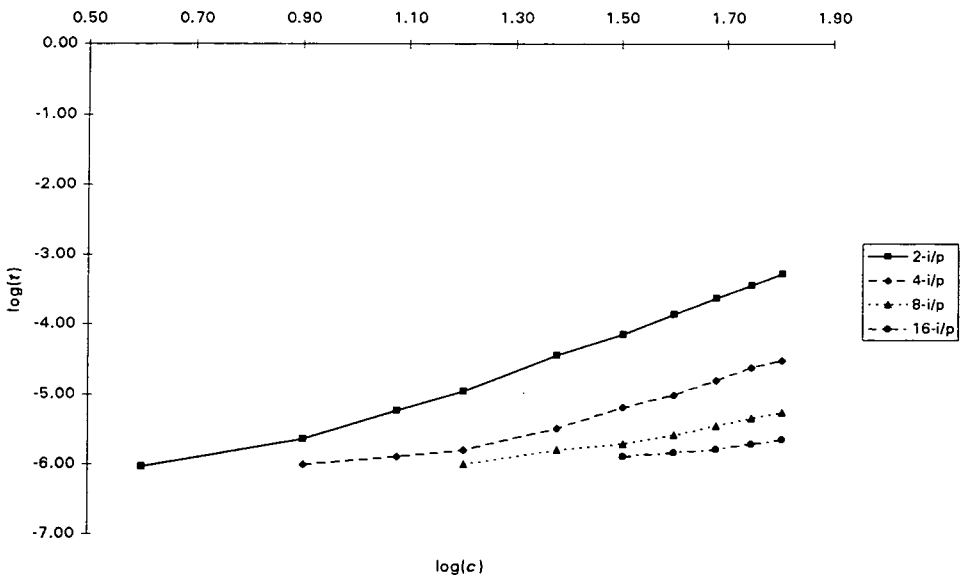

Figure 10-24: DYN–NML–HLE–TSK $\log t$ for bus-type taskforces on a 32-DSP M-P.

logarithmic perspective gives $\log t_2 = 2.286 \log c - 6.728$ and $\log t_{16} = 1.610 \log c - 7.522$, with Pearson product-moment correlation > 0.9918 in both instances.

10.5.3 Schedule Length

Calculating ω_{sch}/ω_{cp} allows these alternative DMC construction techniques to be compared for various values of π. From Figure 10-26, it is evident that employing 16-input primitives is

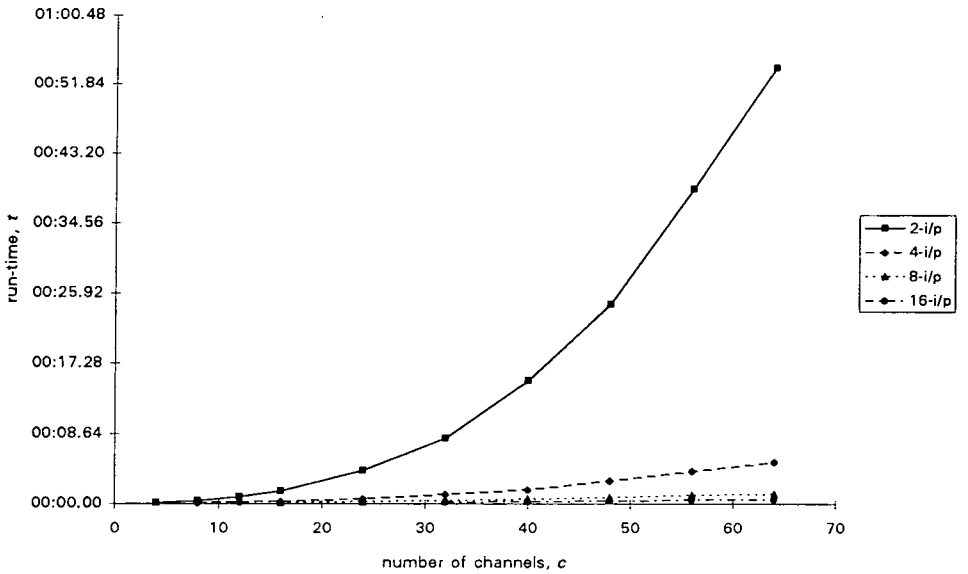

Figure 10-25: DYN–NML–HLE–TSK run-time for tree-type taskforces on a 32-DSP M-P.

the best method of minimising the ω ratio. The results presented in this illustration would tend to suggest that 2-input tasks tend to be more effective than 4- or 8-input alternatives.

Increasing π further, however, shows that this is not the case and the ordering is — 16, 8, 4, 2. It must be remembered that when $\pi = 4.00$, $\omega_{cp} = 18$ and $\omega_{sch}/\omega_{cp} = 1.40$ for 2-input implementations whereas $\omega_{cp} = 36$ and $\omega_{sch}/\omega_{cp} = 1.33$ with 16-input tasks. In other words, the 2-input schedule is 47.50% *less* than that obtained with 16-input primitives.

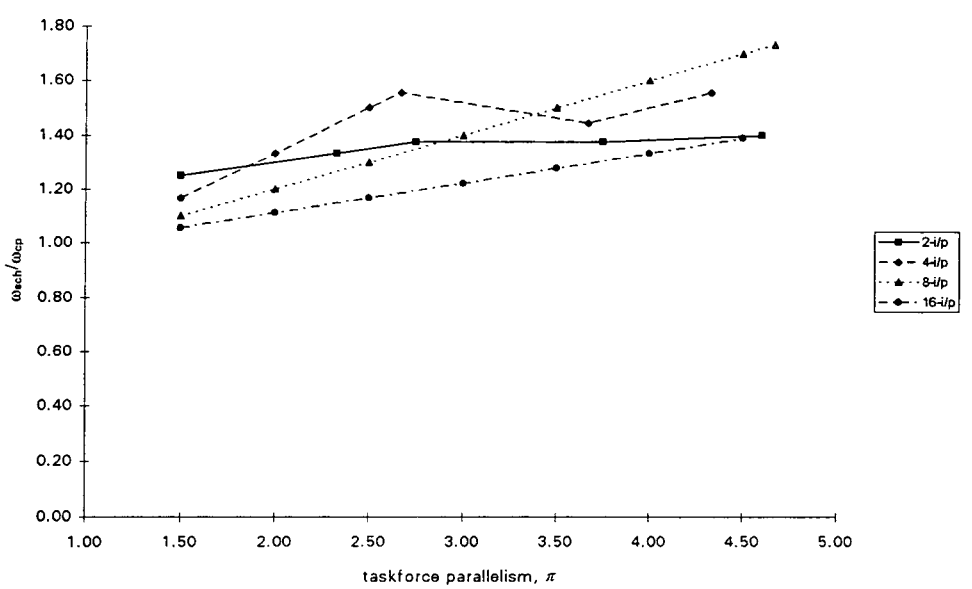

Figure 10-26: DYN–NML–HLE–TSK ω_{sch} for tree-type taskforces on a 32-DSP M-P.

10.5.4 Speed-Up

It is evident from Figure 10-27 that 2-input tasks return the best speed-up performance. While the 2-input curve is close to linear until ω_{sch} reaches ω_{cp} when $n = 48$, this occurs with 4-input primitives when $n = 24$. Similar behaviour is observed for the other cases, except that 8-input converges to the ω_{cp}-constrained value when $n = 12$. The 16-input curve converges when n is as low as 8, such that speed-up is only 28.88% of the 2-input case when $n = 64$.

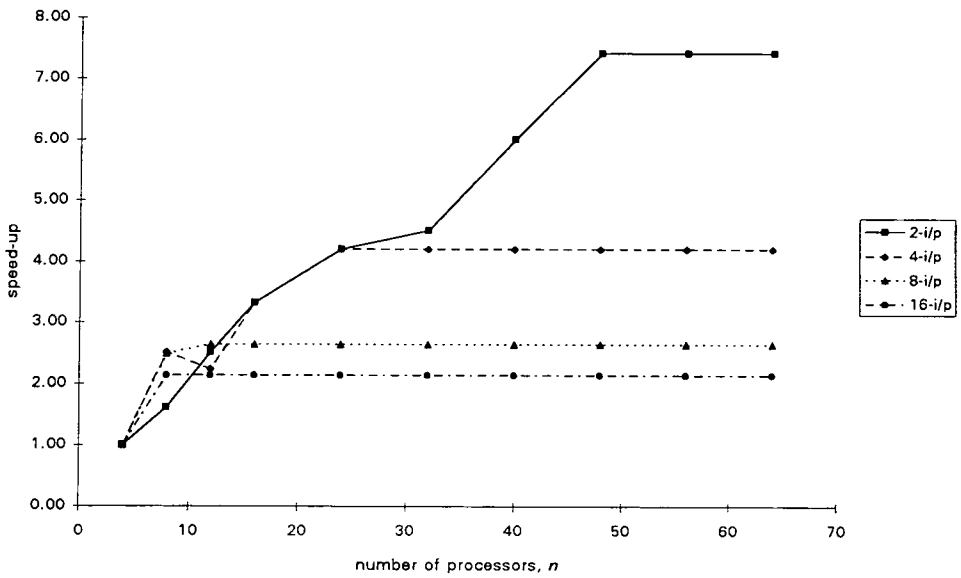

Figure 10-27: DYN–NML–HLE–TSK speed-up for tree-type taskforces.

10.5.5 Scaled Speed-Up

When scaled speed-up (SSU) is the performance metric similar behaviour is observed. All four alternatives are close to linear throughout, with the 2-input characteristic as high as 7.41 when $n = 64$. Using this size of DMC and M-P, the 4-input SSU is 56.67% of this value, with 8- and 16-input primitives yielding 35.71% and 28.91% respectively. At this point, the SSU curves intersect with the corresponding conventional speed-up characteristics.

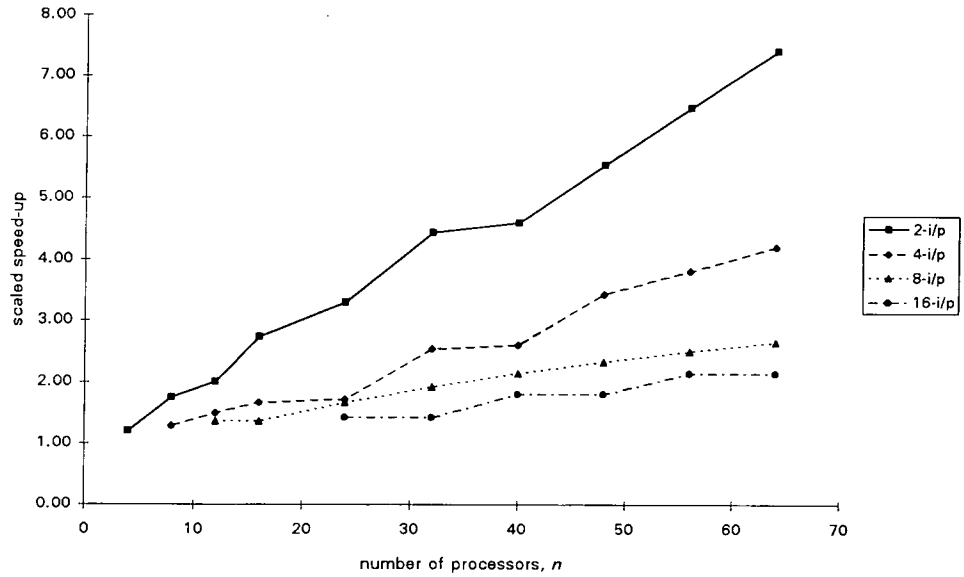

Figure 10-28: DYN–NML–HLE–TSK scaled speed-up for tree-type taskforces.

10.6 Conclusions

In DMC scheduling, the central aim is to efficiently allocate a DMC taskforce with m ASP tasks onto a n-processor M-P architecture. Worst-case $O(n^m)$ trends in AST and anomalies in CPM motivate further research into list-scheduling. MIA and MRC heuristics studied previously, suggest that HLE labels should be revised within the DMC scheduling process.

Equation (10-1) combines aspects of both preferred AST heuristics — MIA and MRC — in order to assign highest priority to that (T_r, P_a) incurring minimum extension of the current partial schedule. Three alternative schemes for task and processor selection — PRO, TSK and ALL — are defined and an upper bound derived for the dynamic strategies supported by *DCS*.[1]

Scheduling an EQ section from Figure 9-1 immediately illustrates some of the advantages engendered by the dynamic approach to DMC taskforce labelling. Scheduling the problem instance shown in Figure 10-4 with DYN–NSL–HLE–TSK not only reduces ω_{sch} to some 90.00% of the DYN–NSL–HLE–NON case, but processor thrashing is also eliminated.

An noted in Section 10.2, the primary objective in this chapter is to determine which, if any, dynamic-labelling scheme is more apt for DMC task scheduling than CPM–xxx–HLE. Rather than extract general trends with randomly-generated taskforces, console designs

1. Unfortunately, no *lower* bound can be derived for these dynamic or, indeed, other list-scheduling disciplines.

entered via schematic capture are used throughout this analysis. With fully-overlapped (FWE) IPC mechanisms, *DCS* run-time for PRO and ALL with bus-configurations is almost identical.

Referring to Section 10.2.1, `dyn_gen_lvl(...)` will generate $O(m^2)$ dynamic levels for PRO and $O(nm^2)$ for ALL. When $m \gg n$, as is typically the case in the DMC problem domain, PRO and ALL complexities tend to the same value in the limit. Employing tree-type taskforces shows that NON and TSK are particularly efficient strategies for generating DMC schedules. Run-time characteristics are effectively linear in the logarithmic domain.

In terms of ω_{sch}, NON and PRO appear to be the best techniques for revising task labels, though TSK gives similar values in many instances. All dynamic schemes exhibit identical speed-up and SSU performance under FWE, intersecting when $n = 1$ and $n = m$ as predicated in Section 7.2.4. While this IPC transport model is perhaps over-optimistic, *HYMIPS* has shown that it is in fact possible to fully overlap computation with IPC.

Repeating this analysis with NSL media shows that *DCS* run-time for bus-configured taskforces is extremely close to that with FWE. While NON yields performance comparable to TSK, it is evident that the choice of P_a on which to schedule T_r is improved by dynamic-labelling when IPC messaging modifies the partial schedule. Selecting the available processor first, PRO, gives run-times well in excess of 5:00.00 for the 64-channel console used here.

With tree-configured taskforces, run-time with NSL is identical to that for FWE transport, except that t_{ALL} is reduced by a factor of 0.560. From Figure 10-14, it is evident that IPC directly affects ω_{sch}/ω_{cp} throughout the entire range of π. NON is the worst scheme studied here due to the current task- and processor-list being ignored. Fortunately, TSK and ALL yield rather better results for this somewhat conservative IPC mechanism.

Figure 10-15 reveals that NSL speed-up is markedly different when compared with FWE. Both NON and PRO converge to a value some 27.04% less than the maximum level determined by ω_{cp}. Both TSK and ALL SSU is no longer linear due to the impact of non-overlapped IPC. PRO is very poor in this regard with PRO $\frac{\partial}{\partial n}$ actually negative for $n > 32$.

While it may not be possible to overlap inter-task transport with computation in some audio M-Ps, it is quite feasible to replicate architectural features. With multiple NSL channels

(NML), *DCS* run-time for bus-configurations is almost identical to FWE. Tree-type DMCs require run-times similar to NSL, as reserving IPC is an integral part of the DYN algorithm.

Linear ω_{sch}/ω_{cp} behaviour is readily apparent for NON and PRO labelling schemes under NML. In fact, PRO is consistently able to minimise the ω ratio for the range of π considered in this analysis. Unfortunately, the performance of TSK and ALL dynamic-labelling schemes proves to be rather erratic, but does lie below PRO in a few problem instances.

Charting conventional speed-up under NML gives characteristics not previously seen in this analysis. TSK is precisely linear sub-unitary until constrained by ω_{cp}, whereas NON gives very variable results with PRO consistently poor. TSK and ALL arrange IPC in such a way as to give identical linear SSU for the range of DMC configurations considered here.

Performance diversity is anticipated when comparing taskforce granularity using DYN–NML–HLE–TSK as n_m varies as Equation (10-6). *DCS* confirms that run-time with larger mix-bus primitives is considerably less than for 2-input variants. Tree-configurations extend t by up to 21.78% due to the increased number of T_r at any step in the DYN scheduling process.

16-input primitives appear to be the best method of minimising ω_{sch}/ω_{cp}. However, parallelism inherent in DMCs means that ω_{sch} for 2-input primitives is 47.50% less than that obtained with 16-input tasks. This is due to ω_{cp} in consoles constructed from 16-input mix-bus tasks being greater than that evident in comparable 2-input implementations.

Due to the increase in task size, large-grain mix-bus primitives exhibit poor speed-up performance. $S(n)$ quickly converges to the ω_{cp}-constrained value, since the opportunities for n-fold parallelism are severely restricted. All four mix-bus structures return linear SSU characteristics, with larger fan-in giving lower SSU as explained for conventional speed-up.

10.7 References

[Graham, 1976] Graham R L: "Bounds on the Performance of Scheduling Algorithms", from *Computer and Job-Shop Scheduling Theory*, Wiley & Sons, 1976, ISBN 0-471-16319-8.

[Linton, 1991] Linton K N (et al.): "On the *Re*-Allocation of Processing Resources for DASP", *IEE Colloquium on Digital Audio Signal Processing*, London, UK, May 1991.

Chapter 11

Conclusions & Further Work

Commercial and creative pressures stimulate considerable interest in the MIMD paradigm for professional digital mixing. This thesis focuses on salient issues in parallel architectures and task scheduling strategies. Herein the author provides new insight into the major hardware and software problems which must be resolved prior to the development of such products.

11.1 Digital Mixing Consoles

A number of factors influence DMC design, ranging from control ergonomics and automation systems through to real-time ASP and underlying M-P architecture. MMI and TDA opportunities are among the most compelling reasons to favour a fully-digital methodology.

11.1.1 Analogue Mixing Consoles

ASP has defined the recorded sound of each musical era, with spatialisation systems stressed today. Although professional consoles contain considerable processing facilities, including basic control of spatial parameters, more comprehensive ASP is still outboard. DSP has added to the repertoire of effects, taking full advantage of decades of evolution in unrelated fields.

Console size and component density are constant design problems if performance levels are to be retained as analogue design endeavours to meet operational requirements. Rather than developments in processing technology, the author predicts that advances in professional console *system* technology will define recorded sound in the coming decade.

While some digital consoles exist in the current marketplace, none provide either the degree or type of functionality originally anticipated by industry professionals. Control surface and automation software are especially critical in this regard. It is therefore essential to investigate DMC implementation issues as they affect both console designer and operator.

11.1.2 Console Digitalisation

Recent trends, including SP-DIF and MADI, augur digital implementation of the sound production process between microphone capsule and loudspeaker cone. A vital stage in DMC design is the ASP sub-system itself. Investigating this aspect not only determines the real-time processing budget, but also reveals redundancy inherent during tracking and mix-down.

Much effort has been invested in advancing digital ASP techniques. In contrast, control surface layout is not generally perceived as important. Freedom from the constraints imposed by mechanical linkage creates a number of opportunities to improve the MMI. Digital implementation makes TDA a technical reality although at significant cost uplift.

Alas, it would seem that the MMI is becoming more maladroit with each advance in surface technology. Improvements can, however, be achieved solely as a result of surface design and not advanced technology. As the heart of any console design must be the ASP engine itself, the second part of this thesis concentrates on this critical aspect of DMC design.

11.2 Parallel Architectures

Programmable DSP devices and architectural issues are investigated from a digital mixing perspective. Alternative topologies are compared as they directly effect DMC performance, modularity and extensibility. Design decisions are justified for a hybrid audio M-P.

11.2.1 Programmable DSP Devices

Floating-point devices largely free the DMC designer from concerns about scaling. Processors such as Motorola's DSP core and the TASP family from Texas Instruments provide stiff competition for ASICs. Although exotic VLSI may achieve another order-of-magnitude improvement in clock rate, industry expectations will undoubtedly match any such advance.

Given stringent VLSI packaging restrictions, byte-wide IPC links provide efficient inter-processor connectivity. While these developments support M-Ps, few devices integrate specific features for DMC systems directly on-chip. None support dedicated delay processing or the additional modulation waveforms required for chorus and pitch-shift algorithms.

In order to implement the console mix function, many instruction cycles are consumed interpolating control-rate data. Multimedia parts such as the Philips Trimedia and SGS-Thompson A110 are emerging with specific DSP features targeted at audio applications. Cost compares favourably with the original pricing structure of the Motorola DSP56001.

11.2.2 Architectural Issues

An original schema of topology metrics, implemented in the *TPG* deliverable, enables quantitative comparison of audio console M-Ps. With a link expansion factor of one, the shared-bus is well-suited to small-scale DMCs until contention limits extensibility. Ring architectures, particularly the tri-valent chordal ring, deserve special attention from the DMC designer. Cross-topology links reduce the maximum inter-node distance δ by a factor of two.

Expanding tree-based DMCs via fan-out b cannot be recommended, whereas incrementing l maintains a fixed number of IPC ports. Less familiar topologies such as cube-connected cycles are among the most attractive constant-valency M-Ps. An imbalance of inter-node traffic is apparent near the root node of a simply b-ary tree. Ring connections in the x-tree ease this bottle-neck while still matching closely in-tree precedence constraints.

If a range of mixing products is the manufacturer's primary goal, trees with restricted fan-out enable a structured upgrade path. At least 3 links must fail in the chordal-ring, x-tree, and cube-connected cycles before a PE is isolated. Unfortunately, the shared-bus is least suitable from this point of view. Chordal-rings and tree structures must be the preferred topologies for DMC implementation, though multi-stage adaptations should not be precluded.

11.2.3 Audio M-P Implementation

Latency in message-passing schemes is dependent on the channel bandwidth, distance and message size. Wormhole routing greatly reduces IPC latency in DMCs as delay is no longer sensitive to the distance involved in sample transport. The IPC latency in a 2-D mesh may be reduced up to $O(\sqrt{n})$ times over a binary hypercube if constant bisection bandwidth can be maintained. Fewer links contribute to the bisection, permitting each channel to be made wider.

Assuming constant PE bandwidth, IPC latency can be reduced by a up to a factor of 2 if a higher-dimensional n-processor hypercube is re-configured as a 2-D mesh. X-tree based

DMCs do not perform especially well under these conditions since 2 connections are required per processor to interconnect the ring at each level. A simple 3-ary tree gives the same wormhole latency improvement over a binary hypercube as a chordal-ring when $k \ll K$.

HYMIPS is not a shared-bus M-P in the classical sense: shared DPR and an alternative semaphore protocol allow pseudo-wormhole messaging to overlap task execution, supporting a range of *virtual* IPC topologies. A *HYMIPS* node has been constructed, together with *HSR* and *HST* SP-DIF peripherals. This hybrid approach to the DMC compute engine offers scalability both in terms of the number of compute PEs per node and the number of nodes.

11.3 Task Scheduling Strategies

In DMC scheduling, the central aim is to efficiently allocate a taskforce with m ASP tasks onto a n-processor M-P. Worst-case trends in AST and anomalies in CPM have instigated research into list-scheduling. Dynamic-labelling captures the significant features of DCS.

11.3.1 Multi-Processor Models

DCS must observe all precedence relations and be non-preemptive. Since parameters are known a priori, the DMC taskforce can be scheduled at compile-time. Conventional speed-up assumes that the ratio of parallel to sequential work is fixed. In situations were both taskforce and M-P are scaled, SSU indicates how performance will vary as the entire console is scaled.

As noted in Chapter 3, the maximum number of pipeline stages in a LPP-based DMC is $k_{max} = 48$. Further analysis shows that this approach to parallelising the DMC taskforce provides opportunities to achieve unitary speed-up and maximum processing efficiency. This technique has been directly applied within the *HYMIPS* audio M-P described previously.

A number of different M-P models have been investigated with respect to task granularity. Unitary speed-up can be achieved by overlapping computation with multiple IPC links. For each model, there is some maximum n that is cost-effective: this number depends on the M-P topology, on IPC mechanisms and, to a degree, on the particular DMC taskforce.

11.3.2 Scheduling Strategies

DMC DAGs have task execution times known a priori. With the complexity of the DCS problem derived in Equation (8-1), the borderline between polynomial and *NP*-complete problems is evident. Important cases such as unit-length tasks with arbitrary precedence for $n \geq 3$, and taskforces with *in-tree* precedence constraints have particular relevance to DCS.

Both SA and GA require empirical tuning of implementation-specific factors to guarantee convergence. Depending on cooling schedules or recombination operators are not options in the DMC environment. Heuristic underestimates — MRC, MIA and MHU — and static-labelling schemes — HLN, HLE, LCN and LCE — leverage off precedence constraints.

The best priority lists are those in which tasks on longest ω_{cp} appear near the head. Only relatively small additions to processor usage are then made at the end of the schedule. Lack of idle time and non-preemption are responsible for the unpredictable behaviour of CPM, when available processors are forced to execute ready tasks prematurely.

11.3.3 DCS Research Framework

Instead of requiring partitioning a priori, LGDF console schematics are inherently partitioned at the ASP task. Behaviour models test AST, CPM and DYN algorithms, as well as validate NSL, NML and FWE IPC mechanisms. Internally, *DCS* retains the current state of the partial schedule together with a list of unscheduled tasks. By referring to the `Prolog` source, the scheduler predicate hierarchy illustrates the benefits of a declarative approach to DCS.

While MPC may be effective as *DCS* nears completion, this underestimate readily causes AST to revert to exhaustive search. Rather better results are reported for MRC and MHU although the latter requires significant run-time uplift as three separate $h(n)$ must be determined for each n. The essentially $O(n^m)$ requirements of AST moves the focus of this research towards deterministic approaches derived from MRC & MIA heuristic underestimates.

With linear taskforces, no improvement is possible by ordering the priority list, whereas HLE returns manifest benefits when task priority is not fully constrained. With HLE consistently returning shortest ω_{sch}, π has a direct bearing on ω_{sch}: $n < \pi$ signifies that

inherent parallelism cannot be fully exploited. In summary, the evidence is clear that the order of non-increasing performance of static-labelling schemes is — HLE, LCN, LCE, HLN, RND.

11.3.4 Dynamic Labelling Schemes

Dynamic labelling combines aspects of both preferred AST heuristics in order to assign highest priority to that (T_r,P_a) incurring minimum extension of the current partial schedule. Three alternative schemes — PRO, TSK and ALL — are employed in the *DCS* deliverable. Scheduling an EQ section in isolation immediately illustrates some of the advantages engendered by dynamic taskforce labelling since 'processor thrashing' is eliminated.

Fully-overlapped (FWE) IPC demonstrates that NON and TSK are efficient strategies for generating DMC schedules. Analysis with NSL media clearly shows that the choice of P_a on which to schedule T_r is improved by dynamic-labelling. *DCS* run-time with NML is similar to NSL, as reserving IPC is an integral part of DYN. TSK and ALL arrange IPC in such a way as to give the following ordering, since ALL is $O(m)$ greater than TSK— TSK, ALL, PRO, NON.

DCS confirms the expected performance diversity when comparing taskforce granularity using DYN–NML–HLE–TSK — the preferred approach to DMC scheduling. Inherent parallelism means that ω_{sch} for small-grain primitives is considerably less than that obtained with large-grain tasks. Large-grain mix-buses exhibit poor speed-up performance as n-fold concurrency is severely restricted. All mix-*tree* structures return linear SSU characteristics.

11.4 Further Work

As the audio community becomes accustomed to buying more technology with less money, manufacturers face increasingly difficult design problems. Concepts from closely related academic subjects are becoming increasingly pertinent in this interdisciplinary industry.

11.4.1 Digital Mixing Consoles

With the availability of 'true' DMCs, the author is convinced that proprietary outboard ASP, and spatialisation algorithms in particular, will migrate directly into the console channel strip.

Using the DISQ M-P, AT&T has demonstrated the use of a known analogue MMI (an SSL *G*-Series) to drive a prototype DMC core, implying interchangeable MMIs in the future.

Automation systems have developed to the extent that many of the final creative decisions can be left until the last possible moment. With the realisation of TDA, the step from manual tracking to computerised mix-down will be seamless. The author perceives that this type of 'tapeless studio' will stimulate fresh ideas in contemporary music production.

11.4.2 Parallel Architectures

In this thesis, the author has clearly the demonstrated the viability of constructing audio M-Ps from off-the-shelf DSP parts. As the *HYMIPS* principle is not unduly restrictive, two further areas of research are implied. Firstly, rather than a dedicated communications processor, ASIC technology presents a cost effective method for message-passing between compute PEs.

Note that the *HYMIPS* backplane has a simple static RAM interface which connects transparently to most programmable DSP designs, irrespective of RWM support. Secondly, with the unit-cost of new DSP devices dropping to acceptable levels, *HYMIPS* II would utilise these novel DSP processor within the same generic structure employed in the original design.

11.4.3 Task Scheduling Strategies

Whether for development or operational purposes, task scheduling strategies are crucial to the realisation of flexible DMC systems for the professional audio mixing environment.

11.4.3.1 Additional Precedence Constraints

Bussel (et al.) [Bussel, 1974] address the general problem of minimising the number of processors, *n*, or the execution time. Reduced to the simplest terms, their algorithm consists of adding precedence constraints (i.e., additional nets) to the original DAG at those points where the number of ready tasks exceeds *n*, or in the terminology developed here, $\pi > 1$.

The effect is to distribute processor demands throughout the length of the graph, without adversely affecting overall ω_{sch}. One technique to add constraints in *DCS* would be to first schedule the DMC taskforce in reverse using LCN or LCE. Information derived from such schedules could then be used to determine where additional arcs may be added.

11.4.3.2 Bus-Configurations Re-Examined

In the results presented for *DCS* algorithms, it is evident that bus-configured DMCs quickly exceed the pre-schedule ω_{cp} test detailed in Section 9.2.4. However, the LPP concept developed in Section 7.3 shows one solution to this problem. Careful insertion of pipeline delays within *DCS* would ensure that linear taskforces directly benefit from this approach.

Alternatively, there is another way to manage DMC taskforces constructed from mix-bus primitives. With interpolated control-rate fader data and audio rate sample data adhering to associative principles, provided extreme numerical effects such as overflow are absent, then any mix-bus structure, $\omega_{cp} = \omega_{seq}$, can be re-configured as a mix-*tree* $\omega_{cp} \ll \omega_{seq}$.

11.4.3.3 Code Generation

DSP assembly code macros, see Figures 9-5 and 10-22, allow sound software engineering practices to be employed. One consequence is, that in order to maintain modularity, no task can retain ownership of DSP ALU registers. When a task is launched, no assumptions can be made about the data held in these registers. The entire task context must be stored in memory.

Modular DSP coding techniques lead to redundant instruction execution. In the code fragment of Figure 11-1 accumulator a is stored as task T_1 completes and is immediately re-loaded with the same value when T_2 commences execution. Unnecessary overhead could be removed by post-allocation boundary optimisation and the code file subsequently compiled.

```
        move    a,x:$100        ; end   of task T1
;-----------------------------------------------
        move    x:$100,a        ; start of task T2
```

Figure 11-1: Example of wasteful code at DMC task boundary.

11.4.4 Manufacturing Considerations

There are many ways to express the performance of a DMC. While the performance metric commonly used is MIPS cost, fault-tolerance and extensibility are just as important.

11.4.4.1 Cost

An understanding of cost is essential for designers to be able to make informed decisions. The cost of DSP components is changing so fast that design decisions must not be based on

today's costs, but on projected costs at the time the product is shipped. For example, in digital audio component costs typically make up 15-33% of the final list price [Scuffham, 1992].

Most companies spend only 8-15% of income on R&D, which suggests the application of fixed-overhead percentages to translate cost into price. However, if this level of R&D is considered as an investment then every pound spent on R&D must generate £7-13 in sales. This alternative point of view would then imply *different* gross margins depending on product.

11.4.4.2 Fault-Tolerance

It is commonly held that M-Ps are inherently tolerant of faults since there is a natural availability of spares. Removing a node effectively removes all the connected arcs, with the result that node failure is more 'damaging' than arc failure. Designing DMCs which are fault-tolerant (and hence reliable) involves a great deal more that simply providing spares, however.

The only alternative is to design multi-DSP systems for maximum resilience and fault-tolerance. Designing for maximum resilience means minimising the use of the least reliable components. Designing for fault-tolerance means two things: firstly, detection of the occurrence of an error, and; secondly, correction and recovery from a detected error.

11.4.4.3 Extensibility

Few customers can afford the initial cost of a large multi-DSP audio system. However, many can afford to grow a large system in increments as their budgets permit. There is also a great risk in designing a multi-DSP M-P welded to a particular ASP application. By incorporating sufficient flexibility, R&D investment can be amortised over a range of end-products.

It is obviously desirable to minimise the increase in cost to that of linear growth. Moreover, the designer must assess how architectural parameters, such as IPC capacity, vary with replication. An important practical consideration, especially for very large-scale ASP, is the space occupied by the system as a whole and, more particularly, by inter-processor wiring.

11.5 Summary

Large DSP systems generally cost more to develop — a product costing 10 times as much to manufacture may cost many times as much to develop. As yet, there is no expedient way to

developing digital mixing products which derive full benefit from digital implementation. *TPG*, *HYMIPS* and *DCS* contribute directly to meeting the demands of audio professionals.

HYMIPS, a hybrid audio M-P targeted at real-time multi-channel ASP, illustrates ways in which programmable DSP parts can be employed to construct real-time ASP engines. *TPG* compares 12 M-P topologies using 13 metrics to demonstrate that chordal-ring, tree and cube-connected cycles are the topologies of choice for architectures underlying DMC systems.

Dynamic-labelling significantly improves performance by over 50.00% when compared to CPM–xxx–HLE alone. Although the inability of DYN–xxx–HLE–TSK to idle processors is an inherent flaw in any list-scheduling strategy, the evidence seems clear that this particular approximation algorithm is well suited to the professional DMC environment.

11.6 References

[Bussell, 1974] Bussel B, Fernandez E and Levy O: "Optimal Scheduling for Homogeneous Multiprocessors", *Proc. IFIP Congress 74*, September 1974.

[Scuffham, 1992] Scuffham M: *JFX Hardware Costing*, private communication, November 1992.

Appendix A

Survey of DSP Processors

Architecturally important programmable DSP processors are reviewed in Table A-1. It should be noted that many popular devices have subsequently been released with faster clock rates.

Company	Part No.	Release Date	MAC (ns)	Comments
AMI	s2811	1978	300	First DSP design, 12/16-bit FX, released 1982
Analog Devices	ADSP-2100/00A	1986/88	125/80	16/40-bit FX
	ADSP-21020	1991	80 or 100	32-bit FP, 40-bit IEEE FP, 2 external buses
AT&T	DSP1	1979	800	Early DSP, 16/20-bit FX, not marketed
	DSP32/32C	1984/88	160/80	32-bit FP
	DSP16/16A	1987/88	55/33	16-bit FX
	DSP3210	1992	30 or 36	Enhanced DSP32, with host port
Fujitsu	MB8764	1983	120	16/26-bit FX
	MB86232	1987	150	32-bit IEEE FP
	MB86224	1989	75	24-bit FX
Hitachi	HD61810	1982	250	12-bit FP
	DSPi	1988	50	Very fast I/O, for image processing
IBM	Hermes	1981	100	Fast MAC time, not marketed
Inmos	IMS A100	1986	(12.5)	16-bit systolic FX, based on 32-tap transversal filter

Table A-1: Summary of important DSP devices from VLSI manufacturers.[a]

Company	Part No.	Release Date	MAC (ns)	Comments
Motorola	DSP56001	1987	60, 75 or 97.5	24-bit FX, aimed at digital audio
	DSP96002	1990	60 or 75	32-bit IEEE FP, 2 external buses
	DSP96001	1990	60 or 75	DSP96002 with 1 external bus
	DSP56116	1991	50	16-bit version of DSP56001
	DSP56002	1993	50	Enhanced DSP56001, with PLL on-chip
National Semiconductor	LM32900	1987	100	16/32-bit FX, no external memory
NEC	μPD7720	1980	250	Heavily used, early FX DSP
	μPD7281	1984	—	Dataflow DSP, for image processing
	μPD77230	1985	150	32-bit FP
	VISP	1989	—	For video, not marketed
	μPD6380	1990	122	Update of 7720, for digital audio
Oki Semiconductor	MSM6992	1986	100	22-bit FP
SGS-Thompson	ST18930	1988	80	16-bit, integer or complex maths
Toshiba	6386/7	1983	250	16/31-bit FX
Sharp	LH9124	1991	25	24-bit block FX, time/freq. domain
	LH9320	1991	—	Address generator for LH9124

Table A-1: Summary of important DSP devices from VLSI manufacturers.[a]

Company	Part No.	Release Date	MAC (ns)	Comments
Texas Instruments	TMS32010	1982	390	Early popular DSP, 16-bit FX
	TMS32020	1985	195	Update of TMS32010
	TMS320C25	1987	100	CMOS update of TMS32020
	TMS320C30	1988	50 or 60	32-bit FP
	TMS320C50	1990	35 or 50	16-bit FX
	TMS320C40	1991	40 or 50	Enhanced 'C30, 6 comms. ports
	TMS320C80	1995	40	Integrates 4 64-bit FP DSPs with a 32-bit RISC CPU
Zoom	ZM6420	1988	350	20-bit FX, serial-bus ASP, on-chip ADC and DACs
Zoran	34161	1986	100	16-bit block FP, high-level DSP instructions
	34322	1988	—	32-bit block FP
	35325	1989	—	32-bit IEEE FP

Table A-1: Summary of important DSP devices from VLSI manufacturers.[a]

a. In this table, the abbreviations FX and FP are used for fixed-point and floating-point, respectively.

Appendix B

Topology Expansion Metrics

The tables below detail the expansion increments for the thirteen M-P topologies considered in Chapter 5. Where more than one metric appears, b and w take precedence over l and d.

Topology	Number of Nodes	Increment	Factor Increase
full interconnection	n	1	$\frac{n+1}{n}$
shared bus	n	1	$\frac{n+1}{n}$
simple ring	n	1	$\frac{n+1}{n}$
chordal ring	n	2	$\frac{n+2}{n}$
b-ary tree	$\frac{b^l-1}{b-1}$	$\frac{(b+1)^l-b^l-n}{b}$	$\frac{(b+1)^l-1}{nb}$
		b^l	$b+\frac{1}{n}$
b-ary x-tree	$\frac{b^l-1}{b-1}$	$\frac{(b+1)^l-b^l-n}{b}$	$\frac{(b+1)^l-1}{nb}$
		b^l	$b+\frac{1}{n}$
nearest-neighbour mesh	w^d	$(w+1)^d-w^d$	$\left(\frac{w+1}{w}\right)^d$
		$(w-1)w^d$	w
spanning-bus hypercube	w^d	$(w+1)^d-w^d$	$\left(\frac{w+1}{w}\right)^d$
		$(w-1)w^d$	w

Table B-1: M-P topology node expansion metrics.

Topology	Number of Nodes	Increment	Factor Increase
dual-bus hypercube	w^d	$(w+1)^d - w^d$	$\left(\frac{w+1}{w}\right)^d$
		$(w-1)\,w^d$	w
torus	w^d	$(w+1)^d - w^d$	$\left(\frac{w+1}{w}\right)^d$
		$(w-1)\,w^d$	w
R-ary N-cube	NR^N	$N[(R+1)^N - R^N]$	$\left(\frac{R+1}{R}\right)^N$
		$[N(R-1)+R]\,R^N$	$\frac{R(N+1)}{N}$
binary hypercube	2^d	2^d	2
cube-connected cycles	$d2^d$	$(d+2)\,2^d$	$\frac{2(d+1)}{d}$

Table B-1: M-P topology node expansion metrics.

Topology	Number of Links	Increment	Factor Increase
full inter-connection	$\frac{n(n-1)}{2}$	n	$\frac{n+1}{n-1}$
shared bus	1	0	1
simple ring	n	1	$\frac{n+1}{n}$
chordal ring	$\left\lfloor \frac{3n}{2} \right\rfloor$	$(n \% 2) + 1$	$\frac{(n \% 2)+1}{\left\lfloor \frac{3n}{2} \right\rfloor} + 1$
b-ary tree	$n-1$	$\frac{(b+1)^l - b^l - n}{b}$	$\frac{(b+1)^l - (b+1)}{b(n-1)}$
		b^l	$\frac{b^l - 1}{b^{l-1} - 1}$

Table B-2: M-P topology link expansion metrics.

Topology	Number of Links	Increment	Factor Increase
b-ary x-tree	$2(n-1)$	$\frac{2[(b+1)^l - b^l - n]}{b}$	$\frac{(b+1)^l - (b+1)}{b(n-1)}$
		$2b^l$	$\frac{b^l - 1}{b^{l-1} - 1}$
nearest-neighbour mesh	$(w-1)dw^{d-1}$	$wd[(w+1)^{d-1} - (w-1)w^{d-2}]$	$\frac{(w+1)^{d-1}}{(w-1)w^{d-2}}$
		$(w-1)w^{d-1}[w(d+1) - d]$	$\frac{w(d+1)}{d}$
spanning-bus hypercube	dw^{d-1}	$d[(w+1)^{d-1} - w^{d-1}]$	$\left(\frac{w+1}{w}\right)^{d-1}$
		$w^{d-1}[w(d+1) - d]$	$\frac{w(d+1)}{d}$
dual-bus hypercube	$2w^{d-1}$	$2[(w+1)^{d-1} - w^{d-1}]$	$\left(\frac{w+1}{w}\right)^{d-1}$
torus	dw^d	$2(w-1)w^{d-1}$	w
		$d[(w+1)^d - w^d]$	$\left(\frac{w+1}{w}\right)^d$
		$w^d[w(d+1) - d]$	$\frac{w(d+1)}{d}$
R-ary N-cube	NR^{N+1}	$N[(R+1)^{N+1} - R^{N+1}]$	$\left(\frac{R+1}{R}\right)^{N+1}$
		$R^{N+1}[R(N+1) - N]$	$\frac{R(N+1)}{N}$
binary hypercube	$d2^{d-1}$	$(d+2)2^{d-1}$	$\frac{2(d+1)}{d}$
cube-connected cycles	$3d2^{d-1}$	$3(d+2)2^{d-1}$	$\frac{2(d+1)}{d}$

Table B-2: M-P topology link expansion metrics.

Appendix C

Topology Analysis Tool

In order to compare alternative M-P topologies, the author has developed a stand-alone add-in for Excel. Written in C, this .XLL implements the topology metrics discussed in Chapter 5.

C.1 Overview

This appendix lists the definition, include and source files required to re-build *TPG* — the *Topology Analysis Tool* — which automatically registers the following worksheet functions:

Topology	Function Name	Arguments[a]
full-interconnection	`fulc`	`n, metric`[b]
shared bus	`bus`	`n, metric`
simple ring	`ring`	`n, metric`
chordal ring	`c_ring`	`n, metric`
b-ary tree	`tree`	`b, l, metric, [expansion]`[c]
b-ary x-tree	`x_tree`	`b, l, metric, [expansion]`
nearest-neighbour mesh	`mesh`	`w, d, metric, [expansion]`
spanning-bus hypercube	`sb_hpc`	`w, d, metric, [expansion]`
dual-bus hypercube	`db_hpc`	`w, d, metric, [expansion]`
torus	`torus`	`w, d, metric, [expansion]`
R-ary *N*-cube	`rn_cube`	`R, N, metric, [expansion]`
binary hypercube	`b_hpc`	`d, metric`
cube-connected cycles	`cc_cycles`	`d, metric`

Table C-1: Worksheet functions supported by *TPG*.

a. square-brackets indicate an optional argument.
b. `metric` determines which topology metric should be returned to the worksheet.
c. `expansion` determines whether the primary or secondary topology parameter should be expanded.

C.2 Definition Files

The *Topology Analysis Tool* .XLL definition file is listed below.

C.2.1 tpg.def

```
; ******************************************************************************
; © 1995     : School of Engineering, University of Durham, Durham, England
;            : Solid State Logic Ltd, Begbroke, Oxford, England
; ******************************************************************************

; ******************************************************************************
; MODULE     : TPG.DEF
;
; NOTES      : Definition of function calls exported from XLL to Excel
;
; AUTHOR     : Ken N LINTON
;
; DATE       : 19th February, 1995
;
; VERSION    : 1.0
; ******************************************************************************

; ==============================================================================
; Pre-amble
; ==============================================================================

LIBRARY     TPG

DESCRIPTION 'Add-ins functions for calculating M-P topology metrics'

EXETYPE     WINDOWS

CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE SINGLE

HEAPSIZE    8192
STACKSIZE   0

; ==============================================================================
; Exported XLL functions
; ==============================================================================

EXPORTS

    ; ---------------------------------------------------------------------------
    ; Mandatory stand-alone DLL functions
    ; ---------------------------------------------------------------------------

    WEP                 @1  RESIDENTNAME
    xlAutoOpen          @2
    xlAutoClose         @3
    xlAutoRegister      @4
    xlAutoRemove        @5
    xlAutoAdd           @6
    xlAddInManagerInfo  @7

    ; ---------------------------------------------------------------------------
    ; User-defined Excel add-in functions
    ; ---------------------------------------------------------------------------

    inf_x_fulintcon     @8
    inf_x_sharedbus     @9
    inf_x_simplerng     @10
    inf_x_chordlrng     @11
    inf_x_simpletre     @12
    inf_x_fulrngtre     @13
    inf_x_nrsnhbmsh     @14
    inf_x_spabushpc     @15
    inf_x_duabushpc     @16
    inf_x_torus         @17
    inf_x_binaryhpc     @18
    inf_x_cubconcyc     @19
    inf_x_raryncube     @20

; ******************************************************************************
```

C.3 Include Files

Topology Analysis Tool includes are listed in this section.

C.3.1 tpg_alg.h

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : TPG_ALG.H                                                      */
/*                                                                            */
/* NOTES     : Analysis functions for multiprocessor topologies               */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 16th February, 1995                                            */
/*                                                                            */
/* VERSION   : 1.0                                                            */
/* ************************************************************************** */

/* -------------------------------------------------------------------------- */
#ifndef TPG_ALG_H
#define TPG_ALG_H
/* -------------------------------------------------------------------------- */

/* ========================================================================== */
/* C headers                                                                  */
/* ========================================================================== */

#include   <stdio.h>

/* ========================================================================== */
/* TPG headers                                                                */
/* ========================================================================== */

#include   "tpg_utl.h"

/* ========================================================================== */
/* Type definitions                                                           */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* TYPEDEF  : enum for ALG error codes                                      */
/* -------------------------------------------------------------------------- */

typedef enum AlgErrCod_
{
    ALG_SUCCESS  = 0,
    ALG_FTL_ERR
} AlgErrCod;

/* -------------------------------------------------------------------------- */
/* TYPEDEF  : enum for topology metric codes                                  */
/* -------------------------------------------------------------------------- */

typedef enum MetricCod_
{
    MT_NUM_NOD,                     /* number of nodes          */
    MT_NUM_CON,                     /* number of connections    */
    MT_NUM_LNK,                     /* number of links          */
    MT_NOD_INC,                     /* node expansion increment */
    MT_NOD_FAC,                     /* node expansion factor    */
    MT_LNK_INC,                     /* link expansion increment */
    MT_LNK_FAC,                     /* link expansion factor    */
    MT_LNK_CON,                     /* link connectivity        */
    MT_NOD_CON,                     /* node connectivity        */
    MT_TOP_DIA,                     /* topology diameter        */
    MT_VALENCY,                     /* valency                  */
    MT_DV_PROD                      /* diameter.valency         */
} MetricCod;

/* -------------------------------------------------------------------------- */
/* TYPEDEF  : enum for expansion increment codes                              */
/* -------------------------------------------------------------------------- */

typedef enum ExpnsnCod_
{
    EX_FRST,                        /* b, w or R */
    EX_SCND                         /* l, d or N */
} ExpnsnCod;

/* ========================================================================== */
/* Macro definitions                                                          */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* MACRO     : alg_m_cllalgfnc(...)                                           */
/*                                                                            */
/* NOTES     : Call topology analysis function                                */
/*                                                                            */
/* INPUTS    : cAlgFnc                - ALG function call                     */
/*                                                                            */
/* OUTPUTS   : <none>                                                         */
/*                                                                            */
/* CALLS     : <cAlgFnc>                                                      */
/*           : utl_f_wrierrlog(...)                                           */
/* -------------------------------------------------------------------------- */

#define alg_m_cllalgfnc( cAlgFnc)                                            \
{                                                                            \
    if ( cAlgFnc != ALG_SUCCESS )                                            \
    {                                                                        \
```

```
        utl_f_wrierrlog( "** ALG call failed: %s\n", #cAlgFnc);          \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);      \
                                                                          \
        return( ALG_FTL_ERR);                                             \
    }                                                                     \
}

/* ------------------------------------------------------------------------ */
/* MACRO     : alg_m_chknulptr(...)                                         */
/*                                                                          */
/* NOTES     : Check value pointer is not NULL                              */
/*                                                                          */
/* INPUTS    : ptr                    - a pointer                           */
/*                                                                          */
/* OUTPUTS   : <none>                                                       */
/*                                                                          */
/* CALLS     : utl_f_wrierrlog(...)                                         */
/* ------------------------------------------------------------------------ */

#define alg_m_chknulptr( ptr)                                             \
{                                                                         \
    if ( ptr == NULL )                                                    \
    {                                                                     \
        utl_f_wrierrlog( "** NULL pointer: %s\n", #ptr);                  \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);      \
                                                                          \
        return( ALG_FTL_ERR);                                             \
    }                                                                     \
}

/* ------------------------------------------------------------------------ */
/* MACRO     : alg_m_chkintgtz(...)                                         */
/*                                                                          */
/* NOTES     : Check integer is greater than zero                           */
/*                                                                          */
/* INPUTS    : nInt                   - an integer                          */
/*                                                                          */
/* OUTPUTS   : <none>                                                       */
/*                                                                          */
/* CALLS     : utl_f_wrierrlog(...)                                         */
/* ------------------------------------------------------------------------ */

#define alg_m_chkintgtz( nInteger)                                        \
{                                                                         \
    if ( nInteger < 1 )                                                   \
    {                                                                     \
        utl_f_wrierrlog( "** %s is less than 1\n", #nInteger);            \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);      \
                                                                          \
        return( ALG_FTL_ERR);                                             \
    }                                                                     \
}

/* ======================================================================== */
/* Public function prototypes                                               */
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_fulintcon(...)                                          */
/*                                                                          */
/* NOTES    : Full-interconnection topology analysis function               */
/*                                                                          */
/* INPUTS   : n                      - number of nodes                      */
/*          : nMetCod                - topology metric code                 */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_fulintcon
(
    int         n,
    MetricCod   nMetCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sharedbus(...)                                          */
/*                                                                          */
/* NOTES    : Shared-bus topology analysis function                         */
/*                                                                          */
/* INPUTS   : n                      - number of nodes                      */
/*          : nMetCod                - topology metric code                 */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_sharedbus
(
    int         n,
    MetricCod   nMetCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
```

```
/* FUNCTION : alg_f_simplerng(...)                                        */
/*                                                                        */
/* NOTES    : Simple ring topology analysis function                      */
/*                                                                        */
/* INPUTS   : n                      - number of nodes                    */
/*          : nMetCod                - topology metric code               */
/*                                                                        */
/* OUTPUTS  : pdMetVal               - metric value                       */
/*                                                                        */
/* CALLS    : SELF                                                        */
/*          : alg_m_chkintgtz(...)                                        */
/*          : alg_m_chknulptr(...)                                        */
/*          : alg_m_cllalgfnc(...)                                        */
/*          : utl_f_wrierrlog(...)                                        */
/*                                                                        */
/* RETURNS  : AlgErrCod                                                   */
/* ---------------------------------------------------------------------- */

AlgErrCod
alg_f_simplerng
(
    int          n,
    MetricCod    nMetCod,
    double      *pdMetVal
);

/* ---------------------------------------------------------------------- */
/* FUNCTION : alg_f_chordlrng(...)                                        */
/*                                                                        */
/* NOTES    : Chordal ring topology analysis function                     */
/*                                                                        */
/* INPUTS   : n                      - number of nodes                    */
/*          : nMetCod                - topology metric code               */
/*                                                                        */
/* OUTPUTS  : pdMetVal               - metric value                       */
/*                                                                        */
/* CALLS    : SELF                                                        */
/*          : alg_m_chkintgtz(...)                                        */
/*          : alg_m_chknulptr(...)                                        */
/*          : alg_m_cllalgfnc(...)                                        */
/*          : utl_f_wrierrlog(...)                                        */
/*                                                                        */
/* RETURNS  : AlgErrCod                                                   */
/* ---------------------------------------------------------------------- */

AlgErrCod
alg_f_chordlrng
(
    int          n,
    MetricCod    nMetCod,
    double      *pdMetVal
);

/* ---------------------------------------------------------------------- */
/* FUNCTION : alg_f_simpletre(...)                                        */
/*                                                                        */
/* NOTES    : Simple b-ary tree topology analysis function                */
/*                                                                        */
/* INPUTS   : b                      - node fanout                        */
/*          : l                      - number of levels                   */
/*          : nMetCod                - topology metric code               */
/*          : nExpCod                - expansion parameter code           */
/*                                                                        */
/* OUTPUTS  : pdMetVal               - metric value                       */
/*                                                                        */
/* CALLS    : SELF                                                        */
/*          : alg_m_chkintgtz(...)                                        */
/*          : alg_m_chknulptr(...)                                        */
/*          : alg_m_cllalgfnc(...)                                        */
/*          : alg_f_trenodinc(...)                                        */
/*          : alg_f_trenodfac(...)                                        */
/*          : alg_f_trelnkinc(...)                                        */
/*          : alg_f_trelnkfac(...)                                        */
/*          : utl_f_wrierrlog(...)                                        */
/*                                                                        */
/* RETURNS  : AlgErrCod                                                   */
/* ---------------------------------------------------------------------- */

AlgErrCod
alg_f_simpletre
(
    int          b,
    int          l,
    MetricCod    nMetCod,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ---------------------------------------------------------------------- */
/* FUNCTION : alg_f_fulrngtre(...)                                        */
/*                                                                        */
/* NOTES    : Full-ring or cross-connected tree topology analysis function */
/*                                                                        */
/* INPUTS   : b                      - node fanout                        */
/*          : l                      - number of levels                   */
/*          : nMetCod                - topology metric code               */
/*          : nExpCod                - expansion parameter code           */
/*                                                                        */
/* OUTPUTS  : pdMetVal               - metric value                       */
/*                                                                        */
/* CALLS    : SELF                                                        */
/*          : alg_m_chkintgtz(...)                                        */
/*          : alg_m_chknulptr(...)                                        */
/*          : alg_m_cllalgfnc(...)                                        */
/*          : alg_f_frtnodinc(...)                                        */
/*          : alg_f_frtnodfac(...)                                        */
/*          : alg_f_frtlnkinc(...)                                        */
/*          : alg_f_frtlnkfac(...)                                        */
/*          : utl_f_wrierrlog(...)                                        */
/*                                                                        */
/* RETURNS  : AlgErrCod                                                   */
/* ---------------------------------------------------------------------- */
```

```
AlgErrCod
alg_f_fulrngtre
(
    int         b,
    int         l,
    MetricCod   nMetCod,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nrsnhbmsh(...)                                          */
/*                                                                          */
/* NOTES    : Nearest-neighbour mesh topology analysis function             */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_nnmnumcon(...)                                          */
/*          : alg_f_nnmnumlnk(...)                                          */
/*          : alg_f_nnmnodinc(...)                                          */
/*          : alg_f_nnmnodfac(...)                                          */
/*          : alg_f_nnmlnkinc(...)                                          */
/*          : alg_f_nnmlnkfac(...)                                          */
/*          : alg_f_nnmlnkcon(...)                                          */
/*          : alg_f_nnmnodcon(...)                                          */
/*          : alg_f_nnmtopdia(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_nrsnhbmsh
(
    int         w,
    int         d,
    MetricCod   nMetCod,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_spabushpc(...)                                          */
/*                                                                          */
/* NOTES    : Spanning-bus hypercube topology analysis function             */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_sbhnodinc(...)                                          */
/*          : alg_f_sbhnodfac(...)                                          */
/*          : alg_f_sbhlnkinc(...)                                          */
/*          : alg_f_sbhlnkfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_spabushpc
(
    int         w,
    int         d,
    MetricCod   nMetCod,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_duabushpc(...)                                          */
/*                                                                          */
/* NOTES    : Dual-bus hypercube topology analysis function                 */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_dbhnodinc(...)                                          */
/*          : alg_f_dbhnodfac(...)                                          */
/*          : alg_f_dbhlnkinc(...)                                          */
/*          : alg_f_dbhlnkfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
```

```
alg_f_duabushpc
(
    int          w,
    int          d,
    MetricCod    nMetCod,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_torus(...)                                              */
/*                                                                          */
/* NOTES    : Torus topology analysis function                              */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_trsnodinc(...)                                          */
/*          : alg_f_trsnodfac(...)                                          */
/*          : alg_f_trslnkinc(...)                                          */
/*          : alg_f_trslnkfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_torus
(
    int          w,
    int          d,
    MetricCod    nMetCod,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rarynube(...)                                           */
/*                                                                          */
/* NOTES    : R-ary N-cube topology analysis function                       */
/*                                                                          */
/* INPUTS   : R                      - node fanout                          */
/*          : N                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_raryncube
(
    int          N,
    int          R,
    MetricCod    nMetCod,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_binaryhpc(...)                                          */
/*                                                                          */
/* NOTES    : Binary hypercube topology analysis function                   */
/*                                                                          */
/* INPUTS   : d                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_binaryhpc
(
    int          d,
    MetricCod    nMetCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_cubconcyc(...)                                          */
/*                                                                          */
/* NOTES    : Cube-connected-cycles topology analysis function              */
/*                                                                          */
/* INPUTS   : d                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
```

```
/* CALLS     : SELF                                                        */
/*           : alg_m_chkintgtz(...)                                        */
/*           : alg_m_chknulptr(...)                                        */
/*           : alg_m_cllalgfnc(...)                                        */
/*           : utl_f_wrierrlog(...)                                        */
/*                                                                         */
/* RETURNS   : AlgErrCod                                                   */
/* ----------------------------------------------------------------------- */

AlgErrCod
alg_f_cubconcyc
(
    int          d,
    MetricCod    nMetCod,
    double      *pdMetVal
);

/* ======================================================================= */
/* Private function prototypes                                             */
/* ======================================================================= */

/* ----------------------------------------------------------------------- */
/* FUNCTION : alg_f_trenodinc(...)                                         */
/*                                                                         */
/* NOTES    : Calculate node expansion increment for a simple b-ary tree   */
/*                                                                         */
/* INPUTS   : b                      - node fanout                         */
/*          : l                      - number of levels                    */
/*          : nExpCod                - expansion parameter code            */
/*                                                                         */
/* OUTPUTS  : pdMetVal               - metric value                        */
/*                                                                         */
/* CALLS    : alg_m_chkintgtz(...)                                         */
/*          : alg_m_chknulptr(...)                                         */
/*          : alg_m_cllalgfnc(...)                                         */
/*          : alg_f_simpletre(...)                                         */
/*          : utl_f_wrierrlog(...)                                         */
/*                                                                         */
/* RETURNS  : AlgErrCod                                                    */
/* ----------------------------------------------------------------------- */

static
AlgErrCod
alg_f_trenodinc
(
    int          b,
    int          l,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ----------------------------------------------------------------------- */
/* FUNCTION : alg_f_trenodfac(...)                                         */
/*                                                                         */
/* NOTES    : Calculate node expansion factor for a simple b-ary tree      */
/*                                                                         */
/* INPUTS   : b                      - node fanout                         */
/*          : l                      - number of levels                    */
/*          : nExpCod                - expansion parameter code            */
/*                                                                         */
/* OUTPUTS  : pdMetVal               - metric value                        */
/*                                                                         */
/* CALLS    : alg_m_chkintgtz(...)                                         */
/*          : alg_m_chknulptr(...)                                         */
/*          : alg_m_cllalgfnc(...)                                         */
/*          : alg_f_simpletre(...)                                         */
/*          : utl_f_wrierrlog(...)                                         */
/*                                                                         */
/* RETURNS  : AlgErrCod                                                    */
/* ----------------------------------------------------------------------- */

static
AlgErrCod
alg_f_trenodfac
(
    int          b,
    int          l,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ----------------------------------------------------------------------- */
/* FUNCTION : alg_f_trelnkinc(...)                                         */
/*                                                                         */
/* NOTES    : Calculate link expansion increment for a simple b-ary tree   */
/*              - same as node expansion increments, since l = n - 1.0     */
/*                                                                         */
/* INPUTS   : b                      - node fanout                         */
/*          : l                      - number of levels                    */
/*          : nExpCod                - expansion parameter code            */
/*                                                                         */
/* OUTPUTS  : pdMetVal               - metric value                        */
/*                                                                         */
/* CALLS    : alg_m_chkintgtz(...)                                         */
/*          : alg_m_chknulptr(...)                                         */
/*          : alg_m_cllalgfnc(...)                                         */
/*          : alg_f_trenodinc(...)                                         */
/*                                                                         */
/* RETURNS  : AlgErrCod                                                    */
/* ----------------------------------------------------------------------- */

static
AlgErrCod
alg_f_trelnkinc
(
    int          b,
    int          l,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ----------------------------------------------------------------------- */
```

```
/* FUNCTION : alg_f_trelnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a simple b-ary tree       */
/*                                                                          */
/* INPUTS   : b                      - node fanout                          */
/*          : l                      - number of levels                     */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_trenodfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_trelnkfac
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_frtnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for a full-ring b-ary tree */
/*                                                                          */
/* INPUTS   : b                      - node fanout                          */
/*          : l                      - number of levels                     */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_fulrngtre(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_frtnodinc
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_frtnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a full-ring b-ary tree    */
/*                                                                          */
/* INPUTS   : b                      - node fanout                          */
/*          : l                      - number of levels                     */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_fulrngtre(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_frtnodfac
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_frtlnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a full-ring b-ary tree */
/*              - twice node expansion increments, since l = 2.0 * (n - 1.0) */
/*                                                                          */
/* INPUTS   : b                      - node fanout                          */
/*          : l                      - number of levels                     */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_fulnodinc(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_frtlnkinc
```

```
(
    int          b,
    int          l,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_frtlnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a full-ring b-ary tree    */
/*                                                                          */
/* INPUTS   : b                      - node fanout                          */
/*          : l                      - number of levels                     */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_fulrngtre(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_frtlnkfac
(
    int          b,
    int          l,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nnmnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for a nearest-neighbour mesh */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_nnmnodinc
(
    int          w,
    int          d,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nnmnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a nearest-neighbour mesh  */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_nnmnodinc
(
    int          w,
    int          d,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nnmnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a nearest-neighbour mesh  */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
```

```
AlgErrCod
alg_f_nnmnodfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nnmlnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a nearest-neighbour mesh */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_trenodinc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_nnmlnkinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nnmlnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a nearest-neighbour mesh  */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_nnmlnkfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sbhnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for a spanning-bus hypercube */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_sbhnodinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sbhnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a spanning-bus hypercube  */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
```

```
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_sbhnodfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sbhlnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a spanning-bus hypercube */
/*                                                                          */
/* INPUTS   : w                    - width of topology                      */
/*          : d                    - number of dimensions                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_sbhlnkinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sbhlnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a spanning-bus hypercube  */
/*                                                                          */
/* INPUTS   : w                    - width of topology                      */
/*          : d                    - number of dimensions                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
```

```
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_sbhlnkfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_dbhnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for a dual-bus hypercube   */
/*                                                                          */
/* INPUTS   : w                    - width of topology                      */
/*          : d                    - number of dimensions                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_dbhnodinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_dbhnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a dual-bus hypercube      */
/*                                                                          */
/* INPUTS   : w                    - width of topology                      */
/*          : d                    - number of dimensions                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
```

```
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_dbhnodfac
(
    int          w,
    int          d,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_dbhlnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a dual-bus hypercube   */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_dbhlnkinc
(
    int          w,
    int          d,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_dbhlnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a dual-bus hypercube      */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
```

```
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_dbhlnkfac
(
    int          w,
    int          d,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_trsnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for a torus                */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_trsnodinc
(
    int          w,
    int          d,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_trsnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a torus                   */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
```

```
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_trsnodfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_trslnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a torus                */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_trslnkinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_trslnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a torus                   */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
```

```
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_trslnkfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rncnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for an R-ary N-cube        */
/*                                                                          */
/* INPUTS   : R                      - node fanout                          */
/*          : N                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_rncnodinc
(
    int         R,
    int         N,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rncnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for an R-ary N-cube           */
/*                                                                          */
/* INPUTS   : R                      - node fanout                          */
/*          : N                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
```

```
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_rncnodfac
(
    int         R,
    int         N,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rnclnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for an R-ary N-cube        */
/*                                                                          */
/* INPUTS   : R                      - node fanout                          */
/*          : N                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_rnclnkinc
(
    int         R,
    int         N,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rnclnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for an R-ary N-cube           */
/*                                                                          */
/* INPUTS   : R                      - node fanout                          */
/*          : N                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_rnclnkfac
(
    int         R,
    int         N,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
);

/* ------------------------------------------------------------------------ */
#endif
/* ------------------------------------------------------------------------ */

/* ************************************************************************ */
```

C.3.2 tpg_inf.h

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, Univeristy of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : TPG_INF.H                                                      */
/*                                                                            */
/* NOTES     : XLL interfaces for multiprocessor topology analysis functions  */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 18th February, 1995                                            */
/*                                                                            */
/* VERSION   : 1.0                                                            */
/* ************************************************************************** */

/* -------------------------------------------------------------------------- */
#ifndef TPG_INF_H
#define TPG_INF_H
/* -------------------------------------------------------------------------- */

/* ========================================================================== */
/* C headers                                                                  */
/* ========================================================================== */

#include    <stdio.h>
#include    <windows.h>

/* ========================================================================== */
/* Symbolic constants                                                         */
/* ========================================================================== */

#define INF_FNC_RWS 13
#define INF_FNC_CLS  7

/* ========================================================================== */
/* Macro definitions                                                          */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* MACRO     : tpg_m_cllalgfnc(...)                                           */
/*                                                                            */
/* NOTES     : Call topology analysis function                                */
/*                                                                            */
/* INPUTS    : cAlgFnc                - ALG function call                     */
/*                                                                            */
/* OUTPUTS   : <none>                                                         */
/*                                                                            */
/* CALLS     : <cAlgFnc>                                                      */
/*           : utl_f_wrierrlog(...)                                           */
/*           : xlo_f_setxlostr(...)                                           */
/* -------------------------------------------------------------------------- */

#define tpg_m_cllalgfnc( cAlgFnc)                                              \
{                                                                              \
    if ( cAlgFnc != ALG_SUCCESS )                                              \
    {                                                                          \
        utl_f_wrierrlog( "** ALG call failed: %s\n", #cAlgFnc);                \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);            \
                                                                               \
        xlo_f_setxlostr( &xResult, "** unable to calculate metric");           \
                                                                               \
        return( &xResult);                                                     \
    }                                                                          \
}

/* ========================================================================== */
/* Public function definitions                                                */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* FUNCTION : inf_x_fulintcon(...)                                            */
/*                                                                            */
/* NOTES    : XLL interface for full-interconnection analysis function        */
/*                                                                            */
/* INPUTS   : pxNumNod              - number of nodes                         */
/*          : pxMetric              - choice of topology metric               */
/*                                                                            */
/* OUTPUTS  : <none>                                                          */
/*                                                                            */
/* CALLS    : ini_m_xlointgtz(...)                                            */
/*          : ini_m_xlometcod(...)                                            */
/*          : tpg_m_cllalgfnc(...)                                            */
/*          : xlo_m_setxlonum(...)                                            */
/*          : alg_f_fulintcon(...)                                            */
/*                                                                            */
/* RETURNS  : LPXLOPER far pascal                                             */
/* -------------------------------------------------------------------------- */

LPXLOPER far pascal
inf_x_fulintcon
(
    LPXLOPER    pxNumNod,
    LPXLOPER    pxMetric
);

/* -------------------------------------------------------------------------- */
/* FUNCTION : inf_x_sharedbus(...)                                            */
/*                                                                            */
/* NOTES    : XLL interface for shared-bus analysis function                  */
/*                                                                            */
/* INPUTS   : pxNumNod              - number of nodes                         */
/*          : pxMetric              - choice of topology metric               */
/*                                                                            */
/* OUTPUTS  : <none>                                                          */
/*                                                                            */
/* CALLS    : ini_m_xlointgtz(...)                                            */
```

```
/*          : ini_m_xlometcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_sharedbus(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_sharedbus
(
    LPXLOPER    pxNumNod,
    LPXLOPER    pxMetric
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_simplerng(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for simple ring analysis function               */
/*                                                                          */
/* INPUTS   : pxNumNod              - number of nodes                       */
/*          : pxMetric              - choice of topology metric             */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_simplerng(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_simplerng
(
    LPXLOPER    pxNumNod,
    LPXLOPER    pxMetric
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_chordlrng(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for chordal ring analysis function              */
/*                                                                          */
/* INPUTS   : pxNumNod              - number of nodes                       */
/*          : pxMetric              - choice of topology metric             */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_chordlrng(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_chordlrng
(
    LPXLOPER    pxNumNod,
    LPXLOPER    pxMetric
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_simpletre(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for simple b-ary tree analysis function         */
/*                                                                          */
/* INPUTS   : pxFanout              - node fanout                           */
/*          : pxLevels              - number of levels                      */
/*          : pxMetric              - choice of topolgy metric              */
/*          : pxExpnsn              - choice of expansion parameter         */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_simpletre(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_simpletre
(
    LPXLOPER    pxFanout,
    LPXLOPER    pxLevels,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_fulrngtre(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for full-ring or cross-connected b-ary tree     */
/*            analysis function                                             */
/*                                                                          */
/* INPUTS   : pxFanout              - node fanout                           */
/*          : pxLevels              - number of levels                      */
/*          : pxMetric              - choice of topology metric             */
/*          : pxExpnsn              - choice of expansion parameter         */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
```

```
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_fulrngtre(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_fulrngtre
(
    LPXLOPER    pxFanout,
    LPXLOPER    pxLevels,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_nrsnhbmsh(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for nearest-neighbour mesh analysis function    */
/*                                                                          */
/* INPUTS   : pxWidth                - width of topology                    */
/*          : pxDimnsn               - number of dimensions                 */
/*          : pxMetric               - choice of topology metric            */
/*          : pxExpnsn               - choice of expansion parameter        */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_nrsnhbmsh(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_nrsnhbmsh
(
    LPXLOPER    pxWidth,
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_spabushpc(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for spanning-bus hypercube analysis function    */
/*                                                                          */
/* INPUTS   : pxWidth                - width of topology                    */
/*          : pxDimnsn               - number of dimensions                 */
/*          : pxMetric               - choice of topology metric            */
/*          : pxExpnsn               - choice of expansion parameter        */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_spabushpc(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_spabushpc
(
    LPXLOPER    pxWidth,
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_duabushpc(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for dual-bus hypercube analysis function        */
/*                                                                          */
/* INPUTS   : pxWidth                - width of topology                    */
/*          : pxDimnsn               - number of dimensions                 */
/*          : pxMetric               - choice of topology metric            */
/*          : pxExpnsn               - choice of expansion parameter        */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_duabushpc(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_duabushpc
(
    LPXLOPER    pxWidth,
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
);

/* ------------------------------------------------------------------------ */
```

```
/* FUNCTION : inf_x_torus(...)                                          */
/*                                                                      */
/* NOTES    : XLL interface for torus analysis function                 */
/*                                                                      */
/* INPUTS   : pxWidth              - width of topology                  */
/*          : pxDimnsn             - number of dimensions               */
/*          : pxMetric             - choice of topology metric          */
/*          : pxExpnsn             - choice of expansion parameter      */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : ini_m_xlointgtz(...)                                      */
/*          : ini_m_xlometcod(...)                                      */
/*          : ini_m_xloexpcod(...)                                      */
/*          : tpg_m_cllalgfnc(...)                                      */
/*          : xlo_m_setxlonum(...)                                      */
/*          : alg_f_torus(...)                                          */
/*                                                                      */
/* RETURNS  : LPXLOPER far pascal                                       */
/* -------------------------------------------------------------------- */

LPXLOPER far pascal
inf_x_torus
(
    LPXLOPER    pxWidth,
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
);

/* -------------------------------------------------------------------- */
/* FUNCTION : inf_x_raryncube(...)                                      */
/*                                                                      */
/* NOTES    : XLL interface for R-ary N-cube analysis function          */
/*                                                                      */
/* INPUTS   : pxFanout             - node fanout                        */
/*          : pxLevels             - number of levels                   */
/*          : pxMetric             - choice of topology metric          */
/*          : pxExpnsn             - choice of expansion parameter      */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : ini_m_xlointgtz(...)                                      */
/*          : ini_m_xlometcod(...)                                      */
/*          : tpg_m_cllalgfnc(...)                                      */
/*          : xlo_m_setxlonum(...)                                      */
/*          : alg_f_raryncube(...)                                      */
/*                                                                      */
/* RETURNS  : LPXLOPER far pascal                                       */
/* -------------------------------------------------------------------- */

LPXLOPER far pascal
inf_x_raryncube
(
    LPXLOPER    pxFanout,
    LPXLOPER    pxLevels,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
);

/* -------------------------------------------------------------------- */
/* FUNCTION : inf_x_binaryhpc(...)                                      */
/*                                                                      */
/* NOTES    : XLL interface for binary hypercube analysis function      */
/*                                                                      */
/* INPUTS   : pxDimnsn             - number of dimensions               */
/*          : pxMetric             - choice of topology metric          */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : ini_m_xlointgtz(...)                                      */
/*          : ini_m_xlometcod(...)                                      */
/*          : tpg_m_cllalgfnc(...)                                      */
/*          : xlo_m_setxlonum(...)                                      */
/*          : alg_f_binaryhpc(...)                                      */
/*                                                                      */
/* RETURNS  : LPXLOPER far pascal                                       */
/* -------------------------------------------------------------------- */

LPXLOPER far pascal
inf_x_binaryhpc
(
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric
);

/* -------------------------------------------------------------------- */
/* FUNCTION : inf_x_cubconcyc(...)                                      */
/*                                                                      */
/* NOTES    : XLL interface for cube-connected-cycles analysis function */
/*                                                                      */
/* INPUTS   : pxDimnsn             - number of dimensions               */
/*          : pxMetric             - choice of topology metric          */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : ini_m_xlointgtz(...)                                      */
/*          : ini_m_xlometcod(...)                                      */
/*          : tpg_m_cllalgfnc(...)                                      */
/*          : xlo_m_setxlonum(...)                                      */
/*          : alg_f_cubconcyc(...)                                      */
/*                                                                      */
/* RETURNS  : LPXLOPER far pascal                                       */
/* -------------------------------------------------------------------- */

LPXLOPER far pascal
inf_x_cubconcyc
(
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric
);
```

```
/* ------------------------------------------------------------------------- */
#endif
/* ------------------------------------------------------------------------- */

/* ************************************************************************* */
```

