

Durham E-Theses

Applications of microprocessors in digital high frequency radio communications

D. R. Isaac

How to cite:

Isaac, D. R. (1981) Applications of microprocessors in digital high frequency radio communications. 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/7532/> 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.

**APPLICATIONS OF MICROPROCESSORS
IN
DIGITAL HIGH FREQUENCY RADIO
COMMUNICATIONS**

by

D.R. Isaac, B.Sc.

A thesis submitted for the degree of Doctor of Philosophy
in the University of Durham, 1981.

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.



Applications of Microprocessors in Digital High Frequency Radio Communications

D R Isaac

Abstract

This thesis describes the application of VLSI devices to channel evaluation and communication techniques over ionospheric radio paths. Digital signal processing techniques using microprocessors and charge coupled devices are described in detail. A novel method for observing interference and fading patterns on HF channels is described. Error control coding schemes and digital modulation techniques are combined in a design for an adaptive modem for use over HF radio links. Results of narrow-band interference measurements, error patterns and coding performance are presented.

Acknowledgements

Firstly, I wish to thank Professor G.G. Roberts for the opportunity of studying in the Applied Physics and Electronics Department in the University of Durham, both as an undergraduate, and as a postgraduate student.

I am particularly grateful to Dr C.T. Spracklen for his help in securing equipment, and for his consistent guidance and advice during the project. Special thanks are due to my girlfriend, Leena, who has provided never-ending encouragement and support throughout the work.

Thanks are also due to Professor E.D.R. Shearman, of the University of Birmingham, for the loan of equipment, and to the Science Research Council for providing a three year studentship.

Glossary of Terms

ACIA	Asynchronous Communications Interface Adapter
A/D	Analogue to Digital
ADC	Analogue to Digital Converter
AM	Amplitude Modulation
ASCII	American Standard Code for Information Interchange
ASK	Amplitude Shift Keying
BCH	Bose Chaudhuri Hocquenghem
BER	Bit Error Rate
BFO	Beat Frequency Oscillator
BS	Block Select
CCD	Charge Coupled Device
CPU	Central Processing Unit
CZT	Chirp-Z Transform
CW	Carrier (or Continuous) Wave
D/A	Digital to Analogue
DAC	Digital to Analogue Converter
DDR	Data Direction Register
DFT	Discrete Fourier Transform
EOF	End of File
EPROM	Erasable Programmable Read Only Memory
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FSK	Frequency Shift Keying
HF	High Frequency
IC	Integrated Circuit
I/O	Input / Output

IRQ	Interrupt Request
LSI	Large Scale Integration
MPU	Microprocessor Unit
MSI	Midwest Scientific Instruments
MUF	Maximum Useable Frequency
NMI	Non Maskable Interrupt
PDR	Peripheral Data Register
PEP	Peak Envelope Power
PIA	Peripheral Interface Adapter
PROM	Programmable Read Only Memory
PSK	Phase Shift Keying
QPSK	Quaternary Phase Shift Keying
RAM	Random Access Memory
RES	Reset
ROM	Read Only Memory
R/W	Read / Write
RX	Receiver
SSB	Single Sideband
SSI	Small Scale Integration
SWTPC	South West Technical Products
TFDM	Time to Frequency Division Multiplexing
TTL	Transistor Transistor Logic
TX	Transmitter
VDU	Visual Display Unit
VHF	Very High Frequency
VLSI	Very Large Scale Integration
VMA	Valid Memory Address

CONTENTS

Glossary of Terms

CHAPTER 1 Introduction

- 1.1 History
- 1.2 The Ionosphere
- 1.3 Problems of the HF channel
- 1.4 Multipath propagation
 - 1.4.1 Overcoming multipath problems
- 1.5 Noise
 - 1.5.1 Overcoming noise problems
- 1.6 Microprocessors and HF radio systems
- 1.7 Conclusion

CHAPTER 2 Spectral Analysis Techniques

- 2.1 Introduction
- 2.2 The Microprocessor
- 2.3 Cassette Interface
- 2.4 The Discrete Fourier Transform
- 2.5 FFT Algorithm
 - 2.5.1 Bit-reversal shuffling
- 2.6 CZT Algorithm
- 2.7 Implementations
- 2.8 FFT implementation in BASIC
- 2.9 FFT implementation in Assembler
- 2.10 CZT implementation
 - 2.10.1 Post-multiplier unit
 - 2.10.2 Post-multiplier design
 - 2.10.3 Test results
- 2.11 Multiply/divide unit
- 2.12 Comparison of algorithms
- 2.13 Conclusion

CHAPTER 3 The HF Spectrogram

- 3.1 Introduction
- 3.2 System principles
- 3.3 System implementation
 - 3.3.1 The lowpass filter
 - 3.3.2 The analogue to digital converter
 - 3.3.3 The digital to analogue converters
 - 3.3.4 Software
- 3.4 Experimental results
- 3.5 Conclusion

CHAPTER 4 The Slave Processor System

- 4.1 Introduction
- 4.2 Principles of operation
- 4.3 Implementation
 - 4.3.1 Block select logic
 - 4.3.2 Address multiplexers
 - 4.3.3 Data bus buffers
 - 4.3.4 Control latch
 - 4.3.5 Clock drivers
 - 4.3.6 Memory
 - 4.3.7 Interfaces
- 4.4 Test results
- 4.5 Construction
- 4.6 Conclusion

CHAPTER 5 Error-control coding

- 5.1 Introduction
- 5.2 Block codes
- 5.3 Cyclic codes
- 5.4 The BCH codes
 - 5.4.1 Decoding BCH codes
- 5.5 Code implementation
 - 5.5.1 Encoder
 - 5.5.2 Decoder
 - 5.5.3 Galois field operations
- 5.6 Results
- 5.7 Burst error correction
- 5.8 Conclusion

CHAPTER 6 HF Interference pattern measurements

- 6.1 Introduction
- 6.2 System operation
- 6.3 Hardware
- 6.4 Software
- 6.5 Experimental Results
- 6.6 Conclusion

CHAPTER 7 The HF Data Modem

- 7.1 Introduction
- 7.2 System philosophy
- 7.3 Transmitter
 - 7.3.1 Data acquisition
 - 7.3.2. Encoding
 - 7.3.3 Modulation
 - 7.3.4 Channel evaluation & subchannel selection
 - 7.3.5 Synchronisation patterns
 - 7.3.6 Construction
 - 7.3.7 Transmitter testing
- 7.4 Receiver philosophy

CHAPTER 1

Introduction

1.1 History

The High Frequency (HF) Radio Band is that part of the electromagnetic spectrum extending from 3 to 30 MHz. Signals transmitted within this frequency range are predominantly propagated via single or multiple reflections from ionized regions within the upper atmosphere. These ionized regions are generally found at heights of 100-400 km. and are collectively known as the ionosphere. The phenomenon of ionospheric propagation has been used for long-distance communication since the pioneering work of Guglielmo Marconi carried out at the beginning of the century. Although the physics of the ionosphere has been studied extensively, the characteristics of the HF channel are often unpredictable. Nevertheless, the ionosphere provides an effective transmission path for a variety of communication traffic and is still widely used to communicate over long distances using a minimum of equipment.

The first successful experiments involving wireless information transmission using electromagnetic waves were based on digital signal representations for reasons of simplicity of the transmitter and receiver structures. This type of communication, called wireless telegraphy, was of great importance during the first half of the 20th century and still exists for certain specialised applications. The invention of electron valves, and later the transistor, gave rise to the development of wireless analogue transmission systems, and, with the discovery of different modulation schemes, the modern



wireless networks for information interchange were evolved; by radio relay, satellite, and ionospheric propagation methods.

The discovery that it was possible to transmit analogue signals by pulse code modulation led to a revival of interest in digital transmission. The successive change from analogue to digital circuits has given rise to the need for digital transmission over wireless networks (1). Considerable efforts are at present being made to investigate the performance of wireless channels in different frequency ranges with respect to digital signal transmission, the HF band being no exception.

Because of the problems associated with digital transmission over HF radio links, this medium has not been used successfully for reliable communication of medium- and high-speed data traffic. Certain investigations and experiments were carried out, mainly in the United States, during the 1950's and early 1960's (5,6), but were largely abandoned in favour of satellite communication links which yield higher bit rates with a reduced probability of error. However, many users, faced with problems of the cost and vulnerability of satellites, are prepared to tolerate a reduction in bit rate in exchange for economical systems using a virtually indestructive communications medium. Indeed, for mobiles operating over long-distance circuits, such as ships and aircraft, the HF path is almost the only alternative to satellite communication. VLF long HF

The recent advances in digital integrated circuit technology have enabled substantial reductions to be made in both the size

and the cost of communication systems (2) and it seems only logical that HF radio systems should also benefit from such developments. The work described in this thesis is concerned with the applications of VLSI (Very Large Scale Integration) technology to channel evaluation and data communication using the HF radio path. It is shown that systems which previously required large amounts of expensive analogue equipment can be realised at lower cost and with increased flexibility using nearly all digital techniques.

1.2 The Ionosphere

An effect of solar radiation is to cause ionisation of certain regions of the earth's atmosphere. This ionisation of gas molecules results in an electron density profile which is non-uniform with height. The region of the atmosphere having the highest electron density is known as the 'ionosphere', and it is this region which is responsible for the phenomenon of ionospheric propagation.

When a radio wave transmitted from the surface of the earth encounters a region of intense ionisation, it will be diverted from its original path by a refractive mechanism. The wave will also be considerably attenuated. At certain frequencies the refractive process is sufficient to 'bend' the wave through an angle exceeding 180° , at which point the attenuated wave will be returned to earth. The frequencies at which this phenomenon most commonly occurs lie within the range 3-30 MHz (the HF radio band). Because the refractive process appears to be one of reflection, this phenomenon is often referred to as 'ionospheric

reflection', a term which is subsequently used in the text.

The vertical electron density profile is non-linear; there are regions of ionisation which are more intense than others. The heights of these regions depend on many factors, one of which is the intensity of the solar radiation which varies from day to night, and with the seasons. In the long term, they are dependent on variations of the solar (sunspot) cycle. There are, however, three regions of chief importance, referred to as 'layers'. These are the E-layer at 120 km. (all figures are approximate), the F1-layer at 200 km., and the F2-layer at 300-400 km. At night, and in mid-winter, the F1 and F2 layers combine to form a single F-layer at 250 km. Below the E-layer there is a D-layer (at 50-90 km.) which is generally more important as an absorber than a reflector of radio waves since the attenuation at this altitude is somewhat greater than at higher regions.

Figure 1.1 is a simplified representation of a typical HF propagation path showing the ionosphere comprising two ionized layers, the E-layer and the F-layer. The signal is returned to earth via a single-hop E-layer mode together with single- and double-hop F-layer modes. Consequently, a short transmitted pulse will have three components when detected at the receiver, each component itself exhibiting time dispersion which causes a corresponding broadening of each received pulse. This latter form of dispersion is due to the fact that the transmitted energy illuminates the ionospheric layers over a relatively large area rather than at a single point; thus the energy of a given

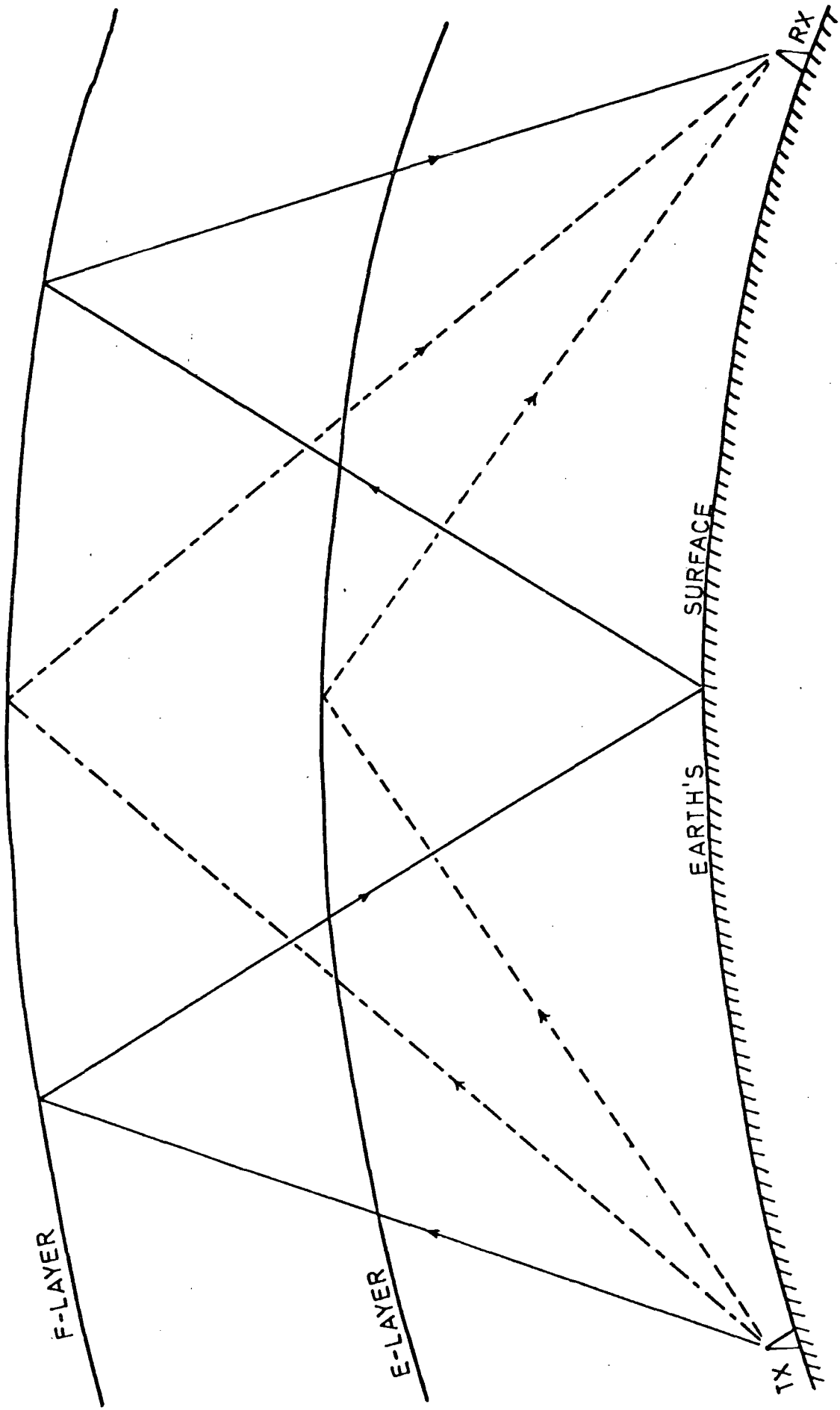


Figure 1.1 Simplified HF Propagation Path

received component can be considered as being made up from the energy of a large number of elemental returns, each with a slightly different propagation time, integrated over an appropriate area of the ionospheric layer.

There are many problems concerning the communicator who wishes to use the ionosphere as the transmission medium. Attempts have been made to model the HF channel (3), but the (largely unpredictable) time variation of the channel characteristics considerably increase the difficulties. Some of the problems are now discussed, and suggestions are included for some ways of combating the detrimental effects of the channel.

1.3 Problems of the HF channel

The problems associated with digital communication via ionospheric propagation may be classified into two broad categories:

(1) Multiple reflections from different layers in the ionosphere give rise to differential time delays in the received signal which results in intersymbol interference and fading.

(2) Noise, both natural and man-made, causes errors in the transmitted data stream.

Both problems are of equal concern in that they corrupt the received data to a sometimes considerable extent. Each will now be discussed separately.

1.4 Multipath propagation

The problem of intersymbol interference is inherent in any digital communications link but is especially important over HF radio links due to the comparatively long relative time delays introduced by multipath (or multimode) propagation. For a typical HF link of, say, 1000 km., the time delay between the shortest and the longest propagation path may be in the order of 2 to 3 milliseconds, the effect of which is to cause intersymbol interference of such severity as to considerably reduce the maximum allowable transmission rate. The time variations of the amplitudes and phases of the individual modes will be independent and any one mode may be dominant at a particular instant. In addition, phase cancellations may cause fading of the received signal. The result of multipath propagation, in any event, is to introduce errors in the transmitted data. Whether or not it is possible to recover these errors is dependent on the interpretation of the data at the receiver.

1.4.1 Overcoming multipath problems

Broadly speaking, the detrimental effects of multipath propagation may be reduced in four ways:

- (1) Choice of a suitable operating frequency
- (2) Time-to-frequency division multiplexing
- (3) Channel equalization
- (4) Diversity techniques

The highest frequency at which ionospheric propagation is possible for a given range is termed the Maximum Useable

Frequency (or MUF) and varies with the time of day and with the seasons as well as being severely affected in an unpredictable manner by solar flares etc. For an HF radio link, as the frequency of operation is increased, the number of propagation modes is reduced until single-mode propagation results at a frequency just below the MUF. It is possible to predict the MUF on a long-term basis, but the short-term fluctuations may considerably alter the prediction figure (4). However, if a large number of channels is available to the user, considerable improvements can result from the correct choice of operating frequency.

Time-to-frequency division multiplexing (TFDM) is a technique used to increase the transmission rate while keeping the signal element duration (frame rate) constant (5,6). The data stream is divided into short blocks for transmission over a number of frequency-parallel sub-channels, all contained within the allocated voice channel, and orthogonally spaced to avoid co-channel interference. Because the frame rate is constant, the signal element duration can be greater than the multipath spread, and the data rate may be increased by a factor which is the number of sub-channels that can be fitted in to the available channel capacity.

Channel equalization involves the construction of a matched filter to combat the degrading effects of the channel. Such a filter normally consists of a tapped delay line, whose tap outputs are weighted and summed to provide the filter output. Convolution of the received signal with the impulse response of

the filter should result in the original signal. Because of the time-varying nature of the HF medium, such an equaliser must be adaptive in nature, the number and position of the delay-line taps being continuously updated to allow for changes in the transmission medium (7). In practice, adaptive equalisers are difficult to implement and can be extremely costly.

Diversity may be employed in time, frequency, or space. A simple time diversity technique is simply to send the message more than once, thereby increasing the probability of correct reception. Dual frequency diversity requires transmission over two channels simultaneously. In one method, both channels are monitored and the channel which yields the most favourable error performance is selected. Two channel filters are therefore required at the receiver and the transmitted spectrum is doubled in width resulting in inefficient spectrum utilisation. Space diversity is the technique most commonly employed (8) and is based on the (usually correct) assumption that the time-variation of the phase cancellations at one point in space will be different to those occurring at another point which is located at a distance away which is comparable to the transmission wavelength. This method is quite effective but requires two receivers and two antennas. In all diversity techniques the problem of designing efficient combiners is a considerable one, and in any case a doubling of the resources is required in some sense.

1.5 Noise

The other major source of errors is noise, which may be naturally occurring, or which may arise from some sort of man-made disturbance. This latter type is often a result of congestion within the HF spectrum creating narrow-band interference from other users of the HF medium. A channel may be allocated to several users, who may interfere with each other if there is propagation between them. Serial data transmission systems (eg. radio telegraphy) suffer considerably from this type of interference (9). Broad band noise may be caused by machinery operating in the vicinity of the receiver although this sort of disturbance is normally localised in nature. A more common source of broad-band noise is that resulting from electric storms or other ionospheric disturbances which may generate a considerable amount of radiated energy in the HF spectrum. This type of noise generally occurs in bursts and is especially prevalent during the summer.

1.5.1 Overcoming noise problems

The detrimental effects of noise on the channel may be dealt with in a number of ways. Front end linearity in the analogue portion of the receiver is an important consideration when attempting to receive signals in the presence of heavy interference. Space diversity may assist in combating localised interference while time and frequency diversity may be effective in avoiding broad- and narrow-band noise respectively. A technique closely associated with TFDM is known as frequency agility which has been found to be effective in combating the effects of narrow-band noise (10) whereby a sounding technique is

used to select the "quietest" channel or in-band subchannel for subsequent data transmission. Sounding techniques are also used to select the best channel from a set of allocated channels by transmitting a sounding signal on each of the channels (11). The receiver then assesses the suitability of each of the channels for data communications and advises the transmitter via a feedback link.

Diversity methods imply some redundancy in their respective domains. A binary system may employ time diversity by the addition of redundant bits to the transmitted data. The redundancy may then be used as an aid in reducing the detrimental effects of noise on the transmitted bit stream.

A trade-off must first be made between the degree of fidelity required in the received data and the resulting reduction in the transmission rate caused by the addition of redundancy. Once this has been determined, the redundant bits can be added in a systematic manner which will then enable certain statistical categories of errors to be corrected and/or detected (12). As an example, one method is to use a block coding scheme in which the data stream is divided into a number of blocks each of which is k bits in length, and each of which is mapped into a 'codeword' of n bits ($n > k$) by adding a number $n-k$ bits which are the results of $n-k$ modulo-2 additions (or parity checks) on the original k data bits. The resulting codewords then form part of a linear block code which obeys a fixed set of parity-check rules. If the parity checks do not agree on reception, errors have occurred and the appropriate bits may be corrected. Some of

the most powerful block codes known are of length $n = 2^{m-1}$ (m is an integer) and are capable of correcting up to t random errors occurring within a code word and require, at most, the addition of mt check bits (13). Errors occurring on HF radio links are generally greater than one bit in length and it is therefore preferable to use codes which can correct bursts of errors. However, random-error-correcting codes can be used very effectively to correct bursts of errors by interleaving groups of codewords such that a burst of consecutive errors is distributed over several codewords, the errors appearing as random to the decoder.

In general, then, the ionosphere can be considered to be an anisotropic time-varying medium which causes the HF path to exhibit error rates far greater than those found in other communication systems (14). The results of multipath propagation and noise are to introduce large numbers of random and burst errors which may sometimes be so great as to render the channel virtually useless. Indeed, to an outsider, communicating via such a medium may seem a near impossible task. However, it seems that adaptive schemes combining several of the above-mentioned techniques may yield substantial performance improvements. Until recently, adaptive schemes have been complex and costly to implement, and almost all systems described in the literature have not had adaptive capability (15).

1.6 Microprocessors and HF Radio Systems

One of the most significant products of the VLSI technology previously discussed is the microprocessor. This is a monolithic device which is obtainable at low cost and which may be made to perform a wide variety of processing tasks by the choice of a suitable sequence of instructions stored in a read-only memory (ROM) unit. The system is configured such that the microprocessor may access the stored instructions and execute them accordingly. Major system changes may be implemented by simply modifying the instruction sequence (the 'software'). A microprocessor system may be made adaptive by determining that the order of execution of the instruction sequence is dependent on previous and/or present events.

This thesis describes the applications of microprocessor techniques to several aspects of digital communications over HF radio channels. The next chapter discusses the implementation of several discrete-time signal processing techniques which are used in subsequent chapters in this thesis. An HF "spectrogram" sounding technique is described in chapter 3, from which a visual display of the time-frequency characteristics of an HF voice channel may be obtained. Information from the display may subsequently be used to determine the suitability of the channel for data transmission.

During the course of the project, it became evident that the data processing requirements for some applications exceeded the capability of a single microprocessor unit. It was for this reason that a multiprocessor system (chapter 4) was developed, in

which several 'slave' processor units operate under control of a central, or 'master' processor. This configuration allows a considerable increase in data throughput over a single processor system. Various error control coding schemes were investigated (chapter 5), leading to the implementation and subsequent field testing of several software-based schemes for the correction of errors occurring over HF radio data links. Chapter 6 presents some results obtained from a detailed investigation into interference phenomena observed in an HF communications channel; the implications for the design of data modems are discussed.

The ideas and results obtained from previous chapters in this thesis were combined in chapter 7 to implement an adaptive HF data modem for data transmission over HF radio links. Signal processing, error-control coding and multiprocessor techniques are used in a low-cost system for medium-speed communication which attempts to overcome the detrimental effects of the HF channel. Some results obtained using the equipment are described in chapter 8.

1.7 Conclusion

The last decade has seen a resurgence of interest in communication via the ionospheric medium. The problems associated with data transmission via this unpredictable channel are considerable, and many techniques have been developed in attempts to combat the degradation imposed on the propagated signal. Some of these techniques have been utilised in systems which have been designed exclusively for HF radio data transmission, but have been complex and costly to implement.

Microprocessors have made substantial inroads in several fields of digital data communications and are well suited to systems requiring adaptability. This thesis describes some applications of these devices to data transmission over HF radio links and shows that considerable savings in hardware requirements may be made by using a primarily software-based approach to system design.

2.1 Introduction

The emergence of digital signal processing as a major discipline began in the mid-1960's when high speed digital computers became available for research and development work (16). Many concepts that form the theoretical basis of digital signal processing, such as the z-transform and Fourier analysis, had been familiar, however, to engineers for a long time. In the ensuing years, the field has matured considerably and its development is intimately tied with technological advancement in device design and fabrication.

Many of the signal processing requirements of HF radio communications systems may be realised using digital techniques implemented with the aid of microprocessors. In general, an analogue signal is sampled using an analogue-to-digital converter, the resulting set of samples is processed by a microprocessor-based system, then converted back to analogue form by a digital-to-analogue converter. The efficiency and accuracy of the signal processing depends to a great extent on the design of the system software.

This chapter introduces the microprocessor chosen for the subsequent work described in this thesis. The design and operation of a novel system used for data storage on magnetic tapes is then discussed. More complex techniques are then introduced, based on the Discrete Fourier Transform (DFT). The DFT plays an important role in the analysis, the design, and the

implementation of digital signal processing systems concerned with HF radio systems and is used in several of the applications described in following chapters. Software and hardware implementations of efficient algorithms for the computation of the DFT are investigated and compared. An assessment is made of the suitability of such implementations for demodulation and spectral analysis in HF radio systems.

2.2 The Microprocessor

The choice of a suitable microprocessor was a primary consideration in the development of the project. A device was required having a comprehensive instruction set, fast execution speed, and for which support facilities were available. The Motorola M6800 (17) satisfied these requirements and was chosen in view of the following merits:

- (1) The M6800 has a powerful and versatile instruction set (72 basic instructions and 7 addressing modes).
- (2) Several economical and flexible development systems were available.
- (3) A cross-assembler and simulator were available on the university computer.
- (4) Software packages, ancillary components, and documentation were readily available.
- (5) Execution time is fast (nominal clock frequency = 1MHz; average instruction time = 4 cycles (approx.)).

The M6800 is a monolithic microprocessor having an 8-bit data bus and a 16-bit address bus. It has six internal registers: the A accumulator (8 bits), the B accumulator (8

bits), an index register (16 bits), a program counter (8 bits), a stack pointer (16 bits) and a condition code (status) register (8 bits). The device requires a non-overlapping bi-phase clock for normal operation.

Two systems based on this device were constructed for the development of software during the project (18). Each system consisted of (1) a "motherboard", with 7 card slots available on the main bus (called the SS-50), and with 8 slots available on the I/O bus (called the SS-30), (2) a CPU card containing the microprocessor, crystal oscillator, baud rate generator, ROM (containing a monitor program), and 128 bytes of "scratchpad" RAM, (3) two 16 kbyte RAM cards, and (4) a serial interface card enabling the system to be controlled from a teletype or a VDU. An EPROM programming card was available, allowing machine code programs (object data) to be transferred from RAM to 2 kbyte EPROMs under software control.

The only mass storage system available during the first few months of the project was a teletype paper tape punch/reader having an operating speed of 110 baud. An editor and an assembler were available as aids to program development; each of these packages, however, required approximately half an hour to load into memory using the paper tape reader. It became evident that a more efficient means of mass storage was necessary if reasonable progress was to be made. It was for this reason that a magnetic tape storage system was subsequently adopted. A novel interface to the tape unit was developed which required a minimum of additional hardware (only 2 I/O lines of a Peripheral

Interface Adapter (PIA) IC were needed). The design of this interface is discussed in the following section and is also described in the publication to be found in Appendix 4 (reference 19).

2.3 Cassette interface

The "Kansas City" standard is commonly regarded as an internationally accepted standard for data recording on audio cassettes. The baseband data signal is used to modulate carriers of 1400 Hz and 2800 Hz using FSK at a rate of 350 bps., ie a logic '0' is represented as 8 cycles of the higher frequency and a logic '1' as 4 cycles of the lower frequency. It was initially decided to design an interface based on this standard but this was later developed into a higher speed system.

The PIA (Peripheral Interface Adapter (20)) is a device which provides an interface between the 6800 microprocessor bus and two external data ports of 8 lines each (the 'A' and the 'B' ports). Each of the 16 lines is compatible with standard TTL logic families and may be programmed for input or output by setting the state of the corresponding bit in an internal "data direction register". For the cassette interface, line A₇ was programmed for output and line B₇ for input. Assembly level software was written which would take data from a specified region of memory and generate the appropriate FSK signal at the output line A₇. Cycle timing of the rectangular modulated signal was achieved using software timing loops and data was transmitted in byte format using a '0' start bit and a '1' stop bit. The TTL level FSK output was connected directly to the

microphone input of the cassette recorder, which incorporated an automatic record level preamplifier. On playback, the audio output from the extension speaker socket was connected directly to the input line B₇ which was continuously sampled by the processor. The period of a waveform cycle was determined by comparing the number of '1' samples counted with a mean threshold value. If the number of samples counted exceeded this value the cycle was of 1400 Hz; if less it was 2800 Hz. The sampling rate was approximately 12 kHz. A decision as to the state of a demodulated data bit could be made after receiving the appropriate number of cycles. The average error rate from this system on playback was found to be better than 1.5×10^{-6} . (ie. less than 1 error per 40 minute recording)

Some experimentation was carried out with adjustment of the recording frequencies and number of cycles per bit. A reliable and much faster system was found to result from using only single cycles of the original two frequencies; ie. a logic '0' takes only half the time of a logic '1' to load and store. An example of the signal generated for the data sequence 10110 is shown in figure 2.1. If the data contains equal numbers of 1's and 0's, the mean bit rate is 1600 bps, more than 5 times that of the Kansas City standard.

It was decided to incorporate error checking by recording the data according to the Motorola "MIKBUG" standard format for recording object data on punched paper tape (21). This format requires data to be transmitted in ASCII character records arranged as in figure 2.2. All information in the record is

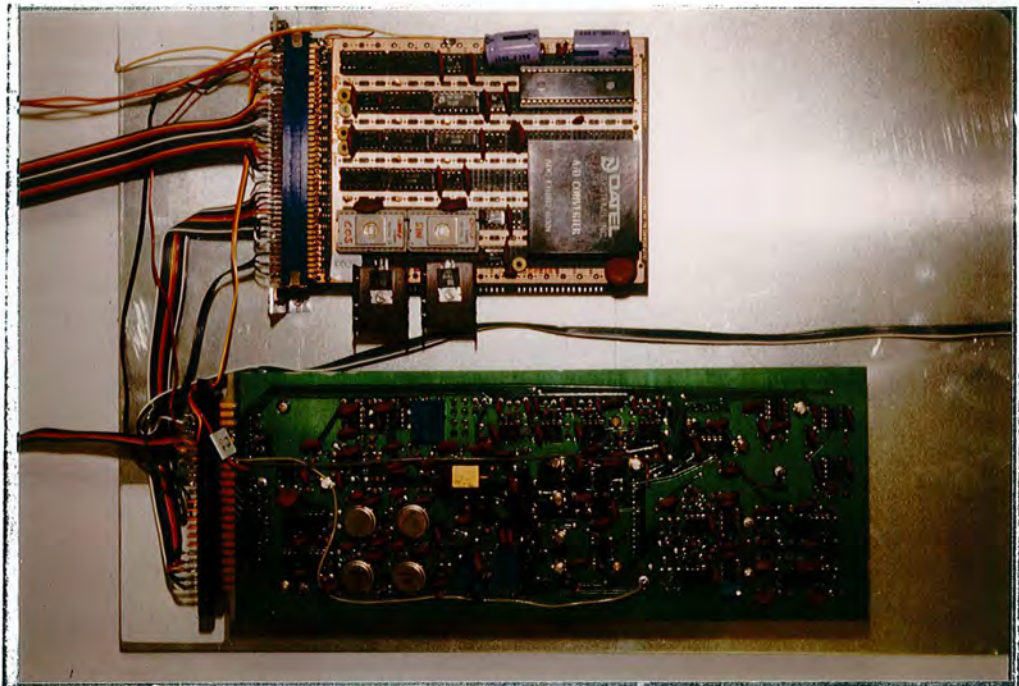


PHOTO 2.1. SHOWING POST-MULTIPLIER UNIT INTERFACED TO CCD EVALUATION MODULE.

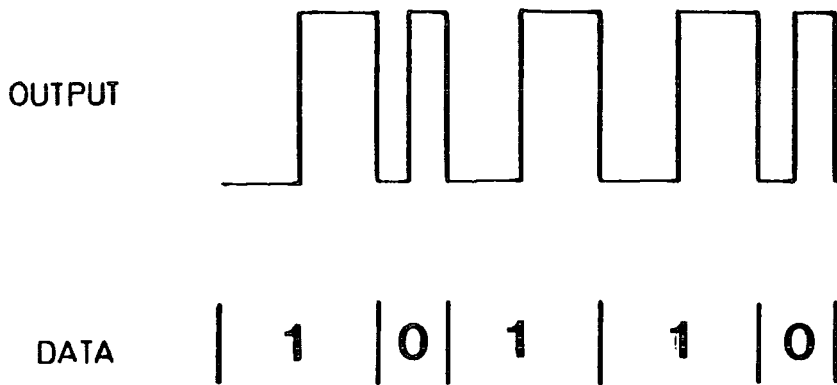


Figure 2.1 Cassette Data Format

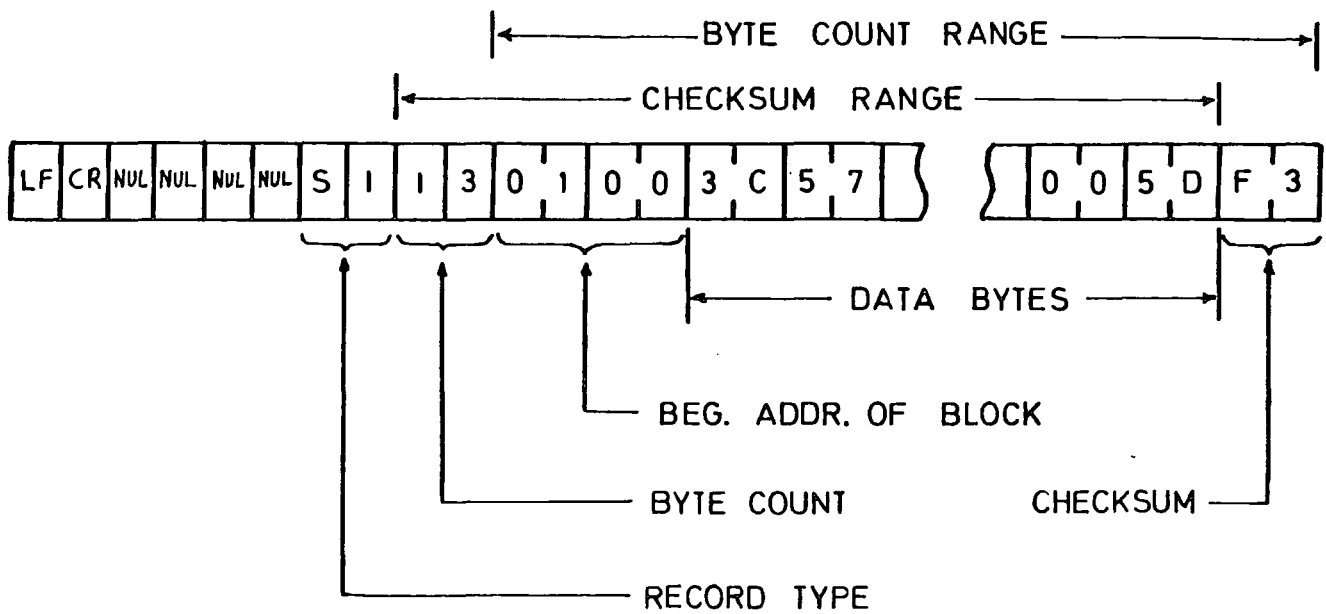


Figure 2.2. "MIKBUG" Tape Format

represented as hexadecimal data. The beginning of the record contains the record type (S1), a byte count which covers all the bytes that follow, and the start address of the data block. Data bytes follow which represent the object data to be stored in memory beginning at the block address and stored in sequential memory locations that follow. At the end of the record is a checksum, which is the 1's complement of the summation (mod 256) of all data bytes in the record, plus the byte count and block address. This value is checked as data is loaded and a '?' is printed if an error is encountered. The tape can then be rewound a few blocks and the faulty block reloaded. The end-of-file (EOF) terminates the data and consists of the characters "S9". The EOF terminates a tape load function.

An assembly listing of the tape store and load programs is included in the published article at the end of this thesis (Appendix 3). The high-speed interface showed error rates at least as good as the Kansas City version, and was used successfully for several months before the purchase of a flexible ('floppy') disc drive and interface board.

During the second year of the project, a triple disc drive and interface board were obtained to facilitate faster file access. Each flexible disc could contain 80 kbytes of storage; it was therefore possible to have access to 240 kbytes of data at any one time. Commercially available software was obtained, including a co-resident assembler, editor, and BASIC interpreter, to assist program development. The teletype was replaced with a VDU for communication with the development system and hard copy

output was obtained from a thermal printer attached to an additional serial interface card connected to the SS-30 bus.

2.4 The Discrete Fourier Transform

It has already been mentioned that the DFT plays an important role in the analysis and processing of HF radio signals. The DFT of a finite length sequence, $\{x(n)\}$, is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} \quad , \quad k = 0,1, \dots , N-1 \quad (2.1)$$

where $W_N = \exp(-j2\pi/N)$.

The resulting sequence, $\{X(k)\}$, is the Fourier representation of the original input sequence. The following sections of this chapter discuss efficient microprocessor implementations of the DFT using software alone, using hardware arithmetic, and with a Charge Coupled Device (CCD), each of which was used in a different application. A description of the algorithms is given, followed by a discussion of the implementations, and finally a comparison of the implementation efficiencies.

2.5 FFT Algorithm

The direct computation of the DFT requires $4N^2$ real multiplications and $N(4N-2)$ real additions, or, equivalently, N^2 complex multiplications and $N(N-1)$ complex additions (22-24). The number of multiplications required is generally accepted as being a meaningful measure of the complexity, or, of the time required to implement a computational algorithm. The amount of computation required to evaluate the DFT directly is approximately

proportional to N^2 ; the computation time required to compute the DFT by this method therefore becomes very large for large values of N .

It is possible to reduce the number of computations required to evaluate the DFT by decomposing the sequence $\{x(n)\}$ into successively smaller subsequences. Algorithms based on this principle are called "decimation in time" algorithms (22,23). If N is an even number, we can consider computing $X(k)$ by separating $\{x(n)\}$ into two $N/2$ -point sequences consisting of the even- and odd-numbered points in $x(n)$.

We then obtain

$$X(k) = \sum_{n \text{ even}} x(n)W_N^{kn} + \sum_{n \text{ odd}} x(n)W_N^{kn}$$

Substituting $n=2r$ for n even and $n=2r+1$ for n odd gives

$$\begin{aligned} X(k) &= \sum_{r=0}^{(N/2)-1} x(2r)W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x(2r+1)W_N^{(2r+1)k} \\ &= \sum_{r=0}^{(N/2)-1} x(2r)(W_N^2)^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1)(W_N^2)^{rk} \end{aligned}$$

but $W_N^2 = \exp(-2j(2\pi/N)) = \exp(-j2\pi/(N/2)) = W_{N/2}$

$$\begin{aligned} \therefore X(k) &= \sum_{r=0}^{(N/2)-1} x(2r)W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1)W_{N/2}^{rk} \quad (2.2) \\ &= G(k) + W_N^k H(k) \end{aligned}$$

Each of the sums in equation 2.2 is recognised as an $N/2$ point DFT. If N is an integer power of 2, each of the sums may be decomposed further into two $(N/4)$ point DFTs and so on until the

stage is reached where there are $N/2$ 2-point DFTs (or "butterfly" computations) to be performed. There will be $\log_2 N$ stages of decomposition in all. The basic butterfly computation is illustrated in figure 2.3 and is described by the following equations:

$$\begin{aligned} X_{m+1}(p) &= X_m(p) + W_N^r X_m(q) \\ X_{m+1}(q) &= X_m(p) - W_N^r X_m(q) \end{aligned}$$

Only one complex multiplication is involved in mapping the point $X_m(p)$ and $X_m(q)$ to $X_{m+1}(p)$ and $X_{m+1}(q)$ respectively. Figure 2.4 shows a graphical representation of the evaluation of an 8-point DFT using the butterfly computation of figure 2.3.

The total number of multiplications required for the above algorithm is $(N/2)\log_2 N$ compared with N^2 multiplications required for direct evaluation of the DFT. The algorithm was derived by Cooley and Tukey in 1965 (25), and is often referred to as the Fast Fourier Transform algorithm (FFT). A flow diagram for the FFT decimation in time algorithm is shown in figure 2.5. Three variables, i , m , and l , are required to index through the complex array $A(*)$. There are $r=\log_2 N$ stages of computation and $N/2$ butterfly computations to be performed at each stage.

2.5.1 Bit-reversal shuffling

If the above algorithm is to be used to produce an output sequence in sequential order, it is necessary to store the input data in non-sequential order. In determining the position of $x(n)$ in the input array, the bits of the binary representation of the index n must be reversed. For example, for a 16 point transform,

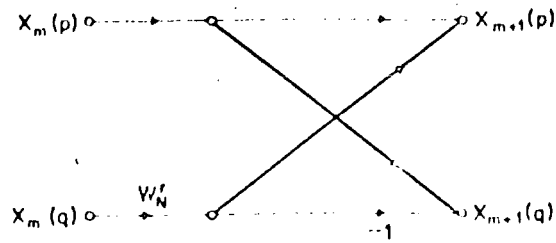


Figure 2.3. Basic Butterfly Computation

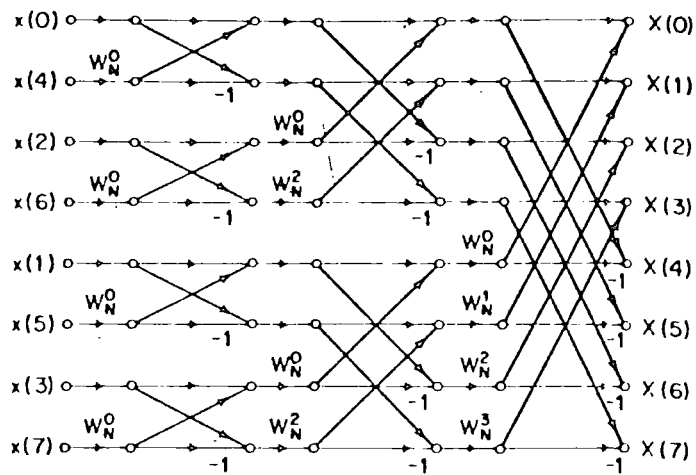
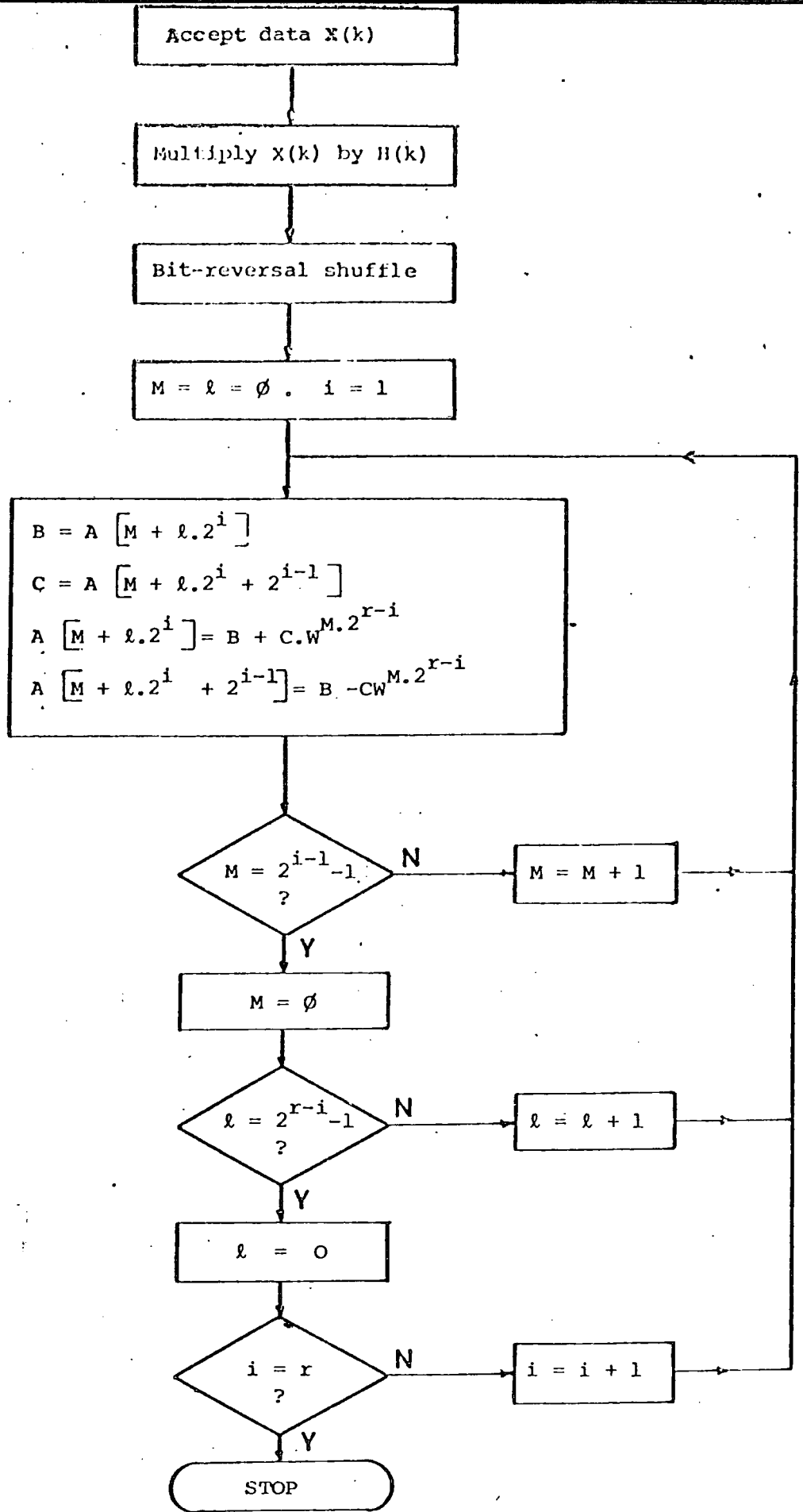


Figure 2.4. Flowgraph of 8-point DFT using butterfly computation of figure 2.3 .



All multiplications are complex. $w^k = e^{-j \left(\frac{2\pi}{N} \right) k}$

FIGURE 2.5. FFT ALGORITHM FLOWCHART

four bits are required for each index and an index $(b_3b_2b_1b_0)$ becomes $(b_0b_1b_2b_3)$. The necessity for bit-reversal shuffling of the sequence $x(n)$ is a result of the manner in which the DFT computation is decomposed into successively smaller DFT computations. Figure 2.6 shows the flowchart of an algorithm which will shuffle the contents of the array $x(i)$, $i=0,1,2 \dots N-1$ in bit-reversed order; the result will be contained in the same array. Each of the numbers $x(i)$ may be complex. Three integer variables, i , j , and k are used to index through the data and the complex variable X is used as a temporary storage location. This algorithm was implemented in BASIC and assembly level language for use with the respective FFT programs.

2.6 CZT Algorithm

Another algorithm for evaluating the DFT, called the "chirp-z" transform (CZT), was derived in 1969 by Rabiner and Schafer (26), and is not restricted to integral powers of N as is the FFT. The expression for the DFT in 2.1 is:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N}, \quad k = 0,1, \dots, N-1$$

where both $x(n)$ and $X(n)$ may be complex. Using Bluestein's identity (27)

$$nk = \frac{1}{2} [n^2 + k^2 - (k-n)^2]$$

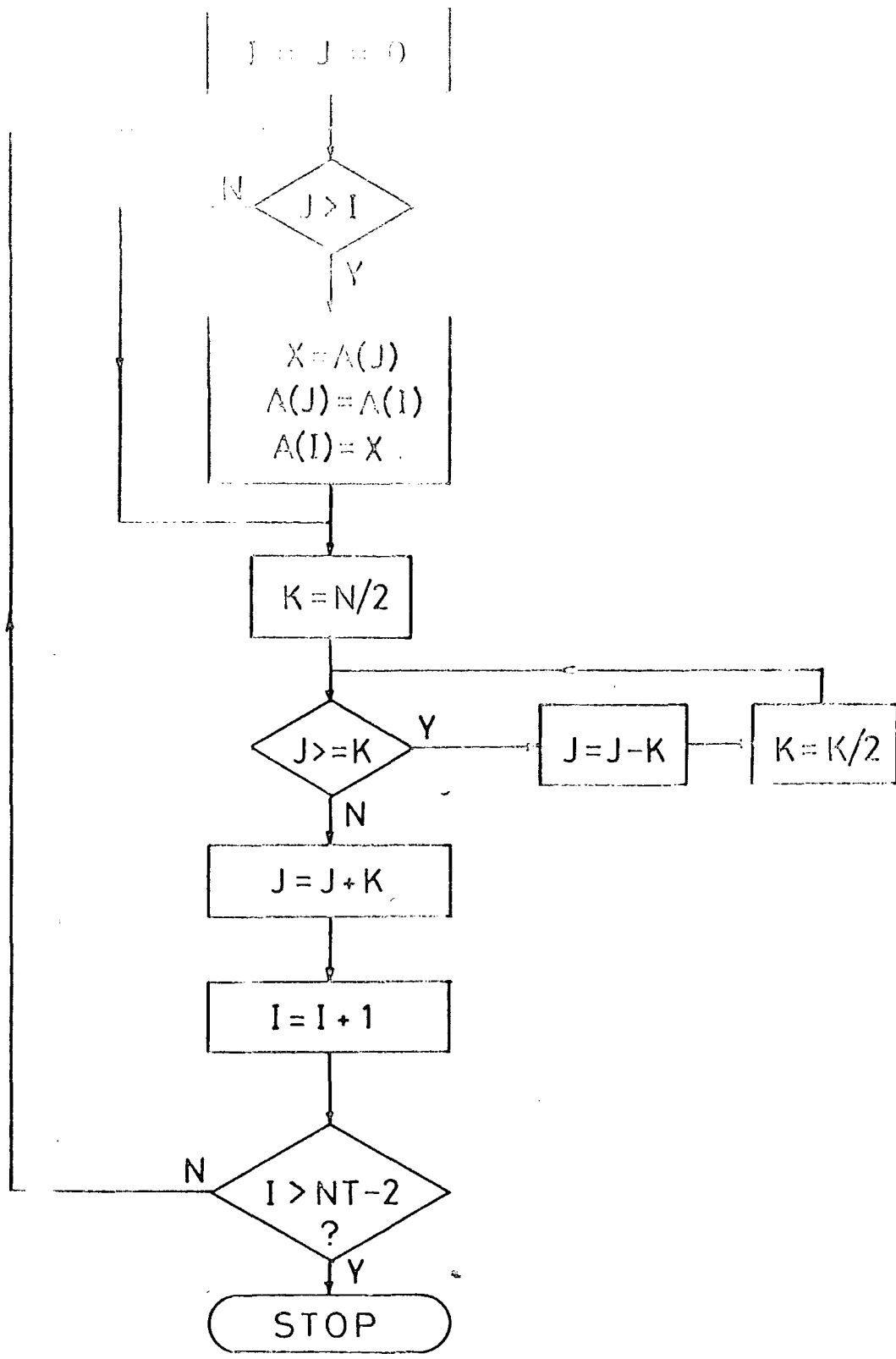


FIGURE 2.6. BIT-REVERSAL SHUFFLING

we obtain

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n)e^{-j\pi n^2/N} e^{j\pi(k-n)^2/N} e^{-j\pi k^2/N} \\ &= e^{-j\pi k^2/N} \sum_{n=0}^{N-1} g(n) e^{j\pi(k-n)^2/N} \quad , k=0,1, \dots N-1 \quad (2.3) \end{aligned}$$

Equation (2.3) is the expression for the CZT. Three stages of computation are required

- (i) Multiply each term $x(n)$ by the complex factor $\exp(-j\pi n^2/N)$ to produce a new sequence $g(n)$.
- (ii) Perform a discrete convolution between the sequence $g(n)$ and the sequence $\exp(j\pi n^2/N)$.
- (ii) Multiply the resulting output sequence by the factor $\exp(-j\pi k^2/N)$ for each point of $X(k)$.

The sequences $\exp(-j\pi n^2/N)$ and $\exp(-j\pi k^2/N)$ can be thought of as complex exponential sequences with linearly increasing frequency. Such signals are called "chirp" signals; hence the name "chirp-z" transform. A method for implementing the CZT using a charge-coupled device is described later in this chapter.

2.7 Implementations

Four methods for implementing the DFT using the two algorithms described were investigated and used in various applications described elsewhere in this thesis:

- (i) Implementation of FFT algorithm using a co-resident BASIC interpreter
- (ii) Assembly-level implementation of the FFT using software

arithmetic

(iii) Assembly-level implementation of the FFT using hardware multiply/divide unit

(iv) Hardware implementation of the CZT using a charge coupled device

2.8 FFT Implementation in BASIC

A program to evaluate the DFT of an N-point real sequence is shown in the listing in appendix 2. The following results may be obtained from this program: printout of input data, printout of output data (real & imaginary components), plot of output data (real & imaginary), printout of power spectrum and plot of power spectrum. A window function may be applied to the input data if a power spectrum is required. A special feature of this program is that the DFT of the original N-point input sequence is evaluated by performing a single N/2-point FFT computation. The spectrum of a purely real sequence exhibits complex conjugate symmetry and it is therefore only worthwhile computing the positive frequency half of the spectrum.

Any asymmetric function may be formed as the sum of two functions, symmetric about some suitable axis, one possessing even and the other odd symmetry. Using this fact it is possible to simultaneously compute the spectra of two N/2-point real sequences using only one N/2-point complex FFT. One sequence $y(n)$ is entered as the real components in an N/2-point complex array; the other, $z(n)$, is entered as the imaginary components. The transform, $X(k)$, of the sum $x(n)=y(n)+jz(n)$ is asymmetric but it is possible to use the symmetry property to obtain the spectra

$Y(k)$ and $Z(k)$ of $y(n)$ and $z(n)$ respectively by a manipulation of the sequence $X(k)$. If the sequences $y(n)$ and $z(n)$ are the even and odd numbered points respectively of an N -point real input sequence, it is possible to obtain the spectrum of this sequence from $Y(k)$ and $Z(k)$. This method is described in reference (24).

The even-numbered points of the real input sequence are entered as the real components of the complex array $A(2,N/2)$, and the odd-numbered points are entered as the imaginary components. An $N/2$ -point DFT is performed by the subroutine at statement 1500 which shuffles the data in bit-reversed order, then evaluates an $N/2$ point FFT. The subroutine at statement 520 manipulates the transformed data to obtain the spectra of the two real sequences. The positive half of the spectrum of the original input data sequence is then found using the subroutine beginning at statement 130. Real and imaginary components of the spectrum may be listed and/or plotted if required. The power spectrum of the transform is found by evaluating the square root of the sum of the squares of the real and imaginary components of each point in the output sequence.

2.9 FFT Implementation in assembler

The FFT algorithm of figure 2.5 was implemented in M6800 assembly language in order to obtain an increase in computation speed over a high level language implementation. A listing of the program is given in appendix 2 and is capable of evaluating the DFT of N complex data points where N is an integer power of 2 between 8 and 256 inclusive. The real and imaginary components of each point are quantised to 8 bits each but arithmetic operations

are carried out to 16 bit accuracy to eliminate quantisation errors introduced between stages of the decimation-in-time implementation. 1 kbyte of memory was therefore necessary to accomodate 256 complex points as 2 bytes were required for each of the real and imaginary components. Data is arranged in the table as follows:

address	data
DBASE+4n	f(n) (real) MSB
DBASE+4*n+1	f(n) (real) LSB
DBASE+4*n+2	f(n) (imag) MSB
DBASE+4*n+3	f(n) (imag) LSB

To avoid computing the sine and cosine values for the weighting factors, a table of lookups was used. Values in the table corresponded to the functions:

$$W_{RE}(n) = \cos (2\pi n/256) \quad n = 0,1,2, \dots 127$$

$$W_{IM}(n) = -\sin (2\pi n/256) \quad "$$

Each complex point was stored as two 16 bit 2s complement numbers in a similar format to the data storage format described above. 512 bytes were therefore required for lookup table storage. The binary representations of the weighting factors were as follows:

W	binary
+1	0100000000000000
0	0000000000000000
-1	1100000000000000

The program is invoked by first loading the data into the data table, loading r (=log₂N) with a value between 3 and 8 and executing the call "JSR FFT". The data is first shuffled in

bit-reversed order by the subroutine SHUF which is a direct implementation of the flowchart of figure 2.6. Successive butterfly computations are then performed in-place each of which involves 4 real 16 x 16 bit multiplications, 3 real 16 + 16 bit additions and 3 real 16 - 16 bit subtractions.

Two implementations were evaluated; the first used a software multiplication routine based on the Booth algorithm of reference (28); the second used a hardware multiply/ divide unit, described in section 2.11. Execution times for both implementations are tabulated in section 2.12. A further improvement in computational efficiency can be obtained by observing that the weighting factor W^0 has real and imaginary coefficients of 1 and 0 respectively and no multiplications are required for this case. The total number of real multiplications required for the N-point DFT therefore reduces to

$$4 \times ((N/2)\log_2 N - (N-1))$$

2.10 Implementation of the CZT

As discussed in section 2.6, the CZT algorithm involves three stages of computation: pre-multiplication, convolution, and post-multiplication (29). The block diagram of a complete transform based on the CZT algorithm is shown in figure 2.7. Pre-multiplication is accomplished by the multipliers to the left of the diagram and post-multiplication by those on the right. The major computing task is the convolution portion; the Reticon R5601 quad chirped transversal filter (30) has been designed to perform this task. This device contains two separate 512-stage MOS charge coupled devices which are used to implement four transversal

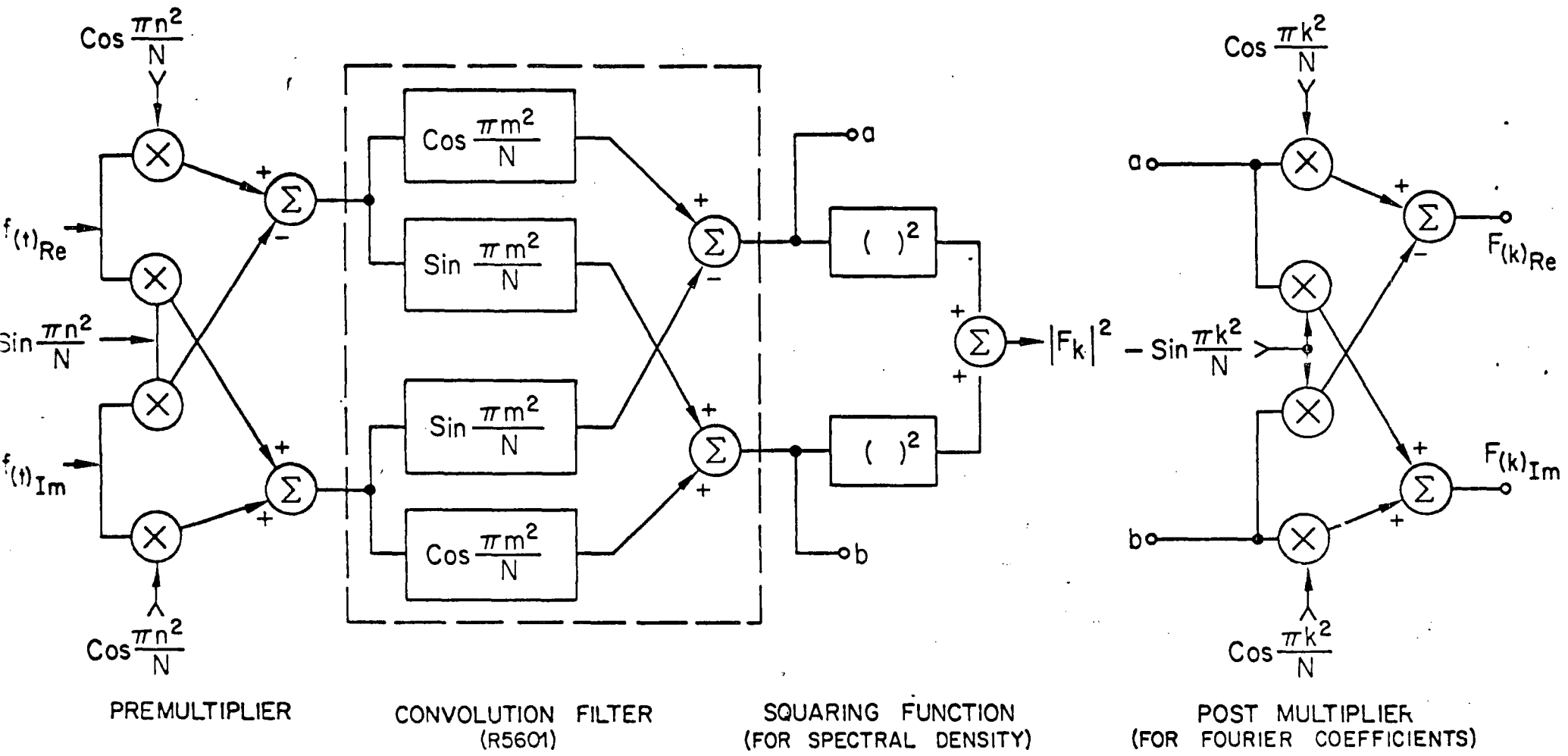


FIGURE 2.7. IMPLEMENTATION OF CHIRP-Z TRANSFORM ALGORITHM.

filters using a split-electrode weighting technique (31). The filter weighting coefficients and internal circuit connections are configured so that the device, in conjunction with additional off-chip components, can implement the CZT algorithm to calculate a 512-point DFT (32,33).

An evaluation module containing the R5601 device was available, which included additional circuitry necessary to compute the power spectrum of an analogue input signal. No phase information is obtainable with this module, as the post-multiplier unit is replaced with a hypotenuse function which recovers the spectral amplitude from the component cosine and sine terms. From (2.3), the squared spectral amplitude of a sequence $x(n)$ can be expressed as:

$$\left| X(k) \right|^2 = \left| \sum_{n=0}^{N-1} x(n) e^{-j\pi n^2/N} e^{j\pi(k-n)^2/N} \right|^2 \quad (2.4)$$

The final phase multiplier term, $e^{-j\pi k^2/N}$ has unit magnitude and has therefore been omitted from the above expression. The input data is stepped each time a new spectral component is calculated. Equation (2.4) then becomes:

$$\left| X_s(k) \right|^2 = \left| \sum_{n=0}^{N-1} x(n+k) e^{-j\pi n^2/N} e^{j\pi(k-n)^2/N} \right|^2$$

The notation $X_s(k)$ indicates a "sliding" CZT.

A further simplification is possible if the input is purely real, as it is in this case. The imaginary input is always zero so that two of the input multipliers may be deleted and the input

circuit simplified. A block diagram of the evaluation module is shown in figure 2.8. The real (analogue) input signal is buffered and converted to discrete-time samples by the input sample-and-hold, then split into the direct and quadrature channels. The sample values are multiplied by the appropriate chirped waveform using multiplying digital-to-analogue converters. The digital inputs to these converters are derived from two 512-by-8 bit ROMs which contain the sampled chirped sine and cosine waveforms. The sampled analogue products are then used for input to the R5601 four-channel convolution filter. Outputs from the filter are sampled and held to give time coincidence of all outputs, and then combined on an rms basis to give the spectral density of the input waveform.

Four clock phases are required by the filter device to propagate the discrete signal packets through the CCD channels. These are designated T1CL, T2CL, T3CL and T4CL, and are generated by a multi-phase clock generator circuit incorporated in the evaluation module which may be driven either from a 1 MHz internal oscillator or from an external trigger source. The sample rate with the on-board oscillator is a nominal 100 kHz, but lower rates are attainable with external triggering. The "address advance" pulse increments a 9 bit counter which addresses the weighting factor PROMs.

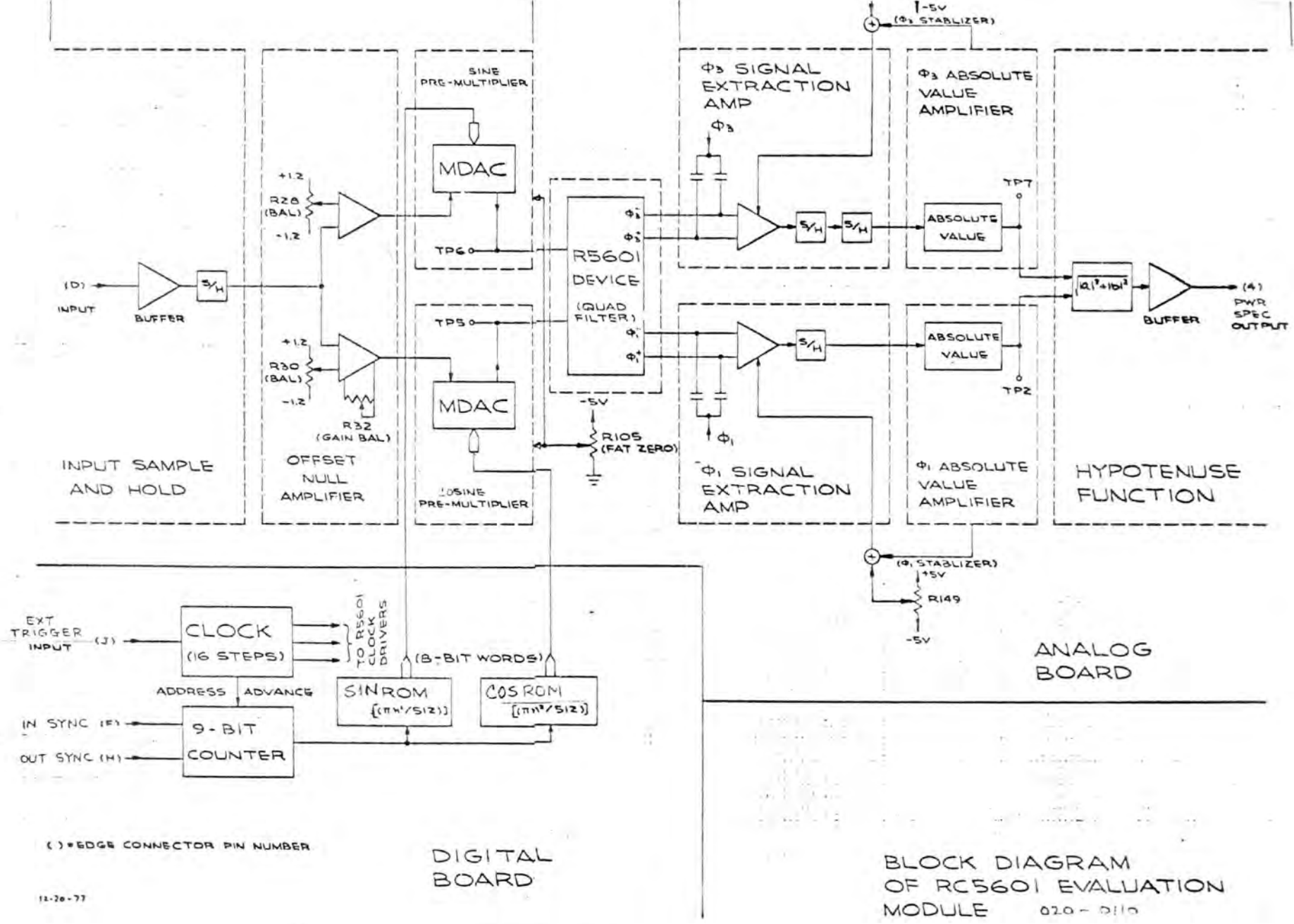


FIGURE 2.8.

2.10.1 Post-multiplier unit

A post-multiplier unit was designed and constructed, enabling the real and imaginary components of the signal spectrum to be determined. The weighting factors for the algorithm illustrated in figure 2.7 are the sine and cosine chirp signals $e^{-j\pi k^2/N}$. However, because the transform is sliding and not stationary, the coefficients $X_s(k)_{Re}$ and $X_s(k)_{Im}$ do not provide a true measure of the input signal phase. A correction factor must be applied if the phase is to be restored. Consider the examples illustrated in figure 2.9 where the 8-point DFT's are evaluated of two sinusoids of frequencies $4 \cdot \pi n/N$ (solid line), and $2 \cdot \pi n/N$ (broken line).

The stationary DFT of an input sample sequence $\{x(n)\}$ is derived from the time samples $x(i)$, $0 \leq i \leq N-1$. The stationary DFT of the first signal in the example results in an output spectrum of $-4j$ at $k=2$ and $+4j$ at $k=6$, while that of the latter results in an output spectrum of similar coefficients at $k=1$ and $k=7$. The sliding DFT, however, is derived from time samples $x(i)$, $n+k \leq i \leq N-1+k$. The sliding DFT results in an output spectrum of $+4j$ ($k=2$) and $-4j$ ($k=6$) for the first signal, and $(\sqrt{8} - \sqrt{8}j)$ ($k=1$) and $(\sqrt{8}j - \sqrt{8})$ ($k=7$) for the latter. Although the magnitude of the power spectrum is the same as for the stationary DFT, the sliding transform imposes an additional phase shift of $2 \cdot \pi k^2/N$ radians, which must be corrected for by multiplying the output spectrum by $e^{-j2 \cdot \pi k^2/N}$.

It has been stated that, in the case of the CZT implementation, the post-multiplication coefficients are the chirp

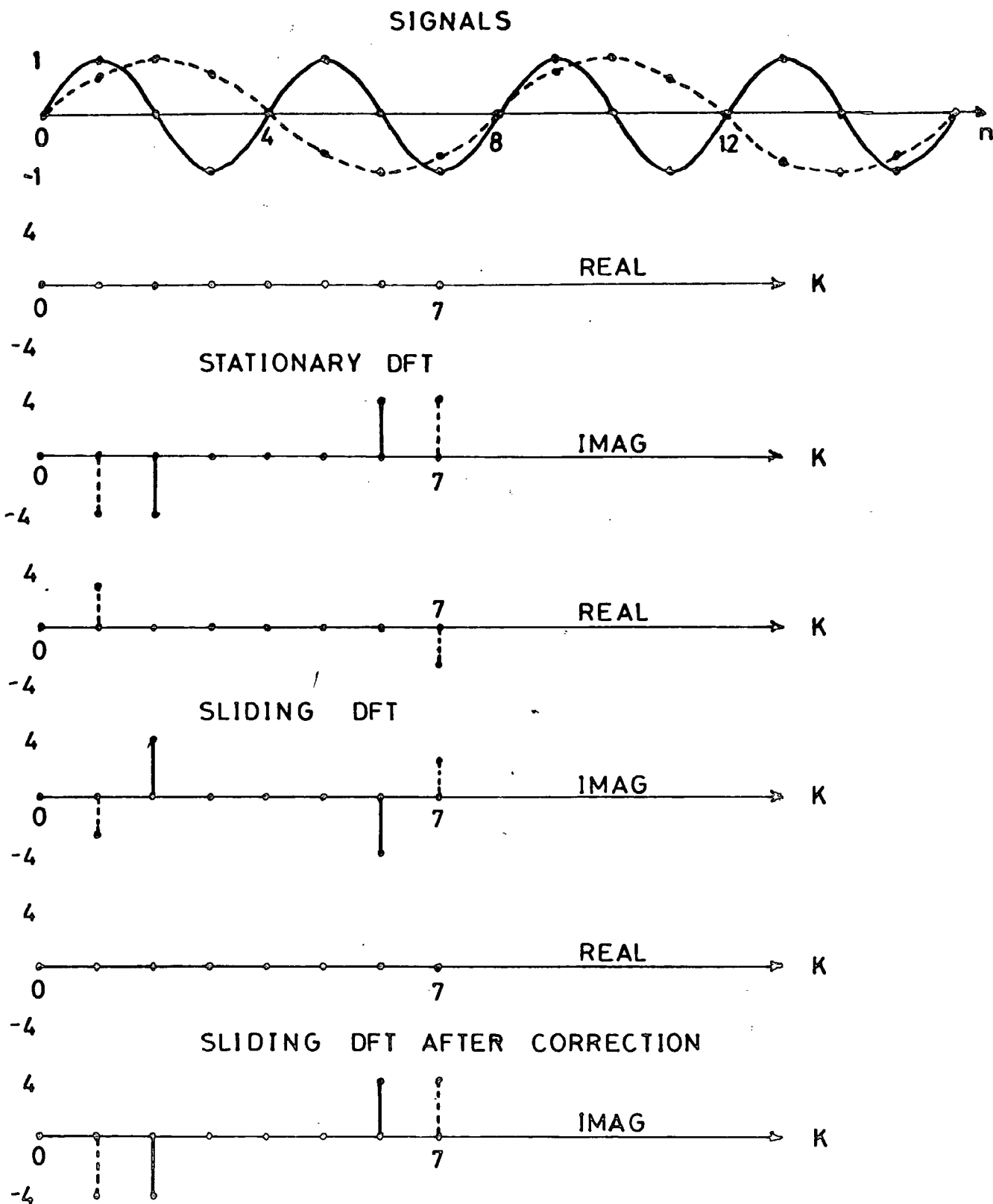


FIGURE 29 SLIDING PHASE CORRECTION.

waveforms given by $e^{-j \cdot \pi k^2/N}$. For the sliding CZT implementation, the overall post-multiplication coefficients must be the product of these chirp waveforms and the phase correction factor. ie.

$$e^{-j \pi k^2/N} \cdot e^{-j 2 \pi k^2/N} = e^{-j 3 \pi k^2/N}$$

2.10.2 Post-multiplier design

A circuit was required which could evaluate the complex multiplication $(a+jb)e^{-j 3 \pi k^2/N}$. The coefficients a and b are the direct and quadrature channels respectively from the transversal filter and are available from the evaluation module as sampled analogue voltages. The cosine and sine chirp waveforms could be stored in PROMs as 8-bit digital words. The transform length was fixed by the filter to be 512 points and each PROM was therefore required to have a 512-byte capacity. It was decided to use Intel 2716 type 2 kbyte EPROMs for two reasons: these devices were readily available at low cost, and a programming facility was already installed in the MSI 6800 development system. Two such EPROMs were programmed with the required chirp waveforms, which were generated using the co-resident BASIC interpreter and converted to offset binary format prior to writing to the device. Four real multiplications and two real additions were required to evaluate the complex multiplication described above. Each multiplication involved finding the product of a sampled bipolar analogue voltage and an 8-bit digital word. A circuit for performing this operation is shown in figure 2.10 and is described as follows:

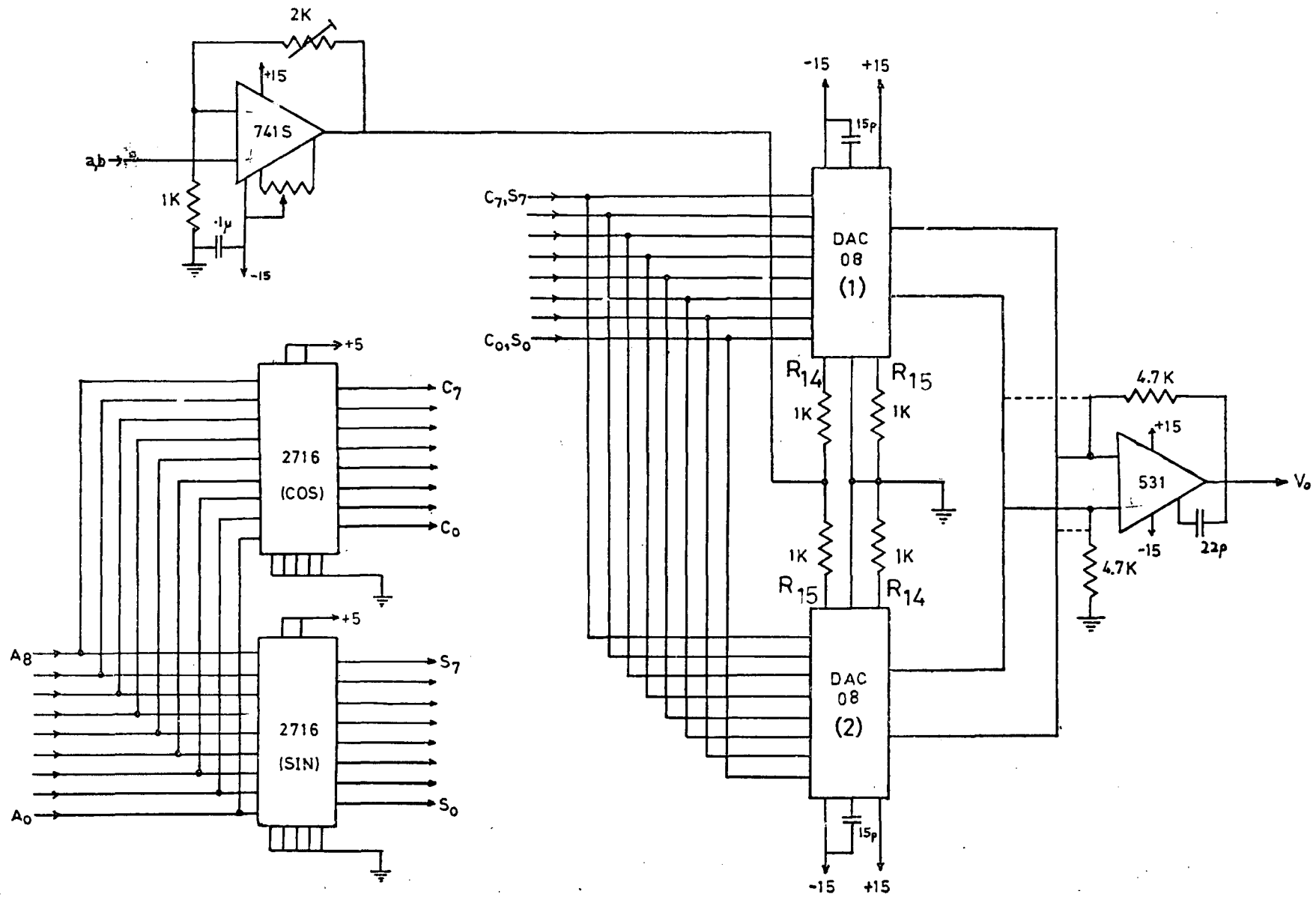


FIGURE 2.10(a). POST-MULTIPLIER UNIT (EPROMs and MDACs).

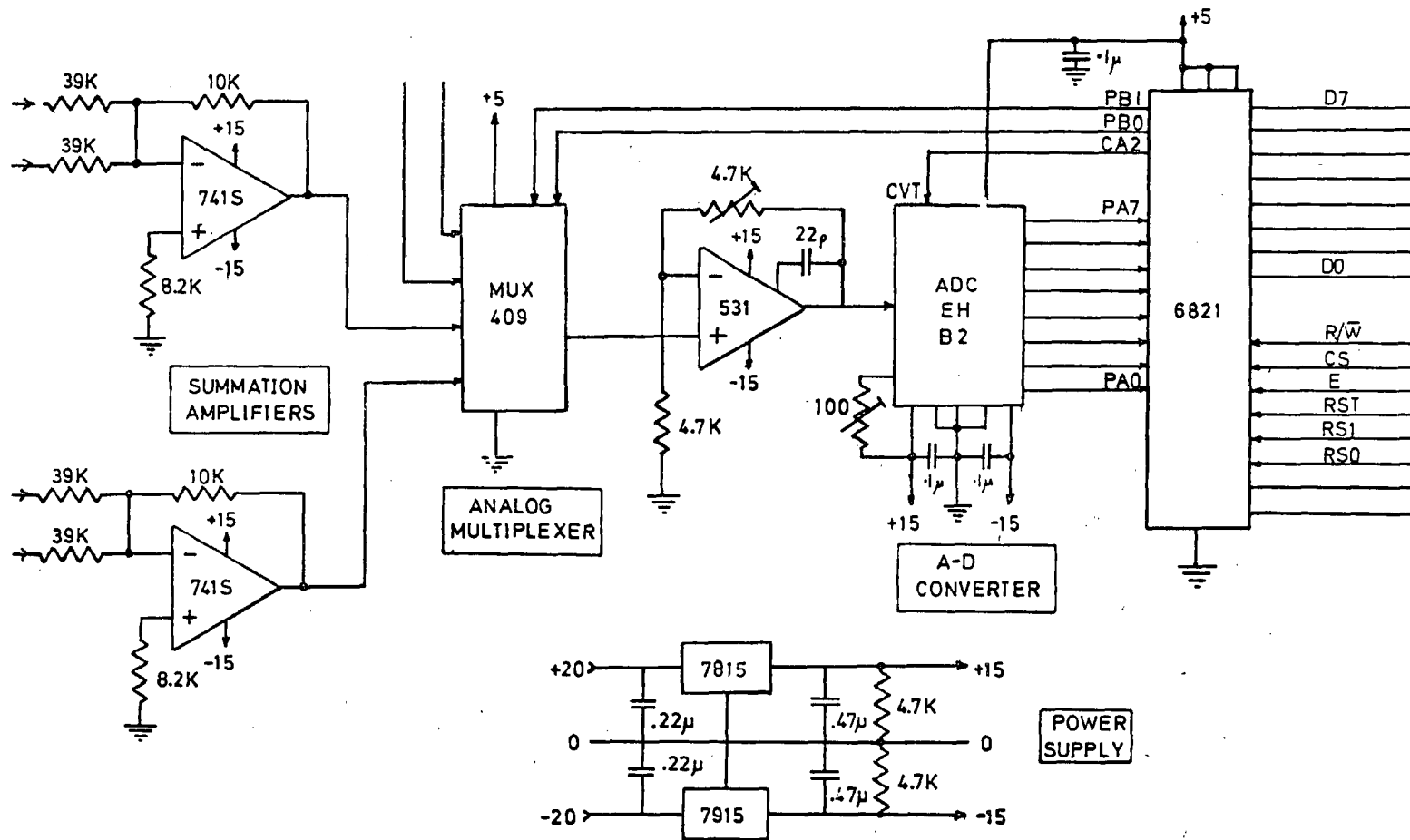


FIGURE 2.10(b). POST-MULTIPLIER UNIT (Summation amps, multiplexer, A-D converter).

The DAC-08 is an inexpensive monolithic digital-to-analogue converter with an 85ns settling time, which provides an output current which is the product of an 8-bit binary word and an input reference current, I_{REF} . The output current, I_o , for a binary input word, W , is given by:

$$I_o = \frac{256}{W} \times I_{REF}$$

Two complementary current outputs are available, which allow the device to be connected in a "symmetrical offset binary" output configuration by converting the currents to voltages and summing using an operational amplifier. Full scale positive and negative voltage outputs are then obtained for input words of \$FF and \$00 respectively. The output voltage is proportional to $(W-128)$, where W is the input word.

Two input configurations are possible to cater for positive or negative reference voltages. For the circuit of figure 2.10(a), current flows into pin 14 of DAC 1 when V_{IN} is positive, but no reference current is available to DAC 2. If V_{IN} is 1V, the reference current is 1mA and the output voltage from the buffer amplifier is $-4.7V \times W$. If V_{IN} is negative, current flows in to pin 14 of DAC 2 and the output voltage is positive. The output voltage is therefore given by

$$V_o = + \frac{4.7 (W-128)}{128} \times V_{IN}$$

Table 2.1 shows the experimentally obtained output voltages for 5 analogue input voltages and 4 digital words presented to the multiplier from a PIA attached to the 6800 development system.

		digital word			
V_i		\$00	\$7F	\$80	\$FF
input voltage	+1V	-4.79V	-0.01V	0.01V	4.79V
	+0.5V	-2.49V	-0.01V	0.01V	2.49V
	0	0	0	0	0
	-0.5V	2.49V	0.01V	-0.01V	-2.49V
	-1V	4.79V	0.01V	-0.01V	-4.79V

Table 2.1

The peak voltages at a and b on the evaluation board were measured for a fixed frequency sinusoidal input and were found to be $\pm 0.5V$. The two non-inverting input buffer amplifiers on the post-multiplier board were arranged to have adjustable gains of between 1 and 3 in order to provide suitable variable references to the multipliers.

The eight output lines from the cosine chirp EPROM were connected to two multiplier circuits to give the products $-a.\cos 3\pi k^2/N$ (V_{TP3}) and $-b.\cos 3\pi k^2/N$ (V_{TP6}) and those from the sine chirp EPROM were connected to multiplier circuits to provide $a.\sin 3\pi k^2/N$ (V_{TP4}) and $-b.\sin 3\pi k^2/N$ (V_{TP5}). The latter product was obtained by using the output buffer amplifier in the non-inverting configuration. The two real products were summed in a wideband summing amplifier to give $X(k)$ (real) and the two imaginary products were summed to provide $X(k)$ (imaginary). Using the component values shown in the circuit diagram, the output voltages were

$$v_o(\text{real}) = -(10/39).(v_{TP3} + v_{TP5})$$

$$v_o(\text{imag}) = -(10/39).(v_{TP4} + v_{TP6})$$

The output voltage range in each case was $\pm 4.8V$.

The summing amplifier outputs were connected to two inputs of a CMOS analogue multiplexer, the output of which was connected via a variable gain buffer amplifier to a high speed ($2\mu\text{s}$ conversion time) bipolar input 8-bit analogue-to-digital converter. Interface to the 6800 microprocessor system was via a single PIA. Four pairs of multiplying D-A converters, as in figure 2.10(a) were necessary to evaluate the four real multiplications required to find the complex product.

The complete circuit was constructed using wirewrap techniques on a 6.5 x 4.4 ins. circuit board. The $\pm 20\text{v}$ supply to the RC5601 evaluation board was reduced to $\pm 15\text{v}$ to supply the post-multiplier by using one 7815 and one 7915 voltage regulator IC. A photograph of the board together with the CCD evaluation board can be seen in photo 2.1.

2.10.3 Test Results

A series of tests were performed on the post-multiplier board and the results compared with theoretical predictions to ensure that the circuit was functioning correctly. The sine and cosine chirp waveforms generated by the unit are shown in photos 2.2 and 2.3. The EPROMs were addressed directly from a PIA installed in the 6800 development system and the digital outputs from the A-D converter were read for a variety of DC analogue inputs and a variety of weighting factors. The results of these tests are shown in table 2.2. a and b are analogue input voltages, ADDR is the EPROM address, COS and SIN are the cosine and sine outputs from the EPROMs. R and I are the real and quadrature output voltages respectively ("ex" and "th" indicate experimental and

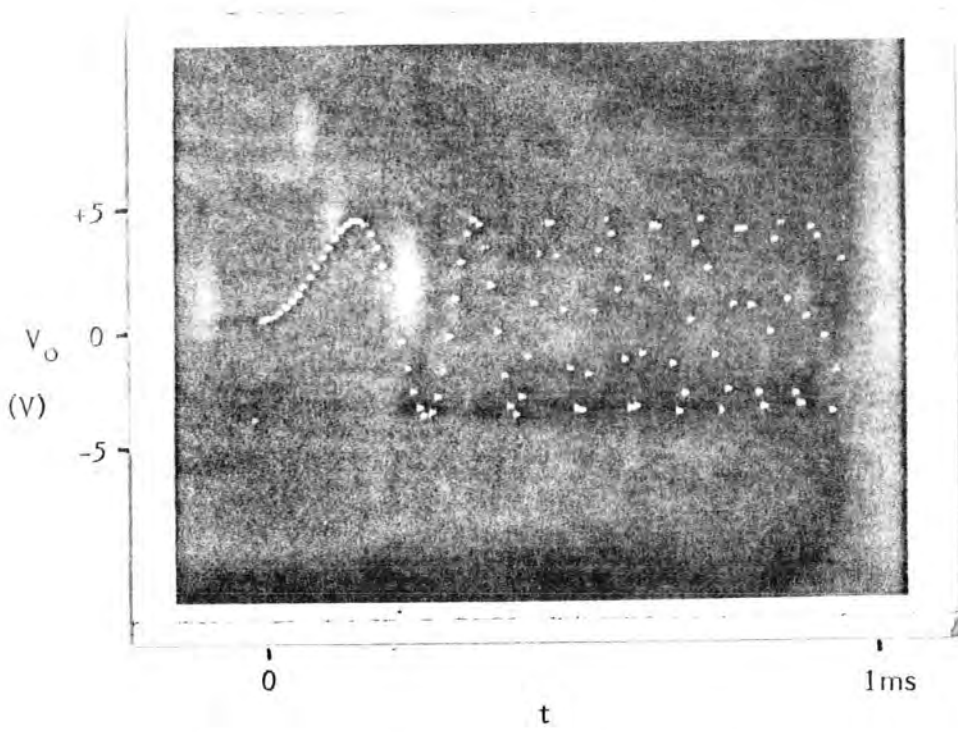


PHOTO 2.2. SINE CHIRP WAVEFORM

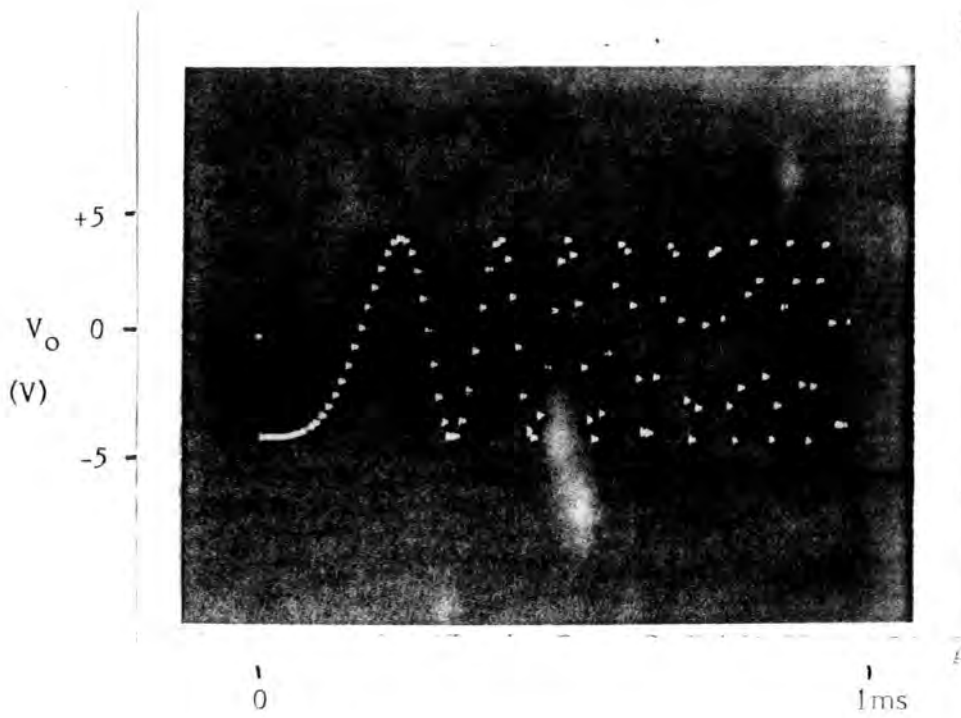


PHOTO 2.3. -COSINE CHIRP WAVEFORM.

theory respectively).

a	b	COS	SIN	R(ex)	R(th)	I(ex)	I(th)	ADDR
-0.5	-0.5	\$40	\$C0	1.18	1.175	0	0	\$08
+0.5	+0.5	\$C0	\$40	1.18	1.175	0	0	\$10
-1.0	-1.0	\$7F	\$FF	2.37	2.35	-2.36	-2.35	\$0E
-1.0	+1.0	\$00	\$FF	0	0	-4.71	-4.70	\$04
+1.0	-1.0	\$FF	\$7F	2.35	2.35	-2.36	-2.35	\$16
-1.0	-0.5	\$FF	\$C0	-1.75	-1.76	-2.73	-2.75	\$17
-0.5	+1.0	\$00	\$00	3.54	3.53	-1.19	-1.18	\$00
-0.5	+0.5	\$FF	\$00	0.01	0	2.36	2.35	\$14

Table 2.2

The CCD evaluation module was interfaced to the microprocessor system and to the post-multiplier unit as illustrated in figure 2.11. The 9 address lines from the on-board counter were connected to the post-multiplier EPROMs, and the "a" and "b" outputs from the filter output buffers were connected to the real and quadrature inputs respectively of the post-multiplier system. The $\overline{T3CL}$ clock line was connected to the CA1 control line of the PIA and the "in sync" input was derived from the CA2 line. The CA2 line was also connected to the "gate enable" input of a pulse generator having a gated output; the latter set to deliver 500 ns wide pulses at a repetition frequency of 20 kHz into the "ext trig" input of the evaluation board. This enabled the system to operate at a sample rate of 20 kHz. The CB2 control line was configured such that a "write to B side data" instruction causes a short pulse to occur immediately after the instruction. This pulse was used to initiate an A-D conversion.