C.3.3 tpg_ini.h

```
/* ************************************************************************* */
/* © 1995    : School of Engineering, Univeristy of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England              */
/* ************************************************************************* */

/* ************************************************************************* */
/* MODULE    : TPG_INI.H                                                     */
/*                                                                           */
/* NOTES     : Initialisation routines to convert between XLOPERs and C types */
/*                                                                           */
/* AUTHOR    : Ken N LINTON                                                  */
/*                                                                           */
/* DATE      : 15th February, 1995                                           */
/*                                                                           */
/* VERSION   : 1.0                                                           */
/* ************************************************************************* */

/* ------------------------------------------------------------------------- */
#ifndef TPG_INI_H
#define TPG_INI_H
/* ------------------------------------------------------------------------- */

/* ========================================================================= */
/* TPG includes                                                              */
/* ========================================================================= */

#include    "tpg_alg.h"
#include    "tpg_utl.h"

/* ========================================================================= */
/* Type definitions                                                          */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* TYPEDEF  : enum for INI error codes                                      */
/* ------------------------------------------------------------------------- */

typedef enum IniErrCod_
{
    INI_SUCCESS  = 0,              /* success                  */
    INI_FTL_ERR,                   /* fatal error              */
    INI_MIS_RQD,                   /* missing required XLOPER */
    INI_MIS_OPT                    /* missing optional XLOPER */
} IniErrCod;

/* ------------------------------------------------------------------------- */
/* TYPEDEF  : enum for required/optional XLOPER                             */
/* ------------------------------------------------------------------------- */

typedef enum XloTypCod_
{
    XT_RQD = 0,
```

```
    XT_OPT
} XloTypCod;

/* ================================================================== */
/* Symbolic constants                                                 */
/* ================================================================== */

#define INI_MET_NUM 3
#define INI_EXP_NUM 6

/* ================================================================== */
/* Initialisation macros                                              */
/* ================================================================== */

/* ------------------------------------------------------------------ */
/* MACRO     : ini_m_cllinifnc(...)                                   */
/*                                                                    */
/* NOTES     : Call XLOPER to C type initialisation function          */
/*                                                                    */
/* INPUTS    : cIniFnc                - INI function call             */
/*                                                                    */
/* OUTPUTS   : <none>                                                 */
/*                                                                    */
/* CALLS     : <cIniFnc>                                              */
/*           : utl_f_wrierrlog(...)                                   */
/* ------------------------------------------------------------------ */

#define ini_m_cllinifnc( cIniFnc)                                     \
{                                                                     \
    if ( cIniFnc != INI_SUCCESS )                                     \
    {                                                                 \
        utl_f_wrierrlog( "** INI call failed: %s\n", #cIniFnc);       \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);   \
                                                                      \
        return( INI_FTL_ERR);                                         \
    }                                                                 \
}

/* ------------------------------------------------------------------ */
/* MACRO     : ini_m_chknulptr(...)                                   */
/*                                                                    */
/* NOTES     : Check value pointer is not NULL                        */
/*                                                                    */
/* INPUTS    : ptr                    - a pointer                     */
/*                                                                    */
/* OUTPUTS   : <none>                                                 */
/*                                                                    */
/* CALLS     : <none>                                                 */
/*           : utl_f_wrierrlog(...)                                   */
/* ------------------------------------------------------------------ */

#define ini_m_chknulptr( ptr)                                         \
{                                                                     \
    if ( ptr == NULL )                                                \
    {                                                                 \
        utl_f_wrierrlog( "** NULL pointer: %s\n", #ptr);              \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);   \
                                                                      \
        return( INI_FTL_ERR);                                         \
    }                                                                 \
}

/* ------------------------------------------------------------------ */
/* MACRO     : ini_m_xlointgtz(...)                                   */
/*                                                                    */
/* NOTES     : Call ini_f_xlointgtz(...) and handle return code       */
/*                                                                    */
/* INPUTS    : pxInput                - input XLOPER passed from Excel */
/*           : aInteger               - integer to be initialised     */
/*           : nArgTyp                - argument type                 */
/*                                                                    */
/* OUTPUTS   : pxResult               - error XLOPER passed back to Excel */
/*                                                                    */
/* CALLS     : ini_f_xlointgtz(...)                                   */
/*           : utl_f_wrierrlog(...)                                   */
/* ------------------------------------------------------------------ */

#define ini_m_xlointgtz( pxInput, aDouble, nArgTyp)                   \
{                                                                     \
    if ( ini_f_xlointgtz( pxInput,                                    \
                          nArgTyp,                                    \
                         #aDouble,                                    \
                         &aDouble,                                    \
                         &xResult) != INI_SUCCESS )                   \
    {                                                                 \
        utl_f_wrierrlog( "** ini_f_xlointgtz(...) failed\n");         \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);   \
                                                                      \
        return( &xResult);                                            \
    }                                                                 \
}

/* ------------------------------------------------------------------ */
/* MACRO     : ini_m_xlometcod(...)                                   */
/*                                                                    */
/* NOTES     : Call ini_f_xlometcod(...) and handle return code       */
/*                                                                    */
/* INPUTS    : pxInput                - input XLOPER passed from Excel */
/*           : aMetCod                - metric code to be initialised */
/*           : nArgTyp                - argument type                 */
/*                                                                    */
/* OUTPUTS   : pxResult               - error XLOPER passed back to Excel */
/*                                                                    */
/* CALLS     : ini_f_xlometcod(...)                                   */
/*           : utl_f_wrierrlog(...)                                   */
/* ------------------------------------------------------------------ */

#define ini_m_xlometcod( pxInput, aMetCod, nArgTyp)                   \
{                                                                     \
    if ( ini_f_xlometcod( pxInput,                                    \
                          nArgTyp,                                    \
                         #aMetCod,                                    \
```

```
                        &aMetCod,                                          \
                        &xResult) != INI_SUCCESS )                         \
    {                                                                      \
        utl_f_wrierrlog( "** ini_f_xlometcod(...) failed\n");              \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);        \
                                                                           \
        return( &xResult);                                                 \
    }                                                                      \
}

/* ------------------------------------------------------------------------ */
/* MACRO     : ini_m_xloexpcod(...)                                         */
/*                                                                          */
/* NOTES     : Call ini_f_xloexpcod(...) and handle return code             */
/*                                                                          */
/* INPUTS    : pxInput                - input XLOPER passed from Excel      */
/*           : aExpCod                - expansion parameter to be initialised */
/*           : nArgTyp                - argument type                       */
/*                                                                          */
/* OUTPUTS   : pxResult               - error XLOPER passed back to Excel   */
/*                                                                          */
/* CALLS     : ini_f_xloexpcod(...)                                         */
/*           : utl_f_wrierrlog(...)                                         */
/* ------------------------------------------------------------------------ */

#define ini_m_xloexpcod( pxInput, aExpCod, nArgTyp)                        \
{                                                                          \
    if ( ini_f_xloexpcod( pxInput,                                         \
                          nArgTyp,                                         \
                         #aExpCod,                                         \
                         &aExpCod,                                         \
                         &xResult) != INI_SUCCESS )                        \
    {                                                                      \
        utl_f_wrierrlog( "** ini_f_xloexpcod(...) failed\n");              \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);        \
                                                                           \
        return( &xResult);                                                 \
    }                                                                      \
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_chkxlomis(...)                                          */
/*                                                                          */
/* NOTES    : Check for missing argument passed in by Excel                 */
/*                                                                          */
/* INPUTS   : pxInput                - input XLOPER passed from Excel       */
/*          : nArgTyp                - argument type                        */
/*          : szInput                - stringized input C-type              */
/*                                                                          */
/* OUTPUTS  : pxResult               - error XLOPER passed back to Excel    */
/*                                                                          */
/* CALLS    : ini_m_chknulptr(...)                                          */
/*          : xlo_m_getxlotyp(...)                                          */
/*          : xlo_f_setxlostr(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

#define ini_m_chkxlomis( pxInput, nArgTyp, szInput, pxResult)              \
{                                                                          \
    IniErrCod   err_cod = INI_SUCCESS;                                     \
                                                                           \
    err_cod = ini_f_chkxlomis( pxInput, nArgTyp, szInput, pxResult);       \
                                                                           \
    if ( err_cod != INI_SUCCESS )                                          \
    {                                                                      \
        switch ( err_cod)                                                  \
        {                                                                  \
            case INI_MIS_OPT:                                              \
            {                                                              \
                /* missing parameter is optional */                        \
                return( INI_SUCCESS);                                      \
            }                                                              \
            case INI_MIS_RQD:                                              \
            {                                                              \
                /* missing parameter is required */                        \
                return( INI_FTL_ERR);                                      \
            }                                                              \
            default:                                                       \
            {                                                              \
                /* ini_f_chkxlomis(...) failed */                          \
                return( INI_FTL_ERR);                                      \
            }                                                              \
        }                                                                  \
    }                                                                      \
}

/* ========================================================================= */
/* Public functions                                                          */
/* ========================================================================= */

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_xlointgtz(...)                                          */
/*                                                                          */
/* NOTES    : Initialise greater-than-zero integer from XLOPER              */
/*                                                                          */
/* INPUTS   : pxInput                - input XLOPER passed from Excel       */
/*          : nArgTyp                - argument type                        */
/*          : szInput                - stringized input C-type              */
/*                                                                          */
/* OUTPUTS  : pInteger               - integer initialised from XLOPER      */
/*          : pxResult               - error XLOPER passed back to Excel    */
/*                                                                          */
/* CALLS    : ini_m_chkxlomis(...)                                          */
/*          : ini_m_cllinifnc(...)                                          */
/*          : xlo_m_getxloint(...)                                          */
/*          : Excel(...)                                                    */
/*          : ini_f_chkintgtz(...)                                          */
/*          : ini_f_chkxloerr(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */
```

```
IniErrCod
ini_f_xlointgtz
(
    const   LPXLOPER    pxInput,
            XloTypCod   nArgTyp,
    const   char       *szInput,
            int        *pInteger,
            LPXLOPER    pxResult
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_xlometcod(...)                                          */
/*                                                                          */
/* NOTES    : Initialise metric code from XLOPER                            */
/*                                                                          */
/* INPUTS   : pxInput                - input XLOPER passed from Excel       */
/*          : nArgTyp                - argument type                        */
/*          : szInput                - stringized input C-type              */
/*                                                                          */
/* OUTPUTS  : pMetCod                - metric code initialised from XLOPER  */
/*          : pxResult               - error XLOPER passed back to Excel    */
/*                                                                          */
/* CALLS    : ini_m_chkxlomis(...)                                          */
/*          : ini_m_cllinifnc(...)                                          */
/*          : xlo_m_getxlostr(...)                                          */
/*          : Excel(...)                                                    */
/*          : ini_f_chkxloerr(...)                                          */
/*          : ini_f_cnvmetcod(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

IniErrCod
ini_f_xlometcod
(
    const   LPXLOPER    pxInput,
            XloTypCod   nArgTyp,
    const   char       *szInput,
            MetricCod  *pMetCod,
            LPXLOPER    pxResult
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_xloexpcod(...)                                          */
/*                                                                          */
/* NOTES    : Initialise expansion parameter code from XLOPER               */
/*                                                                          */
/* INPUTS   : pxInput                - input XLOPER passed from Excel       */
/*          : nArgTyp                - argument type                        */
/*          : szInput                - stringized input C-type              */
/*                                                                          */
/* OUTPUTS  : pExpCod                - expansion parameter code initialised */
/*                                     from XLOPER                          */
/*          : pxResult               - error XLOPER passed back to Excel    */
/*                                                                          */
/* CALLS    : ini_m_chkxlomis(...)                                          */
/*          : ini_m_cllinifnc(...)                                          */
/*          : xlo_m_getxlostr(...)                                          */
/*          : Excel(...)                                                    */
/*          : ini_f_chkxloerr(...)                                          */
/*          : ini_f_cnvexpcod(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

IniErrCod
ini_f_xloexpcod
(
    const   LPXLOPER    pxInput,
            XloTypCod   nArgTyp,
    const   char       *szInput,
            ExpnsnCod  *pExpCod,
            LPXLOPER    pxResult
);

/* ======================================================================== */
/* Private function prototypes                                              */
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_matstrcod(...)                                          */
/*                                                                          */
/* NOTES    : Match string against known strings for a given code           */
/*                                                                          */
/* INPUTS   : szString               - z-string                             */
/*          : nCode                  - code value                           */
/*          : aStrAry                - array of input strings               */
/*          : nNumStr                - number of strings for each code      */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : <none>                                                        */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
IniErrCod
ini_f_matstrcod
(
    const   char       *szString,
            int         nCode,
    const   char       *aStrAry[],
            int         nNumStr
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_chkxlomis(...)                                          */
/*                                                                          */
/* NOTES    : Check for missing argument passed in by Excel                 */
/*                                                                          */
```

```
/* INPUTS   : pxInput                - input XLOPER passed from Excel        */
/*          : nArgTyp                - argument type                         */
/*          : szInput                - stringized input C-type               */
/*                                                                           */
/* OUTPUTS  : pxResult               - error XLOPER passed back to Excel     */
/*                                                                           */
/* CALLS    : ini_m_chknulptr(...)                                           */
/*          : xlo_m_getxlotyp(...)                                           */
/*          : utl_f_wrierrlog(...)                                           */
/*          : xlo_f_setxlostr(...)                                           */
/*                                                                           */
/* RETURNS  : IniErrCod                                                      */
/* ------------------------------------------------------------------------- */

static
IniErrCod
ini_f_chkxlomis
(
    const   LPXLOPER    pxInput,
            XloTypCod   nArgTyp,
    const   char       *szInput,
            LPXLOPER    pxResult
);

/* ------------------------------------------------------------------------- */
/* FUNCTION : ini_f_chkxloerr(...)                                           */
/*                                                                           */
/* NOTES    : Check if XLOPER is an Excel error code                         */
/*                                                                           */
/* INPUTS   : pxInput                - input XLOPER passed from Excel        */
/*          : szInput                - stringized input C-type               */
/*                                                                           */
/* OUTPUTS  : pxResult               - error XLOPER passed back to Excel     */
/*                                                                           */
/* CALLS    : ini_m_chknulptr(...)                                           */
/*          : xlo_m_getxlotyp(...)                                           */
/*          : ini_f_cnverrcod(...)                                           */
/*          : utl_f_wrierrlog(...)                                           */
/*          : xlo_f_setxlostr(...)                                           */
/*                                                                           */
/* RETURNS  : IniErrCod                                                      */
/* ------------------------------------------------------------------------- */

static
IniErrCod
ini_f_chkxloerr
(
    const   LPXLOPER    pxInput,
    const   char       *szInput,
            LPXLOPER    pxResult
);

/* ------------------------------------------------------------------------- */
/* FUNCTION : ini_f_chkintgtz(...)                                           */
/*                                                                           */
/* NOTES    : Check integer is greater than zero                             */
/*                                                                           */
/* INPUTS   : nInput                 - integer value from input XLOPER       */
/*          : szInput                - stringized input C-type               */
/*                                                                           */
/* OUTPUTS  : pxResult               - error XLOPER passed back to Excel     */
/*                                                                           */
/* CALLS    : ini_m_chknulptr(...)                                           */
/*          : utl_f_wrierrlog(...)                                           */
/*          : xlo_f_setxlostr(...)                                           */
/*                                                                           */
/* RETURNS  : IniErrCod                                                      */
/* ------------------------------------------------------------------------- */

static
IniErrCod
ini_f_chkintgtz
(
            int         nInput,
    const   char       *szInput,
            LPXLOPER    pxResult
);

/* ------------------------------------------------------------------------- */
/* FUNCTION : ini_f_cnverrcod(...)                                           */
/*                                                                           */
/* NOTES    : Convert XLOPER to an Excel error code                          */
/*                                                                           */
/* INPUTS   : pxInput                - input XLOPER passed from Excel        */
/*                                                                           */
/* OUTPUTS  : szErrStr               - Excel error string                    */
/*                                                                           */
/* CALLS    : xlo_m_chknulptr(...)                                           */
/*          : xlo_m_getxlotyp(...)                                           */
/*          : utl_f_wrierrlog(...)                                           */
/*                                                                           */
/* RETURNS  : IniErrCod                                                      */
/* ------------------------------------------------------------------------- */

static
IniErrCod
ini_f_cnverrcod
(
    const   LPXLOPER    pxInput,
            char       *szErrStr
);

/* ------------------------------------------------------------------------- */
/* FUNCTION : ini_f_cnvmetcod(...)                                           */
/*                                                                           */
/* NOTES    : Convert z-string to a metric code                              */
/*                                                                           */
/* INPUTS   : szString               - string from input XLOPER              */
/*          : szInput                - stringized input C-type               */
/*                                                                           */
/* OUTPUTS  : pnMetCod               - metric code                           */
/*          : pxResult               - error XLOPER passed back to Excel     */
```

```
/*                                                                        */
/* CALLS     : ini_m_chknulptr(...)                                       */
/*           : ini_f_matstrcod(...)                                       */
/*           : utl_f_wrierrlog(...)                                       */
/*           : xlo_f_setxlostr(...)                                       */
/*                                                                        */
/* RETURNS   : IniErrCod                                                  */
/* ---------------------------------------------------------------------- */

static
IniErrCod
ini_f_cnvmetcod
(
    const   char       *szString,
    const   char       *szInput,
            MetricCod  *pnMetCod,
            LPXLOPER    pxResult
);

/* ---------------------------------------------------------------------- */
/* FUNCTION : ini_f_cnvexpcod(...)                                        */
/*                                                                        */
/* NOTES    : Convert z-string to an expansion parameter code             */
/*                                                                        */
/* INPUTS   : szString              - string from input XLOPER            */
/*          : szInput               - stringized input C-type             */
/*                                                                        */
/* OUTPUTS  : pnExpCod              - expansion parameter code            */
/*          : pxResult              - error XLOPER passed back to Excel   */
/*                                                                        */
/* CALLS    : ini_m_chknulptr(...)                                        */
/*          : ini_f_matstrcod(...)                                        */
/*          : utl_f_wrierrlog(...)                                        */
/*          : xlo_f_setxlostr(...)                                        */
/*                                                                        */
/* RETURNS  : IniErrCod                                                   */
/* ---------------------------------------------------------------------- */

static
IniErrCod
ini_f_cnvexpcod
(
    const   char       *szString,
    const   char       *szInput,
            ExpnsnCod  *pnExpCod,
            LPXLOPER    pxResult
);

/* ---------------------------------------------------------------------- */
#endif
/* ---------------------------------------------------------------------- */

/* ********************************************************************** */
```

C.3.4 tpg_utl.h

```
/* ********************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ********************************************************************** */

/* ********************************************************************** */
/* MODULE    : TPG_UTL.H                                                  */
/*                                                                        */
/* NOTES     : Utility functions for topology analysis XLL                */
/*                                                                        */
/* AUTHOR    : Ken N LINTON                                               */
/*                                                                        */
/* DATE      : 10th February, 1995                                        */
/*                                                                        */
/* VERSION   : 1.0                                                        */
/* ********************************************************************** */

/* ---------------------------------------------------------------------- */
#ifndef TPG_UTL_H
#define TPG_UTL_H
/* ---------------------------------------------------------------------- */

/* ====================================================================== */
/* Symbolic constants                                                     */
/* ====================================================================== */

#define UTL_FIL_NAM "c:\\tpg.log"
#define UTL_STR_MAX 512

/* ====================================================================== */
/* Type definitions                                                       */
/* ====================================================================== */

/* ---------------------------------------------------------------------- */
/* TYPEDEF   : enum for UTL error codes                                   */
/* ---------------------------------------------------------------------- */

typedef enum UtlErrCod_
{
    UTL_SUCCESS  = 0,
    UTL_FTL_ERR
} UtlErrCod;

/* ====================================================================== */
/* Macro definitions                                                      */
/* ====================================================================== */

/* ---------------------------------------------------------------------- */
/* MACRO     : utl_m_chknulptr(...)                                       */
/*                                                                        */
/* NOTES     : Check value pointer is not NULL                            */
/*                                                                        */
```

```
/* INPUTS    : ptr                    - a pointer                          */
/*                                                                         */
/* OUTPUTS   : <none>                                                      */
/*                                                                         */
/* CALLS     : <none>                                                      */
/* ----------------------------------------------------------------------- */

#define utl_m_chknulptr( ptr)                                              \
{                                                                          \
    if ( ptr == NULL )                                                     \
    {                                                                      \
        return( UTL_FTL_ERR);                                              \
    }                                                                      \
}

/* ======================================================================= */
/* Public function prototypes                                              */
/* ======================================================================= */

/* ----------------------------------------------------------------------- */
/* FUNCTION : utl_f_creerrlog(...)                                         */
/*                                                                         */
/* NOTES    : Create error log file and backup previous version, if it exists */
/*                                                                         */
/* INPUTS   : szLogFil               - error log filename, including full path */
/*                                                                         */
/* OUTPUTS  : <none>                                                       */
/*                                                                         */
/* CALLS    : utl_m_chknulptr(...)                                         */
/*                                                                         */
/* RETURNS  : UtlErrCod                                                    */
/* ----------------------------------------------------------------------- */

UtlErrCod
utl_f_creerrlog
(
    const   char   *szLogFil
);

/* ----------------------------------------------------------------------- */
/* FUNCTION : utl_f_wrierrlog(...)                                         */
/*                                                                         */
/* NOTES    : Write formatting and optional parameters to error log file   */
/*                                                                         */
/* INPUTS   : szFmtStr               - formatting string                   */
/*          : ...                    - optional parameters                 */
/*                                                                         */
/* OUTPUTS  : <none>                                                       */
/*                                                                         */
/* CALLS    : utl_m_chknulptr(...)                                         */
/*                                                                         */
/* RETURNS  : UtlErrCod                                                    */
/* ----------------------------------------------------------------------- */

UtlErrCod
utl_f_wrierrlog
(
    const   char   *szFmtStr,
            ...
);

/* ----------------------------------------------------------------------- */
#endif
/* ----------------------------------------------------------------------- */

/* ******************************************************************************** */
```

C.3.5 tpg_win.h

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, Univeristy of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : TPG_WIN.H                                                      */
/*                                                                            */
/* NOTES     : Mandatory functions for stand-alone windows DLL                */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 17th February, 1995                                            */
/*                                                                            */
/* VERSION   : 1.0                                                            */
/* ************************************************************************** */

/* -------------------------------------------------------------------------- */
#ifndef TPG_WIN_H
#define TPG_WIN_H
/* -------------------------------------------------------------------------- */

/* ========================================================================== */
/* SDK headers                                                                */
/* ========================================================================== */

#include    <windows.h>
#include    <xlcall.h>

/* ========================================================================== */
/* Type definitions                                                           */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* TYPEDEF   : Windows error codes                                            */
/* -------------------------------------------------------------------------- */

typedef enum WinErrCod_
{
    WIN_FTL_ERR = 0,
    WIN_SUCCESS = 1
} WinErrCod;

/* ========================================================================== */
/* Public function prototypes                                                 */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* FUNCTION : LibMain(...)                                                    */
/*                                                                            */
/* NOTES     : Initialise interface descriptor strings and open error log file */
/*                                                                            */
/* INPUTS    : <none>                                                         */
/*                                                                            */
/* OUTPUTS   : <none>                                                         */
/*                                                                            */
/* CALLS     : <none>                                                         */
/*                                                                            */
/* RETURNS   : int far pascal                                                 */
/* -------------------------------------------------------------------------- */

int far pascal
LibMain
(
    void
);

/* -------------------------------------------------------------------------- */
/* FUNCTION : WEP(...)                                                        */
/*                                                                            */
/* NOTES     : Windows exit procedure: close error log file before XLL unload */
/*                                                                            */
/* INPUTS    : <none>                                                         */
/*                                                                            */
/* OUTPUTS   : <none>                                                         */
/*                                                                            */
/* CALLS     : <none>                                                         */
/*                                                                            */
/* RETURNS   : int far pascal                                                 */
/* -------------------------------------------------------------------------- */

int far pascal
WEP
(
    void
);

/* -------------------------------------------------------------------------- */
/* FUNCTION : xlAutoOpen(...)                                                 */
/*                                                                            */
/* NOTES     : Register all add-in functions on opening XLL                   */
/*                                                                            */
/* INPUTS    : <none>                                                         */
/*                                                                            */
/* OUTPUTS   : <none>                                                         */
/*                                                                            */
/* CALLS     : <none>                                                         */
/*                                                                            */
/* RETURNS   : int far pascal                                                 */
/* -------------------------------------------------------------------------- */

int far pascal
xlAutoOpen
(
    void
);

/* -------------------------------------------------------------------------- */
```

```
/* FUNCTION : xlAutoClose(...)                                          */
/*                                                                      */
/* NOTES    : Un-register all add-in functions on closing XLL           */
/*                                                                      */
/* INPUTS   : <none>                                                    */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : <none>                                                    */
/*                                                                      */
/* RETURNS  : int far pascal                                            */
/* -------------------------------------------------------------------- */

int far pascal
xlAutoClose
(
    void
);

/* -------------------------------------------------------------------- */
/* FUNCTION : xlAutoRegister(...)                                       */
/*                                                                      */
/* NOTES    : Re-register all add-in functions on re-loading XLL        */
/*                                                                      */
/* INPUTS   : pxName                - add-in function name              */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : <none>                                                    */
/*                                                                      */
/* RETURNS  : LPXLOPER far pascal                                       */
/* -------------------------------------------------------------------- */

LPXLOPER far pascal
xlAutoRegister
(
    const   LPXLOPER    pxName
);

/* -------------------------------------------------------------------- */
/* FUNCTION : xlAutoAdd(...)                                            */
/*                                                                      */
/* NOTES    : Add XLL to start-up .INI file using Excel add-in manager  */
/*                                                                      */
/* INPUTS   : <none>                                                    */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : <none>                                                    */
/*                                                                      */
/* RETURNS  : int far pascal                                            */
/* -------------------------------------------------------------------- */

int far pascal
xlAutoAdd
(
    void
);

/* -------------------------------------------------------------------- */
/* FUNCTION : xlAutoRemove(...)                                         */
/*                                                                      */
/* NOTES    : Remove XLL from start-up .INI file via Excel add-in manager */
/*                                                                      */
/* INPUTS   : <none>                                                    */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : <none>                                                    */
/*                                                                      */
/* RETURNS  : int far pascal                                            */
/* -------------------------------------------------------------------- */

int far pascal
xlAutoRemove
(
    void
);

/* -------------------------------------------------------------------- */
/* FUNCTION : xlAddInManagerInfo(...)                                   */
/*                                                                      */
/* NOTES    : Get XLL description for Excel add-in manager edit dialog  */
/*                                                                      */
/* INPUTS   : xAction               - Excel add-in manager command      */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : <none>                                                    */
/*                                                                      */
/* RETURNS  : LPXLOPER far pascal                                       */
/* -------------------------------------------------------------------- */

LPXLOPER far pascal
xlAddInManagerInfo
(
    const   LPXLOPER xAction
);

/* -------------------------------------------------------------------- */
#endif
/* -------------------------------------------------------------------- */

/* ******************************************************************** */
```

C.3.6 tpg_xlo.h

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : TPG_XLO.H                                                      */
/*                                                                            */
/* NOTES     : Access functions/macros for Excel XLOPER structure             */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 17th February, 1995                                            */
/*                                                                            */
/* VERSION   : 1.0                                                            */
/* ************************************************************************** */

/* -------------------------------------------------------------------------- */
#ifndef TPG_XLO_H
#define TPG_XLO_H
/* -------------------------------------------------------------------------- */

/* ========================================================================== */
/* TPG headers                                                                */
/* ========================================================================== */

#include    "tpg_utl.h"

/* ========================================================================== */
/* Symbolic constants                                                         */
/* ========================================================================== */

#define XLO_STR_MAX 256

/* ========================================================================== */
/* Type definitions                                                           */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* TYPEDEF   : enum for XLO error codes                                     */
/* -------------------------------------------------------------------------- */

typedef enum XloErrCod_
{
    XLO_SUCCESS  = 0,
    XLO_FTL_ERR
} XloErrCod;

/* ========================================================================== */
/* Macro definitions                                                          */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* MACRO     : xlo_m_chknulptr(...)                                           */
/*                                                                            */
/* NOTES     : Check value pointer is not NULL                                */
/*                                                                            */
/* INPUTS    : ptr                    - a pointer                             */
/*                                                                            */
/* OUTPUTS   : <none>                                                         */
/*                                                                            */
/* CALLS     : <none>                                                         */
/* -------------------------------------------------------------------------- */

#define xlo_m_chknulptr( ptr)                                                 \
{                                                                             \
    if ( ptr == NULL )                                                        \
    {                                                                         \
        utl_f_wrierrlog( "** NULL pointer: %s\n", #ptr);                      \
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);           \
                                                                              \
        return( XLO_FTL_ERR);                                                 \
    }                                                                         \
}

/* -------------------------------------------------------------------------- */
/* MACRO     : xlo_m_setxlotyp(...)                                           */
/*                                                                            */
/* NOTES     : Set XLOPER type                                                */
/*                                                                            */
/* INPUTS    : type                   - XLOPER type code                      */
/*                                                                            */
/* OUTPUTS   : pxType                 - XLOPER with type set                  */
/*                                                                            */
/* CALLS     : <none>                                                         */
/* -------------------------------------------------------------------------- */

#define xlo_m_setxlotyp( type, pxType)      ( (pxType)->xltype = type )

/* -------------------------------------------------------------------------- */
/* MACRO     : xlo_m_getxloint(...)                                           */
/*                                                                            */
/* NOTES     : Get XLOPER type                                                */
/*                                                                            */
/* INPUTS    : pxType                 - XLOPER                                */
/*                                                                            */
/* OUTPUTS   : <none>                                                         */
/*                                                                            */
/* CALLS     : <none>                                                         */
/* -------------------------------------------------------------------------- */

#define xlo_m_getxlotyp( pxType)            ( (pxType)->xltype )

/* -------------------------------------------------------------------------- */
/* MACRO     : xlo_m_setxloint(...)                                           */
/*                                                                            */
/* NOTES     : Set XLOPER integer contents and type to xltypeInt              */
/*                                                                            */
```

```
/* INPUTS    : integer                - a short integer                    */
/*                                                                         */
/* OUTPUTS   : pxInteger              - XLOPER containing a short integer */
/*                                                                         */
/* CALLS     : <none>                                                      */
/* ----------------------------------------------------------------------- */

#define xlo_m_setxloint( integer, pxInteger)                              \
{                                                                         \
    (pxInteger)->xltype = xltypeInt;                                      \
    (pxInteger)->val.w  = (integer);                                      \
}                                                                         \

/* ----------------------------------------------------------------------- */
/* MACRO     : xlo_m_getxloint(...)                                        */
/*                                                                         */
/* NOTES     : Get XLOPER contents as a short integer                      */
/*                                                                         */
/* INPUTS    : pxInteger              - XLOPER containing a short integer */
/*                                                                         */
/* OUTPUTS   : <none>                                                      */
/*                                                                         */
/* CALLS     : <none>                                                      */
/* ----------------------------------------------------------------------- */

#define xlo_m_getxloint( pxInteger)          ( (pxInteger)->val.w )

/* ----------------------------------------------------------------------- */
/* MACRO     : xlo_m_setxlonum(...)                                        */
/*                                                                         */
/* NOTES     : Set XLOPER number contents and type to xltypeNum            */
/*                                                                         */
/* INPUTS    : number                 - a double                           */
/*                                                                         */
/* INPUT     : pxNumber               - XLOPER containing a number         */
/*                                                                         */
/* CALLS     : <none>                                                      */
/* ----------------------------------------------------------------------- */

#define xlo_m_setxlonum( number, pxNumber)                                \
{                                                                         \
    (pxNumber)->xltype  = xltypeNum;                                      \
    (pxNumber)->val.num = (number);                                       \
}                                                                         \

/* ----------------------------------------------------------------------- */
/* MACRO     : xlo_m_getxlonum(...)                                        */
/*                                                                         */
/* NOTES     : Get XLOPER contents as a double                             */
/*                                                                         */
/* INPUTS    : pxNumber               - XLOPER containing a number         */
/*                                                                         */
/* OUTPUTS   : <none>                                                      */
/*                                                                         */
/* CALLS     : <none>                                                      */
/* ----------------------------------------------------------------------- */
```

```
#define xlo_m_getxlonum( pxNumber)           ( (pxNumber)->val.num )

/* ----------------------------------------------------------------------- */
/* MACRO     : xlo_m_setxloerr(...)                                        */
/*                                                                         */
/* NOTES     : Set XLOPER error contents and type to xltypeErr             */
/*                                                                         */
/* INPUTS    : error                  - an Excel error code                */
/*                                                                         */
/* OUTPUTS   : pxError                - XLOPER containing an Excel error code */
/*                                                                         */
/* CALLS     : <none>                                                      */
/* ----------------------------------------------------------------------- */

#define xlo_m_setxloerr( error, pxError)                                  \
{                                                                         \
    (pxError)->xltype   = xltypeErr;                                      \
    (pxError)->val.err  = (error);                                        \
}                                                                         \

/* ----------------------------------------------------------------------- */
/* MACRO     : xlo_m_getxloerr(...)                                        */
/*                                                                         */
/* NOTES     : Set XLOPER contents as an Excel error code                  */
/*                                                                         */
/* INPUTS    : pxError                - XLOPER containing an Excel error code */
/*                                                                         */
/* OUTPUTS   : <none>                                                      */
/*                                                                         */
/* CALLS     : <none>                                                      */
/* ----------------------------------------------------------------------- */

#define xlo_m_getxloerr( pxError)            ( (pxError)->val.err )

/* ======================================================================= */
/* Public function prototypes                                              */
/* ======================================================================= */

/* ----------------------------------------------------------------------- */
/* FUNCTION  : xlo_f_setxlostr(...)                                        */
/*                                                                         */
/* NOTES     : Set XLOPER string contents and type to xltypeStr            */
/*                                                                         */
/* INPUTS    : szString               - zero-terminated string             */
/*           : ...                    - optional arguments                 */
/*                                                                         */
/* OUTPUTS   : pxString               - XLOPER containing a byte-counted string */
/*                                                                         */
/* CALLS     : <none>                                                      */
/*                                                                         */
/* RETURNS   : XloErrCod                                                   */
/* ----------------------------------------------------------------------- */

XloErrCod
xlo_f_setxlostr
```

```
(
            LPXLOPER    pxString,
    const   char       *szString,
            ...
);

/* ------------------------------------------------------------------------ */
/* FUNCTION : xlo_f_getxlostr(...)                                          */
/*                                                                          */
/* NOTES    : Get XLOPER string contents as a zero-terminated string        */
/*                                                                          */
/* INPUTS   : pxString              - XLOPER containing a byte-counted string */
/*                                                                          */
/* OUTPUTS  : szString              - zero-terminated string                */
/*                                                                          */
/* CALLS    : <none>                                                        */
/*                                                                          */
/* RETURNS  : XloErrCod                                                     */
/* ------------------------------------------------------------------------ */

XloErrCod
xlo_f_getxlostr
(
    const   LPXLOPER    pxString,
            char       *szString
);

/* ------------------------------------------------------------------------ */
#endif
/* ------------------------------------------------------------------------ */

/* ************************************************************************ */
```

C.4 Source files

Source code for the *Topology Analysis Tool* is listed below.

C.4.1 tpg_alg.c

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : TPG_ALG.C                                                    */
/*                                                                          */
/* NOTES     : Analysis functions for multiprocessor topologies             */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 16th February, 1995                                          */
/*                                                                          */
/* VERSION   : 1.0                                                          */
/* ************************************************************************ */

/* ======================================================================== */
/* C headers                                                                */
/* ======================================================================== */

#include    <math.h>
#include    <stdlib.h>

/* ======================================================================== */
/* TPG headers                                                              */
/* ======================================================================== */

#include    "tpg_alg.h"
#include    "tpg_utl.h"

/* ======================================================================== */
/* Public function definitions                                              */
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_fulintcon(...)                                          */
/*                                                                          */
/* NOTES    : Full-interconnection topology analysis function               */
/*                                                                          */
/* INPUTS   : n                      - number of nodes                      */
/*          : nMetCod                - topology metric code                 */
/*                                                                          */
```

```
/* OUTPUTS   : pdMetVal              - metric value                    */
/*                                                                     */
/* CALLS     : SELF                                                    */
/*           : alg_m_chkintgtz(...)                                    */
/*           : alg_m_chknulptr(...)                                    */
/*           : alg_m_cllalgfnc(...)                                    */
/*           : utl_f_wrierrlog(...)                                    */
/*                                                                     */
/* RETURNS   : AlgErrCod                                               */
/* ------------------------------------------------------------------- */

AlgErrCod
alg_f_fulintcon
(
    int          n,
    MetricCod    nMetCod,
    double      *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( n        );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal)  = n;

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = n * (n - 1.0);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = n * (n - 1.0) / 2.0;

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_NOD_FAC:
        {
            /* node expansion factor */
            (*pdMetVal)  = (n + 1.0) / n;

            break;
        }
        case MT_LNK_INC:
        {
            /* link expansion increment */
            (*pdMetVal)  = n;

            break;
        }
        case MT_LNK_FAC:
        {
            /* link expansion factor */
            if ( n == 1)
            {
                /* catch divide-by-zero */
                (*pdMetVal) = 1.0;
            }
            else
            {
                /* default case */
                (*pdMetVal)  = (n + 1.0) / (n - 1.0);
            }

            break;
        }
        case MT_LNK_CON:
        {
            /* link conectivity */
            (*pdMetVal)  = n - 1.0;

            break;
        }
        case MT_NOD_CON:
        {
            /* node connectivity */
            (*pdMetVal)  = n - 1.0;

            break;
        }
        case MT_TOP_DIA:
        {
            /* topology diameter */
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_VALENCY:
        {
            /* valency */
            (*pdMetVal)  = n - 1.0;

            break;
        }
```

```
        case MT_DV_PROD:
        {
            /* diameter.valency product */
            double  top_dia = 0.0;
            double  valency = 0.0;

            alg_m_cllalgfnc( alg_f_fulintcon( n, MT_TOP_DIA, &top_dia));
            alg_m_cllalgfnc( alg_f_fulintcon( n, MT_VALENCY, &valency));

            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sharedbus(...)                                          */
/*                                                                          */
/* NOTES    : Shared-bus topology analysis function                         */
/*                                                                          */
/* INPUTS   : n                      - number of nodes                      */
/*          : nMetCod                - topology metric code                 */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_sharedbus
(
    int         n,
    MetricCod   nMetCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( n        );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal)  = n;

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = n;

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_NOD_FAC:
        {
            /* node expansion factor */
            (*pdMetVal)  = (n + 1.0) / n;

            break;
        }
        case MT_LNK_INC:
        {
            /* link expansion increment */
            (*pdMetVal)  = 0.0;

            break;
        }
        case MT_LNK_FAC:
        {
            /* link expansion factor */
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_LNK_CON:
        {
            /* link conectivity */
            (*pdMetVal)  = 1.0;
```

```
            break;
        }
        case MT_NOD_CON:
        {
            /* node connectivity */
            (*pdMetVal)  = n;

            break;
        }
        case MT_TOP_DIA:
        {
            /* topology diameter */
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_VALENCY:
        {
            /* valency */
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_DV_PROD:
        {
            /* diameter.valency product */
            double  top_dia = 0.0;
            double  valency = 0.0;

            alg_m_cllalgfnc( alg_f_sharedbus( n, MT_TOP_DIA, &top_dia));
            alg_m_cllalgfnc( alg_f_sharedbus( n, MT_VALENCY, &valency));

            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_simplerng(...)                                          */
/*                                                                          */
/* NOTES    : Simple ring topology analysis function                        */
/*                                                                          */
/* INPUTS   : n                      - number of nodes                      */
/*          : nMetCod                - topology metric code                 */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_simplerng
(
    int         n,
    MetricCod   nMetCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( n        );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal)  = n;

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = 2.0 * n;

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = n;

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_NOD_FAC:
```

```
{
    /* node expansion factor */
    (*pdMetVal)  = (n + 1.0) / n;

    break;
}
case MT_LNK_INC:
{
    /* link expansion increment */
    (*pdMetVal)  = 1.0;

    break;
}
case MT_LNK_FAC:
{
    /* link expansion factor */
    (*pdMetVal)  = (n + 1.0) / n;

    break;
}
case MT_LNK_CON:
{
    /* link conectivity */
    (*pdMetVal)  = 2.0;

    break;
}
case MT_NOD_CON:
{
    /* node connectivity */
    (*pdMetVal)  = 2.0;

    break;
}
case MT_TOP_DIA:
{
    /* topology diameter */
    (*pdMetVal)  = floor( n / 2.0);

    break;
}
case MT_VALENCY:
{
    /* valency */
    (*pdMetVal)  = 2.0;

    break;
}
case MT_DV_PROD:
{
    /* diameter.valency product */
    double  top_dia = 0.0;
    double  valency = 0.0;

    alg_m_cllalgfnc( alg_f_simplerng( n, MT_TOP_DIA, &top_dia));
    alg_m_cllalgfnc( alg_f_simplerng( n, MT_VALENCY, &valency));

            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_chordlrng(...)                                          */
/*                                                                          */
/* NOTES    : Chordal ring topology analysis function                       */
/*                                                                          */
/* INPUTS   : n                      - number of nodes                      */
/*          : nMetCod                - topology metric code                 */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_chordlrng
(
    int         n,
    MetricCod   nMetCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( n        );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal)  = n;
```

```
            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = 2.0 * floor( (3.0 * n) / 2.0);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = floor( (3.0 * n) / 2.0);

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            (*pdMetVal)  = 2.0;

            break;
        }
        case MT_NOD_FAC:
        {
            /* node expansion factor */
            (*pdMetVal)  = (n + 2.0) / n;

            break;
        }
        case MT_LNK_INC:
        {
            /* link expansion increment */
            (*pdMetVal)  = (n % 2) + 1.0;

            break;
        }
        case MT_LNK_FAC:
        {
            /* link expansion factor */
            (*pdMetVal)  = 1.0 + ((n % 2) + 1.0) / floor( (3.0 * n) / 2.0);

            break;
        }
        case MT_LNK_CON:
        {
            /* link conectivity */
            (*pdMetVal)  = 3.0;

            break;
        }
        case MT_NOD_CON:
        {
            /* node connectivity */
            (*pdMetVal)  = 3.0;

            break;
        }
        case MT_TOP_DIA:
        {
            /* topology diameter */
            (*pdMetVal)  = ceil( n / 4.0);

            break;
        }
        case MT_VALENCY:
        {
            /* valency */
            (*pdMetVal)  = 3.0;

            break;
        }
        case MT_DV_PROD:
        {
            /* diameter.valency product */
            double  top_dia = 0.0;
            double  valency = 0.0;

            alg_m_cllalgfnc( alg_f_chordlrng( n, MT_TOP_DIA, &top_dia));
            alg_m_cllalgfnc( alg_f_chordlrng( n, MT_VALENCY, &valency));

            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_simpletre(...)                                          */
/*                                                                          */
/* NOTES    : Simple b-ary tree topology analysis function                  */
/*                                                                          */
/* INPUTS   : b                    - node fanout                            */
/*          : l                    - number of levels                       */
/*          : nMetCod              - topology metric code                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
```

```
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_trenodinc(...)                                          */
/*          : alg_f_trenodfac(...)                                          */
/*          : alg_f_trelnkinc(...)                                          */
/*          : alg_f_trelnkfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_simpletre
(
    int         b,
    int         l,
    MetricCod   nMetCod,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( b       );
    alg_m_chkintgtz( l       );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            if ( l == 1)
            {
                /* catch divide-by-zero */
                (*pdMetVal) = 1.0;
            }
            else
            {
                /* default case */
           (*pdMetVal) = (pow( b, l) - 1.0) / (b - 1.0);
            }

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            double  n = 0.0;

            alg_m_cllalgfnc( alg_f_simpletre( b,
                                              l,
                                              MT_NUM_NOD,
                                              nExpCod,
                                              &n));

            (*pdMetVal)  = 2.0 * (n - 1.0);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            double  n = 0.0;

            alg_m_cllalgfnc( alg_f_simpletre( b,
                                              l,
                                              MT_NUM_NOD,
                                              nExpCod,
                                              &n));

            (*pdMetVal)  = n - 1;

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            alg_m_cllalgfnc( alg_f_trenodinc( b, l, nExpCod, pdMetVal));

            break;
        }
        case MT_NOD_FAC:
        {
            /* node expansion factor */
            alg_m_cllalgfnc( alg_f_trenodfac( b, l, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_INC:
        {
            /* link expansion increment */
            alg_m_cllalgfnc( alg_f_trelnkinc( b, l, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_FAC:
        {
            /* link expansion factor */
            alg_m_cllalgfnc( alg_f_trelnkfac( b, l, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_CON:
        {
            /* link conectivity */
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_NOD_CON:
        {
            /* node connectivity */
```

```
            (*pdMetVal)  = 1.0;

            break;
        }
        case MT_TOP_DIA:
        {
            /* topology diameter */
            (*pdMetVal)  = 2.0 * (l - 1.0);

            break;
        }
        case MT_VALENCY:
        {
            /* valency */
            (*pdMetVal)  = b + 1.0;

            break;
        }
        case MT_DV_PROD:
        {
            /* diameter.valency product */
            double  top_dia = 0.0;
            double  valency = 0.0;

            alg_m_cllalgfnc( alg_f_simpletre( b,
                                              l,
                                              MT_TOP_DIA,
                                              nExpCod,
                                             &top_dia));
            alg_m_cllalgfnc( alg_f_simpletre( b,
                                              l,
                                              MT_VALENCY,
                                              nExpCod,
                                             &valency));

            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_fulrngtre(...)                                          */
/*                                                                          */
/* NOTES    : Full-ring or cross-connected tree topology analysis function  */
/*                                                                          */
/* INPUTS   : b                    - node fanout                            */
/*          : l                    - number of levels                       */
/*          : nMetCod              - topology metric code                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_frtnodinc(...)                                          */
/*          : alg_f_frtnodfac(...)                                          */
/*          : alg_f_frtlnkinc(...)                                          */
/*          : alg_f_frtlnkfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_fulrngtre
(
    int         b,
    int         l,
    MetricCod   nMetCod,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( b       );
    alg_m_chkintgtz( l       );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            if ( l == 1)
            {
                /* catch divide-by-zero */
                (*pdMetVal) = 1.0;
            }
            else
            {
                /* default case */
            (*pdMetVal) = (pow( b, l) - 1.0) / (b - 1.0);
            }

            break;
        }
        case MT_NUM_CON:
        {
```

```
        /* number of connections */
        double  n = 0.0;

        alg_m_cllalgfnc( alg_f_fulrngtre( b,
                                          l,
                                          MT_NUM_NOD,
                                          nExpCod,
                                         &n));

        (*pdMetVal)  = 4.0 * (n - 1.0);

        break;
    }
    case MT_NUM_LNK:
    {
        /* number of links */
        double  n = 0.0;

        alg_m_cllalgfnc( alg_f_fulrngtre( b,
                                          l,
                                          MT_NUM_NOD,
                                          nExpCod,
                                         &n));

        (*pdMetVal)  = 2.0 * (n - 1.0);

        break;
    }
    case MT_NOD_INC:
    {
        /* node expansion increment */
        alg_m_cllalgfnc( alg_f_frtnodinc( b, l, nExpCod, pdMetVal));

        break;
    }
    case MT_NOD_FAC:
    {
        /* node expansion factor */
        alg_m_cllalgfnc( alg_f_frtnodfac( b, l, nExpCod, pdMetVal));

        break;
    }
    case MT_LNK_INC:
    {
        /* link expansion increment */
        alg_m_cllalgfnc( alg_f_frtlnkinc( b, l, nExpCod, pdMetVal));

        break;
    }
    case MT_LNK_FAC:
    {
        /* link expansion factor */
        alg_m_cllalgfnc( alg_f_frtlnkfac( b, l, nExpCod, pdMetVal));

        break;
    }
    case MT_LNK_CON:
    {
        /* link conectivity */
        (*pdMetVal)  = 3.0;

        break;
    }
    case MT_NOD_CON:
    {
        /* node connectivity */
        (*pdMetVal)  = 3.0;

        break;
    }
    case MT_TOP_DIA:
    {
        /* topology diameter */
        (*pdMetVal) = max( 2.0 * ( 1 - max( 3.0 - b/2.0, 1.0) ), 1 - 1.0);

        break;
    }
    case MT_VALENCY:
    {
        /* valency */
        (*pdMetVal)  = b + 3.0;

        break;
    }
    case MT_DV_PROD:
    {
        /* diameter.valency product */
        double  top_dia = 0.0;
        double  valency = 0.0;

        alg_m_cllalgfnc( alg_f_fulrngtre( b,
                                          l,
                                          MT_TOP_DIA,
                                          nExpCod,
                                         &top_dia));
        alg_m_cllalgfnc( alg_f_fulrngtre( b,
                                          l,
                                          MT_VALENCY,
                                          nExpCod,
                                         &valency));

        (*pdMetVal)  = top_dia * valency;

        break;
    }
    default:
    {
        utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

        return( ALG_FTL_ERR);
    }
```

```
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nrsnhbmsh(...)                                          */
/*                                                                          */
/* NOTES    : Nearest-neighbour mesh topology analysis function             */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_nnmnodinc(...)                                          */
/*          : alg_f_nnmnodfac(...)                                          */
/*          : alg_f_nnmlnkinc(...)                                          */
/*          : alg_f_nnmlnkfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_nrsnhbmsh
(
    int         w,
    int         d,
    MetricCod   nMetCod,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal) = pow( w, d);

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal) = 2.0 * (w - 1.0) * d * pow( w, d - 1.0);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal) = (w - 1.0) * d * pow( w, d - 1.0);

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            alg_m_cllalgfnc( alg_f_nnmnodinc( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_NOD_FAC:
        {
            /* node expansion factor */
            alg_m_cllalgfnc( alg_f_nnmnodfac( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_INC:
        {
            /* link expansion increment */
            alg_m_cllalgfnc( alg_f_nnmlnkinc( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_FAC:
        {
            /* link expansion factor */
            alg_m_cllalgfnc( alg_f_nnmlnkfac( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_CON:
        {
            /* link conectivity */
            (*pdMetVal) = d;

            break;
        }
        case MT_NOD_CON:
        {
            /* node connectivity */
            (*pdMetVal) = d;

            break;
        }
        case MT_TOP_DIA:
```

```
        {
            /* topology diameter */
            (*pdMetVal) = d * (w - 1.0);

            break;
        }
        case MT_VALENCY:
        {
            /* valency */
            (*pdMetVal)  = 2.0 * d;

            break;
        }
        case MT_DV_PROD:
        {
            /* diameter.valency product */
            double  top_dia = 0.0;
            double  valency = 0.0;

            alg_m_cllalgfnc( alg_f_nrsnhbmsh( w,
                                              d,
                                              MT_TOP_DIA,
                                              nExpCod,
                                             &top_dia));
            alg_m_cllalgfnc( alg_f_nrsnhbmsh( w,
                                              d,
                                              MT_VALENCY,
                                              nExpCod,
                                             &valency));

            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_spabushpc(...)                                          */
/*                                                                          */
/* NOTES    : Spanning-bus hypercube topology analysis function             */
/*                                                                          */
/* INPUTS   : w                    - width of topology                      */
/*          : d                    - number of dimensions                   */
/*          : nMetCod              - topology metric code                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_sbhnodinc(...)                                          */
/*          : alg_f_sbhnodfac(...)                                          */
/*          : alg_f_sbhlnkinc(...)                                          */
/*          : alg_f_sbhlnkfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_spabushpc
(
    int         w,
    int         d,
    MetricCod   nMetCod,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal) = pow( w, d);

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = d * pow( w, d);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = d * pow( w, d - 1.0);

            break;
        }
        case MT_NOD_INC:
        {
```

```
        /* node expansion increment */
        alg_m_cllalgfnc( alg_f_sbhnodinc( w, d, nExpCod, pdMetVal));

        break;
    }
    case MT_NOD_FAC:
    {
        /* node expansion factor */
        alg_m_cllalgfnc( alg_f_sbhnodfac( w, d, nExpCod, pdMetVal));

        break;
    }
    case MT_LNK_INC:
    {
        /* link expansion increment */
        alg_m_cllalgfnc( alg_f_sbhlnkinc( w, d, nExpCod, pdMetVal));

        break;
    }
    case MT_LNK_FAC:
    {
        /* link expansion factor */
        alg_m_cllalgfnc( alg_f_sbhlnkfac( w, d, nExpCod, pdMetVal));

        break;
    }
    case MT_LNK_CON:
    {
        /* link conectivity */
        (*pdMetVal)  = d;

        break;
    }
    case MT_NOD_CON:
    {
        /* node connectivity */
        (*pdMetVal)  = d * ( w - 1.0);

        break;
    }
    case MT_TOP_DIA:
    {
        /* topology diameter */
        (*pdMetVal)  = d;

        break;
    }
    case MT_VALENCY:
    {
        /* valency */
        (*pdMetVal)  = d;

        break;
    }
    case MT_DV_PROD:
    {
        /* diameter.valency product */
        double  top_dia = 0.0;
        double  valency = 0.0;

        alg_m_cllalgfnc( alg_f_spabushpc( w,
                                          d,
                                          MT_TOP_DIA,
                                          nExpCod,
                                         &top_dia));
        alg_m_cllalgfnc( alg_f_spabushpc( w,
                                          d,
                                          MT_VALENCY,
                                          nExpCod,
                                         &valency));

        (*pdMetVal)  = top_dia * valency;

        break;
    }
    default:
    {
        utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

        return( ALG_FTL_ERR);
    }
  }

  return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_duabushpc(...)                                          */
/*                                                                          */
/* NOTES    : Dual-bus hypercube topology analysis function                 */
/*                                                                          */
/* INPUTS   : w                    - width of topology                      */
/*          : d                    - number of dimensions                   */
/*          : nMetCod              - topology metric code                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_dbhnodinc(...)                                          */
/*          : alg_f_dbhnodfac(...)                                          */
/*          : alg_f_dbhlnkinc(...)                                          */
/*          : alg_f_dbhlnkfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */
```

```
AlgErrCod
alg_f_duabushpc
(
    int          w,
    int          d,
    MetricCod    nMetCod,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal) = pow( w, d);

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = 2.0 * pow( w, d);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = 2.0 * pow( w, d - 1.0);

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            alg_m_cllalgfnc( alg_f_dbhnodinc( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_NOD_FAC:
        {
            /* node expansion factor */
            alg_m_cllalgfnc( alg_f_dbhnodfac( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_INC:
        {
            /* link expansion increment */
            alg_m_cllalgfnc( alg_f_dbhlnkinc( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_FAC:
        {
            /* link expansion factor */
            alg_m_cllalgfnc( alg_f_dbhlnkfac( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_CON:
        {
            /* link conectivity */
            (*pdMetVal)  = 2.0;

            break;
        }
        case MT_NOD_CON:
        {
            /* node connectivity */
            (*pdMetVal)  = 2.0 * (w + 1.0);

            break;
        }
        case MT_TOP_DIA:
        {
            /* topology diameter */
            (*pdMetVal)  = 2.0 * (d - 1.0);

            break;
        }
        case MT_VALENCY:
        {
            /* valency */
            (*pdMetVal)  = 2.0;

            break;
        }
        case MT_DV_PROD:
        {
            /* diameter.valency product */
            double  top_dia = 0.0;
            double  valency = 0.0;

            alg_m_cllalgfnc( alg_f_duabushpc( w,
                                              d,
                                              MT_TOP_DIA,
                                              nExpCod,
                                             &top_dia));
            alg_m_cllalgfnc( alg_f_duabushpc( w,
                                              d,
                                              MT_VALENCY,
                                              nExpCod,
                                             &valency));
```

```
            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_torus(...)                                              */
/*                                                                          */
/* NOTES    : Torus topology analysis function                              */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_trsnodinc(...)                                          */
/*          : alg_f_trsnodfac(...)                                          */
/*          : alg_f_trslnkinc(...)                                          */
/*          : alg_f_trslnkfac(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_torus
(
    int         w,
    int         d,
    MetricCod   nMetCod,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal) = pow( w, d);

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = 2.0 * d * pow( w, d);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = d * pow( w, d);

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            alg_m_cllalgfnc( alg_f_trsnodinc( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_NOD_FAC:
        {
            /* node expansion factor */
            alg_m_cllalgfnc( alg_f_trsnodfac( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_INC:
        {
            /* link expansion increment */
            alg_m_cllalgfnc( alg_f_trslnkinc( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_FAC:
        {
            /* link expansion factor */
            alg_m_cllalgfnc( alg_f_trslnkfac( w, d, nExpCod, pdMetVal));

            break;
        }
        case MT_LNK_CON:
        {
            /* link conectivity */
            (*pdMetVal)  = 2.0 * d;
```

```
            break;
        }
        case MT_NOD_CON:
        {
            /* node connectivity */
            (*pdMetVal)  = 2.0 * d;

            break;
        }
        case MT_TOP_DIA:
        {
            /* topology diameter */
            (*pdMetVal)  = d * floor( w / 2.0);

            break;
        }
        case MT_VALENCY:
        {
            /* valency */
            (*pdMetVal)  = 2.0 * d;

            break;
        }
        case MT_DV_PROD:
        {
            /* diameter.valency product */
            double  top_dia = 0.0;
            double  valency = 0.0;

            alg_m_cllalgfnc( alg_f_torus( w, d, MT_TOP_DIA, nExpCod, &top_dia));
            alg_m_cllalgfnc( alg_f_torus( w, d, MT_VALENCY, nExpCod, &valency));

            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rarynube(...)                                           */
/*                                                                          */
/* NOTES    : R-ary N-cube topology analysis function                       */
/*                                                                          */
/* INPUTS   : R                        - node fanout                        */
/*          : N                        - number of dimensions               */
/*          : nMetCod                  - topology metric code               */
/*          : nExpCod                  - expansion parameter code           */
/*                                                                          */
/* OUTPUTS  : pdMetVal                 - metric value                       */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_raryncube
(
    int         R,
    int         N,
    MetricCod   nMetCod,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( R       );
    alg_m_chkintgtz( N       );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal) = N * pow( R, N);

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = 2.0 * N * pow( R, N + 1.0);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = N * pow( R, N + 1.0);

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
```

```
        alg_m_cllalgfnc( alg_f_rncnodinc( R, N, nExpCod, pdMetVal));

        break;
    }
    case MT_NOD_FAC:
    {
        /* node expansion factor */
        alg_m_cllalgfnc( alg_f_rncnodfac( R, N, nExpCod, pdMetVal));

        break;
    }
    case MT_LNK_INC:
    {
        /* link expansion increment */
        alg_m_cllalgfnc( alg_f_rnclnkinc( R, N, nExpCod, pdMetVal));

        break;
    }
    case MT_LNK_FAC:
    {
        /* link expansion factor */
        alg_m_cllalgfnc( alg_f_rnclnkfac( R, N, nExpCod, pdMetVal));

        break;
    }
    case MT_LNK_CON:
    {
        /* link conectivity */
        (*pdMetVal)  = 2.0 * R;

        break;
    }
    case MT_NOD_CON:
    {
        /* node connectivity */
        (*pdMetVal)  = 2.0 * R;

        break;
    }
    case MT_TOP_DIA:
    {
        /* topology diameter */
        (*pdMetVal)  = N + floor( N / 2.0);

        break;
    }
    case MT_VALENCY:
    {
        /* valency */
        (*pdMetVal)  = 2.0 * R;

        break;
    }
    case MT_DV_PROD:
    {
        /* diameter.valency product */
        double  top_dia = 0.0;
        double  valency = 0.0;

        alg_m_cllalgfnc( alg_f_raryncube( R,
                                          N,
                                          MT_TOP_DIA,
                                          nExpCod,
                                         &top_dia));
        alg_m_cllalgfnc( alg_f_raryncube( R,
                                          N,
                                          MT_VALENCY,
                                          nExpCod,
                                         &valency));

        (*pdMetVal)  = top_dia * valency;

        break;
    }
    default:
    {
        utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

        return( ALG_FTL_ERR);
    }
  }

  return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_binaryhpc(...)                                          */
/*                                                                          */
/* NOTES    : Binary hypercube topology analysis function                   */
/*                                                                          */
/* INPUTS   : d                      - number of dimensions                 */
/*          : nMetCod                - topology metric code                 */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : SELF                                                          */
/*          : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

AlgErrCod
alg_f_binaryhpc
(
    int         d,
    MetricCod   nMetCod,
    double     *pdMetVal
)
```

```
{
    /* error checking */
    alg_m_chkintgtz( d        );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal)  = pow( 2.0, d);

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = d * pow( 2.0, d);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = d * pow( 2.0, d - 1.0);

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            (*pdMetVal)  = pow( 2.0, d);

            break;
        }
        case MT_NOD_FAC:
        {
            /* node expansion factor */
            (*pdMetVal)  = 2.0;

            break;
        }
        case MT_LNK_INC:
        {
            /* link expansion increment */
            (*pdMetVal)  = (d + 2.0) * pow( 2.0, d - 1.0);

            break;
        }
        case MT_LNK_FAC:
        {
            /* link expansion factor */
            (*pdMetVal)  = 2.0 * (d + 1.0) / d;

            break;
        }
        case MT_LNK_CON:
        {
            /* link conectivity */
            (*pdMetVal)  = d;

            break;
        }
        case MT_NOD_CON:
        {
            /* node connectivity */
            (*pdMetVal)  = d;

            break;
        }
        case MT_TOP_DIA:
        {
            /* topology diameter */
            (*pdMetVal)  = d;

            break;
        }
        case MT_VALENCY:
        {
            /* valency */
            (*pdMetVal)  = d;

            break;
        }
        case MT_DV_PROD:
        {
            /* diameter.valency product */
            double  top_dia = 0.0;
            double  valency = 0.0;

            alg_m_cllalgfnc( alg_f_binaryhpc( d, MT_TOP_DIA, &top_dia));
            alg_m_cllalgfnc( alg_f_binaryhpc( d, MT_VALENCY, &valency));

            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_cubconcyc(...)                                          */
```

```
/*                                                                        */
/* NOTES     : Cube-connected-cycles topology analysis function          */
/*                                                                        */
/* INPUTS    : d                       - number of dimensions            */
/*           : nMetCod                 - topology metric code            */
/*                                                                        */
/* OUTPUTS   : pdMetVal                - metric value                    */
/*                                                                        */
/* CALLS     : SELF                                                      */
/*           : alg_m_chkintgtz(...)                                      */
/*           : alg_m_chknulptr(...)                                      */
/*           : alg_m_cllalgfnc(...)                                      */
/*           : utl_f_wrierrlog(...)                                      */
/*                                                                        */
/* RETURNS   : AlgErrCod                                                 */
/* ---------------------------------------------------------------------- */

AlgErrCod
alg_f_cubconcyc
(
    int          d,
    MetricCod    nMetCod,
    double      *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( d        );
    alg_m_chknulptr( pdMetVal);

    /* choose topology metric */
    switch ( nMetCod )
    {
        case MT_NUM_NOD:
        {
            /* number of nodes */
            (*pdMetVal) = d * pow( 2.0, d);

            break;
        }
        case MT_NUM_CON:
        {
            /* number of connections */
            (*pdMetVal)  = 3.0 * d * pow( 2.0, d);

            break;
        }
        case MT_NUM_LNK:
        {
            /* number of links */
            (*pdMetVal)  = 3.0 * d * pow( 2.0, d - 1.0);

            break;
        }
        case MT_NOD_INC:
        {
            /* node expansion increment */
            (*pdMetVal)  = (d + 2.0) * pow( 2.0, d);

            break;
        }
        case MT_NOD_FAC:
        {
            /* node expansion factor */
            (*pdMetVal)  = 2.0 * (d + 1.0) / d;

            break;
        }
        case MT_LNK_INC:
        {
            /* link expansion increment */
            (*pdMetVal)  = 3.0 * (d + 2.0) * pow( 2.0, d - 1.0);

            break;
        }
        case MT_LNK_FAC:
        {
            /* link expansion factor */
            (*pdMetVal)  = 2.0 * (d + 1.0) / d;

            break;
        }
        case MT_LNK_CON:
        {
            /* link conectivity */
            (*pdMetVal)  = 3.0;

            break;
        }
        case MT_NOD_CON:
        {
            /* node connectivity */
            (*pdMetVal)  = 3.0;

            break;
        }
        case MT_TOP_DIA:
        {
            /* topology diameter */
            (*pdMetVal)  = 2.0 * d + floor( d / 2.0) - 1.0;

            break;
        }
        case MT_VALENCY:
        {
            /* valency */
            (*pdMetVal)  = 3.0;

            break;
        }
        case MT_DV_PROD:
        {
            /* diameter.valency product */
```

```
            double  top_dia = 0.0;
            double  valency = 0.0;

            alg_m_cllalgfnc( alg_f_cubconcyc( d, MT_TOP_DIA, &top_dia));
            alg_m_cllalgfnc( alg_f_cubconcyc( d, MT_VALENCY, &valency));

            (*pdMetVal)  = top_dia * valency;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown topology metric: %d\n", nMetCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ================================================================== */
/* Private function definitions                                       */
/* ================================================================== */

/* ------------------------------------------------------------------ */
/* FUNCTION : alg_f_trenodinc(...)                                    */
/*                                                                    */
/* NOTES    : Calculate node expansion increment for a simple b-ary tree */
/*                                                                    */
/* INPUTS   : b                      - node fanout                    */
/*          : l                      - number of levels               */
/*          : nExpCod                - expansion parameter code       */
/*                                                                    */
/* OUTPUTS  : pdMetVal               - metric value                   */
/*                                                                    */
/* CALLS    : alg_m_chkintgtz(...)                                    */
/*          : alg_m_chknulptr(...)                                    */
/*          : alg_m_cllalgfnc(...)                                    */
/*          : alg_f_simpletre(...)                                    */
/*          : utl_f_wrierrlog(...)                                    */
/*                                                                    */
/* RETURNS  : AlgErrCod                                               */
/* ------------------------------------------------------------------ */

static
AlgErrCod
alg_f_trenodinc
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    double  n = 0.0;

    /* error checking */
    alg_m_chkintgtz( b       );
    alg_m_chkintgtz( l       );
    alg_m_chknulptr( pdMetVal);

    /* get number of nodes */
    alg_m_cllalgfnc( alg_f_simpletre( b, l, MT_NUM_NOD, EX_FRST, &n));

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing fanout, b */
            (*pdMetVal)  = ( pow( b + 1.0, l) - pow( b, l) - n ) / b;

            break;
        }
        case EX_SCND:
        {
            /* incrementing levels, l */
            (*pdMetVal)  = pow( b, l);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------ */
/* FUNCTION : alg_f_trenodfac(...)                                    */
/*                                                                    */
/* NOTES    : Calculate node expansion factor for a simple b-ary tree */
/*                                                                    */
/* INPUTS   : b                      - node fanout                    */
/*          : l                      - number of levels               */
/*          : nExpCod                - expansion parameter code       */
/*                                                                    */
/* OUTPUTS  : pdMetVal               - metric value                   */
/*                                                                    */
/* CALLS    : alg_m_chkintgtz(...)                                    */
/*          : alg_m_chknulptr(...)                                    */
/*          : alg_m_cllalgfnc(...)                                    */
/*          : alg_f_simpletre(...)                                    */
/*          : utl_f_wrierrlog(...)                                    */
/*                                                                    */
```

```
/* RETURNS  : AlgErrCod                                                    */
/* ----------------------------------------------------------------------- */

static
AlgErrCod
alg_f_trenodfac
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    double  n = 0.0;

    /* error checking */
    alg_m_chkintgtz( b      );
    alg_m_chkintgtz( l      );
    alg_m_chknulptr( pdMetVal);

    /* get number of nodes */
    alg_m_cllalgfnc( alg_f_simpletre( b, l, MT_NUM_NOD, EX_FRST, &n));

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing fanout, b */
            (*pdMetVal)  = ( pow( b + 1.0, l) - 1.0 ) / (n * b);

            break;
        }
        case EX_SCND:
        {
            /* incrementing levels, l */
            (*pdMetVal)  = b + (1.0 / n);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ----------------------------------------------------------------------- */
/* FUNCTION : alg_f_trelnkinc(...)                                         */
/*                                                                         */
/* NOTES    : Calculate link expansion increment for a simple b-ary tree   */
/*            - same as node expansion increments, since l = n - 1.0       */
/*                                                                         */
/* INPUTS   : b                      - node fanout                         */
/*          : l                      - number of levels                    */
/*          : nExpCod                - expansion parameter code            */
/*                                                                         */
/* OUTPUTS  : pdMetVal               - metric value                        */
/*                                                                         */
/* CALLS    : alg_m_chkintgtz(...)                                         */
/*          : alg_m_chknulptr(...)                                         */
/*          : alg_m_cllalgfnc(...)                                         */
/*          : alg_f_trenodinc(...)                                         */
/*                                                                         */
/* RETURNS  : AlgErrCod                                                    */
/* ----------------------------------------------------------------------- */

static
AlgErrCod
alg_f_trelnkinc
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    double  n = 0.0;

    /* error checking */
    alg_m_chkintgtz( b      );
    alg_m_chkintgtz( l      );
    alg_m_chknulptr( pdMetVal);

    /* call node expansion function for same topology */
    alg_m_cllalgfnc( alg_f_trenodinc( b, l, nExpCod, pdMetVal));

    return( ALG_SUCCESS);
}

/* ----------------------------------------------------------------------- */
/* FUNCTION : alg_f_trelnkfac(...)                                         */
/*                                                                         */
/* NOTES    : Calculate link expansion factor for a simple b-ary tree      */
/*                                                                         */
/* INPUTS   : b                      - node fanout                         */
/*          : l                      - number of levels                    */
/*          : nExpCod                - expansion parameter code            */
/*                                                                         */
/* OUTPUTS  : pdMetVal               - metric value                        */
/*                                                                         */
/* CALLS    : alg_m_chkintgtz(...)                                         */
/*          : alg_m_chknulptr(...)                                         */
/*          : alg_m_cllalgfnc(...)                                         */
/*          : alg_f_trenodfac(...)                                         */
/*          : utl_f_wrierrlog(...)                                         */
/*                                                                         */
```

```
/* RETURNS  : AlgErrCod                                                    */
/* ----------------------------------------------------------------------- */

static
AlgErrCod
alg_f_trelnkfac
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    double  n = 0.0;

    /* error checking */
    alg_m_chkintgtz( b       );
    alg_m_chkintgtz( l       );
    alg_m_chknulptr( pdMetVal);

    /* get number of nodes */
    alg_m_cllalgfnc( alg_f_simpletre( b, l, MT_NUM_NOD, EX_FRST, &n));

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing fanout, b */
            if ( n == 1 )
            {
                /* catch divide-by-zero */
                (*pdMetVal)  = 1.0;
            }
            else
            {
                /* default case */
                (*pdMetVal)  = ( pow( b + 1.0, l) - (b + 1.0) ) /
                               ( b * ( n - 1.0) )               ;
            }

            break;
        }
        case EX_SCND:
        {
            /* incrementing levels, l */
            if ( (b == 1) || (l == 1) )
            {
                /* catch divide-by-zero */
                (*pdMetVal)  = 1.0;
            }
            else
            {
                /* default case */
                (*pdMetVal)  = ( pow( b, l) - 1.0 ) / ( pow( b, l - 1.0) - 1.0 );
            }
            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ----------------------------------------------------------------------- */
/* FUNCTION : alg_f_frtnodinc(...)                                         */
/*                                                                         */
/* NOTES    : Calculate node expansion increment for a full-ring b-ary tree */
/*                                                                         */
/* INPUTS   : b                      - node fanout                         */
/*          : l                      - number of levels                    */
/*          : nExpCod                - expansion parameter code            */
/*                                                                         */
/* OUTPUTS  : pdMetVal               - metric value                        */
/*                                                                         */
/* CALLS    : alg_m_chkintgtz(...)                                         */
/*          : alg_m_chknulptr(...)                                         */
/*          : alg_m_cllalgfnc(...)                                         */
/*          : alg_f_fulrngtre(...)                                         */
/*          : utl_f_wrierrlog(...)                                         */
/*                                                                         */
/* RETURNS  : AlgErrCod                                                    */
/* ----------------------------------------------------------------------- */

static
AlgErrCod
alg_f_frtnodinc
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    double  n = 0.0;

    /* error checking */
    alg_m_chkintgtz( b       );
    alg_m_chkintgtz( l       );
    alg_m_chknulptr( pdMetVal);

    /* get number of nodes */
    alg_m_cllalgfnc( alg_f_fulrngtre( b, l, MT_NUM_NOD, EX_FRST, &n));

    /* choose expansion parameter */
```

```
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing fanout, b */
            (*pdMetVal)  = ( pow( b + 1.0, l) - pow( b, l) - n ) / b;

            break;
        }
        case EX_SCND:
        {
            /* incrementing levels, l */
            (*pdMetVal)  = pow( b, l);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_frtnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a full-ring b-ary tree    */
/*                                                                          */
/* INPUTS   : b                      - node fanout                          */
/*          : l                      - number of levels                     */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_fulrngtre(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_frtnodfac
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    double  n = 0.0;

    /* error checking */
    alg_m_chkintgtz( b       );
    alg_m_chkintgtz( l       );
    alg_m_chknulptr( pdMetVal);

    /* get number of nodes */
    alg_m_cllalgfnc( alg_f_fulrngtre( b, l, MT_NUM_NOD, EX_FRST, &n));

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing fanout, b */
            (*pdMetVal)  = ( pow( b + 1.0, l) - 1.0 ) / (n * b);

            break;
        }
        case EX_SCND:
        {
            /* incrementing levels, l */
            (*pdMetVal) = b + (1.0 / n);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_frtlnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a full-ring b-ary tree */
/*              - twice node expansion increments, since l = 2.0 * (n - 1.0) */
/*                                                                          */
/* INPUTS   : b                      - node fanout                          */
/*          : l                      - number of levels                     */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
```

```
/*          : alg_f_fulnodinc(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_frtlnkinc
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    double  n = 0.0;

    /* error checking */
    alg_m_chkintgtz( b       );
    alg_m_chkintgtz( l       );
    alg_m_chknulptr( pdMetVal);

    /* call node expansion function for same topology */
    alg_m_cllalgfnc( alg_f_frtnodinc( b, l, nExpCod, pdMetVal));
    (*pdMetVal)  *= 2.0;

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_frtlnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a full-ring b-ary tree    */
/*                                                                          */
/* INPUTS   : b                      - node fanout                          */
/*          : l                      - number of levels                     */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_fulrngtre(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_frtlnkfac
(
    int         b,
    int         l,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    double  n = 0.0;

    /* error checking */
    alg_m_chkintgtz( b       );
    alg_m_chkintgtz( l       );
    alg_m_chknulptr( pdMetVal);

    /* get number of nodes */
    alg_m_cllalgfnc( alg_f_fulrngtre( b, l, MT_NUM_NOD, EX_FRST, &n));

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing fanout, b */
            if ( n == 1 )
            {
                /* catch divide-by-zero */
                (*pdMetVal)  = 1.0;
            }
            else
            {
                /* default case */
                (*pdMetVal)  = ( pow( b + 1.0, l) - (b + 1.0) ) /
                               ( b * ( n - 1.0) )               ;
            }

            break;
        }
        case EX_SCND:
        {
            /* incrementing levels, l */
            if ( (b == 1) || (l == 1) )
            {
                /* catch divide-by-zero */
                (*pdMetVal)  = 1.0;
            }
            else
            {
                /* default case */
                (*pdMetVal)  = ( pow( b, l) - 1.0 ) / ( pow( b, l - 1.0) - 1.0 );
            }

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
```

```
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nnmnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for a nearest-neighbour mesh */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*            alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*          :                                                               */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_nnmnodinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal) = pow( w + 1.0, d) - pow( w, d);

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal) = (w - 1.0) * pow( w, d);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nnmnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a nearest-neighbour mesh  */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*            alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*          :                                                               */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_nnmnodfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal) = pow( (w + 1.0) / w, d);

            break;
        }
        case EX_SCND:
        {
```

```
            /* incrementing dimension, d */
            (*pdMetVal) = w;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nnmlnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a nearest-neighbour mesh */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : alg_f_trenodinc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_nnmlnkinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = w        * d * pow( w + 1.0, d - 1.0) -
                           (w - 1.0) * d * pow( w      , d - 1.0) ;

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = (w - 1.0) * pow( w, d - 1.0) *
                           ( w * (d + 1.0) - d )        ;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n",
                             nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_nnmlnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a nearest-neighbour mesh  */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : alg_m_cllalgfnc(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_nnmlnkfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
```

```
{
    /* error checking */
    alg_m_chkintgtz( w        );
    alg_m_chkintgtz( d        );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            if ( w == 1 )
            {
                /* catch divide-by-zero */
                (*pdMetVal) = 1.0;
            }
            else
            {
                /* default case */
                (*pdMetVal)  = ( w          * pow( w + 1.0, d - 1.0) ) /
                               ( (w - 1.0) * pow( w      , d - 1.0) ) ;
            }

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = w * (d + 1.0) / d;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n",
                             nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sbhnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for a spanning-bus hypercube */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_sbhnodinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w        );
    alg_m_chkintgtz( d        );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = pow( w + 1.0, d) - pow( w, d);

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = (w - 1.0) * pow( w, d);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sbhnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a spanning-bus hypercube  */
```

```
/*                                                                          */
/* INPUTS    : w                      - width of topology                  */
/*           : d                      - number of dimensions               */
/*           : nExpCod                - expansion parameter code           */
/*                                                                          */
/* OUTPUTS   : pdMetVal               - metric value                       */
/*                                                                          */
/* CALLS     : alg_m_chkintgtz(...)                                         */
/*           : alg_m_chknulptr(...)                                         */
/*           : utl_f_wrierrlog(...)                                         */
/*                                                                          */
/* RETURNS   : AlgErrCod                                                    */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_sbhnodfac
(
    int          w,
    int          d,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = pow( (w + 1.0) / w, d);

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = w;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sbhlnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a spanning-bus hypercube */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_sbhlnkinc
(
    int          w,
    int          d,
    ExpnsnCod    nExpCod,
    double      *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = d * ( pow( w + 1.0, d - 1.0) - pow( w, d - 1.0) );

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = pow( w, d - 1.0) * ( w * (d + 1.0) - d );

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);
```

```
            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_sbhlnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a spanning-bus hypercube  */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_sbhlnkfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = pow( w + 1.0, d - 1.0) / pow( w, d - 1.0);

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = w * (d + 1.0) / d;
```

```
            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_dbhnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for a dual-bus hypercube   */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_dbhnodinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = pow( w + 1.0, d) - pow( w, d);

            break;
```

```
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = (w - 1.0) * pow( w, d);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_dbhnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a dual-bus hypercube      */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_dbhnodfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = pow( (w + 1.0) / w, d);

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = w;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_dbhlnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a dual-bus hypercube   */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_dbhlnkinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
```

```
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = 2.0 * ( pow( w + 1.0, d - 1.0) - pow( w, d - 1.0) );

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = 2.0 * (w - 1.0) * pow( w, d - 1.0);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_dbhlnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a dual-bus hypercube      */
/*                                                                          */
/* INPUTS   : w                    - width of topology                      */
/*          : d                    - number of dimensions                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_dbhlnkfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double      *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = pow( w + 1.0, d - 1.0) / pow( w, d - 1.0);

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = w;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_trsnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for a torus                */
/*                                                                          */
/* INPUTS   : w                    - width of topology                      */
/*          : d                    - number of dimensions                   */
/*          : nExpCod              - expansion parameter code               */
/*                                                                          */
/* OUTPUTS  : pdMetVal             - metric value                           */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */
```

```
static
AlgErrCod
alg_f_trsnodinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = pow( w + 1.0, d) - pow( w, d);

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = (w - 1.0) * pow( w, d);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_trsnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for a torus                   */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
/*          : d                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_trsnodfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = pow( (w + 1.0) / w, d);

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = w;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_trslnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for a torus                */
/*                                                                          */
/* INPUTS   : w                      - width of topology                    */
```

```
/*          : d                     - number of dimensions                  */
/*          : nExpCod               - expansion parameter code              */
/*                                                                          */
/* OUTPUTS  : pdMetVal              - metric value                          */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_trslnkinc
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = d * ( pow( w + 1.0, d) - pow( w, d) );

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = pow( w, d) * ( w * (d + 1.0) - d );

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}
```

```
/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_trslnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for a torus                   */
/*                                                                          */
/* INPUTS   : w                     - width of topology                     */
/*          : d                     - number of dimensions                  */
/*          : nExpCod               - expansion parameter code              */
/*                                                                          */
/* OUTPUTS  : pdMetVal              - metric value                          */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_trslnkfac
(
    int         w,
    int         d,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( w       );
    alg_m_chkintgtz( d       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing width, w */
            (*pdMetVal)  = pow( w + 1.0, d) / pow( w, d);

            break;
        }
        case EX_SCND:
        {
            /* incrementing dimension, d */
            (*pdMetVal)  = w * (d + 1.0) / d;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
```

```
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rncnodinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion increment for an R-ary N-cube        */
/*                                                                          */
/* INPUTS   : R                      - node fanout                          */
/*          : N                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_rncnodinc
(
    int         R,
    int         N,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( R       );
    alg_m_chkintgtz( N       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing R */
            (*pdMetVal)  = N * ( pow(R + 1.0, N) - pow( R, N) );

            break;
        }
        case EX_SCND:
        {
            /* incrementing N */
            (*pdMetVal)  = ( N * (R - 1.0) + R ) * pow( R, N);

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rncnodfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate node expansion factor for an R-ary N-cube           */
/*                                                                          */
/* INPUTS   : R                      - node fanout                          */
/*          : N                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_rncnodfac
(
    int         R,
    int         N,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( R       );
    alg_m_chkintgtz( N       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing R */
            (*pdMetVal)  = pow( (R + 1.0) / R, N);

            break;
        }
        case EX_SCND:
```

```
        {
            /* incrementing N */
            (*pdMetVal)  = R * (N + 1.0) / N;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rnclnkinc(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion increment for an R-ary N-cube        */
/*                                                                          */
/* INPUTS   : R                      - node fanout                          */
/*          : N                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_rnclnkinc
(
    int         R,
    int         N,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( R       );
    alg_m_chkintgtz( N       );
    alg_m_chknulptr( pdMetVal);

    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing R */
            (*pdMetVal)  = N * ( pow( R + 1.0, N + 1.0) - pow( R, N + 1.0) );

            break;
        }
        case EX_SCND:
        {
            /* incrementing N */
            (*pdMetVal)  = pow( R, N + 1.0) * ( R * (N + 1.0) - N );

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n", nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : alg_f_rnclnkfac(...)                                          */
/*                                                                          */
/* NOTES    : Calculate link expansion factor for an R-ary N-cube           */
/*                                                                          */
/* INPUTS   : R                      - node fanout                          */
/*          : N                      - number of dimensions                 */
/*          : nExpCod                - expansion parameter code             */
/*                                                                          */
/* OUTPUTS  : pdMetVal               - metric value                         */
/*                                                                          */
/* CALLS    : alg_m_chkintgtz(...)                                          */
/*          : alg_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : AlgErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
AlgErrCod
alg_f_rnclnkfac
(
    int         R,
    int         N,
    ExpnsnCod   nExpCod,
    double     *pdMetVal
)
{
    /* error checking */
    alg_m_chkintgtz( R       );
    alg_m_chkintgtz( N       );
    alg_m_chknulptr( pdMetVal);
```

```
    /* choose expansion parameter */
    switch ( nExpCod )
    {
        case EX_FRST:
        {
            /* incrementing R */
            (*pdMetVal)  = pow( R + 1.0, N + 1.0) / pow( R, N + 1.0);

            break;
        }
        case EX_SCND:
        {
            /* incrementing N */
            (*pdMetVal)  = R * (N + 1.0) / N;

            break;
        }
        default:
        {
            utl_f_wrierrlog( "** unknown expansion parameter: %d\n",
                             nExpCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( ALG_FTL_ERR);
        }
    }

    return( ALG_SUCCESS);
}

/* ************************************************************************** */
```

C.4.2 tpg_inf.c

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, Univeristy of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : TPG_INF.C                                                      */
/*                                                                            */
/* NOTES     : XLL interfaces for multiprocessor topology analysis functions  */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 18th February, 1995                                            */
/*                                                                            */
/* VERSION   : 1.0                                                            */
/* ************************************************************************** */

/* ========================================================================== */
/* C headers                                                                  */
/* ========================================================================== */

#include    <string.h>
#include    <windows.h>

/* ========================================================================== */
/* SDK headers                                                                */
/* ========================================================================== */

#include    <xlcall.h>
#include    <framewrk.h>

/* ========================================================================== */
/* TPG headers                                                                */
/* ========================================================================== */

#include    "tpg_inf.h"
#include    "tpg_ini.h"
#include    "tpg_alg.h"
#include    "tpg_xlo.h"

/* ========================================================================== */
/* Global definitions                                                         */
/* ========================================================================== */

const
char   *rgFuncs[INF_FNC_RWS][INF_FNC_CLS] =
{
    {" inf_x_fulintcon",
        " RRR",
        " fulc",
        " n, metric",
        " 1",
```

```
        " Topologies",
        " "
    },
    {" inf_x_sharedbus",
        " RRR",
        " bus",
        " n, metric",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_simplerng",
        " RRR",
        " ring",
        " n, metric",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_chordlrng",
        " RRR",
        " c_ring",
        " n, metric",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_simpletre",
        " RRRRR",
        " tree",
        " b, l, metric, [expansion]",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_fulrngtre",
        " RRRRR",
        " x_tree",
        " b, l, metric, [expansion]",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_nrsnhbmsh",
        " RRRRR",
        " mesh",
        " w, d, metric, [expansion]",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_spabushpc",
        " RRRRR",
        " sb_hpc",
        " w, d, metric, [expansion]",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_duabushpc",
        " RRRRR",
        " db_hpc",
        " w, d, metric, [expansion]",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_torus",
        " RRRRR",
        " torus",
        " w, d, metric, [expansion]",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_raryncube",
        " RRRRR",
        " rn_cube",
        " R, N, metric, [expansion]",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_binaryhpc",
        " RRR",
        " b_hpc",
        " d, metric",
        " 1",
        " Topologies",
        " "
    },
    {" inf_x_cubconcyc",
        " RRR",
        " cc_cycles",
        " d, metric",
        " 1",
        " Topologies",
        " "
    },
};

/* ========================================================================== */
/* Public function definitions                                                */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* FUNCTION : inf_x_fulintcon(...)                                            */
/*                                                                            */
/* NOTES    : XLL interface for full-interconnection analysis function        */
/*                                                                            */
/* INPUTS   : pxNumNod              - number of nodes                         */
/*          : pxMetric              - choice of topology metric               */
```

```
/*                                                                      */
/* OUTPUTS   : <none>                                                   */
/*                                                                      */
/* CALLS     : ini_m_xlointgtz(...)                                     */
/*           : ini_m_xlometcod(...)                                     */
/*           : tpg_m_cllalgfnc(...)                                     */
/*           : xlo_m_setxlonum(...)                                     */
/*           : alg_f_fulintcon(...)                                     */
/*                                                                      */
/* RETURNS   : LPXLOPER far pascal                                      */
/* -------------------------------------------------------------------- */

LPXLOPER far pascal
inf_x_fulintcon
(
    LPXLOPER    pxNumNod,
    LPXLOPER    pxMetric
)
{
    static
    XLOPER      xResult;

    int         n       = 0;
    MetricCod   metric  = MT_NUM_NOD;
    double      dMetVal = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard types */
    ini_m_xlointgtz( pxNumNod, n      , XT_RQD);
    ini_m_xlometcod( pxMetric, metric, XT_RQD);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_fulintcon( n, metric, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxNumNod);
    Excel( xlFree, 0, 1, &pxMetric);

    /* return XLOPER */
    return( &xResult);
}

/* -------------------------------------------------------------------- */
/* FUNCTION : inf_x_sharedbus(...)                                      */
/*                                                                      */
/* NOTES    : XLL interface for shared-bus analysis function            */
/*                                                                      */
/* INPUTS   : pxNumNod              - number of nodes                   */
/*          : pxMetric              - choice of topology metric         */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : ini_m_xlointgtz(...)                                      */
/*          : ini_m_xlometcod(...)                                      */
/*          : tpg_m_cllalgfnc(...)                                      */
/*          : xlo_m_setxlonum(...)                                      */
/*          : alg_f_sharedbus(...)                                      */
/*                                                                      */
/* RETURNS  : LPXLOPER far pascal                                       */
/* -------------------------------------------------------------------- */

LPXLOPER far pascal
inf_x_sharedbus
(
    LPXLOPER    pxNumNod,
    LPXLOPER    pxMetric
)
{
    static
    XLOPER      xResult;

    int         n       = 0;
    MetricCod   metric  = MT_NUM_NOD;
    double      dMetVal = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard types */
    ini_m_xlointgtz( pxNumNod, n      , XT_RQD);
    ini_m_xlometcod( pxMetric, metric, XT_RQD);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_sharedbus( n, metric, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxNumNod);
    Excel( xlFree, 0, 1, &pxMetric);

    /* return XLOPER */
    return( &xResult);
}

/* -------------------------------------------------------------------- */
/* FUNCTION : inf_x_simplerng(...)                                      */
/*                                                                      */
/* NOTES    : XLL interface for simple ring analysis function           */
/*                                                                      */
/* INPUTS   : pxNumNod              - number of nodes                   */
/*          : pxMetric              - choice of topology metric         */
/*                                                                      */
/* OUTPUTS  : <none>                                                    */
/*                                                                      */
/* CALLS    : ini_m_xlointgtz(...)                                      */
```

```
/*          : ini_m_xlometcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_simplerng(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_simplerng
(
    LPXLOPER    pxNumNod,
    LPXLOPER    pxMetric
)
{
    static
    XLOPER      xResult;

    int         n       = 0;
    MetricCod   metric  = MT_NUM_NOD;
    double      dMetVal = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxNumNod, n      , XT_RQD);
    ini_m_xlometcod( pxMetric, metric, XT_RQD);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_simplerng( n, metric, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxNumNod);
    Excel( xlFree, 0, 1, &pxMetric);

    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_chordlrng(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for chordal ring analysis function              */
/*                                                                          */
/* INPUTS   : pxNumNod              - number of nodes                       */
/*          : pxMetric              - choice of topology metric             */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_chordlrng(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_chordlrng
(
    LPXLOPER    pxNumNod,
    LPXLOPER    pxMetric
)
{
    static
    XLOPER      xResult;

    int         n       = 0;
    MetricCod   metric  = MT_NUM_NOD;
    double      dMetVal = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxNumNod, n      , XT_RQD);
    ini_m_xlometcod( pxMetric, metric, XT_RQD);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_chordlrng( n, metric, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxNumNod);
    Excel( xlFree, 0, 1, &pxMetric);

    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_simpletre(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for simple b-ary tree analysis function         */
/*                                                                          */
/* INPUTS   : pxFanout              - node fanout                           */
/*          : pxLevels              - number of levels                      */
/*          : pxMetric              - choice of topolgy metric              */
/*          : pxExpnsn              - choice of expansion parameter         */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
```

```
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_simpletre(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_simpletre
(
    LPXLOPER    pxFanout,
    LPXLOPER    pxLevels,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
)
{
    static
    XLOPER      xResult;

    int         b           = 0;
    int         l           = 0;
    MetricCod   metric      = MT_NUM_NOD;
    ExpnsnCod   expansion   = EX_FRST;
    double      dMetVal     = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard types */
    ini_m_xlointgtz( pxFanout, b        , XT_RQD);
    ini_m_xlointgtz( pxLevels, l        , XT_RQD);
    ini_m_xlometcod( pxMetric, metric   , XT_RQD);
    ini_m_xloexpcod( pxExpnsn, expansion, XT_OPT);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_simpletre( b, l, metric, expansion, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxFanout);
    Excel( xlFree, 0, 1, &pxLevels);
    Excel( xlFree, 0, 1, &pxMetric);
    Excel( xlFree, 0, 1, &pxExpnsn);

    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_fulrngtre(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for full-ring or cross-connected b-ary tree     */
/*            analysis function                                             */
/*                                                                          */
/* INPUTS   : pxFanout          - node fanout                               */
/*          : pxLevels          - number of levels                          */
/*          : pxMetric          - choice of topology metric                 */
/*          : pxExpnsn          - choice of expansion parameter             */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_fulrngtre(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_fulrngtre
(
    LPXLOPER    pxFanout,
    LPXLOPER    pxLevels,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
)
{
    static
    XLOPER      xResult;

    int         b           = 0;
    int         l           = 0;
    MetricCod   metric      = MT_NUM_NOD;
    ExpnsnCod   expansion   = EX_FRST;
    double      dMetVal     = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxFanout, b        , XT_RQD);
    ini_m_xlointgtz( pxLevels, l        , XT_RQD);
    ini_m_xlometcod( pxMetric, metric   , XT_RQD);
    ini_m_xloexpcod( pxExpnsn, expansion, XT_OPT);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_fulrngtre( b, l, metric, expansion, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxFanout);
    Excel( xlFree, 0, 1, &pxLevels);
    Excel( xlFree, 0, 1, &pxMetric);
    Excel( xlFree, 0, 1, &pxExpnsn);
```

```
    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_nrsnhbmsh(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for nearest-neighbour mesh analysis function    */
/*                                                                          */
/* INPUTS   : pxWidth              - width of topology                      */
/*          : pxDimnsn             - number of dimensions                   */
/*          : pxMetric             - choice of topology metric              */
/*          : pxExpnsn             - choice of expansion parameter          */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_nrsnhbmsh(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_nrsnhbmsh
(
    LPXLOPER    pxWidth,
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
)
{
    static
    XLOPER      xResult;

    int         w           = 0;
    int         d           = 0;
    MetricCod   metric      = MT_NUM_NOD;
    ExpnsnCod   expansion   = EX_FRST;
    double      dMetVal     = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxWidth , w        , XT_RQD);
    ini_m_xlointgtz( pxDimnsn, d        , XT_RQD);
    ini_m_xlometcod( pxMetric, metric   , XT_RQD);
    ini_m_xloexpcod( pxExpnsn, expansion, XT_OPT);

    /* call topology analysis algorithm */

    tpg_m_cllalgfnc( alg_f_nrsnhbmsh( w,
                                      d,
                                      metric,
                                      expansion,
                                      &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxWidth );
    Excel( xlFree, 0, 1, &pxDimnsn);
    Excel( xlFree, 0, 1, &pxMetric);
    Excel( xlFree, 0, 1, &pxExpnsn);

    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_spabushpc(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for spanning-bus hypercube analysis function    */
/*                                                                          */
/* INPUTS   : pxWidth              - width of topology                      */
/*          : pxDimnsn             - number of dimensions                   */
/*          : pxMetric             - choice of topology metric              */
/*          : pxExpnsn             - choice of expansion parameter          */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_spabushpc(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_spabushpc
(
    LPXLOPER    pxWidth,
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
)
{
    static
    XLOPER      xResult;

    int         w           = 0;
    int         d           = 0;
    MetricCod   metric      = MT_NUM_NOD;
    ExpnsnCod   expansion   = EX_FRST;
```

```
    double      dMetVal     = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxWidth , w        , XT_RQD);
    ini_m_xlointgtz( pxDimnsn, d        , XT_RQD);
    ini_m_xlometcod( pxMetric, metric   , XT_RQD);
    ini_m_xloexpcod( pxExpnsn, expansion, XT_OPT);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_spabushpc( w, d, metric, expansion, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxWidth );
    Excel( xlFree, 0, 1, &pxDimnsn);
    Excel( xlFree, 0, 1, &pxMetric);
    Excel( xlFree, 0, 1, &pxExpnsn);

    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_duabushpc(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for dual-bus hypercube analysis function        */
/*                                                                          */
/* INPUTS   : pxWidth              - width of topology                      */
/*          : pxDimnsn             - number of dimensions                   */
/*          : pxMetric             - choice of topology metric              */
/*          : pxExpnsn             - choice of expansion parameter          */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_duabushpc(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_duabushpc
(
    LPXLOPER    pxWidth,
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
)
{
    static
    XLOPER      xResult;

    int         w           = 0;
    int         d           = 0;
    MetricCod   metric      = MT_NUM_NOD;
    ExpnsnCod   expansion   = EX_FRST;
    double      dMetVal     = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxWidth,  w        , XT_RQD);
    ini_m_xlointgtz( pxDimnsn, d        , XT_RQD);
    ini_m_xlometcod( pxMetric, metric   , XT_RQD);
    ini_m_xloexpcod( pxExpnsn, expansion, XT_OPT);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_duabushpc( w, d, metric, expansion, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxWidth );
    Excel( xlFree, 0, 1, &pxDimnsn);
    Excel( xlFree, 0, 1, &pxMetric);
    Excel( xlFree, 0, 1, &pxExpnsn);

    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_torus(...)                                              */
/*                                                                          */
/* NOTES    : XLL interface for torus analysis function                     */
/*                                                                          */
/* INPUTS   : pxWidth              - width of topology                      */
/*          : pxDimnsn             - number of dimensions                   */
/*          : pxMetric             - choice of topology metric              */
/*          : pxExpnsn             - choice of expansion parameter          */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : ini_m_xloexpcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_torus(...)                                              */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
```

```
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_torus
(
    LPXLOPER    pxWidth,
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
)
{
    static
    XLOPER      xResult;

    int         w           = 0;
    int         d           = 0;
    MetricCod   metric      = MT_NUM_NOD;
    ExpnsnCod   expansion   = EX_FRST;
    double      dMetVal     = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxWidth,  w        , XT_RQD);
    ini_m_xlointgtz( pxDimnsn, d        , XT_RQD);
    ini_m_xlometcod( pxMetric, metric   , XT_RQD);
    ini_m_xloexpcod( pxExpnsn, expansion, XT_OPT);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_torus( w, d, metric, expansion, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxWidth );
    Excel( xlFree, 0, 1, &pxDimnsn);
    Excel( xlFree, 0, 1, &pxMetric);
    Excel( xlFree, 0, 1, &pxExpnsn);

    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_raryncube(...)                                          */
/*                                                                          */
/* NOTES    : XLL interface for R-ary N-cube analysis function              */
/*                                                                          */
/* INPUTS   : pxFanout              - node fanout                           */
/*          : pxLevels              - number of levels                      */
/*          : pxMetric              - choice of topology metric             */
/*          : pxExpnsn              - choice of expansion parameter         */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : ini_m_xlointgtz(...)                                          */
/*          : ini_m_xlometcod(...)                                          */
/*          : tpg_m_cllalgfnc(...)                                          */
/*          : xlo_m_setxlonum(...)                                          */
/*          : alg_f_raryncube(...)                                          */
/*                                                                          */
/* RETURNS  : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------ */

LPXLOPER far pascal
inf_x_raryncube
(
    LPXLOPER    pxFanout,
    LPXLOPER    pxLevels,
    LPXLOPER    pxMetric,
    LPXLOPER    pxExpnsn
)
{
    static
    XLOPER      xResult;

    int         R           = 0;
    int         N           = 0;
    MetricCod   metric      = MT_NUM_NOD;
    ExpnsnCod   expansion   = EX_FRST;
    double      dMetVal     = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxFanout, R        , XT_RQD);
    ini_m_xlointgtz( pxLevels, N        , XT_RQD);
    ini_m_xlometcod( pxMetric, metric   , XT_RQD);
    ini_m_xloexpcod( pxExpnsn, expansion, XT_OPT);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_raryncube( R, N, metric, expansion, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxFanout);
    Excel( xlFree, 0, 1, &pxLevels);
    Excel( xlFree, 0, 1, &pxMetric);
    Excel( xlFree, 0, 1, &pxExpnsn);

    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : inf_x_binaryhpc(...)                                          */
/*                                                                          */
```

```
/* NOTES     : XLL interface for binary hypercube analysis function          */
/*                                                                           */
/* INPUTS    : pxDimnsn              - number of dimensions                  */
/*           : pxMetric              - choice of topology metric             */
/*                                                                           */
/* OUTPUTS   : <none>                                                        */
/*                                                                           */
/* CALLS     : ini_m_xlointgtz(...)                                          */
/*           : ini_m_xlometcod(...)                                          */
/*           : tpg_m_cllalgfnc(...)                                          */
/*           : xlo_m_setxlonum(...)                                          */
/*           : alg_f_binaryhpc(...)                                          */
/*                                                                           */
/* RETURNS   : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------- */

LPXLOPER far pascal
inf_x_binaryhpc
(
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric
)
{
    static
    XLOPER      xResult;

    int         d       = 0;
    MetricCod   metric  = MT_NUM_NOD;
    double      dMetVal = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxDimnsn, d     , XT_RQD);
    ini_m_xlometcod( pxMetric, metric, XT_RQD);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_binaryhpc( d, metric, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxDimnsn);
    Excel( xlFree, 0, 1, &pxMetric);

    /* return XLOPER */
    return( &xResult);
}

/* ------------------------------------------------------------------------- */
/* FUNCTION : inf_x_cubconcyc(...)                                           */
/*                                                                           */
/* NOTES     : XLL interface for cube-connected-cycles analysis function     */
/*                                                                           */
/* INPUTS    : pxDimnsn              - number of dimensions                  */
/*           : pxMetric              - choice of topology metric             */
/*                                                                           */
/* OUTPUTS   : <none>                                                        */
/*                                                                           */
/* CALLS     : ini_m_xlointgtz(...)                                          */
/*           : ini_m_xlometcod(...)                                          */
/*           : tpg_m_cllalgfnc(...)                                          */
/*           : xlo_m_setxlonum(...)                                          */
/*           : alg_f_cubconcyc(...)                                          */
/*                                                                           */
/* RETURNS   : LPXLOPER far pascal                                           */
/* ------------------------------------------------------------------------- */

LPXLOPER far pascal
inf_x_cubconcyc
(
    LPXLOPER    pxDimnsn,
    LPXLOPER    pxMetric
)
{
    static
    XLOPER      xResult;

    int         d       = 0;
    MetricCod   metric  = MT_NUM_NOD;
    double      dMetVal = 0.0;

    /* clear return XLOPER */
    memset( &xResult, 0, sizeof( xResult));

    /* test input XLOPERs and init standard variables */
    ini_m_xlointgtz( pxDimnsn, d     , XT_RQD);
    ini_m_xlometcod( pxMetric, metric, XT_RQD);

    /* call topology analysis algorithm */
    tpg_m_cllalgfnc( alg_f_cubconcyc( d, metric, &dMetVal));

    /* set return XLOPER */
    xlo_m_setxlonum( dMetVal, &xResult);

    /* free all XLOPERs passed in by Excel */
    Excel( xlFree, 0, 1, &pxDimnsn);
    Excel( xlFree, 0, 1, &pxMetric);

    /* return XLOPER */
    return( &xResult);
}

/* ************************************************************************* */
```

C.4.3 tpg_ini.c

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, Univeristy of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : TPG_INI.C                                                      */
/*                                                                            */
/* NOTES     : Initialisation routines to convert between XLOPERs and C types */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 15th February, 1995                                            */
/*                                                                            */
/* VERSION   : 1.0                                                            */
/* ************************************************************************** */

/* ========================================================================== */
/* C headers                                                                  */
/* ========================================================================== */

#include    <string.h>
#include    <windows.h>

/* ========================================================================== */
/* SDK headers                                                                */
/* ========================================================================== */

#include    <xlcall.h>
#include    <framewrk.h>

/* ========================================================================== */
/* TPG headers                                                                */
/* ========================================================================== */

#include    "tpg_inf.h"
#include    "tpg_ini.h"
#include    "tpg_utl.h"
#include    "tpg_xlo.h"

/* ========================================================================== */
/* Global definitions                                                         */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* GLOBAL    : metric code strings                                            */
/* -------------------------------------------------------------------------- */

const
char    *rgMetCodAry[ (MT_DV_PROD + 1) * INI_MET_NUM] =
{   "nodes",                "n" ,   "1",
    "connections",          "c" ,   "2",
    "links",                "l" ,   "3",
    "node increment",       "ni",   "4",
    "node factor",          "nf",   "5",
    "link increment",       "li",   "6",
    "link factor",          "lf",   "7",
    "link connectivity",    "lc",   "8",
    "node connectivity",    "nc",   "9",
    "topology diameter",    "d" ,  "10",
    "valency",              "v" ,  "11",
    "dv product",           "dv",  "12"
};

/* -------------------------------------------------------------------------- */
/* GLOBAL    : expansion parameter code strings                               */
/* -------------------------------------------------------------------------- */

const
char    *rgExpCodAry[ (EX_SCND + 1) * INI_EXP_NUM] =
{
    "fanout",   "width",        "b",    "w",    "r",    "1",
    "levels",   "dimension",    "l",    "d",    "n",    "2"
};

/* ========================================================================== */
/* Public functions                                                           */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* FUNCTION : ini_f_xlointgtz(...)                                            */
/*                                                                            */
/* NOTES    : Initialise greater-than-zero integer from XLOPER                */
/*                                                                            */
/* INPUTS   : pxInput              - input XLOPER passed from Excel           */
/*          : nArgTyp              - argument type                            */
/*          : szInput              - stringized input C-type                  */
/*                                                                            */
/* OUTPUTS  : pInteger             - integer initialised from XLOPER          */
/*          : pxResult             - error XLOPER passed back to Excel        */
/*                                                                            */
/* CALLS    : ini_m_chkxlomis(...)                                            */
/*          : ini_m_cllinifnc(...)                                            */
/*          : xlo_m_getxloint(...)                                            */
/*          : Excel(...)                                                      */
/*          : ini_f_chkintgtz(...)                                            */
/*          : ini_f_chkxloerr(...)                                            */
/*                                                                            */
/* RETURNS  : IniErrCod                                                       */
/* -------------------------------------------------------------------------- */

IniErrCod
ini_f_xlointgtz
(
    const   LPXLOPER    pxInput,
            XloTypCod   nArgTyp,
    const   char       *szInput,
            int        *pInteger,
```

```
            LPXLOPER    pxResult
)
{
    static
    XLOPER      xInteger;

    /* clear static XLOPER */
    memset( &xInteger, 0, sizeof( xInteger));

    /* check XLOPER is not missing */
    ini_m_chkxlomis( pxInput, nArgTyp, szInput, pxResult);

    /* coerce XLOPER to an integer */
    Excel( xlCoerce, &xInteger, 2, pxInput, TempInt( xltypeInt));

    /* check XLOPER is not an Excel #error */
    ini_m_cllinifnc( ini_f_chkxloerr( &xInteger, szInput, pxResult));

    /* get integer value from coerced XLOPER */
    (*pInteger) = xlo_m_getxloint( &xInteger);

    /* convert integer value */
    ini_m_cllinifnc( ini_f_chkintgtz( (*pInteger), szInput, pxResult));

    return( INI_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_xlometcod(...)                                          */
/*                                                                          */
/* NOTES    : Initialise metric code from XLOPER                            */
/*                                                                          */
/* INPUTS   : pxInput              - input XLOPER passed from Excel         */
/*          : nArgTyp              - argument type                          */
/*          : szInput              - stringized input C-type                */
/*                                                                          */
/* OUTPUTS  : pMetCod              - metric code initialised from XLOPER    */
/*          : pxResult             - error XLOPER passed back to Excel      */
/*                                                                          */
/* CALLS    : ini_m_chkxlomis(...)                                          */
/*          : ini_m_cllinifnc(...)                                          */
/*          : xlo_m_getxlostr(...)                                          */
/*          : Excel(...)                                                    */
/*          : ini_f_chkxloerr(...)                                          */
/*          : ini_f_cnvmetcod(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

IniErrCod
ini_f_xlometcod
(
    const   LPXLOPER    pxInput,
            XloTypCod   nArgTyp,
    const   char       *szInput,
            ExpnsnCod  *pMetCod,
            LPXLOPER    pxResult
)
{
    static
    XLOPER  xString;

    char    szMetCod[XLO_STR_MAX]   = "";

    /* clear static XLOPER */
    memset( &xString, 0, sizeof( xString));

    /* check XLOPER is not missing */
    ini_m_chkxlomis( pxInput, nArgTyp, szInput, pxResult);

    /* coerce XLOPER to a z-string */
    Excel( xlCoerce, &xString, 2, pxInput, TempInt( xltypeStr));

    /* check XLOPER is not an Excel #error */
    ini_m_cllinifnc( ini_f_chkxloerr( &xString, szInput, pxResult));

    /* get z-string from coerced XLOPER */
    xlo_f_getxlostr( &xString, szMetCod);

    /* convert metric code */
    ini_m_cllinifnc( ini_f_cnvmetcod( szMetCod, szInput, pMetCod, pxResult));

    return( INI_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_xloexpcod(...)                                          */
/*                                                                          */
/* NOTES    : Initialise expansion parameter code from XLOPER               */
/*                                                                          */
/* INPUTS   : pxInput              - input XLOPER passed from Excel         */
/*          : nArgTyp              - argument type                          */
/*          : szInput              - stringized input C-type                */
/*                                                                          */
/* OUTPUTS  : pExpCod              - expansion parameter code initialised   */
/*                                   from XLOPER                            */
/*          : pxResult             - error XLOPER passed back to Excel      */
/*                                                                          */
/* CALLS    : ini_m_chkxlomis(...)                                          */
/*          : ini_m_cllinifnc(...)                                          */
/*          : xlo_m_getxlostr(...)                                          */
/*          : Excel(...)                                                    */
/*          : ini_f_chkxloerr(...)                                          */
/*          : ini_f_cnvexpcod(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

IniErrCod
ini_f_xloexpcod
(
    const   LPXLOPER    pxInput,
```

```
            XloTypCod   nArgTyp,
    const   char       *szInput,
            ExpnsnCod  *pExpCod,
            LPXLOPER    pxResult
)
{
    static
    XLOPER  xString;

    char    szExpCod[XLO_STR_MAX]   = "";

    /* clear static XLOPER */
    memset( &xString, 0, sizeof( xString));

    /* check XLOPER is not missing */
    ini_m_chkxlomis( pxInput, nArgTyp, szInput, pxResult);

    /* coerce XLOPER to a string */
    Excel( xlCoerce, &xString, 2, pxInput, TempInt( xltypeStr));

    /* check XLOPER is not an Excel #error */
    ini_m_cllinifnc( ini_f_chkxloerr( &xString, szInput, pxResult));

    /* get z-string from coerced XLOPER */
    xlo_f_getxlostr( &xString, szExpCod);

    /* convert expansion code */
    ini_m_cllinifnc( ini_f_cnvexpcod( szExpCod, szInput, pExpCod, pxResult));

    return( INI_SUCCESS);
}

/* ================================================================== */
/* Private functions                                                  */
/* ================================================================== */

/* ------------------------------------------------------------------ */
/* FUNCTION : ini_f_matstrcod(...)                                    */
/*                                                                    */
/* NOTES    : Match string against known strings for a given code     */
/*                                                                    */
/* INPUTS   : szString              - z-string                        */
/*          : nCode                 - code value                      */
/*          : aStrAry               - array of input strings          */
/*          : nNumStr               - number of strings for each code */
/*                                                                    */
/* OUTPUTS  : <none>                                                  */
/*                                                                    */
/* CALLS    : <none>                                                  */
/*                                                                    */
/* RETURNS  : IniErrCod                                               */
/* ------------------------------------------------------------------ */

static
IniErrCod
ini_f_matstrcod
(
    const   char        *szString,
            int          nCode,
    const   char        *aStrAry[],
            int          nNumStr
)
{
    int     i       = 0;
    long    nBase   = 0;

    /* calculate base address  */
    nBase = nCode * nNumStr;

    /* step through string codes for this metric */
    for ( i = 0; i < nNumStr; i++ )
    {
        /* protect strcmp from access-violation */
        if ( aStrAry[nBase + i] != NULL)
        {
            if ( !strcmp( szString, aStrAry[nBase + i]) )
            {
                /* match found */
                return( INI_SUCCESS);
            }
        }
    }

    /* no match found */
    return( INI_FTL_ERR);
}

/* ------------------------------------------------------------------ */
/* FUNCTION : ini_f_chkxlomis(...)                                    */
/*                                                                    */
/* NOTES    : Check for missing argument passed in by Excel           */
/*                                                                    */
/* INPUTS   : pxInput               - input XLOPER passed from Excel  */
/*          : nArgTyp               - argument type                   */
/*          : szInput               - stringized input C-type         */
/*                                                                    */
/* OUTPUTS  : pxResult              - error XLOPER passed back to Excel */
/*                                                                    */
/* CALLS    : ini_m_chknulptr(...)                                    */
/*          : xlo_m_getxlotyp(...)                                    */
/*          : utl_f_wrierrlog(...)                                    */
/*          : xlo_f_setxlostr(...)                                    */
/*                                                                    */
/* RETURNS  : IniErrCod                                               */
/* ------------------------------------------------------------------ */

static
IniErrCod
ini_f_chkxlomis
(
    const   LPXLOPER    pxInput,
            XloTypCod   nArgTyp,
```

```
    const   char        *szInput,
            LPXLOPER     pxResult
)
{
    /* check for NULL ptrs */
    ini_m_chknulptr( pxInput );
    ini_m_chknulptr( szInput );
    ini_m_chknulptr( pxResult);

    /* check if argument is missing */
    if ( xlo_m_getxlotyp( pxInput) == xltypeMissing )
    {
        switch ( nArgTyp )
        {
            case XT_OPT:
            {
                /* missing optional argument */
                return( INI_MIS_OPT);
            }
            case XT_RQD:
            {
                /* missing required argument */
                utl_f_wrierrlog( "** required parameter is missing: %s\n",
                                 szInput);
                utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

                xlo_f_setxlostr( pxResult, "<%s> is missing", szInput);

                return( INI_MIS_RQD);
            }
            default:
            {
                /* unknown argument type */
                utl_f_wrierrlog( "** unknown argument type: %s = %d\n",
                                 szInput,
                                 nArgTyp);
                utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

                xlo_f_setxlostr( pxResult,
                                 "<%s> unknown argument type",
                                 szInput);

                return( INI_FTL_ERR);
            }
        }
    }

    /* argument is not missing */
    return( INI_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_chkxloerr(...)                                          */
/*                                                                          */
/* NOTES    : Check if XLOPER is an Excel error code                        */
/*                                                                          */
/* INPUTS   : pxInput              - input XLOPER passed from Excel         */
/*          : szInput              - stringized input C-type                */
/*                                                                          */
/* OUTPUTS  : pxResult             - error XLOPER passed back to Excel      */
/*                                                                          */
/* CALLS    : ini_m_chknulptr(...)                                          */
/*          : xlo_m_getxlotyp(...)                                          */
/*          : ini_f_cnverrcod(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*          : xlo_f_setxlostr(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
IniErrCod
ini_f_chkxloerr
(
    const   LPXLOPER    pxInput,
    const   char       *szInput,
            LPXLOPER    pxResult
)
{
    char    szErrStr[XLO_STR_MAX]   = "";

    /* check for NULL pts */
    ini_m_chknulptr( pxInput );
    ini_m_chknulptr( szInput );
    ini_m_chknulptr( pxResult);

    /* if parameter is an Excel #error code */
    if ( xlo_m_getxlotyp( pxInput) == xltypeErr )
    {
        if ( ini_f_cnverrcod( pxInput, szErrStr) == INI_SUCCESS )
        {
            utl_f_wrierrlog( "** Excel error code: %s = %s\n",
                             szInput,
                             szErrStr);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

         xlo_f_setxlostr( pxResult, "<%s> = %s", szInput, szErrStr);
        }
        else
        {
            utl_f_wrierrlog( "** unknown Excel error code: %s\n", szInput);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            xlo_f_setxlostr( pxResult, "<%s> is unknown", szInput);
        }

        return( INI_FTL_ERR);
    }

    return( INI_SUCCESS);
}
```

```
/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_chkintgtz(...)                                          */
/*                                                                          */
/* NOTES    : Check integer is greater than zero                            */
/*                                                                          */
/* INPUTS   : nInput                - integer value from input XLOPER       */
/*          : szInput               - stringized input C-type               */
/*                                                                          */
/* OUTPUTS  : pxResult              - error XLOPER passed back to Excel     */
/*                                                                          */
/* CALLS    : ini_m_chknulptr(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*          : xlo_f_setxlostr(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
IniErrCod
ini_f_chkintgtz
(
            int         nInput,
    const   char       *szInput,
            LPXLOPER    pxResult
)
{
    /* check for NULL ptrs */
    ini_m_chknulptr( szInput );
    ini_m_chknulptr( pxResult);

    /* check integer is greater than zero */
    if ( nInput < 1 )
    {
        utl_f_wrierrlog( "** integer is less than 1: %s = %d\n",
                         szInput,
                         nInput);
        utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

        xlo_f_setxlostr( pxResult,
                         "<%s> (= %d) is less than 1",
                         szInput,
                         nInput);

        return( INI_FTL_ERR);
    }

    return( INI_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_cnverrcod(...)                                          */
/*                                                                          */
/* NOTES    : Convert XLOPER to an Excel error code                         */
/*                                                                          */
/* INPUTS   : pxInput               - input XLOPER passed from Excel        */
/*                                                                          */
/* OUTPUTS  : szErrStr              - Excel error string                    */
/*                                                                          */
/* CALLS    : xlo_m_chknulptr(...)                                          */
/*          : xlo_m_getxlotyp(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
IniErrCod
ini_f_cnverrcod
(
    const   LPXLOPER     pxInput,
            char        *szErrStr
)
{
    unsigned int    nXleCod = 0;

    /* check for NULL ptrs */
    ini_m_chknulptr( pxInput );
    ini_m_chknulptr( szErrStr);

    /* get type of XLOPER */
    nXleCod = xlo_m_getxlotyp( pxInput);

    switch( nXleCod )
    {
        case xlerrNull:
        {
            strcpy( szErrStr, "#NULL!");
            break;
        }
        case xlerrDiv0:
        {
            strcpy( szErrStr, "#DIV/0!");
            break;
        }
        case xlerrValue:
        {
            strcpy( szErrStr, "#VALUE!");
            break;
        }
        case xlerrRef:
        {
            strcpy( szErrStr, "#REF!");
            break;
        }
        case xlerrName:
        {
            strcpy( szErrStr, "#NAME?");
            break;
        }
        case xlerrNum:
        {
            strcpy( szErrStr, "#NUM!");
```

```
            break;
        }
        case xlerrNA:
        {
            strcpy( szErrStr, "#N/A");
            break;
        }
        default:
        {
            /* XLOPER type is an unknown error code */
            utl_f_wrierrlog( "** unknown Excel error code: %u\n", nXleCod);
            utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

            return( INI_FTL_ERR);
        }
    }

    return( INI_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_cnvmetcod(...)                                          */
/*                                                                          */
/* NOTES    : Convert z-string to a metric code                             */
/*                                                                          */
/* INPUTS   : szString              - string from input XLOPER              */
/*          : szInput               - stringized input C-type               */
/*                                                                          */
/* OUTPUTS  : pnMetCod              - metric code                           */
/*          : pxResult              - error XLOPER passed back to Excel     */
/*                                                                          */
/* CALLS    : ini_m_chknulptr(...)                                          */
/*          : ini_f_matstrcod(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*          : xlo_f_setxlostr(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
IniErrCod
ini_f_cnvmetcod
(
    const   char        *szString,
    const   char        *szInput,
            MetricCod   *pnMetCod,
            LPXLOPER     pxResult
)
{
    int nMetIdx = MT_NUM_NOD;

    /* check for NULL ptrs */
    ini_m_chknulptr( szString);
    ini_m_chknulptr( szInput);
    ini_m_chknulptr( pnMetCod);

    /* compare metric parameter with known strings */
    for ( nMetIdx = MT_NUM_NOD; nMetIdx <= MT_DV_PROD; nMetIdx++ )
    {
        if ( ini_f_matstrcod( szString,
                              nMetIdx,
                              rgMetCodAry,
                              INI_MET_NUM) == INI_SUCCESS )
        {
            (*pnMetCod)  = nMetIdx;

            return( INI_SUCCESS);
        }
    }

    /* no match found */
    utl_f_wrierrlog( "** unknown topology metric: %s = %s\n",
                     szInput,
                     szString);
    utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

    xlo_f_setxlostr( pxResult,
                     "<%s> (= %s) is an unknown topology metric",
                     szInput,
                     szString);

    return( INI_FTL_ERR);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : ini_f_cnvexpcod(...)                                          */
/*                                                                          */
/* NOTES    : Convert z-string to an expansion parameter code               */
/*                                                                          */
/* INPUTS   : szString              - string from input XLOPER              */
/*          : szInput               - stringized input C-type               */
/*                                                                          */
/* OUTPUTS  : pnExpCod              - expansion parameter code              */
/*          : pxResult              - error XLOPER passed back to Excel     */
/*                                                                          */
/* CALLS    : ini_m_chknulptr(...)                                          */
/*          : ini_f_matstrcod(...)                                          */
/*          : utl_f_wrierrlog(...)                                          */
/*          : xlo_f_setxlostr(...)                                          */
/*                                                                          */
/* RETURNS  : IniErrCod                                                     */
/* ------------------------------------------------------------------------ */

static
IniErrCod
ini_f_cnvexpcod
(
    const   char        *szString,
    const   char        *szInput,
            ExpnsnCod   *pnExpCod,
            LPXLOPER     pxResult
)
```

```
{
    int nExpIdx = EX_FRST;

    /* check for NULL ptrs */
    ini_m_chknulptr( szString);
    ini_m_chknulptr( szInput);
    ini_m_chknulptr( pnExpCod);

    /* compare expansion increment with known strings */
    for ( nExpIdx = EX_FRST; nExpIdx <= EX_SCND; nExpIdx++ )
    {
        if ( ini_f_matstrcod( szString,
                              nExpIdx,
                              rgExpCodAry,
                              INI_EXP_NUM) == INI_SUCCESS )
        {
            (*pnExpCod)  = nExpIdx;

            return( INI_SUCCESS);
        }
    }

    /* no match found */
    utl_f_wrierrlog( "** unknown expansion parameter: %s = %s\n",
                     szInput;
                     szString);
    utl_f_wrierrlog( "   line %d in %s\n", __LINE__, __FILE__);

    xlo_f_setxlostr( pxResult,
                     "<%s> (= %s) is an unknown expansion parameter",
                     szInput,
                     szString);

    return( INI_FTL_ERR);
}

/* ************************************************************************** */
```

C.4.4 tpg_utl.c

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : TPG_UTL.C                                                      */
/*                                                                            */
/* NOTES     : Utility functions for topology analysis XLL                    */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 10th February, 1995                                            */
/*                                                                            */
/* VERSION   : 1.0                                                            */
/* ************************************************************************** */

/* ========================================================================== */
/* C headers                                                                  */
/* ========================================================================== */

#include    <stdio.h>
#include    <string.h>
#include    <stdarg.h>
#include    <stdlib.h>

/* ========================================================================== */
/* TPG headers                                                                */
/* ========================================================================== */

#include    "tpg_utl.h"

/* ========================================================================== */
/* Public function definitions                                                */
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* FUNCTION : utl_f_creerrlog(...)                                            */
/*                                                                            */
/* NOTES    : Create error log file and backup previous version, if it exists */
/*                                                                            */
/* INPUTS   : szLogFil              - error log filename, including full path */
/*                                                                            */
/* OUTPUTS  : <none>                                                          */
/*                                                                            */
/* CALLS    : utl_m_chknulptr(...)                                            */
/*                                                                            */
/* RETURNS  : UtlErrCod                                                       */
/* -------------------------------------------------------------------------- */

UtlErrCod
utl_f_creerrlog
```

```
(
    const   char    *szLogFil
)
{
    char    *pszOldFil    = NULL;
    char    *szFilExt     = NULL;
    FILE    *pfErrLog     = NULL;

    /* check for NULL ptrs */
    utl_m_chknulptr( szLogFil);

    /* allocate memory for log filename, plus '%' */
    pszOldFil = (char *) calloc( strlen( szLogFil) + 2, sizeof( char));

    /* initialise ptr to start of filename extension */
    szFilExt = strchr( szLogFil, '.' );

    if ( szFilExt == NULL )
    {
        szFilExt = (char *) szLogFil + strlen( szLogFil);
    }

    /* copy log filename, replacing last character before '.' with '%' */
    strncpy( pszOldFil, szLogFil, (szFilExt - 1) - szLogFil);
    strcat( pszOldFil, "%");
    strncat( pszOldFil, szFilExt, strlen( szFilExt));

    /* rename previous log file */
    remove( pszOldFil);
    rename( szLogFil, pszOldFil);

    /* free memory for old log filename */
    free( pszOldFil);

    /* open current log file */
    pfErrLog = fopen( szLogFil, "w");

    if ( pfErrLog == NULL )
    {
        /* error log not opened */
        return( UTL_FTL_ERR);
    }

    /* error log opened OK, so close it */
    fclose( pfErrLog);

    return( UTL_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : utl_f_wrierrlog(...)                                          */
/*                                                                          */
/* NOTES    : Write formatting and optional parameters to error log file    */
/*                                                                          */
/* INPUTS   : szFmtStr              - formatting string                     */
/*          : ...                   - optional parameters                   */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : utl_m_chknulptr(...)                                          */
/*                                                                          */
/* RETURNS  : UtlErrCod                                                     */
/* ------------------------------------------------------------------------ */

UtlErrCod
utl_f_wrierrlog
(
    const   char    *szFmtStr,
            ...
)
{
    char        szLogStr[UTL_STR_MAX]   = "";

    FILE        *pfErrLog               = NULL;
    va_list     vaMarker;

    /* error checking */
    utl_m_chknulptr( szFmtStr);

    /* open existing error log */
    pfErrLog = fopen( UTL_FIL_NAM, "a");

    if ( pfErrLog != NULL )
    {
        /* XLL-friendly vfprintf(...) to write error to log file */
        va_start( vaMarker, szFmtStr);

        _vsnprintf( szLogStr, UTL_STR_MAX, szFmtStr, vaMarker);
        fputs( szLogStr, pfErrLog);

        va_end( vaMarker);
    }

    /* close error log again */
    fclose( pfErrLog);

    return( UTL_SUCCESS);
}

/* ************************************************************************ */
```

C.4.5 tpg_win.c

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : TPG_WIN.C                                                      */
/*                                                                            */
/* NOTES     : Mandatory functions for stand-alone windows DLL                */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 17th February, 1995                                            */
/*                                                                            */
/* VERSION   : 1.0                                                            */
/* ************************************************************************** */

/* ========================================================================== */
/* C headers                                                                  */
/* ========================================================================== */

#include    <ctype.h>
#include    <string.h>
#include    <windows.h>

/* ========================================================================== */
/* SDK headers                                                                */
/* ========================================================================== */

#include    <xlcall.h>
#include    <framewrk.h>

/* ========================================================================== */
/* TPG headers                                                                */
/* ========================================================================== */

#include    "tpg_inf.h"
#include    "tpg_win.h"
#include    "tpg_utl.h"
#include    "tpg_xlo.h"

/* ========================================================================== */
/* External definitions                                                       */
/* ========================================================================== */

extern
const
LPSTR   rgFuncs[INF_FNC_RWS][INF_FNC_CLS];

/* ========================================================================== */
/* Public function definitions                                                */
/* ========================================================================== */

/* ------------------------------------------------------------------------- */
/* FUNCTION : LibMain(...)                                                    */
/*                                                                            */
/* NOTES    : Initialise interface descriptor strings and open error log file */
/*                                                                            */
/* INPUTS   : <none>                                                          */
/*                                                                            */
/* OUTPUTS  : <none>                                                          */
/*                                                                            */
/* CALLS    : <none>                                                          */
/*                                                                            */
/* RETURNS  : int far pascal                                                  */
/* ------------------------------------------------------------------------- */

int far pascal
LibMain
(
    void
)
{
    int i = 0;
    int j = 0;

    /* open error log file */
    if ( utl_f_creerrlog( UTL_FIL_NAM) != UTL_SUCCESS )
    {
        Excel( xlcAlert,
               0,
               2,
               TempStr( " Unable to open error log - TPG not loaded"),
               TempInt( 2));

        return( WIN_FTL_ERR);
    }

    /* byte-count strings in DLL interface descriptor */
    for ( i = 0; i < INF_FNC_RWS; i++ )
    {
        for ( j = 0; j < INF_FNC_CLS; j++ )
        {
            rgFuncs[i][j][0] = (unsigned char) strlen( rgFuncs[i][j] + 1);
        }
    }

    return( WIN_SUCCESS);
}

/* ------------------------------------------------------------------------- */
/* FUNCTION : WEP(...)                                                        */
/*                                                                            */
/* NOTES    : Windows exit procedure: no epilogue required                    */
/*                                                                            */
/* INPUTS   : <none>                                                          */
/*                                                                            */
/* OUTPUTS  : <none>                                                          */
```

```
/*                                                                      */
/* CALLS     : <none>                                                   */
/*                                                                      */
/* RETURNS   : int far pascal                                           */
/* -------------------------------------------------------------------- */

int far pascal
WEP
(
    void
)
{
    return( WIN_SUCCESS);
}

/* -------------------------------------------------------------------- */
/* FUNCTION : xlAutoOpen(...)                                           */
/*                                                                      */
/* NOTES     : Register all add-in functions on opening XLL             */
/*                                                                      */
/* INPUTS    : <none>                                                   */
/*                                                                      */
/* OUTPUTS   : <none>                                                   */
/*                                                                      */
/* CALLS     : <none>                                                   */
/*                                                                      */
/* RETURNS   : int far pascal                                           */
/* -------------------------------------------------------------------- */

int far pascal
xlAutoOpen
(
    void
)
{
    static
    XLOPER  xDLL;

    int     i = 0;

    Excel( xlGetName, &xDLL, 0);

    /* register functions in add-in */
    for ( i = 0; i < INF_FNC_RWS; i++ )
    {
        Excel( xlfRegister,
               0,
               INF_FNC_CLS + 1,
              &xDLL,
               TempStr( rgFuncs[i][0]),
               TempStr( rgFuncs[i][1]),
               TempStr( rgFuncs[i][2]),
               TempStr( rgFuncs[i][3]),
               TempStr( rgFuncs[i][4]),
               TempStr( rgFuncs[i][5]),
               TempStr( rgFuncs[i][6]));
    }

    Excel( xlFree, 0, 1, &xDLL);

    return( WIN_SUCCESS);
}

/* -------------------------------------------------------------------- */
/* FUNCTION : xlAutoClose(...)                                          */
/*                                                                      */
/* NOTES     : Un-register all add-in functions on closing XLL          */
/*                                                                      */
/* INPUTS    : <none>                                                   */
/*                                                                      */
/* OUTPUTS   : <none>                                                   */
/*                                                                      */
/* CALLS     : <none>                                                   */
/*                                                                      */
/* RETURNS   : int far pascal                                           */
/* -------------------------------------------------------------------- */

int far pascal
xlAutoClose
(
    void
)
{
    int i = 0;

    /* un-register functions in add-in */
    for ( i = 0; i < INF_FNC_RWS; i++ )
    {
        Excel( xlfSetName, 0, 1, TempStr( rgFuncs[i][2]));
    }

    return( WIN_SUCCESS);
}

/* -------------------------------------------------------------------- */
/* FUNCTION : xlAutoRegister(...)                                       */
/*                                                                      */
/* NOTES     : Re-register all add-in functions on re-loading XLL       */
/*                                                                      */
/* INPUTS    : pxName                  - add-in function name           */
/*                                                                      */
/* OUTPUTS   : <none>                                                   */
/*                                                                      */
/* CALLS     : <none>                                                   */
/*                                                                      */
/* RETURNS   : LPXLOPER far pascal                                      */
/* -------------------------------------------------------------------- */

LPXLOPER far pascal
xlAutoRegister
(
    const   LPXLOPER    pxName
```

```
)
{
    static
    XLOPER  xDLL;
    static
    XLOPER  xRegId;

    int     i                       = 0;
    char    szName[XLO_STR_MAX] = "";

    xlo_m_setxloerr( xlerrValue, &xRegId);

    /* re-register functions in add-in */
    for ( i = 0; i < INF_FNC_RWS; i++ )
    {
        xlo_f_getxlostr( pxName, szName);

        if ( !strcmp( rgFuncs[i][0], szName) )
        {
            Excel( xlGetName, &xDLL, 0);

            Excel( xlfRegister,
                  &xRegId,
                   INF_FNC_CLS + 1,
                   &xDLL,
                   TempStr( rgFuncs[i][0]),
                   TempStr( rgFuncs[i][1]),
                   TempStr( rgFuncs[i][2]),
                   TempStr( rgFuncs[i][3]),
                   TempStr( rgFuncs[i][4]),
                   TempStr( rgFuncs[i][5]),
                   TempStr( rgFuncs[i][6]));

            Excel( xlFree, 0, 1, &xDLL);

            return( &xRegId);
        }
    }

    return( &xRegId);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : xlAutoAdd(...)                                                */
/*                                                                          */
/* NOTES    : Add XLL to start-up .INI file using Excel add-in manager     */
/*                                                                          */
/* INPUTS   : <none>                                                        */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : <none>                                                        */
/*                                                                          */
/* RETURNS  : int far pascal                                                */
/* ------------------------------------------------------------------------ */

int far pascal
xlAutoAdd
(
    void
)
{
    /* XLL loaded OK */
    Excel( xlcAlert,
           0,
           2,
           TempStr( " Topology Analysis Tool loaded"),
           TempInt( 2));

    return( WIN_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : xlAutoRemove(...)                                             */
/*                                                                          */
/* NOTES    : Remove XLL from start-up .INI file via Excel add-in manager   */
/*                                                                          */
/* INPUTS   : <none>                                                        */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : <none>                                                        */
/*                                                                          */
/* RETURNS  : int far pascal                                                */
/* ------------------------------------------------------------------------ */

int far pascal
xlAutoRemove
(
    void
)
{
    /* XLL removed OK */
    Excel( xlcAlert,
           0,
           2,
           TempStr( " Topology Analysis Tool removed"),
           TempInt( 2));

    return( WIN_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : xlAddInManagerInfo(...)                                       */
/*                                                                          */
/* NOTES    : Get XLL description for Excel add-in manager edit dialog      */
/*                                                                          */
/* INPUTS   : xAction                - Excel add-in manager command         */
/*                                                                          */
/* OUTPUTS  : <none>                                                        */
/*                                                                          */
/* CALLS    : <none>                                                        */
```

```
/*                                                                        */
/* RETURNS   : LPXLOPER far pascal                                        */
/* ---------------------------------------------------------------------- */

LPXLOPER far pascal
xlAddInManagerInfo
(
    const   LPXLOPER xAction
)
{
    static
    XLOPER  xInfo;
    static
    XLOPER  xIntAction;

    Excel( xlCoerce, &xIntAction, 2, xAction, TempInt( xltypeInt));

    /* respond to add-in manager edit event */
    if ( xlo_m_getxloint( &xIntAction) == 1 )
    {
        xlo_f_setxlostr( &xInfo, "Topology Analysis Tool");
    }
    else
    {
        xlo_m_setxloerr( xlerrValue, &xInfo);
    }

    return( &xInfo);
}

/* ********************************************************************** */
```

C.4.6 tpg_xlo.c

```
/* ********************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ********************************************************************** */

/* ********************************************************************** */
/* MODULE    : TPG_XLO.C                                                    */
/*                                                                          */
/* NOTES     : Access functions/macros for Excel XLOPER structure           */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 17th February, 1995                                          */
/*                                                                          */
/* VERSION   : 1.0                                                          */
/* ********************************************************************** */

/* ====================================================================== */
/* C headers                                                              */
/* ====================================================================== */

#include    <stdio.h>
#include    <stdarg.h>
#include    <string.h>
#include    <windows.h>

/* ====================================================================== */
/* SDK headers                                                            */
/* ====================================================================== */

#include    <xlcall.h>

/* ====================================================================== */
/* TPG headers                                                            */
/* ====================================================================== */

#include    "tpg_xlo.h"
#include    "tpg_utl.h"

/* ====================================================================== */
/* Public function definitions                                            */
/* ====================================================================== */

/* ---------------------------------------------------------------------- */
/* FUNCTION : xlo_f_setxlostr(...)                                        */
/*                                                                        */
/* NOTES    : Set XLOPER string contents and type to xltypeStr            */
/*                                                                        */
/* INPUTS   : szString              - zero-terminated string              */
/*          : ...                   - optional arguments                  */
/*                                                                        */
/* OUTPUTS  : pxString              - XLOPER containing a byte-counted string */
```

```
/*                                                                          */
/* CALLS     : <none>                                                       */
/*                                                                          */
/* RETURNS   : XloErrCod                                                    */
/* ------------------------------------------------------------------------ */

XloErrCod
xlo_f_setxlostr
(
            LPXLOPER    pxString,
    const   char       *szFmtStr,
            ...
)
{
    static      char    sString[XLO_STR_MAX]    = "";

    unsigned    char    nBytCnt                 = 0;
                char    szTmpStr[XLO_STR_MAX]   = "";
                va_list vaMarker;

    /* error checking */
    xlo_m_chknulptr( pxString);
    xlo_m_chknulptr( szFmtStr);

    /* clear static char buffer */
    memset( sString, 0, sizeof( sString));

    /* write formatted string to temporary buffer */
    va_start( vaMarker, szFmtStr);
    _vsnprintf( szTmpStr, XLO_STR_MAX - 1, szFmtStr, vaMarker);
    va_end( vaMarker);

    /* get byte count */
    nBytCnt = (unsigned char) strlen( szTmpStr);

    /* copy contents z-string to byte-counted string */
    sString[0] = nBytCnt;
    strncpy( &sString[1], szTmpStr, nBytCnt);

    /* set contents of XLOPER */
    pxString->xltype    = xltypeStr;
    pxString->val.str   = sString;

    return( XLO_SUCCESS);
}

/* ------------------------------------------------------------------------ */
/* FUNCTION : xlo_f_getxlostr(...)                                          */
/*                                                                          */
/* NOTES    : Get XLOPER string contents as a zero-terminated string        */
/*                                                                          */
/* INPUTS   : pxString              - XLOPER containing a byte-counted string */
/*                                                                          */
/* OUTPUTS  : szString              - zero-terminated string                */
/*                                                                          */
/* CALLS    : <none>                                                        */
```

```
/*                                                                          */
/* RETURNS   : XloErrCod                                                    */
/* ------------------------------------------------------------------------ */

XloErrCod
xlo_f_getxlostr
(
    const   LPXLOPER    pxString,
            char       *szString
)
{
    unsigned char   nBytCnt = 0;

    xlo_m_chknulptr( pxString);
    xlo_m_chknulptr( szString);

    /* get length of byte-counted string */
    nBytCnt = pxString->val.str[0];

    if ( nBytCnt == 0 )
    {
        /* copy zero-length z-string, not NULL ptr! */
        strcpy( szString, "");
    }
    else
    {
        /* copy contents of byte-counted string to z-string */
        strncpy( szString, &(pxString->val.str[1]), nBytCnt);
        szString[nBytCnt] = '\0';
    }

    return( XLO_SUCCESS);
}

/* ************************************************************************ */
```

Appendix D

Digital Console Scheduler

In order to compare the performance of scheduling strategies to DMCs, the author has developed a scheduler specifically targetted at the DMC problem domain. This application is coded in the strongly-typed compiled `Prolog` implementation available from Borland.

D.1 Overview

This appendix lists the definition and source files required to re-build *DCS* — the *Digital Console Scheduler*. Table D-1, below, illustrates the various options supported by the *DCS* scheduling engine for static-level (CPM), dynamic-level (DYN) and A* algorithms (AST).

Algorithm[a]	IPC Mechanism	Static Labelling	Dynamic Labelling	Heuristic Underestimate
CPM	FWE (d) NSL NML	HLN HLE (d) LCN LCE RND		
DYN	FWE (d) NSL NML	HLN HLE (d) LCN LCE RND	NON PRO TSK (d) ALL	
AST	FWE (d) NSL NML			NON MPC MIA MRC ALL (d)

Table D-1: Options supported by the *DCS* scheduling engine.

a. (d) indicates that this is the default value for the given option.

D.2 Public Definition Files

The *Digital Console Scheduler* public definition files are listed below.

D.2.1 dcs_ast.pub

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_AST.PUB                                                    */
/*                                                                            */
/* NOTES     : Core predicates for optimal A* DMC scheduling algorithm        */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 10th June, 1995                                                */
/*                                                                            */
/* VERSION   : 8.7                                                            */
/* ************************************************************************** */

/* ========================================================================== */
   global predicates
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : ast_bgn_alg(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Begin A* algorithm with <HrtScheme> underestimate to place */
/*               : <TaskList> tasks with <PrecList> precedence on <ProcList>, */
/*               : binding <Schedule> to the DMC schedule generated           */
/*                                                                            */
/* PARAMETERS    : IpcScheme                                                  */
/*               : HrtScheme                                                  */
/*               : TaskList                                                   */
/*               : PrecList                                                   */
/*               : ProcList                                                   */
/*               : Schedule                                                   */
/*                                                                            */
/* CALLS         : ast_chp_liv(...)                                           */
/*               : ast_gol_nod(...)                                           */
/*               : ast_exp_tre(...)                                           */
/*               : ast_ini_tre(...)                                           */
/*               : sch_cmp_idl(...)                                           */
/*               : utl_lbl_tkf(...)                                           */
/*                                                                            */
/* DATABASE      : bt_open(...)                                               */
/*               : db_flush(...)                                              */
/*               : db_close(...)                                              */
/*               : db_delete(...)                                             */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
ast_bgn_alg
(
    d_ipc_cod,
    d_hrt_cod,
    d_tsk_lst,
    d_pre_lst,
    d_pro_lst,
    d_pro_lst
) - (i, i, i, i, i, o)

/* ************************************************************************** */
```

D.2.2 dcs_clc.pub

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England              */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_CLC.PUB                                                  */
/*                                                                          */
/* NOTES     : Numerical predicates used by DCS modules                     */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 20th May, 1995                                               */
/*                                                                          */
/* VERSION   : 1.2                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   global predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_crt_pth(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Calculate critical-path determined by <PrecList> ensuring */
/*               : all level(...) clauses are removed from internal database */
/*                                                                          */
/* PARAMETERS    : PrecList                                                 */
/*               : CritPath                                                 */
/*                                                                          */
/* CALLS         : clc_ccp_aux(...)                                         */
/*               : utl_lbl_tkf(...)                                         */
/*                                                                          */
/* DATABASE      : retractall( level(...))                                  */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_crt_pth
(
    d_pre_lst,
    d_crt_pth
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_fin_tim(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <SchFin> to finishing-time of <Schedule>, with idle */
/*               : periods completed using sch_cmp_idl(...)                */
/*                                                                          */
/* PARAMETERS    : Schedule                                                 */
/*               : SchFin                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_fin_tim
(
    d_pro_lst,
    d_fin_tim
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_hdw_rsc(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <ProcRsrc> to sum of hardware resource, <ProcList>  */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : ProcRsrc                                                 */
/*                                                                          */
/* CALLS         : clc_chr_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_hdw_rsc
(
    d_pro_lst,
    d_pro_exe
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_max_val(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <Var3> to maximum value of <Var1> and <Var2>        */
/*                                                                          */
/* PARAMETERS    : Var1                                                     */
/*               : Var2                                                     */
/*               : Var3                                                     */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_max_val
```

```
(
    integer,
    integer,
    integer
) - (i, i, o)

determ
clc_max_val
(
    real,
    real,
    real
) - (i, i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_min_val(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <Var3> to minimum value of <Var1> and <Var2>        */
/*                                                                          */
/* PARAMETERS    : Var1                                                     */
/*               : Var2                                                     */
/*               : Var3                                                     */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_min_val
(
    integer,
    integer,
    integer
) - (i, i, o)

determ
clc_min_val
(
    real,
    real,
    real
) - (i, i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_num_ncr(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <NumNCRs> to the number of non-coresident tasks held */
/*               : in <NRCList>                                             */
/*                                                                          */
/* PARAMETERS    : NCRList                                                  */
/*               : NumNCRs                                                  */
/*                                                                          */
/* CALLS         : clc_cnn_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_num_ncr
(
    d_pro_lst,
    d_num_ncr
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_pro_tim(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <ProcTime> to the processing time of the reversed   */
/*               : <TaskList> ignoring idle periods                         */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : ProcTime                                                 */
/*                                                                          */
/* CALLS         : clc_cpt_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_pro_tim
(
    d_tsk_lst,
    d_tsk_exe
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_pro_utl(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <ProcUtil> calculated w.r.t. sample period,         */
/*               : <c_utl_smp>, or <FinishTime>, <c_utl_sch>, for <TaskList> */
/*               : scheduled on <ProcName>                                  */
/*                                                                          */
/* PARAMETERS    : UtlScheme                                                */
/*               : ProcName                                                 */
/*               : TaskList                                                 */
/*               : FinishTime                                               */
/*               : ProcUtil                                                 */
/*                                                                          */
/* CALLS         : clc_pro_tim(...)                                         */
/*               : idb_pro_exe(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */
```