The sinusoidal output from a function generator was connected to a frequency counter and to a comparator, whose output was

clamped with a 5.1V zener diode and fed to the PB7 line of the post-multiplier PIA.

Two programs were written to test the functioning of the post-multiplier unit. The first was not synchronised to the input signal, and the second was synchronised to the zero crossings of the input signal, as detected by the comparator. A description of the operation of the two programs is as follows.

CA2 is initialised to '0', which disables the pulse generator output and holds the on-board 9-bit memory address counter in "reset". A counter within the microprocessor system is then initialised to a value of 512, and the CA2 line is brought to a logic '1', thereby beginning the sampling process. Each transition of $\overline{T3CL}$ causes an interrupt to be generated which is used to decrement the microprocessor counter. After 512 interrupts have been counted, this counter is set to zero and the CCD filter is full (it is a 512-stage device). During the next 256 transitions of $\overline{T3CL}$, the CCD evaluation module outputs frequency samples in the Nyquist band; therefore another 256 interrupts are counted. The real and imaginary outputs from the post-multiplier unit are then read by the microprocessor system using the analogue multiplexer and the A-D converter. The results are stored for further processing.

The sampling frequency was set to 20 kHz and a sinusoidal input of approximately 4.5 kHz was applied to the CCD evaluation module. Sampling was initiated at random intervals and the results were stored in two 256 byte tables in memory. An examination of the results showed that the energy associated with a single

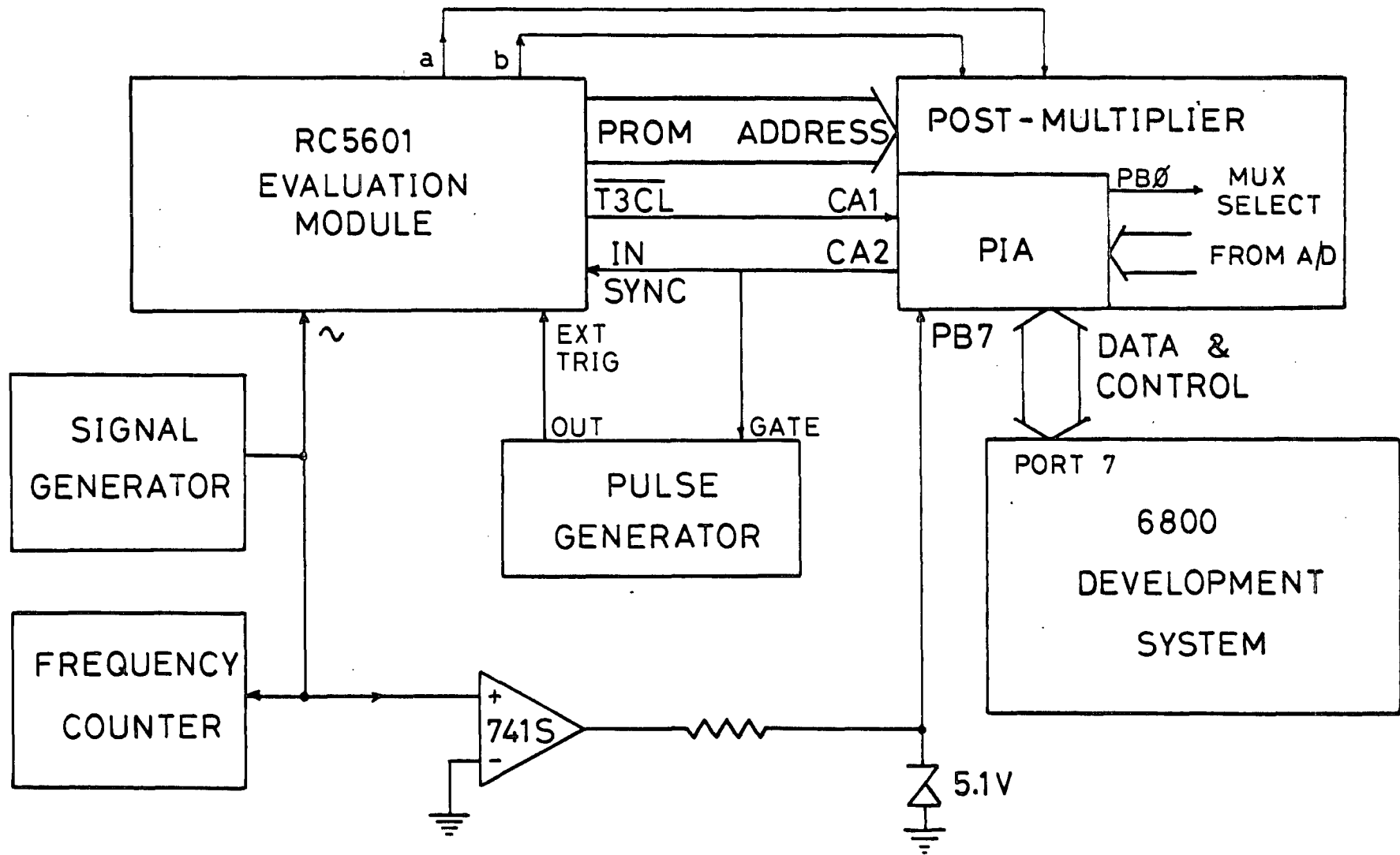


FIGURE 2.11 POST-MULTIPLIER TEST CIRCUIT

frequency carrier tended to be spread over several frequency bins due to the effect of the Hanning window applied to the CCD split electrode weightings.

The experiment was repeated and the resulting coefficients around the frequency of interest were printed on the terminal (in decimal) on each occasion. Figure 2.12(a) shows the results obtained from successive readings of the real and imaginary coefficients. Five pairs of coefficients are printed for each reading; the centre coefficient is that corresponding to the frequency bin of the input signal.

Next, the spectrum analyser was synchronised to the incoming signal by initiating the sampling process only on a positive transition of PB7 (corresponding to a zero-crossing of the input data signal), so as to obtain samples of a sinusoid with zero phase shift. The real and quadrature coefficient magnitudes around the frequency of interest were printed as before, and the experiment was repeated a number of times. A correlation was observed between successive results, as shown in figure 2.12(b).

The differential inputs to the comparator were then reversed, and the experiment was repeated, without varying the magnitude or the frequency of the input signal, to obtain samples of a sinusoid with 180° of phase shift (a negative sinusoid). An examination of the output coefficients around the frequency of interest revealed that the signs of the coefficients were reversed, although the magnitude of the spectrum remained constant. In fact, the coefficients for the first case were predominantly

NOT SYNCHRONISED:

REAL					IMAG				
=====					=====				
-2	-26	-64	-31	-1	-1	-8	-21	-33	0
0	12	43	41	0	-1	-24	-51	-19	0
2	24	53	18	0	0	8	21	24	1
0	-4	9	27	0	0	-24	-65	-40	-2
0	25	63	39	3	0	-4	-9	-7	0
0	-23	-59	-27	0	0	-5	-30	-39	0
0	-9	-6	18	0	0	-22	-64	-45	-2
2	23	63	33	1	0	-3	-9	-4	0
0	2	11	8	0	1	10	23	8	0
0	-7	-10	3	0	0	15	32	54	2
1	13	45	2	0	0	-2	-23	-26	0

FIGURE 2.12(A). TEST OUTPUT DATA AT K=114,115, . . . 118

SYNCHRONISED TO INPUT:

REAL					IMAG				
=====					=====				
0	-4	8	10	0	0	-27	-68	-36	0
0	-5	6	11	0	0	-26	-68	-8	0
0	-4	9	8	0	0	-23	-67	-30	0
0	-4	7	11	0	0	-23	-59	-32	0
0	-6	1	7	0	0	-21	-56	-33	0
0	-3	2	9	0	0	-26	-63	-32	0
0	-3	2	7	0	0	-24	-61	-31	0
0	-5	3	7	0	0	-26	-62	-30	0
0	-4	3	9	0	0	-25	-58	-29	0
0	-4	7	7	0	0	-26	-57	-31	0
0	-1	0	3	0	0	-33	-69	-34	0

FIGURE 2.12(B). TEST OUTPUT DATA AT K=114,115, . . . 118
SAMPLING SYNCHRONISED TO INPUT SIGNAL.

imaginary and negative and for the second case were predominantly imaginary and positive. These results corresponded to the spectrum obtained from samples of a sinusoid and a negative sinusoid respectively, although several unwanted components (with incorrect signs) were also present due to the effect of the Hanning window weighting of the filters.

Several further tests were carried out on the post-multiplier unit to confirm that it was functioning correctly. By using the analogue multiplexer it was possible to select (using software) the real coefficient, the imaginary coefficient (and hence the phase), or the power spectrum, for each frequency bin.

2.11 Multiply-Divide Unit

The hardware multiply-divide unit (34) used in the software FFT implementations is now described. The RCA-CDP1855C is an 8-bit monolithic multiply/ divide unit (MDU) which performs multiplication and division operations on unsigned, binary operators. It is based on a method of multiplying by add and shift right operations and dividing by subtract and shift left operations. The device is structured to permit cascading of identical units to handle operations up to 32 bits.

Two MDU's were cascaded to permit the following operations:

- (i) a 32-bit by 16-bit divide yielding a 16-bit result and a 16-bit remainder.
- (ii) a 16-bit by 16-bit multiplication yielding a 32-bit result

Each MDU has 8 bi-directional tri-state data lines, a read/ $\overline{\text{write}}$ input, a clock input, a reset input, a chip enable input and 3 register select lines. Interface to the MSI 6800 system was provided via I/O port 7 on the system I/O bus. The chip enable signal from the bus was inverted and connected to the CE and RA2 lines on the MDU's. The data lines, the reset line and the R/\overline{W} line from the bus were connected directly to the corresponding pins on the MDU's. RS0 and RS1 were connected to RA0 and RA1 respectively. The addresses of the MDU registers were then as follows:

\$F538	X
\$F539	Z
\$F53A	Y
\$F53B	Status ($R/\overline{W} = 1$)

The circuit diagram of the cascaded MDU system is shown in figure 2.13. Interconnections were made using wirewrap on a 0.1in. matrix circuit board.

For two cascaded MDU's each of the registers X,Y and Z is two bytes wide although each occupies only a single address. Each MDU contains a "sequence counter" which enables registers to be loaded or read sequentially. When the counter matches the chip number (CN1,CN0 lines) the device is selected. For example the first selection of register X for loading loads the most significant MDU, the second selection loads the least significant.

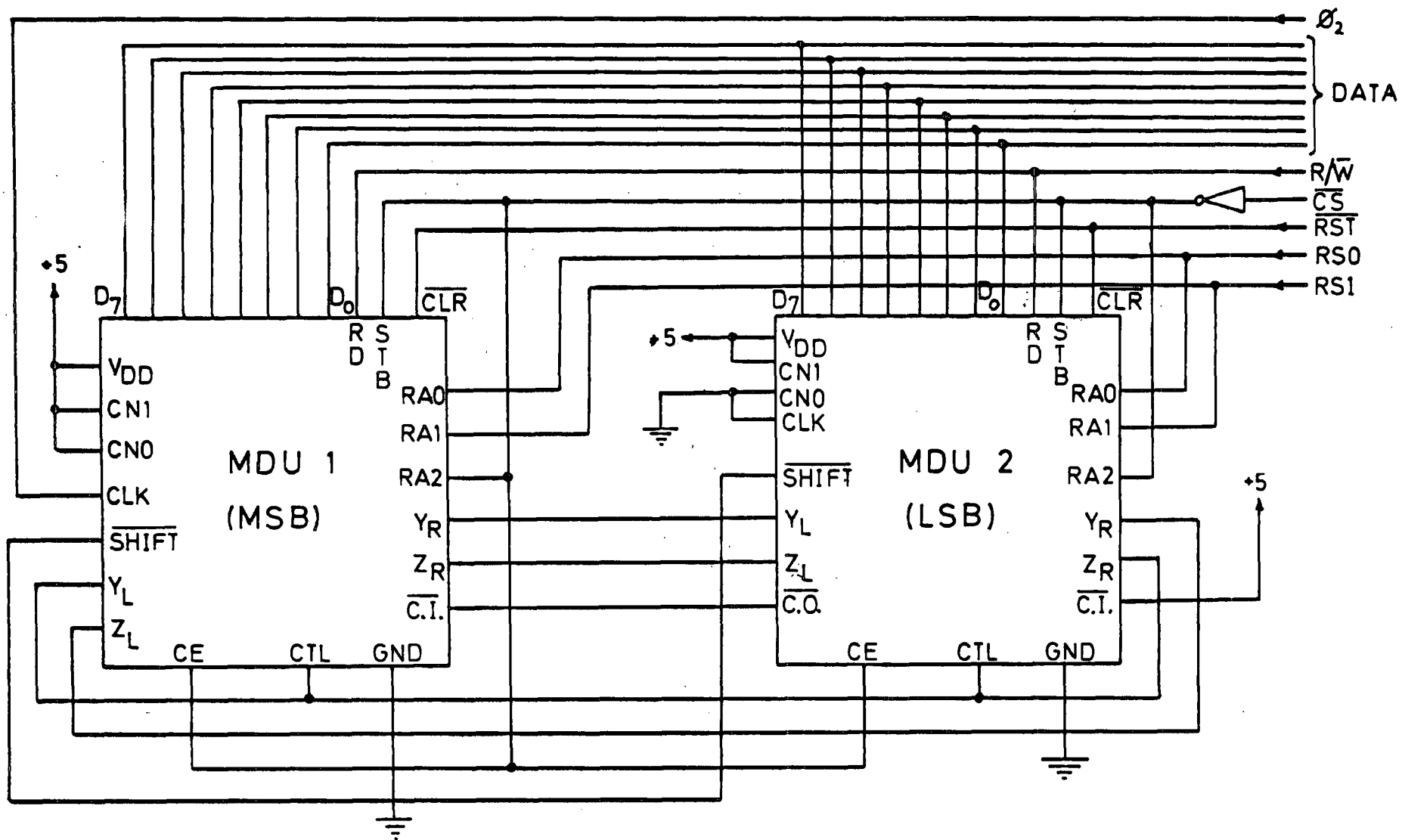


FIGURE 2.13. MULTIPLY-DIVIDE UNIT

To execute a multiply or a divide operation, the data is simply loaded into the appropriate registers, and a control word is written to define the required operation. The results of the operation can then be obtained by reading the appropriate registers.

If N MDU's are cascaded, all operations require $8N+1$ shift pulses. For 2 cascaded devices operating at a frequency of 1 MHz therefore, the time taken for a multiplication or a division is 17 μ s. This is a substantial improvement over a purely software implementation, and was found to provide a considerable improvement in execution time for the Fast Fourier Transform algorithm implementation.

2.12 Comparison of algorithms

The four DFT implementations previously described were compared mainly from point of view of memory requirement and execution time. The first implementation (using a high-level language interpreter) was obviously poor in all respects except for ease of programming and shall not be considered further. The assembly language implementation was quite efficient in terms of memory utilisation and was two orders of magnitude times faster than the BASIC routine. More efficient memory use could have been obtained by computing the butterfly weighting factors only when required; however, this would have led to a considerable increase in execution time. The maximum permissible transform length was 256; if this length had been increased, many single-precision operations would have required double precision, again increasing the required time.

A considerable improvement in execution speed was obtained by using a hardware multiply-divide unit (described in the previous section). This unit was used to (a) multiply the input data sequence by a time window (if required), (b) perform all weighting factor multiplications during the butterfly computations, (c) compute the squares of the Fourier coefficients (if a power spectrum is required) and (d) compute the square root of the sum of the squares of the coefficients using the Newton-Raphson iteration formula (which requires division operations). Table 2.3 below shows figures for execution times for the various algorithms.

N	BASIC	ASSM(1)	ASSM(2)	CZT
8	1.2s	19ms	5.1ms	-
16	2s	54ms	14.2ms	-
32	4.7s	164ms	37.5ms	-
64	10s	429ms	96ms	-
128	22.8s	1.01s	219ms	-
256	48.3s	2.5s	501ms	-
512	105.2s	-	-	5.12ms

Table 2.3

ASSM(1) and ASSM(2) refer to the assembly level FFT routines with and without the hardware multiplier, respectively.

The hardware implementation of the CZT using the charge coupled device was by far the most efficient in terms of execution time. The device itself is primarily for use in power spectrum estimation, and the evaluation board contains peripheral circuitry to evaluate the power spectrum of a fixed-length (512 point) transform. It has been shown that it is possible to use additional circuitry to evaluate the Fourier coefficients, which may

subsequently be used to determine the phase of an input signal.

2.13 Conclusion

The microprocessor chosen for the work in this thesis has been introduced and some preliminary work has been demonstrated on the transfer of data to and from magnetic tapes using a novel software interface. The implementations of efficient algorithms for spectral analysis have been described and investigated. Phase information may be obtained from an analysis of the Fourier coefficients resulting from the Discrete Fourier Transform of an input time signal. This phase information may be utilised to implement a demodulation process in phase modulated data transmission systems for HF radio communications. However, the execution time involved for the algorithm computations may sometimes be excessive and a hardware approach may sometimes be more favourable. An investigation into Fourier transformation using charge coupled devices has been demonstrated which has led to the design and construction of a "post-multiplier" unit for determination of the Fourier coefficients under control of a microprocessor system. This work has formed the basis for a demodulator design for a multitone phase modulated signal for HF data transmission.

The spectral analysis techniques discussed in this chapter have been used extensively in applications described elsewhere in this thesis.

CHAPTER 3

The HF Spectrogram

3.1 Introduction

It has been mentioned that the problems associated with digital data communications over HF radio links generally arise from one of two major causes:

- (i) Multipath propagation caused by multiple reflections from the ionosphere.
- (ii) Noise; originating from both natural and man-made sources.

The phenomenon of multipath propagation can give rise to intersymbol interference if the signal element duration is of insufficient length, and can also cause fading of the received signal resulting from destructive interference. The received signal may also be corrupted by noise, which may be broad- or narrow-band, and which can severely degrade the fidelity of the detected data.

Because of the predominance of analogue traffic over HF radio links, the 3kHz "voice" channel has become accepted as the standard communications channel for such a medium, and requests for frequency allocations are normally granted on condition that the radiated signal shall not occupy a bandwidth which exceeds this figure. This restriction applies even when the information is digital in nature and it therefore becomes the task of the communications engineer to design efficient systems for data links operating over such channels.

It is worthwhile observing the effects of fading and noise on a signal confined to a 3kHz bandwidth in order that the predominant disturbances may be identified. The results may yield clues which could be used to improve system performance by better design, or, if implemented in real time, a number of the available channels may be monitored, and the best chosen for subsequent transmission of data.

3.2 System principles

This chapter describes a system which provides a pictorial display of the time-varying spectral properties of a chosen voice channel. Frequency, on a linear scale from 0 to 3 kHz, is displayed along the y-axis, and time, from $t=0$ to a chosen value, is displayed along the x-axis. The intensity of the display at a particular pair of coordinates indicates the strength of a signal (or noise) which occurred at that particular time and at that particular spot frequency. Because of the digital nature of the system, both time and frequency are quantised, and the display actually consists of a two-dimensional array of points of varying intensity. However, these points can be arranged to be close enough together to provide a quasi-continuous pictorial representation of events occurring within the voice channel over a chosen time interval.

A similar technique was used by R.G.W. Thompson (36,37) to classify HF fading patterns. His analysis was based on observations of a multitone sounding signal from a remote transmitter, and the received data was recorded on an FM tape recorder prior to processing through a spectrum analyser and

mainframe computer. A display was presented on a facsimile receiver coupled to a digital to analogue converter. Disadvantages of the system were cost, complexity, and the time delay necessitated by the pre-recording of the received data. The microprocessor-based system described in this chapter is fast, versatile, and uses a storage oscilloscope to present the spectral information. Such a system could be developed into a small, economical unit suitable for portable operation.

A description of the operation of the system is presented, followed by details of the hardware and software employed in the implementation. Finally, some results are presented, together with suggestions for further development.

3.3 System implementation

Figure 3.1 shows a block diagram of the system. The audio output from an HF receiver is filtered, and then sampled by the A-D converter and microprocessor system. A set of N samples (N is a power of 2, $8 \leq N \leq 256$) is stored in RAM and the Fast Fourier Transform (FFT) algorithm is used to compute the power spectrum of the set which is displayed as a single vertical line on the storage oscilloscope. This line is quantised into $N/2$ discrete points and the intensity of the display at each point is a measure of the magnitude of the power spectrum at the corresponding frequency. The vertical spacing between each point is therefore $2h/N$ cms., where h is the height of the oscilloscope screen. The beam is returned to the x axis and is moved to the right by w/N_t cms., where w is the width of the screen and N_t ($8 \leq N_t \leq 256$) is the number of lines to be plotted. A new set

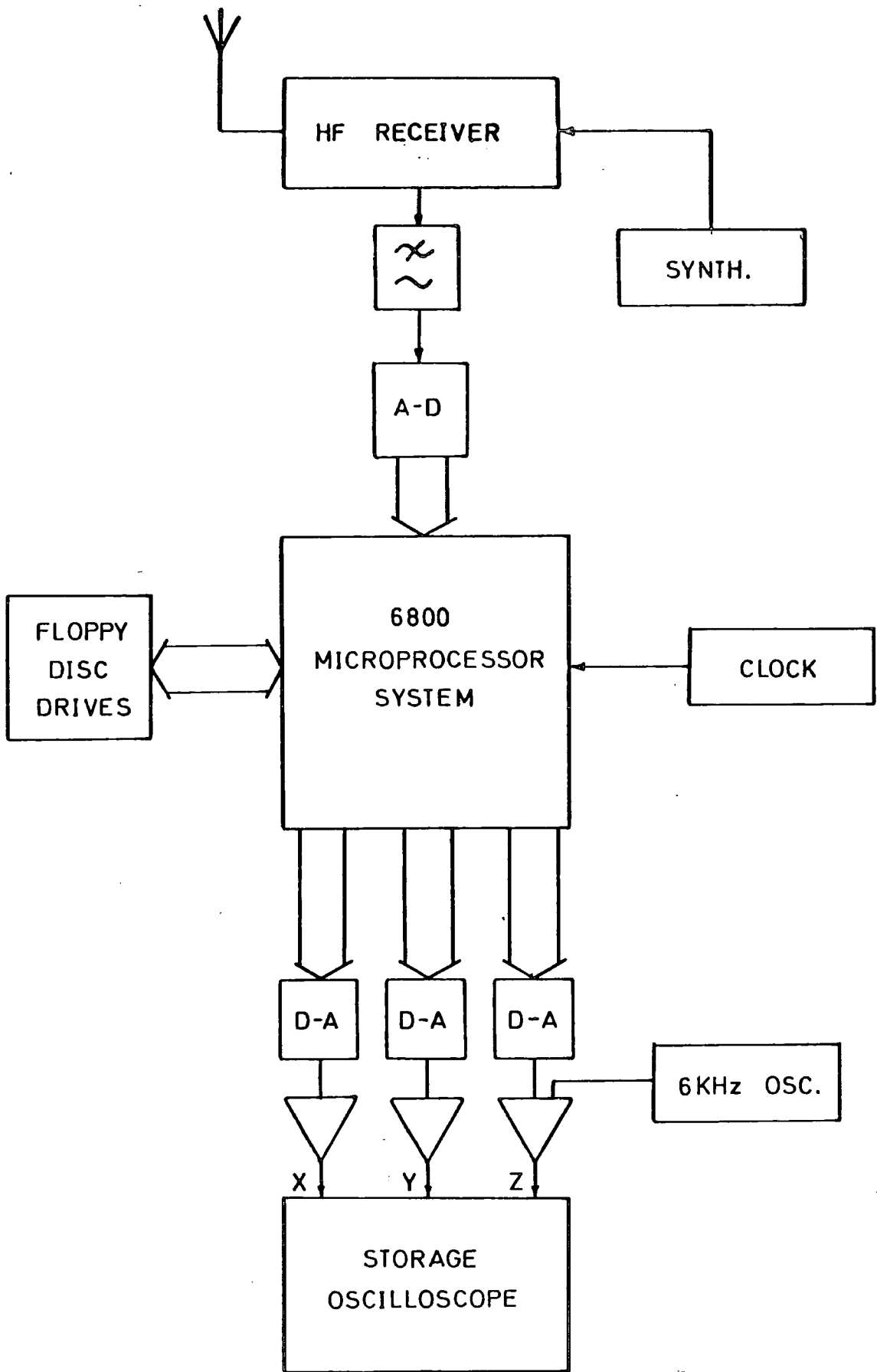


Figure 3.1 HF Spectrogram Block Diagram

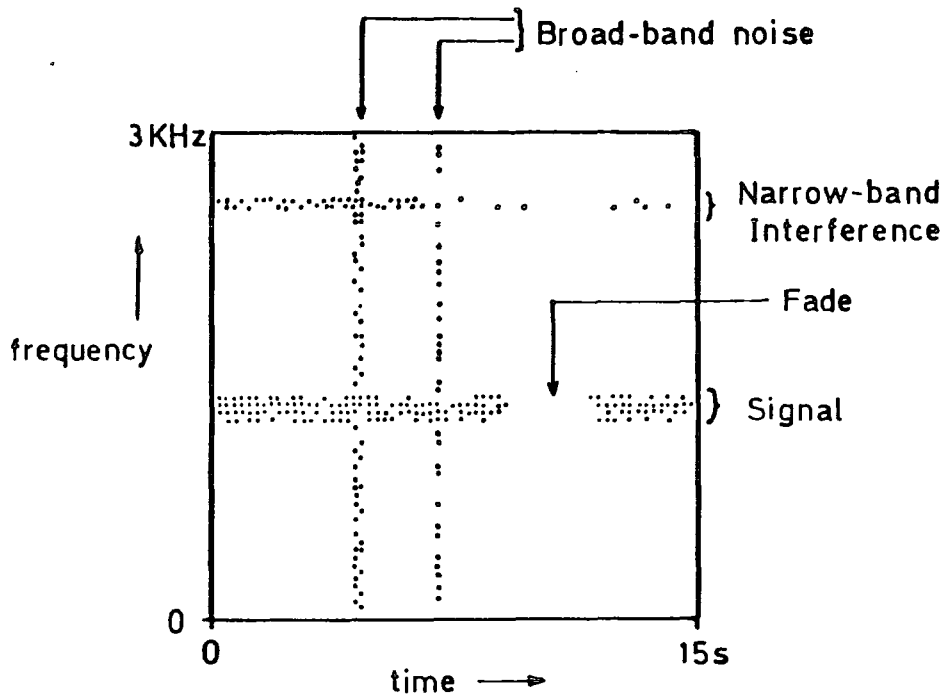


Figure 3.2 Example Display

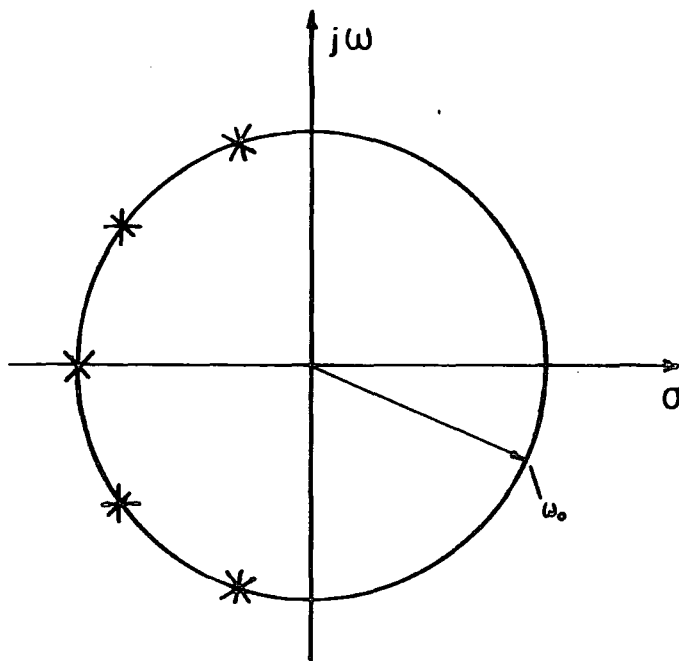


Figure 3.3 S-Plane Representation of Filter

of N samples is then acquired from the receiver and a second line is plotted. This procedure continues until the display is complete. An illustration of the kind of display which might be obtained is shown in figure 3.2.

3.3.1 The lowpass filter

The lowpass filter used in the system was required to have a cutoff frequency of 3 kHz to eliminate out-of-channel noise. The attenuation characteristic was designed to be uniform over the passband, and with a steep rolloff above the cutoff frequency. A two-pole filter would have been sufficient for this application; however a five-pole filter could be realised with little additional complexity and was found to be useful in a later application where a sharp cutoff was essential.

The general form for the squared-magnitude function of a Butterworth filter is given by:

$$\left| H(j\omega) \right|^2 = \frac{1}{1 + \left[\frac{\omega}{\omega_c} \right]^{2n}}$$

where n is the order of the filter and ω_c is the cutoff frequency. The s-plane pole locations which correspond to the denominator polynomial for a fifth-order filter are given by:

$$H(s)H(-s) = \frac{1}{1 + \left[\frac{s}{\omega_c} \right]^{2n}} \quad (n \text{ even}), \quad \frac{1}{1 - \left[\frac{s}{\omega_c} \right]^{2n}} \quad (n \text{ odd})$$

This function has 10 poles equally spaced around a circle of radius ω_c in the s-plane; the n poles to the left of the imaginary axis define the filter (figure 3.3).

The design of the filter was based around the Datel universal hybrid active filter component, model FLT-U2. This dual-in-line package contains four operational amplifiers and a number of passive components. The first three amplifiers are "committed" in the sense that they are internally interconnected with a number of resistors and capacitors in such a way that it is possible to implement a second-order transfer function using the state-variable active filter principle, with the addition of a few external components. The fourth "uncommitted" op amp can be used as a buffer, or to add an independent real pole to the filter characteristics.

For this application, two filter units were cascaded to realise the five-pole Butterworth filter. Each trio of committed amplifiers was used to provide two poles, the uncommitted amplifier in the first unit was used to provide a buffer between the two units and the final, uncommitted, amplifier was used to provide the remaining (real) pole.

The conjugate poles P_1 and P_2 were implemented with the first filter unit.

$$\text{damping factor, } d_1 = \cos \theta_1 = 0.309$$

$$Q_1 = \frac{1}{2d_1} = 1.618$$

Appropriate components were chosen by consulting the manufacturer's data sheet for the FLT-U2. Two sets of tables are provided, one each for the inverting and non-inverting filter configurations. The first filter was operated in the non-inverting

mode and Table 3.1 shows the recommended, calculated, and actual values used in the implementation:

	recommended	calculated	actual
R_{11}	∞	∞	∞
R_{12}	$316k/Q$	195k	180k
R_{13}	$100k/(3.16Q-1)$	24.3k	24k
R_{14}	$5.03 \times 10^7 / f_o$	16.7k	16k
R_{15}	$5.03 \times 10^7 / f_o$	16.7k	16k

Table 3.1.

The uncommitted operational amplifier was wired as a unity-gain non-inverting amplifier to be used as a buffer between the two filter units.

The conjugate poles P_2 and P_4 were implemented using the second filter unit.

$$\text{damping factor, } d_2 = \cos \phi_2 = 0.809$$

$$Q_2 = \frac{1}{2d_2} = 0.618$$

Unit 2 was operated in the inverting mode and Table 3.2 shows the recommended, calculated, and actual component values used in the implementation.

	recommended	calculated	actual
R ₂₁	100k	100k	100k
R ₂₂	∞	∞	∞
R ₂₃	100k/(3.16Q-1)	75k	75k
R ₂₄	5.03 x 10 ⁷ /f _o	16.7k	16k
R ₂₅	5.03 x 10 ⁷ /f _o	16.7k	16k

Table 3.2.

The real pole, P₃, was implemented using the remaining "uncommitted" op amp. A gain of -1 was required, which defined R₇/R₆ = 1. This pole was set to 3kHz by using the capacitor, C, across the feedback resistor R₇.

$$C = \frac{1}{2\pi fR_7}$$

Suitable values were found to be C = 4700pF and R₇ (= R₆) = 11kΩ.

A circuit diagram of the complete filter is shown in figure 3.4. The filter was constructed using a piece of copper strip board, and each of the two cascaded units was tested separately before testing the complete design. The plot of figure 3.5 shows the results obtained for each unit together with the overall response.

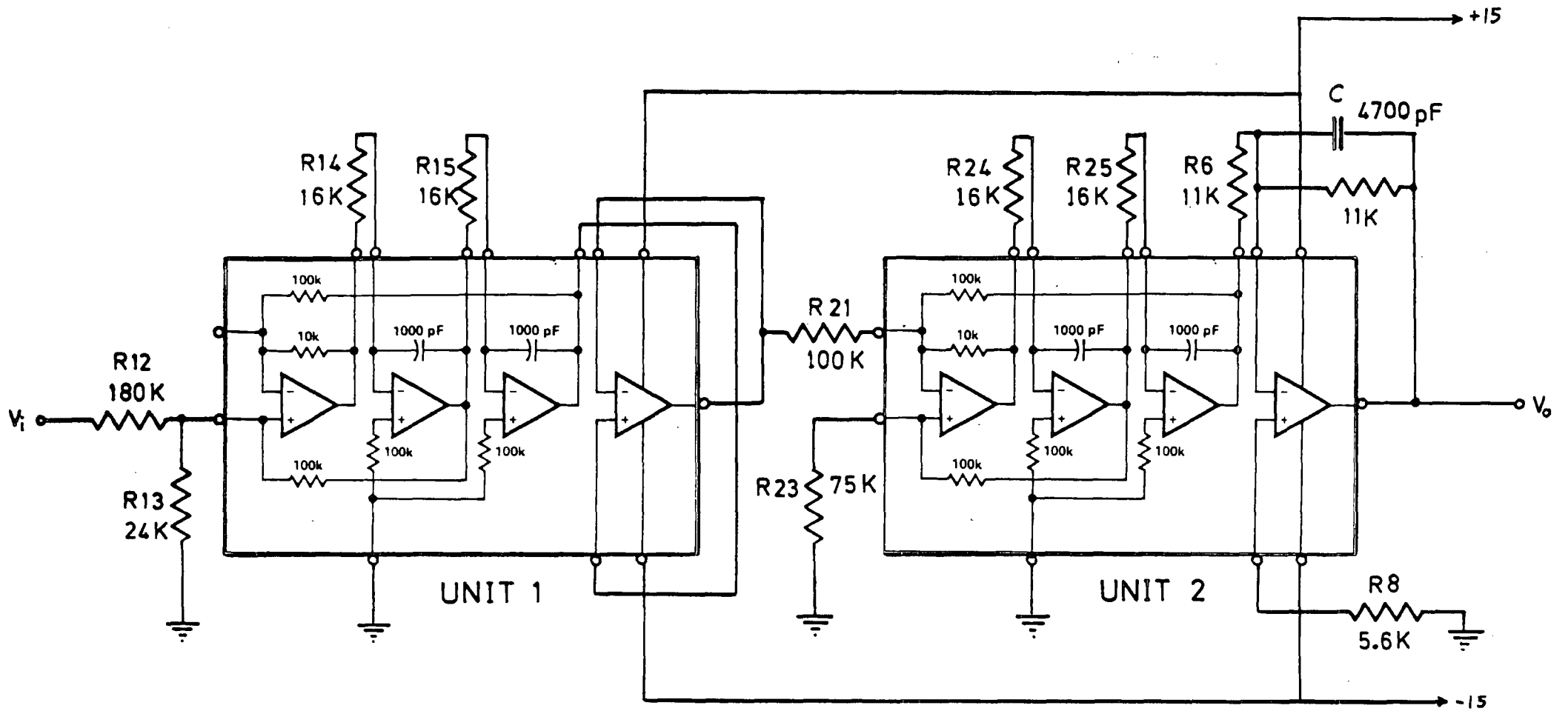


FIGURE 3.4 FILTER CIRCUIT DIAGRAM.

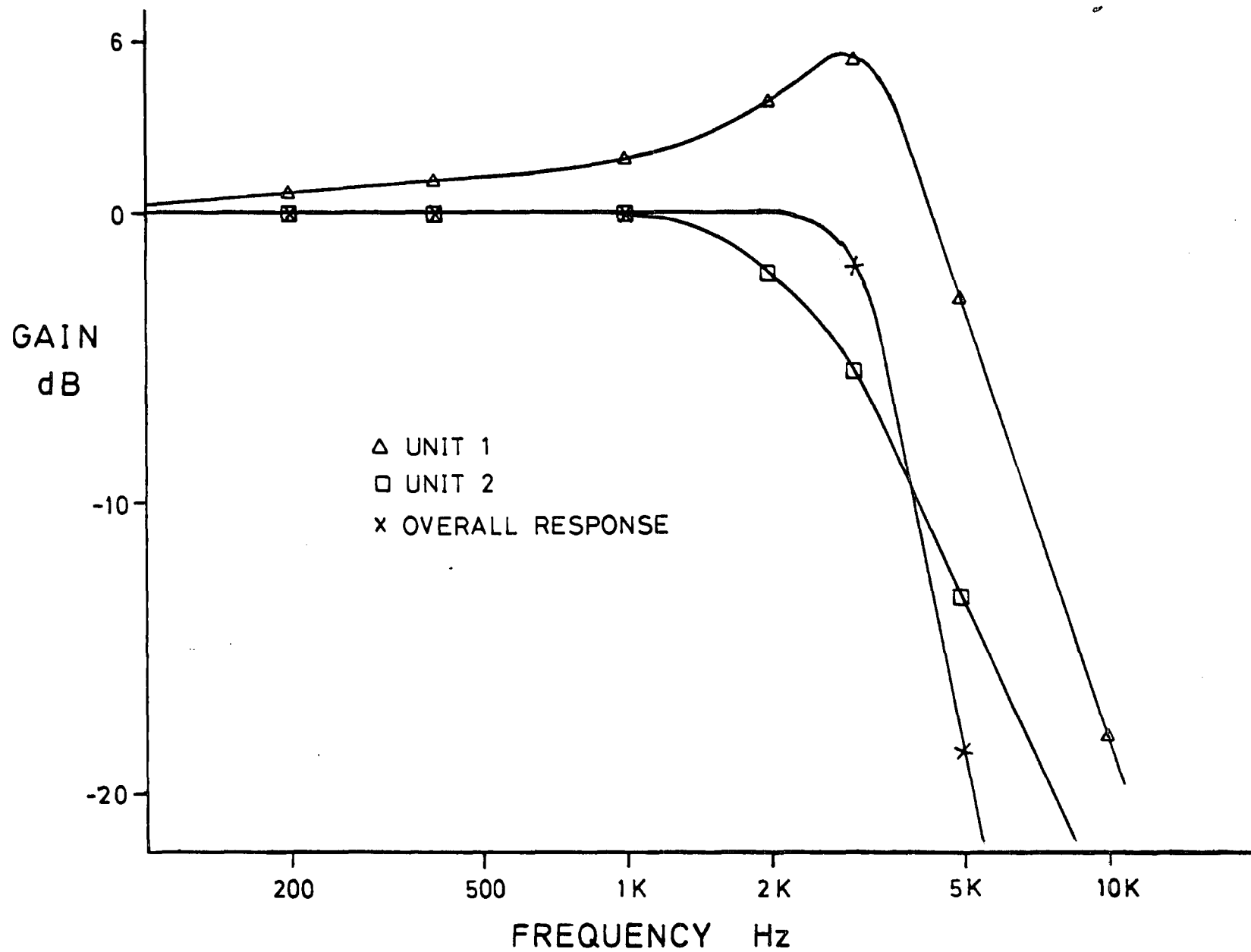


FIGURE 3.5.

FILTER MAGNITUDE - FREQUENCY RESPONSE.

3.3.2 The analogue to digital converter

An 8-bit analogue-to-digital converter was required having a conversion time of 166.6 μ s or less. This figure is the conversion time required for a sampling rate of 6kHz, which is twice the highest frequency component of interest. The conversion system was based around a component (the ZN425E) containing an 8-bit A to D converter together with an 8-bit binary counter, allowing the construction of a successive approximation A to D conversion system with the addition of an external voltage comparator. The clock input to the counter was provided by the ϕ_2 clock of the microprocessor system. For a 1.8MHz clock, this allowed a conversion time of $2^8 / 1.8 \times 10^6 = 142 \mu$ s, which was adequate for the purpose.

Interface to the microprocessor system was provided by a 6821 Peripheral Interface Adapter (PIA) which has two 8-bit peripheral ports which may be software configured for either input or output. The 8-bit digital output from the converter was connected to the 'B' port of this device and the CB2 control line was used to provide the 'convert' command. In order to fully utilise the I/O capabilities of the PIA, it was decided to construct a D-A converter on the same board connected to the 'A' side of the PIA which could then be used to provide the control signal for the oscilloscope Z modulation. A second ZN425E and an operational amplifier were used to construct the D to A system.

A complete circuit diagram of the A-D/ D-A system is shown in figure 3.6. The system was constructed on a board measuring 13.5 x 9 cms. fitted with three 10-way edge connectors, which allowed

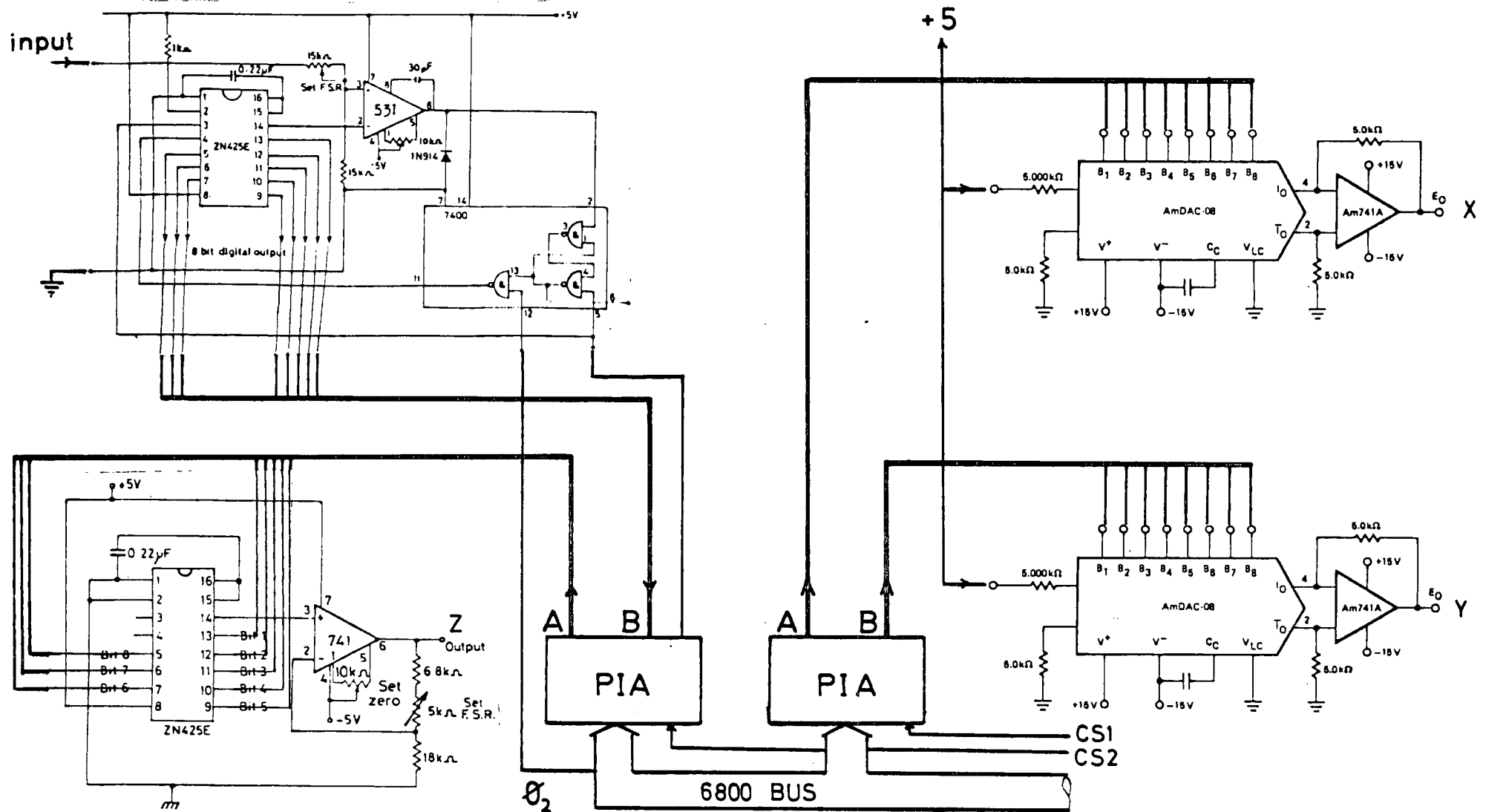


FIGURE 3.6. DIGITAL TO ANALOG / ANALOG TO DIGITAL CONVERSION SYSTEMS

the board to be plugged directly onto the SS-30 I/O bus of the microprocessor system. Printed circuit board techniques were used to define soldering pads for the IC's and all interconnections were made using wirewrap. The A to D converter was tested for linearity and conversion efficiency by plotting digital output against analogue input voltage at a 6kHz sampling rate using software timing. Linearity was found to be better than 0.1%, and a similar figure was obtained from the D to A system, plotting analogue output voltage against digital input.

3.3.3 The digital to analogue converters

Three D to A converters were required for the system; one to provide each of the analogue voltages required to define the coordinates for the display. The system used to define the z-modulation has already been described; two more converters were required for the x and y axes.

The two converters were constructed on the same board and were based around the AMD DAC-08 8-bit D to A converter IC. This component was chosen on grounds of economy (~£3 per chip) and has the added advantage of having a faster settling time (80ns) than many other converters of higher cost. The DAC-08 is a current output converter and requires external resistances to define an analogue output voltage. The internal reference amplifier requires a reference current which can be derived from a stabilised voltage source using an external resistor.

For this application the reference currents were derived from the +5V regulated supply using two $5k\Omega$ resistors. This provided output currents in the range $-1mA \leq I_o \leq 0$. Inverting op-amp

circuits using $5k\Omega$ feedback resistors were used to convert the output currents into voltages in the range $0 \leq V_o \leq 5V$ which were used as the output signals for the x and y axes.