```
determ
clc_pro_utl
(
    d_utl_cod,
    d_pro_nam,
    d_tsk_lst,
    d_fin_tim,
    d_pro_utl
) - (i, i, i, i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_run_tim(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Calculate the <RunTime> required to generate schedule   */
/*                                                                          */
/* PARAMETERS    : RunTime                                                  */
/*                                                                          */
/* CALLS         : clc_sub_cry(...)                                         */
/*                                                                          */
/* DATABASE      : retract( dbatime(...))                                   */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_run_tim
(
    d_sys_tim
) - (o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_seq_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Calculate the sequential schedule <SeqlSchd> for <TaskList> */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : SeqlSchd                                                 */
/*                                                                          */
/* CALLS         : clc_ces_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_seq_sch
(
    d_tsk_lst,
    d_fin_tim
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_sub_cry(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Calculate <Right2> - <Right1> with base <Factor> with carry */
/*               : across to <Left2> if required                            */
/*                                                                          */
/* PARAMETERS    : Left1                                                    */
/*               : Left2                                                    */
/*               : Right1                                                   */
/*               : Right2                                                   */
/*               : Factor                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_sub_cry
(
    integer,
    integer,
    integer,
    integer,
    integer,
    integer
) - (i, o, i, i, o, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_sum_fin(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <FinTot> to sum of finish times of PEs in <ProcList> */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : FinTot                                                   */
/*                                                                          */
/* CALLS         : clc_csf_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_sum_fin
(
    d_pro_lst,
    d_fin_tim
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_thr_put(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <ThroughPut> of <TaskList> as per <Schedule>        */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : Schedule                                                 */
```

```
/*                : ThroughPut                                                  */
/*                                                                              */
/* CALLS          : clc_fin_tim(...)                                            */
/*                : idb_pro_exe(...)                                            */
/*                : lst_num_elm(...)                                            */
/*                                                                              */
/* DATABASE       : --                                                          */
/*                                                                              */
/* BACKTRACKING   : determ                                                      */
/* ---------------------------------------------------------------------------- */

determ
clc_thr_put
(
    d_tsk_lst,
    d_pro_lst,
    d_thr_put
) - (i, i, o)

/* ---------------------------------------------------------------------------- */
/* PREDICATE      : clc_tsk_grn(...)                                            */
/*                                                                              */
/* DESCRIPTION    : Calculate granularity statistics for <TaskList>             */
/*                                                                              */
/* PARAMETERS     : TaskList                                                    */
/*                : MinGran                                                     */
/*                : MaxGran                                                     */
/*                : AvgGran                                                     */
/*                                                                              */
/* CALLS          : clc_ctg_aux(...)                                            */
/*                : clc_seq_sch(...)                                            */
/*                : idb_tsk_exe(...)                                            */
/*                : lst_num_elm(...)                                            */
/*                                                                              */
/* DATABASE       : --                                                          */
/*                                                                              */
/* BACKTRACKING   : determ                                                      */
/* ---------------------------------------------------------------------------- */

determ
clc_tsk_grn
(
    d_tsk_lst,
    d_tsk_exe,
    d_tsk_exe,
    d_tkf_grn
) - (i, o, o, o)

/* ---------------------------------------------------------------------------- */
/* PREDICATE      : clc_tsk_par(...)                                            */
/*                                                                              */
/* DESCRIPTION    : Calculate critical-path, <CritPath>, and taskforce          */
/*                : parallelism, <TaskPara>, for <TaskList> with <PrecList>     */
/*                : precedence relations                                        */
/*                                                                              */
/* PARAMETERS     : TaskList                                                    */
/*                : PrecList                                                    */
/*                : CritPath                                                    */
/*                : TaskPara                                                    */
/*                                                                              */
/* CALLS          : clc_crt_pth(...)                                            */
/*                : clc_seq_sch(...)                                            */
/*                                                                              */
/* DATABASE       : --                                                          */
/*                                                                              */
/* BACKTRACKING   : determ                                                      */
/* ---------------------------------------------------------------------------- */

determ
clc_tsk_par
(
    d_tsk_lst,
    d_pre_lst,
    d_crt_pth,
    d_tkf_par
) - (i, i, o, o)

/* ---------------------------------------------------------------------------- */
/* PREDICATE      : clc_tsk_spd(...)                                            */
/*                                                                              */
/* DESCRIPTION    : Calculate <SpeedUp> % for <Schedule> of <TaskList>          */
/*                                                                              */
/* PARAMETERS     : TaskList                                                    */
/*                : Schedule                                                    */
/*                : SpeedUp                                                     */
/*                                                                              */
/* CALLS          : clc_fin_tim(...)                                            */
/*                : clc_seq_sch(...)                                            */
/*                                                                              */
/* DATABASE       : --                                                          */
/*                                                                              */
/* BACKTRACKING   : determ                                                      */
/* ---------------------------------------------------------------------------- */

determ
clc_tsk_spd
(
    d_tsk_lst,
    d_pro_lst,
    d_sch_spd
) - (i, i, o)

/* ************************************************************************** */
```

D.2.3 dcs_cpm.pub

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_CPM.PUB                                                  */
/*                                                                          */
/* NOTES     : Core predicates for static-level DMC list scheduling         */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 20th May, 1995                                               */
/*                                                                          */
/* VERSION   : 4.6                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   global predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : cpm_bgn_alg(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Begin static labelled <LblScheme> algorithm to schedule  */
/*               : <TaskList> tasks with <PrecList> precedence on <ProcList>, */
/*               : binding <Schedule> to the DMC schedule generated         */
/*                                                                          */
/* PARAMETERS    : IpcScheme                                                */
/*               : LblScheme                                                */
/*               : TaskList                                                 */
/*               : PrecList                                                 */
/*               : ProcList                                                 */
/*               : Schedule                                                 */
/*                                                                          */
/* CALLS         : cpm_cre_sch(...)                                         */
/*               : utl_lbl_tkf(...)                                         */
/*               : utl_srt_prl(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
cpm_bgn_alg
(
    d_ipc_cod,
    d_lbl_cod,
    d_tsk_lst,
    d_pre_lst,
    d_pro_lst,
    d_pro_lst
) - (i, i, i, i, i, o)

/* ************************************************************************ */
```

D.2.4 dcs_dyn.pub

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_DYN.PUB                                                  */
/*                                                                          */
/* NOTES     : Core predicates for dynamic-level DMC list scheduling        */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 28th May, 1995                                               */
/*                                                                          */
/* VERSION   : 7.1                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   global predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_bgn_alg(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Begin dynamically labelled <Dyn> algorithm to schedule   */
/*               : <TaskList> tasks with <PrecList> precedence on <ProcList>, */
/*               : binding <Schedule> to the DMC schedule generated         */
/*                                                                          */
/* PARAMETERS    : Ipc                                                      */
/*               : Lbl                                                      */
/*               : Dyn                                                      */
/*               : TaskList                                                 */
/*               : PrecList                                                 */
/*               : ProcList                                                 */
/*               : Schedule                                                 */
/*                                                                          */
/* CALLS         : dyn_ini_edb(...)                                         */
/*               : dyn_cre_sch(...)                                         */
/*               : utl_lbl_tkf(...)                                         */
/*                                                                          */
/* DATABASE      : db_close(...)                                            */
/*               : db_delete(...)                                           */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
dyn_bgn_alg
(
    d_ipc_cod,
    d_lbl_cod,
    d_dyn_cod,
    d_tsk_lst,
    d_pre_lst,
    d_pro_lst,
    d_pro_lst
) - (i, i, i, i, i, i, o)

/* ************************************************************************ */
```

D.2.5 dcs_edb.pub

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_EDB.PUB                                                    */
/*                                                                            */
/* NOTES     : Display predicates for external database chains and btrees     */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 20th May, 1995                                                 */
/*                                                                            */
/* VERSION   : 2.5                                                            */
/* ************************************************************************** */

/* ========================================================================== */
   global predicates
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : edb_dba_dsp(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Display contents of external DB <DbaSel> on <OutDev>       */
/*                                                                            */
/* PARAMETERS    : DbaSel                                                     */
/*               : OutDev                                                     */
/*                                                                            */
/* CALLS         : edb_edd_aux(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
edb_dba_dsp
(
    db_selector,
    file
) - (i, i)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : edb_btr_dsp(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Write contents of all Btrees held in the external DB       */
/*               : <DbaSel> to the device <OutDev>                            */
/*                                                                            */
/* PARAMETERS    : DbaSel                                                     */
/*               : OutDev                                                     */
/*                                                                            */
/* CALLS         : edb_ebd_aux(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
edb_btr_dsp
(
    db_selector,
    file
) - (i, i)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : edb_dba_sts(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Write statistics from external DB <DbaSel> to <OutDev>     */
/*                                                                            */
/* PARAMETERS    : DbaSel                                                     */
/*               : OutDev                                                     */
/*                                                                            */
/* CALLS         : --                                                         */
/*                                                                            */
/* DATABASE      : db_statistics(...)                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
edb_dba_sts
(
    db_selector,
    file
) - (i, i)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : edb_btr_sts(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Write statistics of all btrees in external DB <DbaSel> to  */
/*               : <OutDev>                                                   */
/*                                                                            */
/* PARAMETERS    : DbaSel                                                     */
/*               : OutDev                                                     */
/*                                                                            */
/* CALLS         : edb_ebs_aux(...)                                           */
/*                                                                            */
/* DATABASE      : db_btrees(...)                                             */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
edb_btr_sts
(
    db_selector,
```

```
    file
) - (i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : edb_mem_sts(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Write memory usage statistics to <OutDev> and bind       */
/*               : <HeapSize> to size of current heap                       */
/*                                                                          */
/* PARAMETERS    : HeapSize                                                 */
/*               : OutDev                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
edb_mem_sts
(
    real,
    file
) - (o, i)

/* ************************************************************************ */
```

D.2.6 dcs_glb.pub

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_GLB.PUB                                                  */
/*                                                                          */
/* NOTES     : Global constants and domains (i.e. forced far *) for DCS     */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 1st June, 1995                                               */
/*                                                                          */
/* VERSION   : 2.5                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   constants
/* ======================================================================== */

% reference numbers and window defaults
c_win_one   =  1
c_win_two   =  2
c_win_thr   =  3
c_win_for   =  4
c_win_mon   =  5
c_win_ref   =  1
c_frm_ctr   = -1
c_win_brd   = "\218\191\192\217\196\179"
c_win_frg   = 30
c_win_bkg   = 31
c_win_idt   =  5
c_win_wid   = 70
c_win_hig   = 10
c_one_org   =  5
c_two_org   = 12

% taskforce, hardware and schedule output files
c_tkf_dir   = "tkf"
c_hdw_dir   = "hdw"
c_sch_dir   = "sch"

% files for schedule output, and dynamic-level and A* external DBs
c_sch_nam   = "tmp.sch"
c_dyn_dba   = "dcs_dyn.bin"
c_ast_dba   = "dcs_ast.bin"

% external database placement, btree key length, and btree order
c_dba_plc   = in_ems
c_btr_len   =  6
c_btr_ord   =  4
```

```
% number of instruction cycles for IPC
c_ipc_tim    =  2

% character codes for ESCAPE and RETURN keys
c_key_esc    = 27
c_key_rtn    = 13

% nulll string and path delimiter
c_nul_str    = ""
c_pth_dlm    = "\\"

% system-wide sampling rate
c_smp_rat    = 48000

% zero offset for dynamic level btree keys
c_off_set    = 100000

/* ========================================================================== */
   global domains
/* ========================================================================== */

% algorithm control parameters
d_alg_cod    = symbol
d_ipc_cod    = symbol
d_lbl_cod    = symbol
d_dyn_cod    = symbol
d_hrt_cod    = symbol

% menu, pre-schedule test and processor utilisation
d_mnu_cod    = symbol
d_tst_cod    = symbol
d_utl_cod    = symbol

% directory, filename, extension and symbolic filenames
d_fil_nam    = string
d_dos_dir    = string
d_dos_ext    = string
d_sym_fil    = symbol

% scheduling statistics
d_pro_rsc    = real
d_pro_utl    = real
d_sch_spd    = real
d_sys_tim    = string
d_thr_put    = real
d_tkf_grn    = real
d_tkf_par    = real

% task levels, critical path, list counts and return status
d_cpm_lvl    = integer
d_crt_pth    = integer
d_dyn_lvl    = integer
d_num_elm    = real
d_num_ncr    = real
d_rtn_stt    = integer

% general list elements
d_int_lst    = integer*
d_rea_lst    = real*
d_str_lst    = string*
d_sym_lst    = symbol*

% task and processor execution time
d_fin_tim    = integer

% task
d_tsk_nam    = symbol
d_tsk_exe    = integer
d_tsk_dom    = tk( d_tsk_nam, d_tsk_exe)
d_tsk_lst    = d_tsk_dom*
d_tsk_typ    = symbol

% precedence relation
d_net_nam    = symbol
d_pre_rel    = prec( d_tsk_nam, d_tsk_nam)
d_pre_lst    = d_pre_rel*

% processor
d_pro_nam    = symbol
d_pro_exe    = integer
d_pro_dom    = pr( d_pro_nam, d_tsk_lst)
d_pro_lst    = d_pro_dom*
d_pro_typ    = symbol

% A* search tree
d_nod_cst    = integer
d_ast_nod    = nd( d_tsk_lst, d_pro_lst, d_fin_tim)

% external database chain, btree key, and dynamic level key offset
d_chn_nam    = string
d_btr_key    = string
d_btr_nam    = string
d_edb_ref    = ref
d_off_set    = real

% structures for algorithm input, algorithm control and external DB control
d_alg_inp    = ai( string, string)
d_alg_ctl    = ac( symbol, symbol, symbol, symbol, symbol)
d_edb_ctl    = ec( db_selector, d_chn_nam, bt_selector)

% external database domains
d_edb_dom    = dlp( d_tsk_nam, d_pro_nam, integer); lf(d_ast_nod, d_nod_cst)

% symbolic file for scheduler output
file         = f_sch_out

% selectors for dynamic-level and A* external databases
db_selector = db_dcs_dyn; db_dcs_ast

/* ************************************************************************** */
```

D.2.7 dcs_idb.pub

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_IDB.PUB                                                  */
/*                                                                          */
/* NOTES     : Access predicates for facts held in internal database        */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 14th May, 1995                                               */
/*                                                                          */
/* VERSION   : 2.2                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   global database
/* ======================================================================== */

% taskforce
task( d_tsk_nam, d_tsk_typ)
tasktype( d_tsk_typ, d_tsk_exe, d_sym_lst, d_sym_lst)
net( d_net_nam, string, d_str_lst)

% hardware
proc( d_pro_nam, d_pro_typ)
proctype( d_pro_typ, d_pro_exe)

% general algorithm facts
prec( d_tsk_nam, d_tsk_nam)
level( d_tsk_nam, d_cpm_lvl)

% sequential cost and algorithm run-time
seqcost( d_exe_tim)
dbatime( integer, integer, integer, integer)

% A* internal facts
node( d_tsk_lst, d_pro_lst, d_exe_tim)
num_sea_nod( real)

/* ======================================================================== */
   global predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE    : idb_sys_tim(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Assert system time into internal database                 */
/*                                                                          */
/* PARAMETERS   : --                                                        */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : assert( dbatime(...))                                     */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
idb_sys_tim
(
)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : idb_tsk_exe(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Non-destructive retract of <ExecTime> of <TaskName>       */
/*                                                                          */
/* PARAMETERS   : TaskName                                                  */
/*              : ExecTime                                                  */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : retract( task(...))                                       */
/*              : assertz( task(...))                                       */
/*              : retract( tasktype(...))                                   */
/*              : assertz( tasktype(...))                                   */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
idb_tsk_exe
(
    d_tsk_nam,
    d_tsk_exe
) - ( i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : idb_tsk_lvl(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Non-destructive retract of <Level> of <TaskName>          */
/*                                                                          */
/* PARAMETERS   : TaskName                                                  */
/*              : Level                                                     */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : retract( level(...))                                      */
/*              : assertz( level(...))                                      */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
idb_tsk_lvl
```

```
(
    d_tsk_nam,
    d_cpm_lvl
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : idb_tsk_com(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Non-destructive succeed if <TaskName1> communicates      */
/*               : directly with <TaskName2>                                */
/*                                                                          */
/* PARAMETERS    : TaskName1                                                */
/*               : TaskName2                                                */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : retract( prec(...))                                      */
/*               : assertz( prec(...))                                      */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
idb_tsk_com
(
    d_tsk_nam,
    d_tsk_nam
) - (o, i), (i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : idb_pre_rel(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Non-destructive succeed if <TaskName1> < <TaskName2>     */
/*                                                                          */
/* PARAMETERS    : TaskName1                                                */
/*               : TaskName2                                                */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : retract( prec(...))                                      */
/*               : assertz( prec(...))                                      */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
idb_pre_rel
(
    d_tsk_nam,
    d_tsk_nam
) - (o, i), (i, o), (i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : idb_pro_exe(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Non-destructive retract of <ExecRsrc> for <ProcName>     */
/*                                                                          */
/* PARAMETERS    : ProcName                                                 */
/*               : ExecRsrc                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : retract( proc(...))                                      */
/*               : assertz( proc(...))                                      */
/*               : retract( proctype(...))                                  */
/*               : assertz( proctype(...))                                  */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
idb_pro_exe
(
    d_pro_nam,
    d_pro_exe
) - (i, o)

/* ************************************************************************ */
```

D.2.8 dcs_lst.pub

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England    */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_LST.PUB                                                  */
/*                                                                          */
/* NOTES     : General-purpose list-handling predicates                     */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 10th January, 1995                                           */
/*                                                                          */
/* VERSION   : 3.8                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   global predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_fst_elm(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Extract first <Elem> from <List>                         */
/*                                                                          */
/* PARAMETERS    : Elem                                                     */
/*               : List                                                     */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
lst_fst_elm
(
    d_int_lst,
    integer
) - (i, o)

determ
lst_fst_elm
(
    d_tsk_lst,
    d_tsk_dom
) - (i, o)

determ
lst_fst_elm
(
    d_pre_lst,
    d_pre_rel
) - (i, o)

determ
lst_fst_elm
(
    d_pro_lst,
    d_pro_dom
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_lst_elm(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Extract last <Elem> from <List>                          */
/*                                                                          */
/* PARAMETERS    : Elem                                                     */
/*               : List                                                     */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
lst_lst_elm
(
    d_int_lst,
    integer
) - (i, o)

determ
lst_lst_elm
(
    d_tsk_lst,
    d_tsk_dom
) - (i, o)

determ
lst_lst_elm
(
    d_pre_lst,
    d_pre_rel
) - (i, o)

determ
lst_lst_elm
(
    d_pro_lst,
    d_pro_dom
) - (i, o)

/* ------------------------------------------------------------------------ */
```

```
/* PREDICATE     : lst_nxt_elm(...)                                        */
/*                                                                         */
/* DESCRIPTION   : Extract next <Elem1> from <List> as <Elem2>             */
/*                                                                         */
/* PARAMETERS    : Elem1                                                   */
/*               : List                                                    */
/*               : Elem2                                                   */
/*                                                                         */
/* CALLS         : SELF                                                    */
/*                                                                         */
/* DATABASE      : --                                                      */
/*                                                                         */
/* BACKTRACKING  : determ                                                  */
/* ----------------------------------------------------------------------- */

determ
lst_nxt_elm
(
    integer,
    d_int_lst,
    integer
) - (i, i, o)

determ
lst_nxt_elm
(
    d_tsk_dom,
    d_tsk_lst,
    d_tsk_dom
) - (i, i, o)

determ
lst_nxt_elm
(
    d_pre_rel,
    d_pre_lst,
    d_pre_rel
) - (i, i, o)

determ
lst_nxt_elm
(
    d_pro_dom,
    d_pro_lst,
    d_pro_dom
) - (i, i, o)

/* ----------------------------------------------------------------------- */
/* PREDICATE     : lst_prv_elm(...)                                        */
/*                                                                         */
/* DESCRIPTION   : Extract previous <Elem1> from <List> as <Elem2>         */
/*                                                                         */
/* PARAMETERS    : Elem1                                                   */
/*               : List                                                    */
/*               : Elem2                                                   */
/*                                                                         */
/* CALLS         : SELF                                                    */
/*                                                                         */
/* DATABASE      : --                                                      */
/*                                                                         */
/* BACKTRACKING  : determ                                                  */
/* ----------------------------------------------------------------------- */

determ
lst_prv_elm
(
    integer,
    d_int_lst,
    integer
) - (i, i, o)

determ
lst_prv_elm
(
    d_tsk_dom,
    d_tsk_lst,
    d_tsk_dom
) - (i, i, o)

determ
lst_prv_elm
(
    d_pre_rel,
    d_pre_lst,
    d_pre_rel
) - (i, i, o)

determ
lst_prv_elm
(
    d_pro_dom,
    d_pro_lst,
    d_pro_dom
) - (i, i, o)

/* ----------------------------------------------------------------------- */
/* PREDICATE     : lst_num_elm(...)                                        */
/*                                                                         */
/* DESCRIPTION   : Count number of elements in <List>                      */
/*                                                                         */
/* PARAMETERS    : List                                                    */
/*               : NumElms                                                 */
/*                                                                         */
/* CALLS         : lst_num_aux(...)                                        */
/*                                                                         */
/* DATABASE      : --                                                      */
/*                                                                         */
/* BACKTRACKING  : determ                                                  */
/* ----------------------------------------------------------------------- */

determ
lst_num_elm
```

```
(
    d_int_lst,
    d_num_elm
) - (i, o)

determ
lst_num_elm
(
    d_tsk_lst,
    d_num_elm
) - (i, o)

determ
lst_num_elm
(
    d_pre_lst,
    d_num_elm
) - (i, o)

determ
lst_num_elm
(
    d_pro_lst,
    d_num_elm
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : lst_ers_one(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Erase one occurance of <Elem> from <List1> to give <List2> */
/*                                                                          */
/* PARAMETERS   : Elem                                                      */
/*              : List1                                                     */
/*              : List2                                                     */
/*                                                                          */
/* CALLS        : SELF                                                      */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
lst_ers_one
(
    integer,
    d_int_lst,
    d_int_lst
) - (i, i, o)

determ
lst_ers_one
(
    d_tsk_dom,
    d_tsk_lst,
    d_tsk_lst
) - (i, i, o)

determ
lst_ers_one
(
    d_pre_rel,
    d_pre_lst,
    d_pre_lst
) - (i, i, o)

determ
lst_ers_one
(
    d_pro_dom,
    d_pro_lst,
    d_pro_lst
) - (i, i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : lst_ers_all(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Erase all occurances of <Elem> from <List1> to give <List2> */
/*                                                                          */
/* PARAMETERS   : Elem                                                      */
/*              : List1                                                     */
/*              : List2                                                     */
/*                                                                          */
/* CALLS        : SELF                                                      */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
lst_ers_all
(
    integer,
    d_int_lst,
    d_int_lst
) - (i, i, o)

determ
lst_ers_all
(
    d_tsk_dom,
    d_tsk_lst,
    d_tsk_lst
) - (i, i, o)

determ
lst_ers_all
(
    d_pre_rel,
    d_pre_lst,
    d_pre_lst
```

```
) - (i, i, o)

determ
lst_ers_all
(
    d_pro_dom,
    d_pro_lst,
    d_pro_lst
) - (i, i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : lst_mem_elm(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Check <Elem> is a member of <List>                        */
/*                                                                          */
/* PARAMETERS   : Elem                                                      */
/*              : List                                                      */
/*                                                                          */
/* CALLS        : SELF                                                      */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
lst_mem_elm
(
    integer,
    d_int_lst
) - (i, i), (o, i)

determ
lst_mem_elm
(
    string,
    d_str_lst
) - (i, i), (o, i)

determ
lst_mem_elm
(
    d_tsk_dom,
    d_tsk_lst
) - (i, i), (o, i)

determ
lst_mem_elm
(
    d_pre_rel,
    d_pre_lst
) - (i, i), (o, i)

determ
lst_mem_elm
(
    d_pro_dom,
    d_pro_lst
) - (i, i), (o, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : lst_mem_ndt(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Generate each <Elem> which is a member of <List>          */
/*                                                                          */
/* PARAMETERS   : Elem                                                      */
/*              : List                                                      */
/*                                                                          */
/* CALLS        : SELF                                                      */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : non-determ                                                */
/* ------------------------------------------------------------------------ */

nondeterm
lst_mem_ndt
(
    integer,
    d_int_lst
) - (i, i), (o, i)

nondeterm
lst_mem_ndt
(
    d_tsk_dom,
    d_tsk_lst
) - (i, i), (o, i)

nondeterm
lst_mem_ndt
(
    d_pre_rel,
    d_pre_lst
) - (i, i), (o, i)

nondeterm
lst_mem_ndt
(
    d_pro_dom,
    d_pro_lst
) - (i, i), (o, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : lst_del_elm(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Delete <Elem> from <List1> and bind <List2> to result     */
/*                                                                          */
/* PARAMETERS   : Elem                                                      */
/*              : List1                                                     */
/*              : List2                                                     */
/*                                                                          */
```

```
/* CALLS        : SELF                                                    */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
lst_del_elm
(
    integer,
    d_int_lst,
    d_int_lst
) - (o, i, o), (o, i, i), (i, i, o)

determ
lst_del_elm
(
    d_tsk_dom,
    d_tsk_lst,
    d_tsk_lst
) - (o, i, o), (o, i, i), (i, i, o)

determ
lst_del_elm
(
    d_pre_rel,
    d_pre_lst,
    d_pre_lst
) - (o, i, o), (o, i, i), (i, i, o)

determ
lst_del_elm
(
    d_pro_dom,
    d_pro_lst,
    d_pro_lst
) - (o, i, o), (o, i, i), (i, i, o)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : lst_del_ndt(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Generate all <Elem> from <List1>, binding <List2> to rest */
/*                                                                        */
/* PARAMETERS   : Elem                                                    */
/*              : List1                                                   */
/*              : List2                                                   */
/*                                                                        */
/* CALLS        : SELF                                                    */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : non-determ                                              */
/* ---------------------------------------------------------------------- */

nondeterm
lst_del_ndt
(
    integer,
    d_int_lst,
    d_int_lst
) - (o, i, o), (o, i, i), (i, i, o)

nondeterm
lst_del_ndt
(
    d_tsk_dom,
    d_tsk_lst,
    d_tsk_lst
) - (o, i, o), (o, i, i), (i, i, o)

nondeterm
lst_del_ndt
(
    d_pre_rel,
    d_pre_lst,
    d_pre_lst
) - (o, i, o), (o, i, i), (i, i, o)

nondeterm
lst_del_ndt
(
    d_pro_dom,
    d_pro_lst,
    d_pro_lst
) - (o, i, o), (o, i, i), (i, i, o)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : lst_apd_lst(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Bind <List3> as <List2> appended to <List1>             */
/*                                                                        */
/* PARAMETERS   : List1                                                   */
/*              : List2                                                   */
/*              : List3                                                   */
/*                                                                        */
/* CALLS        : SELF                                                    */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
lst_apd_lst
(
    d_int_lst,
    d_int_lst,
    d_int_lst
) - (i, i, o)

determ
```

```
lst_apd_lst
(
    d_tsk_lst,
    d_tsk_lst,
    d_tsk_lst
) - (i, i, o)

determ
lst_apd_lst
(
    d_pre_lst,
    d_pre_lst,
    d_pre_lst
) - (i, i, o)

determ
lst_apd_lst
(
    d_pro_lst,
    d_pro_lst,
    d_pro_lst
) - (i, i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : lst_apd_ndt(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Generate all <List3> as <List2> appended to <List1>       */
/*                                                                          */
/* PARAMETERS   : List1                                                     */
/*              : List2                                                     */
/*              : List3                                                     */
/*                                                                          */
/* CALLS        : SELF                                                      */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : non-determ                                                */
/* ------------------------------------------------------------------------ */

nondeterm
lst_apd_ndt
(
    d_int_lst,
    d_int_lst,
    d_int_lst
) - (i, i, o), (o, o, i)

nondeterm
lst_apd_ndt
(
    d_tsk_lst,
    d_tsk_lst,
    d_tsk_lst
) - (i, i, o), (o, o, i)

nondeterm
lst_apd_ndt
(
    d_pre_lst,
    d_pre_lst,
    d_pre_lst
) - (i, i, o), (o, o, i)

nondeterm
lst_apd_ndt
(
    d_pro_lst,
    d_pro_lst,
    d_pro_lst
) - (i, i, o), (o, o, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : lst_rev_lst(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Reverse <List1> to bind <List2>                           */
/*                                                                          */
/* PARAMETERS   : List1                                                     */
/*              : List2                                                     */
/*                                                                          */
/* CALLS        : lst_rev_apd(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
lst_rev_lst
(
    d_int_lst,
    d_int_lst
) - (i, o)

determ
lst_rev_lst
(
    d_tsk_lst,
    d_tsk_lst
) - (i, o)

determ
lst_rev_lst
(
    d_pre_lst,
    d_pre_lst
) - (i, o)

determ
lst_rev_lst
(
    d_pro_lst,
    d_pro_lst
```

```
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_rev_apd(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <List3> = <ltsiL> + <List2>                         */
/*                                                                          */
/* PARAMETERS    : Elem                                                     */
/*               : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
lst_rev_apd
(
    d_int_lst,
    d_int_lst,
    d_int_lst
) - (i, i, o)

determ
lst_rev_apd
(
    d_tsk_lst,
    d_tsk_lst,
    d_tsk_lst
) - (i, i, o)

determ
lst_rev_apd
(
    d_pre_lst,
    d_pre_lst,
    d_pre_lst
) - (i, i, o)

determ
lst_rev_apd
(
    d_pro_lst,
    d_pro_lst,
    d_pro_lst
) - (i, i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_per_mut(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Generate all permutations of <List1> as <List2>          */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_apd_ndt(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : non-determ                                               */
/* ------------------------------------------------------------------------ */

nondeterm
lst_per_mut
(
    d_int_lst,
    d_int_lst
) - (i, o)

nondeterm
lst_per_mut
(
    d_tsk_lst,
    d_tsk_lst
) - (i, o)

nondeterm
lst_per_mut
(
    d_pre_lst,
    d_pre_lst
) - (i, o)

nondeterm
lst_per_mut
(
    d_pro_lst,
    d_pro_lst
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_equ_lst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Check list equality relation, i.e. <List1> = <List2>     */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
lst_equ_lst
(
```

```
    d_int_lst,
    d_int_lst
) - (i, i)

determ
lst_equ_lst
(
    d_tsk_lst,
    d_tsk_lst
) - (i, i)

determ
lst_equ_lst
(
    d_pre_lst,
    d_pre_lst
) - (i, i)

determ
lst_equ_lst
(
    d_pro_lst,
    d_pro_lst
) - (i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_equ_set(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Check set equality relation for two lists                */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_del_ndt(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

nondeterm
lst_equ_set
(
    d_int_lst,
    d_int_lst
) - (i, i)

nondeterm
lst_equ_set
(
    d_tsk_lst,
    d_tsk_lst
) - (i, i)

nondeterm
lst_equ_set
(
    d_pre_lst,
    d_pre_lst
) - (i, i)

nondeterm
lst_equ_set
(
    d_pro_lst,
    d_pro_lst
) - (i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_sub_lst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Subtract <List1> from <List2> to give <List3>            */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*               : List3                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_mem_ndt(...)                                         */
/*               : lst_del_ndt(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

nondeterm
lst_sub_lst
(
    d_int_lst,
    d_int_lst,
    d_int_lst
) - (i, i, o)

nondeterm
lst_sub_lst
(
    d_tsk_lst,
    d_tsk_lst,
    d_tsk_lst
) - (i, i, o)

nondeterm
lst_sub_lst
(
    d_pre_lst,
    d_pre_lst,
    d_pre_lst
) - (i, i, o)

nondeterm
```

```
lst_sub_lst
(
    d_pro_lst,
    d_pro_lst,
    d_pro_lst
) - (i, i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_sub_set(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Check <List1> is a sub-set of <List2>                    */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_mem_elm(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
lst_sub_set
(
    d_int_lst,
    d_int_lst
) - (i, i)

determ
lst_sub_set
(
    d_tsk_lst,
    d_tsk_lst
) - (i, i)

determ
lst_sub_set
(
    d_pre_lst,
    d_pre_lst
) - (i, i)

determ
lst_sub_set
(
    d_pro_lst,
    d_pro_lst
) - (i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_dis_jnt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Check if the set <List1> and <List2> are disjoint        */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : lst_mem_elm(...)                                         */
/*               : lst_mem_ndt(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
lst_dis_jnt
(
    d_int_lst,
    d_int_lst
) - (i, i)

determ
lst_dis_jnt
(
    d_tsk_lst,
    d_tsk_lst
) - (i, i)

determ
lst_dis_jnt
(
    d_pre_lst,
    d_pre_lst
) - (i, i)

determ
lst_dis_jnt
(
    d_pro_lst,
    d_pro_lst
) - (i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_int_sec(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create <List3> as the intersection of <List1> and <List2> */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*               : List3                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_mem_elm(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */
```

```
determ
lst_int_sec
(
    d_int_lst,
    d_int_lst,
    d_int_lst
) - (i, i, o)

determ
lst_int_sec
(
    d_tsk_lst,
    d_tsk_lst,
    d_tsk_lst
) - (i, i, o)

determ
lst_int_sec
(
    d_pre_lst,
    d_pre_lst,
    d_pre_lst
) - (i, i, o)

determ
lst_int_sec
(
    d_pro_lst,
    d_pro_lst,
    d_pro_lst
) - (i, i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : lst_set_uni(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Create <List3> as the union of <List1> and <List2>        */
/*                                                                          */
/* PARAMETERS   : List1                                                     */
/*              : List2                                                     */
/*              : List3                                                     */
/*                                                                          */
/* CALLS        : SELF                                                      */
/*              : lst_mem_elm(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
lst_set_uni
(
    d_int_lst,
    d_int_lst,
    d_int_lst
) - (i, i, o)

determ
lst_set_uni
(
    d_tsk_lst,
    d_tsk_lst,
    d_tsk_lst
) - (i, i, o)

determ
lst_set_uni
(
    d_pre_lst,
    d_pre_lst,
    d_pre_lst
) - (i, i, o)

determ
lst_set_uni
(
    d_pro_lst,
    d_pro_lst,
    d_pro_lst
) - (i, i, o)

/* ************************************************************************ */
```

D.2.9 dcs_sch.pub

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_SCH.PUB                                                    */
/*                                                                            */
/* NOTES     : Application-specific scheduling predicates for DCS engine      */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 15th June, 1995                                                */
/*                                                                            */
/* VERSION   : 7.5                                                            */
/* ************************************************************************** */

/* ========================================================================== */
   global predicates
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : sch_pre_rel(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Fail if any predecessors of <Task> are in <TaskList>       */
/*                                                                            */
/* PARAMETERS    : Task                                                       */
/*               : TaskList                                                   */
/*                                                                            */
/* CALLS         : idb_pre_rel(...)                                           */
/*               : lst_mem_ndt(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

nondeterm
sch_pre_rel
(
    d_tsk_dom,
    d_tsk_lst
) - (i, i)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : sch_act_tsk(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Fail if any predecessors of <Task> in <ProcList> finish    */
/*               : execution after <Fin>                                      */
/*                                                                            */
/* PARAMETERS    : Task                                                       */
/*               : Fin                                                        */
/*               : ProcList                                                   */
/*                                                                            */
/* CALLS         : idb_pre_rel(...)                                           */
/*               : lst_mem_ndt(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
sch_act_tsk
(
    d_tsk_dom,
    d_tsk_exe,
    d_pro_lst
) - (i, i, i)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : sch_pro_rsc(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Fail if executing <Task> on <Proc> with current finishing  */
/*               : time of <FinTime> exceeds processing resource of <Proc>    */
/*                                                                            */
/* PARAMETERS    : Task                                                       */
/*               : ProcName                                                   */
/*               : FinTime                                                    */
/*                                                                            */
/* CALLS         : idb_pro_exe(...)                                           */
/*               : lst_tsk_exe(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
sch_pro_rsc
(
    d_tsk_dom,
    d_pro_nam,
    d_fin_tim
) - (i, i, i)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : sch_non_cor(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Bind <NCRList> to those tasks already scheduled in         */
/*               : <ActiveProcs> which are predecessors of <Task> and         */
/*               : therefore NOT co-resident                                  */
/*                                                                            */
/* PARAMETERS    : Task                                                       */
/*               : ActiveProcs                                                */
/*               : NCRList                                                    */
/*                                                                            */
/* CALLS         : sch_snc_aux(...)                                           */
```

```
/*                                                                      */
/* DATABASE      : --                                                   */
/*                                                                      */
/* BACKTRACKING  : determ                                               */
/* -------------------------------------------------------------------- */

determ
sch_non_cor
(
    d_tsk_dom,
    d_pro_lst,
    d_pro_lst
) - (i, i, o)

/* -------------------------------------------------------------------- */
/* PREDICATE     : sch_ins_ipc(...)                                     */
/*                                                                      */
/* DESCRIPTION   : Schedule IPC according to <IpcScheme> for non-coresident */
/*               : tasks in <NCRList> on processors in <ProcList> at finish */
/*               : time <FinTime>. <TaskList> is bound to IPC required on */
/*               : destination processor                                */
/*                                                                      */
/* PARAMETERS    : IpcScheme                                            */
/*               : NCRList                                              */
/*               : OldProcList                                          */
/*               : ProcList                                             */
/*               : OldFinTime                                           */
/*               : FinTime                                              */
/*               : OldTaskList                                          */
/*               : TaskList                                             */
/*                                                                      */
/* CALLS         : clc_max_val(...)                                     */
/*               : lst_del_ndt(...)                                     */
/*               : lst_num_elm(...)                                     */
/*               : sch_ins_ipc(...)                                     */
/*               : sch_ipc_dst(...)                                     */
/*               : sch_ipc_non(...)                                     */
/*               : sch_ipc_src(...)                                     */
/*                                                                      */
/* DATABASE      : --                                                   */
/*                                                                      */
/* BACKTRACKING  : determ                                               */
/* -------------------------------------------------------------------- */

determ
sch_ins_ipc
(
    d_ipc_cod,
    d_pro_lst,
    d_pro_lst,
    d_pro_lst,
    d_fin_tim,
    d_fin_tim,
    d_tsk_lst,
    d_tsk_lst
) - (i, i, i, o, i, o, i, o)
```

```
/* -------------------------------------------------------------------- */
/* PREDICATE     : sch_ins_tsk(...)                                     */
/*                                                                      */
/* DESCRIPTION   : Insert a task on processor <Proc> with <ActiveProcs> active */
/*               : at <OldFinTime> to give <ProcList> processors active until */
/*               : the finish time <Fin>                                */
/*                                                                      */
/* PARAMETERS    : Proc                                                 */
/*               : ActiveProcs                                          */
/*               : ProcList                                             */
/*               : OldFin                                               */
/*               : Fin                                                  */
/*                                                                      */
/* CALLS         : SELF                                                 */
/*                                                                      */
/* DATABASE      : --                                                   */
/*                                                                      */
/* BACKTRACKING  : determ                                               */
/* -------------------------------------------------------------------- */

determ
sch_ins_tsk
(
    d_pro_dom,
    d_pro_lst,
    d_pro_lst,
    d_fin_tim,
    d_fin_tim
) - ( i, i, o, i, o)

/* -------------------------------------------------------------------- */
/* PREDICATE     : sch_ins_idl(...)                                     */
/*                                                                      */
/* DESCRIPTION   : Insert an idle period on processor <Proc> until the next */
/*               : processor in <ActiveProcs> completes to give <ProcList> */
/*                                                                      */
/* PARAMETERS    : Proc                                                 */
/*               : ActiveProcs                                          */
/*               : ProcList                                             */
/*                                                                      */
/* CALLS         : SELF                                                 */
/*                                                                      */
/* DATABASE      : --                                                   */
/*                                                                      */
/* BACKTRACKING  : determ                                               */
/* -------------------------------------------------------------------- */

determ
sch_ins_idl
(
    d_pro_dom,
    d_pro_lst,
    d_pro_lst
) - ( i, i, o)
```

```
/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_cmp_idl(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Complete idle periods on processors in <ProcList> which  */
/*               : finish executing tasks before the finish time of <Schedule> */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : FinTime                                                  */
/*               : Schedule                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : sch_ins_idl(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
sch_cmp_idl
(
    d_pro_lst,
    d_fin_tim,
    d_pro_lst
) - (i, i, o)

/* ************************************************************************ */
```

D.2.10 dcs_trl.pub

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_TRL.PUB                                                  */
/*                                                                          */
/* NOTES     : Translation predicates for algorithm control symbols         */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 1st March, 1995                                              */
/*                                                                          */
/* VERSION   : 1.1                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   global predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : trl_alg_sym(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Translate <AlgCod> to scheduling algorithm string <AlgStr> */
/*                                                                          */
/* PARAMETERS    : AlgCod                                                   */
/*               : AlgStr                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
trl_alg_sym
(
    d_alg_cod,
    string
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : trl_ipc_sym(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Translate <IpcCod> to IPC scheme string <AlgStr>         */
/*                                                                          */
/* PARAMETERS    : IpcCod                                                   */
/*               : IpcStr                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
```

```
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
trl_ipc_sym
(
    d_ipc_cod,
    string
) - (i, o)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : trl_lbl_sym(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Translate <LblCod> to static labelling string <LblStr>  */
/*                                                                        */
/* PARAMETERS   : LblCod                                                  */
/*              : LblStr                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
trl_lbl_sym
(
    d_lbl_cod,
    string
) - (i, o)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : trl_dyn_sym(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Translate <DynCod> to dynamic labelling string <DynStr> */
/*                                                                        */
/* PARAMETERS   : DynCod                                                  */
/*              : DynStr                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
trl_dyn_sym
(
    d_dyn_cod,
    string
) - (i, o)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : trl_hrt_sym(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Translate <HrtCod> to heuristic string <HrtStr>         */
/*                                                                        */
/* PARAMETERS   : HrtCod                                                  */
/*              : HrtStr                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
trl_hrt_sym
(
    d_hrt_cod,
    string
) - (i, o)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : trl_utl_sym(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Translate <UtlCod> to utilisation string <UtlStr>       */
/*                                                                        */
/* PARAMETERS   : UtlCod                                                  */
/*              : UtlStr                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
trl_utl_sym
(
    d_utl_cod,
    string
) - (i, o)

/* ********************************************************************** */
```

D.2.11 dcs_utl.pub

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England              */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_UTL.PUB                                                  */
/*                                                                          */
/* NOTES     : Utility predicates for DCS scheduling engine                 */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 14th June, 1995                                              */
/*                                                                          */
/* VERSION   : 7.2                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   global predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_srt_prl(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Quick-sort <TaskList1> to bind <TaskList2> to priority list */
/*                                                                          */
/* PARAMETERS    : TaskList1                                                */
/*               : TaskList2                                                */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_apd_lst(...)                                         */
/*               : utl_spl_tkl(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_srt_prl
(
    d_tsk_lst,
    d_tsk_lst
) - (i, o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_cre_tsk(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create list of task <TaskList> from facts in internal DB */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*                                                                          */
/* CALLS         : utl_tsk_aux(...)                                         */
/*                                                                          */
/* DATABASE      : assertz( task(...))                                      */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_cre_tsk
(
    d_tsk_lst
) - (o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_cre_pre(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create list of precedence relationships <PrecList> from  */
/*               : facts in internal DB                                     */
/*                                                                          */
/* PARAMETERS    : PrecList                                                 */
/*                                                                          */
/* CALLS         : utl_pre_aux(...)                                         */
/*                                                                          */
/* DATABASE      : assertz( net(...))                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_cre_pre
(
    d_pre_lst
) - (o)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_cre_pro(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create list of processors <ProcList> from facts help in  */
/*               : internal DB                                              */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*                                                                          */
/* CALLS         : utl_pro_aux(...)                                         */
/*                                                                          */
/* DATABASE      : assertz( proc(...))                                      */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_cre_pro
(
    d_pro_lst
) - (o)

/* ------------------------------------------------------------------------ */
```

```
/* PREDICATE     : utl_lbl_tkf(...)                                        */
/*                                                                         */
/* DESCRIPTION   : Label taskforce with <PrecList> precedence relationships */
/*               : according to <Labelling> static labelling scheme        */
/*                                                                         */
/* PARAMETERS    : Labelling                                               */
/*               : PrecList                                                */
/*                                                                         */
/* CALLS         : utl_ini_lvl(...)                                        */
/*               : utl_lbl_all(...)                                        */
/*                                                                         */
/* DATABASE      : --                                                      */
/*                                                                         */
/* BACKTRACKING  : determ                                                  */
/* ----------------------------------------------------------------------- */

determ
utl_lbl_tkf
(
    d_lbl_cod,
    d_pre_lst
) - (i, i)

/* ----------------------------------------------------------------------- */
/* PREDICATE     : utl_rep_eat(...)                                        */
/*                                                                         */
/* DESCRIPTION   : Generate an unlimited supply of backtracking points     */
/*                                                                         */
/* PARAMETERS    : --                                                      */
/*                                                                         */
/* CALLS         : SELF                                                    */
/*                                                                         */
/* DATABASE      : --                                                      */
/*                                                                         */
/* BACKTRACKING  : non-determ                                              */
/* ----------------------------------------------------------------------- */

nondeterm
utl_rep_eat
(
)

/* ----------------------------------------------------------------------- */
/* PREDICATE     : utl_kyd_pau(...)                                        */
/*                                                                         */
/* DESCRIPTION   : Pause until any character is received from keyboard     */
/*                                                                         */
/* PARAMETERS    : --                                                      */
/*                                                                         */
/* CALLS         : utl_rep_eat(...)                                        */
/*                                                                         */
/* DATABASE      : --                                                      */
/*                                                                         */
/* BACKTRACKING  : determ                                                  */
/* ----------------------------------------------------------------------- */

determ
utl_kyd_pau
(
)

/* ********************************************************************** */
```

D.2.12 dcs_wri.pub

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England              */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_WRI.PUB                                                  */
/*                                                                          */
/* NOTES     : Display predicates for DCS internal lists and other structures */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 1st March, 1995                                              */
/*                                                                          */
/* VERSION   : 2.1                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   global predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_inp_prm(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Write input parameters including taskforce filename      */
/*               : <TkfFil>, hardware filename <HdwFil> and algorithm control */
/*               : parameters <AlgCtl> to default output device             */
/*                                                                          */
/* PARAMETERS    : TkfFil                                                   */
/*               : HdwFil                                                   */
/*               : AlgCtl                                                   */
/*                                                                          */
/* CALLS         : trl_alg_sym(...)                                         */
/*               : trl_ipc_sym(...)                                         */
/*               : trl_lbl_sym(...)                                         */
/*               : trl_dyn_sym(...)                                         */
/*               : trl_hrt_sym(...)                                         */
/*               : wri_dlm_lin(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
wri_inp_prm
(
    d_fil_nam,
    d_fil_nam,
    d_alg_ctl
) - (i, i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_pre_stt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Write pre-schedule statistics to default output device   */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : PrecList                                                 */
/*               : ProcList                                                 */
/*                                                                          */
/* CALLS         : lst_num_elm(...)                                         */
/*               : clc_seq_sch(...)                                         */
/*               : clc_tsk_par(...)                                         */
/*               : clc_tst_grn(...)                                         */
/*               : clc_hdw_rsc(...)                                         */
/*               : wri_dlm_lin(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
wri_pre_stt
(
    d_tsk_lst,
    d_pre_lst,
    d_pro_lst
) - (i, i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_pre_chk(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Write pre-schedule checks to default output device and bind */
/*               : <TestList> to list of check results                      */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : PrecList                                                 */
/*               : ProcList                                                 */
/*               : TestList                                                 */
/*                                                                          */
/* CALLS         : dcs_grn_tst(...)                                         */
/*               : dcs_fit_tst(...)                                         */
/*               : dcs_ctp_tst(...)                                         */
/*               : wri_dlm_lin(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
wri_pre_chk
(
    d_sym_lst
) - (i)
```

```
/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_wat_dog(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Retract search-tree watch-dogs from internal DB and display */
/*                                                                          */
/* PARAMETERS    : AlgCtl                                                   */
/*                                                                          */
/* CALLS         : wri_dlm_lin(...)                                         */
/*                                                                          */
/* DATABASE      : retract( num_sea_nod(...))                               */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
wri_wat_dog
(
    d_alg_ctl
) - (i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_mtp_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Output M-P schedule held in <Schedule> to default device */
/*                                                                          */
/* PARAMETERS    : Schedule                                                 */
/*                                                                          */
/* CALLS         : wri_dlm_lin(...)                                         */
/*               : wri_pro_lst(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
wri_mtp_sch
(
    d_pro_lst
) - (i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_pst_stt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Write post-schedule statistics to default output device */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : Schedule                                                 */
/*                                                                          */
/* CALLS         : clc_run_tim(...)                                         */
/*               : clc_fin_tim(...)                                         */
/*               : clc_tsk_spd(...)                                         */
/*               : clc_thr_put(...)                                         */
/*               : clc_deq_sch(...)                                         */
/*               : wri_dlm_lin(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
wri_pst_stt
(
    d_alg_ctl,
    d_tsk_lst,
    d_pro_lst
) - (i, i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_alg_mon(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Write block to algorithm monitor gauge                   */
/*                                                                          */
/* PARAMETERS    : --                                                       */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
wri_alg_mon
(
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_dcs_tst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Translate <PassFail> and write to default output device */
/*                                                                          */
/* PARAMETERS    : Prompt                                                   */
/*               : PassFail                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
wri_dcs_tst
(
    string,
    d_tst_cod
) - (i, i)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_dlm_lin(...)                                         */
```

```
/*                                                                        */
/* DESCRIPTION  : Write delimiter string of <Chr> into scheduler output file */
/*                                                                        */
/* PARAMETERS   : Chr                                                     */
/*                                                                        */
/* CALLS        : wri_wdl_aux(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
wri_dlm_lin
(
    char
) - (i)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : wri_pre_lst(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write precedence relations in <PrecList> to output device */
/*                                                                        */
/* PARAMETERS   : PrecList                                                */
/*                                                                        */
/* CALLS        : SELF                                                    */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
wri_pre_lst
(
    d_pre_lst
) - (i)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : wri_pro_lst(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write processor states in <ProcList> to output device   */
/*                                                                        */
/* PARAMETERS   : ProcList                                                */
/*                                                                        */
/* CALLS        : SELF                                                    */
/*              : lst_rev_lst(...)                                        */
/*              : wri_tsk_lst(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
wri_pro_lst
(
    d_pro_lst
) - (i)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : wri_pro_utl(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write processor utilisations of type <UtilSym> exhibited in */
/*              : <Schedule> for tasks with sequential schedule <SeqCost> */
/*                                                                        */
/* PARAMETERS   : UtilSym                                                 */
/*              : Schedule                                                */
/*              : SeqCost                                                 */
/*                                                                        */
/* CALLS        : lst_num_elm(...)                                        */
/*              : wri_wpu_aux(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
wri_pro_utl
(
    d_utl_cod,
    d_pro_lst,
    d_fin_tim
) - (i, i, i)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : wri_tsk_lst(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write tasks in <TaskList> to default output device      */
/*                                                                        */
/* PARAMETERS   : TaskList                                                */
/*                                                                        */
/* CALLS        : SELF                                                    */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
wri_tsk_lst
(
    d_tsk_lst
) - (i)

/* ********************************************************************** */
```

D.3 Private Definition Files

DCS private predicate definition files are listed in this section.

D.3.1 dcs_ast.prv

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_AST.PRV                                                    */
/*                                                                            */
/* NOTES     : Core predicates for optimal A* DMC scheduling algorithm        */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 10th June, 1995                                                */
/*                                                                            */
/* VERSION   : 8.7                                                            */
/* ************************************************************************** */

/* ========================================================================== */
   predicates
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : ast_ini_tre(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Initialise search tree for <TaskList> on <ProcList>        */
/*                                                                            */
/* PARAMETERS    : TaskList                                                   */
/*               : ProcList                                                   */
/*                                                                            */
/* CALLS         : ast_key_int(...)                                           */
/*                                                                            */
/* DATABASE      : db_create(...)                                             */
/*               : bt_create(...)                                             */
/*               : bt_close(...)                                              */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
ast_ini_tre
(
    d_tsk_lst,
    d_pro_lst
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : ast_chp_liv(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Bind <Node> to cheapest live node in search tree <BtrSel>  */
/*                                                                            */
/* PARAMETERS    : BteSel                                                     */
/*               : Node                                                       */
/*                                                                            */
/* CALLS         : --                                                         */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
ast_chp_liv
(
    bt_selector,
    d_ast_nod
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : ast_exp_tre(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Generate all successors of <Node>, using <IPCScheme> and   */
/*               : and <HrtScheme>, and insert them into the A* search tree   */
/*                                                                            */
/* PARAMETERS    : IpcScheme                                                  */
/*               : HrtScheme                                                  */
/*               : BtrSel                                                     */
/*               : Node                                                       */
/*                                                                            */
/* CALLS         : ast_ins_new(...)                                           */
/*               : ast_suc_nod(...)                                           */
/*               : wri_alg_mon(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
ast_exp_tre
(
    d_ipc_cod,
    d_hrt_cod,
    bt_selector,
    d_ast_nod
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : ast_suc_nod(...)                                           */
/*                                                                            */
```

```
/* DESCRIPTION  : Generate a successor to <OldNode>, using <IPCScheme>        */
/*                                                                             */
/* PARAMETERS   : IpcScheme                                                    */
/*              : OldNode                                                      */
/*              : Node                                                         */
/*                                                                             */
/* CALLS        : idb_tsk_exe(...)                                             */
/*              : lst_del_ndt(...)                                             */
/*              : sch_act_tsk(...)                                             */
/*              : sch_ins_idl(...)                                             */
/*              : sch_ins_ipc(...)                                             */
/*              : sch_ins_tsk(...)                                             */
/*              : sch_non_cor(...)                                             */
/*              : sch_pre_rel(...)                                             */
/*              : sch_pro_rsc(...)                                             */
/*                                                                             */
/* DATABASE     : --                                                           */
/*                                                                             */
/* BACKTRACKING : non-determ                                                   */
/* ------------------------------------------------------------------------- */

nondeterm
ast_suc_nod
(
    d_ipc_cod,
    d_ast_nod,
    d_ast_nod,
    d_nod_cst
)

/* ------------------------------------------------------------------------- */
/* PREDICATE    : ast_ins_new(...)                                             */
/*                                                                             */
/* DESCRIPTION  : Insert <Node> after calculating heuristic underestimate      */
/*              : given <IpcScheme> and <HrtScheme> and <Cost> of <Node>       */
/*                                                                             */
/* PARAMETERS   : IpcScheme                                                    */
/*              : HrtScheme                                                    */
/*              : BtrSel                                                       */
/*              : G0                                                           */
/*              : Node                                                         */
/*              : Cost                                                         */
/*                                                                             */
/* CALLS        : ast_hrt_val(...)                                             */
/*              : ast_ket_int(...)                                             */
/*              : ast_wat_dog(...)                                             */
/*                                                                             */
/* DATABASE     : --                                                           */
/*                                                                             */
/* BACKTRACKING : determ                                                       */
/* ------------------------------------------------------------------------- */

determ
ast_ins_new
(
    d_ipc_cod,
    d_hrt_cod,
    bt_selector,
    d_fin_tim,
    d_ast_nod,
    d_nod_cst
)

/* ------------------------------------------------------------------------- */
/* PREDICATE    : ast_hrt_val(...)                                             */
/*                                                                             */
/* DESCRIPTION  : Bind <Heuristic> to underestimate determined from            */
/*              : given <IpcScheme> and <HrtScheme>                            */
/*                                                                             */
/* PARAMETERS   : HrtScheme                                                    */
/*              : IpcScheme                                                    */
/*              : Node                                                         */
/*              : Heuristic                                                    */
/*                                                                             */
/* CALLS        : ast_mpc_hrt(...)                                             */
/*              : ast_mia_hrt(...)                                             */
/*              : ast_mpc_hrt(...)                                             */
/*              : clc_max_val(...)                                             */
/*                                                                             */
/* DATABASE     : --                                                           */
/*                                                                             */
/* BACKTRACKING : determ                                                       */
/* ------------------------------------------------------------------------- */

determ
ast_hrt_val
(
    d_hrt_cod,
    d_ipc_cod,
    d_ast_nod,
    integer
)

/* ------------------------------------------------------------------------- */
/* PREDICATE    : ast_mpc_hrt(...)                                             */
/*                                                                             */
/* DESCRIPTION  : Bind <MpcHrt> to MPC heuristic underestimate for <Node>      */
/*                                                                             */
/* PARAMETERS   : Node                                                         */
/*              : MpcHrt                                                       */
/*                                                                             */
/* CALLS        : clc_max_val(...)                                             */
/*              : clc_seq_sch(...)                                             */
/*              : clc_sum_fin(...)                                             */
/*              : lst_num_elm(...)                                             */
/*                                                                             */
/* DATABASE     : --                                                           */
/*                                                                             */
/* BACKTRACKING : determ                                                       */
/* ------------------------------------------------------------------------- */

determ
```

```
ast_mpc_hrt
(
    d_ast_nod,
    d_nod_cst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_mia_hrt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <MiaHrt> to MIA heuristic underestimate for <Node>  */
/*                                                                          */
/* PARAMETERS    : IpcScheme                                                */
/*               : Node                                                     */
/*               : MiaHrt                                                   */
/*                                                                          */
/* CALLS         : ast_mia_tsk(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
ast_mia_hrt
(
    d_ipc_cod,
    d_ast_nod,
    d_nod_cst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_mia_tsk(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <MiaSum> to sum of MIA heuristic underestimates for */
/*               : remaining tasks in <TaskList> on active <ProcList>       */
/*                                                                          */
/* PARAMETERS    : IpcScheme                                                */
/*               : TaskList                                                 */
/*               : ProcList                                                 */
/*               : Fin                                                      */
/*               : OldMiaSum                                                */
/*               : MiaSum                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : ast_mia_pro(...)                                         */
/*               : clc_seq_sch(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
ast_mia_tsk
(
    d_ipc_cod,
    d_tsk_lst,
    d_pro_lst,
    d_fin_tim,
    d_nod_cst,
    d_nod_cst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_mia_pro(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <MiaHrt> to MIA underestimate for <Task> on <ProcList> */
/*                                                                          */
/* PARAMETERS    : IpcScheme                                                */
/*               : Task                                                     */
/*               : OldProcList                                              */
/*               : ProcList                                                 */
/*               : Fin                                                      */
/*               : OldMiaHrt                                                */
/*               : MiaHrt                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : clc_min_val(...)                                         */
/*               : idb_tsk_exe(...)                                         */
/*               : lst_del_elm(...)                                         */
/*               : sch_act_tsk(...)                                         */
/*               : sch_ins_ipc(...)                                         */
/*               : sch_non_cor(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
ast_mia_pro
(
    d_ipc_cod,
    d_tsk_dom,
    d_pro_lst,
    d_pro_lst,
    d_fin_tim,
    d_nod_cst,
    d_nod_cst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_mrc_hrt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <MrcHrt> to MRC heuristic underestimate for <Node>  */
/*                                                                          */
/* PARAMETERS    : Node                                                     */
/*               : MrcHrt                                                   */
/*                                                                          */
/* CALLS         : ast_mrc_pro(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
```

```
/*
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
ast_mrc_hrt
(
    d_ast_nod,
    d_fin_tim
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_mrc_pro(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <MrcHrt> to MRC underestimate for <ProcList>        */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : FinTime                                                  */
/*               : OldMrcHrt                                                */
/*               : MrcHrt                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : ast_mrc_rct(...)                                         */
/*               : clc_max_val(...)                                         */
/*               : idb_tsk_exe(...)                                         */
/*               : idb_tsk_lvl(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
ast_mrc_pro
(
    d_pro_lst,
    d_fin_tim,
    d_fin_tim,
    d_fin_tim
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_mst_rct(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <FinTime> and <Task> to most recent task scheduled in */
/*               : in <TaskList>                                            */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : Task                                                     */
/*               : FinTime                                                  */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
```

```
/* ------------------------------------------------------------------------ */

determ
ast_mst_rct
(
    d_tsk_lst,
    d_tsk_nam,
    d_tsk_exe
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_wat_dog(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Increment search tree watch dog in internal DB           */
/*                                                                          */
/* PARAMETERS    : --                                                       */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : retract( num_sea_nod(...))                               */
/*               : assert( num_sea_nod(...))                                */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
ast_wat_dog
(
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_gol_nod(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <Node> to goal node if found in search tree <BtrSel> */
/*                                                                          */
/* PARAMETERS    : BtrSel                                                   */
/*               : Node                                                     */
/*                                                                          */
/* CALLS         : ast_agn_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
ast_gol_nod
(
    bt_selector,
    d_ast_nod
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_agn_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for ast_gol_nod(...)                 */
```

```
/*                                                                        */
/* PARAMETERS    : BtrSel                                                 */
/*               : Key                                                    */
/*               : Ref                                                    */
/*               : Node                                                   */
/*                                                                        */
/* CALLS         : SELF                                                   */
/*                                                                        */
/* DATABASE      : --                                                     */
/*                                                                        */
/* BACKTRACKING  : determ                                                 */
/* ---------------------------------------------------------------------- */

determ
ast_agn_aux
(
    bt_selector,
    d_btr_key,
    d_edb_ref,
    d_ast_nod
)

/* ---------------------------------------------------------------------- */
/* PREDICATE     : ast_key_int(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Map <NodCst> to a Btree <BtrKey>, represented as a string */
/*                                                                        */
/* PARAMETERS    : BtrKey                                                 */
/*               : NodCst                                                 */
/*                                                                        */
/* CALLS         : --                                                     */
/*                                                                        */
/* DATABASE      : --                                                     */
/*                                                                        */
/* BACKTRACKING  : determ                                                 */
/* ---------------------------------------------------------------------- */

determ
ast_key_int
(
    d_btr_key,
    d_nod_cst
)

/* ********************************************************************** */
```

D.3.2 dcs_clc.prv

```
/* ********************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England           */
/* ********************************************************************** */

/* ********************************************************************** */
/* MODULE    : DCS_CLC.PRV                                                */
/*                                                                        */
/* NOTES     : Numerical predicates used by DCS modules                   */
/*                                                                        */
/* AUTHOR    : Ken N LINTON                                               */
/*                                                                        */
/* DATE      : 20th May, 1995                                             */
/*                                                                        */
/* VERSION   : 1.2                                                        */
/* ********************************************************************** */

/* ====================================================================== */
   predicates
/* ====================================================================== */

/* ---------------------------------------------------------------------- */
/* PREDICATE     : clc_ccp_aux(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Auxiliary predicate for clc_crt_pth(...)               */
/*                                                                        */
/* PARAMETERS    : CritPath1                                              */
/*               : CritPath2                                              */
/*                                                                        */
/* CALLS         : SELF                                                   */
/*               : clc_max_val(...)                                       */
/*                                                                        */
/* DATABASE      : retract( level(...))                                   */
/*                                                                        */
/* BACKTRACKING  : determ                                                 */
/* ---------------------------------------------------------------------- */

determ
clc_ccp_aux
(
    d_crt_pth,
    d_crt_pth
)

/* ---------------------------------------------------------------------- */
/* PREDICATE     : clc_cnn_aux(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Auxiliary predicate for clc_num_ncr(...)               */
/*                                                                        */
/* PARAMETERS    : NCRList                                                */
/*               : NumNCRs1                                               */
/*               : NumNCRs2                                               */
```

```
/*                                                                      */
/* CALLS        : SELF                                                 */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
clc_cnn_aux
(
    d_pro_lst,
    d_num_ncr,
    d_num_ncr
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : clc_chr_aux(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for clc_hdw_rsc(...)             */
/*                                                                      */
/* PARAMETERS   : ProcList                                             */
/*              : ProcRsrc1                                            */
/*              : ProcRsrc2                                            */
/*                                                                      */
/* CALLS        : SELF                                                 */
/*              : idb_pro_exe(...)                                     */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
clc_chr_aux
(
    d_pro_lst,
    d_pro_exe,
    d_pro_exe
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : clc_cpt_aux(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for clc_pro_tim(...)             */
/*                                                                      */
/* PARAMETERS   : TaskList                                             */
/*              : ProcTime1                                            */
/*              : ProcTime2                                            */
/*                                                                      */
/* CALLS        : SELF                                                 */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
clc_cpt_aux
(
    d_tsk_lst,
    d_tsk_exe,
    d_tsk_exe
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : clc_csf_aux(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for clc_sum_fin(...)             */
/*                                                                      */
/* PARAMETERS   : ProcList                                             */
/*              : FinTot1                                              */
/*              : FinTot2                                              */
/*                                                                      */
/* CALLS        : SELF                                                 */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
clc_csf_aux
(
    d_pro_lst,
    d_fin_tim,
    d_fin_tim
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : clc_css_sch(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for clc_seq_sch(...)             */
/*                                                                      */
/* PARAMETERS   : TaskList                                             */
/*              : OldSeqSch                                            */
/*              : SeqSch                                               */
/*                                                                      */
/* CALLS        : SELF                                                 */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
clc_css_aux
(
    d_tsk_lst,
    d_fin_tim,
    d_fin_tim
)
```

```
/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_ctg_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for clc_tsk_grn(...)                 */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : MinGran                                                  */
/*               : MaxGran                                                  */
/*               : AvgGran                                                  */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : clc_max_val(...)                                         */
/*               : clc_min_val(...)                                         */
/*               : idb_tsk_exe(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
clc_ctg_aux
(
    d_tsk_lst,
    d_tsk_exe,
    d_tsk_exe,
    d_tsk_exe,
    d_tsk_exe
)

/* ************************************************************************ */
```

D.3.3 dcs_cpm.prv

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_CPM.PRV                                                  */
/*                                                                          */
/* NOTES     : Core predicates for static-level DMC list scheduling         */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 20th May, 1995                                               */
/*                                                                          */
/* VERSION   : 4.5                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : cpm_cre_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create schedule of <TaskList> on <ProcList> and bind     */
/*               : <Schedule> to result with completed idle periods         */
/*                                                                          */
/* PARAMETERS    : IpcScheme                                                */
/*               : TaskList                                                 */
/*               : ProcList                                                 */
/*               : FinTime                                                  */
/*               : Schedule                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : cpm_sch_nxt(...)                                         */
/*               : sch_cmp_idl(...)                                         */
/*               : wri_alg_mon(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
cpm_cre_sch
(
    d_ipc_cod,
    d_tsk_lst,
    d_pro_lst,
    d_fin_tim,
    d_pro_lst
)
```

```
/* ------------------------------------------------------------------------ */
/* PREDICATE     : cpm_sch_nxt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <TaskList>, <ProcList> and <FinTime> after next step */
/*               : in statically labelled list scheduling discipline        */
/*                                                                          */
/* PARAMETERS    : IpcScheme                                                */
/*               : OldTaskList                                              */
/*               : OldProcList                                              */
/*               : OldFinTime                                               */
/*               : TaskList                                                 */
/*               : ProcList                                                 */
/*               : FinTime                                                  */
/*                                                                          */
/* CALLS         : idb_tsk_exe(...)                                         */
/*               : lst_del_ndt(...)                                         */
/*               : sch_act_tsk(...)                                         */
/*               : sch_ins_idl(...)                                         */
/*               : sch_ins_ipc(...)                                         */
/*               : sch_ins_tsk(...)                                         */
/*               : sch_non_cor(...)                                         */
/*               : sch_pre_rel(...)                                         */
/*               : sch_pro_rsc(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
cpm_sch_nxt
(
    d_ipc_cod,
    d_tsk_lst,
    d_pro_lst,
    d_fin_tim,
    d_tsk_lst,
    d_pro_lst,
    d_fin_tim
)

/* ************************************************************************ */
```

D.3.4 dcs_dyn.prv

```
/* ************************************************************************ */
/* © 1995     : School of Engineering, University of Durham, Durham, England */
/*            : Solid State Logic Ltd, Begbroke, Oxford, England            */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE     : DCS_DYN.PRV                                                 */
/*                                                                          */
/* NOTES      : Core predicates for dynamic-level DMC list scheduling       */
/*                                                                          */
/* AUTHOR     : Ken N LINTON                                                */
/*                                                                          */
/* DATE       : 28th May, 1995                                              */
/*                                                                          */
/* VERSION    : 7.1                                                         */
/* ************************************************************************ */

/* ======================================================================== */
   predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_ini_edb(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Initialise external DB to store dynamic level records    */
/*                                                                          */
/* PARAMETERS    : --                                                       */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
dyn_ini_edb
(
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_cre_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create schedule of <TaskList> on <ProcList> and bind     */
/*               : <Schedule> to result with completed idle periods         */
/*                                                                          */
/* PARAMETERS    : IpcScheme                                                */
/*               : DynScheme                                                */
/*               : TaskList                                                 */
/*               : ProcList                                                 */
/*               : FinTime                                                  */
/*               : Schedule                                                 */
```

```
/*                                                                      */
/* CALLS        : SELF                                                 */
/*              : dyn_sch_nxt(...)                                     */
/*              : sch_cmp_idl(...)                                     */
/*              : wri_alg_mon(...)                                     */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* ------------------------------------------------------------------- */

determ
dyn_cre_sch
(
    d_ipc_cod,
    d_dyn_cod,
    d_tsk_lst,
    d_pro_lst,
    d_fin_tim,
    d_pro_lst
)

/* ------------------------------------------------------------------- */
/* PREDICATE    : dyn_sch_stp(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Create <DynScheme> labelled scheduling step with <IPC> */
/*              : communications and bind <TL>, <PL> and <Fin> to result */
/*                                                                      */
/* PARAMETERS   : IpcScheme                                            */
/*              : DynScheme                                            */
/*              : OldTaskList                                          */
/*              : OldProcList                                          */
/*              : OldFinTime                                           */
/*              : TaskList                                             */
/*              : ProcList                                             */
/*              : FinTime                                              */
/*                                                                      */
/* CALLS        : SELF                                                 */
/*              : dyn_ins_dls(...)                                     */
/*              : dcs_sch_nxt(...)                                     */
/*              : sch_ins_idl(...)                                     */
/*                                                                      */
/* DATABASE     : bt_open(...)                                         */
/*              : bt_close(...)                                        */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* ------------------------------------------------------------------- */

determ
dyn_sch_stp
(
    d_ipc_cod,
    d_dyn_cod,
    d_tsk_lst,
    d_pro_lst,
    d_fin_tim,
    d_tsk_lst,
    d_pro_lst,
    d_fin_tim
)

/* ------------------------------------------------------------------- */
/* PREDICATE    : dyn_ins_dls(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Insert <DynScheme> labelled levels with <IPCScheme> using */
/*              : the external DB control structure <EdbCtl> for the tasks in */
/*              : <TaskList> on the processors <ProcList> at time <FinTime> */
/*                                                                      */
/* PARAMETERS   : DynScheme                                            */
/*              : IpcScheme                                            */
/*              : EdbCtl                                               */
/*              : TaskList                                             */
/*              : ProcList                                             */
/*              : FinTime                                              */
/*                                                                      */
/* CALLS        : dyn_clc_lvl(...)                                     */
/*              : dyn_gen_lvl(...)                                     */
/*              : dyn_key_int(...)                                     */
/*              : sch_act_tsk(...)                                     */
/*              : sch_ins_ipc(...)                                     */
/*              : sch_non_cor(...)                                     */
/*              : sch_pre_rel(...)                                     */
/*              : sch_pro_rsc(...)                                     */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* ------------------------------------------------------------------- */

determ
dyn_ins_dls
(
    d_dyn_cod,
    d_ipc_cod,
    d_edb_ctl,
    d_tsk_lst,
    d_pro_lst,
    d_fin_tim
)

/* ------------------------------------------------------------------- */
/* PREDICATE    : dyn_gen_lvl(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Generate task-processor <Task>-<Proc> pair for <DynScheme> */
/*              : dynamic labelling scheme                             */
/*                                                                      */
/* PARAMETERS   : DynScheme                                            */
/*              : OldTaskList                                          */
/*              : OldProcList                                          */
/*              : Task                                                 */
/*              : Proc                                                 */
/*              : NewTaskList                                          */
```

```
/*                : NewProcList                                          */
/*                                                                       */
/* CALLS          : lst_del_elm(...)                                     */
/*                : lst_del_ndt(...)                                     */
/*                                                                       */
/* DATABASE       : --                                                   */
/*                                                                       */
/* BACKTRACKING   : non-determ                                           */
/* --------------------------------------------------------------------- */

nondeterm
dyn_gen_lvl
(
    d_dyn_cod,
    d_tsk_lst,
    d_pro_lst,
    d_tsk_dom,
    d_pro_dom,
    d_tsk_lst,
    d_pro_lst
)

/* --------------------------------------------------------------------- */
/* PREDICATE      : dyn_clc_lvl(...)                                     */
/*                                                                       */
/* DESCRIPTION    : Bind <DynLvl> to dynamic level using <DynScheme> labelling */
/*                : scheme for <Task> on a processor finishing execution at */
/*                : <ProcFin> with tasks <TaskList> including IPC resources */
/*                                                                       */
/* PARAMETERS     : DynScheme                                            */
/*                : Task                                                 */
/*                : ProcFin                                              */
/*                : TaskList                                             */
/*                : DynLvl                                               */
/*                                                                       */
/* CALLS          : clc_max_val(...)                                     */
/*                : idb_tsk_lvl(...)                                     */
/*                                                                       */
/* DATABASE       : --                                                   */
/*                                                                       */
/* BACKTRACKING   : determ                                               */
/* --------------------------------------------------------------------- */

determ
dyn_clc_lvl
(
    d_dyn_cod,
    d_tsk_dom,
    d_tsk_exe,
    d_tsk_lst,
    d_dyn_lvl
)

/* --------------------------------------------------------------------- */
/* PREDICATE      : dyn_key_int(...)                                     */
/*                                                                       */
/* DESCRIPTION    : Bind <BtrKey> to 'Inverted' 10's complement representation */
/*                : of <DynLvl>: required since <DynLvl> may be negative */
/*                                                                       */
/* PARAMETERS     : BtrKey                                               */
/*                : DynLvl                                               */
/*                : OffSet                                               */
/*                                                                       */
/* CALLS          : --                                                   */
/*                                                                       */
/* DATABASE       : --                                                   */
/*                                                                       */
/* BACKTRACKING   : determ                                               */
/* --------------------------------------------------------------------- */

determ
dyn_key_int
(
    d_btr_key,
    d_dyn_lvl,
    d_off_set
)

/* --------------------------------------------------------------------- */
/* PREDICATE      : dyn_sch_nxt(...)                                     */
/*                                                                       */
/* DESCRIPTION    : Create next scheduling step with <IpcScheme>, indexing the */
/*                : dynamic-levels for task-processor pairs with <EdbCtl> and */
/*                : binding <TaskList>, <ProcList> and <Fin> to the result */
/*                                                                       */
/* PARAMETERS     : IpcScheme                                            */
/*                : EdbCtl                                               */
/*                : OldTaskList                                          */
/*                : OldProcList                                          */
/*                : TaskList                                             */
/*                : ProcList                                             */
/*                : OldFinTime                                           */
/*                : FinTime                                              */
/*                                                                       */
/* CALLS          : dyn_del_dls(...)                                     */
/*                : idb_tsk_exe(...)                                     */
/*                : lst_del_ndt(...)                                     */
/*                : sch_ins_ipc(...)                                     */
/*                : sch_ins_tsk(...)                                     */
/*                : sch_non_cor(...)                                     */
/*                                                                       */
/* DATABASE       : --                                                   */
/*                                                                       */
/* BACKTRACKING   : determ                                               */
/* --------------------------------------------------------------------- */

determ
dyn_sch_nxt
(
    d_ipc_cod,
    d_edb_ctl,
    d_tsk_lst,
```

```
    d_pro_lst,
    d_tsk_lst,
    d_pro_lst,
    d_fin_tim,
    d_fin_tim
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_del_dls(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Delete all task-processor dynamic levels from the external */
/*               : database chain and btree referenced by <EdbCtl>          */
/*                                                                          */
/* PARAMETERS    : EdbCtl                                                   */
/*                                                                          */
/* CALLS         : dyn_ddd_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
dyn_del_dls
(
    d_edb_ctl
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_ddd_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for dyn_del_dls(...)                 */
/*                                                                          */
/* PARAMETERS    : EdbCtl                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
dyn_ddd_aux
(
    d_edb_ctl
)

/* ************************************************************************ */
```

D.3.5 dcs_edb.prv

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_EDB.PRV                                                  */
/*                                                                          */
/* NOTES     : Display predicates for external database chains and btrees   */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 20th May, 1995                                               */
/*                                                                          */
/* VERSION   : 2.5                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : edb_edd_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for edb_dba_dsp(...)                 */
/*                                                                          */
/* PARAMETERS    : DbaSel                                                   */
/*               : OutDev                                                   */
/*                                                                          */
/* CALLS         : edb_wri_chn(...)                                         */
/*                                                                          */
/* DATABASE      : db_chains(...)                                           */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
edb_edd_aux
(
    db_selector
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : edb_wri_chn(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Write all chains held in the external DB <DbaSel>        */
/*                                                                          */
/* PARAMETERS    : DbaSel                                                   */
/*               : OutDev                                                   */
/*                                                                          */
/* CALLS         : edb_all_trm(...)                                         */
/*                                                                          */
```

```
/* DATABASE     : db_chains(...)                                          */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
edb_wri_chn
(
    db_selector
)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : edb_all_trm(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write all terms held in the chain <ChnNam> in <DbaSel>  */
/*                                                                        */
/* PARAMETERS   : DbaSel                                                  */
/*              : OutDev                                                  */
/*                                                                        */
/* CALLS        : edb_wri_trm(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
edb_all_trm
(
    db_selector,
    d_sym_fil
)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : edb_wri_trm(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write external DB term <ThisTerm> with reference <ThisRef> */
/*                                                                        */
/* PARAMETERS   : ThisTerm                                                */
/*              : ThisRef                                                 */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
edb_wri_trm
(
    d_edb_dom,
    d_edb_ref
)

/* ---------------------------------------------------------------------- */
```

```
/* PREDICATE    : edb_ebd_aux(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Auxiliary predicate for edb_btr_dsp(...)                */
/*                                                                        */
/* PARAMETERS   : DbaSel                                                  */
/*                                                                        */
/* CALLS        : edb_wri_btr(...)                                        */
/*                                                                        */
/* DATABASE     : db_btrees(...)                                          */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
edb_ebd_aux
(
    db_selector
)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : edb_ebs_aux(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Auxiliary predicate for edb_btr_sts(...)                */
/*                                                                        */
/* PARAMETERS   : DbaSel                                                  */
/*              : BtrNam                                                  */
/*              : OutDev                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : bt_open(...)                                            */
/*              : bt_statistics(...)                                      */
/*              : bt_close(...)                                           */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
edb_ebs_aux
(
    db_selector,
    d_btr_nam,
    file
)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : edb_wri_btr(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write contents of all Btrees in the external DB <DbaSel> */
/*                                                                        */
/* PARAMETERS   : DbaSel                                                  */
/*                                                                        */
/* CALLS        : edb_all_key(...)                                        */
/*                                                                        */
/* DATABASE     : db_btrees(...)                                          */
/*                                                                        */
```

```
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
edb_wri_btr
(
    db_selector
)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : edb_all_key(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Write contents of Btree <BtrNam> in external DB <DbaSel>  */
/*                                                                          */
/* PARAMETERS   : DbaSel                                                    */
/*              : BtrNam                                                    */
/*                                                                          */
/* CALLS        : edb_btr_chk(...)                                          */
/*              : edb_wri_key(...)                                          */
/*                                                                          */
/* DATABASE     : bt_open(...)                                              */
/*              : bt_close(...)                                             */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
edb_all_key
(
    db_selector,
    d_btr_nam
)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : edb_wri_key(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Write term corresponding to <ThisRef> in the Btree <BtrSel> */
/*              : held in the external DB <DbaSel>                          */
/*                                                                          */
/* PARAMETERS   : DbaSel                                                    */
/*              : BtrSel                                                    */
/*              : ThisRef                                                   */
/*                                                                          */
/* CALLS        : edb_wri_key(...)                                          */
/*              : edb_wri_trm(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
edb_wri_key
(
    db_selector,
    bt_selector,
    d_edb_ref
)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : edb_btr_chk(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Check if there are any keys in the Btree <BtrSel> in the  */
/*              : external DB <DbaSel>. If so bind <FirstRef> to the first  */
/*              : reference number in the DB                                */
/*                                                                          */
/* PARAMETERS   : DbaSel                                                    */
/*              : BtrSel                                                    */
/*              : FirstRef                                                  */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : bt_close(...)                                             */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
edb_btr_chk
(
    db_selector,
    bt_selector,
    d_edb_ref
)

/* ************************************************************************ */
```

D.3.6 dcs_inf.prv

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_INF.PRV                                                    */
/*                                                                            */
/* NOTES     : User interface and top-level predicates for DCS application    */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 20th May, 1995                                                 */
/*                                                                            */
/* VERSION   : 4.5                                                            */
/* ************************************************************************** */

/* ========================================================================== */
   predicates
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : inf_run_dcs(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Run DCS digital console scheduler, first clearing the      */
/*               : internal database and enabling ctrl-break interrupts       */
/*                                                                            */
/* PARAMETERS    : --                                                         */
/*                                                                            */
/* CALLS         : inf_get_inp(...)                                           */
/*               : inf_mon_stt(...)                                           */
/*               : inf_mon_end(...)                                           */
/*               : inf_sch_eng(...)                                           */
/*               : inf_sho_sch(...)                                           */
/*               : inf_sav_sch(...)                                           */
/*                                                                            */
/* DATABASE      : retractall(...)                                            */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
inf_run_dcs
(
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : inf_mon_stt(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Create monitor window & bind <OldDev> to old output device */
/*                                                                            */
/* PARAMETERS    : OldDev                                                     */
/*                                                                            */
/* CALLS         : --                                                         */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
inf_mon_stt
(
    file
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : inf_mon_end(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Return to <OldDev> output device and remove monitor window */
/*                                                                            */
/* PARAMETERS    : OldDev                                                     */
/*                                                                            */
/* CALLS         : --                                                         */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
inf_mon_end
(
    file
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : inf_sch_eng(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Scheduling engine to populate internal database, create    */
/*               : task, precedence and processor lists and run algorithm     */
/*               : to schedule <Taskforce> on <Hardware> with <AlgCtl> control */
/*                                                                            */
/* PARAMETERS    : Taskforce                                                  */
/*               : Hardware                                                   */
/*               : AlgCtl                                                     */
/*                                                                            */
/* CALLS         : inf_ctp_tst(...)                                           */
/*               : inf_fit_tst(...)                                           */
/*               : inf_grn_tst(...)                                           */
/*               : inf_tst_tst(...)                                           */
/*               : inf_bgn_sch(...)                                           */
/*               : utl_cre_pre(...)                                           */
/*               : utl_cre_pro(...)                                           */
/*               : utl_cre_tsk(...)                                           */
/*               : wri_inp_prm(...)                                           */
/*               : wri_mtp_sch(...)                                           */
/*               : wri_pre_stt(...)                                           */
```

```
/*                : wri_pre_chk(...)                                        */
/*                : wri_pst_stt(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

determ
inf_sch_eng
(
    d_fil_nam,
    d_fil_nam,
    d_alg_ctl
)

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_opn_sch(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Open the file <SchNam> in the directory <SchDir> on the */
/*                : the symbolic file <SymFil>                              */
/*                                                                          */
/* PARAMETERS     : SchDir                                                  */
/*                : SchNam                                                  */
/*                : SymFil                                                  */
/*                                                                          */
/* CALLS          : --                                                      */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

determ
inf_opn_sch
(
    d_dos_dir,
    d_fil_nam,
    file
)

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_cre_lst(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Consult the <Taskforce> and <Hardware> descriptions and */
/*                : populate the <TaskList>, <PrecList> and <ProcList> lists */
/*                                                                          */
/* PARAMETERS     : Taskforce                                               */
/*                : Hardware                                                */
/*                : TaskList                                                */
/*                : PrecList                                                */
/*                : ProcList                                                */
/*                                                                          */
/* CALLS          : utl_cre_pre(...)                                        */
/*                : utl_cre_pro(...)                                        */
/*                : utl_cre_tsk(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

determ
inf_cre_lst
(
    d_fil_nam,
    d_fil_nam,
    d_tsk_lst,
    d_pre_lst,
    d_pro_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_pre_tst(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Perform pre-schedule tests and populate <TestList>      */
/*                                                                          */
/* PARAMETERS     : TaskList                                                */
/*                : PrecList                                                */
/*                : ProcList                                                */
/*                : TestList                                                */
/*                                                                          */
/* CALLS          : --                                                      */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

determ
inf_pre_tst
(
    d_tsk_lst,
    d_pre_lst,
    d_pro_lst,
    d_sym_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_bgn_sch(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Begin scheduling algorithm, recording system clock      */
/*                : immediately before and after                            */
/*                                                                          */
/* PARAMETERS     : AlgCtl                                                  */
/*                : TaskList                                                */
/*                : PrecList                                                */
/*                : ProcList                                                */
/*                : Schedule                                                */
/*                                                                          */
/* CALLS          : ast_run_alg(...)                                        */
/*                : cpm_run_alg(...)                                        */
```

```
/*                : dyn_run_alg(...)                                        */
/*                : idb_sys_tim(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

determ
inf_bgn_sch
(
    d_alg_ctl,
    d_tsk_lst,
    d_pre_lst,
    d_pro_lst,
    d_pro_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_get_inp(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Get taskforce file <TkfFil>, hardware file <HdwFil> and */
/*                : algorithm control parameters <AlgCtl> from user         */
/*                                                                          */
/* PARAMETERS     : TkfFil                                                  */
/*                : HdwFil                                                  */
/*                : AlgCtl                                                  */
/*                                                                          */
/* CALLS          : inf_nul_inp(...)                                        */
/*                : inf_get_tkf(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

determ
inf_get_inp
(
    d_fil_nam,
    d_fil_nam,
    d_alg_ctl
)

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_get_tkf(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Using directory <Dir> and filename extension <Ext>, get */
/*                : taskforce filname <Tkf> and then satisfy inf_get_hdw(...) */
/*                                                                          */
/* PARAMETERS     : Dir                                                     */
/*                : Ext                                                     */
/*                : OldInp                                                  */
/*                : Inp                                                     */
/*                : Ctl                                                     */
/*                                                                          */
/* CALLS          : inf_get_hdw(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

determ
inf_get_tkf
(
    d_dos_dir,
    d_dos_ext,
    d_alg_inp,
    d_alg_inp,
    d_alg_ctl
)

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_get_hdw(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Using directory <Dir> and filename extension <Ext>, get */
/*                : hardware filname <Tkf> and then satisfy inf_get_alg(...) */
/*                                                                          */
/* PARAMETERS     : Dir                                                     */
/*                : Ext                                                     */
/*                : OldInp                                                  */
/*                : Inp                                                     */
/*                : Ctl                                                     */
/*                                                                          */
/* CALLS          : inf_get_alg(...)                                        */
/*                : inf_get_tkf(...)                                        */
/*                : inf_nul_ctl(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

determ
inf_get_hdw
(
    d_dos_dir,
    d_dos_ext,
    d_alg_inp,
    d_alg_inp,
    d_alg_ctl
)

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_nul_inp(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Bind <AlgInp> to null algorithm input domain            */
/*                                                                          */
/* PARAMETERS     : AlgInp                                                  */
/*                                                                          */
/* CALLS          : --                                                      */
```

```
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
inf_nul_inp
(
    d_alg_inp
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : inf_nul_ctl(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Bind <AlgCtl> to null algorithm control domain       */
/*                                                                      */
/* PARAMETERS   : AlgCtl                                               */
/*                                                                      */
/* CALLS        : --                                                   */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
inf_nul_ctl
(
    d_alg_ctl
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : inf_mnu_rtn(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Return item <ItmCod> from the user interface menu <MnuCod> */
/*                                                                      */
/* PARAMETERS   : MnuCod                                               */
/*              : ItmCod                                               */
/*                                                                      */
/* CALLS        : inf_mnu_chk(...)                                     */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
inf_mnu_rtn
(
    d_mnu_cod,
    symbol
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : inf_mnu_chk(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Check <UsrChr> is a valid <ItmCod> for the menu <MnuCod> */
/*                                                                      */
/* PARAMETERS   : MnuCod                                               */
/*              : UsrChr                                               */
/*              : ItmCod                                               */
/*                                                                      */
/* CALLS        : inf_mnu_rtn(...)                                     */
/*              : inf_vld_alg(...)                                     */
/*              : inf_vld_ipc(...)                                     */
/*              : inf_vld_lbl(...)                                     */
/*              : inf_vld_dyn(...)                                     */
/*              : inf_vld_hrt(...)                                     */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
inf_mnu_chk
(
    d_mnu_cod,
    char,
    symbol
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : inf_get_alg(...)                                     */
/*                                                                      */
/* DESCRIPTION  : Get scheduling algorithm and then satisfy inf_get_ipc(...) */
/*                                                                      */
/* PARAMETERS   : OldCtl                                               */
/*              : Ctl                                                  */
/*              : Dir                                                  */
/*              : Ext                                                  */
/*              : OldInp                                               */
/*              : Inp                                                  */
/*                                                                      */
/* CALLS        : inf_get_hdw(...)                                     */
/*              : inf_get_ipc(...)                                     */
/*              : inf_mnu_rtn(...)                                     */
/*              : trl_alg_sym(...)                                     */
/*                                                                      */
/* DATABASE     : --                                                   */
/*                                                                      */
/* BACKTRACKING : determ                                               */
/* -------------------------------------------------------------------- */

determ
inf_get_alg
(
    d_alg_ctl,
    d_alg_ctl,
    d_dos_dir,
    d_dos_ext,
```

```
    d_alg_inp,
    d_alg_inp
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_get_ipc(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Get IPC scheme and then satisfy inf_get_lbl(...)        */
/*                                                                          */
/* PARAMETERS    : OldCtl                                                   */
/*               : Ctl                                                      */
/*               : Dir                                                      */
/*               : Ext                                                      */
/*               : OldInp                                                   */
/*               : Inp                                                      */
/*                                                                          */
/* CALLS         : inf_mnu_rtn(...)                                         */
/*               : inf_get_lbl(...)                                         */
/*               : inf_get_alg(...)                                         */
/*               : trl_ipc_sym(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_get_ipc
(
    d_alg_ctl,
    d_alg_ctl,
    d_dos_dir,
    d_dos_ext,
    d_alg_inp,
    d_alg_inp
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_get_lbl(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Get static labelling and then satisfy inf_get_dyn(...)   */
/*                                                                          */
/* PARAMETERS    : OldCtl                                                   */
/*               : Ctl                                                      */
/*               : Dir                                                      */
/*               : Ext                                                      */
/*               : OldInp                                                   */
/*               : Inp                                                      */
/*                                                                          */
/* CALLS         : inf_get_dyn(...)                                         */
/*               : inf_get_hrt(...)                                         */
/*               : inf_get_ipc(...)                                         */
/*               : inf_mnu_rtn(...)                                         */
/*               : trl_alg_sym(...)                                         */
/*               : trl_lbl_sym(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_get_lbl
(
    d_alg_ctl,
    d_alg_ctl,
    d_dos_dir,
    d_dos_ext,
    d_alg_inp,
    d_alg_inp
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_get_dyn(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Get dynamic labelling and then satisfy inf_get_hrt(...)  */
/*                                                                          */
/* PARAMETERS    : OldCtl                                                   */
/*               : Ctl                                                      */
/*               : Dir                                                      */
/*               : Ext                                                      */
/*               : OldInp                                                   */
/*               : Inp                                                      */
/*                                                                          */
/* CALLS         : inf_get_hrt(...)                                         */
/*               : inf_get_lbl(...)                                         */
/*               : inf_mnu_rtn(...)                                         */
/*               : trl_alg_sym(...)                                         */
/*               : trl_dyn_sym(...)                                         */
/*               : trl_ipc_sym(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_get_dyn
(
    d_alg_ctl,
    d_alg_ctl,
    d_dos_dir,
    d_dos_ext,
    d_alg_inp,
    d_alg_inp
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_get_hrt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Get heuristic underestimate and then bind <Inp> and <Ctl> */
/*                                                                          */
```

```
/* PARAMETERS   : OldCtl                                                  */
/*              : Ctl                                                     */
/*              : Dir                                                     */
/*              : Ext                                                     */
/*              : OldInp                                                  */
/*              : Inp                                                     */
/*                                                                        */
/* CALLS        : inf_get_ipc(...)                                        */
/*              : inf_mnu_rtn(...)                                        */
/*              : trl_alg_sym(...)                                        */
/*              : trl_hrt_sym(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
inf_get_hrt
(
    d_alg_ctl,
    d_alg_ctl,
    d_dos_dir,
    d_dos_ext,
    d_alg_inp,
    d_alg_inp
)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : inf_vld_alg(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Bind <AlgCod> to scheduling algorithm given by <MnuChr> */
/*                                                                        */
/* PARAMETERS   : MnuChr                                                  */
/*              : AlgCod                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
inf_vld_alg
(
    char,
    d_alg_cod
)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : inf_vld_ipc(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Bind <IpcCod> to IPC scheme given by <MnuChr>           */
/*                                                                        */
/* PARAMETERS   : MnuChr                                                  */
/*              : IpcCod                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
inf_vld_ipc
(
    char,
    d_ipc_cod
)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : inf_vld_lbl(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Bind <LblCod> to static labelling scheme given by <MnuChr> */
/*                                                                        */
/* PARAMETERS   : MnuChr                                                  */
/*              : LblCod                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
inf_vld_lbl
(
    char,
    d_lbl_cod
)

/* ---------------------------------------------------------------------- */
/* PREDICATE    : inf_vld_dyn(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Bind <DynCod> to dynamic labelling scheme given by <MnuChr> */
/*                                                                        */
/* PARAMETERS   : MnuChr                                                  */
/*              : DynCod                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

determ
inf_vld_dyn
(
```

```
    char,
    d_dyn_cod
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_vld_hrt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <HrtCod> to heuristic underestimate given by <MnuChr> */
/*                                                                          */
/* PARAMETERS    : MnuChr                                                   */
/*               : HrtCod                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_vld_hrt
(
    char,
    d_hrt_cod
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_ini_wat(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Initialise search-tree watch-dogs in internal DB         */
/*                                                                          */
/* PARAMETERS    : AlgCtl                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : assert( num_sea_nod(...))                                */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_ini_wat
(
    d_alg_ctl
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_tst_tst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Fail if any element of <TstLst> corresponds to c_tst_fai */
/*                                                                          */
/* PARAMETERS    : TstLst                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_tst_tst
(
    d_sym_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_ctp_tst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <PassFail> to c_tst_fai if critical path will not fit */
/*               : on any processor, or c_tst_pas otherwise                 */
/*                                                                          */
/* PARAMETERS    : PrecList                                                 */
/*               : ProcList                                                 */
/*               : PassFail                                                 */
/*                                                                          */
/* CALLS         : clc_crt_pth(...)                                         */
/*               : inf_dct_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_ctp_tst
(
    d_pre_lst,
    d_pro_lst,
    d_tst_cod
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_dct_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for inf_ctp_tst(...)                 */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : CritPath                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : idb_pro_exe(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_dct_aux
(
    d_pro_lst,
```

```
    d_crt_pth
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_fit_tst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <PassFail> to c_tst_fai if taskforce <TaskList> will */
/*               : not fit on processing resource <ProcList>, ignoring      */
/*               : precedence and IPC, or c_tst_pas otherwise               */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : ProcList                                                 */
/*               : PassFail                                                 */
/*                                                                          */
/* CALLS         : clc_seq_sch(...)                                         */
/*               : clc_hdw_rsc(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_fit_tst
(
    d_tsk_lst,
    d_pro_lst,
    d_tst_cod
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_grn_tst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <PassFail> to c_tst_fai if any task from <TaskList> */
/*               : will not fit on any processor from <ProcList>            */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : ProcList                                                 */
/*               : PassFail                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : inf_dgt_aux(...)                                         */
/*               : inf_grn_tst(...)                                         */
/*               : idb_tsk_exe(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_grn_tst
(
    d_tsk_lst,
    d_pro_lst,
    d_tst_cod
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_dgt_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for inf_grn_tst(...)                 */
/*                                                                          */
/* PARAMETERS    : TaskTime                                                 */
/*               : ProcList                                                 */
/*                                                                          */
/* CALLS         : idb_pro_exe(...)                                         */
/*               : lst_mem_ndt(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_dgt_aux
(
    d_tsk_exe,
    d_pro_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_sho_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Display DCS output on screen and bind <RtrnStat> to status */
/*               : returned by edit(...)                                    */
/*                                                                          */
/* PARAMETERS    : RtrnStat                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
inf_sho_sch
(
    d_rtn_stt
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_sav_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Prompt user to save schedule and remove all windows      */
/*                                                                          */
/* PARAMETERS    : RtrnStat                                                 */
/*                                                                          */
/* CALLS         : inf_get_fil(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
```

```
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

determ
inf_sav_sch
(
    d_rtn_stt
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : inf_get_fil(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Get filename in which to store current DCS output     */
/*                                                                      */
/* PARAMETERS   : --                                                    */
/*                                                                      */
/* CALLS        : inf_vld_fil(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

determ
inf_get_fil
(
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : inf_vld_fil(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Ensure output filename <SchFil> does not aleady exist */
/*                                                                      */
/* PARAMETERS   : SchFil                                                */
/*                                                                      */
/* CALLS        : inf_ren_fil(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

determ
inf_vld_fil
(
    d_fil_nam
)

/* -------------------------------------------------------------------- */
/* PREDICATE    : inf_ren_fil(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Rename temporary DCS output 'temp.sch' to <SchFil>    */
/*                                                                      */
/* PARAMETERS   : UsrChr                                                */
/*              : SchFil                                                */
/*                                                                      */
/* CALLS        : inf_get_fil(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

determ
inf_ren_fil
(
    char,
    d_fil_nam
)

/* ******************************************************************** */
```

D.3.7 dcs_lst.prv

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_LST.PRV                                                  */
/*                                                                          */
/* NOTES     : General-purpose list-handling predicates                     */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 10th January, 1995                                           */
/*                                                                          */
/* VERSION   : 3.8                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_num_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for lst_num_elm(...)                 */
/*                                                                          */
/* PARAMETERS    : List                                                     */
/*               : NumElms1                                                 */
/*               : NumElms2                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
lst_num_aux
(
    d_tsk_lst,
    d_num_elm,
    d_num_elm
)

determ
lst_num_aux
(
    d_pre_lst,
    d_num_elm,
    d_num_elm
)

determ
lst_num_aux
(
    d_pro_lst,
    d_num_elm,
    d_num_elm
)

determ
lst_num_aux
(
    d_int_lst,
    d_num_elm,
    d_num_elm
)

/* ************************************************************************ */
```

D.3.8 dcs_sch.prv

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England              */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_SCH.PRV                                                  */
/*                                                                          */
/* NOTES     : Application-specific scheduling predicates for DCS engine    */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 15th June, 1995                                              */
/*                                                                          */
/* VERSION   : 7.5                                                          */
/* ************************************************************************ */

/* ======================================================================== */
   predicates
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_ipc_non(...)                                         */
/*                                                                          */
/* DESCRIPTION   : For non-overlapping IPC on a single communication channel, */
/*               : bind <IpcStt> and <IpcEnd> to start and end times of IPC */
/*               : which would have spanned <OldIpcStt> to <OldIpcEnd>      */
/*                                                                          */
/* PARAMETERS    : OldIpcStt                                                */
/*               : OldIpcEnd                                                */
/*               : ProcList                                                 */
/*               : IpcStt                                                   */
/*               : IpcEnd                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : sch_ipc_chk(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
sch_ipc_non
(
    d_fin_tim,
    d_fin_tim,
    d_pro_lst,
    d_tsk_exe,
    d_tsk_exe
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_snc_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for sch_non_cor(...)                 */
/*                                                                          */
/* PARAMETERS    : Task                                                     */
/*               : ActiveProcs                                              */
/*               : OldNCRList                                               */
/*               : NCRList                                                  */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : sch_ncr_tsk(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
sch_snc_aux
(
    d_tsk_dom,
    d_pro_lst,
    d_pro_lst,
    d_pro_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_ncr_tsk(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <NCRTasks> to all tasks in <TaskList> which are     */
/*               : non-coresident predecessors of <Task>                    */
/*                                                                          */
/* PARAMETERS    : Task                                                     */
/*               : TaskList                                                 */
/*               : OldNCRTasks                                              */
/*               : NCRTasks                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : idb_tsk_com(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
sch_ncr_tsk
(
    d_tsk_nam,
    d_tsk_lst,
    d_tsk_lst,
    d_tsk_lst
)

/* ------------------------------------------------------------------------ */
```

```
/* PREDICATE    : sch_ipc_src(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Schedule IPC from <IpcStt> to <IpcEnd> on the source      */
/*              : processor <Processor>, binding <ProcList> and <FinTime>   */
/*              : to the resulting schedule and finishing time              */
/*                                                                          */
/* PARAMETERS   : IpcStt                                                    */
/*              : IpcEnd                                                    */
/*              : Processor                                                 */
/*              : OldProcList                                               */
/*              : ProcList                                                  */
/*              : OldFinTime                                                */
/*              : FinTime                                                   */
/*                                                                          */
/* CALLS        : sch_ins_tsk(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
sch_ipc_src
(
    d_tsk_exe,
    d_tsk_exe,
    d_pro_dom,
    d_pro_lst,
    d_pro_lst,
    d_fin_tim,
    d_fin_tim
)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : sch_ipc_dst(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Schedule IPC from <IpcStt> to <IpcEnd> on the destination */
/*              : processor, binding <TaskList> to the result               */
/*                                                                          */
/* PARAMETERS   : IpcStt                                                    */
/*              : IpcEnd                                                    */
/*              : OldTaskList                                               */
/*              : TaskList                                                  */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
sch_ipc_dst
(
    d_tsk_exe,
    d_tsk_exe,
    d_tsk_lst,
    d_tsk_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE    : sch_ipc_chk(...)                                          */
/*                                                                          */
/* DESCRIPTION  : With processor state <ProList>, check non-overlapping IPC */
/*              : from <IpcStt> to <IpcEnd> can be reserved on a single link */
/*                                                                          */
/* PARAMETERS   : IpcStt                                                    */
/*              : IpcEnd                                                    */
/*              : ProcList                                                  */
/*                                                                          */
/* CALLS        : SELF                                                      */
/*              : lst_mem_ndt(...)                                          */
/*              : lst_nxt_elm(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
sch_ipc_chk
(
    d_fin_tim,
    d_fin_tim,
    d_pro_lst
)

/* ************************************************************************ */
```

D.3.9 dcs_utl.prv

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_UTL.PRV                                                    */
/*                                                                            */
/* NOTES     : Utility predicates for DCS scheduling engine                   */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 14th June, 1995                                                */
/*                                                                            */
/* VERSION   : 7.2                                                            */
/* ************************************************************************** */

/* ========================================================================== */
   predicates
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : utl_spl_tkl(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Classic quick-sort split predicate for utl_srt_prl(...)    */
/*                                                                            */
/* PARAMETERS    : Task                                                       */
/*               : TaskList1                                                  */
/*               : TaskList2                                                  */
/*               : TaskList                                                   */
/*                                                                            */
/* CALLS         : SELF                                                       */
/*               : utl_ord_tsk(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
utl_spl_tkl
(
    d_tsk_dom,
    d_tsk_lst,
    d_tsk_lst,
    d_tsk_lst
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : utl_ord_tsk(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Order <Task2>, <Task1> succeeds if <Task1> has higher level */
/*                                                                            */
/* PARAMETERS    : Task2                                                      */
/*               : Task1                                                      */
/*                                                                            */
/* CALLS         : idb_tsk_lvl(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
utl_ord_tsk
(
    d_tsk_nam,
    d_tsk_nam
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : utl_ini_lvl(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Initialise <LabelScheme> levels for tasks in internal DB   */
/*                                                                            */
/* PARAMETERS    : LabelScheme                                                */
/*                                                                            */
/* CALLS         : utl_chk_lst(...)                                           */
/*                                                                            */
/* DATABASE      : assertz( task(...))                                        */
/*               : retract( task(...))                                        */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
utl_ini_lvl
(
    d_lbl_cod
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : utl_chk_lst(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Label <TaskName> of type <TaskType> using <LabelScheme> and */
/*               : check if this task is last task in internal DB             */
/*                                                                            */
/* PARAMETERS    : LabelScheme                                                */
/*               : TaskName                                                   */
/*               : TaskType                                                   */
/*                                                                            */
/* CALLS         : --                                                         */
/*                                                                            */
/* DATABASE      : assertz( task(...))                                        */
/*               : assertz( level(...))                                       */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */
```

```
determ
utl_chk_lst
(
    d_lbl_cod,
    d_tsk_nam,
    d_tsk_typ
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_lbl_all(...)                                         */
/*                                                                          */
/* DESCRIPTION   : <LabelScheme> label all predecessors of <Task>, with level */
/*               : <Level> level, the list of precedence <PrecList>         */
/*                                                                          */
/* PARAMETERS    : LabelScheme                                              */
/*               : Task                                                     */
/*               : Level                                                    */
/*               : PrecList                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_del_ndt(...)                                         */
/*               : utl_est_exe(...)                                         */
/*               : utl_lvl_col(...)                                         */
/*               : utl_upd_lvl(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : non-determ                                               */
/* ------------------------------------------------------------------------ */

nondeterm
utl_lbl_all
(
    d_lbl_cod,
    d_tsk_nam,
    d_cpm_lvl,
    d_pre_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_lvl_col(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Using <LabelScheme>, determine if <Task2> precedes <Task1> */
/*               : in the precedence relation <Prec>                        */
/*                                                                          */
/* PARAMETERS    : LabelScheme                                              */
/*               : Prec                                                     */
/*               : Task1                                                    */
/*               : Task2                                                    */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_lvl_col
(
    d_lbl_cod,
    d_pre_rel,
    d_tsk_nam,
    d_tsk_nam
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_est_exe(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Using <LabelScheme>, estimate execution time <ExecTime> of */
/*               : of task with <TaskName>                                  */
/*                                                                          */
/* PARAMETERS    : LabelScheme                                              */
/*               : TaskName                                                 */
/*               : ExecTime                                                 */
/*                                                                          */
/* CALLS         : idb_tsk_exe(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_est_exe
(
    d_lbl_cod,
    d_tsk_nam,
    d_tsk_exe
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_upd_lvl(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Update level of <Task> in internal DB if <NewLevel> is   */
/*               : is greater than current task level                       */
/*                                                                          */
/* PARAMETERS    : Task                                                     */
/*               : NewLevel                                                 */
/*                                                                          */
/* CALLS         : utl_uul_aux(...)                                         */
/*                                                                          */
/* DATABASE      : retract( level(...))                                     */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_upd_lvl
(
    d_tsk_nam,
```

```
    d_cpm_lvl
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_uul_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for utl_upd_lvl(...)                 */
/*                                                                          */
/* PARAMETERS    : Task                                                     */
/*               : TaskLevel                                                */
/*               : NewLevel                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : assertz( level(...))                                     */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_uul_aux
(
    d_tsk_nam,
    d_cpm_lvl,
    d_cpm_lvl
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_tsk_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for utl_cre_tsk(...)                 */
/*                                                                          */
/* PARAMETERS    : OldTaskList                                              */
/*               : TaskList                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : utl_nxt_tsk(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_tsk_aux
(
    d_tsk_lst,
    d_tsk_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_nxt_tsk(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Get name <TaskName> and type <TaskType> of next task from */
/*               : taskforce held in internal DB                            */
/*                                                                          */
/* PARAMETERS    : TaskName                                                 */
/*               : TaskType                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : retract( task(...))                                      */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_nxt_tsk
(
    d_tsk_nam,
    d_tsk_typ
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_pre_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for utl_cre_pre(...)                 */
/*                                                                          */
/* PARAMETERS    : OldPrecList                                              */
/*               : PrecList                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : utl_fnt_str(...)                                         */
/*               : utl_net_out(...)                                         */
/*               : utl_nxt_net(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_pre_aux
(
    d_pre_lst,
    d_pre_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_nxt_net(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Get input <NetInput> and list of outputs <NetOutputs> for */
/*               : next taskforce net from netlist held in internal DB      */
/*                                                                          */
/* PARAMETERS    : NetInput                                                 */
/*               : NetOutPut                                                */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : retract( net(...))                                       */
/*               : assertz( net(...))                                       */
/*                                                                          */
```

```
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

determ
utl_nxt_net
(
    string,
    d_str_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_net_out(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create precedence relations <Precs> for all tasks in     */
/*               : <OutTasks> which are fed by <InTask>                     */
/*                                                                          */
/* PARAMETERS    : InTask                                                   */
/*               : OutTasks                                                 */
/*               : OldPrecs                                                 */
/*               : Precs                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : utl_fnt_str(...)                                         */
/*                                                                          */
/* DATABASE      : assert( prec(...))                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_net_out
(
    d_tsk_nam,
    d_str_lst,
    d_pre_lst,
    d_pre_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_fnt_str(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <OutStr> to the first part of <InStr> before any one */
/*               : of the delimiters in <Delimiters>. <RestStr> is bound    */
/*               : to the part of <InStr> following the delimiter           */
/*                                                                          */
/* PARAMETERS    : InStr                                                    */
/*               : Delimiters                                               */
/*               : ConCatStr                                                */
/*               : OutStr                                                   */
/*               : RestStr                                                  */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_fnt_str
(
    string,
    d_str_lst,
    string,
    string,
    string
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_pro_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for utl_cre_pro(...)                 */
/*                                                                          */
/* PARAMETERS    : OldProcList                                              */
/*               : ProcList                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : utl_nxt_pro(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_pro_aux
(
    d_pro_lst,
    d_pro_lst
)

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_nxt_pro(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Get name <ProcName> and type <ProcType> of next processor */
/*               : from hardware description held in internal DB            */
/*                                                                          */
/* PARAMETERS    : ProcName                                                 */
/*               : ProcType                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : retract( proc(...))                                      */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

determ
utl_nxt_pro
(
    d_pro_nam,
    d_pro_typ
```

```
)

/* ************************************************************************** */
```

D.3.10 dcs_wri.prv

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_WRI.PRV                                                    */
/*                                                                            */
/* NOTES     : Display predicates for DCS internal lists and other structures */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 1st March, 1995                                                */
/*                                                                            */
/* VERSION   : 2.1                                                            */
/* ************************************************************************** */

/* ========================================================================== */
   predicates
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : wri_wdl_aux(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Auxiliary predicate for wri_dlm_lin(...)                   */
/*                                                                            */
/* PARAMETERS    : Chr                                                        */
/*               : Cols                                                       */
/*                                                                            */
/* CALLS         : SELF                                                       */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

determ
wri_wdl_aux
(
    char,
    integer
)

/* -------------------------------------------------------------------------- */
/* PREDICATE     : wri_wpu_aux(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Auxiliary predicate for wri_pro_utl(...)                   */
/*                                                                            */
/* PARAMETERS    : UtilSym                                                    */
/*               : Schedule                                                   */
/*               : FinTime                                                    */
/*               : OldSumUtil                                                 */
```

```
/*                : SumUtil                                                 */
/*                                                                          */
/* CALLS          : SELF                                                    */
/*                : clc_pro_utl(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

determ
wri_wpu_aux
(
    d_utl_cod,
    d_pro_lst,
    d_fin_tim,
    d_pro_utl,
    d_pro_utl
)

/* ************************************************************************ */
```

D.4 Prolog Source Files

Source code for the *Digital Console Scheduler* is listed below. Section D.4.7 lists the interface predicates including the main `goal`.

D.4.1 dcs_ast.pro

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_AST.PRO                                                  */
/*                                                                          */
/* NOTES     : Core predicates for optimal A* DMC scheduling algorithm      */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 10th June, 1995                                              */
/*                                                                          */
/* VERSION   : 8.7                                                          */
/* ************************************************************************ */

/* ======================================================================== */
/* Project definition                                                       */
/* ======================================================================== */

project "dcs.prj"

/* ======================================================================== */
/* Public predicate definitions                                             */
/* ======================================================================== */

include "dcs_glb.pub"

include "dcs_ast.pub"
include "dcs_clc.pub"
include "dcs_cpm.pub"
include "dcs_edb.pub"
include "dcs_idb.pub"
include "dcs_lst.pub"
include "dcs_sch.pub"
include "dcs_utl.pub"
include "dcs_wri.pub"

/* ======================================================================== */
```

```
/* Private predicate definitions                                          */
/* ====================================================================== */

include "dcs_ast.prv"

/* ====================================================================== */
   clauses
/* ====================================================================== */

/* ---------------------------------------------------------------------- */
/* PREDICATE     : ast_bgn_alg(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Begin A* algorithm with <HrtScheme> underestimate to place  */
/*               : <TaskList> tasks with <PrecList> precedence on <ProcList>,  */
/*               : binding <Schedule> to the DMC schedule generated       */
/*                                                                        */
/* PARAMETERS    : IpcScheme                                              */
/*               : HrtScheme                                              */
/*               : TaskList                                               */
/*               : PrecList                                               */
/*               : ProcList                                               */
/*               : Schedule                                               */
/*                                                                        */
/* CALLS         : ast_chp_liv(...)                                       */
/*               : ast_gol_nod(...)                                       */
/*               : ast_exp_tre(...)                                       */
/*               : ast_ini_tre(...)                                       */
/*               : sch_cmp_idl(...)                                       */
/*               : utl_lbl_tkf(...)                                       */
/*                                                                        */
/* DATABASE      : bt_open(...)                                           */
/*               : db_flush(...)                                          */
/*               : db_close(...)                                          */
/*               : db_delete(...)                                         */
/*                                                                        */
/* BACKTRACKING  : determ                                                 */
/* ---------------------------------------------------------------------- */

ast_bgn_alg( Ipc, Hrt, TaskList, PrecList, ProcList, Schedule):-
    utl_lbl_tkf( s_lbl_hle, PrecList),
    ast_ini_tre( TaskList, ProcList),
    bt_open( db_dcs_ast, c_ast_dba, BtrSel),
    utl_rep_eat(),
    ast_chp_liv( BtrSel, Node),
    ast_exp_tre( Ipc, Hrt, BtrSel, Node),
    db_flush( db_dcs_ast),
    ast_gol_nod( BtrSel, nd( _, GoalProcList, FinTime)), !,
    sch_cmp_idl( GoalProcList, FinTime, Schedule),
    db_close( db_dcs_ast),
    db_delete( c_ast_dba, c_dba_plc).

/* ---------------------------------------------------------------------- */
/* PREDICATE     : ast_ini_tre(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Initialise search tree for <TaskList> on <ProcList>    */
/*                                                                        */
/* PARAMETERS    : TaskList                                               */
/*               : ProcList                                               */
/*                                                                        */
/* CALLS         : ast_key_int(...)                                       */
/*                                                                        */
/* DATABASE      : db_create(...)                                         */
/*               : bt_create(...)                                         */
/*               : bt_close(...)                                          */
/*                                                                        */
/* BACKTRACKING  : determ                                                 */
/* ---------------------------------------------------------------------- */

ast_ini_tre( TaskList, ProcList):-
    db_create( db_dcs_ast, c_ast_dba, c_dba_plc),
    bt_create( db_dcs_ast, c_ast_dba, BtrSel, c_btr_len, c_btr_ord),
    chain_insertz( db_dcs_ast, leaf_chain, d_edb_dom,
        lf( nd( TaskList, ProcList, 0), 0), InitialRef),
    ast_key_int( InitialKey, 0),
    key_insert( db_dcs_ast, BtrSel, InitialKey, InitialRef),
    bt_close( db_dcs_ast, BtrSel).

/* ---------------------------------------------------------------------- */
/* PREDICATE     : ast_chp_liv(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Bind <Node> to cheapest live node in search tree <BtrSel>  */
/*                                                                        */
/* PARAMETERS    : BteSel                                                 */
/*               : Node                                                   */
/*                                                                        */
/* CALLS         : --                                                     */
/*                                                                        */
/* DATABASE      : --                                                     */
/*                                                                        */
/* BACKTRACKING  : determ                                                 */
/* ---------------------------------------------------------------------- */

ast_chp_liv( BtrSel, Node):-
    key_first( db_dcs_ast, BtrSel, _),
    key_current( db_dcs_ast, BtrSel, Key, Ref),
    ref_term( db_dcs_ast, d_edb_dom, Ref, lf(Node, _)),
    term_delete( db_dcs_ast, leaf_chain, Ref),
    key_delete( db_dcs_ast, BtrSel, Key, Ref).

/* ---------------------------------------------------------------------- */
/* PREDICATE     : ast_exp_tre(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Generate all successors of <Node>, using <IPCScheme> and  */
/*               : and <HrtScheme>, and insert them into the A* search tree  */
/*                                                                        */
/* PARAMETERS    : IpcScheme                                              */
/*               : HrtScheme                                              */
/*               : BtrSel                                                 */
/*               : Node                                                   */
/*                                                                        */
/* CALLS         : ast_ins_new(...)                                       */
/*               : ast_suc_nod(...)                                       */
```

```
/*                : wri_alg_mon(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

ast_exp_tre( IpcScheme, HrtScheme, BtrSel, nd( TaskList, ProcList, G0)):-
    ast_suc_nod( IpcScheme, nd( TaskList, ProcList, G0), NewNode, Cost),
    wri_alg_mon(),
    ast_ins_new( IpcScheme, HrtScheme, BtrSel, G0, NewNode, Cost),
    fail.

ast_exp_tre( _, _, _, _).

/* ------------------------------------------------------------------------ */
/* PREDICATE      : ast_suc_nod(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Generate a successor to <OldNode>, using <IPCScheme>    */
/*                                                                          */
/* PARAMETERS     : IpcScheme                                               */
/*                : OldNode                                                 */
/*                : Node                                                    */
/*                                                                          */
/* CALLS          : idb_tsk_exe(...)                                        */
/*                : lst_del_ndt(...)                                        */
/*                : sch_act_tsk(...)                                        */
/*                : sch_ins_idl(...)                                        */
/*                : sch_ins_ipc(...)                                        */
/*                : sch_ins_tsk(...)                                        */
/*                : sch_non_cor(...)                                        */
/*                : sch_pre_rel(...)                                        */
/*                : sch_pro_rsc(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : non-determ                                              */
/* ------------------------------------------------------------------------ */

ast_suc_nod( Ipc, nd( OldTL, OldPL, Fin1), nd( TL, PL, Fin), Cost):-
    % generator predicates
    lst_del_ndt( Task, OldTL, TL),
    lst_del_ndt( Proc, OldPL, Act1),
    Proc = pr(P,[tk(T,F)|D]),
    % check precedence and active tasks
    sch_pre_rel( Task, TL),
    sch_act_tsk( Task, F, Act1),
    % allocate IPC resources
    sch_non_cor( Task, Act1, NCRs),
    sch_ins_ipc( Ipc, NCRs, Act1, Act2, Fin1, Fin2, [tk(T,F)|D], IpcTL),
    sch_pro_rsc( Task, P, Fin2),
    % calculate finish time
    Task = tk( TN, _),
    idb_tsk_exe( TN, ExecTime),
    IpcTL = [tk(_,IpcF)|_],
    Time = IpcF + ExecTime,
    % insert task and calculate cost
    sch_ins_tsk( pr(P, [tk(TN,Time)|IpcTL]), Act2, PL, Fin2, Fin),
    Cost = Fin - Fin1.

ast_suc_nod( _, nd( TL, OldPL, Fin), nd( TL, PL, Fin), 0):-
    OldPL = [pr(P,[tk(T,F)|Done])|Act1],
    sch_ins_idl( pr(P,[tk(T,F)|Done]), Act1, PL).

/* ------------------------------------------------------------------------ */
/* PREDICATE      : ast_ins_new(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Insert <Node> after calculating heuristic underestimate */
/*                : given <IpcScheme> and <HrtScheme> and <Cost> of <Node>  */
/*                                                                          */
/* PARAMETERS     : IpcScheme                                               */
/*                : HrtScheme                                               */
/*                : BtrSel                                                  */
/*                : G0                                                      */
/*                : Node                                                    */
/*                : Cost                                                    */
/*                                                                          */
/* CALLS          : ast_hrt_val(...)                                        */
/*                : ast_ket_int(...)                                        */
/*                : ast_wat_dog(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

ast_ins_new( IpcScheme, HrtScheme, BtrSel, G0, Node, Cost):-
    G = G0 + Cost,
    ast_hrt_val( HrtScheme, IpcScheme, Node, H),
    F = G + H, !,
    ast_wat_dog(),
    chain_insertz( db_dcs_ast, leaf_chain, d_edb_dom, lf(Node, F), NewRef),
    ast_key_int( NewKey, F),
    key_insert( db_dcs_ast, BtrSel, NewKey, NewRef).

/* ------------------------------------------------------------------------ */
/* PREDICATE      : ast_hrt_val(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Bind <Heuristic> to underestimate determined from       */
/*                : given <IpcScheme> and <HrtScheme>                       */
/*                                                                          */
/* PARAMETERS     : HrtScheme                                               */
/*                : IpcScheme                                               */
/*                : Node                                                    */
/*                : Heuristic                                               */
/*                                                                          */
/* CALLS          : ast_mpc_hrt(...)                                        */
/*                : ast_mia_hrt(...)                                        */
/*                : ast_mpc_hrt(...)                                        */
/*                : clc_max_val(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
```

```
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

ast_hrt_val( s_hrt_non, _, _, 0):- !.

ast_hrt_val( s_hrt_mpc, _, Node, MpcHrt):- !,
    ast_mpc_hrt( Node, MpcHrt).

ast_hrt_val( s_hrt_mia, Ipc, Node, MiaHrt):- !,
    ast_mia_hrt( Ipc, Node, MiaHrt).

ast_hrt_val( s_hrt_mrc, _, Node, MrcHrt):- !,
    ast_mrc_hrt( Node, MrcHrt).

ast_hrt_val( s_hrt_all, Ipc, Node, Hrt):- !,
    ast_mpc_hrt( Node, MpcHrt),
    ast_mia_hrt( Ipc, Node, MiaHrt),
    ast_mrc_hrt( Node, MrcHrt),
    clc_max_val( MpcHrt, MiaHrt, TmpHrt),
    clc_max_val( TmpHrt, MrcHrt, Hrt).

/* ---------------------------------------------------------------------- */
/* PREDICATE     : ast_mpc_hrt(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Bind <MpcHrt> to MPC heuristic underestimate for <Node> */
/*                                                                        */
/* PARAMETERS    : Node                                                   */
/*               : MpcHrt                                                 */
/*                                                                        */
/* CALLS         : clc_max_val(...)                                       */
/*               : clc_seq_sch(...)                                       */
/*               : clc_sum_fin(...)                                       */
/*               : lst_num_elm(...)                                       */
/*                                                                        */
/* DATABASE      : --                                                     */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

ast_mpc_hrt( nd( Tasks, Procs, Fin), MpcHrt):-
    clc_seq_sch( Tasks, TaskTot),
    clc_sum_fin( Procs, FinTot),
    lst_num_elm( Procs, NumProcs),
    HrtFin = (TaskTot + FinTot) / NumProcs,
    OldHrt = HrtFin - Fin,
    clc_max_val( OldHrt, 0, MpcHrt).

/* ---------------------------------------------------------------------- */
/* PREDICATE     : ast_mia_hrt(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Bind <MiaHrt> to MIA heuristic underestimate for <Node> */
/*                                                                        */
/* PARAMETERS    : IpcScheme                                              */
/*               : Node                                                   */
/*               : MiaHrt                                                 */
/*                                                                        */
/* CALLS         : ast_mia_tsk(...)                                       */
/*                                                                        */
/* DATABASE      : --                                                     */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

ast_mia_hrt( IpcScheme, nd( Tasks, Procs, Fin), MiaHrt):-
    ast_mia_tsk( IpcScheme, Tasks, Procs, Fin, 0, MiaHrt).

/* ---------------------------------------------------------------------- */
/* PREDICATE     : ast_mia_tsk(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Bind <MiaSum> to sum of MIA heuristic underestimates for */
/*               : remaining tasks in <TaskList> on active <ProcList>     */
/*                                                                        */
/* PARAMETERS    : IpcScheme                                              */
/*               : TaskList                                               */
/*               : ProcList                                               */
/*               : Fin                                                    */
/*               : OldMiaSum                                              */
/*               : MiaSum                                                 */
/*                                                                        */
/* CALLS         : SELF                                                   */
/*               : ast_mia_pro(...)                                       */
/*               : clc_seq_sch(...)                                       */
/*                                                                        */
/* DATABASE      : --                                                     */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

ast_mia_tsk( _, [], _, _, MiaSum, MiaSum):- !.

ast_mia_tsk( IpcScheme, [Task|RestTasks], ProcList, Fin, OldMiaSum, MiaSum):-
    clc_seq_sch( [Task|RestTasks], SeqSch),
    MiaIni = SeqSch,
    ast_mia_pro( IpcScheme, Task, ProcList, ProcList, Fin, MiaIni, Mia),
    NewMiaSum = OldMiaSum + Mia,
    ast_mia_tsk( IpcScheme, RestTasks, ProcList, Fin, NewMiaSum, MiaSum).

/* ---------------------------------------------------------------------- */
/* PREDICATE     : ast_mia_pro(...)                                       */
/*                                                                        */
/* DESCRIPTION   : Bind <MiaHrt> to MIA underestimate for <Task> on <ProcList> */
/*                                                                        */
/* PARAMETERS    : IpcScheme                                              */
/*               : Task                                                   */
/*               : OldProcList                                            */
/*               : ProcList                                               */
/*               : Fin                                                    */
/*               : OldMiaHrt                                              */
/*               : MiaHrt                                                 */
/*                                                                        */
/* CALLS         : SELF                                                   */
```

```
/*                : clc_min_val(...)                                        */
/*                : idb_tsk_exe(...)                                        */
/*                : lst_del_elm(...)                                        */
/*                : sch_act_tsk(...)                                        */
/*                : sch_ins_ipc(...)                                        */
/*                : sch_non_cor(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

ast_mia_pro( _, _, [], _, _, Hrt, Hrt):- !.

ast_mia_pro( IpcScheme, Task, [Proc|RestProcs], ProcList, Fin, OldHrt, Hrt):-
    lst_del_elm( Proc, ProcList, Act1),
    Proc = pr(_,[tk(T,F)|D]),
    % check active tasks
    sch_act_tsk( Task, F, Act1), !,
    % allocate IPC resources
    sch_non_cor( Task, Act1, NCRs),
    sch_ins_ipc( IpcScheme, NCRs, Act1, _, Fin, _, [tk(T,F)|D], IpcTL),
    % calc finish time
    Task = tk( TN, _),
    idb_tsk_exe( TN, ExecTime),
    IpcTL = [tk(_,IpcF)|_],
    Time = IpcF + ExecTime,
    % calc heuristic value
    ThisHrt = Time - Fin,
    clc_min_val( OldHrt, ThisHrt, NewHrt),
    ast_mia_pro( IpcScheme, Task, RestProcs, ProcList, Fin, NewHrt, Hrt).

ast_mia_pro( IpcScheme, Task, [_|RestProcs], ProcList, Fin, OldHrt, Hrt):-
    ast_mia_pro( IpcScheme, Task, RestProcs, ProcList, Fin, OldHrt, Hrt).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_mrc_hrt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <MrcHrt> to MRC heuristic underestimate for <Node>  */
/*                                                                          */
/* PARAMETERS    : Node                                                     */
/*                 MrcHrt                                                   */
/*                                                                          */
/* CALLS         : ast_mrc_pro(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

ast_mrc_hrt( nd(_, ProcList, FinTime), MrcHrt):-
    ast_mrc_pro( ProcList, FinTime, 0, MrcHrt).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_mrc_pro(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <MrcHrt> to MRC underestimate for <ProcList>        */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*                 FinTime                                                  */
/*                 OldMrcHrt                                                */
/*                 MrcHrt                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                 ast_mrc_rct(...)                                         */
/*                 clc_max_val(...)                                         */
/*                 idb_tsk_exe(...)                                         */
/*                 idb_tsk_lvl(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

ast_mrc_pro( [], _, MrcHrt, MrcHrt):- !.

ast_mrc_pro( [pr(_,TaskList)|RestProcs], FinTime, OldMrcHrt, MrcHrt):-
    ast_mst_rct( TaskList, Task, TaskFinTime), !,
    idb_tsk_lvl( Task, Level),
    idb_tsk_exe( Task, ExecTime),
    RemainingCP = Level - ExecTime,
    Heuristic = (TaskFinTime - FinTime) + RemainingCP,
    clc_max_val( OldMrcHrt, Heuristic, NewMrcHrt),
    ast_mrc_pro( RestProcs, FinTime, NewMrcHrt, MrcHrt).

ast_mrc_pro( [_|RestProcs], FinTime, OldMrcHrt, MrcHrt):-
    ast_mrc_pro( RestProcs, FinTime, OldMrcHrt, MrcHrt).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_mst_rct(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <FinTime> and <Task> to most recent task scheduled in */
/*                 in <TaskList>                                            */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*                 Task                                                     */
/*                 FinTime                                                  */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

ast_mst_rct( [tk(idle,_)], _, _):- !, fail.

ast_mst_rct( [tk(Task, FinTime)|_], Task, FinTime):-
    Task <> "idle", Task <> "ipc", !.

ast_mst_rct( [tk(idle,FinTime),tk(Task, _)|_], Task, FinTime):-
    Task <> "idle", Task <> "ipc", !.
```

```
ast_mst_rct( [_|RestTasks], Task, FinTime):-
    ast_mst_rct( RestTasks, Task, FinTime).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_wat_dog(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Increment search tree watch dog in internal DB           */
/*                                                                          */
/* PARAMETERS    : --                                                       */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : retract( num_sea_nod(...))                               */
/*               : assert( num_sea_nod(...))                                */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

ast_wat_dog():-
    retract( num_sea_nod( OldNum)), !,
    NewNum = OldNum + 1,
    assert( num_sea_nod( NewNum)).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_gol_nod(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <Node> to goal node if found in search tree <BtrSel>*/
/*                                                                          */
/* PARAMETERS    : BtrSel                                                   */
/*               : Node                                                     */
/*                                                                          */
/* CALLS         : ast_agn_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

ast_gol_nod( BtrSel, Node):-
    key_first( db_dcs_ast, BtrSel, _),
    key_current( db_dcs_ast, BtrSel, Key, Ref),
    ast_agn_aux( BtrSel, Key, Ref, Node).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_agn_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for ast_gol_nod(...)                 */
/*                                                                          */
/* PARAMETERS    : BtrSel                                                   */
/*               : Key                                                      */
/*               : Ref                                                      */
/*               : Node                                                     */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

ast_agn_aux( _, _, Ref, nd([],P,G)):-
    ref_term( db_dcs_ast, d_edb_dom, Ref, lf(nd([], P, G), _)), !.

ast_agn_aux( BtrSel, Key, _, Node):-
    key_next( db_dcs_ast, BtrSel, _),
    key_current( db_dcs_ast, BtrSel, NextKey, NextRef),
    NextKey = Key,
    ast_agn_aux( BtrSel, Key, NextRef, Node).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : ast_key_int(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Map <NodCst> to a Btree <BtrKey>, represented as a string */
/*                                                                          */
/* PARAMETERS    : BtrSel                                                   */
/*               : NodCst                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

ast_key_int( Key, Integer):-
    bound( Integer),
    free( Key), !,
    str_int( String, Integer),
    str_len( String, StringLength),
    BlankStringLength = c_btr_len - StringLength,
    str_len( BlankString, BlankStringLength),
    concat( BlankString, String, Key).

ast_key_int( Key, Integer):-
    str_int( Key, Integer).

/* ************************************************************************ */
```

D.4.2 dcs_clc.pro

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_CLC.PRO                                                    */
/*                                                                            */
/* NOTES     : Numerical predicates used by DCS modules                       */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 20th May, 1995                                                 */
/*                                                                            */
/* VERSION   : 1.2                                                            */
/* ************************************************************************** */

/* ========================================================================== */
/* Project definition file                                                    */
/* ========================================================================== */

project "dcs.prj"

/* ========================================================================== */
/* Public definition files                                                    */
/* ========================================================================== */

include "dcs_glb.pub"

include "dcs_clc.pub"
include "dcs_idb.pub"
include "dcs_lst.pub"
include "dcs_utl.pub"

/* ========================================================================== */
/* Private predicate definitions                                              */
/* ========================================================================== */

include "dcs_clc.prv"

/* ========================================================================== */
   clauses
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : clc_crt_pth(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Calculate critical-path determined by <PrecList> ensuring  */
/*               : all level(...) clauses are removed from internal database  */
/*                                                                            */
/* PARAMETERS    : PrecList                                                   */
/*               : CritPath                                                   */
/*                                                                            */
/* CALLS         : clc_ccp_aux(...)                                           */
/*               : utl_lbl_tkf(...)                                           */
/*                                                                            */
/* DATABASE      : retractall( level(...))                                    */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

clc_crt_pth( PrecList, CritPath):-
    utl_lbl_tkf( s_lbl_hle, PrecList),
    clc_ccp_aux( 0, CritPath),
    retractall( level( _, _)).

/* -------------------------------------------------------------------------- */
/* PREDICATE     : clc_ccp_aux(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Auxiliary predicate for clc_crt_pth(...)                   */
/*                                                                            */
/* PARAMETERS    : CritPath1                                                  */
/*               : CritPath2                                                  */
/*                                                                            */
/* CALLS         : SELF                                                       */
/*               : clc_max_val(...)                                           */
/*                                                                            */
/* DATABASE      : retract( level(...))                                       */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

clc_ccp_aux( OldCritPath, CritPath):-
    retract( level( _, Level)), !,
    clc_max_val( OldCritPath, Level, NewCritPath),
    clc_ccp_aux( NewCritPath, CritPath).

clc_ccp_aux( CritPath, CritPath).

/* -------------------------------------------------------------------------- */
/* PREDICATE     : clc_fin_tim(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Bind <SchFin> to finishing-time of <Schedule>, with idle   */
/*               : periods completed using sch_cmp_idl(...)                   */
/*                                                                            */
/* PARAMETERS    : Schedule                                                   */
/*               : SchFin                                                     */
/*                                                                            */
/* CALLS         : --                                                         */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

clc_fin_tim( [pr( _, [tk( _, TskFin)|_])|_], SchFin):-
    SchFin = TskFin.
```

```
/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_hdw_rsc(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <ProcRsrc> to sum of hardware resource, <ProcList>  */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : ProcRsrc                                                 */
/*                                                                          */
/* CALLS         : clc_chr_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_hdw_rsc( ProcList, ProcRsrc):-
    clc_chr_aux( ProcList, 0, ProcRsrc).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_chr_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for clc_hdw_rsc(...)                 */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : ProcRsrc1                                                */
/*               : ProcRsrc2                                                */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : idb_pro_exe(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_chr_aux( [], ProcRsrc, ProcRsrc):- !.

clc_chr_aux( [pr( ProcName, _)|RestProcs], OldProcRsrc, ProcRsrc):-
    idb_pro_exe( ProcName, ProcCycles),
    NewProcRsrc = OldProcRsrc + ProcCycles,
    clc_chr_aux( RestProcs, NewProcRsrc, ProcRsrc).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_max_val(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <Var3> to maximum value of <Var1> and <Var2>        */
/*                                                                          */
/* PARAMETERS    : Var1                                                     */
/*               : Var2                                                     */
/*               : Var3                                                     */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_max_val( X, Y, X):- X > Y, !.

clc_max_val( _, Y, Y).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_min_val(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <Var3> to minimum value of <Var1> and <Var2>        */
/*                                                                          */
/* PARAMETERS    : Var1                                                     */
/*               : Var2                                                     */
/*               : Var3                                                     */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_min_val( X, Y, X):- X < Y, !.

clc_min_val( _, Y, Y).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_num_ncr(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <NumNCRs> to the number of non-coresident tasks held */
/*               : in <NRCList>                                             */
/*                                                                          */
/* PARAMETERS    : NCRList                                                  */
/*               : NumNCRs                                                  */
/*                                                                          */
/* CALLS         : clc_cnn_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_num_ncr( NCRList, NumNCRs):-
    clc_cnn_aux( NCRList, 0, NumNCRs).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_cnn_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for clc_num_ncr(...)                 */
/*                                                                          */
/* PARAMETERS    : NCRList                                                  */
/*               : NumNCRs1                                                 */
/*               : NumNCRs2                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
```

```
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

clc_cnn_aux( [], NumNCRs, NumNCRs):- !.

clc_cnn_aux( [pr( _,TaskList)|RestProcs], OldNumNCRs, NumNCRs):-
    lst_num_elm( TaskList, NumNCRInc),
    NewNumNCRs = OldNumNCRs + NumNCRInc,
    clc_cnn_aux( RestProcs, NewNumNCRs, NumNCRs).

/* -------------------------------------------------------------------- */
/* PREDICATE    : clc_pro_utl(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Bind <ProcUtil> calculated w.r.t. sample period,      */
/*              : <s_utl_smp>, or <FinishTime>, <s_utl_sch>, for <TaskList> */
/*              : scheduled on <ProcName>                               */
/*                                                                      */
/* PARAMETERS   : symbol                                                */
/*              : ProcName                                              */
/*              : TaskList                                              */
/*              : FinishTime                                            */
/*              : ProcUtil                                              */
/*                                                                      */
/* CALLS        : clc_pro_tim(...)                                      */
/*              : idb_pro_exe(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

clc_pro_utl( s_utl_smp, ProcName, TaskList, _, ProcUtil):- !,
    idb_pro_exe( ProcName, Cycles),
    clc_pro_tim( TaskList, TotalProcTime),
    ProcUtil = (TotalProcTime / Cycles) * 100.

clc_pro_utl( s_utl_sch, _, TaskList, FinishTime, ProcUtil):-
    clc_pro_tim( TaskList, TotalProcTime),
    ProcUtil = (TotalProcTime / FinishTime) * 100.

/* -------------------------------------------------------------------- */
/* PREDICATE    : clc_pro_tim(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Bind <ProcTime> to the processing time of the reversed */
/*              : <TaskList> ignoring idle periods                      */
/*                                                                      */
/* PARAMETERS   : TaskList                                              */
/*              : ProcTime                                              */
/*                                                                      */
/* CALLS        : clc_cpt_aux(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

clc_pro_tim( TaskList, ProcTime):-
    clc_cpt_aux( TaskList, 0, ProcTime).

/* -------------------------------------------------------------------- */
/* PREDICATE    : clc_cpt_aux(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for clc_pro_tim(...)              */
/*                                                                      */
/* PARAMETERS   : TaskList                                              */
/*              : ProcTime1                                             */
/*              : ProcTime2                                             */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

clc_cpt_aux( [], TotalProcTime, TotalProcTime):- !.

clc_cpt_aux( [tk("idle",_)|RestOfTasks], ProcTime, TotalProcTime):- !,
    clc_cpt_aux( RestOfTasks, ProcTime, TotalProcTime).

clc_cpt_aux( [tk("ipc",_)|RestOfTasks], ProcTime, TotalProcTime):- !,
    clc_cpt_aux( RestOfTasks, ProcTime, TotalProcTime).

clc_cpt_aux( [tk(_,FT2),tk(T1,FT1)|RestOfTasks], ProcTime, TotalProcTime):-
    NewProcTime = ProcTime + (FT2 - FT1),
    clc_cpt_aux( [tk(T1,FT1)|RestOfTasks], NewProcTime, TotalProcTime).

/* -------------------------------------------------------------------- */
/* PREDICATE    : clc_run_tim(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Calculate the <RunTime> required to generate schedule */
/*                                                                      */
/* PARAMETERS   : RunTime                                               */
/*                                                                      */
/* CALLS        : clc_sub_cry(...)                                      */
/*                                                                      */
/* DATABASE     : retract( dbatime(...))                                */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

clc_run_tim( RunTime):-
    retract( dbatime( H1,M1,S1,C1)),
    retract( dbatime( H2,M2,S2,C2)), !,
    clc_sub_cry( S1, NewS1, C1, C2, AnsC, 100),
    clc_sub_cry( M1, NewM1, NewS1, S2, AnsS, 60),
    clc_sub_cry( H1, NewH1, NewM1, M2, AnsM, 60),
    AnsH = H2 - NewH1,
    format( RunTime, "%d:%d:%d.%d", AnsH, AnsM, AnsS, AnsC).
```

```
/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_seq_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Calculate the sequential schedule <SeqlSchd> for <TaskList> */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : SeqlSchd                                                 */
/*                                                                          */
/* CALLS         : clc_css_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_seq_sch( TaskList, SeqlSchd):-
    clc_css_aux( TaskList, 0, SeqlSchd).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_css_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for clc_seq_sch(...)                 */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : OldSeqSch                                                */
/*               : SeqSch                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_css_aux( [], SeqSchd, SeqSchd):- !.

clc_css_aux( [tk( _, ExecTime)|OtherTasks], OldSeqlSchd, SeqlSchd):-
    NewSeqlSchd = OldSeqlSchd + ExecTime,
    clc_css_aux( OtherTasks, NewSeqlSchd, SeqlSchd).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_sub_cry(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Calculate <Right2> - <Right1> with base <Factor> with carry */
/*               : across to <Left2> if required                            */
/*                                                                          */
/* PARAMETERS    : Left1                                                    */
/*               : Left2                                                    */
/*               : Right1                                                   */
/*               : Right2                                                   */
/*               : Factor                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_sub_cry( Left1, NewLeft1, Right1, Right2, AnsRight, Factor):-
    (Right2 - Right1) < 0, !,
    AnsRight = (Right2 - Right1) + Factor,
    NewLeft1 = Left1 + 1.

clc_sub_cry( Left1, Left1, Right1, Right2, AnsRight, _):-
    AnsRight = (Right2 - Right1).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_sum_fin(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <FinTot> to sum of finish times of PEs in <ProcList> */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : FinTot                                                   */
/*                                                                          */
/* CALLS         : clc_csf_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_sum_fin( ProcList, FinTot):-
    clc_csf_aux( ProcList, 0, FinTot).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_csf_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for clc_sum_fin(...)                 */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : FinTot1                                                  */
/*               : FinTot2                                                  */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

clc_csf_aux( [], FinTot, FinTot):- !.

clc_csf_aux( [pr(_, [tk(_,F)|_])|RestProcs], OldFinTot, FinTot):-
    NewFinTot = OldFinTot + F,
    clc_csf_aux( RestProcs, NewFinTot, FinTot).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : clc_thr_put(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <ThroughPut> of <TaskList> as per <Schedule>        */
/*                                                                          */
```

```
/* PARAMETERS   : TaskList                                                  */
/*              : Schedule                                                  */
/*              : ThroughPut                                                */
/*                                                                          */
/* CALLS        : clc_fin_tim(...)                                          */
/*              : idb_pro_exe(...)                                          */
/*              : lst_num_elm(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

clc_thr_put( TaskList, [pr( ProcName, Tasks)|_], ThroughPut):-
    lst_num_elm( TaskList, NumTasks),
    clc_fin_tim( [pr( ProcName, Tasks)], FinTime),
    idb_pro_exe( ProcName, ProcCycles),
    ThroughPut = (NumTasks / FinTime) * ProcCycles.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : clc_tsk_grn(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Calculate granularity statistics for <TaskList>           */
/*                                                                          */
/* PARAMETERS   : TaskList                                                  */
/*              : MinGran                                                   */
/*              : MaxGran                                                   */
/*              : AvgGran                                                   */
/*                                                                          */
/* CALLS        : clc_ctg_aux(...)                                          */
/*              : clc_seq_sch(...)                                          */
/*              : idb_tsk_exe(...)                                          */
/*              : lst_num_elm(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

clc_tsk_grn( TaskList, MinGran, MaxGran, AvgGran):-
    clc_seq_sch( TaskList, SeqlSchd),
    lst_num_elm( TaskList, NumTasks),
    AvgGran  = SeqlSchd / NumTasks,
    TaskList = [tk( TaskName, _)|_],
    idb_tsk_exe( TaskName, ExecTime),
    clc_ctg_aux( TaskList, ExecTime, 0, MinGran, MaxGran).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : clc_ctg_aux(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Auxiliary predicate for clc_tsk_grn(...)                  */
/*                                                                          */
/* PARAMETERS   : TaskList                                                  */
/*              : MinGran                                                   */
/*              : MaxGran                                                   */
/*              : AvgGran                                                   */
/*                                                                          */
/* CALLS        : SELF                                                      */
/*              : clc_max_val(...)                                          */
/*              : clc_min_val(...)                                          */
/*              : idb_tsk_exe(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

clc_ctg_aux( [], MinTask, MaxTask, MinTask, MaxTask):- !.

clc_ctg_aux( [tk( TaskName, _)|RestTasks], OldMin, OldMax, MinTask, MaxTask):-
    idb_tsk_exe( TaskName, ExecTime),
    clc_min_val( ExecTime, OldMin, NewMin),
    clc_max_val( ExecTime, OldMax, NewMax),
    clc_ctg_aux( RestTasks, NewMin, NewMax, MinTask, MaxTask).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : clc_tsk_par(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Calculate critical-path, <CritPath>, and taskforce        */
/*              : parallelism, <TaskPara>, for <TaskList> with <PrecList>   */
/*              : precedence relations                                      */
/*                                                                          */
/* PARAMETERS   : TaskList                                                  */
/*              : PrecList                                                  */
/*              : CritPath                                                  */
/*              : TaskPara                                                  */
/*                                                                          */
/* CALLS        : clc_crt_pth(...)                                          */
/*              : clc_seq_sch(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

clc_tsk_par( TaskList, PrecList, CritPath, TaskPara):-
    clc_seq_sch( TaskList, SeqSchLen),
    clc_crt_pth( PrecList, CritPath),
    TaskPara = SeqSchLen / CritPath.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : clc_tsk_spd(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Calculate <SpeedUp> % for <Schedule> of <TaskList>        */
/*                                                                          */
/* PARAMETERS   : TaskList                                                  */
/*              : Schedule                                                  */
/*              : SpeedUp                                                   */
/*                                                                          */
/* CALLS        : clc_fin_tim(...)                                          */
/*              : clc_seq_sch(...)                                          */
/*                                                                          */
```

```
/* DATABASE      : --                                                    */
/*                                                                       */
/* BACKTRACKING : determ                                                 */
/* --------------------------------------------------------------------- */

clc_tsk_spd( TaskList, Schedule, SpeedUp):-
    clc_seq_sch( TaskList, SeqSch),
    clc_fin_tim( Schedule, FinTime),
    SpeedUp = (SeqSch / FinTime) * 100.