3.3.4 Software

All software for the spectrogram system was composed using the microprocessor assembly language. Assembly-level programming is preferable for real-time applications where speed is of considerable importance. Efficient memory utilisation is possible, which cannot be achieved using a high-level language compiler. Programs were edited and assembled using the microprocessor development system together with the disc-resident editor and mnemonic assembler packages. The software can be divided into 4 parts:

- (1) sampling and windowing
- (2) computation of the Discrete Fourier Transform (DFT)
- (3) power spectrum estimation
- (4) plotting of results

Each of these will now be discussed in turn:

In order to estimate a power spectrum in the range $0 \leq f < 1/T$ Hz, it is necessary (by Nyquist's sampling theorem) to sample a signal at a rate of $2/T$ Hz. In this application the frequency band of interest was in the range $0 \leq f < 3$ kHz, necessitating a sampling rate of 6kHz. The sampling routine employed a software timing loop to define the interval between the acquisition of

samples from the A-D converter. This routine was used to acquire N 8-bit samples for each vertical line on the plot which were collected over a period of $N/(6 \times 10^3)$ seconds ($8 \leq N \leq 256$, $\log_2 N = \text{integer}$). Each sample was converted to its 2's complement representation, scaled down by a factor N, and stored as the most significant byte in a dual byte storage location.

Where a finite length sequence of N samples is used to represent an infinite sequence, the finite sequence is the result of multiplying the infinite sequence by a rectangular 'window' sequence consisting of N samples of unity magnitude. The Fourier Transform of the resulting sequence is then the convolution of the transform of the infinite sequence with the transform of the rectangular window. The latter is of $\sin(x)/x$ form and produces undesirable side lobes in the power spectrum. These side lobes can be reduced by using a window which has unity magnitude at its centre but tapers to zero at each end. One such window is the 'Hamming' window, which was chosen for this application. For the rectangular window the first side lobe is only 13dB down from the main peak, whereas it is 40dB for the Hamming window (see figure 3.7). This extra suppression of the side lobes is achieved at the expense of a slightly wider main lobe.

The Hamming window is defined by the following equation:

$$w(n) = 0.54 - 0.46 \cos \left[\frac{2\pi n}{N-1} \right], \quad 0 \leq n \leq N-1$$

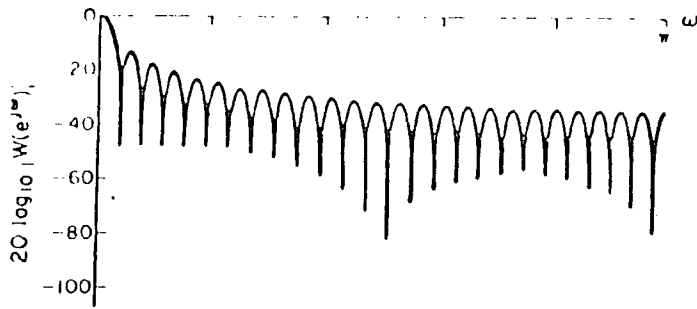


Figure 3.7(a). Fourier Transform of Rectangular Window.

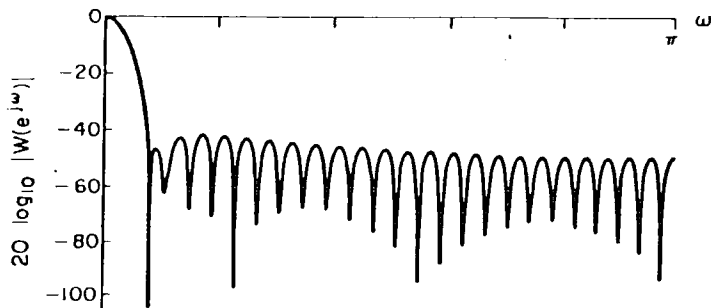


Figure 3.7(b). Fourier Transform of Hamming Window.

It is sometimes referred to as a 'raised-cosine' window. Multiplication of the input sequence by the window was achieved by storing samples of this function in a table in memory and multiplying each of the input samples by the corresponding stored window sample.

Evaluation of the DFT of the windowed input sequence was implemented using the Cooley-Tukey Fast Fourier Transform algorithm. This algorithm requires only $(N/2)\log_2 N$ complex multiplications per transform, as opposed to $4N^2$ multiplications for a direct implementation of the DFT equation. A detailed description of this algorithm and its implementation has been given in chapter 2. The DFT of a sequence $\{x(n)\}$ of N samples is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}, \quad k = 0, 1, \dots, N-1$$

The resulting complex sequence $\{X(k)\}$ was stored in a table of $4N$ bytes; two bytes were allocated for each of the real and imaginary components of the $X(k)$.

The power spectrum of the sequence $\{x(n)\}$ is defined as $\{|X(k)|\}$. The modulus of each of the $X(k)$ was calculated by finding:

$$|X(k)| = \sqrt{X(k)_{RE}^2 + X(k)_{IM}^2}$$

The squares were computed using a 16 by 16 bit multiplication routine (using Booth's algorithm (16)) and the square roots were estimated using the Newton-Raphson recursion formula.

3.4 Results

Vertical calibration of the spectrogram was checked by applying sine waves of different frequencies to the system input and noting the vertical positions of the horizontal lines produced on the display. Each of the eight 1cm divisions on the y-axis represented 375Hz on a linear frequency scale. Gains of the x and y display amplifiers were adjusted to allow the display to fill the storage oscilloscope screen. A DC level produced a horizontal line across the bottom of the screen; a 3kHz sinusoidal input produced a horizontal line across the top. Figure 3.8 shows the display obtained with a 1.5kHz, 5v pk-pk sinusoid with a 3v superimposed DC level.

The audio output from a Racal RA17 HF receiver was filtered and used as the input to the spectrogram system. Unfortunately the receiver was found to have a slightly unstable BFO, which tended to spread the signal in the frequency domain. For all of the following results time is quantised along the x axis into 2^7 points, and frequency is quantised along the y axis into 2^6 points. This produces a display of 8192 discrete points in time-frequency space. The delay loop parameter was adjusted to produce a complete frame in 15 seconds.

Figure 3.9 shows the display obtained while monitoring a high-speed (~ 40 wpm) morse code transmitter centred on approximately 4.7MHz. Some intermittent narrow-band noise can be seen at the high frequency end of the channel (ie. at the very top of the display). Fades of up to one second were observed which are indicated by gaps in the displayed signal. Occasional

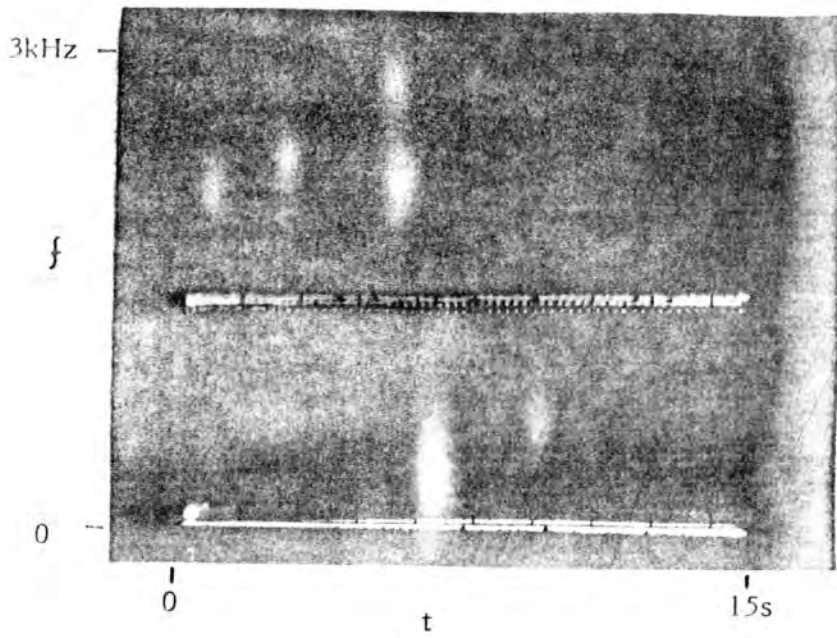


FIGURE 3.8

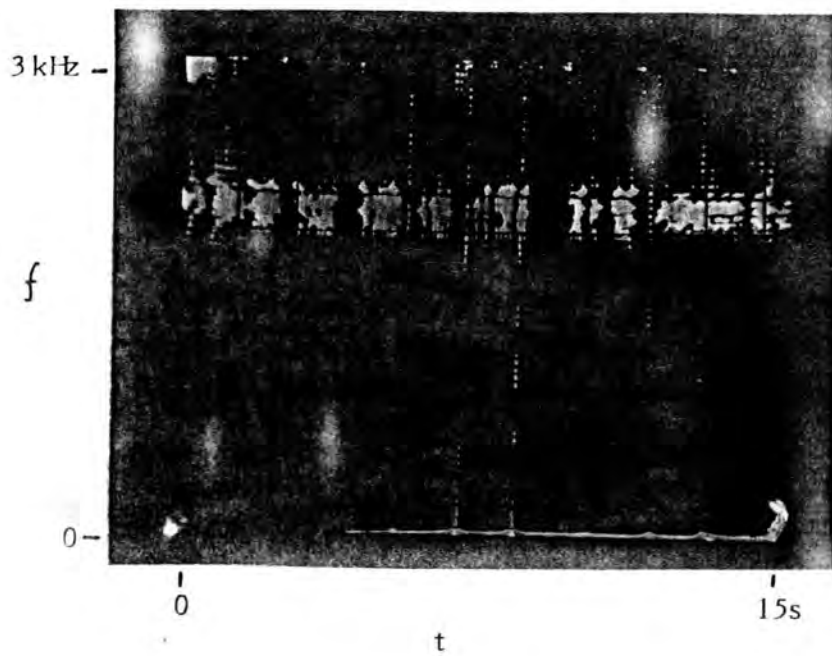


FIGURE 3.9

wide-band noise bursts are displayed as vertical lines traversing the whole channel.

Figure 3.10 is the display obtained while monitoring a voice channel containing two high-speed (~40 wpm) CW transmitters both of which were of equal average strength and separated in frequency by approximately 1kHz. It was found difficult to audibly decode either signal using a 3kHz receiver bandwidth. The two signals can be clearly seen on the spectrogram display. Frequency selective fades were observed audibly and can be seen on the display; ie. the horizontal positions of a gap in one signal (indicating a fadeout) do not always correspond to gaps in the other. A few broad-band noise spikes are again evident as thin vertical lines across the display.

The next two displays were the result of monitoring two 2-tone FSK signals of different data rates. The first (figure 3.11) shows the display resulting from observation of an estimated 150bps signal. The two tones can be clearly seen; short fast fades and bursts of noise are visible and were confirmed by audible monitoring of the receiver output. The display of figure 3.12 illustrates a 2-tone FSK signal of a higher data rate, estimated at 300bps. The frequency spreading is greater than for the slower-rate signal, although in both cases the spread was larger than expected, mainly due to the BFO instability. Frequency selective fading is evident from the virtual disappearance (for the first 4 seconds of the display) of the higher frequency tone. Broad-band noise was more severe than for the previous signal.

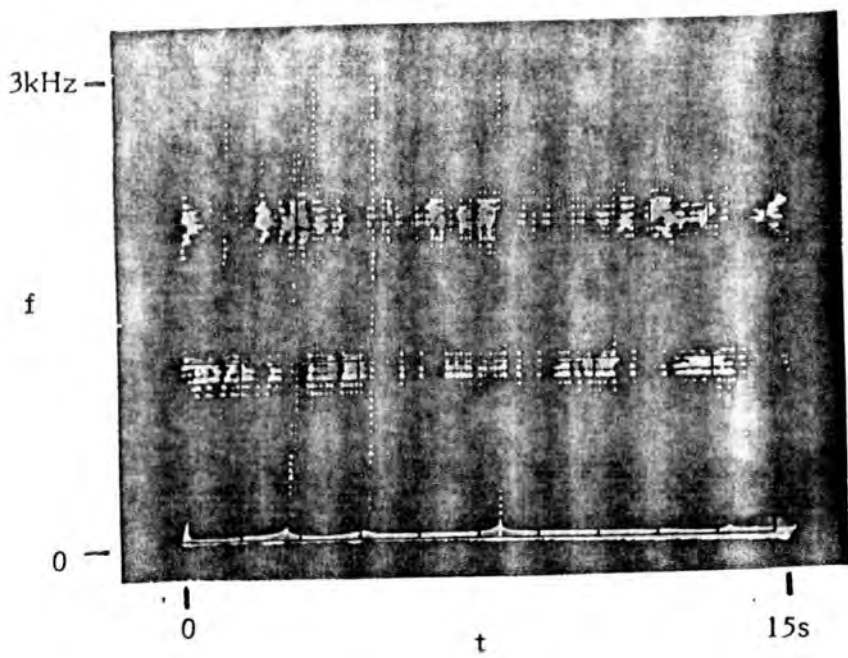


FIGURE 3.10

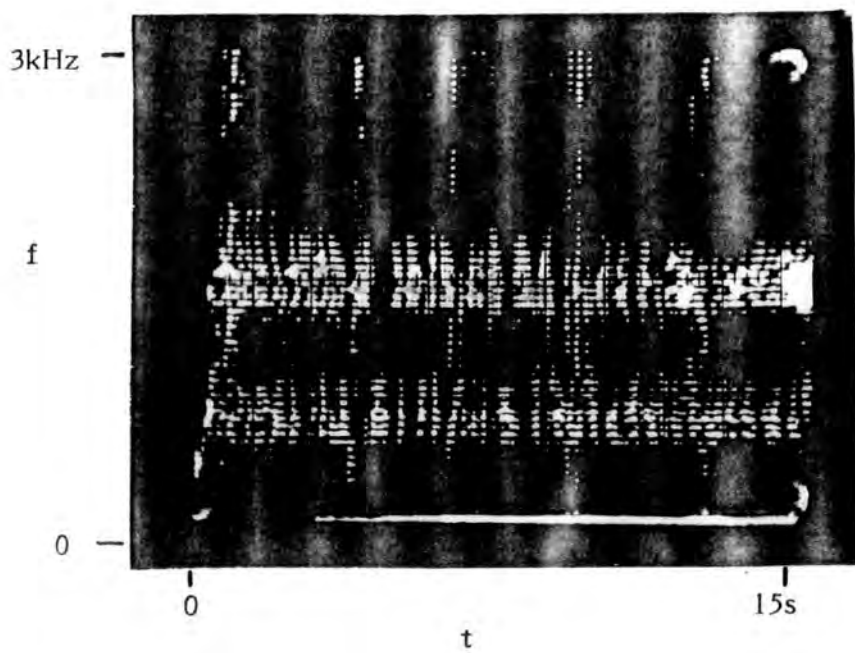


FIGURE 3.11

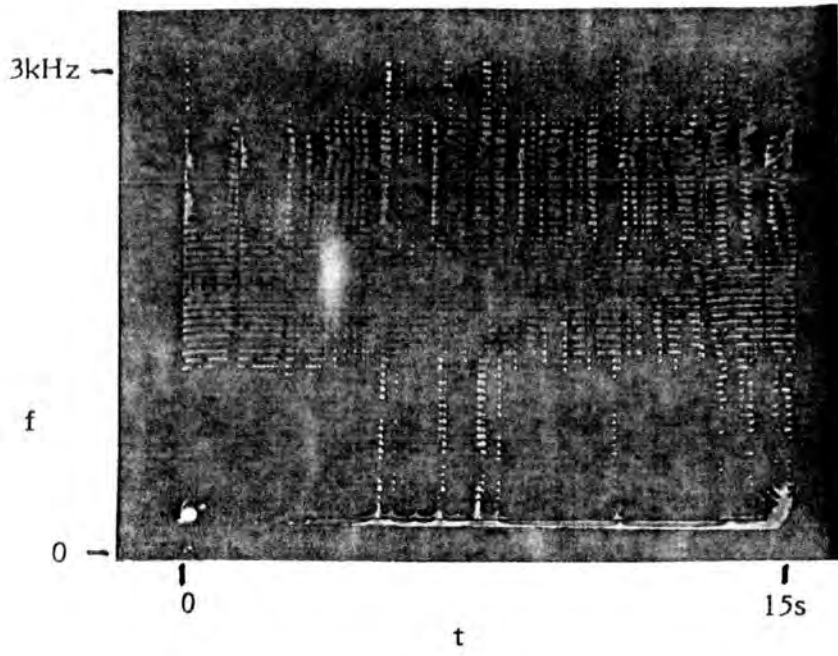


FIGURE 3.12

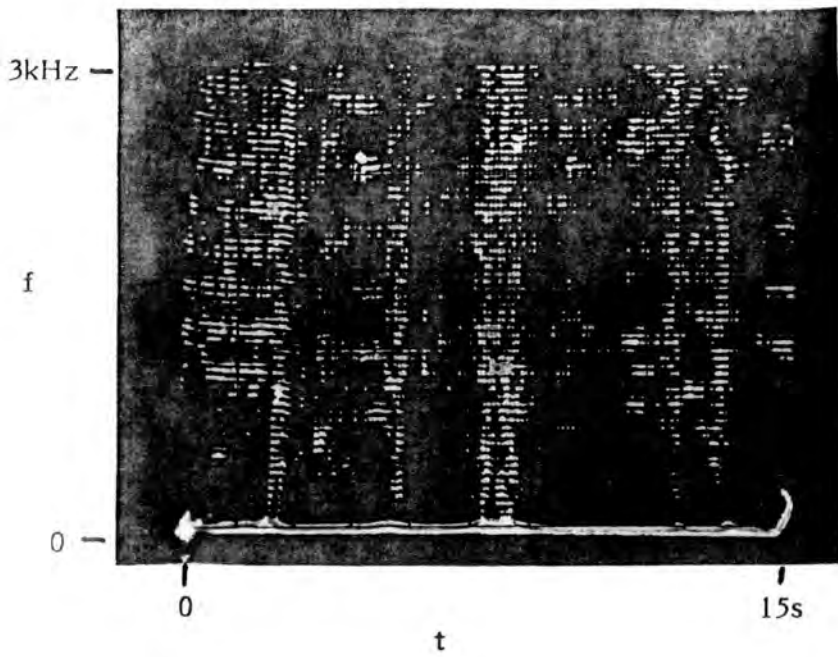


FIGURE 3.13

Figure 3.13 shows the spectrogram obtained from an AM broadcasting station centred on 7.42MHz observed during the early evening. The station was broadcasting orchestral music and was therefore broad band. Very deep frequency-flat fades were observed which can be identified by the large vacant areas in the display.

3.5 Conclusion

This chapter has described an economical microprocessor-based system for evaluation of an HF radio voice channel. The pictorial representation of the time-varying spectral properties of the channel enables the predominant disturbances to be identified. Broad-band noise bursts and narrow-band interference may be observed and fades on a known signal may be identified as frequency-flat or frequency-selective. The microprocessor implementation allows the system to present information in real time, which an operator may use to assess the suitability of a radio channel for data transmission.

Use of the spectrogram over a period of weeks indicated that the predominant disturbances tended to be narrow-band interference from other users of the channel. In most cases the spectral occupancy was limited to less than 20% of the overall bandwidth; this is examined in more detail in chapter 6. The results from the spectrogram would tend to indicate that benefit may be obtained from either (a) dynamic channel selection, where a change of channel frequency is made, or (b) dynamic in-band frequency allocation, where the spectrum of the transmitted signal is arranged to occupy the interference-free regions of the channel.

Frequency-selective and frequency-flat fades were observed from spectrograms of known signals. Frequency selective fades were observed to traverse the channel completely, usually in a short time ($< 2s$). Broad band noise bursts encompassing the whole of the channel spectrum were frequently observed. A data communications system for use over HF channels should therefore provide protection against long-term narrow-band interference phenomena and short-term broad-band fading and noise.

CHAPTER 4 The Slave Processor System

4.1 Introduction

It has been mentioned in the introductory chapter that the data processing tasks for some applications described in this thesis exceeded the capability of a single microprocessor unit. An example is the parallel HF modem transmitter, to be discussed in chapter 7, in which a modulated multitone waveform is to be generated digitally while simultaneously encoding and interleaving incoming data. The modem receiver is required to demodulate the received signal waveform and to decode the demodulated data. The serial processing capability of a single microprocessor is not sufficient to complete the required programming tasks in the available time.

As a result of these requirements, a small, self-contained microcomputer unit was developed which could perform a proportion of the processing tasks required by the overall system. This unit is connected into the system in a "master-slave" configuration such that the central (or "master") processor can assign tasks to one or more local (or "slave") processors. The slave processors then operate transparently to the central processor freeing the latter to perform other system tasks, returning at a later stage to restart or reallocate tasks as necessary. This form of distributed processing is useful where many of the system operations are repetitive, and can be implemented in microprocessor systems at low cost and with a considerable increase in overall processing power. This chapter describes a system implemented using the Motorola 6800

microprocessor which has subsequently been used in the HF transmitting and receiving equipment to be described in later chapters.

The task to be performed by the slave unit is loaded by the master into a localised 1 kbyte of RAM to which only the master and the slave concerned have access. Parameters to be processed by the task are also transferred into the slave processor's RAM and the task is initiated by a reset sequence on the slave processor which is under control of the master. Several slaves may be initialised by the master in this way, which are later checked using status bytes located within the RAM area to ascertain that the allocated tasks have been successfully completed. The results of a processed task are extracted from the slave memory and the task restarted using a different set of parameters, or a new task may be allocated. Because of the way in which the hardware is configured it is also possible for the master to dynamically access the slave memory without disturbing the flow of execution of the slave program. This can be useful when the slave is performing a task which is to be executed continuously, such as the generation of a voltage waveform in real time.

As an example of an application for such a system, suppose it is required to generate a sinusoidal waveform of which the frequency, amplitude and phase may be varied. The frequency parameter may specify the length of the steps to be taken through the lookup table so that if every sample is selected the frequency is f , if every 2nd sample is taken (step length = 2) it

is $2f$ and so on. The samples may be multiplied by a factor, A , to determine the amplitude, and the phase can be modified by specifying the starting point in the table. The selected sample values are converted into a real waveform by addressing a parallel interface connected between the slave processor unit and a digital-to-analogue converter. This is one example of many applications for such a system. Some of the system requirements and principles of operation are now discussed, followed by a description of the implementation.

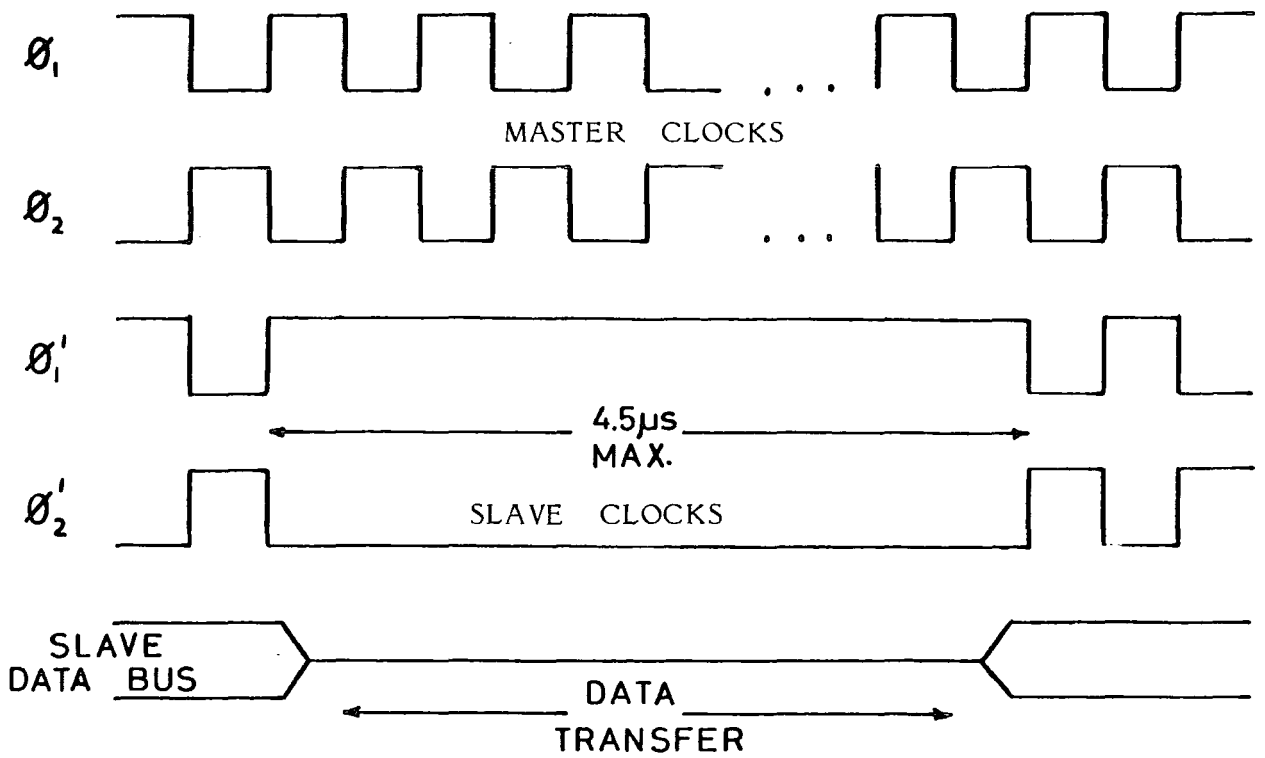
4.2 Principles of operation

The master-slave interface requires that the slave processor unit appears to the master as a continuous area of memory which may be read from, or written to, by the master processor regardless of the operation of the slave. Another requirement is that the master may have access to the more important slave processor control lines. It was decided that this may be most easily accomplished by assigning the lowest address of the slave memory area to a write-only control latch which is available only to the master unit. In this way the reset and interrupt sequences may be controlled by the master.

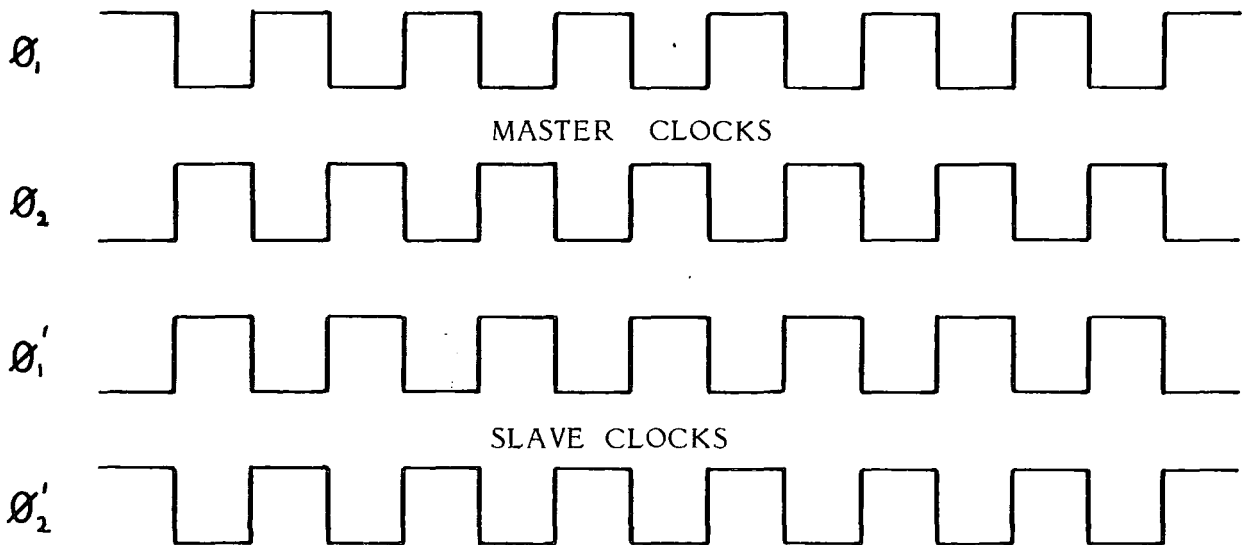
The requirement that the master and slave processors may attempt to simultaneously access a common area of memory may lead to conflicts when both are attempting to access the same byte. Possible ways of resolving address conflicts may be found by examining the requirements for the 6800 microprocessor clocks: A biphasic clock of frequency not greater than 2.0MHz must be provided in which the two phases are non-overlapping. The phases

are designated \emptyset_1 and \emptyset_2 and are used to synchronise all data transfers to and from the microprocessor. The processor sets up an address during \emptyset_1 which becomes stable during the first half of \emptyset_1 and is stable throughout \emptyset_2 . Data transfer (in a direction determined by the state of the read-write line) occurs during the fall of \emptyset_2 when the byte of data on the data bus is latched into the microprocessor or into memory. In the normal system configuration recommended by Motorola (38), the memory is allowed at least half of \emptyset_1 and all of \emptyset_2 in which to respond. Two possible methods of arranging the clock signals to avoid addressing conflicts in the master-slave system are shown in figure 4.1 and are described in the following two paragraphs.

If a conflict is to occur, it will begin during \emptyset_1 , when an address is set up which is inside the slave memory area. This may be avoided by suspending execution of the slave processor program until access by the master is complete. This can be achieved by holding, or "stretching" the clock line to the slave processor while at the same time removing the slave from the busses by multiplexing the address lines and placing the data lines in high-impedance (tri-state) mode. However, because the internal registers of the 6800 CPU are dynamic, the clock may be stretched only to an upper limit of 4.5 μ s, beyond which destruction of internally held data may occur (39). In the situation where the slave RAM is being continually accessed by the master processor this limit may inadvertently be exceeded. It is for this reason that the following alternative method for avoiding conflicts was adopted in the final system.



(a) Clock Stretching



(b) Antiphase Operation

FIGURE 4.1 ADDRESS CONFLICT AVOIDANCE

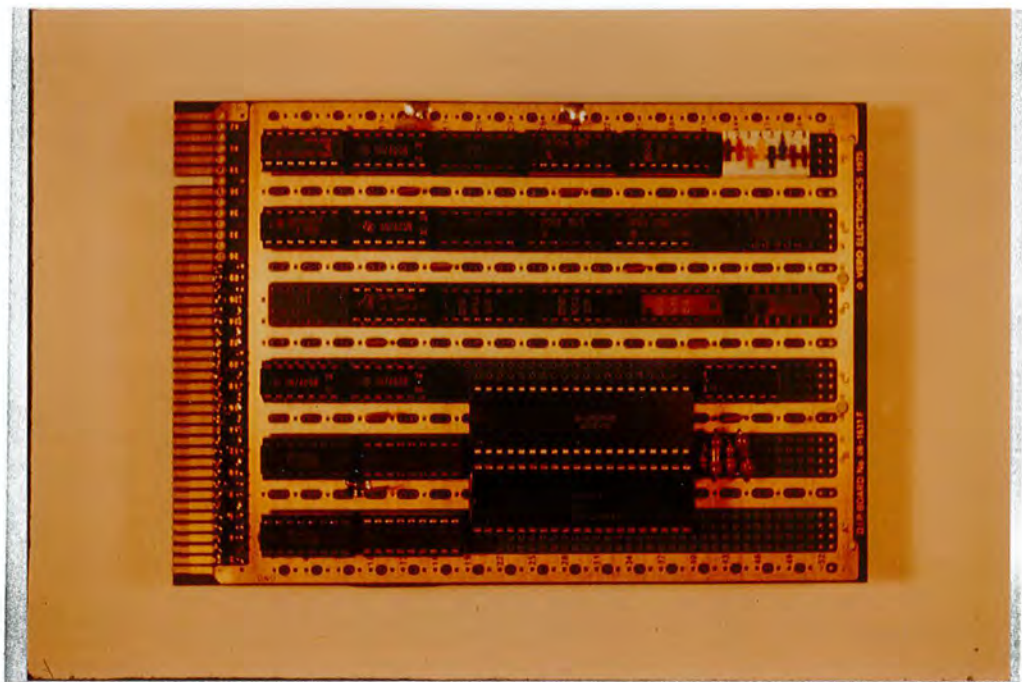


PHOTO 4.1(a). SLAVE PROCESSOR PROTOTYPE BOARD (TOP).

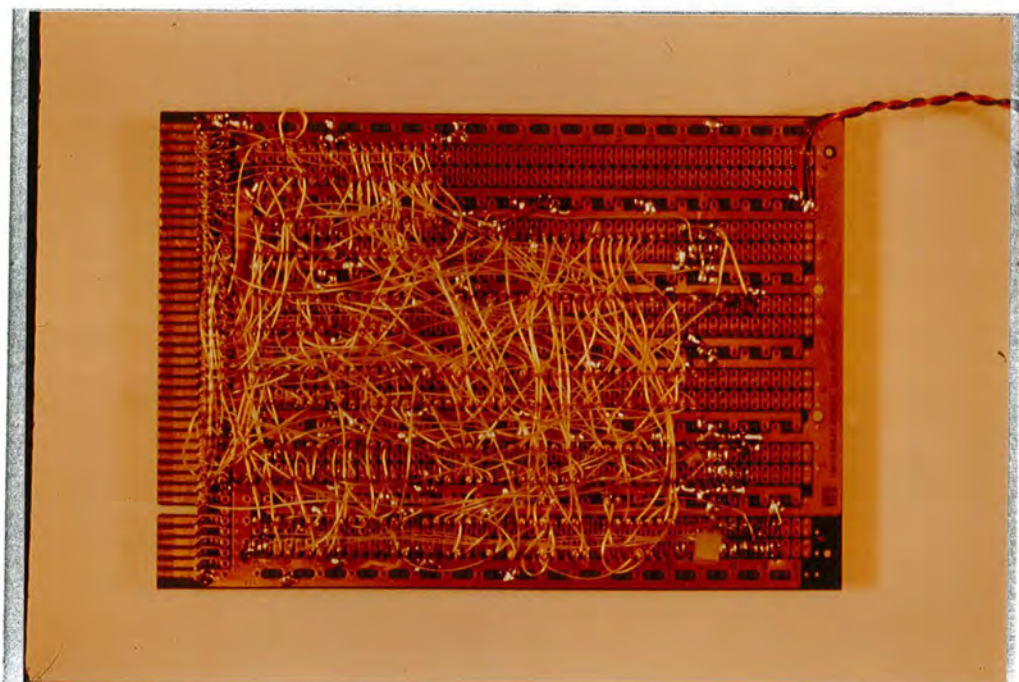


PHOTO 4.1(b). SLAVE PROCESSOR PROTOTYPE BOARD
(UNDERSIDE).

By running the two processors in antiphase, the slave processor is in ϕ_2 when the master is in ϕ_1 and vice versa. Furthermore, the address bus is multiplexed so that the address lines corresponding to whichever processor is in ϕ_2 are always connected to the slave RAM. The master may access the RAM provided the block is selected by decoding the high order address lines of the master processor address bus. Memory access is permitted only during ϕ_2 of either processor, which places an upper limit on the memory access time of one half of the clock period, since only ϕ_2 is available for memory address set-up. Nevertheless, static RAM with a sufficiently short access time is available at reasonable prices and this latter method for avoiding address conflicts was chosen in preference to clock stretching.

This completes the description of the operating principles of the master-slave configuration, the constituent components of which are now discussed in more detail.

4.3 Implementation

Implementation of the slave processor system is outlined in the block diagram of figure 4.2 in which the various functions are grouped into a set of distinct units:

- (1) block select logic
- (2) address line multiplexers
- (3) data tri-state buffers and enable logic
- (4) control latch and enable logic
- (5) clock drivers
- (6) memory and enable logic

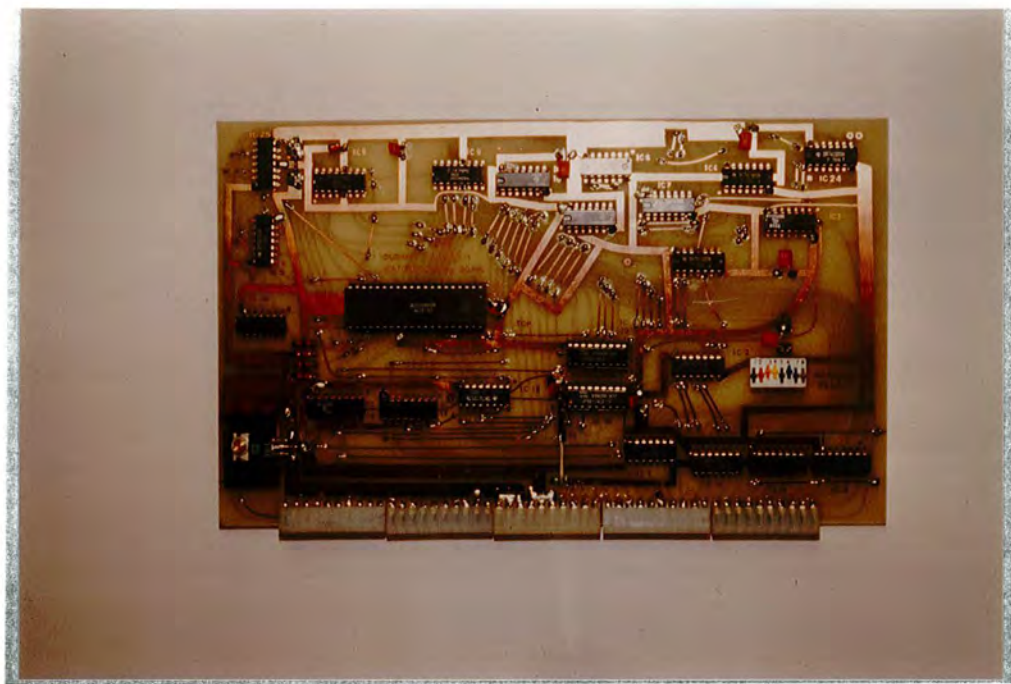


PHOTO 4.2. SLAVE PROCESSOR UNIT. PRINTED CIRCUIT BOARD IMPLEMENTATION

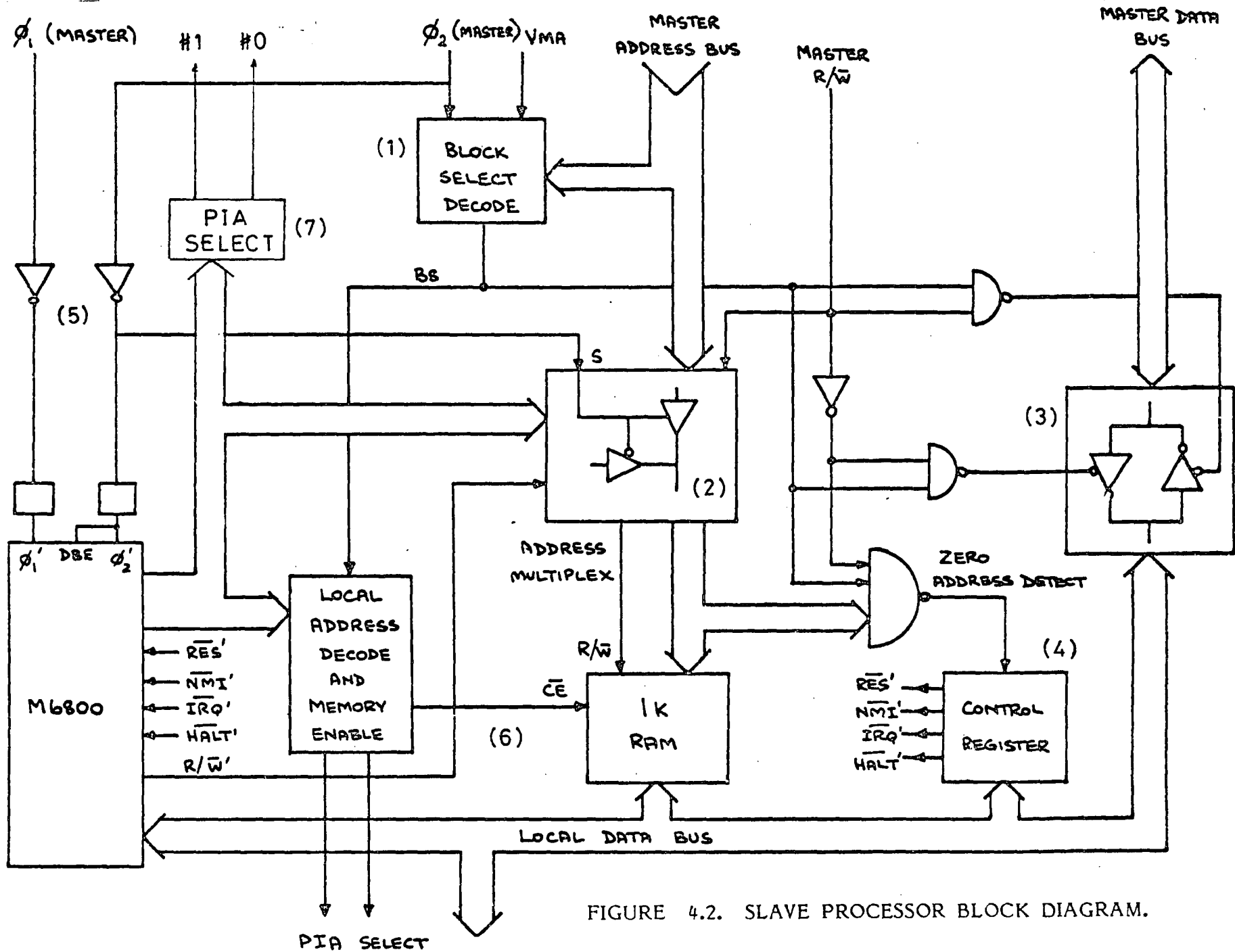


FIGURE 4.2. SLAVE PROCESSOR BLOCK DIAGRAM.

(7) PIA enable logic

The functions and implementation of each unit are discussed separately and reference should be made to the circuit diagram of figure 4.3 ((a) and (b)).

4.3.1 Block select logic

Each of the two inputs of six exclusive-OR gates are connected to a switch and to one of the six high-order address lines respectively. The other sides of the switches are grounded and the gate outputs are combined using two NOR gates and a single NAND gate. The switches may be used to manually locate the 1kbyte of memory anywhere on a 1k boundary within the available address space. VMA and $\overline{\phi}_2$ are also included in the decoding to ensure that the address received is a valid one. In a system which uses more than one slave processor, each may be switch selected to reside within a different segment of memory.

4.3.2 Address multiplexers

The ten low-order address lines from the master processor and the R/ \overline{W} line are multiplexed with the low-order slave processor address lines into the local memory address bus. The local ϕ_2 clock is used to control the address routing.

4.3.3 Data bus buffers

The data bus interface is provided by two bi-directional tri-state buffers. The two control lines, $\overline{\text{Transmit Enable}}$ ($\overline{\text{TE}}$) and $\overline{\text{Receive Enable}}$ ($\overline{\text{RE}}$), allow three possible functions: Data is passed from the master processor data bus to the slave data bus, or from the memory to the master data bus, or both sides of the

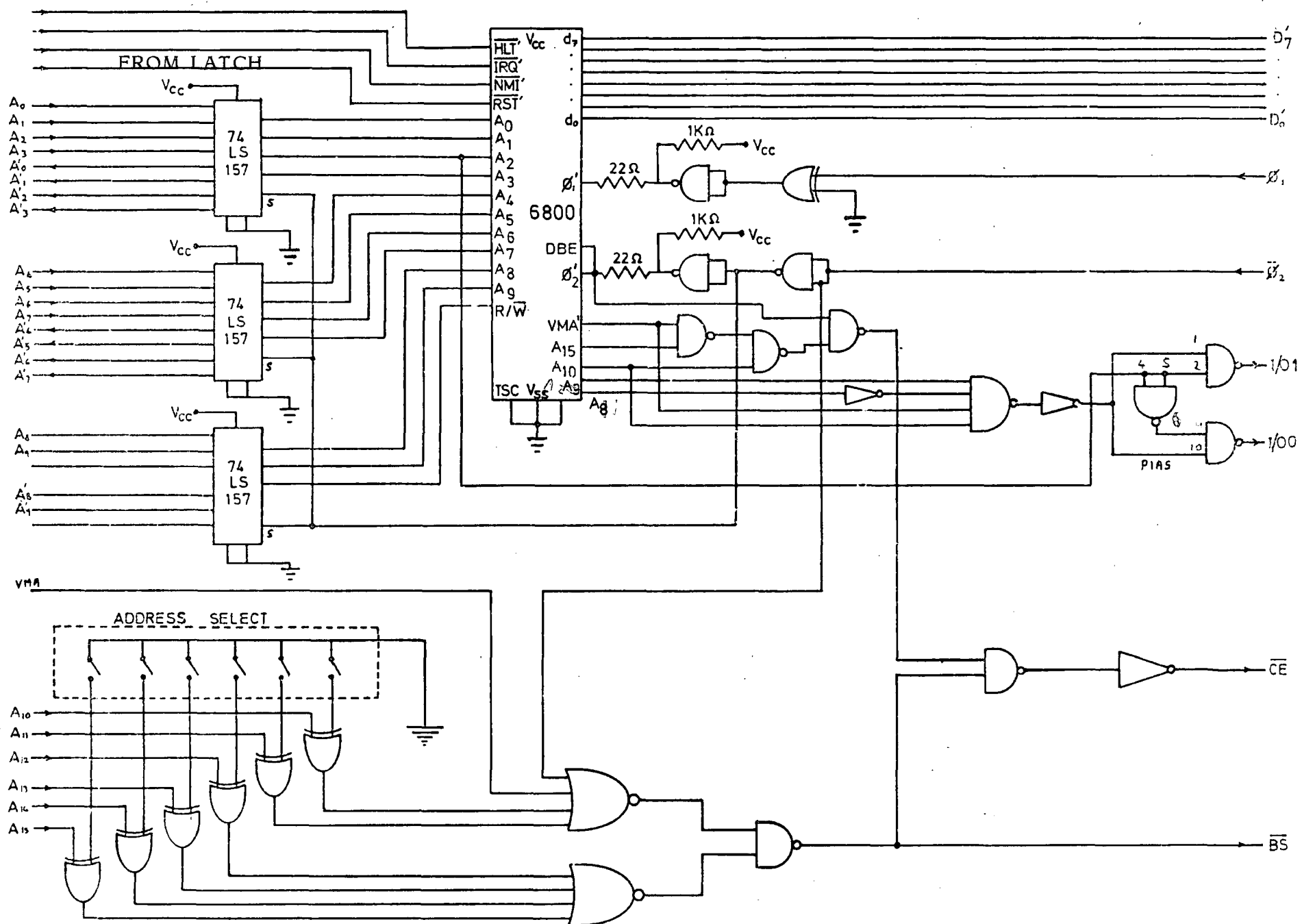


FIGURE 4.3(a). SLAVE PROCESSOR SYSTEM. (address multiplexers, block select, PIA select.)