/* ********************************************************************* */
```

D.4.3 dcs_cpm.pro

```
/* ********************************************************************* */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England              */
/* ********************************************************************* */

/* ********************************************************************* */
/* MODULE    : DCS_CPM.PRO                                               */
/*                                                                       */
/* NOTES     : Core predicates for static-level DMC list scheduling      */
/*                                                                       */
/* AUTHOR    : Ken N LINTON                                              */
/*                                                                       */
/* DATE      : 20th May, 1995                                            */
/*                                                                       */
/* VERSION   : 4.5                                                       */
/* ********************************************************************* */

/* ===================================================================== */
/* Project definition                                                    */
/* ===================================================================== */

project "dcs.prj"

/* ===================================================================== */
/* Public predicate definitions                                          */
/* ===================================================================== */

include "dcs_glb.pub"

include "dcs_clc.pub"
include "dcs_cpm.pub"
include "dcs_idb.pub"
include "dcs_lst.pub"
include "dcs_sch.pub"
include "dcs_utl.pub"
include "dcs_wri.pub"

/* ===================================================================== */
/* Private predicate definitions                                         */
/* ===================================================================== */

include "dcs_cpm.prv"

/* ===================================================================== */
   clauses
/* ===================================================================== */

/* --------------------------------------------------------------------- */
/* PREDICATE     : cpm_bgn_alg(...)                                      */
/*                                                                       */
/* DESCRIPTION   : Begin static labelled <LblScheme> algorithm to schedule */
/*               : <TaskList> tasks with <PrecList> precedence on <ProcList>, */
```

```
/*                : binding <Schedule> to the DMC schedule generated          */
/*                                                                            */
/* PARAMETERS     : IpcScheme                                                 */
/*                : LblScheme                                                 */
/*                : TaskList                                                  */
/*                : PrecList                                                  */
/*                : ProcList                                                  */
/*                : Schedule                                                  */
/*                                                                            */
/* CALLS          : cpm_cre_sch(...)                                          */
/*                : utl_lbl_tkf(...)                                          */
/*                : utl_srt_prl(...)                                          */
/*                                                                            */
/* DATABASE       : --                                                        */
/*                                                                            */
/* BACKTRACKING   : determ                                                    */
/* -------------------------------------------------------------------------- */

cpm_bgn_alg( IpcScheme, LblScheme, TaskList, PrecList, ProcList, Schedule):-
    utl_lbl_tkf( LblScheme, PrecList),
    utl_srt_prl( TaskList, PriorityList),
    cpm_cre_sch( IpcScheme, PriorityList, ProcList, 0, Schedule).

/* -------------------------------------------------------------------------- */
/* PREDICATE      : cpm_cre_sch(...)                                          */
/*                                                                            */
/* DESCRIPTION    : Create schedule of <TaskList> on <ProcList> and bind      */
/*                : <Schedule> to result with completed idle periods          */
/*                                                                            */
/* PARAMETERS     : IpcScheme                                                 */
/*                : TaskList                                                  */
/*                : ProcList                                                  */
/*                : FinTime                                                   */
/*                : Schedule                                                  */
/*                                                                            */
/* CALLS          : SELF                                                      */
/*                : cpm_sch_nxt(...)                                          */
/*                : sch_cmp_idl(...)                                          */
/*                : wri_alg_mon(...)                                          */
/*                                                                            */
/* DATABASE       : --                                                        */
/*                                                                            */
/* BACKTRACKING   : determ                                                    */
/* -------------------------------------------------------------------------- */

cpm_cre_sch( _, [], ProcList, FinTime, Schedule):- !,
    sch_cmp_idl( ProcList, FinTime, Schedule).

cpm_cre_sch( IpcScheme, OldTL, OldPL, OldFin, ProcList):-
    cpm_sch_nxt( IpcScheme, OldTL, OldPL, OldFin, NewTL, NewPL, NewFin),
    wri_alg_mon(),
    cpm_cre_sch( IpcScheme, NewTL, NewPL, NewFin, ProcList).

/* -------------------------------------------------------------------------- */
/* PREDICATE      : cpm_sch_nxt(...)                                          */
/*                                                                            */
/* DESCRIPTION    : Bind <TaskList>, <ProcList> and <FinTime> after next step */
/*                : in statically labelled list scheduling discipline         */
/*                                                                            */
/* PARAMETERS     : IpcScheme                                                 */
/*                : OldTaskList                                               */
/*                : OldProcList                                               */
/*                : OldFinTime                                                */
/*                : TaskList                                                  */
/*                : ProcList                                                  */
/*                : FinTime                                                   */
/*                                                                            */
/* CALLS          : idb_tsk_exe(...)                                          */
/*                : lst_del_ndt(...)                                          */
/*                : sch_act_tsk(...)                                          */
/*                : sch_ins_idl(...)                                          */
/*                : sch_ins_ipc(...)                                          */
/*                : sch_ins_tsk(...)                                          */
/*                : sch_non_cor(...)                                          */
/*                : sch_pre_rel(...)                                          */
/*                : sch_pro_rsc(...)                                          */
/*                                                                            */
/* DATABASE       : --                                                        */
/*                                                                            */
/* BACKTRACKING   : determ                                                    */
/* -------------------------------------------------------------------------- */

cpm_sch_nxt( IpcScheme, OldTL, [pr(P, [tk(T,F)|D])|Act1], Fin1, TL, PL, Fin):-
    lst_del_ndt( Task, OldTL, TL),
    sch_pre_rel( Task, TL),
    sch_act_tsk( Task, F, Act1),
    sch_non_cor( Task, Act1, NCRs),
    sch_ins_ipc( IpcScheme, NCRs, Act1, Act2, Fin1, Fin2, [tk(T,F)|D], IpcTL),
    sch_pro_rsc( Task, P, Fin2), !,
    Task = tk( TN, _),
    idb_tsk_exe( TN, ExecTime),
    IpcTL = [tk(_,IpcF)|_],
    Time = IpcF + ExecTime,
    sch_ins_tsk( pr(P, [tk(TN,Time)|IpcTL]), Act2, PL, Fin2, Fin).

cpm_sch_nxt( _, TL, [pr(P, [tk(T,F)|D])|Act1], Fin, TL, PL, Fin):-
    sch_ins_idl( pr(P, [tk(T,F)|D]), Act1, PL).

/* ************************************************************************** */
```

D.4.4 dcs_dyn.pro

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_DYN.PRO                                                    */
/*                                                                            */
/* NOTES     : Core predicates for dynamic-level DMC list scheduling          */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 28th May, 1995                                                 */
/*                                                                            */
/* VERSION   : 7.1                                                            */
/* ************************************************************************** */

/* ========================================================================== */
/* Project definition file                                                    */
/* ========================================================================== */

project "dcs.prj".

/* ========================================================================== */
/* Public definition files                                                    */
/* ========================================================================== */

include "dcs_glb.pub"

include "dcs_clc.pub"
include "dcs_dyn.pub"
include "dcs_edb.pub"
include "dcs_idb.pub"
include "dcs_lst.pub"
include "dcs_sch.pub"
include "dcs_utl.pub"
include "dcs_wri.pub"

/* ========================================================================== */
/* Private definition files                                                   */
/* ========================================================================== */

include "dcs_dyn.prv"

/* ========================================================================== */
   clauses
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : dyn_bgn_alg(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Begin dynamically labelled <Dyn> algorithm to schedule     */
/*               : <TaskList> tasks with <PrecList> precedence on <ProcList>, */
/*               : binding <Schedule> to the DMC schedule generated           */
/*                                                                            */
/* PARAMETERS    : Ipc                                                        */
/*               : Lbl                                                        */
/*               : Dyn                                                        */
/*               : TaskList                                                   */
/*               : PrecList                                                   */
/*               : ProcList                                                   */
/*               : Schedule                                                   */
/*                                                                            */
/* CALLS         : dyn_ini_edb(...)                                           */
/*               : dyn_cre_sch(...)                                           */
/*               : utl_lbl_tkf(...)                                           */
/*                                                                            */
/* DATABASE      : db_close(...)                                              */
/*               : db_delete(...)                                             */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

dyn_bgn_alg( Ipc, Lbl, Dyn, TaskList, PrecList, ProcList, Schedule):-
    % statically label taskforce
    utl_lbl_tkf( Lbl, PrecList),
    % initialise external database
    dyn_ini_edb(),
    % schedule the priority list
    dyn_cre_sch( Ipc, Dyn, TaskList, ProcList, 0, Schedule),
    % close and delete external dbase used for DLs
    db_close( db_dcs_dyn),
    db_delete( c_dyn_dba, c_dba_plc).

/* -------------------------------------------------------------------------- */
/* PREDICATE     : dyn_ini_edb(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Initialise external DB to store dynamic level records      */
/*                                                                            */
/* PARAMETERS    : --                                                         */
/*                                                                            */
/* CALLS         : --                                                         */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

dyn_ini_edb():-
    db_create( db_dcs_dyn, c_dyn_dba, c_dba_plc),
    bt_create( db_dcs_dyn, c_dyn_dba, DynLvlBtSel, c_btr_len, c_btr_ord),
    bt_close(  db_dcs_dyn, DynLvlBtSel).

/* -------------------------------------------------------------------------- */
/* PREDICATE     : dyn_cre_sch(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Create schedule of <TaskList> on <ProcList> and bind       */
/*               : <Schedule> to result with completed idle periods           */
```

```
/*                                                                      */
/* PARAMETERS    : IpcScheme                                            */
/*               : DynScheme                                            */
/*               : TaskList                                             */
/*               : ProcList                                             */
/*               : FinTime                                              */
/*               : Schedule                                             */
/*                                                                      */
/* CALLS         : SELF                                                 */
/*               : dyn_sch_nxt(...)                                     */
/*               : sch_cmp_idl(...)                                     */
/*               : wri_alg_mon(...)                                     */
/*                                                                      */
/* DATABASE      : --                                                   */
/*                                                                      */
/* BACKTRACKING  : determ                                               */
/* -------------------------------------------------------------------- */

dyn_cre_sch( _, _, [], ProcList, FinTime, Schedule):- !,
    sch_cmp_idl( ProcList, FinTime, Schedule).

dyn_cre_sch( Ipc, Dyn, OldTL, OldPL, OldFin, PL):-
    dyn_sch_stp( Ipc, Dyn, OldTL, OldPL, OldFin, NewTL, NewPL, NewFin),
    wri_alg_mon(),
    dyn_cre_sch( Ipc, Dyn, NewTL, NewPL, NewFin, PL).

/* -------------------------------------------------------------------- */
/* PREDICATE     : dyn_sch_stp(...)                                     */
/*                                                                      */
/* DESCRIPTION   : Create <DynScheme> labelled scheduling step with <IPC>  */
/*               : communications and bind <TL>, <PL> and <Fin> to result */
/*                                                                      */
/* PARAMETERS    : IpcScheme                                            */
/*               : DynScheme                                            */
/*               : OldTaskList                                          */
/*               : OldProcList                                          */
/*               : OldFinTime                                           */
/*               : TaskList                                             */
/*               : ProcList                                             */
/*               : FinTime                                              */
/*                                                                      */
/* CALLS         : SELF                                                 */
/*               : dyn_ins_dls(...)                                     */
/*               : dcs_sch_nxt(...)                                     */
/*               : sch_ins_idl(...)                                     */
/*                                                                      */
/* DATABASE      : bt_open(...)                                         */
/*               : bt_close(...)                                        */
/*                                                                      */
/* BACKTRACKING  : determ                                               */
/* -------------------------------------------------------------------- */

dyn_sch_stp( Ipc, Dyn, OldTL, OldPL, OldFin, TL, PL, Fin):-
    bt_open( db_dcs_dyn, c_dyn_dba, BtrSel),
    % insert dynamic levels
    EdbCtl = ec( db_dcs_dyn, s_dyn_chn, BtrSel),
    dyn_ins_dls( Dyn, Ipc, EdbCtl, OldTL, OldPL, OldFin),
    % schedule next admissible task
    dyn_sch_nxt( Ipc, EdbCtl, OldTL, OldPL, TL, PL, OldFin, Fin), !,
    bt_close( db_dcs_dyn, BtrSel).

dyn_sch_stp( _, _, TaskList, [RdyPro|Act], Fin, TaskList, ProcList, Fin):-
    sch_ins_idl( RdyPro, Act, ProcList).

/* -------------------------------------------------------------------- */
/* PREDICATE     : dyn_ins_dls(...)                                     */
/*                                                                      */
/* DESCRIPTION   : Insert <DynScheme> labelled levels with <IPCScheme> using  */
/*               : the external DB control structure <EdbCtl> for the tasks in */
/*               : <TaskList> on the processors <ProcList> at time <FinTime>  */
/*                                                                      */
/* PARAMETERS    : DynScheme                                            */
/*               : IpcScheme                                            */
/*               : EdbCtl                                               */
/*               : TaskList                                             */
/*               : ProcList                                             */
/*               : FinTime                                              */
/*                                                                      */
/* CALLS         : dyn_clc_lvl(...)                                     */
/*               : dyn_gen_lvl(...)                                     */
/*               : dyn_key_int(...)                                     */
/*               : sch_act_tsk(...)                                     */
/*               : sch_ins_ipc(...)                                     */
/*               : sch_non_cor(...)                                     */
/*               : sch_pre_rel(...)                                     */
/*               : sch_pro_rsc(...)                                     */
/*                                                                      */
/* DATABASE      : --                                                   */
/*                                                                      */
/* BACKTRACKING  : determ                                               */
/* -------------------------------------------------------------------- */

dyn_ins_dls( Dyn, Ipc, ec(Dba, Chn, Btr), TaskList, ProcList, FinTime):-
    dyn_gen_lvl( Dyn, TaskList, ProcList, Task, Proc, TL, Act1),
    % decompose task and processor clauses
    Task = tk( TN,_),
    Proc = pr( P, [tk(T,F)|D]),
    % check prec and active tasks
    sch_pre_rel( Task, TL),
    sch_act_tsk( Task, F, Act1),
    % calc earliest availability of Ipc
    sch_non_cor( Task, Act1, NCRs),
    sch_ins_ipc( Ipc, NCRs, Act1, _, FinTime, NewFin, [tk(T,F)|D], IpcTL),
    % check pro resource constraints
    sch_pro_rsc( Task, P, NewFin),
    % calc dynamic level and insert in database
    dyn_clc_lvl( Dyn, Task, F, IpcTL, DynLvl),
    chain_insertz( Dba, Chn, d_edb_dom, dlp(TN, P, DynLvl), Ref),
    % 'inverted' 10's complement representation of DL
    dyn_key_int( Key, DynLvl, c_off_set),
    key_insert( Dba, Btr, Key, Ref),
    fail.
```

```
dyn_ins_dls( _, _, _, _, _, _).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_gen_lvl(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Generate task-processor <Task>-<Proc> pair for <DynScheme> */
/*                 : dynamic labelling scheme                               */
/*                                                                          */
/* PARAMETERS    : DynScheme                                                */
/*                 : OldTaskList                                            */
/*                 : OldProcList                                            */
/*                 : Task                                                   */
/*                 : Proc                                                   */
/*                 : NewTaskList                                            */
/*                 : NewProcList                                            */
/*                                                                          */
/* CALLS         : lst_del_elm(...)                                         */
/*                 : lst_del_ndt(...)                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : non-determ                                               */
/* ------------------------------------------------------------------------ */

dyn_gen_lvl( s_dyn_non, OldTL, OldPL, Task, Proc, NewTL, NewPL):- !,
    % select task with lowest index
    lst_del_elm( Task, OldTL, NewTL),
    % generator predicate
    lst_del_ndt( Proc, OldPL, NewPL).

dyn_gen_lvl( s_dyn_pro, OldTL, OldPL, Task, Proc, NewTL, NewPL):- !,
    % select processor with lowest index
    lst_del_elm( Proc, OldPL, NewPL),
    % generator predicate
    lst_del_ndt( Task, OldTL, NewTL).

dyn_gen_lvl( s_dyn_tsk, OldTL, OldPL, Task, Proc, NewTL, NewPL):- !,
    % select task with lowest index
    lst_del_elm( Task, OldTL, NewTL),
    % generator predicate
    lst_del_ndt( Proc, OldPL, NewPL).

dyn_gen_lvl( s_dyn_all, OldTL, OldPL, Task, Proc, NewTL, NewPL):-
    % generator predicates for all pairings
    lst_del_ndt( Task, OldTL, NewTL),
    lst_del_ndt( Proc, OldPL, NewPL).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_clc_lvl(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <DynLvl> to dynamic level using <DynScheme> labelling */
/*                 : scheme for <Task> on a processor finishing execution at */
/*                 : <ProcFin> with tasks <TaskList> including IPC resources */
/*                                                                          */
/* PARAMETERS    : DynScheme                                                */
/*                 : Task                                                   */
/*                 : ProcFin                                                */
/*                 : TaskList                                               */
/*                 : DynLvl                                                 */
/*                                                                          */
/* CALLS         : clc_max_val(...)                                         */
/*                 : idb_tsk_lvl(...)                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

dyn_clc_lvl( s_dyn_non, tk(TskNam, _), _, _, DynLvl):- !,
    idb_tsk_lvl( TskNam, CpmLvl),
    DynLvl = CpmLvl.

dyn_clc_lvl( _, tk(TskNam, _), ProFin, [tk(_, IpcFin)|_], DynLvl):- !,
    idb_tsk_lvl( TskNam, StaLvl),
    clc_max_val( ProFin, IpcFin, DatAvl),
    DynLvl = StaLvl - DatAvl.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_key_int(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <BtrKey> to 'Inverted' 10's complement representation */
/*                 : of <DynLvl>: required since <DynLvl> may be negative   */
/*                                                                          */
/* PARAMETERS    : BtrKey                                                   */
/*                 : DynLvl                                                 */
/*                 : OffSet                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

dyn_key_int( Key, DynLvl, OffSet):-              % +ve case
    bound( DynLvl),
    free( Key),
    DynLvl > 0, !,
    KeyDynLvl = OffSet - DynLvl,
    str_real( TempKey, KeyDynLvl),
    frontchar( Key, '0', TempKey).

dyn_key_int( Key, DynLvl, OffSet):-              % -ve case
    bound( DynLvl),
    free( Key), !,
    KeyDynLvl = OffSet - DynLvl,
    str_real( Key, KeyDynLvl).

dyn_key_int( Key, DynLvl, OffSet):-              % -ve case
    bound( Key),
    free( DynLvl),
```

```
    str_int( Key, OldDynLvl),
    OldDynLvl > OffSet, !,
    DynLvl =  OffSet - OldDynLvl.

dyn_key_int( Key, DynLvl, OffSet):-              % +ve case
    bound( Key),
    free( DynLvl),
    str_int( Key, OldDynLvl),
    DynLvl = - OldDynLvl + OffSet.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_sch_nxt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create next scheduling step with <IpcScheme>, indexing the */
/*               : dynamic-levels for task-processor pairs with <EdbCtl> and */
/*               : binding <TaskList>, <ProcList> and <Fin> to the result   */
/*                                                                          */
/* PARAMETERS    : IpcScheme                                                */
/*               : EdbCtl                                                   */
/*               : OldTaskList                                              */
/*               : OldProcList                                              */
/*               : TaskList                                                 */
/*               : ProcList                                                 */
/*               : OldFinTime                                               */
/*               : FinTime                                                  */
/*                                                                          */
/* CALLS         : dyn_del_dls(...)                                         */
/*               : idb_tsk_exe(...)                                         */
/*               : lst_del_ndt(...)                                         */
/*               : sch_ins_ipc(...)                                         */
/*               : sch_ins_tsk(...)                                         */
/*               : sch_non_cor(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

dyn_sch_nxt( Ipc, ec(DbaSel, ChnNam, BtrSel), OldTL, OldPL, TL, PL, Fin1, Fin):-
    % Get lowest ready task, available processor pairing
    key_first( DbaSel, BtrSel, Ref), !,
    ref_term( DbaSel, d_edb_dom, Ref, dlp( TaskName, P, _)),
    % get task and processor
    lst_del_ndt( Task, OldTL, TL),
    Task = tk( TaskName, _), !,
    lst_del_ndt( Proc, OldPL, Act1),
    Proc = pr( P, [tk(T,F)|D]), !,
    % allocate IPC resources
    sch_non_cor( Task, Act1, NCRs),
    sch_ins_ipc( Ipc, NCRs, Act1, Act2, Fin1, Fin2, [tk(T,F)|D], IpcTL),
    % calc task finish time
    IpcTL = [tk(_, IpcF)|_],
    idb_tsk_exe( TaskName, ExecTime),
    Time = IpcF + ExecTime,
    % allocate task
    sch_ins_tsk( pr(P,[tk(TaskName,Time)|IpcTL]), Act2, PL, Fin2, Fin),
    % delete all DLs from database
    dyn_del_dls( ec(DbaSel, ChnNam, BtrSel)).

dyn_sch_nxt( _, ec(DbaSel, ChnNam, BtrSel), _, _, _, _, _, _):-
    % ensuse idle time is scheduled
    dyn_del_dls( ec(DbaSel, ChnNam, BtrSel)),
    bt_close( DbaSel, BtrSel),
    fail.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_del_dls(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Delete all task-processor dynamic levels from the external */
/*               : database chain and btree referenced by <EdbCtl>          */
/*                                                                          */
/* PARAMETERS    : EdbCtl                                                   */
/*                                                                          */
/* CALLS         : dyn_ddd_aux(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

dyn_del_dls( ec(DbaSel, ChnNam, BtrSel)):-
    key_first( DbaSel, BtrSel, _), !,
    dyn_ddd_aux( ec(DbaSel, ChnNam, BtrSel)).

dyn_del_dls( _).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : dyn_ddd_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for dyn_del_dls(...)                 */
/*                                                                          */
/* PARAMETERS    : EdbCtl                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

dyn_ddd_aux( ec(DbaSel, ChnNam, BtrSel)):-
    % Get Key for current Ref
    key_current( DbaSel, BtrSel, ThisKey, ThisRef),
    % Delete Key from btree and term from chain
    key_delete( DbaSel, BtrSel, ThisKey, ThisRef),
    term_delete( DbaSel, ChnNam, ThisRef),
    fail.

dyn_ddd_aux( ec(DbaSel, ChnNam, BtrSel)):-
    key_next( DbaSel, BtrSel, _), !,
    dyn_ddd_aux( ec(DbaSel, ChnNam, BtrSel)).
```

```
dyn_ddd_aux( _).

/* ************************************************************************** */
```

D.4.5 dcs_edb.pro

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_EDB.PRO                                                    */
/*                                                                            */
/* NOTES     : Display predicates for external database chains and btrees     */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 20th May, 1995                                                 */
/*                                                                            */
/* VERSION   : 2.5                                                            */
/* ************************************************************************** */

/* ========================================================================== */
/* Project definition file                                                    */
/* ========================================================================== */

project "dcs.prj"

/* ========================================================================== */
/* Predicate definition files                                                 */
/* ========================================================================== */

include "dcs_glb.pub"

include "dcs_edb.pub"
include "dcs_dyn.pub"

/* ========================================================================== */
/* Public definition files                                                    */
/* ========================================================================== */

include "dcs_edb.prv"

/* ========================================================================== */
   clauses
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : edb_dba_dsp(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Display contents of external DB <DbaSel> on <OutDev>       */
/*                                                                            */
/* PARAMETERS    : DbaSel                                                     */
/*               : OutDev                                                     */
/*                                                                            */
/* CALLS         : edb_edd_aux(...)                                           */
/*                                                                            */
```

```
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

edb_dba_dsp( DbaSel, OutDev):-
    writedevice( OrgDev),
    writedevice( OutDev),
    write( "=====================================================\n"),
    write( "Database : ", DbaSel, "\n"),
    write( "=====================================================\n"),
    edb_edd_aux( DbaSel),
    write( "=====================================================\n\n"),
    writedevice( OrgDev).

/* -------------------------------------------------------------------- */
/* PREDICATE    : edb_edd_aux(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for edb_dba_dsp(...)              */
/*                                                                      */
/* PARAMETERS   : DbaSel                                                */
/*              : OutDev                                                */
/*                                                                      */
/* CALLS        : edb_wri_chn(...)                                      */
/*                                                                      */
/* DATABASE     : db_chains(...)                                        */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

edb_edd_aux( DbaSel):-
    % check if there are any chains in the EDB
    db_chains( DbaSel, _), !,
    edb_wri_chn( DbaSel).

edb_edd_aux( _):-
    write( "\tNo chains in database\n").

/* -------------------------------------------------------------------- */
/* PREDICATE    : edb_wri_chn(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Write all chains held in the external DB <DbaSel>     */
/*                                                                      */
/* PARAMETERS   : DbaSel                                                */
/*              : OutDev                                                */
/*                                                                      */
/* CALLS        : edb_all_trm(...)                                      */
/*                                                                      */
/* DATABASE     : db_chains(...)                                        */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

edb_wri_chn( DbaSel):-
    db_chains( DbaSel, ChnNam),
    write( "        ----------------------------------------------\n"),
    write( "        Chain : ", ChnNam, "\n"),
    write( "        ----------------------------------------------\n"),
    edb_all_trm( DbaSel, ChnNam),
    write( "        ----------------------------------------------\n"),
    fail.

edb_wri_chn( _).

/* -------------------------------------------------------------------- */
/* PREDICATE    : edb_all_trm(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Write all terms held in the chain <ChnNam> in <DbaSel> */
/*                                                                      */
/* PARAMETERS   : DbaSel                                                */
/*              : OutDev                                                */
/*                                                                      */
/* CALLS        : edb_wri_trm(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

edb_all_trm( DbaSel, ChnNam):-
    chain_terms( DbaSel, ChnNam, d_edb_dom, Term, Ref),
    edb_wri_trm( Term, Ref),
    fail.

edb_all_trm( _, _).

/* -------------------------------------------------------------------- */
/* PREDICATE    : edb_wri_trm(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Write external DB term <ThisTerm> with reference <ThisRef> */
/*                                                                      */
/* PARAMETERS   : ThisTerm                                              */
/*              : ThisRef                                               */
/*                                                                      */
/* CALLS        : --                                                    */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

edb_wri_trm( dlp( Task, Proc, DL), ThisRef):- !,
    write( "\t\tRef : ", ThisRef, "\n"),
    write( "\t\t\tTask : ", Task, "\n"),
    write( "\t\t\tProc : ", Proc, "\n"),
    write( "\t\t\tDL   : ", DL,   "\n").

edb_wri_trm( _, ThisRef):-
    write( "\t\tRef : ", ThisRef, "\n"),
    write( "\t\t\t** Unknown term\n").

/* -------------------------------------------------------------------- */
```

```
/* PREDICATE     : edb_btr_dsp(...)                                          */
/*                                                                           */
/* DESCRIPTION   : Write contents of all Btrees held in the external DB      */
/*               : <DbaSel> to the device <OutDev>                           */
/*                                                                           */
/* PARAMETERS    : DbaSel                                                    */
/*               : OutDev                                                    */
/*                                                                           */
/* CALLS         : edb_ebd_aux(...)                                          */
/*                                                                           */
/* DATABASE      : --                                                        */
/*                                                                           */
/* BACKTRACKING  : determ                                                    */
/* ------------------------------------------------------------------------- */

edb_btr_dsp( DbaSel, OutDev):-
    writedevice( OrgDev),
    writedevice( OutDev),
    write( "=========================================================\n"),
    write( "Database : ", DbaSel, "\n"),
    write( "=========================================================\n"),
    edb_ebd_aux( DbaSel),
    write( "=========================================================\n\n"),
    writedevice( OrgDev).

/* ------------------------------------------------------------------------- */
/* PREDICATE     : edb_ebd_aux(...)                                          */
/*                                                                           */
/* DESCRIPTION   : Auxiliary predicate for edb_btr_dsp(...)                  */
/*                                                                           */
/* PARAMETERS    : DbaSel                                                    */
/*                                                                           */
/* CALLS         : edb_wri_btr(...)                                          */
/*                                                                           */
/* DATABASE      : db_btrees(...)                                            */
/*                                                                           */
/* BACKTRACKING  : determ                                                    */
/* ------------------------------------------------------------------------- */

edb_ebd_aux( DbaSel):-
    % check if there are any btrees in the EDB
    db_btrees( DbaSel, _), !,
    edb_wri_btr( DbaSel).

edb_ebd_aux( _):-
    write( "\tNo btrees in database\n").

/* ------------------------------------------------------------------------- */
/* PREDICATE     : edb_wri_btr(...)                                          */
/*                                                                           */
/* DESCRIPTION   : Write contents of all Btrees in the external DB <DbaSel>  */
/*                                                                           */
/* PARAMETERS    : DbaSel                                                    */
/*                                                                           */
/* CALLS         : edb_all_key(...)                                          */
/*                                                                           */
/* DATABASE      : db_btrees(...)                                            */
/*                                                                           */
/* BACKTRACKING  : determ                                                    */
/* ------------------------------------------------------------------------- */

edb_wri_btr( DbaSel):-
    db_btrees( DbaSel, BtrNam),
    write( "        ----------------------------------------------\n"),
    write( "        Btree : ", BtrNam, "\n"),
    write( "        ----------------------------------------------\n"),
    edb_all_key( DbaSel, BtrNam),
    write( "        ----------------------------------------------\n"),
    fail.

edb_wri_btr( _).

/* ------------------------------------------------------------------------- */
/* PREDICATE     : edb_all_key(...)                                          */
/*                                                                           */
/* DESCRIPTION   : Write contents of Btree <BtrNam> in external DB <DbaSel>  */
/*                                                                           */
/* PARAMETERS    : DbaSel                                                    */
/*               : BtrNam                                                    */
/*                                                                           */
/* CALLS         : edb_btr_chk(...)                                          */
/*               : edb_wri_key(...)                                          */
/*                                                                           */
/* DATABASE      : bt_open(...)                                              */
/*               : bt_close(...)                                             */
/*                                                                           */
/* BACKTRACKING  : determ                                                    */
/* ------------------------------------------------------------------------- */

edb_all_key( DbaSel, BtrNam):-
    bt_open( DbaSel, BtrNam, BtrSel),
    edb_btr_chk( DbaSel, BtrSel, FirstRef), !,
    edb_wri_key( DbaSel, BtrSel, FirstRef),
    bt_close( DbaSel, BtrSel).

edb_all_key( _, _).

/* ------------------------------------------------------------------------- */
/* PREDICATE     : edb_btr_chk(...)                                          */
/*                                                                           */
/* DESCRIPTION   : Check if there are any keys in the Btree <BtrSel> in the  */
/*               : external DB <DbaSel>. If so bind <FirstRef> to the first  */
/*               : reference number in the DB                                */
/*                                                                           */
/* PARAMETERS    : DbaSel                                                    */
/*               : BtrSel                                                    */
/*               : FirstRef                                                  */
/*                                                                           */
/* CALLS         : --                                                        */
/*                                                                           */
/* DATABASE      : bt_close(...)                                             */
/*                                                                           */
```

```
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

edb_btr_chk( DbaSel, BtrSel, FirstRef):-
    % check if there are any keys in btree
    key_first( DbaSel, BtrSel, FirstRef), !.

edb_btr_chk( DbaSel, BtrSel, _):-
    write( "\t\tNo keys in btree\n"),
    % btree was empty so make sure its closed
    bt_close( DbaSel, BtrSel),
    % now fail cleanly after btree is closed
    fail.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : edb_wri_key(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Write term corresponding to <ThisRef> in the Btree <BtrSel> */
/*              : held in the external DB <DbaSel>                          */
/*                                                                          */
/* PARAMETERS   : DbaSel                                                    */
/*              : BtrSel                                                    */
/*              : ThisRef                                                   */
/*                                                                          */
/* CALLS        : edb_wri_key(...)                                          */
/*              : edb_wri_trm(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

edb_wri_key( DbaSel, BtrSel, ThisRef):-
    ref_term( DbaSel, d_edb_dom, ThisRef, ThisTerm),
    key_current( DbaSel, BtrSel, Key, _),
    write( "\t\tKey : ", Key, "\n"),
    edb_wri_trm( ThisTerm, ThisRef),
    fail.

edb_wri_key( DbaSel, BtrSel, _):-
    key_next( DbaSel, BtrSel, NextRef), !,
    edb_wri_key( DbaSel, BtrSel, NextRef).

edb_wri_key( DbaSel, _, _):-
    db_flush( DbaSel).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : edb_dba_sts(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Write statistics from external DB <DbaSel> to <OutDev>    */
/*                                                                          */
/* PARAMETERS   : DbaSel                                                    */
/*              : OutDev                                                    */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : db_statistics(...)                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

edb_dba_sts( DbaSel, OutDev):-
    db_statistics( DbaSel, NumTerms, MemSize, DbaSize, FreeSize),
    writedevice( OrgDev),
    writedevice( OutDev),
    write( "=====================================================\n"),
    write( "Database : ", DbaSel, "\n"),
    write( "=====================================================\n"),
    write( "\tNo. of Terms  : ", NumTerms, "\n"),
    write( "\tMemory Size   : ", MemSize,   "\n"),
    write( "\tDatabase Size : ", DbaSize, "\n"),
    write( "\tFree Size     : ", FreeSize,  "\n"),
    write( "=====================================================\n\n"),
    writedevice( OrgDev).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : edb_btr_sts(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Write statistics of all btrees in external DB <DbaSel> to */
/*              : <OutDev>                                                  */
/*                                                                          */
/* PARAMETERS   : DbaSel                                                    */
/*              : OutDev                                                    */
/*                                                                          */
/* CALLS        : edb_ebs_aux(...)                                          */
/*                                                                          */
/* DATABASE     : db_btrees(...)                                            */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

edb_btr_sts( DbaSel, OutDev):-
    db_btrees( DbaSel, BtrNam),
    edb_ebs_aux( DbaSel, BtrNam, OutDev),
    fail.

edb_btr_sts( _, _).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : edb_ebs_aux(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Auxiliary predicate for edb_btr_sts(...)                  */
/*                                                                          */
/* PARAMETERS   : DbaSel                                                    */
/*              : BtrNam                                                    */
/*              : OutDev                                                    */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : bt_open(...)                                              */
/*              : bt_statistics(...)                                        */
/*              : bt_close(...)                                             */
```

```
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

edb_ebs_aux( DbaSel, BtrNam, OutDev):-
    bt_open( DbaSel, BtrNam, BtrSel),
    bt_statistics( DbaSel, BtrSel, NoOfKeys, NoOfPages,
        Depth, KeyLen, Order, PageSize),
    writedevice( OrgDev),
    writedevice( OutDev),
    write( "=========================================================\n"),
    write( "Btree : ", BtrNam, "\n"),
    write( "=========================================================\n"),
    write( "\tNo. of Keys  : ", NoOfKeys,  "\n"),
    write( "\tNo. of Pages : ", NoOfPages, "\n"),
    write( "\tDepth        : ", Depth,     "\n"),
    write( "\tKey Length   : ", KeyLen,    "\n"),
    write( "\tOrder        : ", Order,     "\n"),
    write( "\tPageSize     : ", PageSize,  "\n"),
    write( "=========================================================\n\n"),
    writedevice( OrgDev),
    bt_close( DbaSel, BtrSel).

/* ---------------------------------------------------------------------- */
/* PREDICATE    : edb_mem_sts(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write memory usage statistics to <OutDev> and bind      */
/*              : <HeapSize> to size of current heap                      */
/*                                                                        */
/* PARAMETERS   : HeapSize                                                */
/*              : OutDev                                                  */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

edb_mem_sts( HeapSize, OutDev):-
    writedevice( OrgDev),
    writedevice( OutDev),
    storage( StackSize, HeapSize, TrailSize),
    write( "=========================================================\n"),
    write( "Memory Usage :\n"),
    write( "=========================================================\n"),
    write( "\tStack Size : ", StackSize, "\n"),
    write( "\tHeap Size  : ", HeapSize,  "\n"),
    write( "\tTrail Size : ", TrailSize, "\n"),
    write( "=========================================================\n\n"),
    writedevice( OrgDev).

/* ********************************************************************** */
```

D.4.6 dcs_idb.pro

```
/* ********************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ********************************************************************** */

/* ********************************************************************** */
/* MODULE    : DCS_IDB.PRO                                                */
/*                                                                        */
/* NOTES     : Access predicates for facts held in internal database      */
/*                                                                        */
/* AUTHOR    : Ken N LINTON                                               */
/*                                                                        */
/* DATE      : 14th May, 1995                                             */
/*                                                                        */
/* VERSION   : 2.2                                                        */
/* ********************************************************************** */

/* ====================================================================== */
/* Project definition                                                     */
/* ====================================================================== */

project "dcs.prj"

/* ====================================================================== */
/* Public predicate definitions                                           */
/* ====================================================================== */

include "dcs_glb.pub"

include "dcs_idb.pub"

/* ====================================================================== */
   clauses
/* ====================================================================== */

/* ---------------------------------------------------------------------- */
/* PREDICATE    : idb_sys_tim(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Assert system time into internal database               */
/*                                                                        */
/* PARAMETERS   : --                                                      */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : assert( dbatime(...))                                   */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

idb_sys_tim():-
    time( Hours, Minutes, Seconds, Cents),
    assert( dbatime( Hours, Minutes, Seconds, Cents)).
```

```
/* ------------------------------------------------------------------------ */
/* PREDICATE     : idb_tsk_exe(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Non-destructive retract of <ExecTime> of <TaskName>      */
/*                                                                          */
/* PARAMETERS    : TaskName                                                 */
/*               : ExecTime                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : retract( task(...))                                      */
/*               : assertz( task(...))                                      */
/*               : retract( tasktype(...))                                  */
/*               : assertz( tasktype(...))                                  */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

idb_tsk_exe( TaskName, ExecTime):-
    retract( task( TaskName, TaskType)), !,
    assertz( task( TaskName, TaskType)),
    retract( tasktype( TaskType, ExecTime, Inputs, Outputs)), !,
    assertz( tasktype( TaskType, ExecTime, Inputs, Outputs)).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : idb_tsk_lvl(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Non-destructive retract of <Level> of <TaskName>         */
/*                                                                          */
/* PARAMETERS    : TaskName                                                 */
/*               : Level                                                    */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : retract( level(...))                                     */
/*               : assertz( level(...))                                     */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

idb_tsk_lvl( TaskName, Level):-
    retract( level( TaskName, Level)), !,
    assertz( level( TaskName, Level)).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : idb_tsk_com(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Non-destructive succeed if <TaskName1> communicates      */
/*               : directly with <TaskName2>                                */
/*                                                                          */
/* PARAMETERS    : TaskName1                                                */
/*               : TaskName2                                                */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : retract( prec(...))                                      */
/*               : assertz( prec(...))                                      */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

idb_tsk_com( TaskName1, TaskName2):-
    retract( prec( TaskName1, TaskName2)), !,
    assertz( prec( TaskName1, TaskName2)).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : idb_pre_rel(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Non-destructive succeed if <TaskName1> < <TaskName2>     */
/*                                                                          */
/* PARAMETERS    : TaskName1                                                */
/*               : TaskName2                                                */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*                                                                          */
/* DATABASE      : retract( prec(...))                                      */
/*               : assertz( prec(...))                                      */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

idb_pre_rel( TaskName1, TaskName2):-
    retract( prec( TaskName1, TaskName2)), !,
    assertz( prec( TaskName1, TaskName2)).

idb_pre_rel( TaskName1, TaskName2):-
    retract( prec( Task, TaskName2)),  !,
    assertz( prec( Task, TaskName2)),
    % catch cycles through SP-DIF interface
    Task <> "extconnector",
    idb_pre_rel( TaskName1, Task).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : idb_pro_exe(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Non-destructive retract of <ExecRsrc> for <ProcName>     */
/*                                                                          */
/* PARAMETERS    : ProcName                                                 */
/*               : ExecRsrc                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : retract( proc(...))                                      */
/*               : assertz( proc(...))                                      */
/*               : retract( proctype(...))                                  */
/*               : assertz( proctype(...))                                  */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

idb_pro_exe( ProcName, ProcRsrc):-
```

```
    retract( proc( ProcName, ProcType)),   !,
    assertz( proc( ProcName, ProcType)),
    retract( proctype( ProcType, ProcRsrc)), !,
    assertz( proctype( ProcType, ProcRsrc)).

/* ************************************************************************** */
```

D.4.7 dcs_inf.pro

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_INF.PRO                                                    */
/*                                                                            */
/* NOTES     : User interface and top-level predicates for DCS application    */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 20th May, 1995                                                 */
/*                                                                            */
/* VERSION   : 4.5                                                            */
/* ************************************************************************** */

/* ========================================================================== */
/* Project definition file                                                    */
/* ========================================================================== */

project "dcs.prj"

/* ========================================================================== */
/* Public definition files                                                    */
/* ========================================================================== */

include "dcs_glb.pub"

include "dcs_ast.pub"
include "dcs_clc.pub"
include "dcs_cpm.pub"
include "dcs_dyn.pub"
include "dcs_idb.pub"
include "dcs_lst.pub"
include "dcs_trl.pub"
include "dcs_utl.pub"
include "dcs_wri.pub"

/* ========================================================================== */
/* Private definition files                                                   */
/* ========================================================================== */

include "dcs_inf.prv"

/* ========================================================================== */
   goal
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE    : --                                                          */
/*                                                                            */
```

```
/* DESCRIPTION  : Goal for DCS executable                                  */
/*                                                                         */
/* PARAMETERS   : --                                                       */
/*              : --                                                       */
/*                                                                         */
/* CALLS        : inf_run_dcs(...)                                         */
/*                                                                         */
/* DATABASE     : --                                                       */
/*                                                                         */
/* BACKTRACKING : determ                                                   */
/* ----------------------------------------------------------------------- */

inf_run_dcs().

/* ======================================================================= */
   clauses
/* ======================================================================= */

/* ----------------------------------------------------------------------- */
/* PREDICATE    : inf_run_dcs(...)                                         */
/*                                                                         */
/* DESCRIPTION  : Run DCS digital console scheduler, first clearing the    */
/*              : internal database and enabling ctrl-break interrupts     */
/*                                                                         */
/* PARAMETERS   : --                                                       */
/*                                                                         */
/* CALLS        : inf_get_inp(...)                                         */
/*              : inf_mon_stt(...)                                         */
/*              : inf_mon_end(...)                                         */
/*              : inf_sch_eng(...)                                         */
/*              : inf_sho_sch(...)                                         */
/*              : inf_sav_sch(...)                                         */
/*                                                                         */
/* DATABASE     : retractall(...)                                          */
/*                                                                         */
/* BACKTRACKING : determ                                                   */
/* ----------------------------------------------------------------------- */

inf_run_dcs():-
    % clear internal DB and enable ctrl-break
    system( "cls"),
    retractall( _),
    break( on),
    % get inputs from user
    inf_get_inp( Taskforce, Hardware, AlgCtl),
    % start algorithm monitor and then scheduling engine
    inf_mon_stt( OldDev),
    inf_sch_eng( Taskforce, Hardware, AlgCtl),
    inf_mon_end( OldDev),
    % show scheduler output and save to disk
    inf_sho_sch( RtrnStat),
    inf_sav_sch( RtrnStat).

/* ----------------------------------------------------------------------- */
/* PREDICATE    : inf_mon_stt(...)                                         */
/*                                                                         */
/* DESCRIPTION  : Create monitor window & bind <OldDev> to old output device */
/*                                                                         */
/* PARAMETERS   : OldDev                                                   */
/*                                                                         */
/* CALLS        : --                                                       */
/*                                                                         */
/* DATABASE     : --                                                       */
/*                                                                         */
/* BACKTRACKING : determ                                                   */
/* ----------------------------------------------------------------------- */

inf_mon_stt( OldDev):-
    textmode( Rows, _),
    writedevice( OldDev),
    MonWinTop = (Rows div 2) - 1,
    makewindow( c_win_mon, c_win_frg, c_win_bkg, " DCS Algorithm Monitor ",
        MonWinTop, c_win_idt, 3, c_win_wid),
    writedevice( screen).

/* ----------------------------------------------------------------------- */
/* PREDICATE    : inf_mon_end(...)                                         */
/*                                                                         */
/* DESCRIPTION  : Return to <OldDev> output device and remove monitor window */
/*                                                                         */
/* PARAMETERS   : OldDev                                                   */
/*                                                                         */
/* CALLS        : --                                                       */
/*                                                                         */
/* DATABASE     : --                                                       */
/*                                                                         */
/* BACKTRACKING : determ                                                   */
/* ----------------------------------------------------------------------- */

inf_mon_end( OldDev):-
    writedevice( OldDev),
    removewindow( c_win_mon, c_win_ref).

/* ----------------------------------------------------------------------- */
/* PREDICATE    : inf_sch_eng(...)                                         */
/*                                                                         */
/* DESCRIPTION  : Scheduling engine to populate internal database, create  */
/*              : task, precedence and processor lists and run algorithm   */
/*              : to schedule <Taskforce> on <Hardware> with <AlgCtl> control */
/*                                                                         */
/* PARAMETERS   : Taskforce                                                */
/*              : Hardware                                                 */
/*              : AlgCtl                                                   */
/*                                                                         */
/* CALLS        : inf_ctp_tst(...)                                         */
/*              : inf_fit_tst(...)                                         */
/*              : inf_grn_tst(...)                                         */
/*              : inf_tst_tst(...)                                         */
/*              : inf_bgn_sch(...)                                         */
/*              : utl_cre_pre(...)                                         */
/*              : utl_cre_pro(...)                                         */
/*              : utl_cre_tsk(...)                                         */
```

```
/*                : wri_inp_prm(...)                                        */
/*                : wri_mtp_sch(...)                                        */
/*                : wri_pre_stt(...)                                        */
/*                : wri_pre_chk(...)                                        */
/*                : wri_pst_stt(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

inf_sch_eng( Taskforce, Hardware, AlgCtl):-
    % open scheduler output file
    inf_opn_sch( c_sch_dir, c_sch_nam, f_sch_out),
    % create tasks, precedence and processor lists
    inf_cre_lst( Taskforce, Hardware, TaskList, PrecList, ProcList),
    % write pre-schedule info
    wri_inp_prm( Taskforce, Hardware, AlgCtl),
    wri_pre_stt( TaskList, PrecList, ProcList),
    % write pre-schedule test results
    inf_pre_tst( TaskList, PrecList, ProcList, TestList),
    wri_pre_chk( TestList),
    % check if any test failed
    inf_tst_tst( TestList),
    % initialise watch-dog and begin scheduler
    inf_ini_wat( AlgCtl),
    inf_bgn_sch( AlgCtl, TaskList, PrecList, ProcList, Schedule), !,
    % write M-P schedule
    wri_mtp_sch( Schedule),
    % write post-schedule statistics to output
    wri_pst_stt( AlgCtl, TaskList, Schedule),
    closefile( f_sch_out).

inf_sch_eng( _, _, _):-
    % catch non-feasible schedule
    wri_mtp_sch( []),
    closefile( f_sch_out).

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_opn_sch(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Open the file <SchNam> in the directory <SchDir> on the */
/*                : the symbolic file <SymFil>                              */
/*                                                                          */
/* PARAMETERS     : SchDir                                                  */
/*                : SchNam                                                  */
/*                : SymFil                                                  */
/*                                                                          */
/* CALLS          : --                                                      */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

inf_opn_sch( SchDir, SchNam, SymFil):-
    concat( SchDir, c_pth_dlm, SchPth),
    concat( SchPth, SchNam, SchFil),
    openwrite( SymFil, SchFil),
    writedevice( SymFil).

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_cre_lst(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Consult the <Taskforce> and <Hardware> descriptions and */
/*                : populate the <TaskList>, <PrecList> and <ProcList> lists */
/*                                                                          */
/* PARAMETERS     : Taskforce                                               */
/*                : Hardware                                                */
/*                : TaskList                                                */
/*                : PrecList                                                */
/*                : ProcList                                                */
/*                                                                          */
/* CALLS          : utl_cre_pre(...)                                        */
/*                : utl_cre_pro(...)                                        */
/*                : utl_cre_tsk(...)                                        */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

inf_cre_lst( Taskforce, Hardware, TaskList, PrecList, ProcList):-
    % create tasks & precedence lists
    consult( Taskforce),
    utl_cre_tsk( TaskList),
    utl_cre_pre( PrecList),
    % create processor list
    consult( Hardware),
    utl_cre_pro( ProcList).

/* ------------------------------------------------------------------------ */
/* PREDICATE      : inf_pre_tst(...)                                        */
/*                                                                          */
/* DESCRIPTION    : Perform pre-schedule tests and populate <TestList>      */
/*                                                                          */
/* PARAMETERS     : TaskList                                                */
/*                : PrecList                                                */
/*                : ProcList                                                */
/*                : TestList                                                */
/*                                                                          */
/* CALLS          : --                                                      */
/*                                                                          */
/* DATABASE       : --                                                      */
/*                                                                          */
/* BACKTRACKING   : determ                                                  */
/* ------------------------------------------------------------------------ */

inf_pre_tst( TaskList, PrecList, ProcList, [GranTest, FitTest, CritTest]):-
    inf_grn_tst( TaskList, ProcList, GranTest),
    inf_fit_tst( TaskList, ProcList, FitTest),
    inf_ctp_tst( PrecList, ProcList, CritTest).
```

```
/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_bgn_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Begin scheduling algorithm, recording system clock       */
/*               : immediately before and after                             */
/*                                                                          */
/* PARAMETERS    : AlgCtl                                                   */
/*               : TaskList                                                 */
/*               : PrecList                                                 */
/*               : ProcList                                                 */
/*               : Schedule                                                 */
/*                                                                          */
/* CALLS         : ast_run_alg(...)                                         */
/*               : cpm_run_alg(...)                                         */
/*               : dyn_run_alg(...)                                         */
/*               : idb_sys_tim(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

inf_bgn_sch( AlgCtl, TaskList, PrecList, ProcList, Schedule):-
    AlgCtl = ac(s_alg_cpm, Ipc, Lbl, _, _), !,
    idb_sys_tim(),
    cpm_bgn_alg( Ipc, Lbl, TaskList, PrecList, ProcList, Schedule),
    idb_sys_tim().

inf_bgn_sch( AlgCtl, TaskList, PrecList, ProcList, Schedule):-
    AlgCtl = ac(s_alg_dyn, Ipc, Lbl, Dyn, _), !,
    idb_sys_tim(),
    dyn_bgn_alg( Ipc, Lbl, Dyn, TaskList, PrecList, ProcList, Schedule),
    idb_sys_tim().

inf_bgn_sch( AlgCtl, TaskList, PrecList, ProcList, Schedule):-
    AlgCtl = ac(s_alg_ast, Ipc, _, _, Hrt), !,
    idb_sys_tim(),
    ast_bgn_alg( Ipc, Hrt, TaskList, PrecList, ProcList, Schedule),
    idb_sys_tim().

inf_bgn_sch( _, _, _, _, _):-
    beep,
    fail.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_get_inp(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Get taskforce file <TkfFil>, hardware file <HdwFil> and  */
/*               : algorithm control parameters <AlgCtl> from user          */
/*                                                                          */
/* PARAMETERS    : TkfFil                                                   */
/*               : HdwFil                                                   */
/*               : AlgCtl                                                   */
/*                                                                          */
/* CALLS         : inf_nul_inp(...)                                         */
/*               : inf_get_tkf(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

inf_get_inp( TkfFil, HdwFil, AlgCtl):-
    makewindow( c_win_one, c_win_frg, c_win_bkg, " DCS Input Parameters ",
        0, c_win_idt, c_win_hig, c_win_wid),
    clearwindow,
    disk( Dir),
    inf_nul_inp( NulInp),
    inf_get_tkf( Dir, "*.dba", NulInp, ai(TkfFil, HdwFil), AlgCtl), !,
    removewindow( c_win_one, c_win_ref).

inf_get_inp( _, _, _):-
    removewindow( c_win_one, c_win_ref),
    removewindow( c_win_two, c_win_ref),
    system( "cls"),
    fail.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_get_tkf(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Using directory <Dir> and filename extension <Ext>, get  */
/*               : taskforce filname <Tkf> and then satisfy inf_get_hdw(...)*/
/*                                                                          */
/* PARAMETERS    : Dir                                                      */
/*               : Ext                                                      */
/*               : OldInp                                                   */
/*               : Inp                                                      */
/*               : Ctl                                                      */
/*                                                                          */
/* CALLS         : inf_get_hdw(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

inf_get_tkf( Dir, Ext, ai(_, Hdw), Inp, Ctl):-
    write( "console taskforce       : "),
    makewindow( c_win_two, c_win_frg, c_win_bkg, " Console Taskforces ",
        c_two_org, c_win_idt, c_win_hig, c_win_wid),
    concat( c_pth_dlm, c_tkf_dir, TkfDir),
    concat( Dir, TkfDir, TkfPth),
    dir( TkfPth, Ext, UprTkf, 0, 0, 0), !,
    upper_lower( UprTkf, Tkf),
    removewindow( c_win_two, c_win_ref),
    write( Tkf), nl,
    inf_get_hdw( Dir, Ext, ai(Tkf, Hdw), Inp, Ctl).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_get_hdw(...)                                         */
/*                                                                          */
```

```
/* DESCRIPTION  : Using directory <Dir> and filename extension <Ext>, get     */
/*              : hardware filname <Tkf> and then satisfy inf_get_alg(...)    */
/*                                                                            */
/* PARAMETERS   : Dir                                                         */
/*              : Ext                                                         */
/*              : OldInp                                                      */
/*              : Inp                                                         */
/*              : Ctl                                                         */
/*                                                                            */
/* CALLS        : inf_get_alg(...)                                            */
/*              : inf_get_tkf(...)                                            */
/*              : inf_nul_ctl(...)                                            */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_get_hdw( Dir, Ext, ai(Tkf, _), Inp, Ctl):-
    write( "M-P architecture        : "),
    makewindow( c_win_two, c_win_frg, c_win_bkg, " M-P Architectures ",
        c_two_org, c_win_idt, c_win_hig, c_win_wid),
    concat( c_pth_dlm, c_hdw_dir, HdwDir),
    concat( Dir, HdwDir, HdwPth),
    dir( HdwPth, Ext, UprHdw, 0, 0, 0), !,
    upper_lower( UprHdw, Hdw),
    removewindow( c_win_two, c_win_ref),
    write( Hdw), nl, nl,
    inf_nul_ctl( NulCtl),
    inf_get_alg( NulCtl, Ctl, Dir, Ext, ai(Tkf, Hdw), Inp).

inf_get_hdw( Dir, Ext, OldInp, Inp, Ctl):-
    removewindow( c_win_two, c_win_ref),
    clearwindow(),
    inf_get_tkf( Dir, Ext, OldInp, Inp, Ctl).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : inf_nul_inp(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Bind <AlgInp> to null algorithm input domain                */
/*                                                                            */
/* PARAMETERS   : AlgInp                                                      */
/*                                                                            */
/* CALLS        : --                                                          */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_nul_inp( ai(c_nul_str, c_nul_str)).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : inf_nul_ctl(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Bind <AlgCtl> to null algorithm control domain              */
/*                                                                            */
/* PARAMETERS   : AlgCtl                                                      */
/*                                                                            */
/* CALLS        : --                                                          */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_nul_ctl( ac(s_alg_nul, s_ipc_nul, s_lbl_nul, s_dyn_nul, s_hrt_nul)).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : inf_mnu_rtn(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Return item <ItmCod> from the user interface menu <MnuCod>  */
/*                                                                            */
/* PARAMETERS   : MnuCod                                                      */
/*              : ItmCod                                                      */
/*                                                                            */
/* CALLS        : inf_mnu_chk(...)                                            */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_mnu_rtn( MnuCod, ItmCod):-
    readchar( UsrChr),
    inf_mnu_chk( MnuCod, UsrChr, ItmCod),
    removewindow( c_win_two, c_win_ref).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : inf_mnu_chk(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Check <UsrChr> is a valid <ItmCod> for the menu <MnuCod>    */
/*                                                                            */
/* PARAMETERS   : MnuCod                                                      */
/*              : UsrChr                                                      */
/*              : ItmCod                                                      */
/*                                                                            */
/* CALLS        : inf_mnu_rtn(...)                                            */
/*              : inf_vld_alg(...)                                            */
/*              : inf_vld_ipc(...)                                            */
/*              : inf_vld_lbl(...)                                            */
/*              : inf_vld_dyn(...)                                            */
/*              : inf_vld_hrt(...)                                            */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_mnu_chk( _, c_key_esc, _):- !,
    fail.
```

```
inf_mnu_chk( s_mnu_alg, UsrChr, ItmCod):-
    inf_vld_alg( UsrChr, ItmCod), !.

inf_mnu_chk( s_mnu_ipc, UsrChr, ItmCod):-
    inf_vld_ipc( UsrChr, ItmCod), !.

inf_mnu_chk( s_mnu_lbl, UsrChr, ItmCod):-
    inf_vld_lbl( UsrChr, ItmCod), !.

inf_mnu_chk( s_mnu_dyn, UsrChr, ItmCod):-
    inf_vld_dyn( UsrChr, ItmCod), !.

inf_mnu_chk( s_mnu_hrt, UsrChr, ItmCod):-
    inf_vld_hrt( UsrChr, ItmCod), !.

inf_mnu_chk( MnuCod, _, ItmCod):-
    inf_mnu_rtn( MnuCod, ItmCod).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : inf_get_alg(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Get scheduling algorithm and then satisfy inf_get_ipc(...) */
/*                                                                          */
/* PARAMETERS   : OldCtl                                                    */
/*              : Ctl                                                       */
/*              : Dir                                                       */
/*              : Ext                                                       */
/*              : OldInp                                                    */
/*              : Inp                                                       */
/*                                                                          */
/* CALLS        : inf_get_hdw(...)                                          */
/*              : inf_get_ipc(...)                                          */
/*              : inf_mnu_rtn(...)                                          */
/*              : trl_alg_sym(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

inf_get_alg( ac(_, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, OldInp, Inp):-
    write( "scheduling algorithm    : "),
    makewindow( c_win_two, c_win_frg, c_win_bkg, " Scheduling Strategies ",
        c_two_org, c_win_idt, 5, c_win_wid),
    write( "    1. CPM    \n"),
    write( "    2. DYN (d)\n"),
    write( "    3. AST"),
    gotowindow( c_win_one),
    inf_mnu_rtn( s_mnu_alg, Alg), !,
    trl_alg_sym( Alg, AlgStr),
    write( AlgStr), nl,
    inf_get_ipc( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, OldInp, Inp).

inf_get_alg( _, Ctl, Dir, Ext, ai(Tkf, Hdw), Inp):-
    removewindow( c_win_two, c_win_ref),
    clearwindow(),
    write( "console taskforce       : ", Tkf), nl,
    inf_get_hdw( Dir, Ext, ai(Tkf, Hdw), Inp, Ctl).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : inf_get_ipc(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Get IPC scheme and then satisfy inf_get_lbl(...)          */
/*                                                                          */
/* PARAMETERS   : OldCtl                                                    */
/*              : Ctl                                                       */
/*              : Dir                                                       */
/*              : Ext                                                       */
/*              : OldInp                                                    */
/*              : Inp                                                       */
/*                                                                          */
/* CALLS        : inf_mnu_rtn(...)                                          */
/*              : inf_get_lbl(...)                                          */
/*              : inf_get_alg(...)                                          */
/*              : trl_ipc_sym(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

inf_get_ipc( ac(Alg, _, Lbl, Dyn, Hrt), Ctl, Dir, Ext, OldInp, Inp):-
    write( "IPC scheme              : "),
    makewindow( c_win_two, c_win_frg, c_win_bkg, " IPC Schemes ",
        c_two_org, c_win_idt, 5, c_win_wid),
    write( "    1. FWE (d)\n"),
    write( "    2. NSL    \n"),
    write( "    3. NML"),
    gotowindow( c_win_one),
    inf_mnu_rtn( s_mnu_ipc, Ipc), !,
    trl_ipc_sym( Ipc, IpcStr),
    write( IpcStr), nl,
    inf_get_lbl( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, OldInp, Inp).

inf_get_ipc( OldCtl, Ctl, Dir, Ext, ai(Tkf, Hdw), Inp):-
    removewindow( c_win_two, c_win_ref),
    clearwindow(),
    write( "console taskforce       : ", Tkf), nl,
    write( "M-P architecture        : ", Hdw), nl, nl,
    inf_get_alg( OldCtl, Ctl, Dir, Ext, ai(Tkf, Hdw), Inp).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : inf_get_lbl(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Get static labelling and then satisfy inf_get_dyn(...)    */
/*                                                                          */
/* PARAMETERS   : OldCtl                                                    */
/*              : Ctl                                                       */
/*              : Dir                                                       */
/*              : Ext                                                       */
/*              : OldInp                                                    */
/*              : Inp                                                       */
```

```
/*                                                                        */
/* CALLS        : inf_get_dyn(...)                                        */
/*              : inf_get_hrt(...)                                        */
/*              : inf_get_ipc(...)                                        */
/*              : inf_mnu_rtn(...)                                        */
/*              : trl_alg_sym(...)                                        */
/*              : trl_lbl_sym(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

inf_get_lbl( OldCtl, Ctl, Dir, Ext, OldInp, Inp):-
    OldCtl = ac( s_alg_ast, _, _, _, _), !,
    inf_get_hrt( OldCtl, Ctl, Dir, Ext, OldInp, Inp).

inf_get_lbl( ac(Alg, Ipc, _, Dyn, Hrt), Ctl, Dir, Ext, OldInp, Inp):-
    write( "static labelling scheme : "),
    makewindow( c_win_two, c_win_frg, c_win_bkg, " Static Labelling Schemes ",
        c_two_org, c_win_idt, 7, c_win_wid),
    write( "    1. HLN    \n"),
    write( "    2. HLE (d)\n"),
    write( "    3. LCN    \n"),
    write( "    4. LCE    \n"),
    write( "    5. RND"),
    gotowindow( c_win_one),
    inf_mnu_rtn( s_mnu_lbl, Lbl), !,
    trl_lbl_sym( Lbl, LblStr),
    write( LblStr), nl,
    inf_get_dyn( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, OldInp, Inp).

inf_get_lbl( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, ai(Tkf, Hdw), Inp):-
    removewindow( c_win_two, c_win_ref),
    clearwindow(),
    trl_alg_sym( Alg, AlgStr),
    write( "console taskforce       : ", Tkf), nl,
    write( "M-P architecture        : ", Hdw), nl, nl,
    write( "scheduling strategy     : ", AlgStr), nl,
    inf_get_ipc( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, ai(Tkf, Hdw), Inp).

/* ---------------------------------------------------------------------- */
/* PREDICATE    : inf_get_dyn(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Get dynamic labelling and then satisfy inf_get_hrt(...) */
/*                                                                        */
/* PARAMETERS   : OldCtl                                                  */
/*              : Ctl                                                     */
/*              : Dir                                                     */
/*              : Ext                                                     */
/*              : OldInp                                                  */
/*              : Inp                                                     */
/*                                                                        */
/* CALLS        : inf_get_hrt(...)                                        */
/*              : inf_get_lbl(...)                                        */
/*              : inf_mnu_rtn(...)                                        */
/*              : trl_alg_sym(...)                                        */
/*              : trl_dyn_sym(...)                                        */
/*              : trl_ipc_sym(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

inf_get_dyn( Ctl, Ctl, _, _, Inp, Inp):-
    Ctl = ac( s_alg_cpm, _, _, _, _), !.

inf_get_dyn( ac(Alg, Ipc, Lbl, _, Hrt), Ctl, Dir, Ext, OldInp, Inp):-
    write( "dynamic labelling scheme: "),
    makewindow( c_win_two, c_win_frg, c_win_bkg, " Dynamic Labelling Schemes ",
        c_two_org, c_win_idt, 6, c_win_wid),
    write( "    1. NON  \n"),
    write( "    2. PRO  \n"),
    write( "    3. TSK  \n"),
    write( "    4. ALL (d)"),
    gotowindow( c_win_one),
    inf_mnu_rtn( s_mnu_dyn, Dyn), !,
    trl_dyn_sym( Dyn, DynStr),
    write( DynStr), nl,
    inf_get_hrt( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, OldInp, Inp).

inf_get_dyn( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, ai(Tkf, Hdw), Inp):-
    removewindow( c_win_two, c_win_ref),
    clearwindow(),
    trl_alg_sym( Alg, AlgStr),
    trl_ipc_sym( Ipc, IpcStr),
    write( "console taskforce       : ", Tkf), nl,
    write( "M-P architecture        : ", Hdw), nl, nl,
    write( "scheduling strategy     : ", AlgStr), nl,
    write( "IPC scheme              : ", IpcStr), nl,
    inf_get_lbl( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, ai(Tkf, Hdw), Inp).

/* ---------------------------------------------------------------------- */
/* PREDICATE    : inf_get_hrt(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Get heuristic underestimate and then bind <Inp> and <Ctl> */
/*                                                                        */
/* PARAMETERS   : OldCtl                                                  */
/*              : Ctl                                                     */
/*              : Dir                                                     */
/*              : Ext                                                     */
/*              : OldInp                                                  */
/*              : Inp                                                     */
/*                                                                        */
/* CALLS        : inf_get_ipc(...)                                        */
/*              : inf_mnu_rtn(...)                                        */
/*              : trl_alg_sym(...)                                        */
/*              : trl_hrt_sym(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
```

```
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

inf_get_hrt( Ctl, Ctl, _, _, Inp, Inp):-
    Ctl = ac(s_alg_dyn, _, _, _, _), !.

inf_get_hrt( OldCtl, ac(Alg, Ipc, Lbl, Dyn, Hrt), _, _, Inp, Inp):-
    OldCtl = ac(Alg, Ipc, Lbl, Dyn, _),
    write( "heuristic underestimate : "),
    makewindow( c_win_two, c_win_frg, c_win_bkg, " Heuristic Underestimates ",
        c_two_org, c_win_idt, 7, c_win_wid),
    write( "    1. NON  \n"),
    write( "    2. MPC  \n"),
    write( "    3. MIA  \n"),
    write( "    4. MRC  \n"),
    write( "    5. ALL (d)"),
    gotowindow( c_win_one),
    inf_mnu_rtn( s_mnu_hrt, Hrt), !,
    trl_hrt_sym( Hrt, HrtStr),
    write( HrtStr), nl.

inf_get_hrt( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, ai(Tkf, Hdw), Inp):-
    removewindow( c_win_two, c_win_ref),
    clearwindow(),
    trl_alg_sym( Alg, AlgStr),
    write( "console taskforce       : ", Tkf), nl,
    write( "M-P architecture        : ", Hdw), nl, nl,
    write( "scheduling strategy     : ", AlgStr), nl,
    inf_get_ipc( ac(Alg, Ipc, Lbl, Dyn, Hrt), Ctl, Dir, Ext, ai(Tkf, Hdw), Inp).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : inf_vld_alg(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Bind <AlgCod> to scheduling algorithm given by <MnuChr>   */
/*                                                                          */
/* PARAMETERS   : MnuChr                                                    */
/*              : AlgCod                                                    */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

inf_vld_alg( '1', s_alg_cpm):- !.

inf_vld_alg( '2', s_alg_dyn):- !.

inf_vld_alg( 'd', s_alg_dyn):- !.

inf_vld_alg( '3', s_alg_ast):- !.

inf_vld_alg(   _, s_alg_err):- beep, fail.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : inf_vld_ipc(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Bind <IpcCod> to IPC scheme given by <MnuChr>             */
/*                                                                          */
/* PARAMETERS   : MnuChr                                                    */
/*              : IpcCod                                                    */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

inf_vld_ipc( '1', s_ipc_fwe):- !.

inf_vld_ipc( 'd', s_ipc_fwe):- !.

inf_vld_ipc( '2', s_ipc_nsl):- !.

inf_vld_ipc( '3', s_ipc_nml):- !.

inf_vld_ipc(   _, s_ipc_err):- beep, fail.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : inf_vld_lbl(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Bind <LblCod> to static labelling scheme given by <MnuChr> */
/*                                                                          */
/* PARAMETERS   : MnuChr                                                    */
/*              : LblCod                                                    */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

inf_vld_lbl( '1', s_lbl_hln):- !.

inf_vld_lbl( '2', s_lbl_hle):- !.

inf_vld_lbl( 'd', s_lbl_hle):- !.

inf_vld_lbl( '3', s_lbl_lcn):- !.

inf_vld_lbl( '4', s_lbl_lce):- !.

inf_vld_lbl( '5', s_lbl_rnd):- !.

inf_vld_lbl(   _, s_lbl_err):- beep, fail.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : inf_vld_dyn(...)                                          */
/*                                                                          */
```

```
/* DESCRIPTION  : Bind <DynCod> to dynamic labelling scheme given by <MnuChr> */
/*                                                                            */
/* PARAMETERS   : MnuChr                                                      */
/*              : DynCod                                                      */
/*                                                                            */
/* CALLS        : --                                                          */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_vld_dyn( '1', s_dyn_non):- !.

inf_vld_dyn( '2', s_dyn_pro):- !.

inf_vld_dyn( '3', s_dyn_tsk):- !.

inf_vld_dyn( '4', s_dyn_all):- !.

inf_vld_dyn( 'd', s_dyn_all):- !.

inf_vld_dyn(   _, s_dyn_err):- beep, fail.

/* -------------------------------------------------------------------------- */
/* PREDICATE    : inf_vld_hrt(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Bind <HrtCod> to heuristic underestimate given by <MnuChr>  */
/*                                                                            */
/* PARAMETERS   : MnuChr                                                      */
/*              : HrtCod                                                      */
/*                                                                            */
/* CALLS        : --                                                          */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_vld_hrt( '1', s_hrt_non):- !.

inf_vld_hrt( '2', s_hrt_mpc):- !.

inf_vld_hrt( '3', s_hrt_mia):- !.

inf_vld_hrt( '4', s_hrt_mrc):- !.

inf_vld_hrt( '5', s_hrt_all):- !.

inf_vld_hrt( 'd', s_hrt_all):- !.

inf_vld_hrt(   _, s_hrt_err):- beep, fail.

/* -------------------------------------------------------------------------- */
/* PREDICATE    : inf_ini_wat(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Initialise search-tree watch-dogs in internal DB            */
/*                                                                            */
/* PARAMETERS   : AlgCtl                                                      */
/*                                                                            */
/* CALLS        : --                                                          */
/*                                                                            */
/* DATABASE     : assert( num_sea_nod(...))                                   */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_ini_wat( ac(s_alg_ast, _, _, _, _)):- !,
    assert( num_sea_nod( 1)).

inf_ini_wat( _).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : inf_tst_tst(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Fail if any element of <TstLst> corresponds to s_tst_fai    */
/*                                                                            */
/* PARAMETERS   : TstLst                                                      */
/*                                                                            */
/* CALLS        : SELF                                                        */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_tst_tst( []):- !.

inf_tst_tst( [s_tst_fai|_]):- !, fail.

inf_tst_tst( [_|RestTests]):-
    inf_tst_tst( RestTests).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : inf_ctp_tst(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Bind <PassFail> to s_tst_fai if critical path will not fit  */
/*              : on any processor, or s_tst_pas otherwise                    */
/*                                                                            */
/* PARAMETERS   : PrecList                                                    */
/*              : ProcList                                                    */
/*              : PassFail                                                    */
/*                                                                            */
/* CALLS        : clc_crt_pth(...)                                            */
/*              : inf_dct_aux(...)                                            */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

inf_ctp_tst( _, [], s_tst_fai):- !.
```

```
inf_ctp_tst( PrecList, ProcList, s_tst_pas):-
    clc_crt_pth( PrecList, CritPath),
    inf_dct_aux( ProcList, CritPath), !.

inf_ctp_tst( _, _, s_tst_fai).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_dct_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for inf_ctp_tst(...)                 */
/*                                                                          */
/* PARAMETERS    : ProcList                                                 */
/*               : CritPath                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : idb_pro_exe(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

inf_dct_aux( [], _):- !.

inf_dct_aux( [pr( ProcName, _)|RestProcs], CritPath):-
    idb_pro_exe( ProcName, ProcExec),
    ProcExec > CritPath,
    inf_dct_aux( RestProcs, CritPath).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_fit_tst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <PassFail> to s_tst_fai if taskforce <TaskList> will */
/*               : not fit on processing resource <ProcList>, ignoring      */
/*               : precedence and IPC, or s_tst_pas otherwise               */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : ProcList                                                 */
/*               : PassFail                                                 */
/*                                                                          */
/* CALLS         : clc_seq_sch(...)                                         */
/*               : clc_hdw_rsc(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

inf_fit_tst( TaskList, ProcList, s_tst_pas):-
    clc_seq_sch( TaskList, SeqlSchd),
    clc_hdw_rsc( ProcList, HdwrRsrc),
    HdwrRsrc > SeqlSchd, !.

inf_fit_tst( _, _, s_tst_fai).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_grn_tst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <PassFail> to s_tst_fai if any task from <TaskList> */
/*               : will not fit on any processor from <ProcList>            */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : ProcList                                                 */
/*               : PassFail                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : inf_dgt_aux(...)                                         */
/*               : inf_grn_tst(...)                                         */
/*               : idb_tsk_exe(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

inf_grn_tst( [], _, s_tst_pas):-        !.

inf_grn_tst( [tk( TaskName, _)|RestTasks], ProcList, PassFail):-
    idb_tsk_exe( TaskName, TaskTime),
    inf_dgt_aux( TaskTime,  ProcList),  !,
    inf_grn_tst( RestTasks, ProcList, PassFail).

inf_grn_tst( _, _, s_tst_fai).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_dgt_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for inf_grn_tst(...)                 */
/*                                                                          */
/* PARAMETERS    : TaskTime                                                 */
/*               : ProcList                                                 */
/*                                                                          */
/* CALLS         : idb_pro_exe(...)                                         */
/*               : lst_mem_ndt(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

inf_dgt_aux( TaskTime, ProcList):-
    lst_mem_ndt( Processor, ProcList),
    Processor = pr( ProcName, _),
    idb_pro_exe( ProcName, ProcTime),
    TaskTime > ProcTime, !,
    fail.

inf_dgt_aux( _, _).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_sho_sch(...)                                         */
```

```
/*                                                                        */
/* DESCRIPTION  : Display DCS output on screen and bind <RtrnStat> to status */
/*              : returned by edit(...)                                   */
/*                                                                        */
/* PARAMETERS   : RtrnStat                                                */
/*                                                                        */
/* CALLS        : --                                                      */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

inf_sho_sch( RtrnStat):- !,
    textmode( Rows, Cols),
    HghtWin1 = 4,
    HghtWin3 = 3,
    OrigWin3 = Rows - HghtWin3,
    HghtWin2 = Rows - HghtWin1 - HghtWin3,
    makewindow( c_win_one, c_win_frg, c_win_bkg,
        " DCS - Digital Console Scheduler ", 0, 0, HghtWin1, Cols),
    write( " 1995\t: School of Engineering, University of Durham\n"),
    write( "     \t: Solid State Logic, Begbroke, Oxford, England"),
    makewindow( c_win_two, c_win_frg, c_win_bkg, c_nul_str,
        HghtWin1, 0, HghtWin2, Cols),
    makewindow( c_win_thr, c_win_frg, c_win_bkg, c_nul_str,
        OrigWin3, 0, HghtWin3, Cols),
    writedevice( screen),
    write( "         After viewing schedule, press F10 to SAVE or Esc to EXIT"),
    gotowindow( c_win_two),
    concat( c_sch_dir, c_pth_dlm, SchPth),
    concat( SchPth, c_sch_nam, SchFil),
    file_str( SchFil, InpStr),
    edit( InpStr, _, c_nul_str, c_nul_str, c_nul_str, 1, c_nul_str,
        0, 0, 0, 1, _, RtrnStat).

/* ---------------------------------------------------------------------- */
/* PREDICATE    : inf_sav_sch(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Prompt user to save schedule and remove all windows     */
/*                                                                        */
/* PARAMETERS   : RtrnStat                                                */
/*                                                                        */
/* CALLS        : inf_get_fil(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

inf_sav_sch( 0):-
    textmode( Rows, _),
    HghtWin4 = 5,
    OrigWin4 = ((Rows - HghtWin4) / 2) - 1,
    makewindow( c_win_for, c_win_frg, c_win_bkg, " Save DCS Output ",
        OrigWin4, 12, HghtWin4, 57),
    clearwindow(),
    write( "save DCS output as (?.sch): "),
    inf_get_fil(),
    fail.

inf_sav_sch( _):-
    removewindow( c_win_one, c_win_ref),
    removewindow( c_win_two, c_win_ref),
    removewindow( c_win_thr, c_win_ref).

/* ---------------------------------------------------------------------- */
/* PREDICATE    : inf_get_fil(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Get filename in which to store current DCS output       */
/*                                                                        */
/* PARAMETERS   : --                                                      */
/*                                                                        */
/* CALLS        : inf_vld_fil(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

inf_get_fil():-
    readln( UsrStr),
    isname( UsrStr),
    concat( UsrStr, ".sch", SchFil),
    inf_vld_fil( SchFil), !.

inf_get_fil():-
    removewindow( c_win_for, c_win_ref).

/* ---------------------------------------------------------------------- */
/* PREDICATE    : inf_vld_fil(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Ensure output filename <SchFil> does not aleady exist   */
/*                                                                        */
/* PARAMETERS   : SchFil                                                  */
/*                                                                        */
/* CALLS        : inf_ren_fil(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

inf_vld_fil( SchFil):-
    concat( c_sch_dir, c_pth_dlm, SchPth),
    concat( SchPth, c_sch_nam, SchFil), !,
    removewindow( c_win_for, c_win_ref).

inf_vld_fil( SchNam):-
    concat( c_sch_dir, c_pth_dlm, SchPth),
    concat( SchPth, SchNam, NewSchFil),
    not( existfile( NewSchFil)), !,
```

```
    concat( SchPth, c_sch_nam, TmpSchFil),
    renamefile( TmpSchFil, NewSchFil),
    removewindow( c_win_for, c_win_ref).

inf_vld_fil( SchFil):-
    clearwindow,
    write( "Schedule to be saved as (?.sch): "),
    cursor( Row, Col),
    write( SchFil),
    write( "\n\n          ! ", SchFil, " already exists"),
    cursor( Row, Col),
    readchar( UsrChr),
    inf_ren_fil( UsrChr, SchFil).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : inf_ren_fil(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Rename temporary DCS output 'temp.sch' to <SchFil>       */
/*                                                                          */
/* PARAMETERS    : UsrChr                                                   */
/*               : SchFil                                                   */
/*                                                                          */
/* CALLS         : inf_get_fil(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

inf_ren_fil( UsrChr, SchNam):-
    char_int( UsrChr, c_key_rtn), !,
    concat( c_sch_dir, c_pth_dlm, SchPth),
    concat( SchPth, c_sch_nam, TmpSchFil),
    concat( SchPth, SchNam, NewSchFil),
    renamefile( TmpSchFil, NewSchFil),
    removewindow( c_win_for, c_win_ref).

inf_ren_fil( UsrChr, _):-
    unreadchar( UsrChr),
    clearwindow(),
    write( "save DCS output as (?.sch): "),
    inf_get_fil().

/* ************************************************************************ */
```

D.4.8 dcs_lst.pro

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_LST.PRO                                                  */
/*                                                                          */
/* NOTES     : General-purpose list-handling predicates                     */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 10th January, 1995                                           */
/*                                                                          */
/* VERSION   : 3.8                                                          */
/* ************************************************************************ */

/* ======================================================================== */
/* Project definition                                                       */
/* ======================================================================== */

project "dcs.prj"

/* ======================================================================== */
/* Public predicate definitions                                             */
/* ======================================================================== */

include "dcs_glb.pub"

include "dcs_dyn.pub"
include "dcs_lst.pub"

/* ======================================================================== */
/* Private predicate definitions                                            */
/* ======================================================================== */

include "dcs_lst.prv"

/* ======================================================================== */
   clauses
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_fst_elm(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Extract first <Elem> from <List>                         */
/*                                                                          */
/* PARAMETERS    : Elem                                                     */
/*               : List                                                     */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
```

```
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

lst_fst_elm( [Elem|_], Elem).