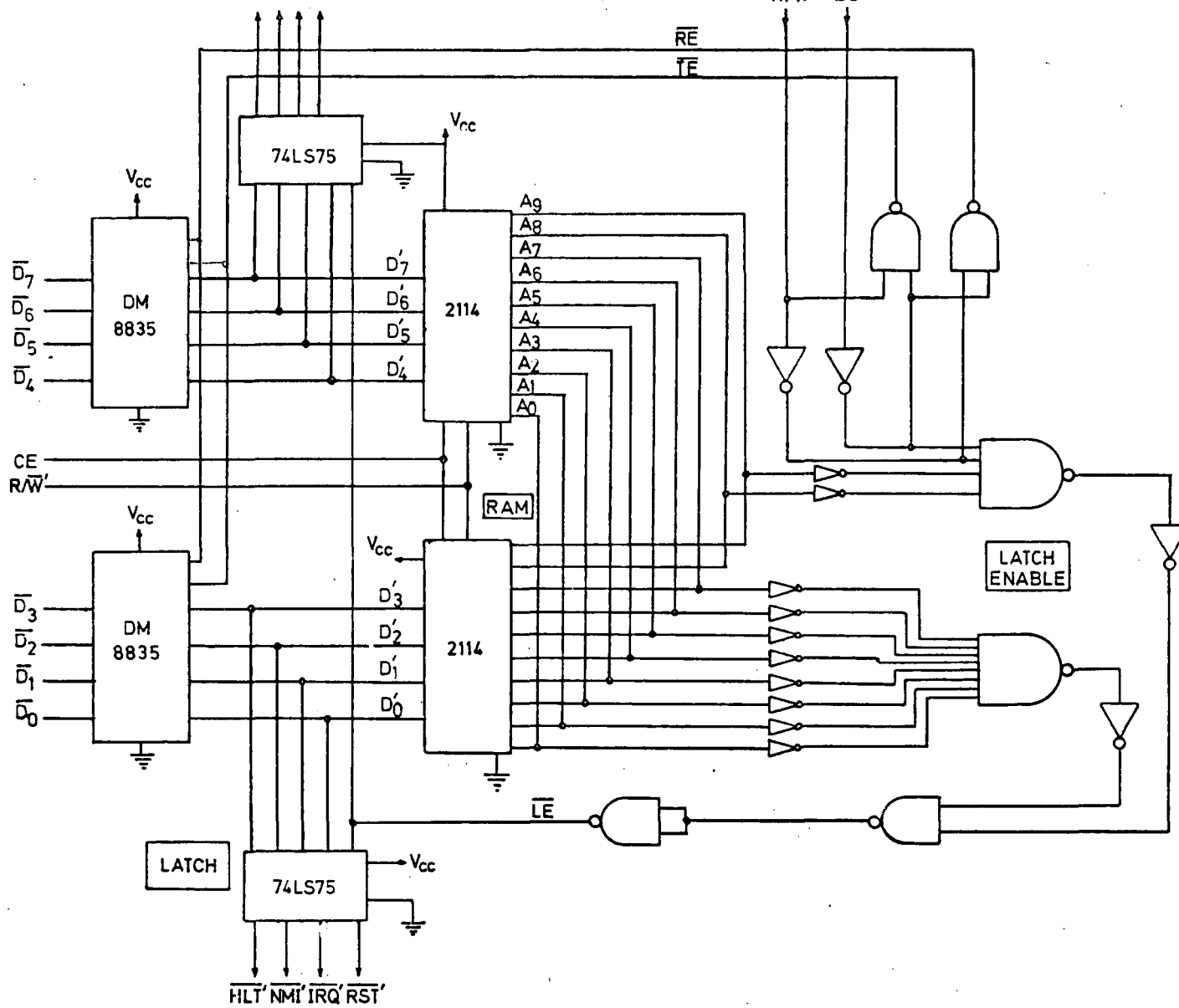


FIGURE 4.3(b). SLAVE PROCESSOR SYSTEM (Data buffers, memory, control latch).

buffers enter the high impedance state thereby preventing interaction between the two busses. This latter state is required to isolate the slave unit from the master when the master is accessing a memory location outside the boundaries of the slave address space. \overline{RE} and \overline{TE} are derived from NAND operations of BS with $\overline{R/\overline{W}}$ and R/\overline{W} respectively and determine the direction of data flow (if any).

4.3.4 Control latch

Two quad latches are selected to reside at the base of the slave processor address space by using ten inverters on the low order address lines to detect the 'zero' address. BS has been included in the select logic to uniquely identify a particular pair of slave latches within the overall system and R/\overline{W} is used to ensure that the latches are "write-only". The four 'Q' outputs from one latch are connected to the \overline{RES} , \overline{NMI} , \overline{IRQ} , and \overline{HALT} control lines of the slave processor, and the outputs from the second latch are left unconnected, to be user defined at a later stage. Data may be written to, but not read from, the control latch by the master processor. Because the latch is invisible to the slave processor, the bottom memory location in the RAM may be freely used by the slave and may also be read by the master.

4.3.5 Clock drivers

The clock signals required for the 6800 microprocessor are not TTL compatible and need to be derived using open-collector drivers with pullup resistors. The antiphase operation of the slave unit (with respect to the master) requires that the clock

signals from the master bus be inverted. This is achieved using a NAND gate as an inverter and an exclusive-OR gate with one input grounded. The latter has been included to equalise gate delays which might otherwise cause unacceptable overlapping of the two clock phases.

4.3.6 Memory

Two RAM IC's, each having a capacity of 1k x 4 bits and an access time of 150 ns were used to provide 1k bytes of continuous memory capable of operating with clock frequencies in excess of 2.0 MHz. The enable lines were tied together and to the output of the chip enable circuitry. The memory must be enabled (a) when the master processor addresses the slave memory block, (b) when the slave processor addresses the lowest 1k bytes of its address space, and (c) when the slave processor addresses any of the eight top locations of its address space (\$FFF8-\$FFFF) which contain the interrupt and restart vector pointers. Two external chip enable lines are provided to enable two 6821 Peripheral Interface Adapters (PIA's) which must remain inactive during memory access cycles. The truth table of table 4.1 determines the select logic required for memory enable decoding. Implementation was achieved using four NAND gates and one inverter. VMA and \emptyset_2 were included to permit only valid addresses.

4.3.7 PIAs

The PIA select circuitry allows the inclusion of two 6821 chips in the slave processor system. Each PIA occupies 4 addresses and address lines A12, A11, A10 and A2 were decoded to

locate the PIAs at \$0C00 to \$0C03 and \$0C04 to \$0C07. Address lines A₁₅ and A₀ were used to reference the internal PIA registers. The truth table (table 4.1) determines (1) that the memory is disabled when selecting a PIA, and (2) that the PIAs may be addressed by the slave processor only.

A ₁₅	A ₁₀	A ₉	A ₈	\overline{CE}	\overline{PIAS}	
0	0	X	X	\emptyset_2	1	Internal RAM
0	1	0	0	1	1	
0	1	0	1	1	1	
0	1	1	0	1	1	
0	1	1	1	1	0	PIA select
1	X	X	X	\emptyset_2	1	Reset vector

Table 4.1

This completes the description of the functional blocks of the slave system. The prototype was constructed on a standard wirewrap board measuring 6.5 x 4.5 ins. and included a single PIA. Photos 4.1(a) and 4.1(b) show top and underside views of the assembled system. SSI components were chosen from the 74LS' series of IC's, and high frequency versions of the LSI chips were used for 2MHz. operation. Connection to the master system was via a 32-way ribbon cable approximately 0.5m in length. Initial tests on the system proved unsuccessful when operating at a clock frequency of 2.0 MHz; however a successful series of diagnostics were performed at 1.0 MHz which will now be described.

4.4 Test Results

The switches on the slave processor board were set to locate the RAM at addresses \$C000-\$C3FF. This is achieved by the following combination, where a '1' indicates a closed switch:

SW5	SW4	SW3	SW2	SW1	SW0
0	0	1	1	1	1

Since the latch is always located at the base address of the slave RAM, its global address in this case was \$C000. The data format of the latch was defined by the system hardware to be:

d7	d6	d5	d4	d3	d2	d1	d0
X	X	X	X	HALT	NMI	IRQ	RES

The following diagnostics were performed to ensure that the slave system was functioning correctly. The assembler written programs referred to in the text were assembled using the co-resident mnemonic assembler.

(1) Memory diagnostic. The slave processor was halted by using the system monitor to write \$00 to the slave latch. The memory diagnostic program 'CDAT-1', written by John Christenson of Motorola Inc. (Appendix 2), was used to test the slave memory (with the exception of the latch address) for faulty bits and convergent address problems. Successful execution of this test ensured that the memory was working satisfactorily.

(2) Reset/interrupt sequence testing. The program listed in Appendix 2 was used to test the slave reset and interrupt operations. For a base address of \$C000, the reset and interrupt vector pointers reside at the following global addresses:

\$C3F8-9	IRQ
\$C3FA-B	SWI
\$C3FC-D	NMI
\$C3FE-F	RES

The local addresses of the interrupt and reset sequences were loaded into the corresponding vector locations and each sequence

was checked by toggling the appropriate interrupt/reset line to initiate the appropriate sequence. Toggling was done by using the system monitor to write to the slave latch. Each test sequence was designed to write a particular byte into global location \$C001 (local address \$0001). This number could be read by the system monitor to check for correct execution and could also be dynamically altered without affecting execution flow of the slave program.

(3) Parallel processing. The third diagnostic was used to demonstrate the parallel processing capability of the master-slave configuration. The example chosen was to evaluate the expression:

$$(a \times b) + (c \times d)$$

where a,b,c and d are 2-byte 2's complement numbers. The program listed in Appendix 2 uses two multiplication routines; one in master RAM, the other in slave RAM. The two numbers a and b were used as arguments for the slave; b x c was evaluated by the master. A comparison of the execution times required for evaluation of the expression by the master only, and by the master-slave configuration showed a two-fold increase in speed by the latter over the former, as would be expected. Note that in programs which require a slave system stack, as in this example, it is necessary to define a local stack pointer; initialising the pointer immediately below the vector space allows the stack to extend downwards through the slave memory.

4.5 Printed circuit construction

The slave circuit was transferred onto a dual sided printed circuit board designed to fit onto the SS-50 bus using standard

Molex connectors. The artwork was drawn twice full size using transfers and tapes, then photo-reduced to a correctly dimensioned "positive" image. The front and back images were carefully aligned and fastened together at two sides to enable the blank board to be slid in between before exposing to ultra-violet light and etching in the usual way. The printed circuit version of the system was found to perform satisfactorily with a 2.0 MHz. clock frequency and three such boards were produced from the original mask. Photo 4.2 shows the printed circuit board implementation.

4.6 Conclusion

The design and implementation of a distributed microprocessor system has been discussed, in which a number of "slave" microcomputer units are controlled by a "master" processor. The master processor designates tasks to the slave units, and instructs them to execute those tasks when required. Parameters may be passed to and from the slave memory in much the same way as parameters are transferred to and from subroutines. However, by operating the slave processors in antiphase to the master, the slave memory may be dynamically accessed by the master, without disturbing execution of the slave program. The increase in processing power which may be achieved using such a system has been demonstrated with the use of examples; this advantage will become further evident in later chapters.

CHAPTER 5

Error-control coding

5.1 Introduction

From a technical point of view, a generalised data communications system may be regarded as consisting of three basic blocks: the transmitter, the channel and the receiver. The transmitter has the task of assigning an analogue waveform to each possible sequence of digits received as input from the data source. This is the process of modulation. The analogue waveforms are propagated through the channel and are then interpreted individually at the receiver so that the output of the receiver detector is a sequence of digits representing best estimates of the transmitted data. The channel in this case is known as the "modulation channel".

The above generalised communications system may be viewed in its entirety as a strictly digital channel. In the binary case this channel accepts 0's and 1's at its input and usually reproduces them at its output. Occasionally, however, because of noise and other channel impairments, the output digits do not agree with the input digits and errors have occurred. Each message is associated with a sequence of bits to be passed through the digital channel. In order that they may be distinguished, it is desirable to associate with messages bit sequences which are as different as possible from one another. This may be achieved by adding redundant bits to each message sequence so that a message sequence of k bits is transmitted as a block of n bits, where $n > k$. The communications system may now be regarded as having the form of figure 5.1, where the "encoder"



FIGURE 5.1 CODING CHANNEL MODEL

adds redundant bits to the source data in a systematic manner. The "decoder" removes the redundancy after transmission over the digital channel (known as the "coding channel") and may attempt to detect or correct errors introduced by the channel.

The components of the coding channel in HF radio systems are the HF transmitter, the HF radio path and the HF receiver. The transmitter and the receiver include the digital modulator and demodulator respectively. As a first step towards minimising the errors, close attention should be paid to the modulation scheme to reduce the effects of intersymbol interference and noise. However, the short-term variations in the characteristics of the HF channel are largely unpredictable, and the channel impairments often result in extremely high error rates. It is therefore desirable to add redundancy in the manner described if these errors are to be eliminated.

This chapter discusses the microprocessor implementation of block coding schemes for random error correction, and shows how the blocks may be interleaved to correct bursts of errors, such as those observed on the HF coding channel. Field test results are discussed in a later chapter.

5.2 Block codes

If the redundancy added to the message digits is to be utilised by the decoder for error control, the redundant bits must be added in a systematic and predetermined manner. An effective way is to use a parity check block coding scheme in which a number $(n-k)$ of modulo-2 sums of (or parity checks on) various digits of a k -bit message digit sequence are computed and

appended to the information digits. The n-bit block is then shifted out onto the channel. The data rate is reduced by a factor k/n , known as the "code efficiency". After transmission through the coding channel, the same parity checks are computed at the decoder; if they do not agree, then errors must have been introduced by the channel. In the binary case, if it is possible to locate the errors, they may be corrected.

An (n,k) block code is defined as the collection of 2^k n-tuples produced by encoding all possible k-tuples according to some pre-determined set of parity-check rules. The encoding of a data block into a code word can be represented mathematically as:

$$c = dG$$

where c is an n-bit code word represented as an n-place row vector (n-tuple), and d is a k-bit data block represented as a k-place row vector (k-tuple). The k-by-n matrix G is the generator matrix of the code and has the form:

$$G = [I_k \ P]$$

where I_k is the identity matrix of order k, and P is an arbitrary k-by-(n-k) matrix. The leftmost k symbols of c are therefore identical to the corresponding symbols of d , while the rightmost n-k symbols are modulo-2 sums of, or parity checks on, various symbols of d .

In order that a code may detect up to t errors per codeword, the minimum Hamming distance between words in the code must be at least $t+1$. If the code is to correct up to t errors per

codeword, the minimum Hamming distance between codewords must be increased to $2t+1$. The problem is to choose the matrix P to maximise the minimum distance between codewords.

5.3 Cyclic codes

Much of the research in coding theory has been concentrated on a small subclass of block codes, the cyclic codes (40). These codes possess a fair amount of mathematical "structure", allowing codes to be designed having good error-correcting properties and which may be implemented with a minimum of hardware or software. A cyclic (n,k) code, a linear block code of length n having k information symbols, has the property that every cyclic shift of a code word is another code word.

That is if:

$$c = \{c_{n-1}, c_{n-2}, \dots, c_0\}$$

is a code word, so are:

$$\{c_{n-2}, c_{n-3}, \dots, c_{n-1}\}$$

$$\{c_{n-3}, c_{n-4}, \dots, c_{n-2}\}$$

:

$$\{c_0, c_{n-1}, \dots, c_1\}$$

The elements of each code word can be treated as coefficients of a polynomial of degree $n-1$. The codeword can be represented as a code polynomial; that is,

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \dots + c_1x + c_0$$

That $c(x)$ is a code word implies that $x^i c(x)$ modulo (x^n+1) is also a code word for all i . The polynomial representation of the

bottom row of the matrix G defined earlier is known as the "generator polynomial", $g(x)$, of the code. All other rows in the matrix of a cyclic code are multiples of this polynomial; given the generator polynomial of the code, it is possible to construct the generator matrix (41). Since $g(x)$ has degree $n-k$, there are 2^k polynomials, ie. those of degree less than k , which can be multiplied by $g(x)$ to yield a polynomial of degree less than n . Clearly, there is a one-to-one correspondence between these polynomials and the 2^k words in the code. It may be proved (12) that the generator polynomial, $g(x)$, is always a divisor of x^n+1 .

It may be shown (41) that encoding a k -bit data block by multiplying it by the generator matrix G is equivalent to the following polynomial operation. The polynomial representation of the information block, denoted by $d(x)$, has degree less than k ; therefore $x^{n-k}d(x)$ has degree less than n . Also

$$\frac{x^{n-k}d(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)} \quad (5.1)$$

where $q(x)$ has degree less than k and $r(x)$ has degree less than $n-k$, the degree of $g(x)$. Thus the polynomial $c(x) = x^{n-k}d(x) + r(x)$ is divisible by $g(x)$ and is a code word in the code generated by $g(x)$. This word consists of the unaltered k -bit information block followed by $n-k$ linear combinations of the

information bits.

If c is a code word in the code generated by the matrix G defined previously, then

$$c \begin{bmatrix} P \\ I_{n-k} \end{bmatrix} = 0$$

Hence, any n -tuple e that is not a code word gives:

$$e \begin{bmatrix} P \\ I_{n-k} \end{bmatrix} = s \neq 0$$

The vector s is an $(n-k)$ -tuple referred to as the "syndrome" of the n -tuple. Every n -tuple has one, and only one, syndrome. The syndrome is obtained by encoding the information section of an n -tuple and adding (modulo-2) the resultant check bits to the corresponding bits of the parity section of the n -tuple.

Let $e(x) = e_d(x) + e_p(x)$ where $e_d(x)$ and $e_p(x)$ are the data and parity sections of the n -tuple $e(x)$ respectively. The syndrome $s(x)$ is given by:

$$s(x) = e_p(x) + r(x)$$

where $r(x)$ is the residue obtained by dividing $e_d(x)$ by $g(x)$. But since $e_p(x)$ has degree less than $g(x)$, this is identical to the residue obtained by dividing $e(x) = e_d(x) + e_p(x)$ by $g(x)$. That is,

$$s(x) = \text{rem} \frac{e(x)}{g(x)}$$

Since $g(x)$ divides every code word $c(x)$, the syndrome is identical

to that of $e(x) + c(x)$.

5.4 The BCH codes

The BCH (Bose-Chaudhuri-Hocquenghem) codes (42) are a class of cyclic codes of particular interest. They are defined in terms of the BCH bound, a statement of which is given later. The proof of this bound may be found in reference (12).

The "primitive" BCH codes are of length $2^m - 1$ (m integer) and require, at most, mt check bits to correct up to t errors per codeword. It has been shown that every binary cyclic code of length n is completely determined by its generator polynomial, $g(x)$, a divisor of $x^n + 1$. The polynomial $x^n + 1$ with binary coefficients may be factored into n linear factors:

$$(x + \alpha_1)(x + \alpha_2) \dots (x + \alpha_n)$$

where the roots, α , are elements of some larger field. These n roots can be shown to form a cyclic group under the operation multiplication. That is, for some (primitive) root, α , the n roots can be expressed as

$$\alpha^1, \alpha^2, \dots, \alpha^{n-2}, \alpha^{n-1}, \alpha^n = 1 = \alpha^0$$

The lowest-degree polynomial with binary coefficients which divides $x^n + 1$ and of which α^i is a root is referred to as the minimum polynomial of α^i , and is designated $m_i(x)$. If such a polynomial is considered to have integer coefficients, all coefficients of the polynomial $m_i^2(x) - m_i(x^2)$ are even. In the binary case:

$$m_i^2(x) = m_i(x^2)$$

and if α^i is a root of $m_i(x)$, so are α^{2i} , α^{4i} , α^{8i} The number of roots is the degree of $m_i(x)$. Tables exist for values of n to determine which roots of x^n+1 are roots of a given divisor of x^n+1 , or equivalently, which polynomial is the minimum function of a given root of x^n+1 .

The BCH bound can now be stated as follows: The minimum distance of the code generated by $g(x)$ must be greater than the largest number of consecutive roots of $g(x)$. (the j roots α^{i+j} , \dots , α^{i+1} are "consecutive" for $0 \leq i \leq n-1$.) Since $g(x)$ has degree $n-k$, exactly $n-k$ of the roots of x^n+1 are roots of $g(x)$.

Encoding the BCH codes is a straightforward procedure, and obeys the general rules for encoding of cyclic codes. The data polynomial is multiplied by x^{n-k} and divided by the generator polynomial. The residue of this division is added to the data polynomial to form the code word.

5.4.1 Decoding BCH codes

A decoding algorithm for BCH codes that can be implemented with a reasonable amount of equipment or software has been devised by Peterson (43).

The basic decoding problem may be outlined as follows. Consider a code word $c(x)$ transmitted through a noisy channel. Let $e(x)$ denote the error polynomial added to $c(x)$ by the channel. Decoding consists of determining $e(x)$ from $s(x)$, the syndrome calculated from the received polynomial $r(x) = c(x) + e(x)$. Then $c(x)$ is determined by adding $e(x)$ to $r(x)$.

A primitive BCH code has the generator polynomial:

$$g(x) = \text{LCM} [m_1(x), m_3(x), \dots, m_{2t-1}(x)]$$

ie., $m_1(x)$, $m_3(x)$, ..., and $m_{2t-1}(x)$ divide $g(x)$ and hence $c(x)$. For the Peterson decoding algorithm it is necessary to compute t partial syndromes by dividing $r(x)$ successively by $m_1(x)$, $m_3(x)$, ..., $m_{2t-1}(x)$. The residue obtained by dividing $r(x)$ by $m_i(x)$ is denoted S_i , and is called the i th partial syndrome. The first partial syndrome can be obtained by dividing $r(x)$ by $m_1(x)$

$$S_1(x) = \text{rem} \frac{r(x)}{m_1(x)}$$

But $r(x) = c(x) + e(x)$ and since $m_1(x)$ divides $c(x)$, this gives

$$S_1(x) = \text{rem} \frac{e(x)}{m_1(x)}$$

If $e(x)$ has only one nonzero term, that is, $e(x) = x^i$, $0 \leq i \leq n-1$, $S_1(x)$ is the unique residue corresponding to that value of i . If j errors occur, $S_1(x)$ is the sum of the residues corresponding to the various errors. That is, if β_j represents the residue corresponding to the j th error,

$$S_1 = \beta_1 + \beta_2 + \dots + \beta_j$$

The β_j are known as the error-locator numbers. If it is possible to determine each of these numbers it is possible to correct the errors since each number is associated with a particular position in the word. The set of residues form a Galois field, and arithmetic operations must be carried out within this field.

There are 2^m residues in the field, each of which is m bits in length; the field consists of 2^m m -tuples obtained by dividing x^i , $0 \leq i \leq n-1$ by $m_1(x)$.

The "elementary symmetric functions", σ , are related to the error locator numbers as follows:

$$\begin{aligned}\sigma_1 &= \beta_1 + \beta_2 + \dots + \beta_t \\ \sigma_2 &= \beta_1\beta_2 + \beta_1\beta_3 + \dots + \beta_{t-1}\beta_t \\ \sigma_t &= \beta_1\beta_2\beta_3 \dots \beta_t\end{aligned}$$

Having determined the partial syndromes, it is possible to find the elementary symmetric functions. The error locator numbers may then be found by substituting field elements in the equation:

$$\sum(x) = (x + \beta_1)(x + \beta_2) \dots (x + \beta_t) = 0$$

The binary representations of the exponents of the field elements then point to the bits in error which may subsequently be corrected (44).

5.5 Code Implementation

Several hardware and high-level language implementations of the BCH coding/decoding algorithms have been described in the literature (45-48). This section discusses the microprocessor implementation of these algorithms for real-time operation. It will be shown that the necessary Galois field arithmetic operations may be performed without difficulty using the 6800 microprocessor. Codes of several lengths were investigated; however, special reference is made to the (15,7) dual error correcting code, subsequently employed in the HF modem/codec discussed in chapter 7.

5.5.1 Encoder

It has been shown that a k-bit data block may be encoded into an n-bit cyclic codeword by performing the polynomial operation of equation (5.1) and adding the residue $r(x)$ to the original data block. Encoding therefore simply consists of determining $r(x)$.

The polynomial division was implemented in software (see "BCHCOD", appendix 2), using the "EOR" (exclusive-OR) and the "ASL" (arithmetic shift left) microprocessor instructions to simulate the operation of a shift register implementation.

The (15,7) code has the generator polynomial

$$\begin{aligned}g(x) &= m_1(x) m_3(x) \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)\end{aligned}$$

Encoding a data polynomial $d(x)$ is equivalent to finding

$$c(x) = x^8 d(x) + \text{rem} \frac{x^8 d(x)}{g(x)}$$

Multiplication of $d(x)$ by x^8 simply involves shifting the 7-bit data block 8 places to the left; ie. transferring it to the next highest byte.

Modulo-2 division by the generator polynomial was implemented as follows. The binary representation of $g(x)$ is

111010001

This is stored as a left justified double byte which is lined up with the two bytes containing the product $x^8 d(x)$. A bit-by-bit EXclusive OR operation is then performed and the result is shifted left until left justified. The result becomes the new

dividend and the procedure is repeated until a total of eight shifts have been performed. The final result is the residue of the division of $x^8d(x)$ by $g(x)$, which is added to $x^8d(x)$ to produce the 15-bit code word.

5.5.2 Decoder implementation

Section 5.4.1 has discussed the operations necessary to implement the Peterson decoding algorithm for BCH codes. Three stages are required:

- (1) Computation of the partial syndromes
- (2) Calculation of the elementary symmetric functions
- (3) Determination of the error locator numbers

Software was written to implement this algorithm for the (15,7) code. (see "BCHDEC", appendix 2) Two partial syndromes are required, S_1 , and S_3 . These were obtained using polynomial division operations, computed in a similar manner to the encoding procedure. Steps (2) and (3) required operations to be performed in the Galois field. Arithmetic operations in the field may be easily implemented on a microprocessor system as the following section describes.

5.5.3 Galois field operations

Arithmetic operations were to be performed in the Galois field of 2^m (=15) elements. A representation of this field can be formed using the primitive polynomial $m_1(x) = x^4 + x + 1$ (a factor of $g(x)$) and is shown in figure 5.2. The field consists of all polynomials of degree $m-1$ or less; the field elements can be represented as 4-bit binary numbers.

$\alpha^0 = 1$	$=$	1	$=$	1	0	0	0
$\alpha^1 =$	α	$=$	0	1	0	0	0
$\alpha^2 =$	α^2	$=$	0	0	1	0	0
$\alpha^3 =$	α^3	$=$	0	0	0	1	0
$\alpha^4 =$	$1 + \alpha$	$=$	1	1	0	0	0
$\alpha^5 =$	$\alpha + \alpha^2$	$=$	0	1	1	0	0
$\alpha^6 =$	$\alpha^2 + \alpha^3$	$=$	0	0	1	1	0
$\alpha^7 =$	$1 + \alpha + \alpha^3$	$=$	1	1	0	1	0
$\alpha^8 =$	$1 + \alpha^2$	$=$	1	0	1	0	0
$\alpha^9 =$	$\alpha + \alpha^3$	$=$	0	1	0	1	0
$\alpha^{10} =$	$1 + \alpha + \alpha^2$	$=$	1	1	1	0	0
$\alpha^{11} =$	$\alpha + \alpha^2 + \alpha^3$	$=$	0	1	1	1	0
$\alpha^{12} =$	$1 + \alpha + \alpha^2 + \alpha^3$	$=$	1	1	1	1	0
$\alpha^{13} =$	$1 + \alpha^2 + \alpha^3$	$=$	1	0	1	1	0
$\alpha^{14} =$	$1 + \alpha^3$	$=$	1	0	0	1	0
$\alpha^{15} =$	$1 = \alpha^0$	$=$	1	0	0	0	1

FIGURE 5.2 REPRESENTATION OF GALOIS FIELD $GF(2^4)$.

eg. the field element α^8 can be represented as 0101.
Methods for implementing operations in the field are described as follows:

(i) Addition. Field elements may be added (modulo 2) term by term in the ordinary way.

(ii) Subtraction. Subtraction of field elements is the same as addition.

(iii) Multiplication. The rule for multiplication is to multiply in the ordinary way, reducing the answer modulo-2 and modulo- $m_1(x)$ to a polynomial of degree $m-1$ or less. This is done by considering the equation $m_1(x) = 0$ and using the equation to eliminate terms of power greater than $m-1$. Implementation on a microprocessor can be done using tables of exponents and field elements. Two tables are arranged in memory; the addresses of table 1 correspond to the exponents of α , and the data corresponds to the field elements. The addresses of table 2 correspond to the field elements and the data corresponds to the exponents. To multiply two field elements together, the exponents of the elements are found from table 2, added together modulo-15, and the product field element is determined from table 1. For example, suppose it is required to multiply together the field elements $(1+\alpha)$ and $(1+\alpha^2)$. These elements correspond to the hexadecimal numbers 03 and 05. From table 2, the exponents are found to be 04 and 08. The sum of the exponents is 0C, and the product of the field elements is determined (from table 1) to be 0F, which corresponds to the polynomial $1+\alpha+\alpha^2+\alpha^3$.

(iv) division. Division is performed in a similar manner to multiplication, except that the exponents of α^i are subtracted and not added.

Having computed the partial syndromes S_1 and S_3 , it is possible to find the partial syndrome S_2

$$S_2 = S_1^2$$

The error locator numbers are found from:

$$S_1 = \beta_1 + \beta_2$$

$$S_2 = \beta_1^2 + \beta_2^2$$

and the elementary symmetric functions are:

$$\sigma_1 = \beta_1 + \beta_2$$

$$\sigma_2 = \beta_1 \beta_2$$

We find that

$$\sigma_1 = S_1$$

$$\sigma_2 = S_2 + \frac{S_3}{S_1}$$

If field elements are substituted into the equation

$$x^2 + \sigma_1 x + \sigma_2 = 0$$

the errors may be located, and hence corrected. A flowgraph of the complete decoding procedure is shown in figure 5.3. An assembler listing of the implementation of this algorithm is shown in the listing BCHDEC in appendix 2.

5.6 Results

The software implementation of the decoding algorithm was tested by encoding all possible 7-tuples, adding all possible 15-bit error patterns to the resulting codewords, then decoding the corrupted words and comparing with the original codewords to

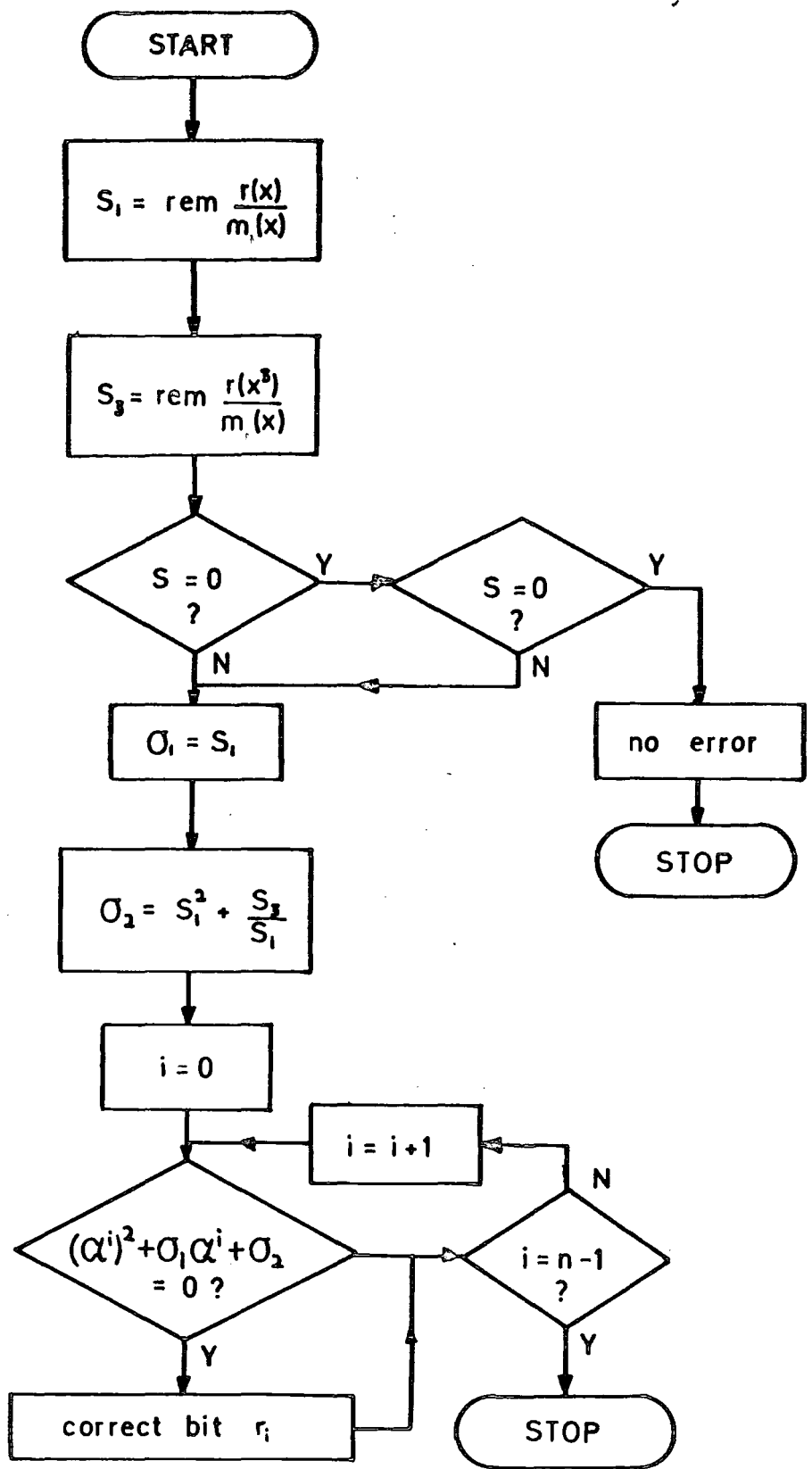


FIGURE 5.3 DECODING PROCEDURE FLOWCHART

determine if the errors had been corrected. For a t-error-correcting code there are:

$$\sum_{i=1}^t \frac{n!}{i!(n-i)!}$$

correctable error patterns. For the (15,7) dual-error-correcting code there are 120 correctable patterns of which 15 contain single errors and 105 contain double errors. We may also consider the all-zero 15-tuple to be an allowed error pattern as it results in an uncorrupted word. The test results indicated that all single and double errors were corrected but that correction was not possible for error patterns having a weight exceeding 2.

Measurements were taken of the time required to encode a 7-bit data word and of the time required to decode a received word containing 0, 1, or 2 errors. A slave processor unit was used to time the procedure to the nearest 10 μ s. In each case the average time taken for all codewords was measured, together with the best (shortest) and worst times. Best and worst times corresponded to the all-zero and the all-ones codewords respectively. The results are tabulated in table 5.1.

		best	average	worst
	encoding	0.11ms	0.12ms	0.13ms
decoding with:	0 errors	0.31ms	0.59ms	0.85ms
	1 error	1.33ms	1.56ms	1.79ms
	2 errors	1.68ms	1.87ms	2.08ms

Table 5.1. Execution times.

The limitation on the maximum possible data throughput for this coding scheme was determined by the worst case decoding time for 2-error correction. If the maximum decoding time for an n-bit codeword is t_{\max} and the code rate is r, the maximum data throughput is:

$$\frac{rn}{t_{\max}} = \frac{0.47 \times 15}{2.08 \times 10^{-3}} = 3.36 \text{ kb/s.}$$

Memory utilisation was found to be only 25 bytes for the encoder and 272 bytes for the decoder (including lookup tables).

5.7 Burst error correction

So far, the codes discussed have been cyclic block codes capable of correcting up to t random errors in a block of n bits. Many channels do not exhibit random error characteristics; it is known that HF radio channels are primarily subject to "bursts" of errors. The random error correcting codes described may therefore not always be effective over links using the HF channel. In general, three methods are known for combating the effects of burst errors:

- (i) Long, random-error-correcting codes may be designed to have large minimum distance, enabling a large number of errors to be corrected per block.
- (ii) Specialised "burst-trapping" codes may be used (49,50).
- (iii) Codewords in a random error correcting code may be interleaved to spread the errors over a large number of codewords.

Method (i) necessitates a large amount of hardware or software at the decoder. If the code is to be used purely to

correct burst errors, the random error correcting capability of the code is effectively wasted. This is intuitively obvious if it is considered that there are many more ways of arranging t random errors in a block of n bits than there are ways of arranging t consecutive errors. Codes having large minimum distance have low efficiency; consequently the data rate may be considerably reduced.

Method (ii) requires a good knowledge of the channel statistics. This is not usually possible for HF links (51). If both random and burst errors occur over the channel, such codes may be ineffective.

Method (iii) appears to be the most favourable for data transmission over the HF channel. The method of interleaving can be illustrated as follows. d codewords in a t -error-correcting cyclic (n,k) block code are arranged as rows in a matrix (figure 5.4). The coefficient $c_{i,j}$ represents the j th bit of the i th codeword. The matrix is transmitted as the transpose, ie. column by column, so that a burst of consecutive errors will be spread over more than one codeword. Provided that no more than t errors occur per codeword, all the errors may be corrected at the decoder. The burst correcting ability, b , of the interleaved code is $b=dt$ and d is known as the "interleaving depth". If the length of the error burst is less than dt some random errors may also be corrected. Note that if exactly b errors occur, no more errors may be corrected in that matrix. On average the burst correcting ability of the interleaved code is subject to there being an error-free interval or "guard time" of at least dn bits, thus

$C_{0,n-1}$	$C_{0,n-2}$	$C_{0,n-3}$	$C_{0,n-4}$.	.	.	$C_{0,1}$	$C_{0,0}$
$C_{1,n-1}$	$C_{1,n-2}$	$C_{1,n-3}$	$C_{1,n-4}$.	.	.	$C_{1,1}$	$C_{1,0}$
$C_{2,n-1}$	$C_{2,n-2}$	$C_{2,n-3}$	$C_{2,n-4}$.	.	.	$C_{2,1}$	$C_{2,0}$
.
.
$C_{d-2,n-1}$	$C_{d-2,n-2}$	$C_{d-2,n-3}$	$C_{d-2,n-4}$.	.	.	$C_{d-2,1}$	$C_{d-2,0}$
$C_{d-1,n-1}$	$C_{d-1,n-2}$	$C_{d-1,n-3}$	$C_{d-1,n-4}$.	.	.	$C_{d-1,1}$	$C_{d-1,0}$

FIGURE 5.4 ILLUSTRATING INTERLEAVING OF CODEWORDS TO COMBAT BURST ERRORS.

ensuring that the number of errors does not exceed the error correcting capability of the code (52).

Bit interleaving of block codewords may easily be implemented in microprocessor systems using software as will be illustrated in a later chapter.

5.8 Conclusion

This chapter has discussed several properties of cyclic block codes and has shown how these codes may be interleaved to eliminate the burst errors experienced over HF radio links. In particular, a microprocessor implementation of a coding/decoding scheme for the Bose-Chaudhuri-Hocquenghem codes has been discussed in depth. The finite field arithmetic operations required for the decoding algorithm for this code may be easily implemented in real time using conventional microprocessors and allow medium-speed data transmission with a high degree of error protection. Microprocessor implementations of error control are attractive in an adaptive coding environment, as the coding scheme may be changed by simple modifications to the system software.

CHAPTER 6 HF Interference Pattern Measurements

6.1 Introduction

Many of the errors occurring over HF radio data links may be attributed to interference caused by other users of the spectrum (53). A reduction in the error rate is possible if the spectrum of the transmitted signal is arranged to avoid those regions of the channel which contain interference (54). For optimum performance, the spectral distribution of the signal must be adapted to suit the prevailing interference conditions.

An experiment is described in this chapter, in which microprocessor data logging and analysis techniques were used to investigate the fine-grain structure of interference occurring within an HF voice channel. Spectral analysis of the channel was achieved using a charge coupled device to evaluate the chirp-z transform algorithm discussed in section 2.6 of this thesis. The interference measurements were based on an estimation of the power density fluctuations in each of 64 equal-width frequency windows contained within the channel. Results are presented showing the distribution of the occupancy of the windows by interfering signals, and the probability of the occurrence of interference over an interval of time.

6.2 System operation

The system was based around the RC5601 power spectral density board, described in chapter 2. This is an evaluation module based around a CCD quad chirped transversal filter and can be used to calculate power spectral densities from a 512-point

transform by the chirp-z transform algorithm. The module forms a discrete-time spectrum analyser, selecting and outputting the magnitude and frequencies of the spectral components of an analog input waveform. The analysis band extends from zero to the Nyquist frequency (one-half the sample frequency). A mirror image also appears extending from the sample frequency down to the Nyquist frequency. The resolution bandwidth is (1/512) of the sample frequency.

For this application the CCD evaluation module was externally clocked to provide a sampling rate of 6 kHz. The analysis band therefore extended from 0 to 3 kHz, and the resolution bandwidth was 11.7 Hz. The 256 frequency points were reduced to 64 frequency "windows" by combining the energies present in 4 adjacent points. Adjacent windows overlapped because of Hanning windowing of the CCD filters which tended to spread a spectral line over more than 1 point. The overall effect was to produce a series of 64 overlapping bandpass filters. The amplitude-frequency characteristics of the equivalent filters are shown in figure 6.1, and were derived from the following considerations. A computer simulation using the Hanning window equation applied to the CCD transversal filters showed that a sinusoidal input of frequency f_1 , coincident with a spectral point, k_1 , has its energy distributed as 50% at k_1 , 24% at k_1+1 , 24% at k_1-1 , and 2% outside this region. As shown in the diagram, only 75% of the energy is captured by the filter encompassing k_1 , the remainder falling outside the range of the summation. The response at frequency f_1 is therefore $20 \log_{10}(0.75) = -2.5$ dB. The overall response of the

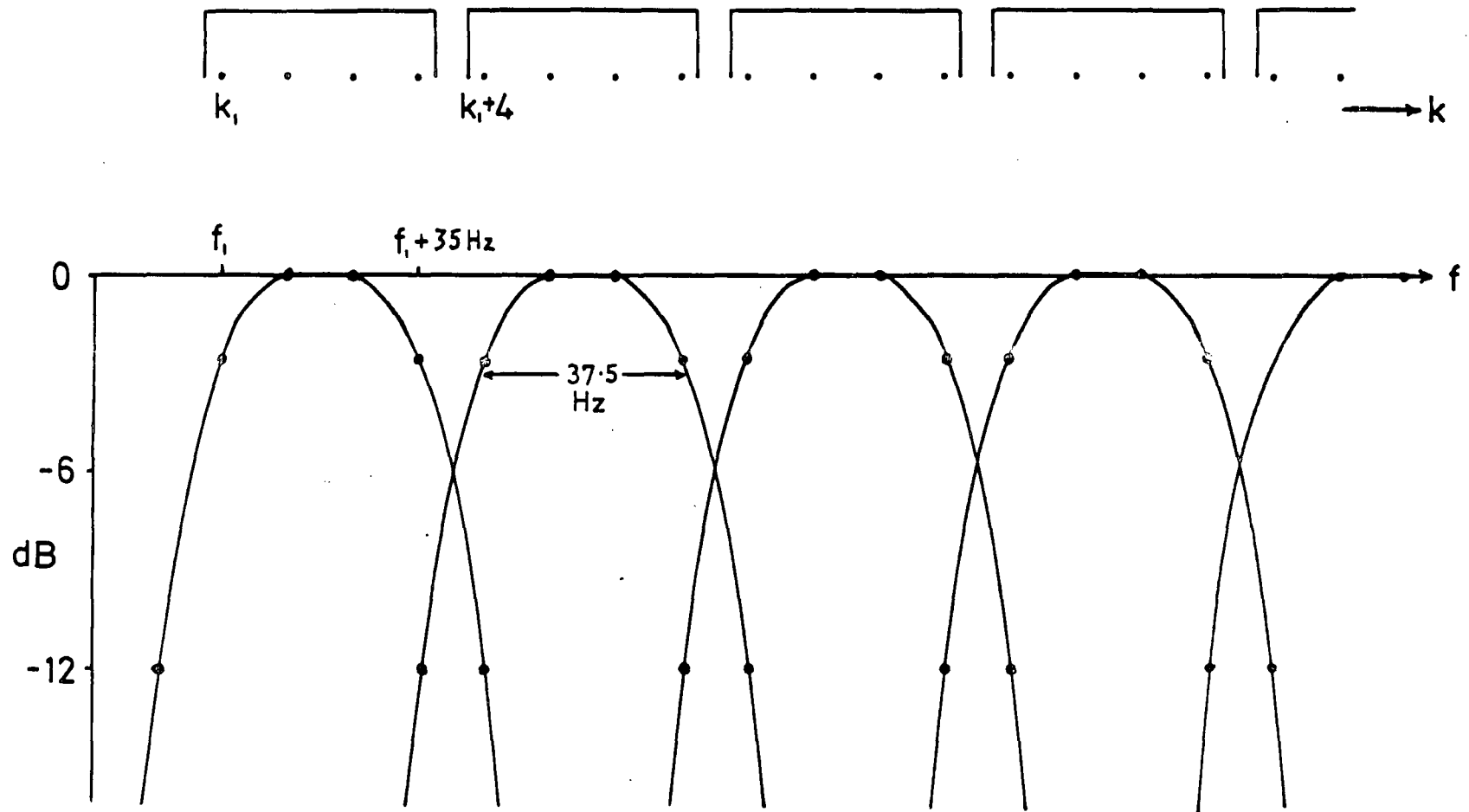


FIGURE 6.1. Equivalent Filter: Amplitude - Frequency Response

equivalent filters was computed from similar considerations.

At a sampling rate of 6 kHz, a complete transform is computed in $(512/6 \times 10^3) = 85.3$ ms. For this experiment, the power ^{avg}spectrum within the windows was integrated over 2.7s every 16s. This required 32 transforms to be evaluated and averaged at 16s intervals. The output for each spectral point was sampled by an 8-bit analogue-to-digital converter and averaged over the 32 transforms. Each point was then quantised to 16 levels of amplitude, assigned 0-F (hexadecimal). After each 16s interval, the level for each of the 64 points across the frequency range (0-3 kHz) was printed on the terminal, together with the current time of day. Results were also routed to a floppy disc file for subsequent analysis. A 1Hz signal source was used to interrupt the processor at 1s intervals to allow a real-time clock to be maintained within the system.

6.3 Hardware

Figure 6.2 shows a block diagram of the equipment used in the experiment. An inverted-V half-wave dipole antenna was cut to resonate at 4.7925 MHz and was mounted 25m above ground, in a north-south orientation. The antenna was connected to a synthesised communications receiver (the RF-505A), whose audio output was connected to the analogue input of the CCD evaluation module. The CCD module was externally triggered from a signal generator set up to provide pulses of 500 ns duration with a pulse repetition frequency of 6 kHz. The module was synchronised to the microprocessor system by generating an interrupt after each output sample. The output samples were converted to digital

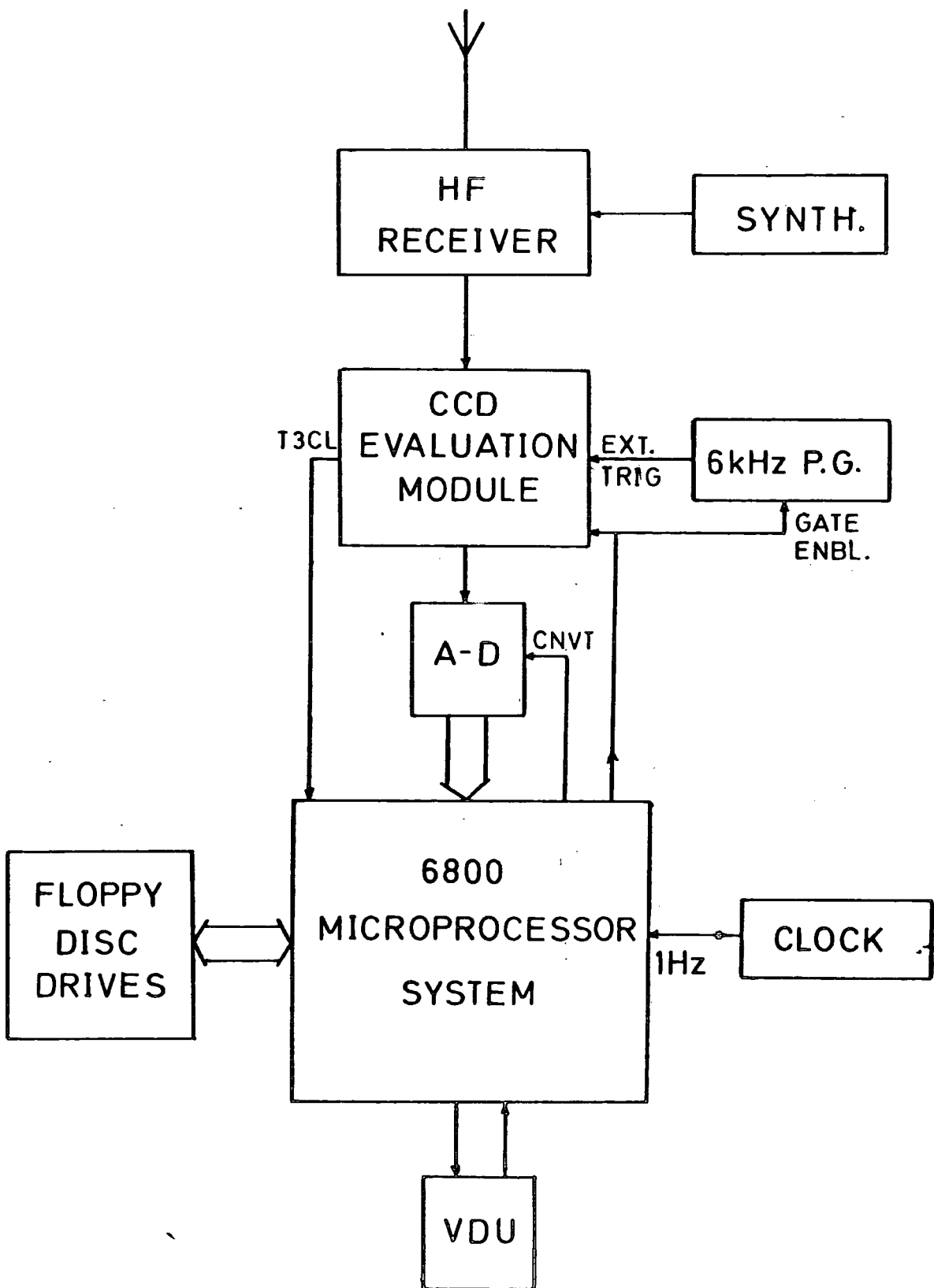


Figure 6.2. Equipment Block Diagram

form using an 8-bit analogue-to-digital converter, and the resulting digital signal was processed as described in the next section. The 1 Hz real-time clock signal was derived from an external digital clock module.

Two $600\ \Omega$ independent sideband audio outputs are available from the RF-505A receiver. The lower sideband output was selected and was attenuated through a potential divider network to be used as input to the CCD evaluation module. This module accepts an input time signal of 1V peak-peak maximum and produces a spectral output of 4V maximum. The output voltage peak for a constant frequency sinusoidal input will be proportional to the input voltage over this range, as the module computes the rms spectral density of the input signal.

The maximum audio output from the RF receiver was obtained by applying a sine wave from an RF signal generator to the antenna terminal of the receiver and using the BFO to generate a beat note. The attenuator was then adjusted to produce a maximum undistorted output from the evaluation module at the beat note frequency. Following this, the signal generator was removed and the antenna was reconnected to the receiver. The background noise was observed at the output of the CCD evaluation module and was found to produce an average noise output (in the frequency domain) of 0.17V. The dynamic range of the spectral density output was therefore $20\log_{10}(4/0.17) = 27.4\text{dB}$. This is somewhat less than the dynamic range of signals received at the front end of the receiver; however, the receiver AGC results in a dynamic range reduction in the audio stages.

The voltage output from the CCD evaluation module was linearly quantised to 16 levels using software (described in the next section). The first quantisation level therefore represented an interference level of $20\log_{10}(0.25/0.17) = 3.3\text{dB}$ above the background noise level. This is the interference threshold; ie. the threshold above which a frequency window may be said to contain interference.

6.4 Software

The system software was required to (a) read the sampled analog output data from the CCD evaluation module, (b) find the average magnitude of each group of 4 consecutive frequency bins, (c) average the results over 32 transforms and (d) quantise each averaged magnitude to 16 levels. Routines were also written to store, on flexible disc, the time of day and the number of windows containing any interference.

After system initialisation, the real time clock parameters were set up by entry from the system terminal. At this point, the "in sync" (to the CCD module) and the "gate enable" (to the pulse generator) signals were held at logic '0' by the CB2 output line from the PIA. This caused the 9-bit address counter to be reset. When this line was brought to a '1' state, the sampling clock (6 kHz pulse generator) was enabled, and the address counter was automatically incremented on each clock pulse. The $\overline{T3CL}$ output from the evaluation module is phase 3 of the 4-phase clock required to transport the charge packets along the CCD. The transition of $\overline{T3CL}$ coincides with valid output data and was used to cause an IRQ interrupt sequence to be generated by the

PIA.

The first 512 clock periods served merely to load the filter. The Nyquist frequency band was available at the output during the next 256 clock periods, but no output data was sampled until the following 256 clock periods, during which time the signal band was available. It was then necessary to introduce a delay of 256 clock periods, during which time the next Nyquist band appeared at the output. The filter was then operating in a continuous serial mode, outputting one sample in the frequency domain for each sample clocked in the time domain. The process was continued until 32 sets of frequency-domain samples had been collected.

The results from the 32 transforms were averaged and quantised to 16 levels for each frequency window. At each 16s interval, the time of day and the averaged results were printed as a single line of output on the hard-copy terminal. The number of windows in which any interference had been observed was written as a single byte to a flexible disc file, using the disc file management system. The time of day was also stored on the disc, as three binary-coded-decimal numbers. Each disc record therefore contained four bytes of data. The disc file records were subsequently analysed and used to produce the results discussed in the next section.

6.5 Experimental Results

A typical printout of the results obtained during the experiment is shown in figure 6.3. Each printer column to the right of the time corresponds to one of the 64 frequency windows. The magnitude of the response obtained for each window is printed as a single hexadecimal digit in the range 1-F. A magnitude of 0 corresponds to a clear window, for which nothing is printed. Two narrow-band interfering signals can be seen at 1550 Hz and at 2480 Hz. Another (weaker) signal appears at 2300 Hz.

Data for each half-hour interval was analysed and the results were plotted of the number of windows in which **any** interference was detected during the half-hour period. ie. in which the interference level exceeded the previously defined threshold. The results (figure 6.4) are plotted as a percentage of the total number of windows vs. Universal Time for a 24 hour period. For example, between 0330 and 0400, interference was detected in 70% of the total channel bandwidth at some time during the half hour. The data for this experiment was collected during the period 10th-17th March, 1980.

The distribution of interference within the channel was observed to vary considerably over the 24 hours. A high proportion of the channel was occupied during the hours of darkness, when the receiver was subjected to interference from distant sources. A "quiet" interval of about 4 hours was observed around mid-day when no interference was detected.

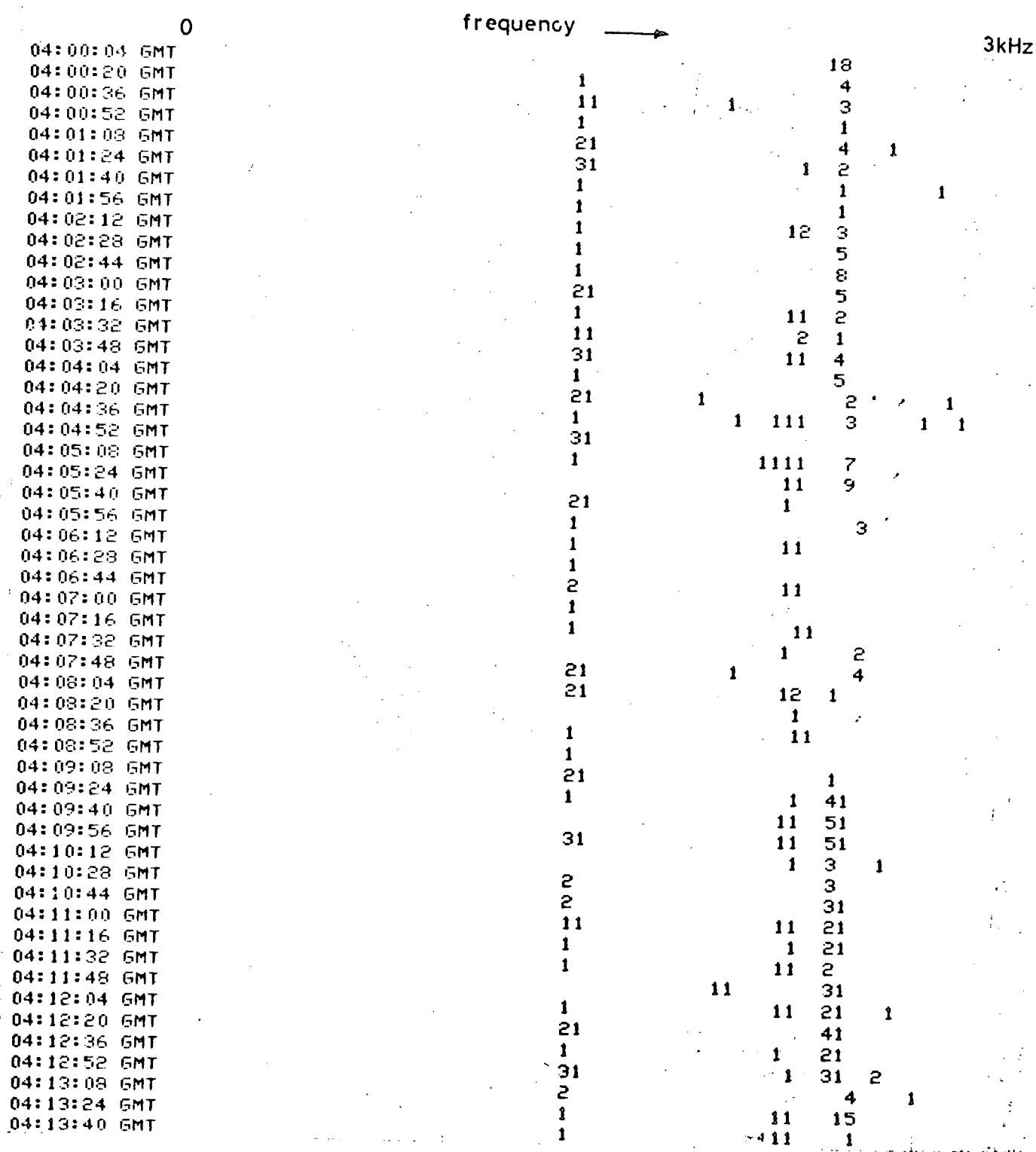
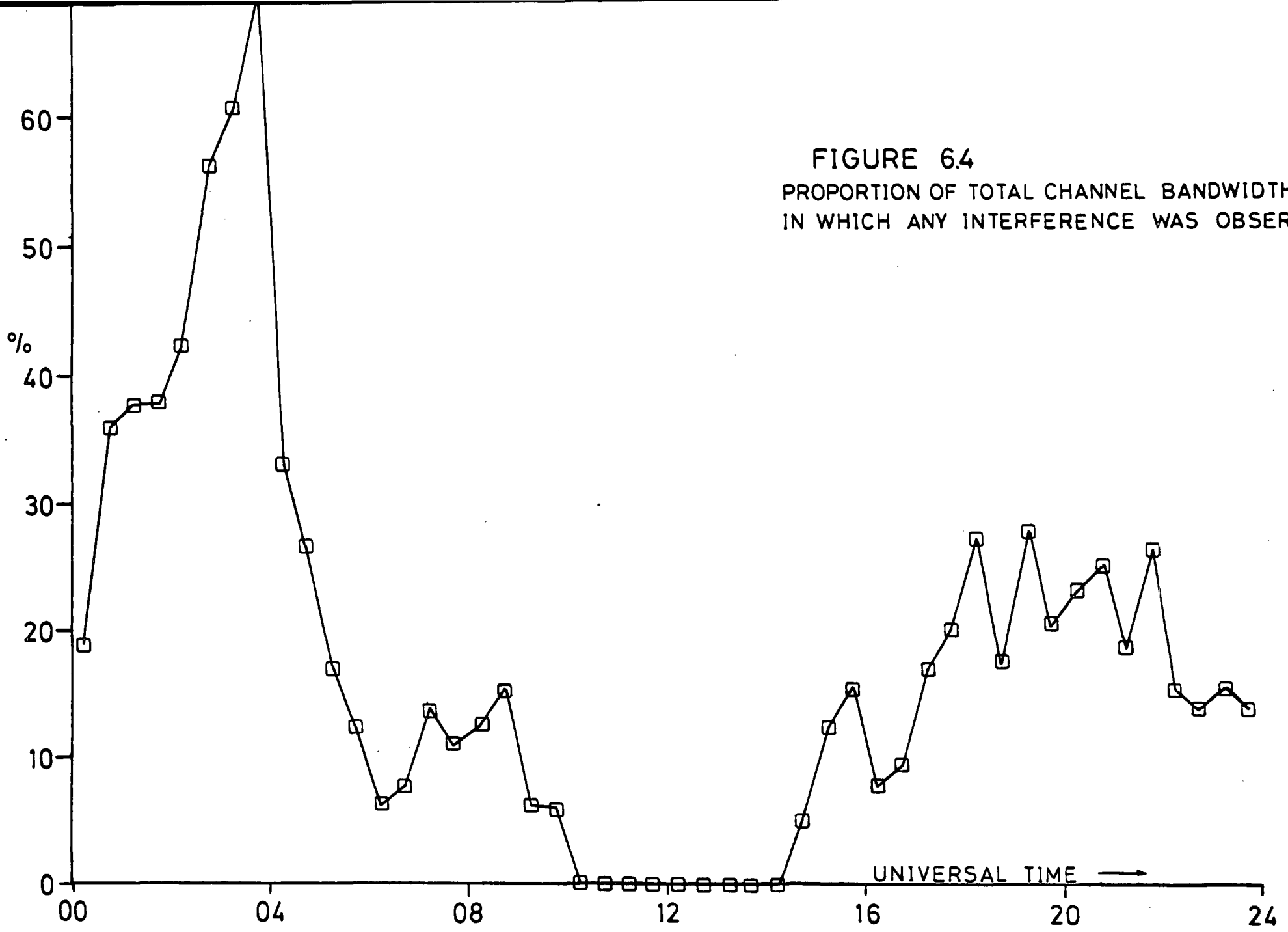


FIGURE 6.3. OUTPUT DATA PRINTOUT.

FIGURE 6.4
PROPORTION OF TOTAL CHANNEL BANDWIDTH
IN WHICH ANY INTERFERENCE WAS OBSERVED



The stored data was analysed to determine the proportion of time that the channel was completely free from interference; ie. (subject to propagation being available) the proportion of time that the channel could be used for transmission to its full capacity. The results were plotted for half-hourly intervals and are shown in figure 6.5. eg. between 0430 and 0500 hrs the channel was found to be completely free from interference for 25% of the time. Interference was heavy during the night and the channel was occupied to some extent for >95% of the time. There appeared to be a steady improvement in the interference-free time between 0400 and 0630, followed by a slight deterioration and another improvement leading to the 4 hour "quiet" time. This was followed by a gradual deterioration after 1430, probably caused by a lengthening of the skip distance. A noticeable characteristic of the plotted results is that the changes from one half-hour to the next are gradual, with no abrupt transitions.

The stored data was again analysed, to obtain more information on the spectral distribution of the interference within the channel. The proportion of time during which n or more frequency windows were occupied are plotted, for each half-hour interval, in figures 6.6 ($n=1,2,3$) and 6.7 ($n=4,5,6$) (both plots are on the same scale). Note that the result for $n=1$ is simply the inverse of the result of figure 6.5. The percentage falls off rapidly as n increases and becomes very small for $n > 6$. This is indicative of the predominance of narrow-band interference. The results were combined to produce an overall probability of spectral occupancy against bandwidth

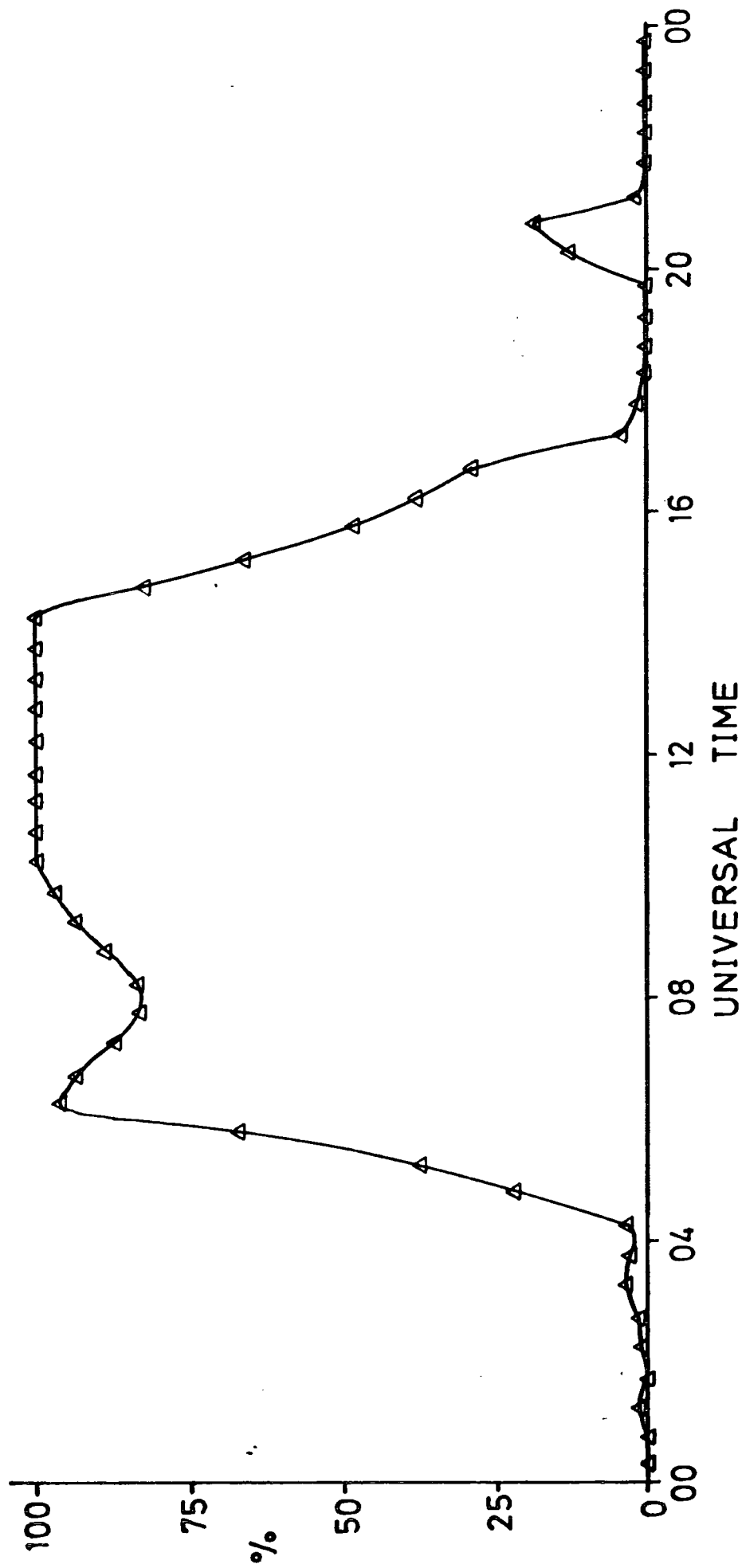


FIGURE 6.5. PROPORTION OF TIME DURING WHICH CHANNEL WAS CLEAR.

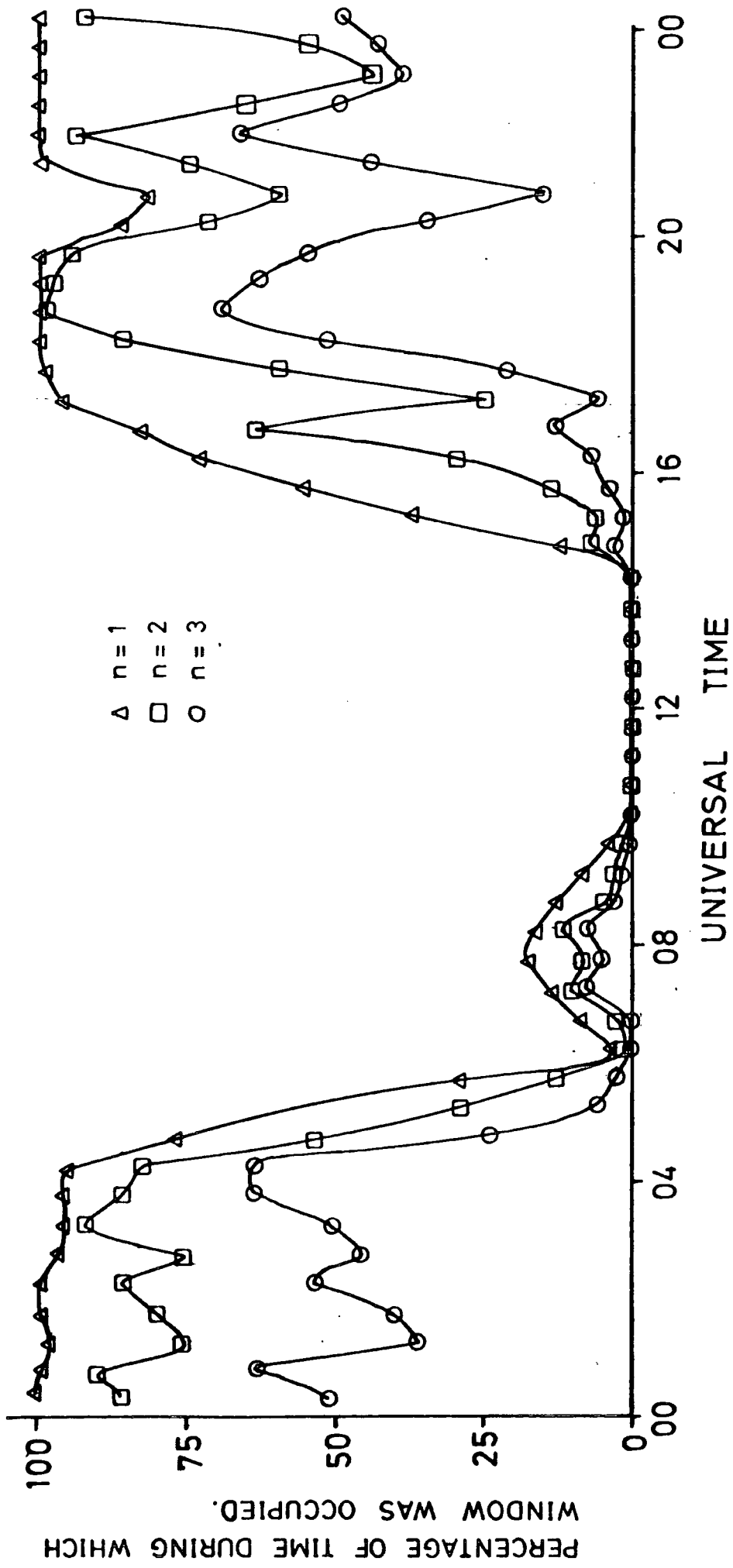


FIGURE 6.6. WINDOW OCCUPANCY

PERCENTAGE OF TIME DURING WHICH WINDOW WAS OCCUPIED.

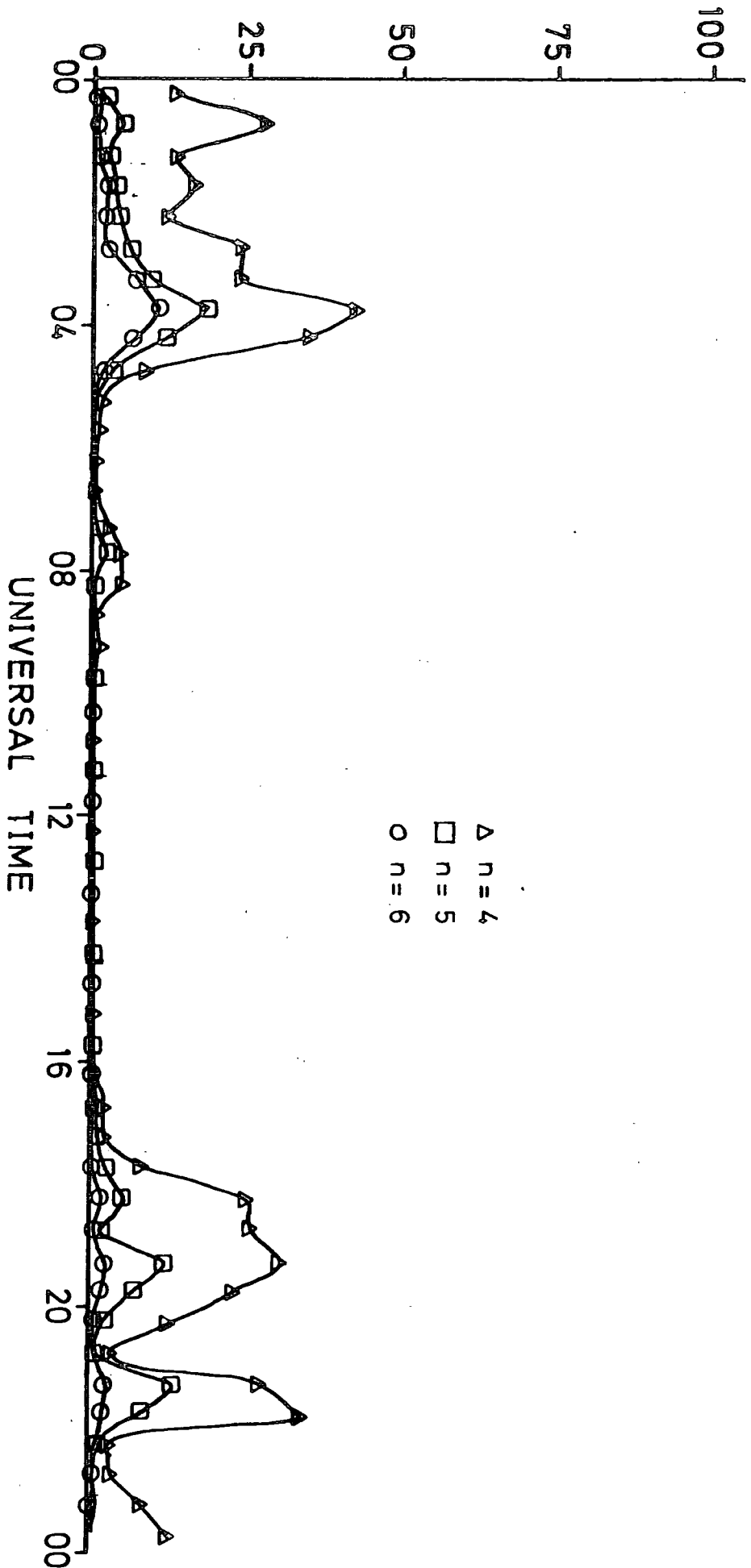


FIGURE 6.7. WINDOW OCCUPANCY

(figure 6.8). Because of the finite width of the frequency windows, the results are plotted as bar graphs, with discrete points at $n=0$ indicating the probability of a completely free channel. The results appeared to fall into two distinct regions; low interference levels were observed for an eight-hour interval during the daytime; high interference levels were observed during other times, when the skip distance was greater. Results were plotted for these two extremes ie. "day" (0600-1430), and "night" (1430-0600), respectively, and also for a 24 hour average. It can be seen that the probability of a clear channel is much greater during the day, and that at night there is a high probability that narrow-band interference will be present.

6.6 Conclusion

This chapter has described an experiment undertaken, using a microprocessor system and a charge coupled device, to observe the characteristics of interference occurring in an HF radio voice channel. It has been shown that there is a high probability that the channel will contain interference of a narrow-band nature, and that the interference distribution may vary considerably with time, becoming quite severe during the night hours. It is highly probable ($> 97\%$) that the bandwidth of the interference will be narrow (< 200 Hz); however, the distribution of the interference is likely to change rapidly, especially during the night. This confirms the view that benefit may be obtained by using a frequency-agile transmission system to avoid those regions of the transmission channel containing interference.

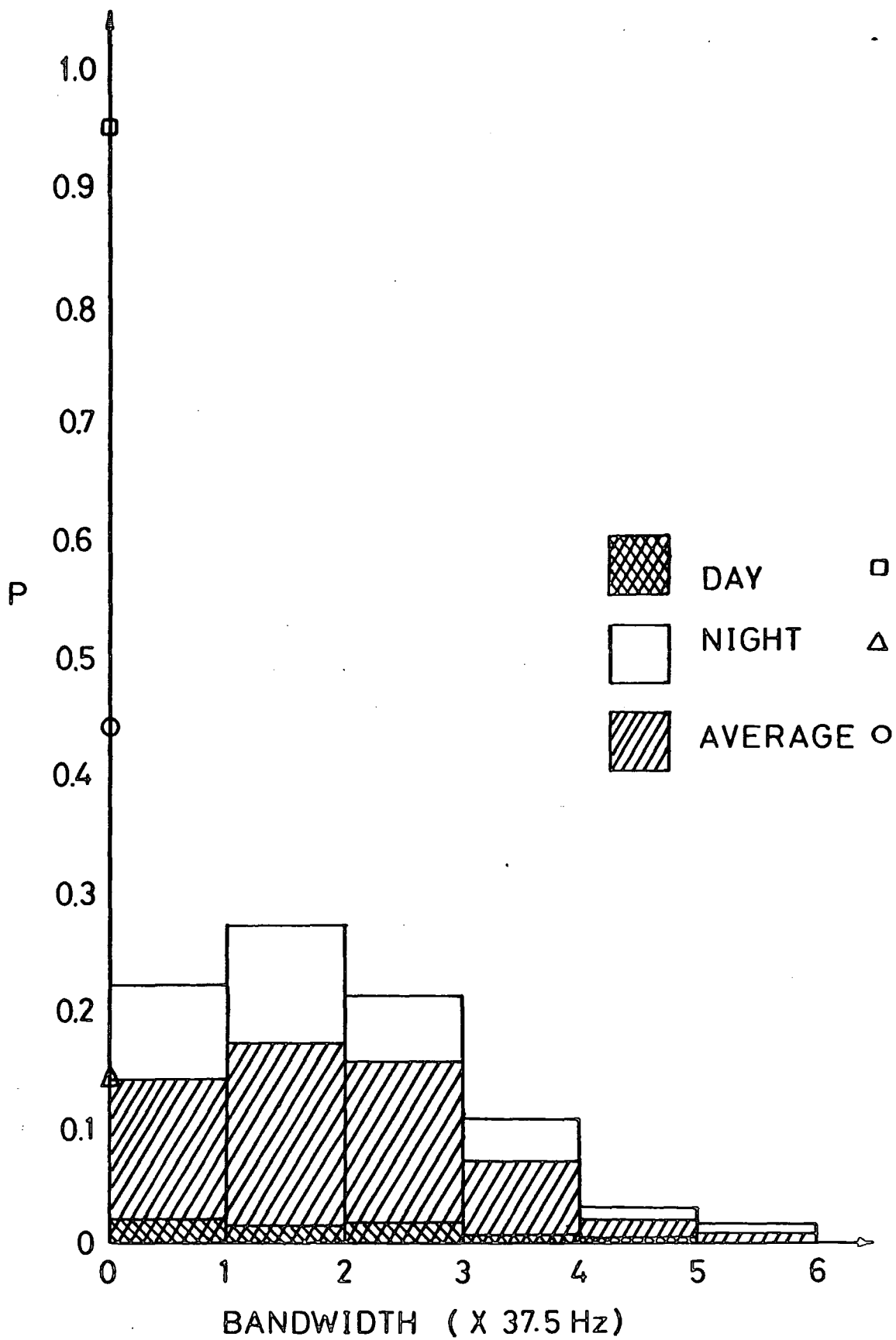


FIGURE 6.8 PROBABILITY OF SPECTRAL OCCUPANCY

7.1 Introduction

The resurgence of interest in digital communications via HF radio links has led to a renewed search for low-cost modems suitable for data transmission over ionospheric paths. As previously discussed, signals propagated over the HF path are plagued by noise and multipath distortions which may often result in intolerably high error rates (55). The time-varying nature of the disturbances imply that some form of adaptive control is required if the errors are to be eliminated.

The suitability of microprocessor systems for the implementation of signal processing and coding techniques has been demonstrated in chapters 2 and 5. Microprocessors are also extremely attractive for use in an adaptive environment as the course of execution of machine instructions may be made dependent on previous and current events. Non-adaptive modems based on these devices have been demonstrated in the literature (56-59), as have some adaptive schemes which require feedback links from the receiver to the transmitter (60).

This chapter describes the design and development of a medium-speed adaptive HF data modem/codec, based on microprocessor-implemented signal processing and coding techniques, which does not require the use of a feedback link. Results from earlier chapters were considered when formulating the design, which may be implemented at an appreciably lower cost than previous systems.

It has been shown that a primary source of errors over HF radio links is narrow-band interference from other users of the spectrum. The spectral distribution of the modulated output signal from the modem is arranged to occupy the interference-free regions of the radio voice channel. Forward error correction is applied in an attempt to combat transient broad-band disturbances. The modem is described in the published paper of Appendix 3 (reference 61).

7.2 System philosophy

It has been shown (62,63) that the optimum frame (signal element) duration for transmission over a dispersive medium is equal to $\sqrt{L/B}$, where L is the time spread introduced by the medium and B is the frequency spread. For HF channels, both the time spread caused by multipath and the frequency spread caused by Doppler shifting may vary considerably and may depend on the time of day and on the operating frequency. For a medium haul HF link, the time spread may be in the order of several milliseconds, and the Doppler shifting may be of the order of a few Hz. Insertion of typical values in the above formula yields frame rates in the order of tens of Hz. A frame rate of 75 Hz has been found to be a reasonable compromise; the signal element duration is of sufficient length to combat the effects of multipath distortion, yet is short enough to ensure that minimal phase distortion occurs over a single frame. Serial binary-modulated data transmission schemes are therefore limited to a data rate of 75 bps, which does not fully utilise the available bandwidth.

It is possible to increase the transmission rate while preserving the frame rate by time division multiplexing the data for transmission over a number of frequency-parallel subchannels, orthogonally spaced within the voice channel. The data rate is increased by a factor equal to the number of subchannels employed. The spectral distribution of the transmitted signal then depends on the location of the subcarriers. Capacitative coupling and other band-limiting effects inherent in conventional HF radio equipment usually restrict the useable region of an HF voice channel to between 300 Hz and about 2.8 kHz. Sixteen orthogonally spaced slots were allocated within this region for the location of the subcarriers, ie. from 450 Hz to 2700 Hz, with 150 Hz spacings.

It would be possible to utilise the channel to its full capacity by employing subcarriers at all adjacent orthogonal frequency slots. However, results from chapter 6 of this thesis and from work in the literature (53) indicate that there is a high probability that the channel will contain interference, and it is therefore preferable to distribute the signal spectrum in the noise-free regions only. It has been shown that the interference is predominantly narrow-band, but that its distribution may vary rapidly with time. In view of this, a 50% spectral occupancy was adopted, the subcarriers occupying one-half of the available subcarrier slots.

It is also possible to increase the transmission rate while preserving the frame rate by employing a Q-ary modulation scheme (64). In the binary case, it is practical to use only values of

Q which are integer powers of 2, for which the transmission rate is increased by a factor $n = \log_2 Q$. The penalty paid is the increase in vulnerability to noise, which (for PSK) is calculated to be 3dB and 8.5dB for $Q=4$ and $Q=8$ respectively (65). 4-phase modulation (QPSK) can therefore be used to double the effective data rate with only a small reduction in tolerance to additive noise (66) and has been used successfully for data transmission over HF links (67-69). However, because of the phase perturbations observed over such links (70), coherent detection of such a signal becomes virtually impossible and differential phase encoding must be used. This imposes an additional degradation of 2.3 dB for QPSK (71) resulting in a total of 5.3 dB degradation over the optimum coherent biphase detector. The differentially modulated QPSK signal (DQPSK), however, has a spectral occupancy identical to a serially modulated biphase system and gains an advantage in this respect.

The results presented in chapter 6 have indicated that a reduction in the error rate might be obtained if the signal spectrum is adjusted to suit the prevailing channel interference conditions. A system in which the frequencies of the signal carriers are dynamically adjusted is called a "frequency agile" system. A serial in-band frequency-agile FSK modem has been demonstrated by Darnell (54), and has been found to yield improvements over the standard tone allocations. The modem described in this chapter estimates the spectral distribution of the interference present in the voice channel at intervals, and reallocates the distribution of the subcarriers accordingly. An assumption was made that, over medium distances (1000 km. or

less), the interference characteristics at the receiver will be roughly similar to those observed at the transmitter. This assumption was later proved to be correct, as will be demonstrated. The noise estimation may therefore be carried out at the transmitter site, and no feedback link is required. The receiver must be advised, however, as to the new distribution of the transmitted spectrum. This is achieved by transmitting a high-redundancy "advisory sequence" on a set of subcarriers having a fixed frequency allocation.

The receiver demodulator must distinguish the transmitted tones and extract the phase information for each tone. It would be possible to use a bank of conventional narrow-band filters to perform the demodulation process. However a frequency agile system would require a large number of filters, of which only a few would be in use at any one time, making the system uneconomical. It is preferable to use digital processing techniques to implement matched filter detection by means of the Discrete Fourier Transform. Phase information for each subchannel may be obtained by observation of the complex coefficients of the DFT slots corresponding to occupied subchannel frequencies. Comparison detection allows the differential phases (and hence the data) to be extracted without the need for a pilot tone or a locally generated reference phasor.

The design, construction and testing of the transmitter is now described, followed by a discussion of the receiver design.

7.3 Transmitter

A block diagram of the transmitter is shown in figure 7.1, the constituent components of which are discussed in this section. The master processor system includes a 6800 CPU, 4 kbytes of RAM and 2 kbytes of EPROM containing the system software, a listing of which may be found in appendix 2.

The tasks performed by the transmitter fall into four broad categories:

- (1) data acquisition
- (2) encoding for error control
- (3) modulation
- (4) channel evaluation & subchannel selection

It was found that a single 6800 microprocessor system was not able to perform all of the required tasks in the available time. One of the main reasons for this was that the modulating waveform must be generated continuously, while simultaneously encoding incoming data. Time-sharing of tasks would inevitably lead to breaks in the analogue signal if this was generated in software. In view of this, a 'master-slave' configuration was used, as described in chapter 4. A 'slave' processor unit was allocated the task of modulation, and the remaining tasks were performed by the 'master'. The advantages of this approach have already been discussed.

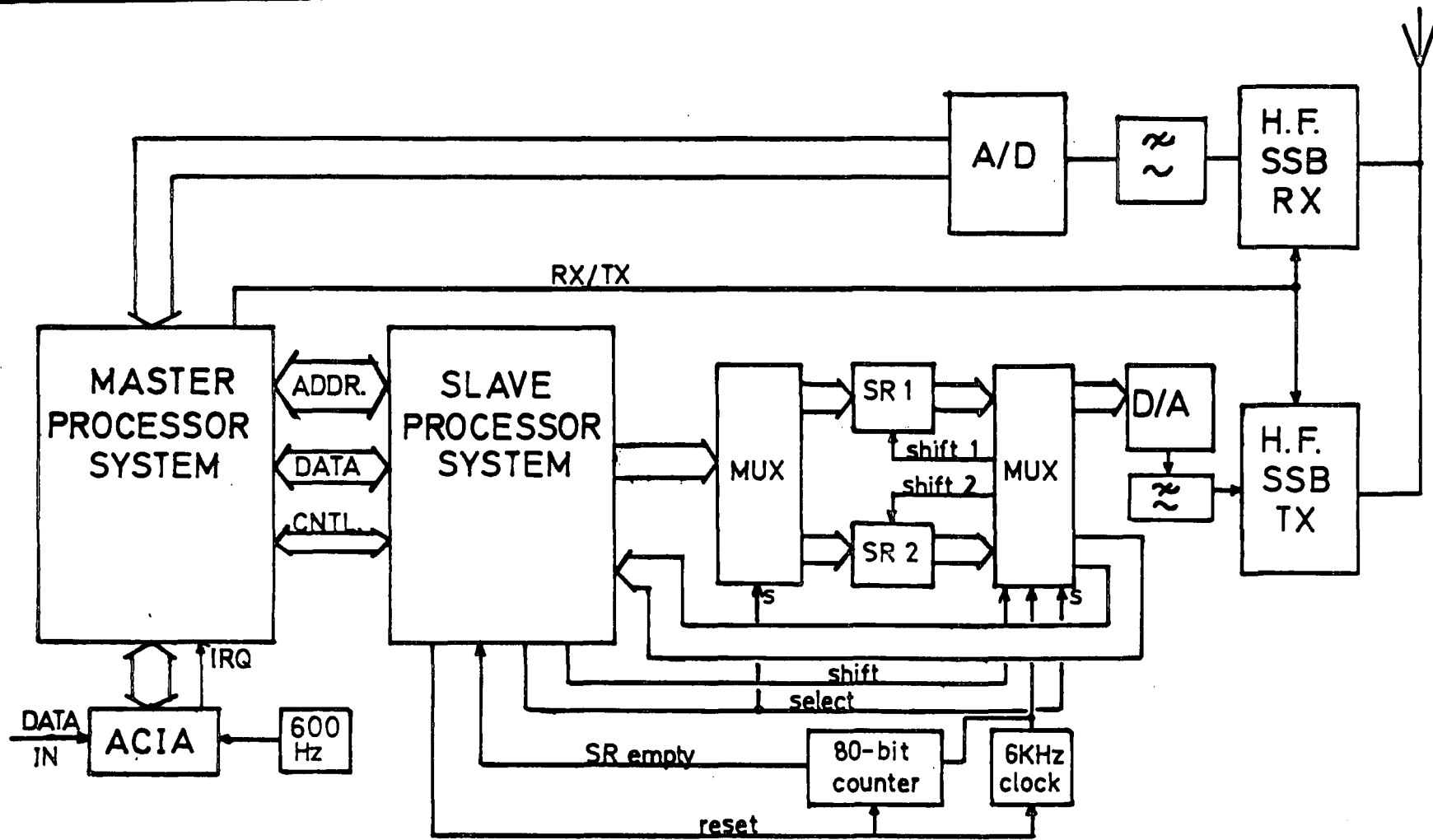


FIGURE 7.1. TRANSMITTER BLOCK DIAGRAM.

7.3.1 Data acquisition

An asynchronous serial interface IC was available which allowed 7-bit characters having start and stop bits to be received and transmitted at a variety of data rates. This IC was used in the system to accept characters at a rate of 600 bps. The interface control register was programmed by the system software such that an interrupt was generated on receipt of a complete character. Each 7-bit character was mapped into a (15,7) BCH codeword (see chapter 5), and was further processed to allow interleaving of 16 codewords along each of the 16 data subchannels. It was necessary to receive 256 characters before transmission could commence (because of the interleaving requirement), after which time characters were accepted continuously via the interrupt driven input.

7.3.2 Encoding for error control

The forward error correction scheme chosen for this system was a (15,7) block code interleaved to depth 16 along each of the 16 data channels. The code has been fully described in chapter 5 of this thesis and the improvement in error rate obtained in practice using this scheme will be demonstrated in the next chapter. Each data character received through the serial interface was coded on receipt and the resulting codeword was loaded into a table of 16 codewords contained in 32 bytes of memory. When the table was full, the bits were interleaved and the whole table was transferred to one of 16 tables containing data ready for transmission via the modem. Each one of these 16 tables corresponded to each of the data channels; a pair of tables therefore corresponded to one subchannel frequency, the

bits in the tables being used to select the differential phase for each signal element. The organisation of the data in time-frequency space is shown in figure 7.2., where $c_{i,j}$ is the j th bit of the i th codeword.

7.3.3 Modulation

Differential four-phase (quaternary) shift keying (DQPSK) was adopted as the modulation scheme for the system. The z-plane representation of a suitable phase encoding scheme is shown in figure 7.3 where the axes form the decision thresholds and the bit-pairs corresponding to the phases are arranged in a Gray code around the unit circle to minimise the bit error probability.