/* -------------------------------------------------------------------- */
/* PREDICATE    : lst_lst_elm(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Extract last <Elem> from <List>                       */
/*                                                                      */
/* PARAMETERS   : Elem                                                  */
/*              : List                                                  */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

lst_lst_elm( [Elem], Elem):- !.

lst_lst_elm( [_|Tail], Elem):-
    lst_lst_elm( Tail, Elem).

/* -------------------------------------------------------------------- */
/* PREDICATE    : lst_nxt_elm(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Extract next <Elem1> from <List> as <Elem2>           */
/*                                                                      */
/* PARAMETERS   : Elem1                                                 */
/*              : List                                                  */
/*              : Elem2                                                 */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

lst_nxt_elm( Elem1, [Elem1, Elem2|_], Elem2):- !.

lst_nxt_elm( Elem1, [_|Tail], Elem2):-
    lst_nxt_elm( Elem1, Tail, Elem2).

/* -------------------------------------------------------------------- */
/* PREDICATE    : lst_prv_elm(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Extract previous <Elem1> from <List> as <Elem2>       */
/*                                                                      */
/* PARAMETERS   : Elem1                                                 */
/*              : List                                                  */
/*              : Elem2                                                 */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

lst_prv_elm( Elem1, [Elem2, Elem1|_], Elem2):- !.

lst_prv_elm( Elem1, [_|Tail], Elem2):-
    lst_prv_elm( Elem1, Tail, Elem2).

/* -------------------------------------------------------------------- */
/* PREDICATE    : lst_num_elm(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Count number of elements in <List>                    */
/*                                                                      */
/* PARAMETERS   : List                                                  */
/*              : NumElms                                               */
/*                                                                      */
/* CALLS        : lst_num_aux(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

lst_num_elm( List, NumElms):-
    lst_num_aux( List, 0, NumElms).

/* -------------------------------------------------------------------- */
/* PREDICATE    : lst_num_aux(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for lst_num_elm(...)              */
/*                                                                      */
/* PARAMETERS   : List                                                  */
/*              : NumElms1                                              */
/*              : NumElms2                                              */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

lst_num_aux( [], NumElms, NumElms):- !.

lst_num_aux( [_|Tail], OldNumElms, NumElms):-
    NewNumElms = OldNumElms + 1,
    lst_num_aux( Tail, NewNumElms, NumElms).

/* -------------------------------------------------------------------- */
/* PREDICATE    : lst_ers_one(...)                                      */
/*                                                                      */
```

```
/* DESCRIPTION  : Erase one occurance of <Elem> from <List1> to give <List2>  */
/*                                                                            */
/* PARAMETERS   : Elem                                                        */
/*              : List1                                                       */
/*              : List2                                                       */
/*                                                                            */
/* CALLS        : SELF                                                        */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

lst_ers_one( _, [], []):- !.

lst_ers_one( Elem, [Elem|Tail], Tail):- !.

lst_ers_one( Elem1, [Elem2|Tail1], [Elem2|Tail2]):-
    lst_ers_one( Elem1, Tail1, Tail2).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : lst_ers_all(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Erase all occurances of <Elem> from <List1> to give <List2> */
/*                                                                            */
/* PARAMETERS   : Elem                                                        */
/*              : List1                                                       */
/*              : List2                                                       */
/*                                                                            */
/* CALLS        : SELF                                                        */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

lst_ers_all( _, [], []):- !.

lst_ers_all( Elem, [Elem|Tail], List):- !,
    lst_ers_all( Elem, Tail, List).

lst_ers_all( Elem1, [Elem2|Tail1], [Elem2|Tail2]):-
    lst_ers_all( Elem1, Tail1, Tail2).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : lst_mem_elm(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Check <Elem> is a member of <List>                          */
/*                                                                            */
/* PARAMETERS   : Elem                                                        */
/*              : List                                                        */
/*                                                                            */
/* CALLS        : SELF                                                        */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

lst_mem_elm( Head, [Head|_]):- !.

lst_mem_elm( Elem, [_|Tail]):-
    lst_mem_elm( Elem, Tail).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : lst_mem_ndt(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Generate each <Elem> which is a member of <List>            */
/*                                                                            */
/* PARAMETERS   : Elem                                                        */
/*              : List                                                        */
/*                                                                            */
/* CALLS        : SELF                                                        */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : non-determ                                                  */
/* -------------------------------------------------------------------------- */

lst_mem_ndt( Head, [Head|_]).

lst_mem_ndt( Elem, [_|Tail]):-
    lst_mem_ndt( Elem, Tail).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : lst_del_elm(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Delete <Elem> from <List1> and bind <List2> to result       */
/*                                                                            */
/* PARAMETERS   : Elem                                                        */
/*              : List1                                                       */
/*              : List2                                                       */
/*                                                                            */
/* CALLS        : SELF                                                        */
/*                                                                            */
/* DATABASE     : --                                                          */
/*                                                                            */
/* BACKTRACKING : determ                                                      */
/* -------------------------------------------------------------------------- */

lst_del_elm( Elem, [Elem|List], List):- !.

lst_del_elm( Elem1, [Elem2|List1], [Elem2|List2]):-
    lst_del_elm( Elem1, List1, List2).

/* -------------------------------------------------------------------------- */
/* PREDICATE    : lst_del_ndt(...)                                            */
/*                                                                            */
/* DESCRIPTION  : Generate all <Elem> from <List1>, binding <List2> to rest   */
/*                                                                            */
/* PARAMETERS   : Elem                                                        */
/*              : List1                                                       */
```

```
/*                : List2                                                    */
/*                                                                           */
/* CALLS          : SELF                                                     */
/*                                                                           */
/* DATABASE       : --                                                       */
/*                                                                           */
/* BACKTRACKING   : non-determ                                               */
/* ------------------------------------------------------------------------- */

lst_del_ndt( Elem, [Elem|List], List).

lst_del_ndt( Elem1, [Elem2|List1], [Elem2|List2]):-
    lst_del_ndt( Elem1, List1, List2).

/* ------------------------------------------------------------------------- */
/* PREDICATE      : lst_apd_lst(...)                                         */
/*                                                                           */
/* DESCRIPTION    : Bind <List3> as <List2> appended to <List1>              */
/*                                                                           */
/* PARAMETERS     : List1                                                    */
/*                : List2                                                    */
/*                : List3                                                    */
/*                                                                           */
/* CALLS          : SELF                                                     */
/*                                                                           */
/* DATABASE       : --                                                       */
/*                                                                           */
/* BACKTRACKING   : determ                                                   */
/* ------------------------------------------------------------------------- */

lst_apd_lst( [], List, List):- !.

lst_apd_lst( [Elem|Tail1], List, [Elem|Tail2]):-
    lst_apd_lst( Tail1, List, Tail2).

/* ------------------------------------------------------------------------- */
/* PREDICATE      : lst_apd_ndt(...)                                         */
/*                                                                           */
/* DESCRIPTION    : Generate all <List3> as <List2> appended to <List1>      */
/*                                                                           */
/* PARAMETERS     : List1                                                    */
/*                : List2                                                    */
/*                : List3                                                    */
/*                                                                           */
/* CALLS          : SELF                                                     */
/*                                                                           */
/* DATABASE       : --                                                       */
/*                                                                           */
/* BACKTRACKING   : non-determ                                               */
/* ------------------------------------------------------------------------- */

lst_apd_ndt( [], List, List).

lst_apd_ndt( [Elem|Tail1], List, [Elem|Tail2]):-
    lst_apd_ndt( Tail1, List, Tail2).

/* ------------------------------------------------------------------------- */
/* PREDICATE      : lst_rev_lst(...)                                         */
/*                                                                           */
/* DESCRIPTION    : Reverse <List1> to bind <List2>                          */
/*                                                                           */
/* PARAMETERS     : List1                                                    */
/*                : List2                                                    */
/*                                                                           */
/* CALLS          : lst_rev_apd(...)                                         */
/*                                                                           */
/* DATABASE       : --                                                       */
/*                                                                           */
/* BACKTRACKING   : determ                                                   */
/* ------------------------------------------------------------------------- */

lst_rev_lst( List1, List2):-
    lst_rev_apd( List1, [], List2).

/* ------------------------------------------------------------------------- */
/* PREDICATE      : lst_rev_apd(...)                                         */
/*                                                                           */
/* DESCRIPTION    : Bind <List3> = <ltsiL> + <List2>                         */
/*                                                                           */
/* PARAMETERS     : Elem                                                     */
/*                : List1                                                    */
/*                : List2                                                    */
/*                                                                           */
/* CALLS          : SELF                                                     */
/*                                                                           */
/* DATABASE       : --                                                       */
/*                                                                           */
/* BACKTRACKING   : determ                                                   */
/* ------------------------------------------------------------------------- */

lst_rev_apd( [], List, List):- !.

lst_rev_apd( [Elem|List1], List2, List3):-
    lst_rev_apd( List1, [Elem|List2], List3).

/* ------------------------------------------------------------------------- */
/* PREDICATE      : lst_per_mut(...)                                         */
/*                                                                           */
/* DESCRIPTION    : Generate all permutations of <List1> as <List2>          */
/*                                                                           */
/* PARAMETERS     : List1                                                    */
/*                : List2                                                    */
/*                                                                           */
/* CALLS          : SELF                                                     */
/*                : lst_apd_ndt(...)                                         */
/*                                                                           */
/* DATABASE       : --                                                       */
/*                                                                           */
/* BACKTRACKING   : non-determ                                               */
/* ------------------------------------------------------------------------- */

lst_per_mut( List1, [Head|Tail1]):-
```

```
    lst_apd_ndt( List2, WorkList, List1),
    WorkList = [Head|Tail2],
    lst_apd_ndt( List2, Tail2, List3),
    lst_per_mut( List3, Tail1).

lst_per_mut( [], []).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_equ_lst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Check list equality relation, i.e. <List1> = <List2>     */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

lst_equ_lst( List, List).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_equ_set(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Check set equality relation for two lists                */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_del_ndt(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

lst_equ_set( List, List):- !.

lst_equ_set( [Elem|Tail], List):-
    lst_del_ndt( Elem, List, WorkList),
    lst_equ_set( Tail, WorkList).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_sub_lst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Subtract <List1> from <List2> to give <List3>            */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*               : List3                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_mem_ndt(...)                                         */
/*               : lst_del_ndt(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

lst_sub_lst( [], List, List):- !.

lst_sub_lst( [Elem|Tail], List1, List2):-
    lst_mem_ndt( Elem, List1),
    lst_del_ndt( Elem, List1, List3),
    lst_sub_lst( Tail, List3, List2).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_sub_set(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Check <List1> is a sub-set of <List2>                    */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_mem_elm(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

lst_sub_set( [], []):- !.

lst_sub_set( [Elem|Tail], List):-
    lst_mem_elm( Elem, List),
    lst_sub_set( Tail, List).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_dis_jnt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Check if the set <List1> and <List2> are disjoint        */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*                                                                          */
/* CALLS         : lst_mem_elm(...)                                         */
/*               : lst_mem_ndt(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

lst_dis_jnt( List1, List2):-
    lst_mem_ndt( Elem, List1),
    not( lst_mem_elm( Elem, List2)), !.
```

```
/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_int_sec(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create <List3> as the intersection of <List1> and <List2> */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*               : List3                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_mem_elm(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

lst_int_sec( [], _, []):- !.

lst_int_sec( [Elem|Tail1], List, [Elem|Tail2]):-
    lst_mem_elm( Elem, List), !,
    lst_int_sec( Tail1, List, Tail2).

lst_int_sec( [_|Tail], List1, List2):-
    lst_int_sec( Tail, List1, List2).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : lst_set_uni(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create <List3> as the union of <List1> and <List2>       */
/*                                                                          */
/* PARAMETERS    : List1                                                    */
/*               : List2                                                    */
/*               : List3                                                    */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_mem_elm(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

lst_set_uni( [], List, List).

lst_set_uni( [Elem|Tail], List1, List2):-
    lst_mem_elm( Elem, List1), !,
    lst_set_uni( Tail, List1, List2).

lst_set_uni( [Elem|Tail], List1, [Elem|List2]):-
    lst_set_uni( Tail, List1, List2).

/* ************************************************************************ */
```

D.4.9 dcs_sch.pro

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_SCH.PRO                                                  */
/*                                                                          */
/* NOTES     : Application-specific scheduling predicates for DCS engine    */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 15th June, 1995                                              */
/*                                                                          */
/* VERSION   : 7.5                                                          */
/* ************************************************************************ */

/* ======================================================================== */
/* Project definition                                                       */
/* ======================================================================== */

project "dcs.prj"

/* ======================================================================== */
/* Public predicate definitions                                             */
/* ======================================================================== */

include "dcs_glb.pub"

include "dcs_clc.pub"
include "dcs_idb.pub"
include "dcs_lst.pub"
include "dcs_sch.pub"
include "dcs_utl.pub"

/* ======================================================================== */
/* Private predicate definitions                                            */
/* ======================================================================== */

include "dcs_sch.prv"

/* ======================================================================== */
   clauses
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_pre_rel(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Fail if any predecessors of <Task> are in <TaskList>     */
/*                                                                          */
/* PARAMETERS    : Task                                                     */
/*               : TaskList                                                 */
```

```
/*                                                                      */
/* CALLS        : idb_pre_rel(...)                                      */
/*              : lst_mem_ndt(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

sch_pre_rel( tk( TaskName1, _), TaskList):-
    lst_mem_ndt( Task2, TaskList),
    Task2 = tk( TaskName2, _),
    idb_pre_rel( TaskName2, TaskName1), !,
    fail.

sch_pre_rel( _, _).

/* -------------------------------------------------------------------- */
/* PREDICATE    : sch_act_tsk(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Fail if any predecessors of <Task> in <ProcList> finish */
/*              : execution after <Fin>                                 */
/*                                                                      */
/* PARAMETERS   : Task                                                  */
/*              : Fin                                                   */
/*              : ProcList                                              */
/*                                                                      */
/* CALLS        : idb_pre_rel(...)                                      */
/*              : lst_mem_ndt(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

sch_act_tsk( tk( Task, _), Fin, ProcList):-
    lst_mem_ndt( Proc, ProcList),
    Proc = pr( _, [tk(T1,F1)|_]),
    Fin < F1,
    idb_pre_rel( T1, Task), !,
    fail.

sch_act_tsk( _, _, _).

/* -------------------------------------------------------------------- */
/* PREDICATE    : sch_pro_rsc(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Fail if executing <Task> on <Proc> with current finishing */
/*              : time of <FinTime> exceeds processing resource of <Proc> */
/*                                                                      */
/* PARAMETERS   : Task                                                  */
/*              : ProcName                                              */
/*              : FinTime                                               */
/*                                                                      */
/* CALLS        : idb_pro_exe(...)                                      */
/*              : lst_tsk_exe(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

sch_pro_rsc( tk( TaskName, _), ProcName, FinTime):-
    idb_tsk_exe( TaskName, TaskExec),
    idb_pro_exe( ProcName, ProcExec),
    FinTime + TaskExec <= ProcExec, !.

/* -------------------------------------------------------------------- */
/* PREDICATE    : sch_non_cor(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Bind <NCRList> to those tasks already scheduled in    */
/*              : <ActiveProcs> which are predecessors of <Task> and    */
/*              : therefore NOT co-resident                             */
/*                                                                      */
/* PARAMETERS   : Task                                                  */
/*              : ActiveProcs                                           */
/*              : NCRList                                               */
/*                                                                      */
/* CALLS        : sch_snc_aux(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

sch_non_cor( Task, ActiveProcs, NCRList):-
    sch_snc_aux( Task, ActiveProcs, [], NCRList).

/* -------------------------------------------------------------------- */
/* PREDICATE    : sch_snc_aux(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for sch_non_cor(...)              */
/*                                                                      */
/* PARAMETERS   : Task                                                  */
/*              : ActiveProcs                                           */
/*              : OldNCRList                                            */
/*              : NCRList                                               */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*              : sch_ncr_tsk(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

sch_snc_aux( _, [], NCRList, NCRList):- !.

sch_snc_aux( tk(TN, TF), [pr( PN, TL)|RestProcs], OldNCRList, NCRList):-
    sch_ncr_tsk( TN, TL, [], NCRTasks), !,
    sch_snc_aux( tk(TN,TF), RestProcs, [pr(PN, NCRTasks)|OldNCRList], NCRList).
```

```
sch_snc_aux( Task, [_|RestProcs], OldNCRList, NCRList):-
    sch_snc_aux( Task, RestProcs, OldNCRList, NCRList).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_ncr_tsk(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Bind <NCRTasks> to all tasks in <TaskList> which are     */
/*               : non-coresident predecessors of <Task>                    */
/*                                                                          */
/* PARAMETERS    : Task                                                     */
/*               : TaskList                                                 */
/*               : OldNCRTasks                                              */
/*               : NCRTasks                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : idb_tsk_com(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

sch_ncr_tsk( _, [], [], _):- !, fail.

sch_ncr_tsk( Task, [tk( PrecTask, _)|RestTasks], OldNCRTasks, NCRTasks):-
    idb_tsk_com( PrecTask, Task), !,
    sch_ncr_tsk( Task, RestTasks, [tk( PrecTask, 0)|OldNCRTasks], NCRTasks).

sch_ncr_tsk( Task, [_|RestTasks], OldNCRTasks, NCRTasks):- !,
    sch_ncr_tsk( Task, RestTasks, OldNCRTasks, NCRTasks).

sch_ncr_tsk( _, [], NCRTasks, NCRTasks).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_ins_ipc(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Schedule IPC according to <IpcScheme> for non-coresident */
/*               : tasks in <NCRList> on processors in <ProcList> at finish */
/*               : time <FinTime>. <TaskList> is bound to IPC required on   */
/*               : destination processor                                    */
/*                                                                          */
/* PARAMETERS    : IpcScheme                                                */
/*               : NCRList                                                  */
/*               : OldProcList                                              */
/*               : ProcList                                                 */
/*               : OldFinTime                                               */
/*               : FinTime                                                  */
/*               : OldTaskList                                              */
/*               : TaskList                                                 */
/*                                                                          */
/* CALLS         : clc_max_val(...)                                         */
/*               : lst_del_ndt(...)                                         */
/*               : lst_num_elm(...)                                         */
/*               : sch_ins_ipc(...)                                         */
/*               : sch_ipc_dst(...)                                         */
/*               : sch_ipc_non(...)                                         */
/*               : sch_ipc_src(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

sch_ins_ipc( _, [], ProcList, ProcList, FinTime, FinTime, TL, TL):- !.

sch_ins_ipc( s_ipc_fwe, _, ProcList, ProcList, FinTime, FinTime, TL, TL):- !.

sch_ins_ipc( s_ipc_nsl, [pr(P,NCRs)|Tail], OldPL, PL, OldF, F, OldTL, TL):-
    lst_del_ndt( Proc, OldPL, PL1),
    Proc = pr(P, [tk(_,TF)|_]), !,
    OldTL = [tk(_,PF)|_],
    clc_max_val( PF, TF, OldIpcStt),
    lst_num_elm( NCRs, NumNCRTasks),
    OldIpcEnd = OldIpcStt + (NumNCRTasks * c_ipc_tim),
    sch_ipc_non( OldIpcStt, OldIpcEnd, OldPL, IpcStt, IpcEnd),
    sch_ipc_src( IpcStt, IpcEnd, Proc, PL1, NewPL, OldF, NewF),
    sch_ipc_dst( IpcStt, IpcEnd, OldTL, NewTL),
    sch_ins_ipc( s_ipc_nsl, Tail, NewPL, PL, NewF, F, NewTL, TL).

sch_ins_ipc( s_ipc_nml, [pr(P, _)|Tail], OldPL, PL, OldF, F, OldTL, TL):-
    lst_del_ndt( Proc, OldPL, PL1),
    Proc = pr(P, [tk(_,TF)|_]), !,
    OldTL = [tk(_,PF)|_],
    clc_max_val( PF, TF, IpcStt),
    IpcEnd = IpcStt + c_ipc_tim,
    sch_ipc_src( IpcStt, IpcEnd, Proc, PL1, NewPL, OldF, NewF),
    sch_ipc_dst( IpcStt, IpcEnd, OldTL, NewTL),
    sch_ins_ipc( s_ipc_nml, Tail, NewPL, PL, NewF, F, NewTL, TL).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_ipc_non(...)                                         */
/*                                                                          */
/* DESCRIPTION   : For non-overlapping IPC on a single communication channel, */
/*               : bind <IpcStt> and <IpcEnd> to start and end times of IPC */
/*               : which would have spanned <OldIpcStt> to <OldIpcEnd>      */
/*                                                                          */
/* PARAMETERS    : OldIpcStt                                                */
/*               : OldIpcEnd                                                */
/*               : ProcList                                                 */
/*               : IpcStt                                                   */
/*               : IpcEnd                                                   */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : sch_ipc_chk(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

sch_ipc_non( OldIpcStt, OldIpcEnd, ProcList, IpcStt, IpcEnd):-
    sch_ipc_chk( OldIpcStt, OldIpcEnd, ProcList), !,
```

```
    IpcStt = OldIpcStt,
    IpcEnd = OldIpcEnd.

sch_ipc_non( OldIpcStt, OldIpcEnd, ProcList, IpcStt, IpcEnd):-
    NewIpcStt = OldIpcStt + c_ipc_tim,
    NewIpcEnd = OldIpcEnd + c_ipc_tim,
    sch_ipc_non( NewIpcStt, NewIpcEnd, ProcList, IpcStt, IpcEnd).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_ipc_chk(...)                                         */
/*                                                                          */
/* DESCRIPTION   : With processor state <ProcList>, check non-overlapping IPC */
/*               : from <IpcStt> to <IpcEnd> can be reserved on a single link */
/*                                                                          */
/* PARAMETERS    : IpcStt                                                   */
/*               : IpcEnd                                                   */
/*               : ProcList                                                 */
/*                                                                          */
/* CALLS         : SELF                                                     */
/*               : lst_mem_ndt(...)                                         */
/*               : lst_nxt_elm(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

sch_ipc_chk( IpcStt, IpcEnd, [pr(_,TaskList)|_]):-
    lst_mem_ndt( ThisTask, TaskList),
    ThisTask = tk( ipc, OldIpcEnd),
    lst_nxt_elm( tk(ipc, OldIpcEnd), TaskList, PrevTask),
    PrevTask = tk( _, OldIpcStt),
    IpcStt < OldIpcEnd,
    IpcEnd > OldIpcStt, !,
    fail.

sch_ipc_chk( IpcStt, IpcEnd, [_|RestProcs]):- !,
    sch_ipc_chk( IpcStt, IpcEnd, RestProcs).

sch_ipc_chk( _, _, _).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_ipc_src(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Schedule IPC from <IpcStt> to <IpcEnd> on the source     */
/*               : processor <Processor>, binding <ProcList> and <FinTime>  */
/*               : to the resulting schedule and finishing time             */
/*                                                                          */
/* PARAMETERS    : IpcStt                                                   */
/*               : IpcEnd                                                   */
/*               : Processor                                                */
/*               : OldProcList                                              */
/*               : ProcList                                                 */
/*               : OldFinTime                                               */
/*               : FinTime                                                  */
/*                                                                          */
/* CALLS         : sch_ins_tsk(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

sch_ipc_src( IpcStt, IpcEnd, pr(P,[tk(idle,TF)|Done]), OldPL, PL, OldF, F):-
    IpcStt > TF, !,
    TaskList = [tk(ipc,IpcEnd), tk(idle,IpcStt)|Done],
    sch_ins_tsk( pr(P, TaskList), OldPL, PL, OldF, F).

sch_ipc_src( IpcStt, IpcEnd, pr(P,[tk(T,TF)|Done]), OldPL, PL, OldF, F):-
    IpcStt > TF, !,
    TaskList = [tk(ipc,IpcEnd), tk(idle,IpcStt), tk(T,TF)|Done],
    sch_ins_tsk( pr(P, TaskList), OldPL, PL, OldF, F).

sch_ipc_src( _, IpcEnd, pr(P,Done), OldPL, PL, OldF, F):-
    sch_ins_tsk( pr(P, [tk(ipc,IpcEnd)|Done]), OldPL, PL, OldF, F).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_ipc_dst(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Schedule IPC from <IpcStt> to <IpcEnd> on the destination */
/*               : processor, binding <TaskList> to the result              */
/*                                                                          */
/* PARAMETERS    : IpcStt                                                   */
/*               : IpcEnd                                                   */
/*               : OldTaskList                                              */
/*               : TaskList                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

sch_ipc_dst( IpcStt, IpcEnd, [tk(idle,TF)|Done], TaskList):-
    IpcStt > TF, !,
    TaskList = [tk(ipc,IpcEnd), tk(idle,IpcStt)|Done].

sch_ipc_dst( IpcStt, IpcEnd, [tk(T,TF)|Done], TaskList):-
    IpcStt > TF, !,
    TaskList = [tk(ipc,IpcEnd), tk(idle,IpcStt), tk(T,TF)|Done].

sch_ipc_dst( _, IpcEnd, OldTL, [tk(ipc,IpcEnd)|OldTL]).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : sch_ins_tsk(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Insert a task on processor <Proc> with <ActiveProcs> active */
/*               : at <OldFinTime> to give <ProcList> processors active until */
/*               : the finish time <Fin>                                    */
/*                                                                          */
/* PARAMETERS    : Proc                                                     */
```

```
/*                : ActiveProcs                                          */
/*                : ProcList                                             */
/*                : OldFin                                               */
/*                : Fin                                                  */
/*                                                                       */
/* CALLS          : SELF                                                 */
/*                                                                       */
/* DATABASE       : --                                                   */
/*                                                                       */
/* BACKTRACKING   : determ                                               */
/* --------------------------------------------------------------------- */

sch_ins_tsk( pr(P,[tk(S,A)|D]), OldProcList, ProcList, Fin, Fin):-
    OldProcList = [pr(P1,[tk(T,B)|D1])|L],
    ProcList    = [pr(P,[tk(S,A)|D]),pr(P1,[tk(T,B)|D1])|L],
    A < B, !.

sch_ins_tsk( Proc, OldProcList, [pr(P1,[tk(T,B)|D1])|L1], OldFin, Fin):-
    Proc        = pr(P,[tk(S,A)|D]),
    OldProcList = [pr(P1,[tk(T,B)|D1])|L], !,
    sch_ins_tsk( pr(P,[tk(S,A)|D]), L, L1, OldFin, Fin).

sch_ins_tsk( pr(P,[tk(S,F)|D]), [], [pr(P,[tk(S,F)|D])], _, Fin):-
    Fin = F.

/* --------------------------------------------------------------------- */
/* PREDICATE      : sch_ins_idl(...)                                     */
/*                                                                       */
/* DESCRIPTION    : Insert an idle period on processor <Proc> until the next */
/*                : processor in <ActiveProcs> completes to give <ProcList> */
/*                                                                       */
/* PARAMETERS     : Proc                                                 */
/*                : ActiveProcs                                          */
/*                : ProcList                                             */
/*                                                                       */
/* CALLS          : SELF                                                 */
/*                                                                       */
/* DATABASE       : --                                                   */
/*                                                                       */
/* BACKTRACKING   : determ                                               */
/* --------------------------------------------------------------------- */

sch_ins_idl( pr(P,[tk(idle,A)|D]), [pr(P1,[tk(T1,B)|D1])|L], ProcList):-
    A < B, !,
    ProcList = [pr(P,[tk(idle,B)|D]),pr(P1,[tk(T1,B)|D1])|L].

sch_ins_idl( pr(P,[tk(T,A)|D]), [pr(P1,[tk(T1,B)|D1])|L], ProcList):-
    A < B, !,
    ProcList = [pr(P,[tk(idle,B),tk(T,A)|D]),pr(P1,[tk(T1,B)|D1])|L].

sch_ins_idl( pr(P,[tk(T,A)|D]), OldProcList, [pr(P1,[tk(T1,B)|D1])|L1]):-
    OldProcList = [pr(P1,[tk(T1,B)|D1])|L],
    sch_ins_idl( pr(P,[tk(T,A)|D]), L, L1).

/* --------------------------------------------------------------------- */
/* PREDICATE      : sch_cmp_idl(...)                                     */
/*                                                                       */
/* DESCRIPTION    : Complete idle periods on processors in <ProcList> which */
/*                : finish executing tasks before the finish time of <Schedule> */
/*                                                                       */
/* PARAMETERS     : ProcList                                             */
/*                : FinTime                                              */
/*                : Schedule                                             */
/*                                                                       */
/* CALLS          : SELF                                                 */
/*                : sch_ins_idl(...)                                     */
/*                                                                       */
/* DATABASE       : --                                                   */
/*                                                                       */
/* BACKTRACKING   : determ                                               */
/* --------------------------------------------------------------------- */

sch_cmp_idl( [pr(P,[tk(T,F)|D])|RestProcs], FinTime, Schedule):-
    F < FinTime, !,
    sch_ins_idl( pr(P,[tk(T,F)|D]), RestProcs, NewProcList),
    sch_cmp_idl( NewProcList, FinTime, Schedule).

sch_cmp_idl( Schedule, _, Schedule).

/* ********************************************************************* */
```

D.4.10 dcs_trl.pro

```
/* ************************************************************************ */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England             */
/* ************************************************************************ */

/* ************************************************************************ */
/* MODULE    : DCS_TRL.PRO                                                  */
/*                                                                          */
/* NOTES     : Translation predicates for algorithm control symbols         */
/*                                                                          */
/* AUTHOR    : Ken N LINTON                                                 */
/*                                                                          */
/* DATE      : 1st March, 1995                                              */
/*                                                                          */
/* VERSION   : 1.1                                                          */
/* ************************************************************************ */

/* ======================================================================== */
/* Project definition                                                       */
/* ======================================================================== */

project "dcs.prj"

/* ======================================================================== */
/* Public predicate definitions                                             */
/* ======================================================================== */

include "dcs_glb.pub"

include "dcs_trl.pub"

/* ======================================================================== */
   clauses
/* ======================================================================== */

/* ------------------------------------------------------------------------ */
/* PREDICATE     : trl_alg_sym(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Translate <AlgCod> to scheduling algorithm string <AlgStr> */
/*                                                                          */
/* PARAMETERS    : AlgCod                                                   */
/*               : AlgStr                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

trl_alg_sym( s_alg_cpm, "CPM: static priority list"):- !.

trl_alg_sym( s_alg_dyn, "DYN: dynamic priority list"):- !.

trl_alg_sym( s_alg_ast, "AST: A* dynamic programming"):- !.

trl_alg_sym( s_alg_nul, "--"):- !.

trl_alg_sym(          _, "! unknown algorithm"):- beep.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : trl_ipc_sym(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Translate <IpcCod> to IPC scheme string <AlgStr>         */
/*                                                                          */
/* PARAMETERS    : IpcCod                                                   */
/*               : IpcStr                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

trl_ipc_sym( s_ipc_fwe, "FWE: fully-overlapped with execution"):- !.

trl_ipc_sym( s_ipc_nsl, "NSL: non-overlapped, single link"):- !.

trl_ipc_sym( s_ipc_nml, "NML: non-overlapped, multiple links"):- !.

trl_ipc_sym( s_ipc_nul, "--"):- !.

trl_ipc_sym(          _, "! unknown IPC scheme"):- beep.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : trl_lbl_sym(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Translate <LblCod> to static labelling string <LblStr>   */
/*                                                                          */
/* PARAMETERS    : LblCod                                                   */
/*               : LblStr                                                   */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

trl_lbl_sym( s_lbl_hln, "HLN: highest level, no estimated times"):- !.

trl_lbl_sym( s_lbl_hle, "HLE: highest level, estimated times"):- !.

trl_lbl_sym( s_lbl_lcn, "LCN: lowest co-level, no estimated times"):- !.

trl_lbl_sym( s_lbl_lce, "LCE: lowest co-level, estimated times"):- !.
```

```
trl_lbl_sym( s_lbl_rnd, "RND: random labelling"):- !.

trl_lbl_sym( s_lbl_nul, "--"):- !.

trl_lbl_sym(          _, "! unknown labelling scheme"):- beep.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : trl_dyn_sym(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Translate <DynCod> to dynamic labelling string <DynStr>   */
/*                                                                          */
/* PARAMETERS   : DynCod                                                    */
/*              : DynStr                                                    */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

trl_dyn_sym( s_dyn_non, "NON: no dynamic labelling scheme"):- !.

trl_dyn_sym( s_dyn_pro, "PRO: processor with smallest index"):- !.

trl_dyn_sym( s_dyn_tsk, "TSK: task with highest index"):- !.

trl_dyn_sym( s_dyn_all, "ALL: all task & processor combinations"):- !.

trl_dyn_sym( s_dyn_nul, "--"):- !.

trl_dyn_sym(          _, "! unknown labelling scheme"):- beep.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : trl_hrt_sym(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Translate <HrtCod> to heuristic string <HrtStr>           */
/*                                                                          */
/* PARAMETERS   : HrtCod                                                    */
/*              : HrtStr                                                    */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

trl_hrt_sym( s_hrt_non, "NON: no heuristic underestimate"):- !.

trl_hrt_sym( s_hrt_mpc, "MPC: minimum processor cost"):- !.

trl_hrt_sym( s_hrt_mia, "MIA: minimum independent assignment"):- !.

trl_hrt_sym( s_hrt_mrc, "MRC: maximum remaining critical-path"):- !.

trl_hrt_sym( s_hrt_all, "ALL: maximum of all underestimates"):- !.

trl_hrt_sym( s_hrt_nul, "--"):- !.

trl_hrt_sym(          _, "! unknown heuristic"):- beep.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : trl_utl_sym(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Translate <UtlCod> to utilisation string <UtlStr>         */
/*                                                                          */
/* PARAMETERS   : UtlCod                                                    */
/*              : UtlStr                                                    */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

trl_utl_sym( s_utl_sch, "SCH: relative to schedule end"):- !.

trl_utl_sym( s_utl_smp, "SMP: relative to sampling rate"):- !.

trl_utl_sym(          _, "! unknown utilisation"):- beep.

/* ************************************************************************ */
```

D.4.11 dcs_utl.pro

```
/* ************************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England   */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England               */
/* ************************************************************************** */

/* ************************************************************************** */
/* MODULE    : DCS_UTL.PRO                                                    */
/*                                                                            */
/* NOTES     : Utility predicates for DCS scheduling engine                   */
/*                                                                            */
/* AUTHOR    : Ken N LINTON                                                   */
/*                                                                            */
/* DATE      : 14th June, 1995                                                */
/*                                                                            */
/* VERSION   : 7.2                                                            */
/* ************************************************************************** */

/* ========================================================================== */
/* Project definition                                                         */
/* ========================================================================== */

project "dcs.prj"

/* ========================================================================== */
/* Public predicate definitions                                               */
/* ========================================================================== */

include "dcs_glb.pub"

include "dcs_clc.pub"
include "dcs_idb.pub"
include "dcs_lst.pub"
include "dcs_utl.pub"

/* ========================================================================== */
/* Private predicate definitions                                              */
/* ========================================================================== */

include "dcs_utl.prv"

/* ========================================================================== */
   clauses
/* ========================================================================== */

/* -------------------------------------------------------------------------- */
/* PREDICATE     : utl_srt_prl(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Quick-sort <TaskList> and bind <PrioList> to priority list */
/*                                                                            */
/* PARAMETERS    : TaskList                                                   */
/*               : PrioList                                                   */
/*                                                                            */
/* CALLS         : SELF                                                       */
/*               : lst_apd_lst(...)                                           */
/*               : utl_spl_tkl(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

utl_srt_prl( [], []):- !.

utl_srt_prl( [Head|Tail], PrioList):-
    utl_spl_tkl( Head, Tail, List1, List2),
    utl_srt_prl( List1, List1a),
    utl_srt_prl( List2, List2b),
    lst_apd_lst( List1a, [Head|List2b], PrioList).

/* -------------------------------------------------------------------------- */
/* PREDICATE     : utl_spl_tkl(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Classic quick-sort split predicate for utl_srt_prl(...).   */
/*                                                                            */
/* PARAMETERS    : Task                                                       */
/*               : TaskList1                                                  */
/*               : TaskList2                                                  */
/*               : TaskList                                                   */
/*                                                                            */
/* CALLS         : SELF                                                       */
/*               : utl_ord_tsk(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
/* BACKTRACKING  : determ                                                     */
/* -------------------------------------------------------------------------- */

utl_spl_tkl( tk(T2,E2), [tk(T1,E1)|X], [tk(T1,E1)|Y], Z):-
    utl_ord_tsk( T2,T1), !,
    utl_spl_tkl( tk(T2,E2), X, Y, Z).

utl_spl_tkl( tk(T2,E2), [tk(T1,E1)|X], Y, [tk(T1,E1)|Z]):- !,
    utl_spl_tkl( tk(T2,E2), X, Y, Z).

utl_spl_tkl( _, [], [], []).

/* -------------------------------------------------------------------------- */
/* PREDICATE     : utl_ord_tsk(...)                                           */
/*                                                                            */
/* DESCRIPTION   : Order <Task2>, <Task1> succeeds if <Task1> has higher level */
/*                                                                            */
/* PARAMETERS    : Task2                                                      */
/*               : Task1                                                      */
/*                                                                            */
/* CALLS         : idb_tsk_lvl(...)                                           */
/*                                                                            */
/* DATABASE      : --                                                         */
/*                                                                            */
```

```
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

utl_ord_tsk( Task2, Task1):-
    idb_tsk_lvl( Task1, Level1),
    idb_tsk_lvl( Task2, Level2), !,
    Level1 > Level2.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : utl_lbl_tkf(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Label taskforce with <PrecList> precedence relationships  */
/*              : according to <Labelling> static labelling scheme          */
/*                                                                          */
/* PARAMETERS   : Labelling                                                 */
/*              : PrecList                                                  */
/*                                                                          */
/* CALLS        : utl_ini_lvl(...)                                          */
/*              : utl_lbl_all(...)                                          */
/*                                                                          */
/* DATABASE    : --                                                         */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

utl_lbl_tkf( s_lbl_rnd, _):- !,
    utl_ini_lvl( s_ini_rnd).

utl_lbl_tkf( Labelling, PrecList):-
    utl_ini_lvl( s_ini_nil),
    utl_lbl_all( Labelling, extconnector, 0, PrecList),
    fail.

utl_lbl_tkf( _, _).

/* ------------------------------------------------------------------------ */
/* PREDICATE    : utl_ini_lvl(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Initialise <Labelling> levels for tasks in internal DB    */
/*                                                                          */
/* PARAMETERS   : Labelling                                                 */
/*                                                                          */
/* CALLS        : utl_chk_lst(...)                                          */
/*                                                                          */
/* DATABASE     : assertz( task(...))                                       */
/*              : retract( task(...))                                       */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

utl_ini_lvl( Labelling):-
    assertz( task( marker, marker)),
    retract( task( Task, Type)),
    utl_chk_lst( Labelling, Task, Type), !.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : utl_chk_lst(...)                                          */
/*                                                                          */
/* DESCRIPTION  : Label <TaskName> of type <TaskType> using <Labelling> and */
/*              : check if this task is last task in internal DB            */
/*                                                                          */
/* PARAMETERS   : Labelling                                                 */
/*              : TaskName                                                  */
/*              : TaskType                                                  */
/*                                                                          */
/* CALLS        : --                                                        */
/*                                                                          */
/* DATABASE     : assertz( task(...))                                       */
/*              : assertz( level(...))                                      */
/*                                                                          */
/* BACKTRACKING : determ                                                    */
/* ------------------------------------------------------------------------ */

utl_chk_lst( _, marker, _):- !.

utl_chk_lst( s_ini_rnd, TaskName, TaskType):- !,
    assertz( task( TaskName, TaskType)),
    random( 100, RandomLevel),
    assertz( level( TaskName, RandomLevel)),
    fail.

utl_chk_lst( s_ini_nil, TaskName, TaskType):-
    assertz( task( TaskName, TaskType)),
    assertz( level( TaskName, 0)),
    fail.

/* ------------------------------------------------------------------------ */
/* PREDICATE    : utl_lbl_all(...)                                          */
/*                                                                          */
/* DESCRIPTION  : <Labelling> label all predecessors of <Task>, with level  */
/*              : <Level> level, the list of precedence <PrecList>          */
/*                                                                          */
/* PARAMETERS   : LabelScheme                                               */
/*              : Task                                                      */
/*              : Level                                                     */
/*              : PrecList                                                  */
/*                                                                          */
/* CALLS        : SELF                                                      */
/*              : lst_del_ndt(...)                                          */
/*              : utl_est_exe(...)                                          */
/*              : utl_lvl_col(...)                                          */
/*              : utl_upd_lvl(...)                                          */
/*                                                                          */
/* DATABASE     : --                                                        */
/*                                                                          */
/* BACKTRACKING : non-determ                                                */
/* ------------------------------------------------------------------------ */

utl_lbl_all( LabelScheme, Task, Level, PrecList):-
    lst_del_ndt( Prec, PrecList, NewPrecList),
    utl_lvl_col( LabelScheme, Prec, Task, Task1),
    utl_est_exe( LabelScheme, Task1, ExecTime),
```

```
    NewLevel = Level + ExecTime,
    utl_upd_lvl( Task1, NewLevel),
    utl_lbl_all( LabelScheme, Task1, NewLevel, NewPrecList).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_lvl_col(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Using <Labelling>, determine if <Task2> precedes <Task1> */
/*               : in the precedence relation <Prec>                        */
/*                                                                          */
/* PARAMETERS    : LabelScheme                                              */
/*               : Prec                                                     */
/*               : Task1                                                    */
/*               : Task2                                                    */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

utl_lvl_col( s_lbl_hln, Prec, Task1, Task2):- !,
    Prec = prec( Task2, Task1).

utl_lvl_col( s_lbl_hle, Prec, Task1, Task2):- !,
    Prec = prec( Task2, Task1).

utl_lvl_col( s_lbl_lcn, Prec, Task1, Task2):- !,
    Prec = prec( Task1, Task2).

utl_lvl_col( s_lbl_lce, Prec, Task1, Task2):-
    Prec = prec( Task1, Task2).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_est_exe(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Using <Labelling>, estimate execution time <ExecTime> of */
/*               : of task with <TaskName>                                  */
/*                                                                          */
/* PARAMETERS    : Labelling                                                */
/*               : TaskName                                                 */
/*               : ExecTime                                                 */
/*                                                                          */
/* CALLS         : idb_tsk_exe(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

utl_est_exe( s_lbl_hln, _, 1):-                !.

utl_est_exe( s_lbl_hle, TaskName, ExecTime):- !,
    idb_tsk_exe( TaskName, ExecTime).

utl_est_exe( s_lbl_lcn, _, 1):-                !.

utl_est_exe( s_lbl_lce, TaskName, ExecTime):-
    idb_tsk_exe( TaskName, ExecTime).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_upd_lvl(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Update level of <Task> in internal DB if <NewLevel> is   */
/*               : is greater than current task level                       */
/*                                                                          */
/* PARAMETERS    : Task                                                     */
/*               : NewLevel                                                 */
/*                                                                          */
/* CALLS         : utl_uul_aux(...)                                         */
/*                                                                          */
/* DATABASE      : retract( level(...))                                     */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

utl_upd_lvl( Task, NewLevel):-
    retract( level( Task, TaskLevel)), !,
    utl_uul_aux( Task, TaskLevel, NewLevel).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_uul_aux(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Auxiliary predicate for utl_upd_lvl(...)                 */
/*                                                                          */
/* PARAMETERS    : Task                                                     */
/*               : TaskLevel                                                */
/*               : NewLevel                                                 */
/*                                                                          */
/* CALLS         : --                                                       */
/*                                                                          */
/* DATABASE      : assertz( level(...))                                     */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

utl_uul_aux( Task, TaskLevel, NewLevel):-
    NewLevel > TaskLevel, !,
    assertz( level( Task, NewLevel)).

utl_uul_aux( Task, TaskLevel, _):-
    assertz( level( Task, TaskLevel)).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : utl_cre_tsk(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Create list of task <TaskList> from facts in internal DB */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*                                                                          */
/* CALLS         : utl_tsk_aux(...)                                         */
```

```
/*                                                                      */
/* DATABASE     : assertz( task(...))                                   */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_cre_tsk( TaskList):-
    assertz( task("marker", "marker")),
    utl_tsk_aux( [], TaskList).

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_tsk_aux(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for utl_cre_tsk(...)              */
/*                                                                      */
/* PARAMETERS   : OldTaskList                                           */
/*              : TaskList                                              */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*              : utl_nxt_tsk(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_tsk_aux( OldTaskList, TaskList):-
    utl_nxt_tsk( TaskName, TaskType), !,
    assertz( task( TaskName, TaskType)),
    retract( tasktype( TaskType, ExecTime, Inputs, Outputs)), !,
    assert( tasktype( TaskType, ExecTime, Inputs, Outputs)),
    utl_tsk_aux( [tk(TaskName, ExecTime)|OldTaskList], TaskList).

utl_tsk_aux( TaskList, TaskList).

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_nxt_tsk(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Get name <TaskName> and type <TaskType> of next task from */
/*              : taskforce held in internal DB                         */
/*                                                                      */
/* PARAMETERS   : TaskName                                              */
/*              : TaskType                                              */
/*                                                                      */
/* CALLS        : --                                                    */
/*                                                                      */
/* DATABASE     : retract( task(...))                                   */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_nxt_tsk( TaskName, TaskType):-
    retract( task( TaskName, TaskType)),
    TaskName <> "extconnector", !,
    TaskName <> "marker".
```

```
/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_cre_pre(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Create list of precedence relationships <PrecList> from */
/*              : facts in internal DB                                  */
/*                                                                      */
/* PARAMETERS   : PrecList                                              */
/*                                                                      */
/* CALLS        : utl_pre_aux(...)                                      */
/*                                                                      */
/* DATABASE     : assertz( net(...))                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_cre_pre( PrecList):-
    assertz( net( "marker", "marker", ["marker"])),
    utl_pre_aux( [], PrecList).

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_pre_aux(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for utl_cre_pre(...)              */
/*                                                                      */
/* PARAMETERS   : OldPrecList                                           */
/*              : PrecList                                              */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*              : utl_fnt_str(...)                                      */
/*              : utl_net_out(...)                                      */
/*              : utl_nxt_net(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_pre_aux( OldPrecList, PrecList):-
    utl_nxt_net( NetInputPin, NetOutputPins), !,
    utl_fnt_str( NetInputPin, [".."], "", NetInputTask, _),
    utl_net_out( NetInputTask, NetOutputPins, OldPrecList, NewPrecList),
    utl_pre_aux( NewPrecList, PrecList).

utl_pre_aux( PrecList, PrecList).

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_nxt_net(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Get input <NetInput> and list of outputs <NetOutputs> for */
/*              : next net from taskforce held in internal DB           */
/*                                                                      */
/* PARAMETERS   : NetInput                                              */
/*              : NetOutPut                                             */
/*                                                                      */
/* CALLS        : --                                                    */
/*                                                                      */
```

```
/* DATABASE     : retract( net(...))                                    */
/*              : assertz( net(...))                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_nxt_net( NetInput, NetOutput):-
    retract( net( NetName, NetInput, NetOutput)), !,
    NetName <> "marker",
    assertz( net( NetName, NetInput, NetOutput)).

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_net_out(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Create precedence relations <Precs> for all tasks in  */
/*              : <OutTasks> which are fed by <InTask>                  */
/*                                                                      */
/* PARAMETERS   : InTask                                                */
/*              : OutTasks                                              */
/*              : OldPrecs                                              */
/*              : Precs                                                 */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*              : utl_fnt_str(...)                                      */
/*                                                                      */
/* DATABASE     : assert( prec(...))                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_net_out( _, [], Precs, Precs):- !.

utl_net_out( InTask, [OutTask|RestOutTasks], OldPrecs, Precs):-
    utl_fnt_str( OutTask, [".“], "", OutTN, _),
    assert( prec( InTask, OutTN)),
    utl_net_out( InTask, RestOutTasks, [prec(InTask,OutTN)|OldPrecs], Precs).

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_fnt_str(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Bind <OutStr> to the first part of <InStr> before any one */
/*              : of the delimiters in <Delimiters>. <RestStr> is bound */
/*              : to the part of <InStr> following the delimiter        */
/*                                                                      */
/* PARAMETERS   : InStr                                                 */
/*              : Delimiters                                            */
/*              : ConCatStr                                             */
/*              : OutStr                                                */
/*              : RestStr                                               */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_fnt_str( InStr, Delimiters, ConCatStr, OutStr, RestStr):-
    fronttoken( InStr, Token, NewRestStr),
    not( lst_mem_elm( Token, Delimiters)), !,
    concat( ConCatStr, Token, NewConCatStr),
    utl_fnt_str( NewRestStr, Delimiters, NewConCatStr, OutStr, RestStr).

utl_fnt_str( InStr, _, OutStr, OutStr, RestStr):-
    fronttoken( Instr, _, RestStr).

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_cre_pro(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Create list of processors <ProcList> from facts held in */
/*              : internal DB                                           */
/*                                                                      */
/* PARAMETERS   : ProcList                                              */
/*                                                                      */
/* CALLS        : utl_pro_aux(...)                                      */
/*                                                                      */
/* DATABASE     : assertz( proc(...))                                   */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_cre_pro( ProcList):-
    assertz( proc( "marker", "marker")),
    utl_pro_aux( [], ProcList).

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_pro_aux(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for utl_cre_pro(...)              */
/*                                                                      */
/* PARAMETERS   : OldProcList                                           */
/*              : ProcList                                              */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*              : utl_nxt_pro(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_pro_aux( OldProcList, ProcList):-
    utl_nxt_pro( ProcName, ProcType), !,
    assertz( proc( ProcName, ProcType)),
    retract( proctype( ProcType, InstCycles)), !,
    assert( proctype( ProcType, InstCycles)),
    utl_pro_aux( [pr(ProcName, [tk(idle,0)])|OldProcList], ProcList).

utl_pro_aux( ProcList, ProcList).

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_nxt_pro(...)                                      */
```

```
/*                                                                      */
/* DESCRIPTION  : Get name <ProcName> and type <ProcType> of next processor  */
/*              : from hardware description held in internal DB          */
/*                                                                      */
/* PARAMETERS   : ProcName                                              */
/*              : ProcType                                              */
/*                                                                      */
/* CALLS        : --                                                    */
/*                                                                      */
/* DATABASE     : retract( proc(...))                                   */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_nxt_pro( ProcName, ProcType):-
    retract( proc( ProcName, ProcType)), !,
    ProcName <> "marker".

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_rep_eat(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Generate an unlimited supply of backtracking points   */
/*                                                                      */
/* PARAMETERS   : --                                                    */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : non-determ                                            */
/* -------------------------------------------------------------------- */

utl_rep_eat().

utl_rep_eat():-
    utl_rep_eat().

/* -------------------------------------------------------------------- */
/* PREDICATE    : utl_kyd_pau(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Pause until any character is received from keyboard   */
/*                                                                      */
/* PARAMETERS   : --                                                    */
/*                                                                      */
/* CALLS        : utl_rep_eat(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

utl_kyd_pau():-
    utl_rep_eat(),
    readchar( _), !.

/* ******************************************************************** */
```

D.4.12 dcs_wri.pro

```
/* ******************************************************************** */
/* © 1995    : School of Engineering, University of Durham, Durham, England  */
/*           : Solid State Logic Ltd, Begbroke, Oxford, England         */
/* ******************************************************************** */

/* ******************************************************************** */
/* MODULE    : DCS_WRI.PRO                                              */
/*                                                                      */
/* NOTES     : Display predicates for DCS internal lists and other structures */
/*                                                                      */
/* AUTHOR    : Ken N LINTON                                             */
/*                                                                      */
/* DATE      : 1st March, 1995                                          */
/*                                                                      */
/* VERSION   : 2.1                                                      */
/* ******************************************************************** */

/* ==================================================================== */
/* Project definition                                                   */
/* ==================================================================== */

project "dcs.prj"

/* ==================================================================== */
/* Public predicate definitions                                         */
/* ==================================================================== */

include "dcs_glb.pub"

include "dcs_clc.pub"
include "dcs_idb.pub"
include "dcs_lst.pub"
include "dcs_trl.pub"
include "dcs_wri.pub"

/* ==================================================================== */
/* Private predicate definitions                                        */
/* ==================================================================== */

include "dcs_wri.prv"

/* ==================================================================== */
   clauses
/* ==================================================================== */

/* -------------------------------------------------------------------- */
/* PREDICATE    : wri_inp_prm(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Write input parameters including taskforce filename   */
/*              : <TkfFil>, hardware filename <HdwFil> and algorithm control */
/*              : parameters <AlgCtl> to default output device          */
/*                                                                      */
```

```
/* PARAMETERS   : TkfFil                                                  */
/*              : HdwFil                                                  */
/*              : AlgCtl                                                  */
/*                                                                        */
/* CALLS        : trl_alg_sym(...)                                        */
/*              : trl_ipc_sym(...)                                        */
/*              : trl_lbl_sym(...)                                        */
/*              : trl_dyn_sym(...)                                        */
/*              : trl_hrt_sym(...)                                        */
/*              : wri_dlm_lin(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

wri_inp_prm( TkfFil, HdwFil, ac(Alg, Ipc, Lbl, Dyn, Hrt)):-
    wri_dlm_lin( '-'),
    write( "    Inputs: \n"),
    wri_dlm_lin( '-'),
    % write taskforce and hardware files
    write( "    console taskforce       : ", TkfFil, "\n"),
    write( "    hardware architecture   : ", HdwFil,  "\n\n"),
    % translate algorithm control symbols
    trl_alg_sym( Alg, AlgStr),
    trl_ipc_sym( Ipc, IpcStr),
    trl_lbl_sym( Lbl, LblStr),
    trl_dyn_sym( Dyn, DynStr),
    trl_hrt_sym( Hrt, HrtStr),
    % write algorithm control parameters
    write( "    scheduling strategy     : ", AlgStr, "\n"),
    write( "    IPC scheme              : ", IpcStr, "\n"),
    write( "    static labelling scheme : ", LblStr, "\n"),
    write( "    dynamic labelling scheme: ", DynStr, "\n"),
    write( "    heuristic underestimate : ", HrtStr, "\n"),
    wri_dlm_lin( '-'), nl.

/* ---------------------------------------------------------------------- */
/* PREDICATE    : wri_pre_stt(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write pre-schedule statistics to default output device  */
/*                                                                        */
/* PARAMETERS   : TaskList                                                */
/*              : PrecList                                                */
/*              : ProcList                                                */
/*                                                                        */
/* CALLS        : lst_num_elm(...)                                        */
/*              : clc_seq_sch(...)                                        */
/*              : clc_tsk_par(...)                                        */
/*              : clc_tst_grn(...)                                        */
/*              : clc_hdw_rsc(...)                                        */
/*              : wri_dlm_lin(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

wri_pre_stt( TaskList, PrecList, ProcList):-
    wri_dlm_lin( '-'),
    write( "    Pre-Schedule Statistics: \n"),
    wri_dlm_lin( '-'),
    % number list elements
    lst_num_elm( TaskList, NumTasks),
    lst_num_elm( PrecList, NumPrecs),
    lst_num_elm( ProcList, NumProcs),
    % calc additional statistics
    clc_seq_sch( TaskList, SeqlSchd),
    clc_tsk_par( TaskList, PrecList, CritPath, TaskPara),
    clc_tsk_grn( TaskList, MinmGran, MaxmGran, AvrgGran),
    clc_hdw_rsc( ProcList, ProcRsrc),
    % write number of elements
    writef( "    number of tasks, m      : %6d           \n", NumTasks),
    writef( "    precedence relations    : %6d           \n", NumPrecs),
    writef( "    number of processors, n : %6d         \n\n", NumProcs),
    % write static taskforce statistics
    writef( "    minimum task granularity: %6d    cycles \n", MinmGran),
    writef( "    maximum task granularity: %6d    cycles \n", MaxmGran),
    writef( "    average task granularity: %9.2f cycles\n\n", AvrgGran),
    % write dynamic taskforce statistics
    writef( "    sequential schedule     : %6d    cycles \n", SeqlSchd),
    writef( "    critical path length    : %6d    cycles \n", CritPath),
    writef( "    taskforce parallelism   : %9.2f       \n\n", TaskPara),
    % write hardware resource
    writef( "    total hardware resource : %6f    cycles \n", ProcRsrc),
    wri_dlm_lin( '-'), nl.

/* ---------------------------------------------------------------------- */
/* PREDICATE    : wri_pre_chk(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write pre-schedule checks to default output device and bind */
/*              : <TestList> to list of check results                     */
/*                                                                        */
/* PARAMETERS   : TaskList                                                */
/*              : PrecList                                                */
/*              : ProcList                                                */
/*              : TestList                                                */
/*                                                                        */
/* CALLS        : dcs_grn_tst(...)                                        */
/*              : dcs_fit_tst(...)                                        */
/*              : dcs_ctp_tst(...)                                        */
/*              : wri_dlm_lin(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

wri_pre_chk( [GranTest, FitTest, CritTest]):-
    wri_dlm_lin( '-'),
    write( "    Pre-Schedule Checks: \n"),
    wri_dlm_lin( '-'),
```

```
    wri_dcs_tst( "maximum task granularity:", GranTest),
    wri_dcs_tst( "total resource capacity :", FitTest),
    wri_dcs_tst( "critical path vs.sample :", CritTest),
    wri_dlm_lin( '-'), nl.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_wat_dog(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Retract search-tree watch-dogs from internal DB and display */
/*                                                                          */
/* PARAMETERS    : AlgCtl                                                   */
/*                                                                          */
/* CALLS         : wri_dlm_lin(...)                                         */
/*                                                                          */
/* DATABASE      : retract( num_sea_nod(...))                               */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

wri_wat_dog( ac(s_alg_ast, _, _, _, _)):- !,
    wri_dlm_lin( '-'),
    write( "    Search-Tree Watch Dog: \n"),
    wri_dlm_lin( '-'),
    retract( num_sea_nod( SearchNodes)), !,
    writef( "    number of search nodes  : %d\n", SearchNodes),
    wri_dlm_lin( '-'), nl.

wri_wat_dog( _).

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_mtp_sch(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Output M-P schedule held in <Schedule> to default device */
/*                                                                          */
/* PARAMETERS    : Schedule                                                 */
/*                                                                          */
/* CALLS         : wri_dlm_lin(...)                                         */
/*               : wri_pro_lst(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

wri_mtp_sch( _):-
    wri_dlm_lin( '-'),
    write( "    Multi-Processor Schedule: \n"),
    wri_dlm_lin( '-'),
    fail.

wri_mtp_sch( Schedule):-
    wri_pro_lst( Schedule),
    fail.

wri_mtp_sch( []):-
    write( "    ! DCS failed to generate a feasible schedule\n"),
    beep,
    fail.

wri_mtp_sch( _):-
    wri_dlm_lin( '-'), nl.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_pst_stt(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Write post-schedule statistics to default output device */
/*                                                                          */
/* PARAMETERS    : TaskList                                                 */
/*               : Schedule                                                 */
/*                                                                          */
/* CALLS         : clc_run_tim(...)                                         */
/*               : clc_fin_tim(...)                                         */
/*               : clc_tsk_spd(...)                                         */
/*               : clc_thr_put(...)                                         */
/*               : clc_deq_sch(...)                                         */
/*               : wri_dlm_lin(...)                                         */
/*                                                                          */
/* DATABASE      : --                                                       */
/*                                                                          */
/* BACKTRACKING  : determ                                                   */
/* ------------------------------------------------------------------------ */

wri_pst_stt( _, _, Schedule):-
    wri_dlm_lin( '-'),
    write( "    Post-Schedule Statistics: \n"),
    wri_dlm_lin( '-'),
    clc_run_tim( RunTime),
    clc_fin_tim( Schedule, FinTime),
    writef( "    algorithm run-time       : %s              \n", RunTime),
    writef( "    schedule length          : %6d    cycles\n\n", FinTime),
    fail.

wri_pst_stt( ac( s_alg_ast, _, _, _, _), _, _):-
    retract( num_sea_nod( NumNode)),
    writef( "    number of search nodes   : %6f    nodes \n\n", NumNode),
    fail.

wri_pst_stt( _, TaskList, Schedule):-
    clc_tsk_spd( TaskList, Schedule, SpeedUp),
    clc_thr_put( TaskList, Schedule, ThruPut),
    clc_seq_sch( TaskList, SeqlSchd),
    writef( "    taskforce speed-up       : %9.2f %%            \n", SpeedUp),
    writef( "    schedule throughput      : %9.2 tasks/sample\n\n", ThruPut),
    wri_pro_utl( s_utl_sch, Schedule, SeqlSchd), nl,
    wri_pro_utl( s_utl_smp, Schedule, SeqlSchd),
    wri_dlm_lin( '-'), nl.

/* ------------------------------------------------------------------------ */
/* PREDICATE     : wri_alg_mon(...)                                         */
/*                                                                          */
/* DESCRIPTION   : Write block to algorithm monitor gauge                   */
/*                                                                          */
```

```
/* PARAMETERS   : --                                                    */
/*                                                                      */
/* CALLS        : --                                                    */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

wri_alg_mon( ):-
    existwindow( c_win_mon), !,
    writedevice( OldDev),
    writedevice( screen),
    write( "±"),
    writedevice( OldDev).

wri_alg_mon( ).

/* -------------------------------------------------------------------- */
/* PREDICATE    : wri_dcs_tst(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Translate <PassFail> and write to default output device */
/*                                                                      */
/* PARAMETERS   : Prompt                                                */
/*              : PassFail                                              */
/*                                                                      */
/* CALLS        : --                                                    */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

wri_dcs_tst( Prompt, s_tst_pas):- !,
    writef( "    %s %s\n", Prompt, "passed").

wri_dcs_tst( Prompt, s_tst_fai):-
    writef( "    %s %s\n", Prompt, "failed").

/* -------------------------------------------------------------------- */
/* PREDICATE    : wri_dlm_lin(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Write delimiter string of <Chr> into scheduler output file */
/*                                                                      */
/* PARAMETERS   : Chr                                                   */
/*                                                                      */
/* CALLS        : wri_wdl_aux(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

wri_dlm_lin( Chr):-
    textmode( _, Cols),
    wri_wdl_aux( Chr, Cols).

/* -------------------------------------------------------------------- */
/* PREDICATE    : wri_wdl_aux(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Auxiliary predicate for wri_dlm_lin(...)              */
/*                                                                      */
/* PARAMETERS   : Chr                                                   */
/*              : Cols                                                  */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

wri_wdl_aux( _, 0):- !,
    write( "\n").

wri_wdl_aux( Character, OldNum):-
    NewNum= OldNum - 1,
    write( Character),
    wri_wdl_aux( Character, NewNum).

/* -------------------------------------------------------------------- */
/* PREDICATE    : wri_pre_lst(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Write precedence relations in <PrecList> to output device */
/*                                                                      */
/* PARAMETERS   : PrecList                                              */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*                                                                      */
/* DATABASE     : --                                                    */
/*                                                                      */
/* BACKTRACKING : determ                                                */
/* -------------------------------------------------------------------- */

wri_pre_lst( []):- !.

wri_pre_lst( [Prec|RestPrecs]):-
    write( Prec, "\n"),
    wri_pre_lst( RestPrecs).

/* -------------------------------------------------------------------- */
/* PREDICATE    : wri_pro_lst(...)                                      */
/*                                                                      */
/* DESCRIPTION  : Write processor states in <ProcList> to output device */
/*                                                                      */
/* PARAMETERS   : ProcList                                              */
/*                                                                      */
/* CALLS        : SELF                                                  */
/*              : lst_rev_lst(...)                                      */
/*              : wri_tsk_lst(...)                                      */
/*                                                                      */
/* DATABASE     : --                                                    */
```

```
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

wri_pro_lst( []):- !.

wri_pro_lst( [pr( ProcName, Tasks)|RestProcs]):-
    writef( "    %s:\n", ProcName),
    lst_rev_lst( Tasks, RevTasks),
    wri_tsk_lst( RevTasks),
    wri_pro_lst( RestProcs).

/* ---------------------------------------------------------------------- */
/* PREDICATE    : wri_pro_utl(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write processor utilisations of type <UtilSym> exhibited in */
/*              : <Schedule> for tasks with sequential schedule <SeqCost> */
/*                                                                        */
/* PARAMETERS   : UtilSym                                                 */
/*              : Schedule                                                */
/*              : SeqCost                                                 */
/*                                                                        */
/* CALLS        : lst_num_elm(...)                                        */
/*              : wri_wpu_aux(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

wri_pro_utl( UtilSym, Schedule, SeqCost):-
    trl_utl_sym( UtilSym, UtilStr),
    writef( "    processor utilisations  : %s\n", UtilStr),
    wri_wpu_aux( UtilSym, Schedule, SeqCost, 0, SumUtil),
    lst_num_elm( Schedule, NumProcs),
    MeanProcUtil = SumUtil / NumProcs,
    writef( "        mean    : %6.2f%%\n", MeanProcUtil).

/* ---------------------------------------------------------------------- */
/* PREDICATE    : wri_wpu_aux(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Auxiliary predicate for wri_pro_utl(...)                */
/*                                                                        */
/* PARAMETERS   : UtilSym                                                 */
/*              : Schedule                                                */
/*              : FinTime                                                 */
/*              : OldSumUtil                                              */
/*              : SumUtil                                                 */
/*                                                                        */
/* CALLS        : SELF                                                    */
/*              : clc_pro_utl(...)                                        */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

wri_wpu_aux( _, [], _, SumUtil, SumUtil):- !.

wri_wpu_aux( UtilSym, [pr(ProcName, Tasks)|Procs], FinTime, OldSum, SumUtil):-
    clc_pro_utl( UtilSym, ProcName, Tasks, FinTime, ProcUtil),
    writef( "        %s: %6.2f%%\n", ProcName, ProcUtil),
    NewSum = OldSum + ProcUtil,
    wri_wpu_aux( UtilSym, Procs, FinTime, NewSum, SumUtil).

/* ---------------------------------------------------------------------- */
/* PREDICATE    : wri_tsk_lst(...)                                        */
/*                                                                        */
/* DESCRIPTION  : Write tasks in <TaskList> to default output device      */
/*                                                                        */
/* PARAMETERS   : TaskList                                                */
/*                                                                        */
/* CALLS        : SELF                                                    */
/*                                                                        */
/* DATABASE     : --                                                      */
/*                                                                        */
/* BACKTRACKING : determ                                                  */
/* ---------------------------------------------------------------------- */

wri_tsk_lst( []):- !.

wri_tsk_lst( [tk(idle, ExecTime)|RestTasks]):- !,
    writef( "            %s, %d\n", "idle", ExecTime),
    wri_tsk_lst( RestTasks).

wri_tsk_lst( [tk(TaskName, ExecTime)|RestTasks]):-
    writef( "        %s, %d\n", TaskName, ExecTime),
    wri_tsk_lst( RestTasks).

/* ********************************************************************** */
```

Appendix E

Original Publications

The research contained in this thesis has been reported both at international conference and invited elocution.

E.1 Conference Papers

Work described herein has resulted in the following conference publications:

- Linton K N, Terepin S, and Purvis A: "Parallel Digital Signal Processing for Audio Engineering", *88th Audio Engineering Society Convention*, Montreux, Switzerland, March 1990, preprint 2917 (H-2).
- Linton K N, Gould G L, Terepin S, and Purvis A: "Optimising Massive Parallel Architectures for Real-Time Digital Audio", *89th Audio Engineering Society Convention*, Los Angeles, USA, September 1990, preprint 2972 (F-II-6).
- Linton K N, Gould G L, Terepin S, and Purvis A: "Multiprocessor Architectures and Allocation Strategies for Digital Audio Mixing Consoles", *Reproduced Sound 6*, Windermere, Great Britain, November 1990.
- Linton K N, Terepin S, and Purvis A: "Task Scheduling Strategies for Digital Mixing Consoles", *90th Audio Engineering Society Convention*, Paris, France, February 1991, preprint 3075 (G-3).
- Linton K N, Gould G L, Terepin S, and Purvis A: "Real-Time Multi-Channel Digital Audio Processing: Scalable Parallel Architectures and Taskforce Scheduling Strategies", *1991 International Conference on Acoustics Speech and Signal Processing*, Toronto, Canada, May 1991, paper A2.7.†

---

†. Reprinted in this thesis as Appendix F.

- Linton K N, Gould G L, Terepin S, and Purvis A: "On the *Re*-Allocation of Hardware Resources for Digital Audio Signal Processing", *IEE Colloquium on Digital Audio Signal Processing*, London, UK, May 1991, pp. 7/1-7/4.
- Gould G L, Linton K N, Terepin S, and Purvis A: "A Scalable Hybrid Multiprocessor for Real-Time Digital Audio Signal Processing", *IEE Colloquium on Digital Audio Signal Processing*, London, UK, May 1991, pp. 9/1-9/4.