Each subcarrier conveys two bits of information per signal element and there is a total of eight subcarriers to be located in a possible 16 orthogonally spaced frequency slots. 16 bits are therefore transmitted per signal element, which are divided into 8 pairs; a pair of bits determines the phase on the corresponding sub-channel. The signal over one element may be described as:

$$f(t) = \sum \cos (\omega k_c t + \theta_c) , \quad 0 \leq t < 13.33 \text{ ms.}$$

where $\omega = 2\pi 150$ Hz, k_c is an integer between 3 and 18 and θ_c is the phase corresponding to subchannel c (c is an integer). θ_c may have values $\pi/4, 3\pi/4, 5\pi/4, 7\pi/4$.

The composite modulating signal was to be generated digitally, then low-pass filtered to confine the spectrum to the

SUBCHANNEL
NUMBER

DATA BITS

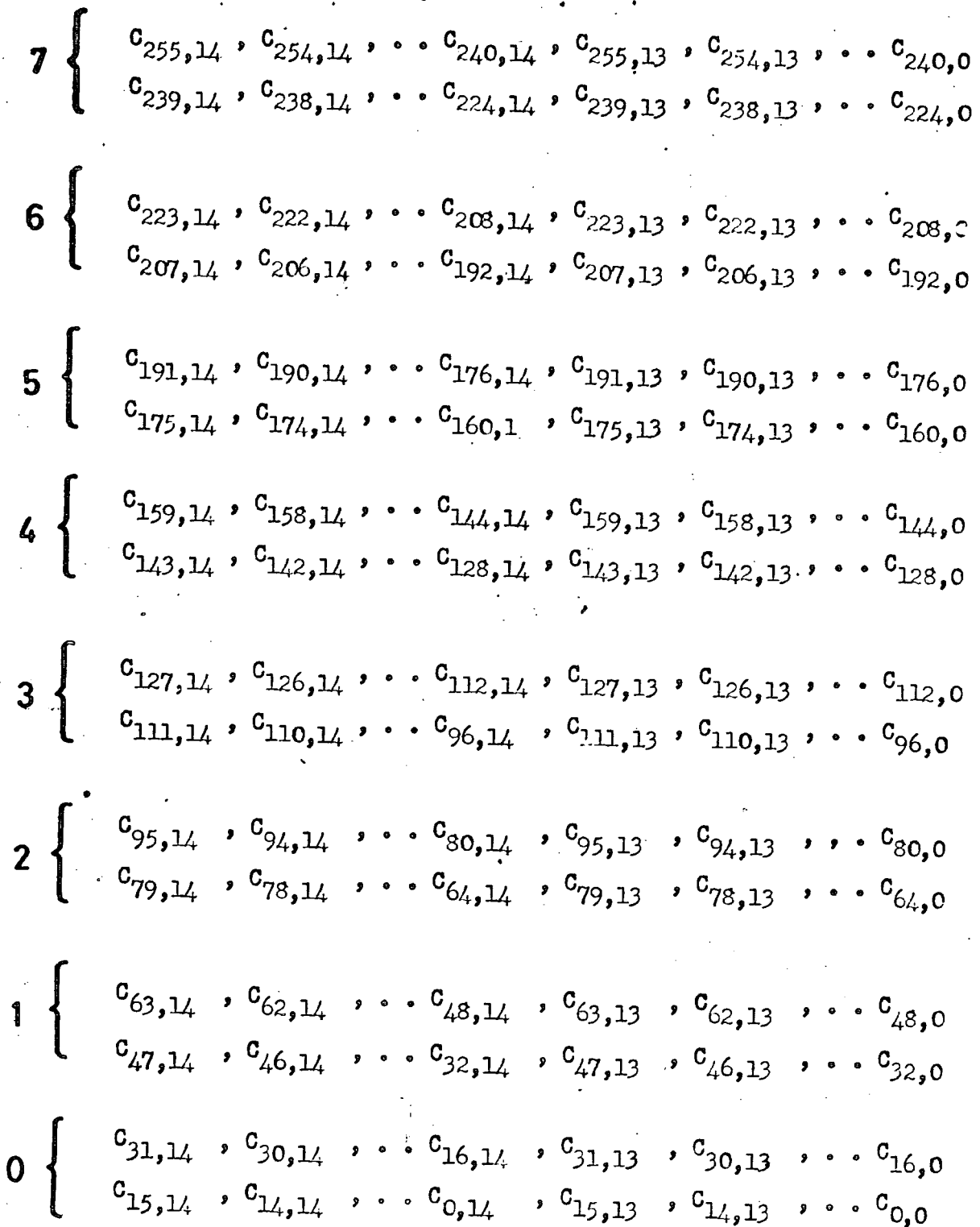


FIGURE 7.2. DATA ORGANISATION IN
TIME-FREQUENCY SPACE.

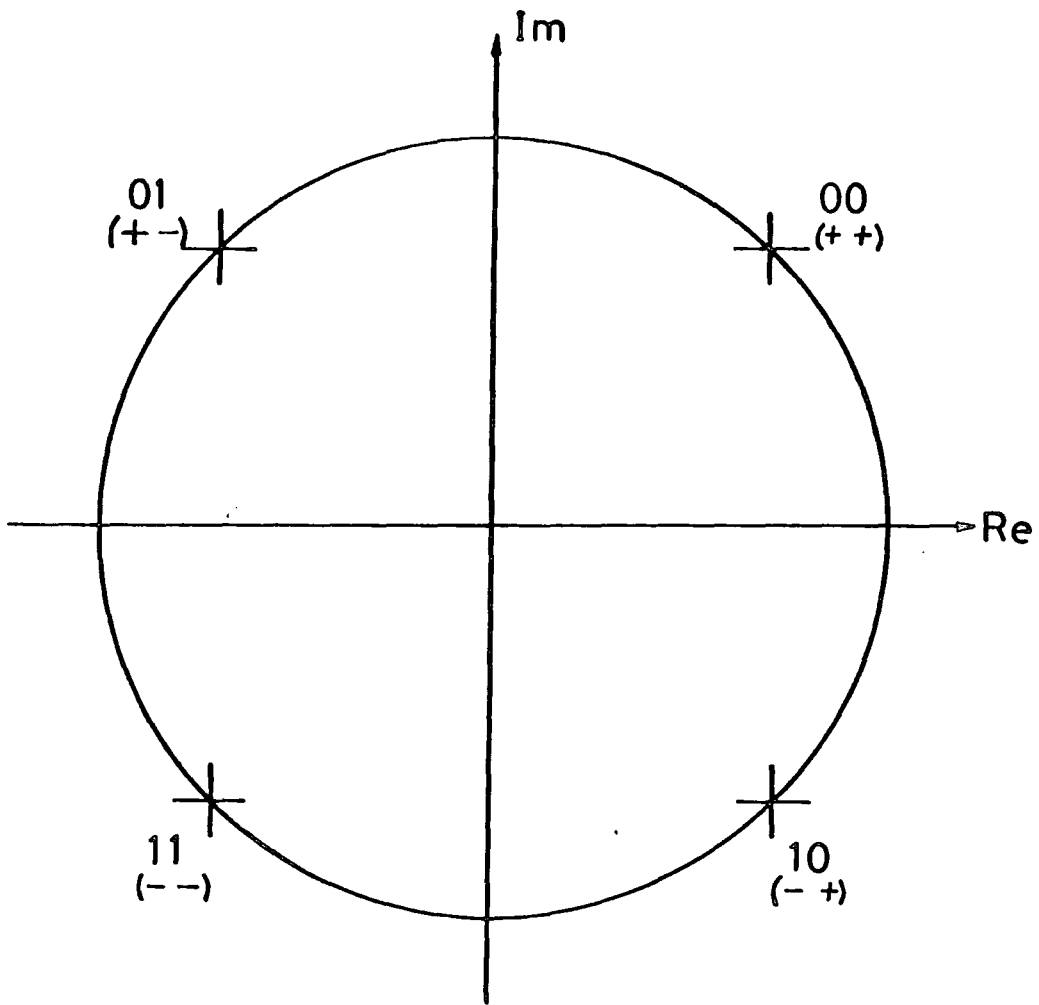


FIGURE 7.3. Z-PLANE REPRESENTATION OF QPSK.

required signal band. A lookup table stored in the slave memory was used to generate samples of this signal. The table consists of N samples of a cosine waveform, equally spaced in time, each of which is stored as an 8-bit 2's complement number. If consecutive samples from the table are output at a sampling rate of f_s , the fundamental frequency of the resulting waveform is f_s/N . Multiples of this frequency may be generated by stepping through the table at different rates, so that if the step length is l , the frequency generated is lf_s/N . The table may be regarded as circular; the index of the current sample is always calculated modulo- N . Nyquist's sampling theorem requires that at least 2 samples per waveform cycle must be available to define a frequency, implying that the sampling rate must be at least twice the frequency of the highest component. For this application a sampling rate of 6kHz is sufficient to guarantee this condition, and if frequencies are to be generated in multiples of $f_c = 150$ Hz (to satisfy the orthogonality constraint), the length of the lookup table must be $N = f_s/f_c = 40$ samples.

The frame rate of the transmitted signal is one-half of the frequency spacing so that a complete frame is generated using 80 samples at the 6kHz sampling rate. The phase of a particular subchannel is determined by the starting point in the lookup table. Phases of $n\pi/4$ ($n = 1,3,5,7$) are required, corresponding to the dibits 00,01,11,10 respectively. The required phase is generated by starting at sample number $nN/8 = 5n$.

As an example of the preceding discussion, suppose it is required to generate a subchannel on 900 Hz having a phase of

$5 \cdot \pi/4$ radians. The step length is $900/f_c = 6$ and the starting point in the table is 25. The first 10 indices of the 80 samples required to generate the subchannel are then:

25,31,37,3,9,15,21,27,33,39

Modulator circuit

Consideration was given to the possibility of generating the composite multitone waveform using a purely software approach. One method would be to use an interrupt processing routine to generate a new sample of this waveform at each sampling period. However, the overhead required in adopting this method represented a considerable proportion of the overall processing time, and it was not possible to complete the required processing in the available baud time of 13.33 ms. Additional hardware was therefore designed and built to reduce the computational requirements of the modulation process. Operation of the modulator hardware is described in this section. A circuit diagram is shown in figure 7.4.

The modulator circuit is interfaced to the slave processor via two PIAs (Peripheral Interface Adapters). That to the left of the circuit diagram is referred to as the 'input PIA' and that to the right is the 'output PIA'. Twelve of the 16 I/O port lines from the input PIA are connected to the inputs of four hex tri-state buffers, arranged as two pairs. The buffer outputs are connected to the inputs of six quad 80-bit static shift registers, arranged as two sets of three, subsequently referred to as SR1 and SR2. The shift register outputs are connected to six 8-to-4 line multiplexers, also arranged as two sets of three. The outputs

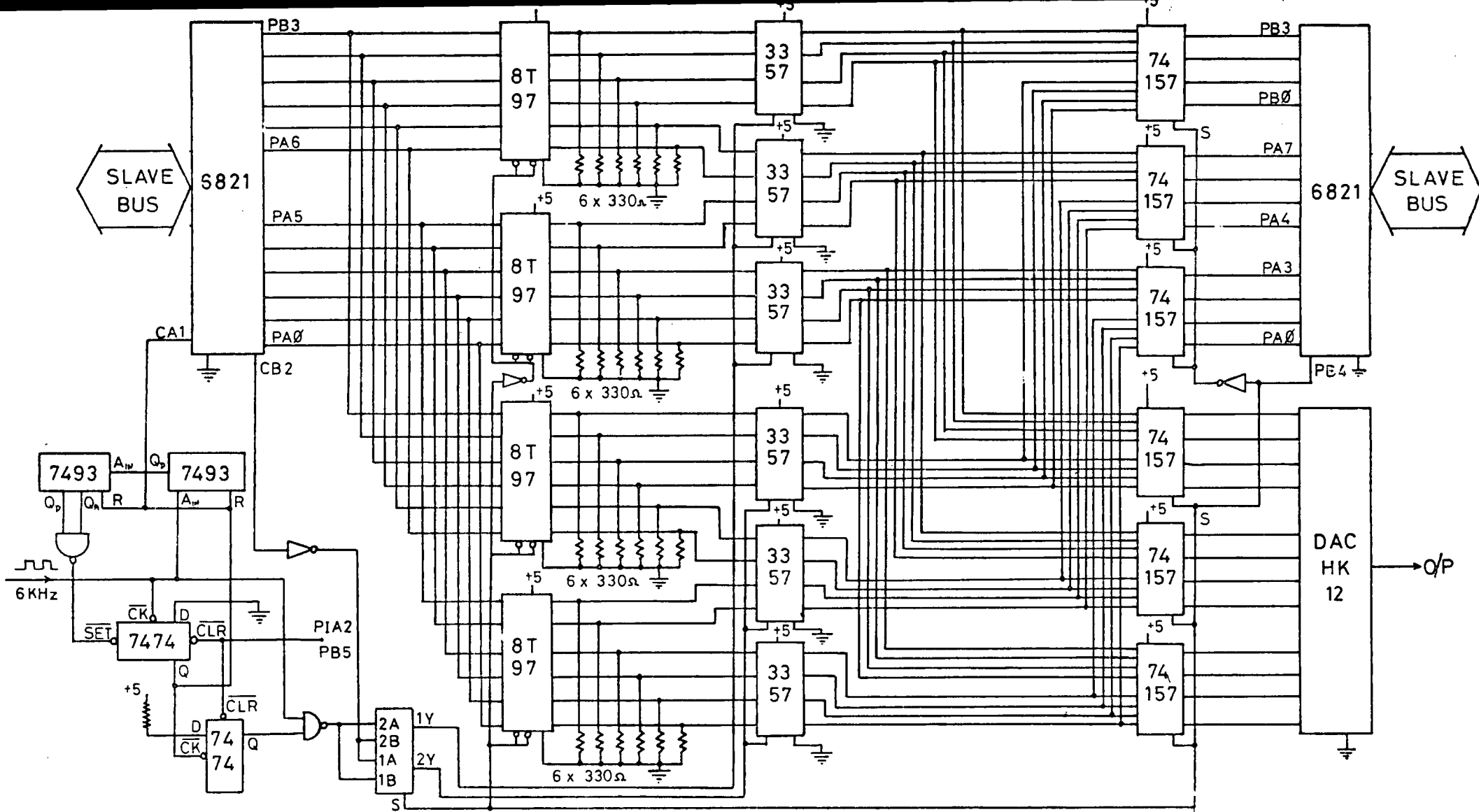


FIGURE 4.4. MODULATOR CIRCUIT DIAGRAM

from the first set of multiplexers are connected to twelve I/O lines of the output PIA, and the outputs from the second set are connected to the inputs of a 12-bit digital to analogue converter. The output from a 6kHz square wave generator (implemented using a crystal oscillator and a divider circuit) is connected to the input of two cascaded 4-bit binary counters. The Q_A and Q_D outputs from the most significant counter are NANDed together and are used to 'set' a D-type bistable on the count of 80. The output from this bistable then resets the counters and is used to interrupt the slave processor unit via the CA1 line on the input PIA. The remaining bistable in the 7474 IC is used to enable the 6kHz sampling clock to the input of a multiplexer. The inverted signal from the CB2 line of the input PIA is also connected to the input side of this multiplexer. The multiplexer is configured so that the 1Y output is derived from the CB2 line while the 2Y output is derived from the sampling clock, and vice versa.

The enable lines to the data buffers are inverted with respect to each other so that while one pair is enabled, the other is not. The outputs of the pair that is not enabled are pulled to logic '0' by the resistors. ^(automatically being shift reg) The select lines on the multiplexers are also inverted with respect to each other so that the outputs from the shift registers whose inputs are derived from the enabled data buffers are routed to the output PIA, while the outputs from the other shift registers are routed to the D/A converter. The 'shift' signal for the former is obtained from the CB2 output line of the input PIA, via the remaining multiplexer, and the shift signal for the latter is derived from the sampling clock.

After loading, the slave processor memory contains the program for control of the modulation, a 40-point cosine lookup table, and two 8-byte tables containing the step lengths and starting points for each subchannel. Operation of the circuit is as follows:

The multiplexers are switched so that the PIA output lines are routed into the input of register SR1 and the outputs of SR1 are routed into the input PIA. Samples for the first subchannel are selected from the lookup table using the corresponding step length and starting point. Each sample is written into the eight least significant bits of SR1; the register is full after 80 "write" instructions. The first sample entered is then read from the shift register output, added to the first sample of the next subchannel and the result (now 9 bits) is written back into the shift register. After 80 shifts the register contains the sum of the samples for the first two subchannels. The procedure is continued for all eight subchannels. The multiplexers are then switched so that the "shift" line to SR1 is derived from the 6kHz clock and the shift line for SR2 is derived from CB2. The output from SR1 is now routed to the D to A converter and the register contents are shifted out at the sampling rate of 6 kHz. During this time the register SR2 is loaded with samples for the next frame in a similar manner. When SR2 is full, the multiplexers are again switched and the contents are routed to the D to A converter. During one frame, therefore, the contents of one register are shifted out to the converter at the sampling rate while the slave processor is constructing samples for the next frame using the other shift register. When all the samples in a

register have been shifted out, the counter is used to interrupt the processor which then switches the multiplexer select line and begins to construct the next frame.

The frequencies and phases of the subchannels for each frame are determined by the contents of two 8-byte tables which contain the step lengths and starting points respectively. There are actually two pairs of tables; the contents of one pair are used by the slave to compute samples for the frame currently under construction, while parameters are loaded into the other pair from the master for use by the slave during the following element. During one frame, therefore, the master has control of one pair of tables while the slave has control of the other. Upon receipt of an interrupt from the slave (indicating completion of transmission of a frame), control of the tables is reversed.

To summarise the preceding discussion, three processes occur simultaneously during one signal frame. Samples for the current frame are shifted into the D-A converter from one set of shift registers by the 6kHz sampling clock; samples for the next frame are computed by the slave in the second set of shift registers with phase and frequency information obtained from one pair of tables. Frequency and phase information for the third frame is passed from the master into the second pair of tables in the slave.

7.3.4 Channel evaluation & subchannel selection

Following each message transmission, the HF transceiver is switched to the 'receive' mode (by means of a Tx/Rx reed relay), and the audio output of the receiver is sampled by the A-D converter. 64 samples are acquired at a sampling rate of 9.6 KHz and an in-place FFT is computed (see chapter 2). The resulting DFT frequency slots are therefore in multiples of 150 Hz, and the power spectrum of the slots from 450 Hz to 2700 Hz inclusive (the 16 available subchannel slots) is estimated from the phasor magnitudes. The sampling and transformation processes are repeated 8 times and the resulting power spectra are averaged to provide a reasonable estimation of the noise present in each subchannel slot over an observation interval of approximately 3.5s. This represents only 2% of the overall transmission time and therefore does not significantly affect the data rate. A number sorting routine (CHSORT) is then used to select the eight "quietest" slots for subsequent transmission. The quietest slots are taken to be ~~the~~ those eight which exhibit the least average interference from the total of sixteen available slots. A typical distribution of the signal spectrum is illustrated in figure 7.5.

In order that the receiver may be advised as to the subchannel frequencies to be used for subsequent data transmission, a coded sequence containing this information is transmitted immediately before the frequency change is effected. The sequence comprises eight (15,7) BCH codewords; the information section of each is formatted as follows:

$$n_2 \ n_1 \ n_0 \ s_3 \ s_2 \ s_1 \ s_0$$

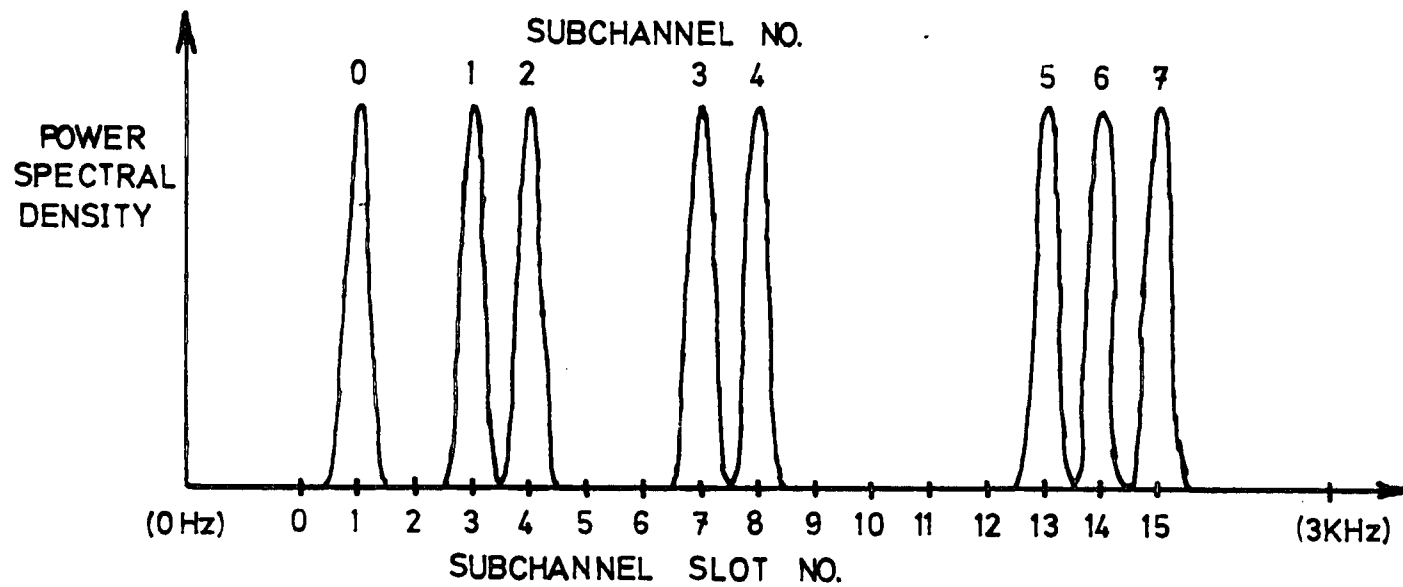


FIGURE 7.5. TYPICAL SPECTRAL DISTRIBUTION OF SUBCHANNELS.

The vector $\mathbf{n} = (n_2 n_1 n_0)$ represents a subchannel number in the range 0-7 and the vector $\mathbf{s} = (s_3 s_2 s_1 s_0)$ represents a subchannel of frequency $(450 + (s \times 150))$ Hz. A code vector \mathbf{c} is found from the matrix operation:

$$\mathbf{c} = [\mathbf{ns}] \mathbf{G}$$

where \mathbf{G} is the code generator matrix.

The code is fully described in chapter 5. The set of 8 code vectors are arranged in time-frequency space as illustrated in figure 7.6, where $c_{i,j}$ is the j th bit of the i th codeword. It can be seen that pairs of codewords are interleaved to depth 2 along each data channel allowing correction of 4 errors per data channel. Because of the 4-phase modulation scheme there are two data channels per subchannel frequency, resulting in a total of 16 data subchannels. For each signal element, the differential phase transmitted on a subchannel is determined by the corresponding dibit. There is a four-fold spectral redundancy in the transmitted data which compensates for any narrow-band interference which may be present. The advisory sequence is always transmitted on a set of subchannels having a fixed frequency allocation. Transfer to the new subchannels occurs immediately after the advisory sequence transmission. If a fixed frequency allocation was not used, a situation might arise where an error occurs in the decoding of the advisory sequence, resulting in the loss of all subsequent messages. The fixed allocation allows the receiver to recover after a single message block, since the location of the subchannels for the advisory sequence is always known.

SUBCHANNEL
FREQUENCY

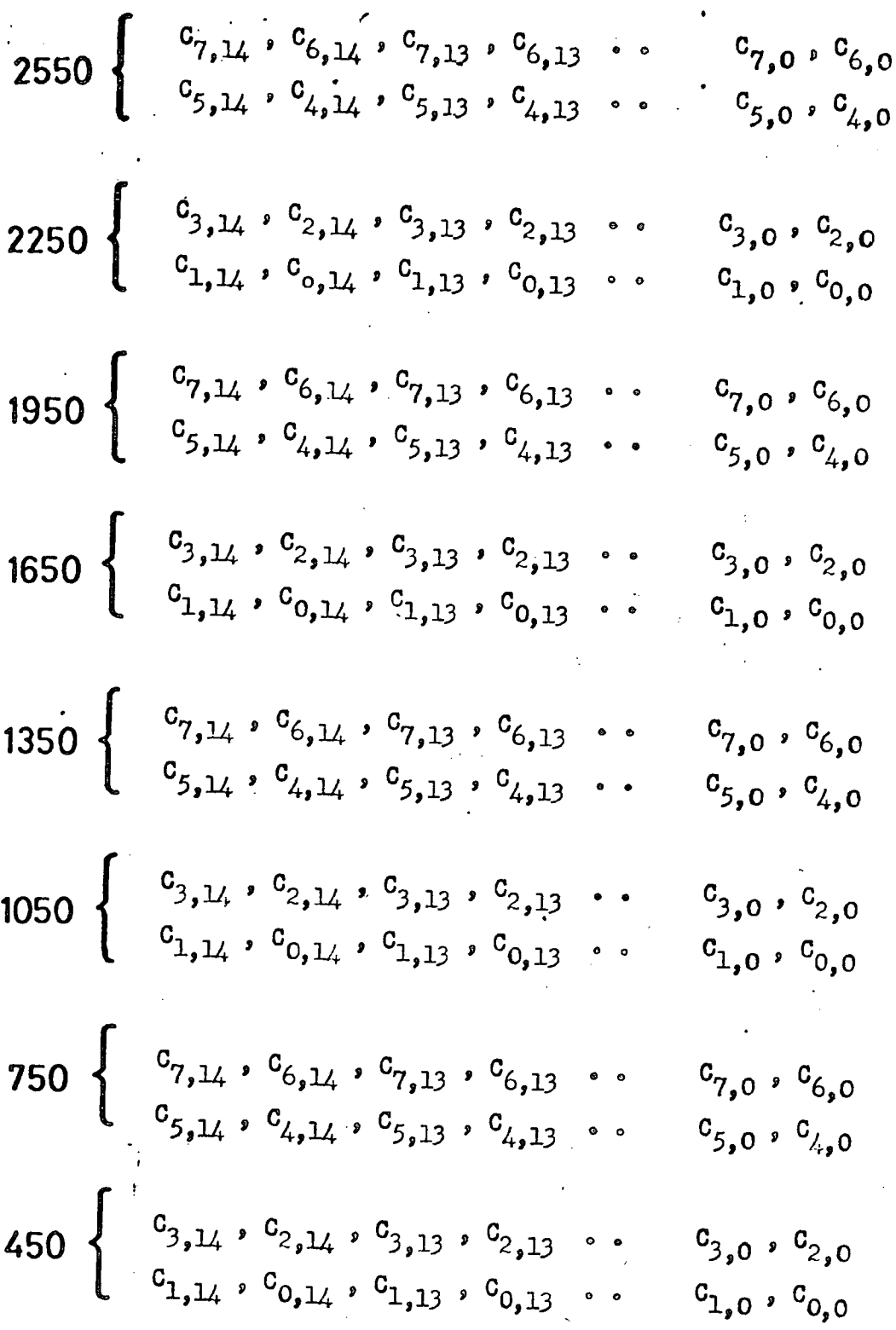


FIGURE 7.6. ADVISORY SEQUENCE FORMAT IN
TIME-FREQUENCY SPACE.

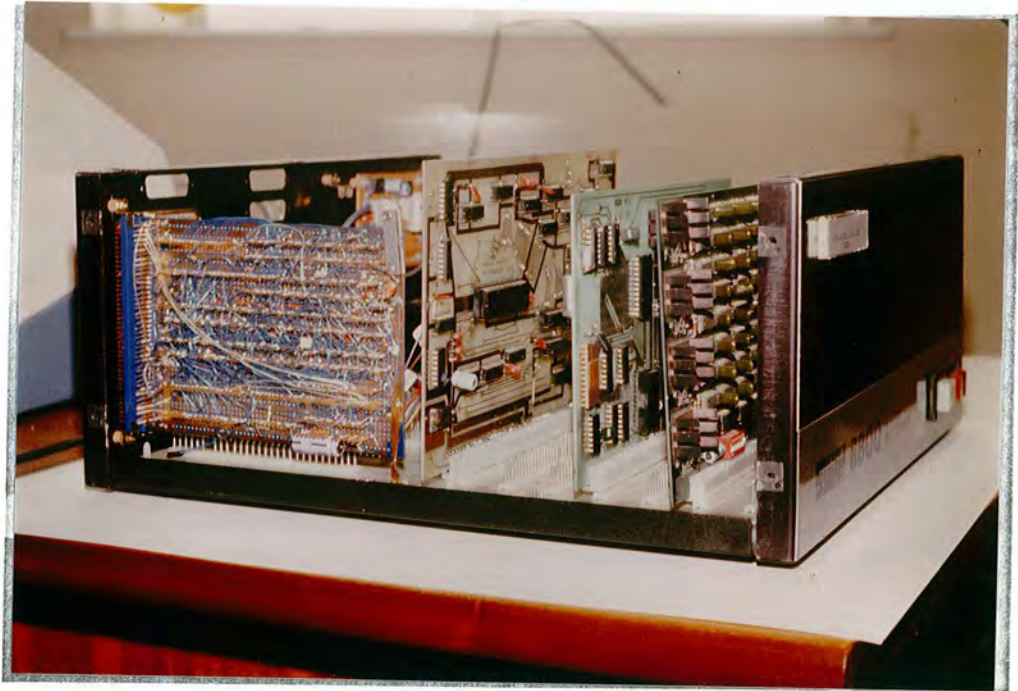


FIGURE 7.7(a). TRANSMITTER HARDWARE (SIDE).

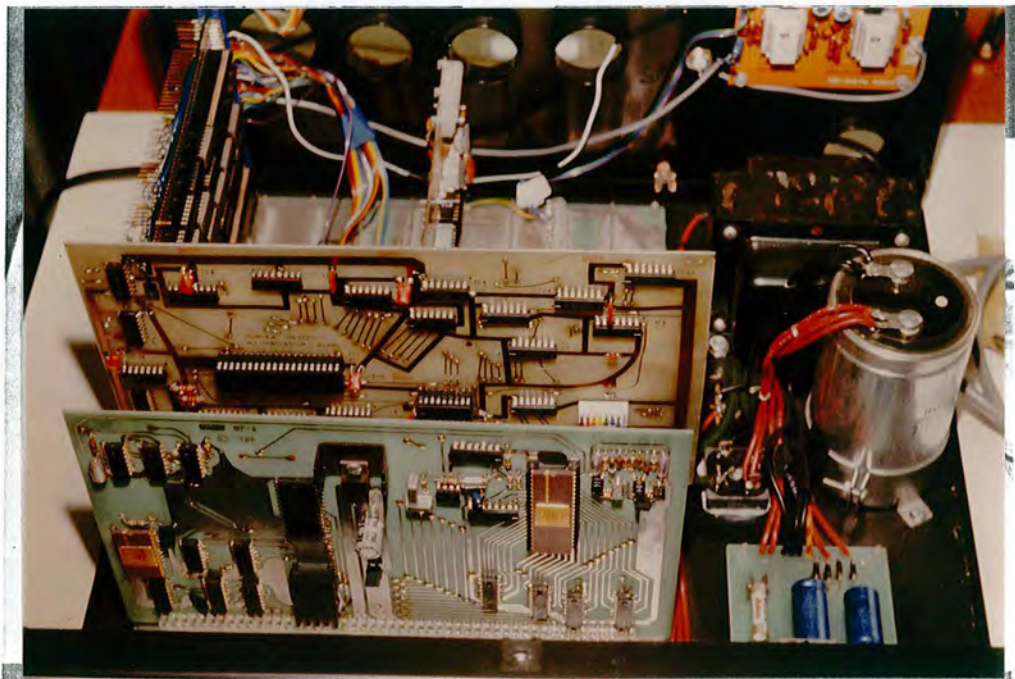


FIGURE 7.7(b). TRANSMITTER HARDWARE (FRONT).

7.3.5 Synchronisation patterns

In order that the receiver may gain element synchronisation, a sequence of phase reversals is transmitted on alternate subchannel slots prior to the frame synchronisation sequence. The phases are equal for each subcarrier during a signal element. The phase reversal sequence is sent over each transmitted subchannel frequency. The element synchronisation pattern is followed by a frame synchronisation pattern, comprising a 31-bit m-sequence pattern. This pattern is transmitted in parallel on all subchannel frequencies. The receiver must attempt to correlate the received synchronisation pattern with the stored sequence.

7.3.6 Construction

The basis for the construction of the transmitter system was the SWTP 6800 development system. The monitor PROM^{label} from the "motherboard" of this system was modified to allow insertion of a 2 kbyte EPROM containing the transmitter software. A slave processor unit (see chapter 4) was also mounted onto the motherboard and was interfaced to the modulator wirewrap circuit board via a length of ribbon cable. The A-D conversion system (used to sample the receiver audio output) was constructed on a wirewrap board having a 30-way edge connector to allow insertion onto the I/O bus on the motherboard. A serial interface board was also mounted on this bus, as was an additional board containing a PIA and two reed relays to control (a) enabling of the D-A output signal and (b) the Tx/Rx relay on the HF transceiver. The analogue filter (see chapter 2) was constructed using copper stripboard and later was mounted on the back panel of the system. Photographs of the transmitter system hardware are shown in

figures 7.7 (a) and (b).

7.3.7 Transmitter testing

Because the transmitter is entirely software controlled, it is possible to implement any of the three digital modulation schemes (ASK, PSK or FSK), using up to 8 subcarrier frequencies, by simple software modifications.

To test the transmitter operation, the software was initially set up to generate a phase reversal sequence on a single frequency of 600 Hz. The photograph of figure 7.8 shows the transmitter output signal on the upper oscilloscope trace and the modulator counter reset signal on the lower trace. It can be seen that one signal element comprises 8 cycles of the subcarrier, as expected, and that the counter reset signal appears at the end of a signal element, indicating to the slave processor (via the interrupt routine) that the element is complete. The CCD spectral evaluation module (see chapter 2) was used to display the power spectrum of the voiceband signal, the result of which is shown in figure 7.9. A single peak can be seen in the signal band output, and also in the CCD Nyquist band region.

The software was then configured to permit transmission of sequences using 2,4, and 8 subcarriers. The power spectra obtained from these signals is shown in figures 7.10 (a), (b) and (c) respectively. For the latter case it can be seen that the envelope of the spectrum is not flat and that the transmitted Nyquist band is evident, in addition to the required signal band. These problems were later overcome by improved filtering of the

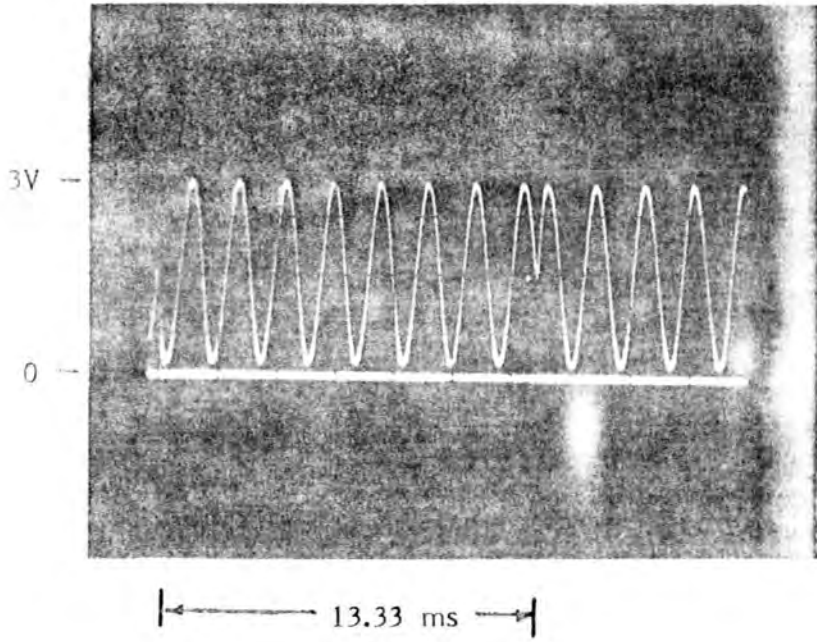


FIGURE 7.8. SINGLE SUBCARRIER PHASE REVERSAL SEQUENCE

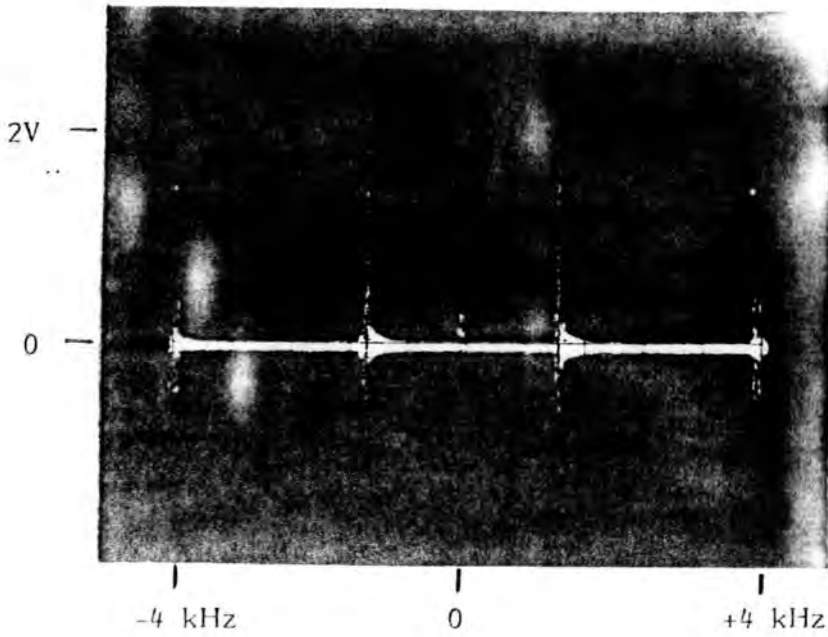


FIGURE 7.9. SINGLE SUBCARRIER POWER SPECTRUM.

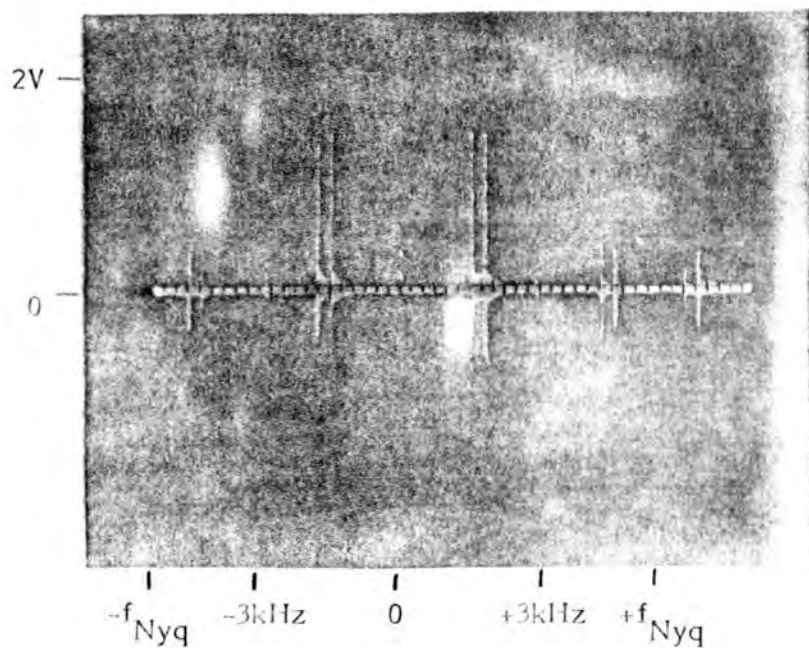


FIGURE 7.10(a). 2 SUBCARRIER POWER SPECTRUM

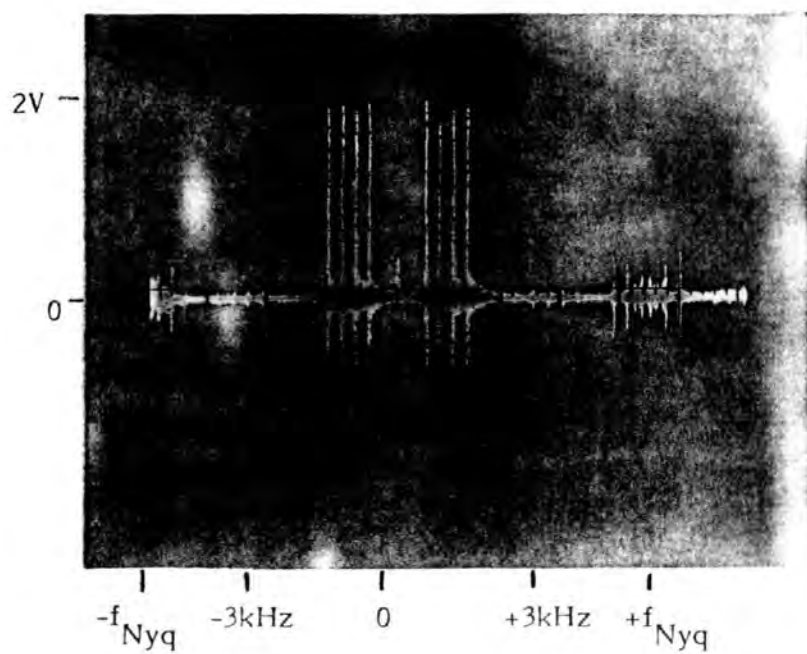


FIGURE 7.10(b). 4 SUBCARRIER POWER SPECTRUM.

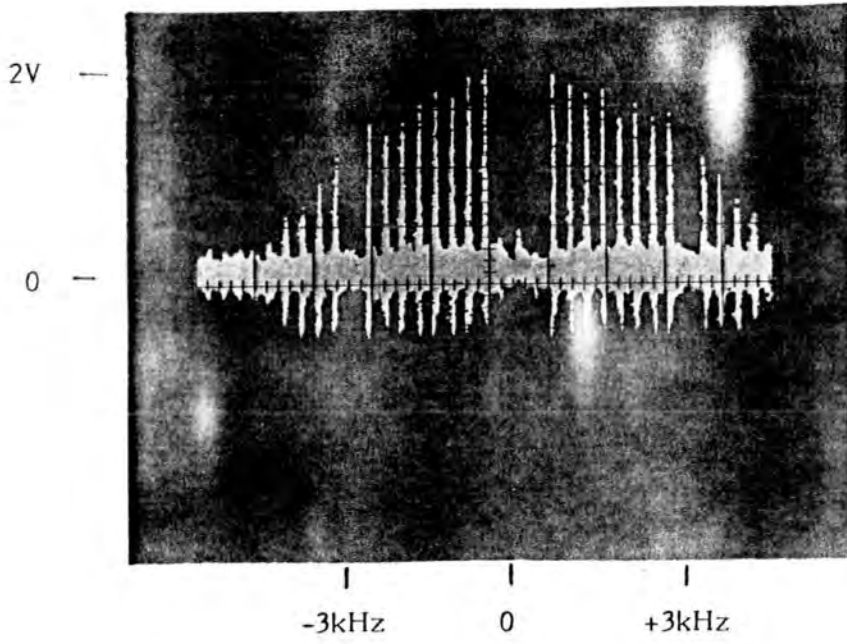


FIGURE 7.10(e). 8 SUBCARRIER POWER SPECTRUM.

D-A converter output signal using the 5-pole filter described in chapter 2, which provided a flat response over the voice channel and a steep rolloff above the 3 kHz band edge.

Although some difficulties were experienced with the receiver demodulation (to be discussed later), it was possible to test the effectiveness of the frequency agility of the transmitter over an HF link. An SSB HF transceiver was available (Collins, type KWT-6), which was installed at the Leicester university field site at Oadby, near Leicester. The antenna socket was connected to an east-west orientated inverted-V half-wave dipole (cut to resonate at 4.7925 MHz) with the apex mounted at a height of approximately 30 ft above ground. An HF receiver (RF Comm. Inc., model RF-505A) was located at Durham; the antenna socket was connected to an east-west orientated half-wave dipole mounted at approximately 100 ft above ground.

The transmitter computing equipment was connected to the HF transceiver as follows. The Tx/Rx reed relay in the microprocessor system was wired to the Tx/Rx switch in the transceiver. The 'transmit' or 'receive' modes could then be controlled automatically. The audio output of the transceiver was connected to the analogue input of the A/D converter. Some difficulty was experienced at first, as the audio gain in the receiver section of the transceiver tended to decrease during the first hour after switching on, after which time the gain remained stable. The gains were therefore set up after the initial "warm-up" period. It was also found necessary to introduce a 0.5s delay after switching from "receive" to "transmit", to allow the

transceiver internal relays to "make" correctly before commencing transmission.

The spectrum of the received signal at Durham was monitored using the HF spectrogram described in chapter 3. Following this, the "quietest" subchannels at the receiving station were noted during intervals of no transmission. During this time, the transmitter itself was estimating the optimum subchannel slots for subsequent transmission. By observing the radiated spectrum from the transmitter during the next message (ie. by determining the subchannel slots occupied by the signal), it was possible to compare the prediction at the receiving station with that at the transmitting station. Qualitative observations indicated that the prediction at the transmitting site was generally in agreement with that at the receiver.

Observations indicated that a much greater level of agility occurred during the evening, when the interference level was higher, than during the early afternoon (when no detectable interference was observed). On no occasion were more than 4 subchannels reallocated after each message block (transmission time of 3.4 mins), the most frequent number being 2. However, during noisy channel conditions, a reallocation was made on approximately 1 out of every 2 occasions. During midday, little interference was observed, and the frequency allocation remained fixed for more than one hour. The subchannel allocation was observed to coincide with the optimum, as noted at the receiver, on approximately 90% of occasions, indicating that interference observations made at the transmitter site are usually coincident

with those made at the receiver.

7.4 Receiver philosophy

Conventional parallel data modems use narrow-bandpass filter banks to separate out the subcarrier frequencies on reception. This technique suffers from two disadvantages. The cost of constructing a bank of such filters is extremely high and, in a frequency-agile environment, the centre frequencies of the filters must be made adaptive or extra filters must be added, both of which still further increase the cost.

It is possible to use digital techniques to perform matched-filter detection of the received multi-subchannel signal. The Discrete Fourier Transform (DFT) of a finite set of samples may be evaluated and, if it is ensured that the samples all pertain to one signal element, the response of the DFT frequency slots may be arranged to match the spectrum of the transmitted signal. If phase modulation is used, then the Fourier coefficients corresponding to a matching frequency slot (or "bin") will allow determination of the phase for the received signal element. That is, if at the end of a received signal element, the DFT of the sampled version of that element is evaluated, the DFT slots corresponding to the transmitted frequencies may be used to extract the phase for each of the subcarriers.