E.2 Invited Papers

Following invitation, aspects of the research reported here have also been presented in:

- Linton K N: *Real-Time Digital Audio Processing: Parallel Architectures and Resource Allocation Techniques*, Invited Lecture to the British Section of the Audio Engineering Society, IBA, London, Great Britain, July 1991.
- Linton K N: "Architectural Issues for Parallel DSP Audio Systems", Invited Paper presented at the *AES UK DSP Conference*, London, September 1992, pp. 24-36.†

†. Reprinted in this thesis as Appendix G.

Appendix F

ICASSP '91 Paper Reprint

(See over for a reprint of the paper presented by the author at the IEEE International Conference on Acoustics Speech and Signal Processing, Toronto, Canada, May 1991).

REAL-TIME MULTI-CHANNEL DIGITAL AUDIO PROCESSING: Scalable Parallel Architectures and Taskforce Scheduling Strategies

Ken N Linton†, G Lee Gould†, Stephen Terepin#, and Alan Purvis†

† Audio Engineering Research Group, SEAS, University of Durham, Durham, England, DH1 3LE
# Solid State Logic Limited, Begbroke, Oxford, England, OX5 1RU

ABSTRACT

In this paper we address the problems associated with the implementation of large-scale digital audio signal processing systems. For example, the audio mixing console requires the realisation of an architecture comprising in the order of 100 digital signal processors (DSPs). A hybrid multiprocessor, offering efficient computation with minimal communication overhead, is described and its suitability to the real-time environment discussed. Task allocation has a substantial impact on the efficient utilisation of such a DSP-engine. We are interested in algorithms that can produce optimal schedules in reasonable time and fast sub-optimal algorithms. An integrated software environment allows graphical input of DSP taskforces and hardware resource schematics, and the assessment of preferred scheduling strategies. We summarise the theoretical background, discuss the results obtained, and assess the implications of this research to the real-time audio processing environment.

1 Introduction

A number of multiprocessor architectures have been proposed in response to an ever-growing need for speeding up DSP-intensive applications. With the widespread availability of programmable DSP devices, a single scalable multiprocessor exhibits the ability to provide good performance for a range of real-time digital audio applications. One such application is the audio mixing console which, when implemented digitally, demands a computation engine rated in thousands of MIPS. With current technology, a topology comprising in the order of a 100 devices must be realised.

In order to efficiently develop such a system and allow unused processing power to be reallocated by the user in the operating environment, task scheduling strategies are necessary to efficiently distribute the set of processing algorithms required — termed the *taskforce* — across the architecture. A good scheduling algorithm will maximise and balance the utilisation of processing resources, and minimise the communication between processors. In addition any strategy must observe all precedence relations, must be non-preemptive, and must not allow distributed execution of any task.

Sections 1 and 2 describe the preferred approaches to this problem, and their relative performance is discussed in Section 3. In Section 4 the architecture of the hybrid multiprocessor is detailed, with Sections 5 and 6 reviewing the data transfer techniques used. Finally, the current state of this research project and proposed further work is summarised.

2 Dynamic Programming

Dynamic Programming is employed in problems where finding a solution can be reduced to a sequence of interrelated decisions forming a path in a state-space. Scheduling taskforces on a multiprocessor architecture can be cast in these terms, as shown in Fig. 1. Each alternative decision from a node corresponds to scheduling a different task, or idle period, on a processor. At each step the non-intuitive option of scheduling an idle period, lasting until the next processor has completed its currently assigned task, must be allowed to guarantee optimality. Here the search is termed informed, as a set of alternative candidate nodes at each decision is kept, corresponding to the whole growing edge of the search tree.

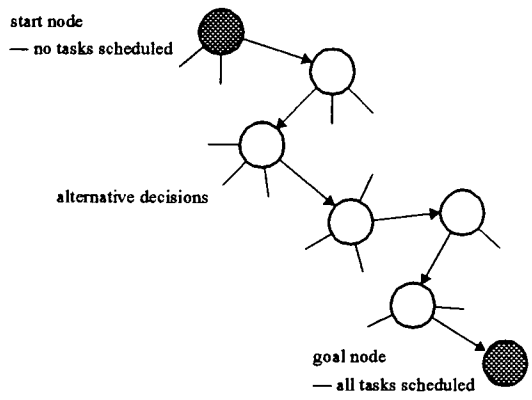

Fig. 1: Task scheduling in terms of state-space searching.

The A* algorithm employs the best-first principle at each decision point, combined with problem-specific heuristics to draw the search process quickly towards a goal node, thereby avoiding aimless paths. Our implementation uses the minimum processor cost underestimate [1]. This corresponds to the optimistic estimate of the finishing time of the partial schedule, completed with all currently unassigned tasks. It is computed under the assumption that both the precedence relations, and the non-preemption and distributed execution constraints are relaxed.

3 Critical Path Method

In list scheduling strategies the tasks are first ordered according to some external priority scheme, and then scheduled according to this regime. One technique used to form such a priority list is the Critical Path Method, originally investigated by Hu [2].

Although only optimal for taskforces consisting of equal duration tasks and exhibiting in-tree precedence constraints, the concept of *Hu-level* is widely applicable. A task t_i is given the label x_i, where x_i is the length of the longest path from t_i to the root task, which is given the label '0'. Tasks that are one unit removed from the root task are given the label '1', and so on as Fig. 2. The priority list is then ordered by non-increasing Hu-level, and the taskforce scheduled according to this regime.

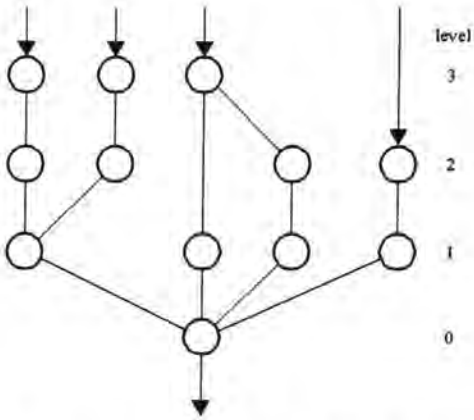

Fig. 2: Hu-level labelling of a console taskforce.

Typically several root tasks are present, corresponding to a console taskforce with multiple outputs. In these cases, an artificial or "dummy" root task can be added to the taskforce. Tasks with execution times greater than one are represented by an indivisible chain of unit-weight tasks whose sum equals the weight of the original task. It can be shown that the level of the chain-task furthest from the root is the same as the level of the multi-unit task.

4 Scheduling Results

As a simple example, the 4:1 console taskforce of Fig. 3 requires 136 cycles of sequential processing, has an ideal schedule length of 62 cycles, and (ignoring bus delays) can be allocated optimally onto 4 processors as Fig. 4 and Table 1. Investigations with full channel strip processing enabled show that the A* strategy results in considerable scheduling overhead when compared to the CPM approach. Furthermore, such large problem instances can result in extremely large search trees as the A* policy may degenerate towards an exhaustive search [3].

The level of performance achieved by the Hu-level Critical Path algorithm is typically well inside the theoretical performance bound of a factor of 2 from optimal. Results indicate that this approach produces such schedules in relatively short execution times. This can be attributed to the fact that in most cases console taskforces are configured as trees, or a forest of trees. In particular, the reader's attention is drawn to the allocation of a 16:8 console onto a eight processor architecture. Dynamics processing functionality was removed from four of the sixteen channels, allowing this 16:8 taskforce to be squeezed within the constraints imposed by the professional sampling rate [4].

In such large taskforces replacing the 2-input mix tasks with 8-input primitives, requiring 10 instruction cycles, can greatly

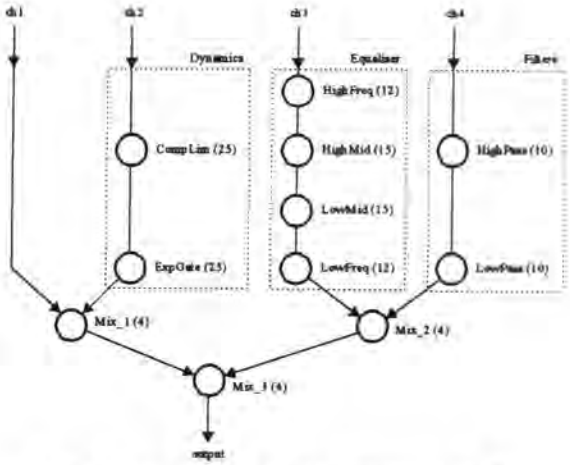

Fig. 3: Example 4-input:1-output console taskforce.

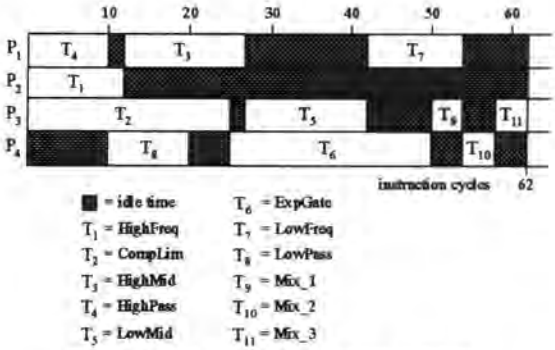

Fig. 4: Optimal allocation of example taskforce.

Algorithm	Sequential Schedule	Parallel Schedule	Algorithm Runtime	Speed-up	Processor Efficiencies
A*	136	62	4:73	219.4%	59.7% 19.4% 77.4% 62.9%
Hu-level	136	62	1:93	219.4%	59.7% 19.4% 77.4% 62.9%

Table 1: Performance assessments for above schedule.

reduce the scheduling overhead. This is due to two reasons: firstly, the number of tasks is considerably reduced, thereby significantly reducing the run-time of the A* and, to a lesser degree, CPM algorithm; secondly, the range of task size ratio is reduced by a factor of 2.6, so reducing the number of A* search-space nodes. However, this loss of relatively small tasks, and the tight packing previously available, means that high processor efficiencies are not as readily achieved.

5 Hybrid Hardware Module

The Inmos transputer is capable of parallel communication and computation, providing an ideal building block for the construction of scalable concurrent systems. However, the transputer is not effective at executing the data flow algorithms

required in real-time multi-channel digital audio applications [5]. In contrast, DSPs such as the Motorola DSP56001 are specifically designed to efficiently implement such algorithms [6]. Due to inherent communication overheads, scalable architectures are not easily implemented with these devices alone. A hybrid multiprocessor assembled from nodes consisting of several Motorola DSP56001s hosted by a single transputer offers efficient computation with minimal communication overhead.

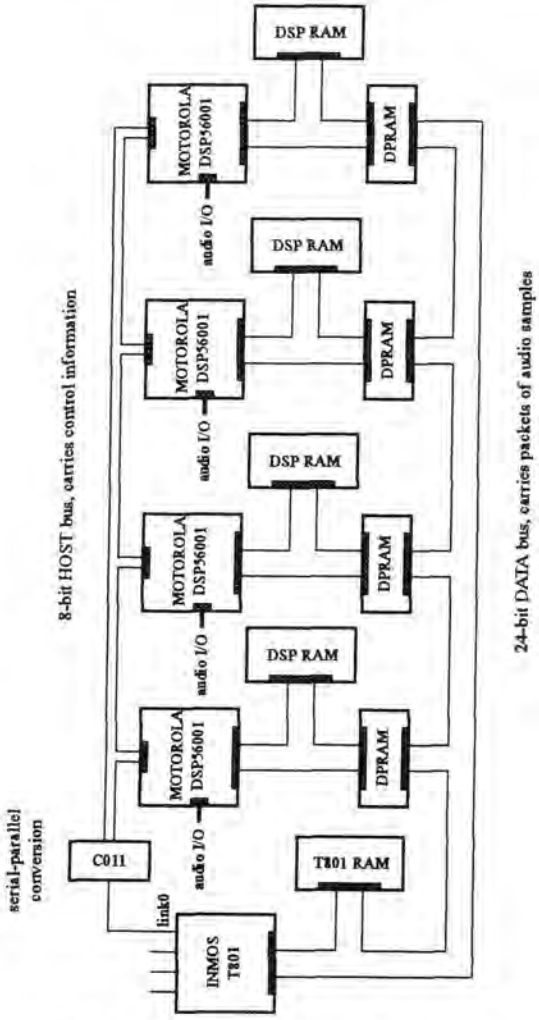

Fig. 5: Schematic diagram of hybrid processing node.

Inter-node communication is carried out using the transputer serial links, with one link reserved for broadcasting low bandwidth control data to the DSPs via a IMSC011 link adapter. Assuming a sampling frequency of 48kHz and that block moves are used for data transfer, then each transputer is capable of supporting up to 34 audio paths in this trivalent configuration, Fig. 6. Intra-node communication is achieved through the non-multiplexed external memory interface (EMI) of the IMST801 part. Each DSP possess an exclusive block of dual-ported RAM (DPR): the other port of the DPR is mapped into the transputer's EMI via appropriate decode logic.

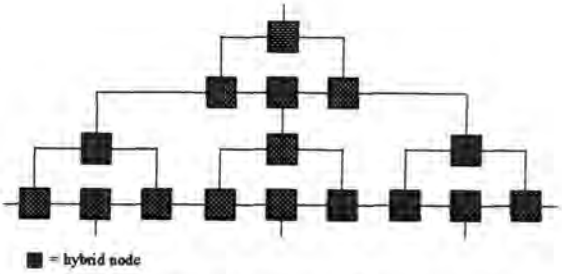

Fig. 6: Ternary tree configuration with hanging pipeline.

6 Intra-Node Communication

The transputer controls data transfer within the processing node. Data vectors pertaining to a particular DSP are externally placed by the transputer into the appropriate DPR. An arbitrary, and time varying, *soft* configuration of the processors may be defined in software and mapped onto the hard configuration imposed by the hardware. Thus complex configurations, such as a hypercube or any random topology, may be realised with this system. Data integrity and synchronisation is assured by the use of semaphores [7]. The DPR is divided into domains which contain both input and output vectors, and an associated Boolean semaphore as Fig. 7. A processor will only effect a data transfer if the semaphore is in the correct state.

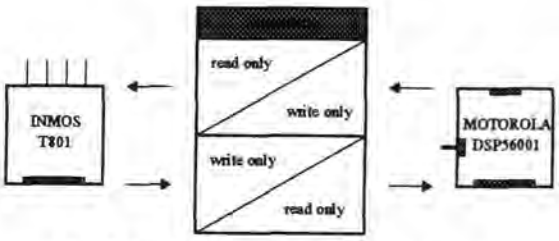

Fig. 7: Single domain DPR partitioning scheme.

Two of the major problems that arise in multi-processor systems are bus bottlenecking and spin-locking [8]. Bus bottlenecking is considerably reduced in this architecture as each DSPs bus is processor exclusive. Spin-locking occurs when a processor repeatedly tests and fails a semaphore, resulting in increased processor idle time. Although appropriate scheduling strategies can minimise spin-locking, a proposed extension to the single domain scheme promises to reduce the effects further. Here the DPR is partitioned into two domains, each of which is accessed in a cyclic manner by both the transputer and the DSPs.

7 Semaphore Management

It is important to reduce the overhead associated with accessing and testing semaphores as much as possible. The DSP sees the semaphore as occupying a single word in external memory space and it is a simple matter to read a semaphore and to act upon its state. The approach taken with the transputer is somewhat different, however. Time critical sections of code are written in transputer assembly for optimum performance and embedded in an Occam2 harness, as Fig. 8. The transputer uses a byte-oriented addressing scheme, beginning at the most negative address and working upwards. Instructions may be

stripped, and execution time minimised, by treating the semaphore as boolean values occupying contiguous byte locations in external memory space. This requires that two address decode schemes be implemented for the transputer.

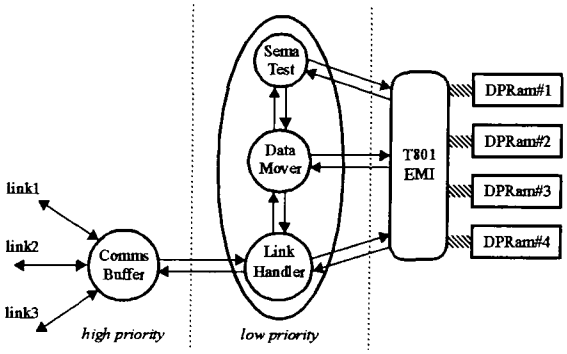

Fig. 8: Occam2 harness for dependent data paths.

The first, mapping contiguous blocks of DPR into contiguous words of transputer memory, is used for data areas. The second, mapping non-contiguous bytes contained in different DPR domains into contiguous bytes of transputer memory, for the semaphores. Due to the prefixing nature of transputer code, the semaphores may only occupy the sixteen locations beginning at byte location 0 to ensure zero prefixing. A restriction with this approach is that a maximum of sixteen semaphores may be used for maximum performance, and that additional address decoding needs to be utilised. It follows that a maximum of eight DSPs, using the twin domain approach, may be supported by a single transputer at full semaphore access speed.

8 Conclusions

These encouraging results indicate that the sub-optimal strategy, based on the Hu-level Critical Path Method, consistently produces good-quality schedules with minimum overhead. Correspondingly this approach has direct application for the reallocation of unused processing resources by the engineer in the operating environment. In contrast, the optimal A* Dynamic Programming implementation may be utilised in the development cycle when assessing alternative multiprocessor designs, or in the production of pre-defined allocations yielding maximum system performance for a given architecture. Further results extend this concept by sacrificing channel processing to provide reverberation and other time-domain effects processing on a given hardware resource.

A target hybrid multiprocessor architecture has been described. The multiprocessor combines the efficiency of communication exhibited by the Inmos IMST801 transputer, and the computational power of the Motorola DSP56001. This novel architecture is fully scalable and capable of supporting additional processing elements whilst incurring negligible communications overhead. Further research will investigate the application of these strategies to architectures configured from the revolutionary Texas Instruments TMS320C40 part, and techniques for embedding resource re-allocation methods on such platforms [9].

Acknowledgements

This research programme is supported jointly by Solid State Logic Limited, British Gas PLC, the University of Durham, and the Department of Education for Northern Ireland.

References

[1] **Sinclair** J B: "Efficient Computation of Optimal Assignments for Distributed Tasks", *Journal of Parallel and Distributed Computing*, Vol. 4, 1987, pp. 342-363

[2] **Hu** T C: "Parallel Sequencing and Assembly Line Problems", *Operations Research*, Vol. 9, Part 6, 1961

[3] **Linton** K N, **Gould** G L, **Terepin** S, and **Purvis** A: "Optimising Massive Parallel Architectures for Real-Time Digital Audio", *89th Audio Engineering Society Convention*, Los Angeles, USA, September 1990

[4] **Linton** K N, **Terepin** S, and **Purvis** A: "Taskforce Scheduling Strategies for Digital Mixing Consoles", *90th Audio Engineering Society Convention*, Paris, France, February 1991

[5] **Gould** G L, **Bowler** I, and **Purvis** A: "Real-Time Multi-Channel Digital Filtering on the Transputer", *IEE CA-DSP*, Hong Kong, September 1989

[6] **Lee** E A: "Programmable DSP Architectures: Part II", *IEEE ASSP Magazine*, Vol. 6, No. 1, January 1989

[7] **Jagadish** N, **Mohan Kamar** J, and **Patnaik** L M: "An Efficient Scheme for Interprocessor Communication using Dual-Ported RAMs", *IEEE Micro Magazine*, Vol. 9, No. 5, October 1989

[8] **Anderson** T E, **Lazowska** E D, and **Levy** H M: "The Performance Implication of Thread Management Alternatives for Shared Memory Multiprocessors", *IEEE Trans. on Computers*, Vol. 38, No 12, December 1989

[9] **Linton** K N: *Real-Time Digital Audio Processing: Parallel Architectures and Resource Allocation Techniques*, Invited Lecture to the British Section of the Audio Engineering Society, IBA, London, Great Britain, July 1991

Appendix G

AES UK DSP Paper Reprint

(See over for a reprint of the invited paper presented by the author at the Audio Engineering Society UK DSP Conference, London, Great Britain, September 1992).

ARCHITECTURAL ISSUES FOR PARALLEL DSP AUDIO SYSTEMS

KEN N LINTON

University of Durham
South Road, Durham, DH1 3LE, UK

The design and implementation of audio systems through analogue technology is a well understood art. With the success of both professional and consumer digital formats, the audio industry is moving quickly towards an entirely digital signal path. In this paper, the principle architectural issues of implementing digital audio systems with general purpose digital signal processors are discussed. Particular attention is given to the design and performance aspects appropriate to professional audio products.

INTRODUCTION

Digital signal processing (DSP) devices are inherently programmable. Instead of using different types of hardware to accomplish different processing functions, DSPs just perform different sets of summations, delays and multiplies grouped together as processing algorithms. With a DSP-based audio system, the limiting factor is the total amount of processing power available rather than the quantity and type of processing hardware provided. The modularity, flexibility, and reliability of DSP makes it attractive for all manner of applications, including those traditionally implemented by analogue technology.

The advent of relatively low cost DSPs has made multi-DSP processing an economic reality in today's digital audio environment. As the audio community becomes accustomed to buying more technology with less money, design engineers face increasingly difficult design problems. In most cases, this requires that the designer opt for a software-based multiprocessor (M-P) system built from freely available off-the-shelf components from a range of manufacturers. This strategy allows a quicker design cycle and a flexible approach to the design of professional audio products.

1 DESIGN CONSTRAINTS

In the design of parallel DSP audio systems, the first three constraints are those inherent in conventional uni-processor designs: processor speed, memory resource and input/output (I/O) bandwidth. Since most applications require more processing power than is provided by a single DSP device, a multi-DSP architecture must be created. Memory capacity is not usually of concern since DSPs can address much more memory than can be accessed in a sample period. As most DSPs offer independent I/O, supporting ADC/DACs and AES-EBU/SPDIF interfaces, multi-DSP M-Ps offer excellent I/O bandwidth which scales with the number of processors.

1.1 Initial Decisions

The two most fundamental decisions which must be taken very early on in the design process are, firstly, how powerful each DSP should be, and, secondly, how many processors should be supported. For a system of required performance P, and an ideal architecture containing n processors, each with an individual processing capability of p, the hyperbolic relationship between n and p, as Figure 1, defines a span of possible architectures satisfying [1]. Therefore, one could use a small number of very powerful (and expensive) processors, or a large number of relatively slow (and cheap) processors. The use of large numbers of DSPs is made attractive by the development of relatively cheap DSP devices suitable for professional audio applications.

The cost advantage of using low-speed technology is balanced by the degradation in efficiency that inevitably occurs as the number of processors in a system increases. Moreover, the complexity of programming a M-P with many DSPs far exceeds the complexity of programming an architecture

consisting of just a few processors. Consequently, although economics might enhance the attractiveness of a system with many low-speed DSPs, the advantages disappear if efficiency is not held high. When the parallelism cannot be tapped effectively, it simply adds to system cost and complexity. This demands that multi-DSP systems are assembled in such a way that the performance of each DSP is effectively harnessed.

1.2 Implementing Parallelism

There are two fundamental ways in which DSPs can be composed to create parallel architectures. Perhaps the simplest way to introduce parallelism is to replicate a component *n* times, as shown in Figure 2a. To exploit this form of parallelism, the digital audio streams processed must be separated in space into individual channels. For this reason this form of parallelism is known as spatial parallelism. A typical example from everyday life is the familiar row of checkout desks in a supermarket.

The other fundamental way of introducing parallelism is to partition the audio processing into a number of steps, as shown in Figure 2b, which when applied sequentially to each sample perform the original processing desired. In other words, the task is partitioned in time, with each step of the application being applied to a separate sample. For this reason this form of parallelism is known as temporal parallelism. The application of temporal parallelism in digital audio produces pipelined structures.

1.3 Processor Pipelining

Pipelining is an implementation technique whereby multiple tasks are simultaneously overlapped in time [2]. In fact, pipelining is the key implementation technique used to make fast DSP devices. In a processor pipeline, as in an assembly line, the work required to complete a processing application is broken into smaller pieces or stages. The stages are then connected one to the next to form a pipe, as Figure 3a. Samples enter at one end, are processed through the stages, and exit at the other end, as detailed in Figure 3b.

In digital audio systems response is perceived as intimate if the system operates with a latency, or accumulated processing delay, of less than one millisecond. To meet the necessary frequency response requirements, each DSP in the audio system must take a new set of input samples every 20.8μs, given the professional sampling rate of 48kHz. This corresponds to a maximum processor pipeline of 48 stages between any input and any output.

1.4 Additional Constraints

Three additional design constraints, which are not usually associated with uni-processor systems, stem from the need for parallel architectures. Firstly, with multiple DSPs it becomes necessary to exchange audio samples and control parameters so some type of interprocessor communications (IPC) network is required. Novel techniques are required to tie together conventional DSPs in large digital audio systems. Secondly, because the processors must work together they are required to synchronize to coordinate such activities as access to shared data. Finally, cost-effective programmable DSP systems necessitate M-P architectures which incorporate sufficient flexibility in their design to support several different audio processing applications.

2 M-P CONFIGURATIONS

Most professional audio applications require a processing rate much higher than can be provided by any single DSP device. For example, a 2-channel third-octave graphic equaliser operating at the professional sample rate (48kHz) requires about three 10.25 MIPS DSPs. A zero chip interface may be implemented by connecting the serial links of two or more DSPs. Unfortunately, the relative slowness of this technique restricts this approach to a limited number of applications. For faster data transfer, the parallel data bus must be used to connect multiple DSPs in tightly coupled M-P architectures.

Conceptually, the simplest way to interconnect multiple DSPs is to provide full connectivity between processors. However, due to the sheer hardware costs involved, designers must forgo this luxury. Shared-memory architectures are probably the most common form of M-P, since they are the natural extension of conventional uni-processor systems. A radically different communication mechanism is message-passing. In message-passing architectures DSPs communicate by sending and receiving messages via dedicated communication channels [1].

2.1 Shared-Memory Design

The simplest way to construct an audio M-P is to connect the DSPs on a shared bus, providing shared common memory to all processors, as shown in Figure 4. In order to isolate the processors from each other so that they can

simultaneously access their local memories, bidirectional bus transceivers are required. The bus is then active only when data is transferred between DSPs, allowing the bus to remain free during processing which does not demand IPC. If the local DSP RAMs were not present, there would be severe contention on the bus, causing arbitration delays to reduce performance.

Unfortunately, in the case of processors such as the DSP56001 there is no provision to force the external memory port to quickly tri-state (isolate) in a deterministic manner. Since the address bus is always driven (except by granting the bus), one DSP's local RAM cannot be directly accessed by another DSP while the first accesses its internal RAM. Additional bus transceivers would be required to allow one DSP to quickly access another DSP's local RAM without shorting its main bus, increasing board real-estate and power consumption [3].

2.2 Arbitration Schemes

Arbitration schemes are necessary to resolve the situation when multiple processors need to access the same piece of data at the same time. Most DSP devices have provision for asynchronous sharing of their external memory port, with the use of bus-request (BR) and bus-grant (BG) signals. M-P architectures which use BR and BG signals to directly share a common bus typically require a hardwired master-slave relationship between the interconnected DSPs. This situation is certainly suitable for many product designs. However, software control is to be preferred, as this provides an equal relationship between the processors.

A second disadvantage of this technique is that the process of granting the bus is too time consuming for most real-time audio applications, as there is little time in one 48kHz sample period for arbitration of the bus. In synchronous designs, the most serious problem with this technique is the indeterminate timing of the bus-granting process. For example, the BR to BG delay for a 20.5 MHz (10.25 MIPS) DSP56001 is specified to take between 73 to 240ns for zero-wait states [4]. Since this delay may be more than one instruction cycle, synchronous DSPs cannot be kept in step when sharing a common bus via the BR/BG arbitration technique.

2.3 Summary

An *n*-processor system requires a bus whose bandwidth is of the order of *n*-times that of a uni-processor bus. Consequently, the major shortcoming of shared-memory M-Ps is self-evident: the data transfer capacity between processors is determined by the bandwidth of the bus and is therefore constant. As the number of DSPs grows, the shared bus between DSP modules becomes a serious bottle-neck to IPC. This factor limits the number of processors that can be usefully incorporated into such a system, and hence fixes an upper limit on extensibility.

Typically, bus-oriented systems can support 10 DSPs efficiently and possibly stretch to 20 or 30. Beyond this range, bus contention leads to degraded performance to the extent that such systems are unlikely to support large audio systems unless a technological breakthrough provides a very high bandwidth bus at relatively low cost. A possibility for the future is to implement large digital audio systems with optic fibre links. However, the low cost and simplicity of bus-structured M-Ps has proved to be advantageous for a number of small-scale parallel DSP systems, yielding high performance/cost ratios without any pretensions of scalability.

3 SYNCHRONIZATION MECHANISMS

Multiple DSPs can be synchronized at the instruction cycle level through the use of a common clock. However, *clock skew* can be a considerable problem especially at high clock rates in large M-P architectures. Alternatively, synchronization mechanisms are used to ensure that shared data is always valid in asynchronous designs. This is one of the most difficult and error-prone types of DSP programming that exists, because it involves the understanding of the potential simultaneous actions of multiple DSPs.

3.1 Semaphores

The dictionary defines semaphore as "signalling by flags". Semaphore flags are used like locks to shared resources, such as hard disk buffers and interprocessor message queues, and so permit only one DSP at a time to gain access. Two different operations are performed on the semaphore: the request operation which attempts to gain access and the release operation which signals the termination of the access. These operations are used to guarantee mutual exclusion, meaning that only one processor is able to access shared data at any given time, locking out all other processors. This occurs from the time a request is granted until the time that the semaphore is released.

At one extreme a whole shared memory can be controlled by a single semaphore, whereas at the

other extreme every data item could be individually locked. The former precludes any parallel operations on the shared data, of course, while in the latter case the overhead of performing the locking operations would be prohibitive. Typically, this overhead is distributed across several data values, with one semaphore locking the corresponding data structure.

3.2 Test-and-Set

One of the most basic synchronizing techniques is the *test-and-set* primitive [5]. Firstly, the DSP desiring to obtain exclusive access to the shared data, reads the corresponding semaphore flag. It then tests this prior semaphore value to determine if it was successful (i.e. a *0*) and, finally, writes a *1* to the semaphore location. As a result, the test-and-set operation forces the semaphore flag to be set to *1*, whether or not it was set beforehand. If the semaphore was not already set, then this DSP can access the shared resource. All other DSPs are blocked because the semaphore is now set. As a result, only one DSP at a time has permission via the semaphore.

To ensure that it can be used successfully, the test-and-set operation must be uninterruptible. That is, once it is initiated and the read access is completed, no other access can be made to the semaphore until the semaphore is rewritten during the second step of the test-and-set. If an intervening access were permitted, synchronization could fail. This is because one DSP may test the semaphore and, before it can set it, another DSP might test it. In this case, both processors believe they have the semaphore. Most DSPs support an indivisible READ/MODIFY/WRITE instruction, which ensures that the address bus remains active throughout the entire operation.

If several DSPs execute a test-and-set on a semaphore concurrently, the requests will be serialised and executed one by one due to the indivisible nature of the READ/MODIFY/WRITE operation. Given serial execution, no more than one DSP from a set of concurrent requesters can observe a *0* semaphore value and thereby gain access to the shared resource. When it has finished with the resource the processor must release the semaphore. To achieve this, the owner DSP does no more than clear the flag by writing a *0* into the semaphore location. Note, it is not necessary to perform a READ/MODIFY/WRITE instruction to unlock the semaphore.

3.3 Spin-Lock

The test-and-set is only half of the synchronisation mechanism: the other is the action taken depending on whether the semaphore has been granted or not. A question which immediately arises is what to do with a DSP which is waiting to access a shared resource. One mechanism which gives fast entry to a shared resource is the *spin-lock*. Thus, if the lock has been granted, the processor continues on to use the shared resource. If not, then a spin-lock occurs, and the DSP must *spin* backwards and re-execute the test-and-set, repeating the process until the lock is granted, as Figure 5. There are two major drawbacks to using spin-locks [6].

Firstly, a DSP which is polling a semaphore is not doing any useful work. Secondly, it is consuming memory cycles in the bank containing the semaphore value and, if several processors are waiting at the same semaphore, memory bandwidth is rapidly consumed. This contention causes additional cycles of delay while a DSP is attempting to release a lock, which magnifies the effect of the bottle-neck at the semaphore. As a result, spin-locks waste valuable MIPS by dedicating potentially useful DSP instruction cycles to the effort of repeatedly testing semaphores.

4 PERFORMANCE ISSUES

Characterising the performance of DSP engines is an important exercise if designs are to be compared, particularly with respect to scalability. A realistic assessment must take into account both the architecture and the applications for which it is intended. Characterising a particular audio application includes considering processing, memory, I/O, and IPC resources required. Similarly, proposed architectures should be considered in terms of the hardware resources available to the range of applications being considered [7].

4.1 MIPS

A number of popular measures have been adopted in the quest for a standard measure of M-P performance. Small, key pieces extracted from real audio applications, such as biquad filter algorithms, can provide useful processing kernels for performance evaluation. One alternative to time is the parameter MIPS million instructions per second. For example, Table 1 details a hypothetical processing budget for a comprehensive digital mixing console [8]. Since

MIPS is the rate of operations per unit time, faster architectures have a higher MIPS rating.

Although MIPS is easy to understand, the problems with using MIPS as a measure for comparison are two-fold. Firstly, MIPS is dependent on instruction set, making it difficult to compare M-P systems based on DSPs with different instruction sets. Secondly, MIPS can vary widely between different processing applications on the same audio system. As a result, the only consistent and reliable measure of the performance of M-P systems for a particular audio product is the actual execution time of the required processing algorithms.

4.2 Efficiency

In M-P systems *efficiency* is defined in terms of the ratio of time spent executing useful DSP routines compared to that consumed by IPC, synchronisation and other overheads. When a M-P is operating at peak performance, all processors are engaged in useful work. Moreover, no processor is executing an instruction that would not be executed if the same application was executing on a single DSP. In this state, all n processors are contributing to effective performance, and the MIPS is increased by a factor of n — linear *speed-up*. Unfortunately, this is rarely achieved due to the overheads incurred as a direct result of the use of parallelism.

In one case, designers started out with the knowledge that in previous M-Ps each additional DSP contributed only 0.8 times the actual performance of each processor already in the system [1]. Hence, points on the speed-up curve for such a system would typically be 1, 1.8, 2.5, ..., as shown in the 'actual' curve of Figure 6. Such diminishing returns make each successive processor less and less cost effective. Efficiency is clearly a major concern in the design of digital audio products based on M-P architectures: a design that uses $2n$ DSPs inefficiently cannot compete on a cost basis with a design that uses n identical processors twice as efficiently.

4.3 Load Balancing

There are various options when executing an audio processing application consisting of m processing algorithms on an n processor multi-DSP system. If all the work is allocated to one DSP and the remaining processors ignored, the IPC overhead is minimised but the system will perform no better than a single DSP. Conversely, if the work is split between the n processors, considerable overhead is incurred. Here, performance could be up to n-times that of a single DSP device, as shown in the ideal curve of Figure 6. Experiments with M-P architectures, however, have demonstrated behaviour in operational systems is more like the 'actual' curve.

In practice, the speed-up increases significantly only for the first few additional processors. At some point the speed-up may actually begin to decrease with each additional DSP this can be as few as 10 depending on the architecture. This decrease in expected speed-up is due to excessive IPC and synchronization overheads, incurred when processing algorithms resident on different DSPs must communicate with each other. The resultant degradation in performance for incremental increase in hardware resources is known as the *saturation effect*. Consequently, the efficiency of a system depends critically on how the workload is balanced between the processors.

5 FURTHER CONSIDERATIONS

There are many ways to express the performance of a DSP system. For example, the metric of performance commonly assumed is MIPS, but cost, reliability and extensibility are just as important.

5.1 Cost

An understanding of cost is essential for designers to be able to make intelligent decisions about whether or not a particular processor, I/O or interconnection component should be incorporated in design. The cost of DSP components is changing so fast that good designers are basing design decisions not on today's costs, but on projected costs at the time the product is shipped. Although they are far from representing what the customer must pay, cost of components confine a designer's desires. In digital audio systems, component costs typically make up 15-33% of the final list price.

Many engineers are surprised to find that most companies spend only 8-15% of their income on R&D. This information suggests that a company uniformly applies fixed-overhead percentages to turn cost into price. Another point of view is that R&D should be considered an investment, and so an investment of 8% to 15% of income means that every pound spent on R&D must generate £7 to £13 in sales. This alternative point of view then suggests different gross margins for each product depending on the number sold and the size of the investment [2].

Large expensive DSP systems generally cost more to develop — a product costing 10 times as much to manufacture may cost many times as much to develop. Since large audio systems generally do not sell as well as small ones, the gross margin must be greater on big machines for the company to maintain a profitable return on R&D investment. This places large DSP systems in double jeopardy, since there are fewer sold and they require larger R&D costs.

Few customers can afford the initial cost of a large multi-DSP system. However, many can afford to grow a large system in increments as they discover the need for more processing power and as their budgets permit. There is also a great risk in designing a multi-DSP architecture welded to a particular audio processing application. By incorporating sufficient flexibility into a design to cover several applications, the R&D investment can be amortised over a range of end-products constructed from affordable DSP modules.

5.2 Reliability

The reliability of M-P systems is often considered to be an issue which is of secondary importance to the task of designing for maximum throughput. It is a commonly held belief that M-P architectures are inherently tolerant of faults since the replication of processing elements leads to the natural availability of spares. Designing machines which are fault-tolerant (and hence reliable) involves a great deal more that simply providing spares, however. Each fault must be located before any hardware re-configuration can be performed.

As in any electronic system, the most unreliable elements in a multi-DSP architecture are the electrical connections between physically distinct component parts. Since interconnection networks normally contain large numbers of wires and connectors, intermittent faults caused by momentary interruptions are not uncommon. Therefore the protocol for data-movement through the network should be robust and capable of detecting errors. More permanent faults can result in one or more DSPs being unable to communicate with the rest of the M-P.

Since faults cannot be avoided, the only alternative is to design multi-DSP systems for maximum resilience and fault-tolerance. Designing for maximum resilience means discovering which components are least reliable, and either minimising their use or making them more reliable. Designing for fault-tolerance means two things: firstly, designing systems with the ability to detect the occurrence of an error, and secondly imbuing those systems with the ability to correct and recover from an error.

5.3 Extensibility

It is important to leave room for future expansion of an audio M-P as one runs out of processing power. For example, a shared bus would typically experience heavy traffic with only 10 DSP modules. Many professional audio applications require much more processing power than this. Therefore, a fundamental consideration is the effect on the system of altering the degree of replication, usually upward. When assessing the extensibility of a particular M-P machine, a designer must consider several aspects of the design.

Firstly, as the number of DSP modules is increased the total cost of the system must also increase. It is obviously desirable to minimise this increase in cost to that of linear growth. Secondly, the designer must assess how the basic architecture parameters, such as IPC capacity, vary with replication. A further important practical consideration, especially for very large scale M-Ps, is the space occupied by the system as a whole and, more particularly, by the interprocessor wiring.

If the degree of replication is to be very large then a careful analysis of the rate of growth in hardware complexity must be performed. However, the ability to extend a particular multi-DSP architecture is clearly only important within a given range, since commercial systems have a finite lifetime and end-users have finite budgets. In the longer term, as configurations become larger, the effects of scaling will become more important. The extensibility of M-P architectures is still a subject which is of much research interest.

6 MESSAGE-PASSING DESIGN

Various interconnection schemes have been suggested for message-passing between conventional DSP processors. Here, some of the approaches which utilise widely available integrated devices are considered.

6.1 First-In First-Out Buffers

One technique is to employ first-in first-out (FIFO) buffers. It is important to monitor the boundary conditions (FULL or EMPTY) of these devices, as failure to act on their occurrence will result in data overflow or underflow. The current FIFO generation signals the EMPTY, HALF-FULL and FULL conditions by asserting corresponding

external pins. The EMPTY and FULL flags are also fed back internally and inhibit further read and write until the FIFO is no longer empty or full.

The increasing use of high-speed applications has created a demand for a faster and smarter generation of FIFOs. Flagged FIFOs offer the basic features discussed above while providing two additional programmable flags: ALMOST-EMPTY and ALMOST-FULL. These flags can be used as early warning flags in critical real-time applications such as pipeline DSP for digital audio applications. Current devices provide for unlimited expansion while maintaining a 50ns fall-through time [9].

6.2 Dual-Ported RAM

Due to their high bandwidth and message passing flexibility, dual-ported RAMs (DPRs) can be used to link multiple high-performance DSPs. Several manufacturers produce DPRs in many configurations, all of which operate at static RAM speeds (50-150ns) and have two independent external memory ports [9]. This allows two processors to share the same block of physical memory in their respective address spaces. The two DSPs can access data in two memory locations simultaneously and asynchronously.

This approach clearly out-performs a discrete parts design where two processors must synchronize through arbitration for access to a bus which is used to access one location at a time in a standard single-port RAM. In this way, the integrated DPR approach removes synchronization requirements at the memory's bus access level. Nevertheless, synchronization must be performed at other levels to ensure data integrity and thus proper system operation.

6.2.1 Busy Logic

A problem can occur with DPR memories when both ports attempt to access the same address at the same time. There are two significant cases: when one port is trying to read the same data that the other port is writing and, when both ports attempt to write to the same word at the same time. If one port is reading while the other port is writing, the data on the read side will be changing during the read and a read error can result. If both ports attempt to write at the same time, the memory cell is being driven by both sides and the result can be a random combination of both data words.

Busy logic solves this problem by detecting when both sides are using the same location at the same time. The BUSY output pins are suitable for attachment to the WAIT inputs of most DSPs. This approach is very straightforward and flexible, and has the benefit that a processor cannot be locked out of the RAM longer than the access period of the other processor. Note that although one or the other processor may have to wait occasionally, the throughput loss is minimal since the probability of both DSPs using the same location at the same time is very small.

6.2.2 Semaphore Support

Software constraints may require mutual exclusion at the data structure level rather than at the memory location level. Instead of comparing addresses on every cycle and occasionally asserting BUSY, other DPR devices employ additional circuitry to support semaphores, thus ensuring that only one side has permission to use a block of memory. The conventional test-and-set semaphore operation requires that the two memory accesses are indivisible: some DPR devices employ a twist by using set-and-test.

The *set* corresponds to a request and the *test* checks to see if the request was granted. The indivisible double access requirement is avoided because, as soon as the request is made by one DSP, the grant is blocked on the other side. As a result, semaphores can be supported for DSPs which do not provide an indivisible READ/MODIFY/WRITE instruction. Since there is no hardware relationship between semaphores and DPR locations, the block sizes and locations, and semaphore association can be defined in software. This offers the system designer considerable flexibility.

6.2.3 DPR Memory Expansion

DPR chips can be combined to form large dual-port memories. Expansion in memory depth with DPRs is similar to that used for conventional RAMs, where the top address bits enable individual memory blocks. DPRs can also be expanded in width. However, in this case, we have a subtle problem. If the addresses for both ports arrive simultaneously at both RAMs, it is possible for both sides to simultaneously assert BUSY, causing both DSPs to wait indefinitely for their ports to become free. The solution to this BUSY lock-up problem is to use the arbitration logic in only one DPR.

An example of expanded DPRs used for DSP-to-DSP communication is shown in Figure 7. Here, two DSP56001s communicate using three 8-bit wide DPR chips in a MASTER/SLAVE

configuration to provide a word-wide dual-ported memory. If the MASTER device activates BUSY, the dedicated DPR SLAVE chips, which incorporate a BUSY input pin, will internally disable their write enables. Note that the BUSY signals from the MASTER are also fed to the WAIT pins of the DSPs, forcing them to stall when their side of the DPR block is BUSY.

6.2.4 DPR and FIFOs Compared

Each 2K x 24-bit DPR block provided for inter-DSP communication requires three 48-pin ICs, plus components for memory decoding. A bi-directional 2K x 24-bit FIFO channel requires a total of six 28-pin ICs. With each doubling in size of a DPR block, 2 address pins must be added to the chip package. Larger FIFOs never need to change their package size, since the cells within the FIFO are not directly addressed. Thus FIFO technology scales better than comparable DPR technology. However, DPRs provide considerably more communications flexibility.

6.3 Novel Devices

Serving as a complex four bus interconnection network, quad-ported RAMs (QPRs) greatly simplify the task of creating generalised M-P systems for audio processing with conventional DSPs. Although typically three times as expensive as comparable DPR devices, QPRs have a considerable interconnectivity advantage over DPRs. For example, in order to fully interconnect four DSPs requires 1 bank of QPR as opposed to 6 banks of DPR, as shown in Figure 8.

Implementing multi-DSP systems with processors not specifically designed for parallel processing can cause IPC to quickly saturate device I/O and adversely affect processing efficiency. DSP manufacturers made the first step in addressing the need for parallel processing by providing designers with two external memory interfaces. Novel devices such as the Texas Instruments TMS320C40 go several steps further by incorporating on-chip hardware to support dedicated communications ports [10].

Each of the six bidirectional communication ports provides glueless processor-to-processor communication at a transfer rate of 20 Mbytes/s. Without these ports, the corresponding 120 Mbytes/s of processor throughput would have to be squeezed into one or both of the external memory interfaces, as Figure 9. With the communication ports, IPC bandwidth is plentiful and scales with the number of processors. In this way the TMS320C40 can support of a wide variety of complex M-P configurations to meet the needs of real-time audio processing applications.

7 CONCLUSIONS

Parallel DSP systems are an economic reality in today's digital audio environment, due to the wide availability of low cost DSPs devices. The fundamental advantage of such multi-DSP designs is their generality, since software algorithms govern the manipulation of samples and so dictate the audio processing performed. Although it is easy to replicate the processing resources required for multi-DSP audio systems, such replication can introduce serious problems with respect to communications bandwidth.

IPC networks are of fundamental importance in parallel DSP systems as they directly effect system throughput and have a bearing on modularity, extensibility and overall system performance. In order to implement digital audio systems through such replication, interconnection networks are required which provide sufficient IPC bandwidth for the range of processing applications being planned.

Utilising design techniques from conventional uni-processor systems allows small M-P systems to communicate through a common bus and shared memory. However, the inherent advantages of low cost and simplicity must be balanced against the significant problems encountered when this approach is used for large-scale audio systems.

Integrated interconnection components, such as FIFOs and multi-ported RAMs, provide simple static RAM interfaces which connect transparently to almost any existing DSP design. As a result, they enable conventional DSP modules designed for uni-processor or small shared-bus systems to be directly employed in a variety of scalable M-P architectures. In particular, the multi-ported RAM scheme presents a cost effective method for message-passing between DSP modules.

Knowledge accumulated through the research of the past 25 years into parallel architectures and programming is making considerable impact on the design of products for the digital audio environment. For example, software techniques, such as semaphore flags, guarantee the integrity of audio samples shared by multiple DSPs. Task allocation strategies minimise the effect of IPC saturation and maximise the processing efficiency of multi-DSP systems [7].

Novel devices, such as the Texas Instruments TMS320C40, are beginning to emerge which integrate multiple IPC ports directly on-chip with high-speed DSP cores. Although currently expensive, their cost compares favourably with the pricing of conventional DSPs, such as the Motorola DSP56001, when they were first released.

The design of multi-DSP audio products is extremely challenging: one cannot simply lash together 100 DSP devices and expect to obtain 100-times the performance of a sole DSP. Although the amount of raw processing power is important for audio processing applications, factors such as efficiency and cost are of considerable consequence in the marketplace. Understanding of the architectural approaches available and the implications of design decisions enables the construction of well-engineered parallel DSP audio systems.

REFERENCES

[1] Ibbett, R and Topham, N P: "Architectures of High Performance Computers Volume II", Macmillan Education Ltd., 1989

[2] Hennessey, J L and Patterson, D A: "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Inc., 1990

[3] Snell, J M: "Multiprocessor DSP Architectures and Implications for Software", Proceedings of the AES 7th International Conference, Toronto, Canada, May 1989

[4] DSP56000/DSP56001 Digital Signal Processor User's Guide, Revision 2, Motorola Inc., 1990

[5] Stone, H S: "High-Performance Computer Architecture", Addison-Wesley, 1987

[6] Linton K N, Gould G L, Terepin S, and Purvis A: "Real-Time Multi-Channel Digital Audio Processing: Scalable Parallel Architectures and Taskforce Scheduling Strategies", 1991 International Conference on Acoustics Speech and Signal Processing, Toronto, Canada, May 1991

[7] Linton, K. N., Gould, G. L., Terepin S, and Purvis A: "Optimising Massive Parallel Architectures for Real-Time Digital Audio", 89th Audio Engineering Society Convention, Los Angeles, USA, September 1990

[8] Eastty, P: "Digital Audio Processing on a Grand Scale", 81st AES Convention, Los Angeles, USA, November 1986

[9] Specialised Memories Databook, Integrated Device Technology Inc., 1991

[10] TMS320C40 User's Guide, Revision A, Texas Instruments Inc., 1991

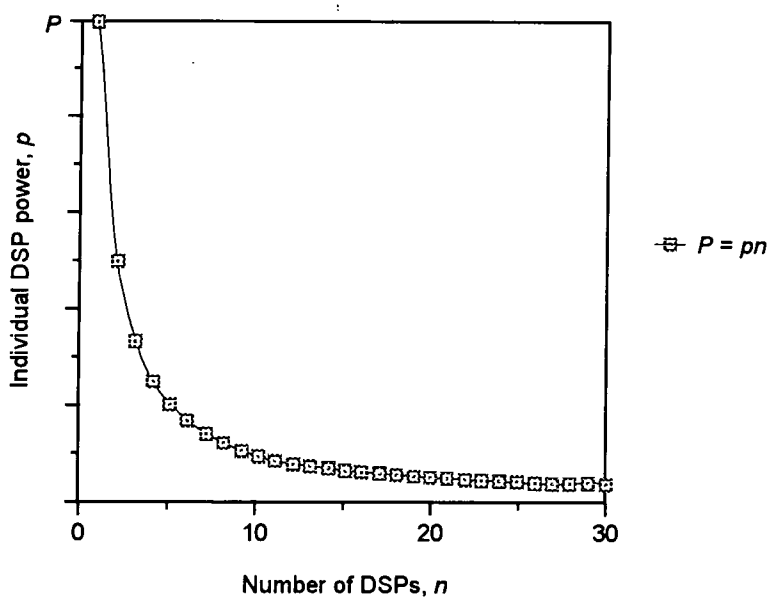

Figure 1: Individual DSP Power against number of DSPs required

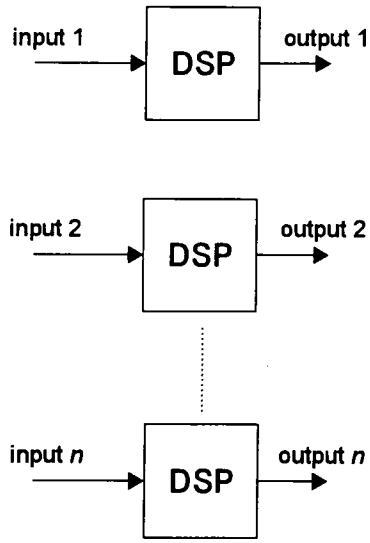

Figure 2a: Exploiting spatial parallelism

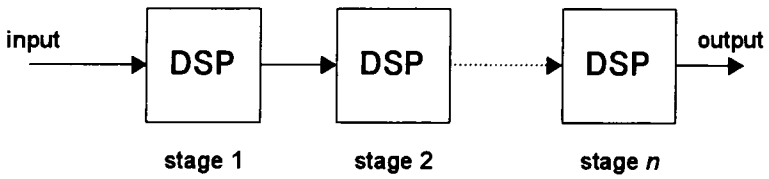

Figure 2b: Exploiting temporal parallelism

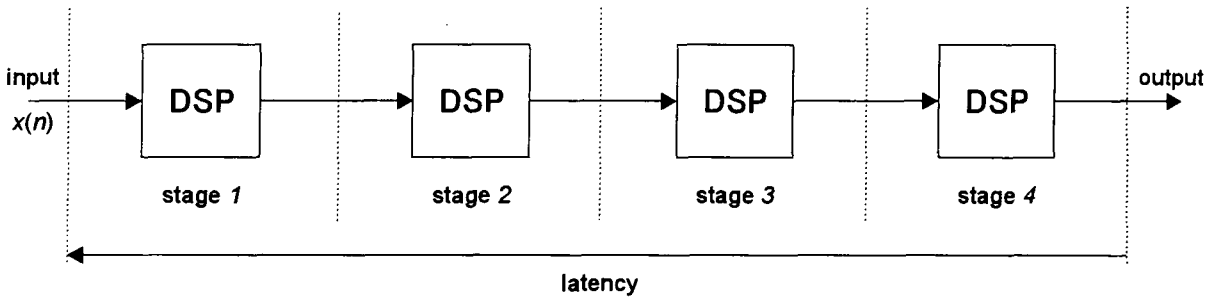

Figure 3a: Processor pipelining with DSPs: four-stage linear pipeline

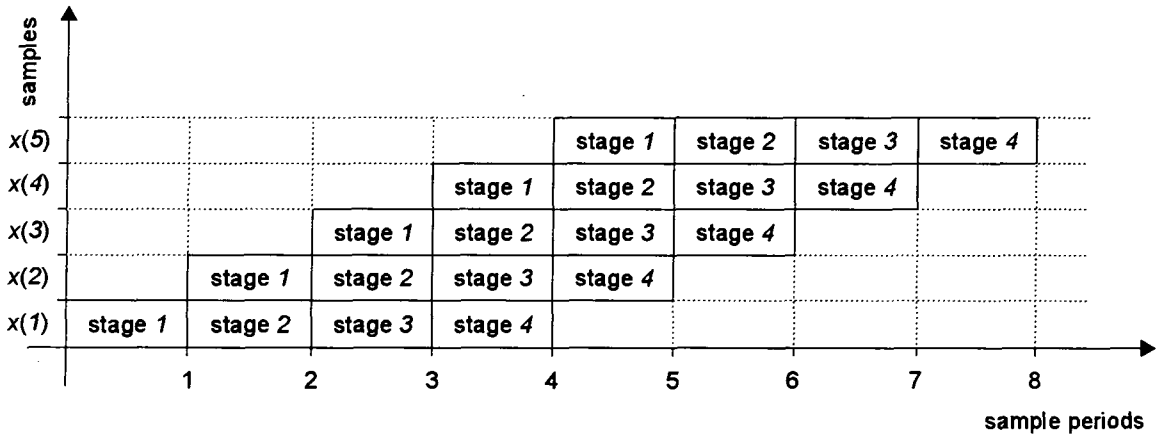

Figure 3b: Detail of overlapped processing in 4-stage linear pipeline

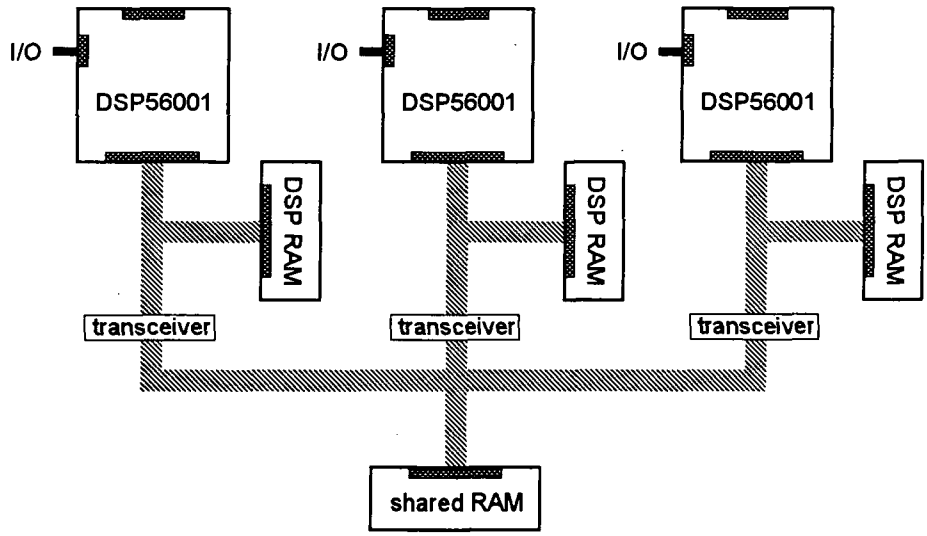

Figure 4: Example shared-memory multiprocessor architecture

Digital Mixing Console Design	Processing Budget
No. of functions for each channel strip	80 discrete and 30 continuous
No. of functions for complete 64 channel console	7000 functions
No. of functions at professional sample rate of 48 kHz	350 million functions/s
Processing requirement, at 3 instructions per function	> 1000 MIPS
Size of M-P architecture, using 10 MIPS DSPs	> 100 processors

Table 1: Processing budget for a hypothetical digital mixing console

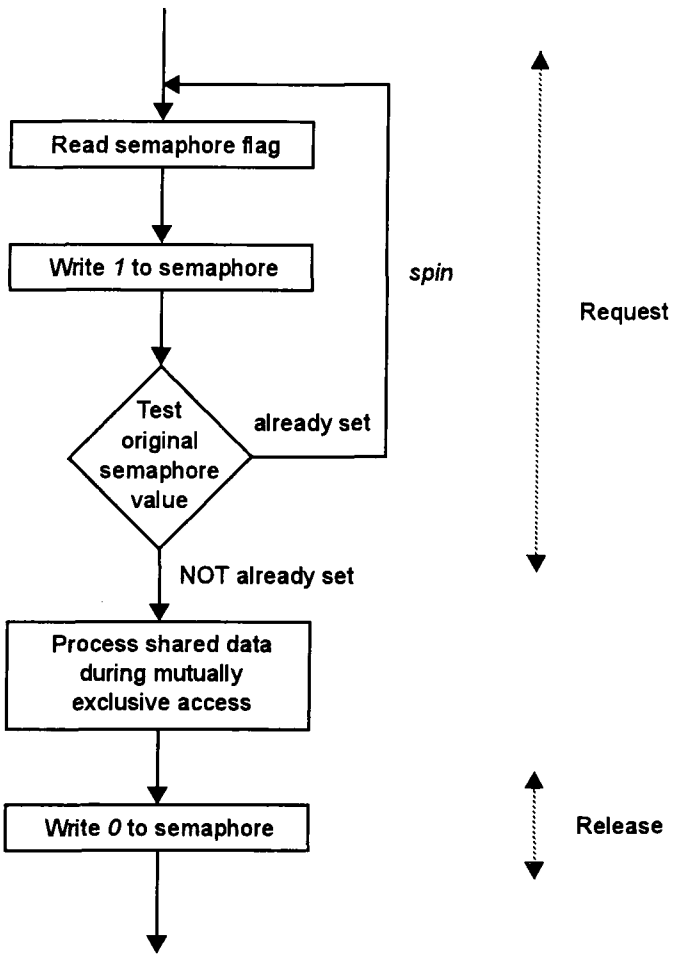

Figure 5: Flow chart of *test-and-set* and *spin-lock* semaphore operations

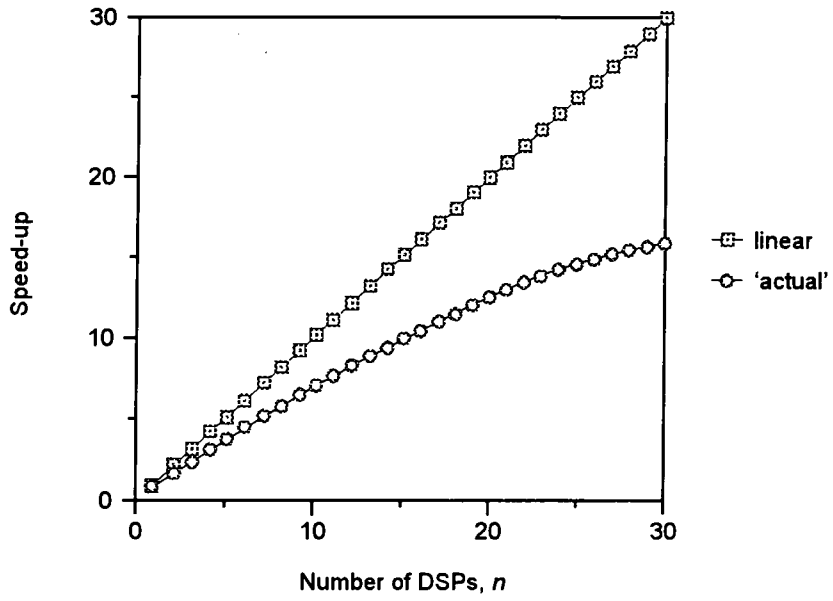

Figure 6: Speed-up curves for multi-DSP architectures

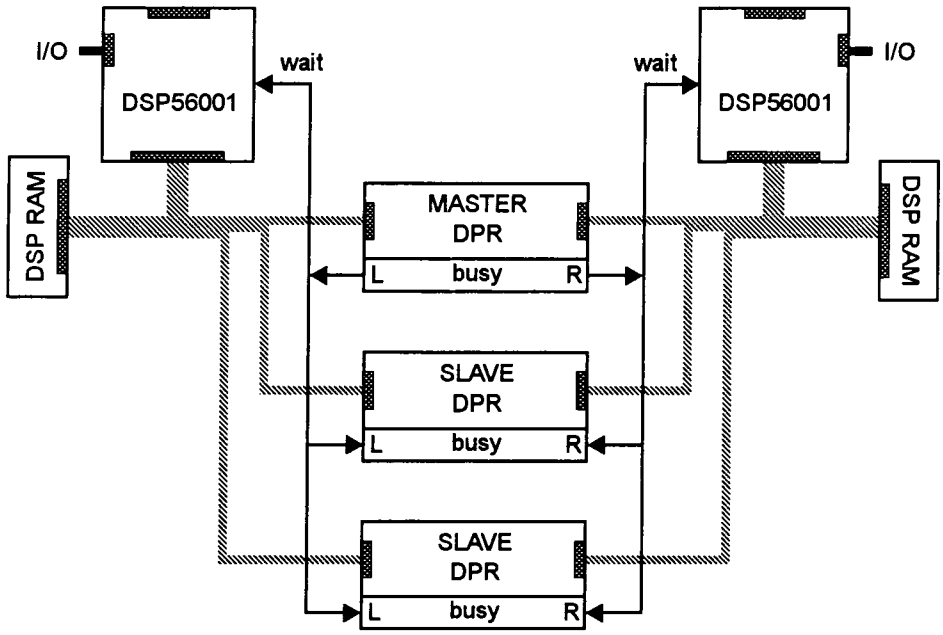

Figure 7: DSP-to-DSP communications using width-expanded dual-ported RAMs

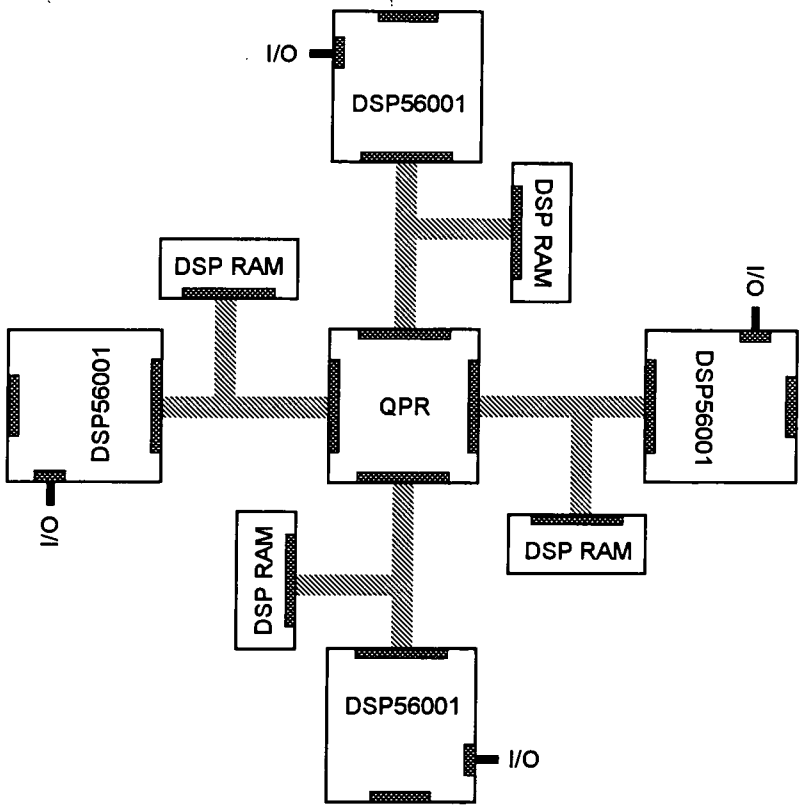

Figure 8: Using quad-ported RAM to fully interconnect conventional DSP processors

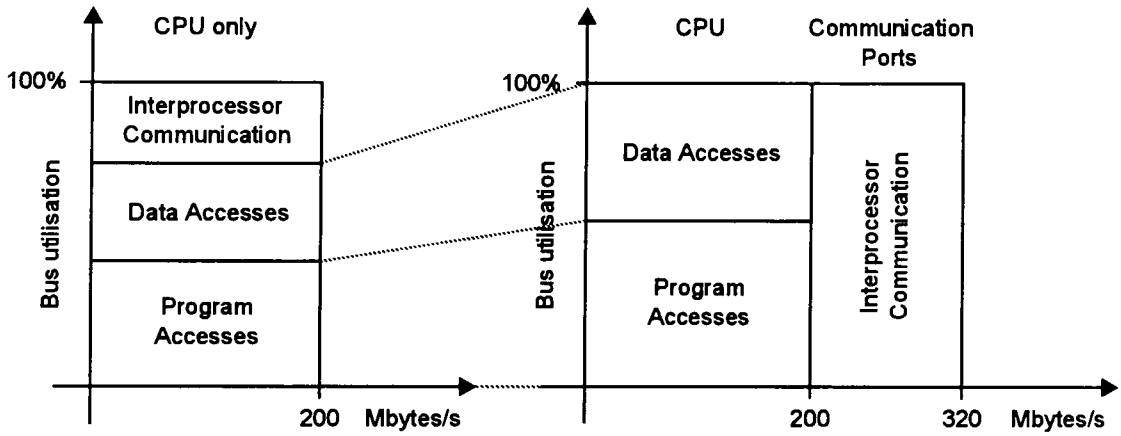

Figure 9: Advantages of using dedicated communications ports on TMS320C40

Appendix H

Glossary of Abbreviations

A glossary of abbreviations used in this thesis is given below. Abbreviations which appear in only one section may not be listed here, but are always defined when they are first used.

A-D	: analogue-to-digital
ADC	: analogue to digital conversion (or converter, depending on context)
ADT	: automatic double-tracking
AES	: Audio Engineering Society
ALL	: all task-processor pairs/heuristics — DYN/HRT control code used in *DCS*
ALU	: arithmetic logic unit
ANSI	: American National Standards Institute
AOT	: adjusted on test
ASIC	: application specific integrated circuit
ASP	: audio signal processing
AST	: A* dynamic programming
BG	: bus grant
BR	: bus request
CD	: compact disc
CO	: combinatorial optimisation
CMOS	: complementary metal oxide semiconductor
CPM	: critical-path method
CPU	: central processing unit

D-A : digital-to-analogue

DAC : digital-to-analogue conversion (or converter, depending on context)

DLL : dynamic-link library

DMA : direct memory access

DMC : digital mixing console

DPR : dual-ported RAM

DSP : digital signal processing (or processor, depending on context)

DYN : dynamic-level list scheduling

EBU : European Broadcasting Union

EIAJ : (Japanese Electronics Manufacturers' Association).

EQ : equalisation

EMI : external memory interface

EVB : evaluation board

FET : field effect transistor

FFT : fast Fourier transform

FIFO : first-in first-out

FIR : finite impulse response

FM : frequency modulation

FWE : fully-overlapped with execution

GA : genetic algorithm

G-P : general-purpose

H-F : high-frequency

HLE : highest level first, estimated times

HLN : highest level first, no estimated times

HMF : high mid-frequency

H-P : high-pass

HRTF : head-related transfer-function

I/O : input/output

IEEE : Institute of Electrical and Electronic Engineers

IID : inter-aural intensity difference

IIR : infinite impulse response

IN : interconnection network

IPC : inter-processor communication

ISO : International Standards Organisation

ISPW : IRCAM Signal Processing Workstation

ITD : inter-aural time delay

LC : link connectivity

LCE : lowest co-level first, estimated times

LCN : lowest co-level first, no estimated times

L-P : low-pass

LCD : liquid crystal display

LED : light emitting diode

L-F : low-frequency

LFO : low-frequency oscillator

LGDF : large-grain data-flow

LMF : low mid-frequency

μP : micro-processor

MAC : multiply accumulate

MADI : multi-channel audio digital interface

MFLOPS : million floating-point operations per second

MIA : minimum independent assignment

MIDI : musical instrument digital interface

MIMD : multiple-instruction multiple-data

MIPS : million instructions per second

MISD : multiple-instruction single-data

MMI : man-machine interface

M-P : multi-processor

MPC : minimum processor cost

MRC : maximum remaining critical-path

MTBF : mean time between failures

NC : node connectivity

NML : non-overlapped, multiple links

NMOS : n-type metal oxide semiconductor

NON : none — DYN/HRT control code used in *DCS*

NP : non-deterministic polynomial-time

NSL : non-overlapped, single link

P : polynomial-time

PAM : pulse-amplitude modulation

PC : program counter

PCM : pulse-code modulation

PE : processing element

PFL : pre-fader listen

PPM : peak programme meter

PRO : processor with lowest index

PWM : pulse-width modulation

QPR : quad-ported RAM

R&D : research and development

RAM : random access memory

RISC : reduced instruction set computer

RND : random static labelling

ROM : read only memory

RSS : Roland Surround Sound

SA : simulated annealing

SCFET : smallest co-level first, estimated times

SCFNET : smallest co-level first, no estimated times

SCI : serial communications interface

SIMD : single-instruction multiple-data

SISD : single-instruction single-data

SMPTE : Society of Motion Picture and Television Engineers

SNR : signal-to-noise ratio

SP-DIF : Sony/Phillips Digital Interface Format

SPW : Signal Processing Worksystem

SQNR : signal-to-quantisation-noise ratio

SSI : synchronous serial interface

SU : super-unitary

TDA : total dynamic automation

TDM : time-division multiplexing

TI : Texas Instruments

TLB : translation lookaside buffer

TRC : Torus Routing Chip

TSK : task with lowest static level

TSP : travelling salesman problem

VCA : voltage-controlled amplifier

VDU : visual display unit

VHDL : VLSI hardware description language

VLSI : very large-scale integration

VU : volume unit

XLL : Excel add-in DLL

Appendix I

Bibliography

While not explicitly referenced within the main text, several books contain subject matter of particular relevance to the research reported in this thesis. Titles in this category include:

[Bokhari, 1987] Bokhari S: *Assignment Problems in Parallel and Distributed Computing*, Kluwer Academic, 1987, ISBN 0-89838-240-8.

[Bratko, 1989] Bratko P: *Prolog Programming for Artificial Intelligence*, second edition, Addison-Wesley, 1989, ISBN 0-13-114375-3.

[Burnham, 1989] Burnham J and Hall S: *Prolog Programming and Applications*, Academic Press, ISBN 0-12-174265-1.

[Coffman, 1976] Coffman E G (ed.): *Computer and Job-Shop Scheduling Theory*, Wiley & Sons, 1976, ISBN 0-471-16319-8.

[Conway, 1967] Conway R W, Maxwell W L and Miller L W: *Theory of Scheduling*, Addison-Wesley, 1967, ISBN 0-876-26410-0.

[Ford, 1962] Ford L R and Fulkerson D R: *Flows in Networks*, Rand Corporation Research Studies, Princeton University Press, 1962.

[Gelenbe, 1989] Gelenbe E: *Multiprocessor Performance*, Wiley & Sons, 1989, ISBN 0-471-92392-3.

[Goldberg, 1989] Goldberg D E: *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 1989, ISBN 0-201-15767-5.

[Ibaraki, 1988] Ibaraki T and Katoh N: *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, 1988, ISBN 0-262-09027-9.

[Kernighan, 1988] Kernigham B W and Ritchie D M: *The C Programming Language*, Prentice-Hall, 1988, ISBN 0-13-119371-6.

[Oppenheim, 1989] Oppenheim A V and Schafer R W: *Discrete-Time Signal Processing*, Prentice-Hall, 1989, ISBN 0-13-216771-9.

[Pearl, 1985] Pearl J: *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, 1985, ISBN 0-201-05594-5.

[Sarkar, 1989] Sarkar V: *Partitioning and Scheduling Parallel Programs for Multiprocessors*, Pitman, 1989, ISBN 0-273-08802-5.

[Sedgewick, 1988] Sedgewick R: *Algorithms*, Addison-Wesley, 1988, ISBN 0-201-06673-4.

[Watkinson, 1988] Watkinson J: *The Art of Digital Audio*, Focal Press, 1988, ISBN 0-240-51270-7.

Colophon

This thesis was written, illustrated and published using Frame Technology 's *FrameMaker* on NeXT and IBM-PC compatible workstations. Analytical results from both *TPG* and *DCS* deliverables were analysed and charted using Microsoft *Excel.* Additional graphics were generated using *Mathematica*, from Wolfram Research, running under *NeXTstep.*

The *TPG* `C` source included in Appendix C was developed using Microsoft's *Visual C++* IDE and built as an XLL dynamic-link library for *Excel.* Borland's *Turbo Prolog* compiler was used to develop the *DCS* `Prolog` application presented in Appendix D. Version control for both projects was managed with *SourceSafe* from One Tree Software.

UNIV DUNDEE