The functions of the demodulator are twofold: (i) to ensure that all the samples for the DFT calculation pertain to a unique signal element and (ii) to determine the phase from the computed Fourier coefficients. The first criterion may be ensured by observing the variation of the phasor magnitude (for a matched

frequency slot) as the DFT is computed for successive sets of samples over a sequence of phase reversals. The magnitude of the phasor as a function of time is a triangular wave as shown in figure 7.11. The peaks of this waveform then indicate the correct synchronising instants. The second function may be obtained by observing the signs of the real and imaginary coefficients, once synchronisation has been achieved. If the phases are permitted to take on 4 possible values, as in figure 7.3, then, for example, a positive real component indicates that the phasor is located in the right-hand half of the complex (z) plane, and a positive imaginary component indicates a position in the upper half of the plane. This determines a differential phase of $\pi/4$, and the data may be obtained by finding the dibit corresponding to the phase difference between this phase and the phase determined for the preceding signal element (because of the differential PSK modulation scheme).

Once element synchronisation has been achieved, it is a fairly simple matter to obtain frame synchronisation by correlating the demodulated data with a start-of-message synchronisation pattern. This pattern must exhibit good correlation properties when preceded by a sequence of keying inversions. In other words, the correlation coefficient should exhibit a large peak at the synchronising instant and a small amplitude at other instants. A 31-bit m-sequence is known to yield good correlation properties and was chosen as the frame synchronising sequence.

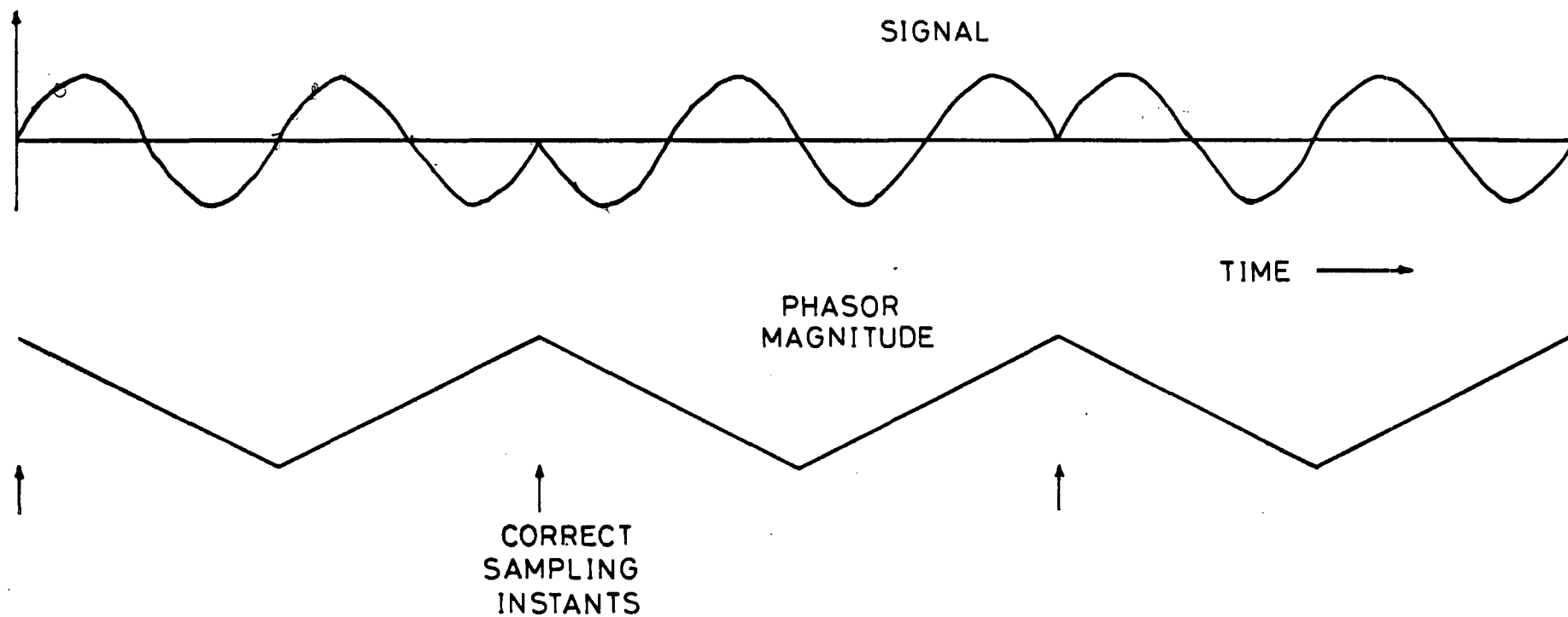


FIGURE 7.11. PHASOR MAGNITUDE VARIATION OVER PHASE REVERSAL SEQUENCE.

In addition to performing the demodulation and synchronisation processes, the receiver must be capable of de-interleaving and decoding the demodulated data. The implementation of these operations has been described in chapter 5.

7.5 Receiver implementation

The computational requirements in the receiver system for the data modem were considerably greater than for the transmitter. The major problem was the demodulation process for which it was necessary to compute the DFT of samples of successive signal elements. A frequency resolution of 150 Hz was required because the available subchannel slots were integer multiples of this frequency. Since the highest permitted subchannel frequency was 2700 Hz it was necessary, at least (by Nyquist's sampling theorem), to compute a transform of length $(2700/150) \times 2 = 36$ points. A decimation in time FFT algorithm is suitable for evaluation of transforms of length 2^m (m integer), therefore m was required to be at least 6. The required sampling rate is therefore 9.6 kHz and the resulting DFT slots are integer multiples of 150 Hz from 0 to 4.8 kHz. (The slots above 2.70 kHz are therefore redundant.) It has been shown in chapter 2 of this thesis that the computation time required for a microprocessor implementation of a transform of this length, even using hardware multiplication, far exceeds the signal element duration of 13.33 ms. Consideration was therefore given to a hardware implementation of the DFT using equipment discussed in chapter 2.

A Charge Coupled Device (CCD) was available which, with additional circuitry, could compute a fixed length (512 point) DFT in 5.12 ms or more, depending on the clock rate. The device contains two 512-stage MOS "bucket-brigade" devices which are used to implement four transversal filters using a split-electrode weighting technique. The filters are used in a "chirp-z" implementation of the DFT algorithm and the device is supplied with circuitry which allows the power spectrum of an analog input waveform to be evaluated. It is necessary to include additional circuitry if, as in this case, the Fourier coefficients are required. The device, its operation, and the design and construction of a module suitable for extracting the complex coefficients from the resulting transform, have been discussed in chapter 2. It has also been shown that the phase of an input signal may subsequently be determined. Reference should be made to this chapter in the subsequent discussion. A receiver design was attempted which utilised the device and its associated circuitry to perform the demodulation process. However, in the course of experimentation, several difficulties became apparent.

At a clock rate of 38.4 kHz, it is possible to acquire 512 samples in precisely the baud time, ie. 13.33 ms. At this sampling rate the frequency domain resolution is 75 Hz. Adjacent subchannel frequency slots are therefore separated by 1 frequency bin and this scheme appears to present a possible solution to the demodulation problem. However, because of the weighting applied to the split electrodes in the CCD filters, a time window is applied to the incoming signal which effectively spreads the power in each frequency bin over several adjacent bins. There is

therefore considerable overlapping between adjacent frequency slots, rendering it impossible to determine the Fourier coefficients using the chosen sampling frequency. It is possible to increase the resolution by decreasing the sampling frequency. However, to ensure that all samples pertain to a single element, it is then necessary to reduce the transform length, which in this case is impossible as the length is fixed by hardware.

To overcome the problem of overlap of nearby frequency bins, the possibility of spreading the input signal spectrum over the frequency range of the CCD was considered. A greater separation between subchannel frequency bins could then be achieved. The CCD operates in a serial fashion; one sample in the time domain is clocked into the device as one sample in the frequency domain is clocked out. To evaluate the DFT of a set of 512 samples of an input signal, it is necessary to enter a replica of the input signal into the device as the frequency domain samples are clocked out, if the true spectrum is to be obtained. To spread the spectrum of the input signal from the HF receiver over the frequency range of the CCD it was necessary to (a) sample the input signal and store the resulting samples, (b) enter the samples into the CCD, then replicate the samples until the CCD was full, (c) obtain the spectrum from the CCD output while entering further replicas of the input signal. It was calculated that, even if the samples were entered into the device at the specified maximum rate of 100 kHz (using a direct memory access (DMA) arrangement), the total time required for all operations was in the order of 20 ms, which was greater than the signal element

duration. These problems therefore precluded the use of the device for the demodulation process. It is envisaged, however, that the advent of new bit-slice microprocessors will permit the computation of the DFT in the required time.

7.6 Phase detection

Some experimentation was carried out into phase demodulation of a single tone carrier using the technique of maximising the phasor magnitude, mentioned in section 7.4. Details of this work are now described.

An experiment was set up to demodulate a PSK modulated carrier using software. The transmitter routines were modified to provide, at the transmitter output, a sequence of phase reversals on a 1200 Hz carrier. The signal was sampled by the receiver at a rate of 4800 Hz, resulting in 64 samples for each signal element. The 64 samples were reduced to 32 samples by averaging samples $x(n)$, $x(n+32)$, and a 32-point DFT was computed for each element, as described below. The bandwidth of each of the DFT slots in this case is 150 Hz, which matches the spectrum of the transmitted signal. If a 32-point DFT is computed on 32 successive samples, the received signal will be contained within the DFT slot at $k=8$.

For the 32 point DFT:

$$F(k) = \sum_{n=0}^{31} f(n) e^{-j2\pi kn/32} \quad k= 0,1, \dots 31.$$

and

$$\begin{aligned} F(8) &= \sum_{n=0}^{31} f(n) e^{-j\pi n/2} \\ &= \sum_{r=0}^7 f(4r) - \sum_{r=0}^7 f(4r+2) + j \left[\sum_{r=0}^7 f(4r+3) - \sum_{r=0}^7 f(4r+1) \right] \end{aligned}$$

The computation of the DFT slot at $k=8$ therefore requires 16 additions and 16 subtractions. No multiplications are needed.

The magnitude of the function is:

$$|F| = \sqrt{\left[\sum_{r=0}^7 f(4r) - \sum_{r=0}^7 f(4r+2) \right]^2 + \left[\sum_{r=0}^7 f(4r+3) - \sum_{r=0}^7 f(4r+1) \right]^2}$$

The receiver system for the tests employed 2 slave processors. The first was used simply to interrupt the second at intervals of 13.33 ms, ie. at the signal element duration. The received analog waveform was sampled by the second slave processor using an 8-bit A-D converter and PIA. Computation of the Fourier coefficients was implemented using the master processor. To locate the correct sampling instants, the magnitude of the current phasor (at $k=8$) was compared with the magnitude of the previous phasor. If the current phasor magnitude was greater than the previous, the timing was advanced or retarded by appropriately adjusting the counter in the first slave. If the previous adjustment was a retardation, then a further retardation was made,

the aim being to maximise the phasor magnitude. Similarly, if the previous adjustment was an advancement, a further advancement was made. If the current phasor magnitude was less than the previous, the timing was made opposite to the previous adjustment.

Once the correct synchronising instants were located, the phase was determined by observing the signs of the Fourier coefficients. It was necessary to make a trade-off between the maximum time required for synchronisation and the amount of tolerable phase jitter. If the timing step adjustment (for a retardation or an advancement) is large, then the time taken to synchronise will be small (because it will take less time to reach the point where the phasor magnitude is maximised), but the jitter will be large because a continual adjustment is being made around the point of maximum magnitude. For a 4-phase system, the amount of tolerable jitter must be less than one-eighth of the period of the highest frequency subcarrier. In the case of the multi-subchannel modem, the highest frequency subcarrier is 2700 Hz, so the maximum tolerable jitter is $46 \mu\text{s}$ either side of the correct sampling instants. A more realistic figure in a noisy environment might be $20 \mu\text{s}$. If the step length is set to this figure, the worst case synchronisation time must then be $(13.33\text{ms}/20\mu\text{s}) \times 13.33\text{ms} = 8.88\text{s}$. This is an excessively long time and will result in a considerable reduction in data throughput owing to the long phase reversal sequence required. It is preferable to begin with a large step length and then to reduce the step length as an improvement is observed. A method that was found to be successful was to double the step length if a

deterioration in sync was observed (up to a maximum of 2.5 ms) and to halve the length if an improvement was noticed (down to a minimum of 20 μ s). The jitter is then minimised, and phase lock is achieved in a much shorter time (worst case was observed to be 0.19s).

7.7 Conclusion

This chapter has described the design of an adaptive modem for use over HF radio channels which are subject to multipath distortion and noise effects. It has been shown that several modem techniques (TDFM, FEC and frequency agility) may be combined in one system which may be implemented at very low cost by employing nearly-all digital techniques. The use of microprocessors in such designs allows major system changes (such as a change in the modulation or the coding scheme) to be effected by simple software modifications. The use of VLSI technology also allows the physical size of the system to be kept to a minimum.

The transmitter has been described in detail and has been shown to operate successfully over a real HF link. An novel aspect of the transmitter system is the slave processor/ shift register hardware used to generate the modulated signal. However, some difficulties were encountered with the receiver implementation which have been discussed in section 7.5.

Many designs for HF modems have been described in the literature, some of which have been mentioned at the beginning of this chapter and in other areas of this thesis. However, most are extremely costly and non-adaptive, and for these reasons have not presented an economically viable alternative to satellites for

long-distance communications. This chapter has discussed the design of an economical adaptive modem based on discrete signal processing techniques which attempts to overcome many of the problems encountered with previous systems.

CHAPTER 8 Error patterns & coding performance

8.1 Introduction

Radio signals propagated via single or multiple reflections from the ionosphere are frequently subjected to severe levels of amplitude and phase disturbances. The effects on a serial data stream are to cause large numbers of errors which will considerably degrade the fidelity of the received data. The signal distortion arises from (i) intersymbol interference caused by multipath propagation, and (ii) additive noise from natural and man-made sources. The errors which result from these effects are, to a large extent, unpredictable, and will result in a wide range of error rates for a given channel. The error rates may vary by several orders of magnitude even over a relatively short time span. The error distribution is often distinctly non-random in nature, and clusters of errors may occur as a result of noise or fading.

This chapter describes an experiment undertaken to investigate the statistical properties of the error patterns observed over a medium-haul HF radio data link and to assess the performance of the real-time error correction scheme described in chapter 5, with and without interleaving of the codewords. It will be shown that errors occurring over the link are significantly non-random in nature and that bit-interleaved short-length random-error-correcting codes can be effectively used to combat such errors. Results are presented showing the deviations from the theoretical random distributions and the apparent time-varying nature of the error statistics.

Some work on the applications of coding in HF communications systems has been described in the literature (72-75). Much of this, however, has involved recording of the received data; the analysis being performed later using a mainframe computer. The work in this chapter uses microprocessor techniques to perform decoding and error-pattern recording in real time.

8.2 HF Equipment

Experimental tests were carried out over a 250 km south-north path between Leicester and Durham using an HF data link centred on 4.7925 MHz. Permission to use this frequency for data transmission was granted by the Home Office subject to the station callsign (G9BLD) being transmitted in morse code at regular intervals. The HF transmitting equipment comprised a Collins KWT-6 single sideband suppressed carrier transceiver tuned to output 40W PEP into an east-west orientated half-wave dipole antenna situated at a height of approximately 10m above ground level. The transceiver uses valve technology and is constructed in modular form; the synthesised VFO, sideband generator, power amplifier, receiver, and tuning unit are located in separate sections. The audio input frequency response of the transmitter was plotted in the laboratory. The results are shown in figure 8.1 and indicate that the response is reasonably flat within the voice channel spectrum.

The receiving equipment at Durham comprised an inverted-"V" half-wave dipole antenna with the apex situated 25m above ground level (also orientated east-west) feeding into an RF Communications Inc. RF-505A synthesised HF communications

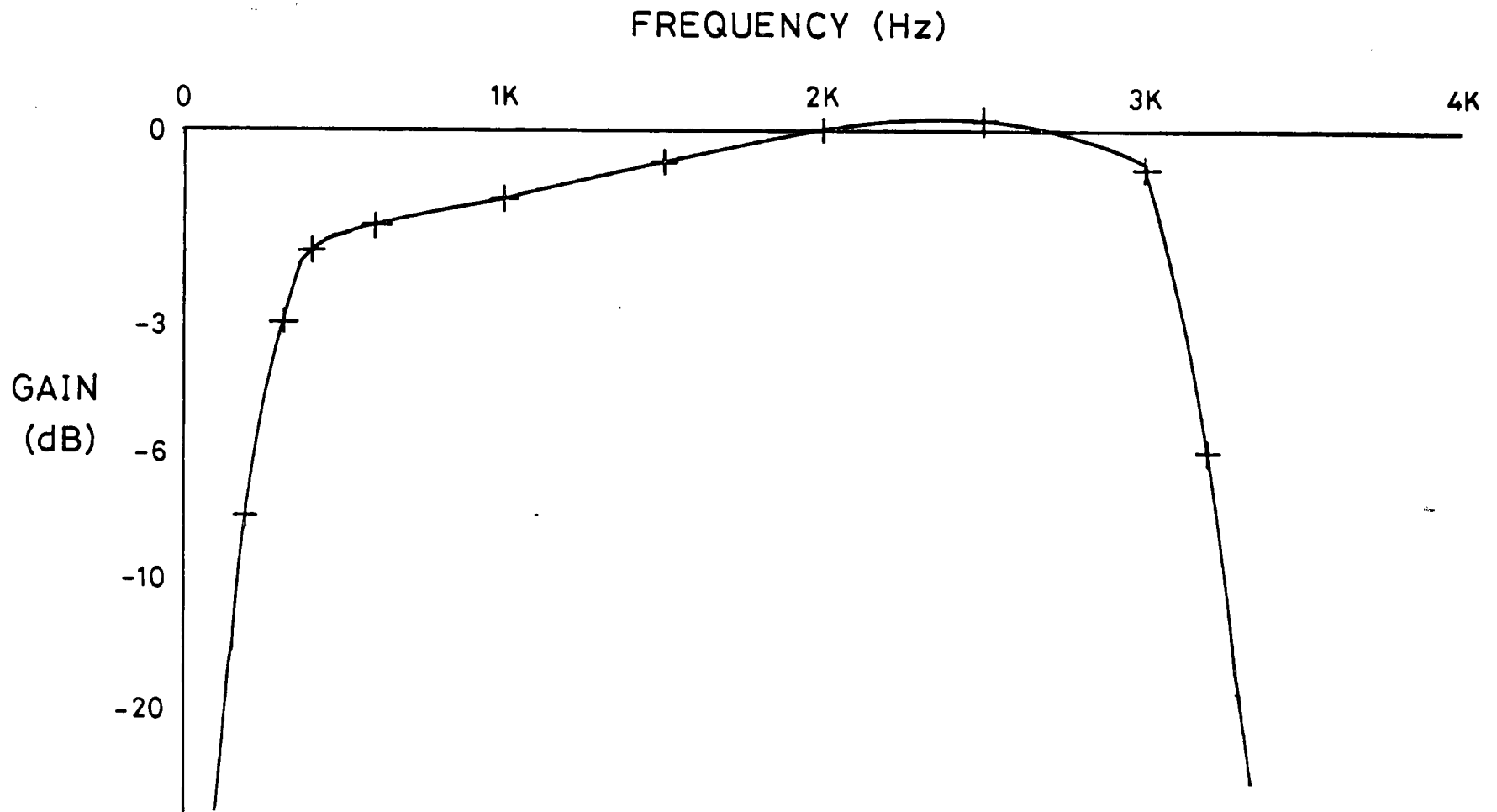


FIGURE 8.1. KWT-6 TX AUDIO INPUT FREQUENCY RESPONSE

receiver. This receiver uses transistor and IC technology and has 3kHz wide independent sideband outputs, as well as CW and AM reception facilities.

8.3 Computing Equipment

All of the computing equipment used for the experiment was based around the Motorola 6800 microprocessor. The transmitting equipment hardware was that described in the previous chapter. It comprised a master CPU board with an EPROM containing the transmitter software, a slave processor unit (described in chapter 4), modulator board and filter unit. The software was configured to produce a binary amplitude modulated audio-frequency carrier at 75 bps at the output of the filter which was then used as input to the HF transmitter. The ASK modulation scheme was chosen to ensure a statistical independence of the errors at the detector. If a differential PSK system was used, the errors would tend to occur in pairs, as the differential detector requires a recovery time of one signal element following the detection of an erroneous bit. The basis of the receiving system was an MSI 6800 microprocessor development system containing two slave processor units, and interfaced to a triple minifloppy disc drive. A real-time clock facility was incorporated within the receiving system which is fully described in section 8.6.7.

The test data format is now described, followed by a more detailed discussion of the transmitter and receiver configurations. Test results are then presented and discussed.

8.4 Data format

Message sequences of 896 bits were used as data for the tests, comprising four consecutive sequences of 32 random 7-bit characters. Two reasons for choosing the 7-bit character format were (a) that each character could conveniently be encoded into a (15,7) codeword and (b) the increasing popularity of the 7-bit ASCII character set as a replacement for the 5-bit BAUDOT code.

The transmission format was as follows:

- (1) A sequence of 128 characters, each preceded by a "1" start bit and terminated by a "0" stop bit.
- (2) The same sequence, with each character forming the information section of a 15-bit error correcting codeword
- (3) The sequence of codewords transmitted in (2), but interleaved to a depth of 16.

Each of the above sequences was preceded by a synchronisation preamble consisting of a series of amplitude inversions, a 15-bit m-sequence and a 7-bit message identification pattern. The m-sequence allowed 3 random errors to occur before synchronisation failure and the identification pattern permitted 1 error before recognition failure.

As mentioned previously, it was necessary to transmit the station callsign in morse code at frequent intervals. The transmitter software included an automatic morse code transmission routine used to transmit the message " DE G9BLD" (the allocated call-sign) at a speed of 12 wpm, approximately every 20 minutes.

8.5 Transmitter

The microprocessor controlled transmitter system hardware has been described in chapter 7. The software may be categorised as follows:

- (1) System initialisation & main procedure
- (2) Slave processor bootstrapping
- (3) Carrier frequency synthesis
- (4) Modulation
- (5) Data encoding and bit interleaving
- (6) Morse code transmission

All software for implementing the above processes was written in M6800 assembly language and assembled into object code format using the co-resident mnemonic assembler. An EPROM programmer, connected to the SS-30 bus of the microprocessor development system, was used to transfer the contents of the object code file onto a single 2 kbyte EPROM. A listing of the transmitter software is shown in the listing in Appendix 2, and a system memory map is shown in figure 8.2. The transmitter routines reside in the EPROM which occupies the top 2 kbytes of the memory address space. The highest two locations contain the program start address (or "reset vector"). The slave processor memory has the (master) address space \$C000-\$C3FF; the slave control latch is therefore at address \$C000, ie. at the bottom of the slave address space. The master system RAM occupies the 4 kbyte space from \$0000-\$0FFF and a single PIA is located at addresses \$8010-\$8013. The two least significant bits of the "A" side of this PIA are used to control a pair of reed relays, one of which determines the transceiver operating mode (Tx or Rx); the other

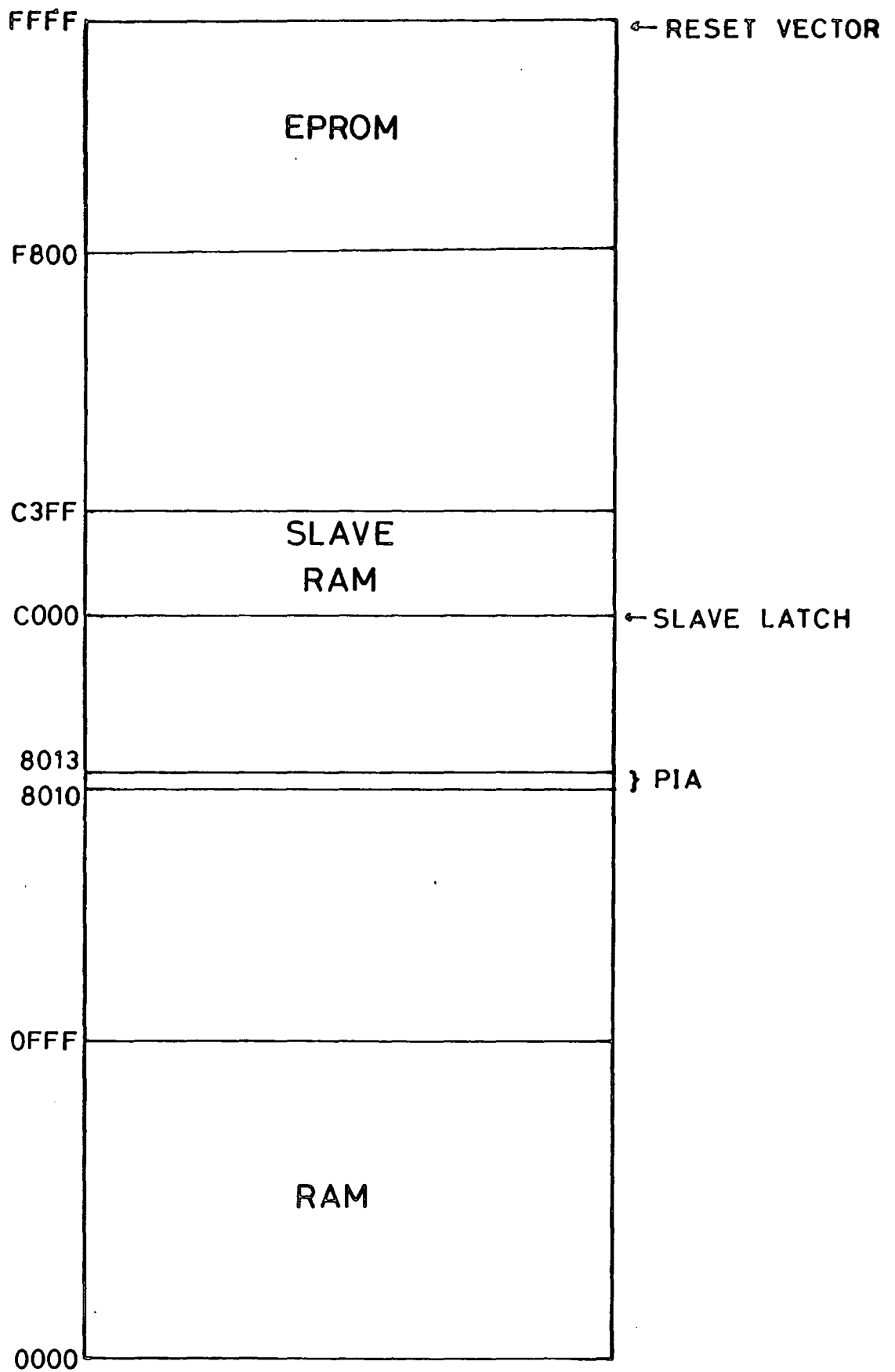


FIGURE 8.2 TX SYSTEM MEMORY MAP

is used to gate the filtered synthesised audio output signal to the transmitter audio input circuitry.

Reference should be made to the transmitter software, listed in Appendix 2. The main procedure begins at the label "INITLSE". The transceiver is initially set to "receive" mode and the filter output is not gated to the transmitter. The slave processor "reset" line is brought low and the slave processor routines are loaded into the slave RAM using the bootstrap loader subroutine. The phase and frequency parameters are then set up in the appropriate tables in slave memory (see chapter 4, section 4.3.6, for details) to initialise all subchannels to a phase of π radians relative to the start of a signal element, and having a frequency of 1500 Hz (centre of the voice channel). The slave processor is then brought into reset by writing a "1" to the slave reset line, the transceiver is switched to "transmit" and the station callsign is transmitted in morse code at a speed of 12 wpm by the morse code transmission subroutine. The first sequence consisting of the uncoded message, followed by the coded message, and finally the coded message with bit interleaving are all transmitted twice. Transmission of the three sequences is repeated in this manner 8 times before the station callsign is transmitted again. 75 bps.

8.5.1. Bootstrap loader

The bootstrap loader routine simply maps the slave processor control program from the EPROM into the slave RAM area. Slave processor execution is suspended during this procedure by holding the slave reset line at logic '0'. The beginning and end

addresses of the area to be mapped are contained in Y,Y+1 and Z,Z+1. The index register points to the start address of the area into which the data is to be mapped.

8.5.2 Carrier frequency synthesis

Synthesis of the AF carrier is performed by the slave processor and the modulation board containing the shift registers and D/A converter. The synthesis of a multitone carrier using this system has been described in chapter 4. For this experiment a single tone was required, and all eight subchannels were set to the same frequency and phase. The first 40 bytes of the slave memory contained (after bootstrapping) the lookup table required for carrier generation, consisting of equally spaced samples of a sinusoid. The slave reset vector is located in the top two bytes of the slave address space, ie. at local (slave) addresses \$03FE and \$03FF, and the local IRQ vector is located at \$03F8 and \$03F9. An IRQ is initiated by the modulator circuit immediately following the transmission of samples for a complete signal element.

8.5.3 Modulation

An ASK modulation scheme at 75 bps was used for the experiment. The transmitter software was configured to generate an on-off keyed audio tone at 1500 Hz. Because of the "lookup table" method of carrier synthesis, the phase at the start of a transmitted signal element is always constant. The modulation process is carried out by the subroutine "SNBITS", which transmits N left-justified bits contained in the A accumulator, where the bit count, N, is contained in the B accumulator. If a

bit is "0", the step lengths in the slave processor subchannel table are all set to zero, thereby inhibiting carrier transmission during that element. If the bit is "1" the step lengths are all set to 20 and the element comprises 20 cycles of a 1500 Hz sinusoid.

8.5.4 Morse code transmission routine

This routine is used to transmit the message "DE G9BLD" when called. The output gating relay is keyed according to the bits in a stored sequence which represents the message. The relative timing of the elements comprising the morse characters was arranged to conform to the following internationally accepted schedule (9):

dash	=	3 dots
element space	=	1 dot
letter space	=	3 dots
word space	=	6 dots

The key is initially set to "open" - ie. the carrier is not gated to the transmitter. Bits are read sequentially from the sequence "MMES" and if the bit that is read is a "1" the state of the key is switched; if it is "0" nothing is done. After each bit has been read, and the key switched if necessary, a fixed delay is introduced which governs the overall transmission rate of the morse message. The sequence required to transmit the characters "DE G9BLD" is shown in figure 8.3.

8.6 Receiver

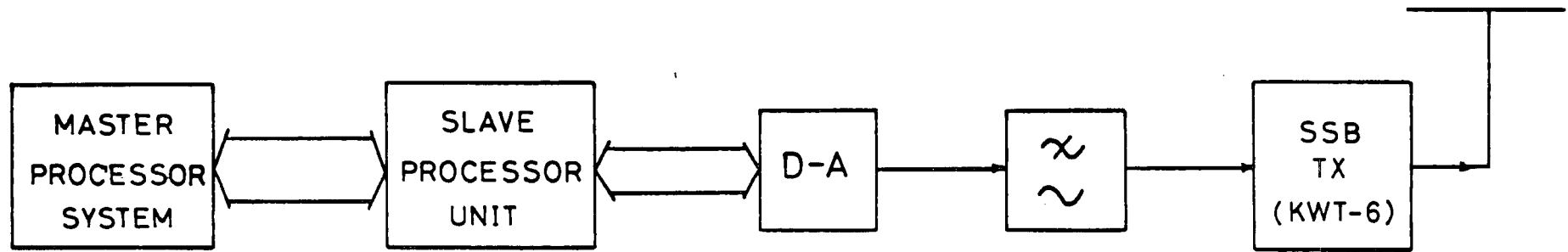
The receiving equipment consisted of the HF radio receiver with dipole antenna, ASK demodulator unit, MSI 6800 development system with 2 slave processor units (see chapter 4), triple floppy disc drives, and printer. The hardware configuration of the system is shown in figure 8.4. One slave processor was required for real-time decoding and de-interleaving of the received data, the other was used as a timer to locate the correct sampling instants on the demodulated signal. The floppy disc drives were used to store statistical information about the received data for later analysis. The receiver functions may be categorised as follows:

- (1) demodulation
- (2) synchronisation
- (3) de-interleaving & decoding
- (4) error counting
- (5) error pattern recording
- (6) disc management
- (7) real-time clock

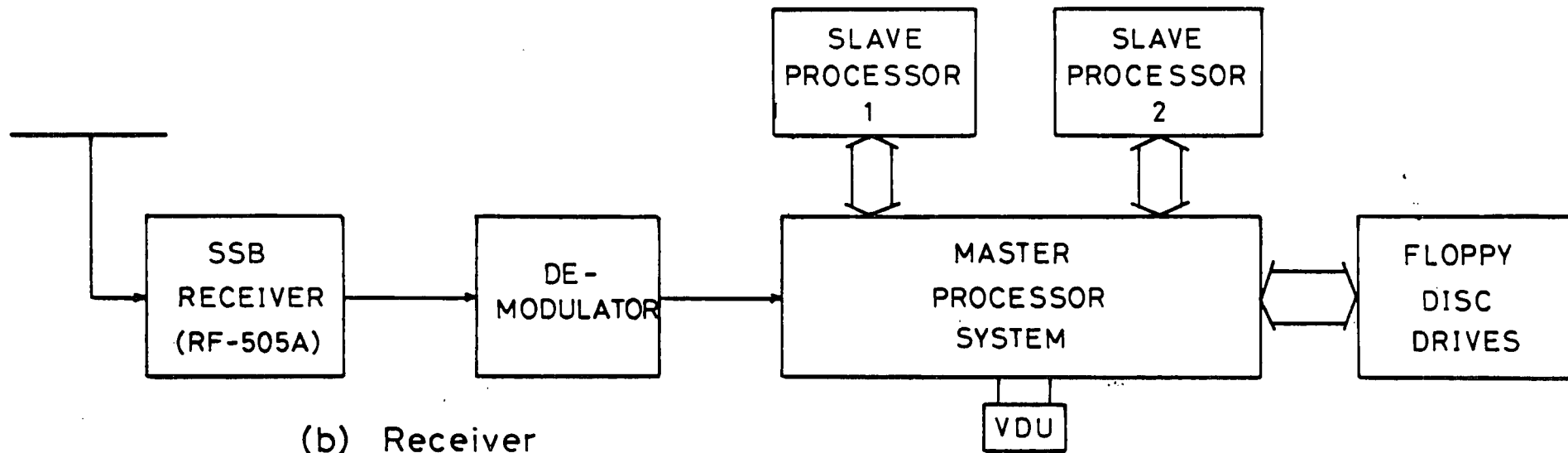
Each function is now described in turn.

8.6.1 Demodulator

The circuit of figure 8.5 was devised to permit incoherent envelope detection of the received signal. The USB audio output from the HF receiver was buffered and half-wave rectified by the diode. The envelope was extracted by low-pass filtering using C and R_1 , then buffered and finally converted to a rectangular baseband signal using the Schmitt trigger. The latter was preferred to a straightforward comparator in order to minimise



(a) Transmitter (Leicester)



(b) Receiver

FIGURE 8.3. HARDWARE CONFIGURATION

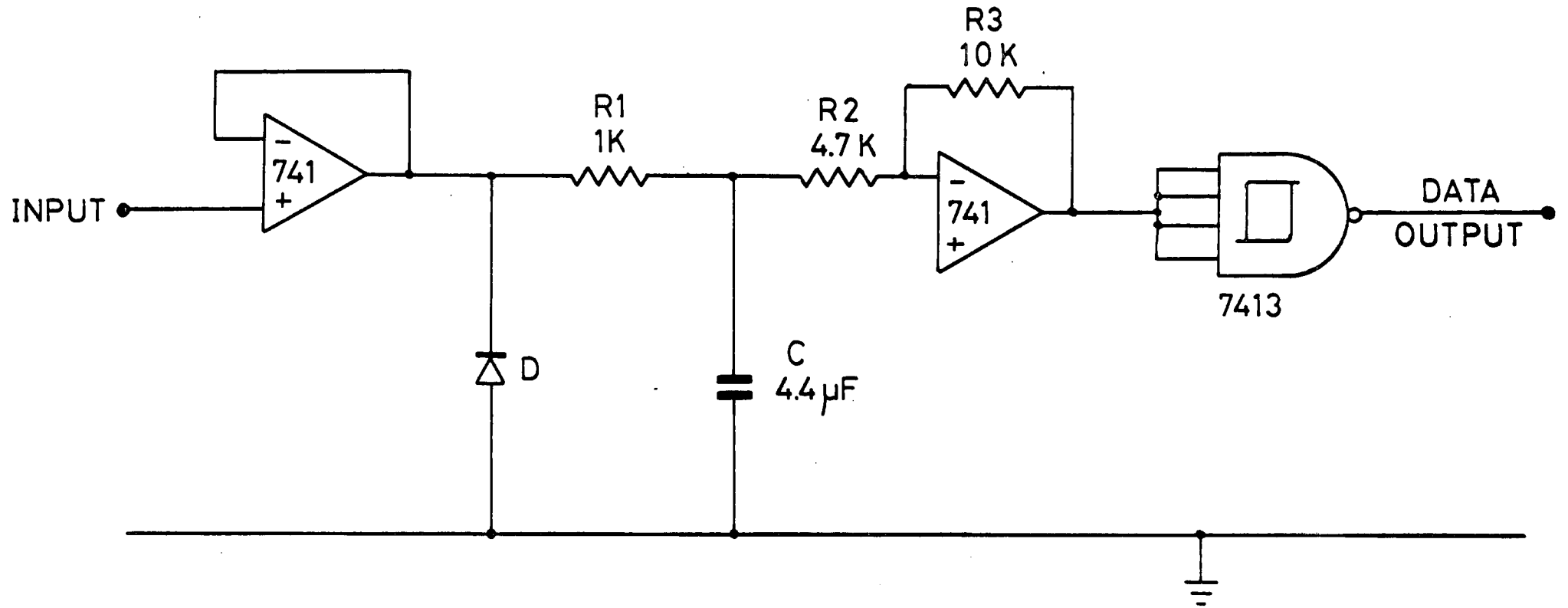


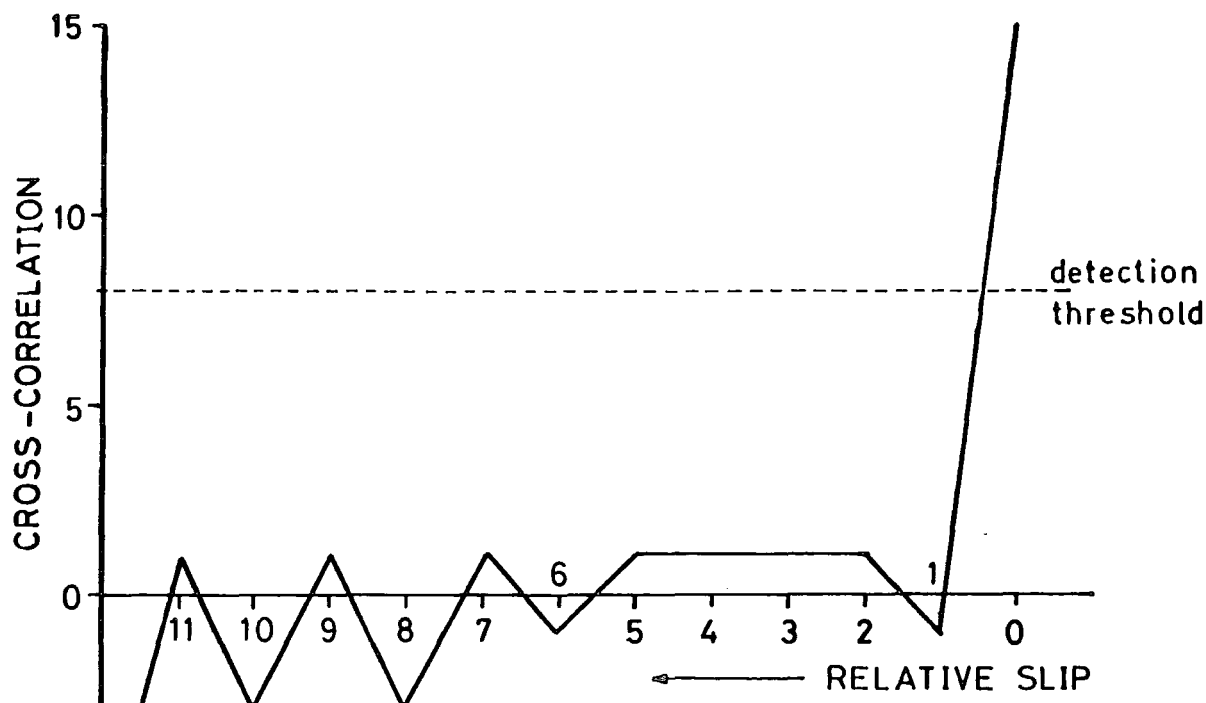
FIGURE 8.5. DEMODULATOR CIRCUIT

noise effects. The output data signal was then fed to the most significant bit of one port of a PIA interfaced to the main processor system.

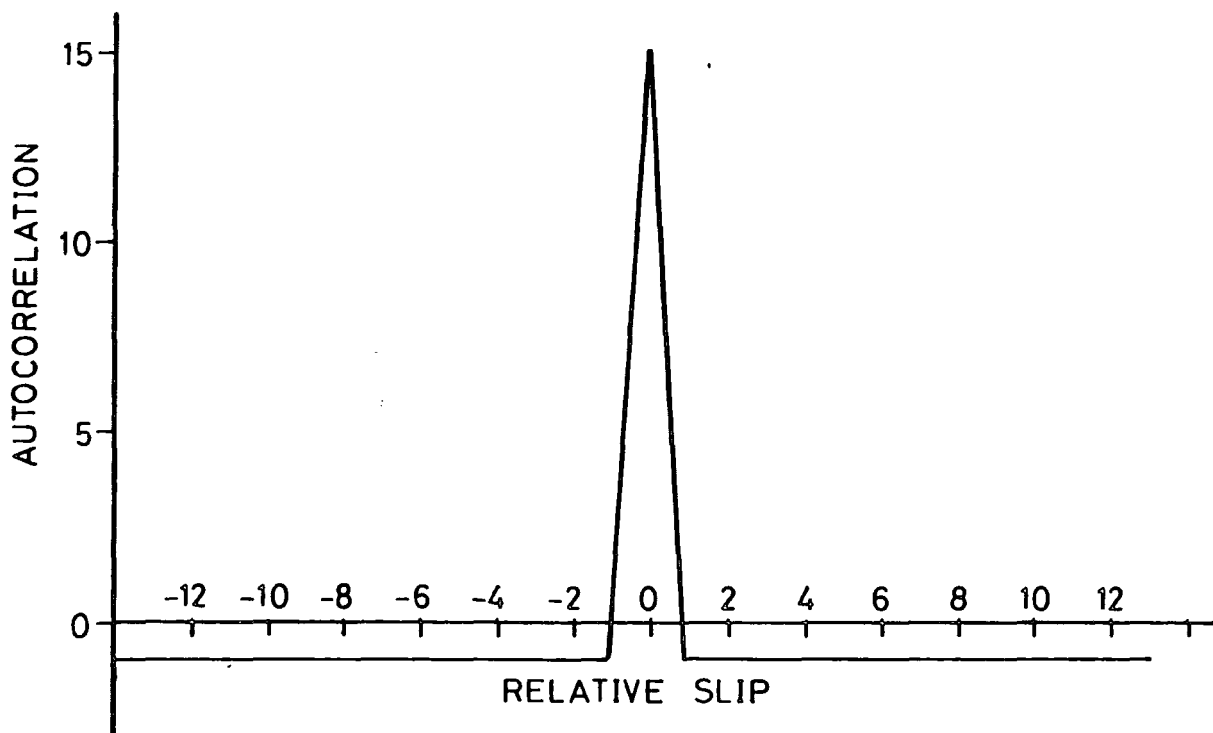
8.6.2 Synchronisation

The preamble sequence transmitted immediately before each message was used to obtain element and frame synchronisation. The purpose of element synchronisation was to ensure that each signal element was sampled as near as possible to its mid point. The first part of the preamble consisted of a sequence of amplitude inversions which was used by the receiver to locate the correct sampling instant to within 2% accuracy. Once synchronised, a software timing loop in the second slave processor unit was used to interrupt the master processor at the centre of each signal element.

Frame synchronisation was achieved using a 15-bit m-sequence which was known to yield good correlation properties when preceded by the amplitude inversion sequence. The cross-correlation between the stored m-sequence and the synchronisation preamble is shown in figure 8.7(a). The autocorrelation property of the same sequence is illustrated in figure 8.7(b). During synchronisation, each received bit is shifted into a 2-byte buffer (SNPAT) and a correlation is performed with the stored sequence (MSEQ) by finding the Hamming weight of the result of an exclusive-OR operation between the low-order 15 bits of SNPAT and the sequence MSEQ. A low weight indicates a good correlation. A correlation factor > 8 indicates detection of the synchronisation pattern. The distance from the detection threshold to the correlation peak is 7



(a) CROSS CORRELATION WITH SYNCHRONISATION PREAMBLE.



(b) AUTOCORRELATION FUNCTION

FIGURE 8.7. m-SEQUENCE PROPERTIES

which allows up to 3 random errors in the sync pattern before the receiver fails to detect it. (This is because a distance of at least $2t+1$ is required if t errors are to be corrected.)

The message identification pattern was transmitted immediately following the m-sequence, and consists of one of three codewords from an m-sequence of length 7, the codeword indicating whether the message is uncoded, coded, or coded and interleaved. The distance between different words in this code is 4, which allows correction of a single error.

8.6.3 De-interleaving & decoding

Slave processor 2 was used for real-time de-interleaving and decoding of the received data. Operations by the slave were carried out on one of two pairs of tables containing the data while operations on the other pair were performed by the master. One table in each pair contained data for de-interleaving (if required); the other contained received words for decoding. Two control parameters were passed to the slave from the master indicating (a) which pair of tables were to be operated on (TBFLAG), and (b) whether or not de-interleaving was required. A pictorial representation of the operations is shown in figure 8.8. The de-interleaving process was executed by the subroutine DINTLV and the resulting words were entered into the appropriate decoding table as they were extracted. On completion of this operation, the 16 received words in the decoding table were decoded using the algorithm described in chapter 5. Each decoded word was returned to its corresponding position in the table. A count of the total number of errors corrected during decoding was kept in the

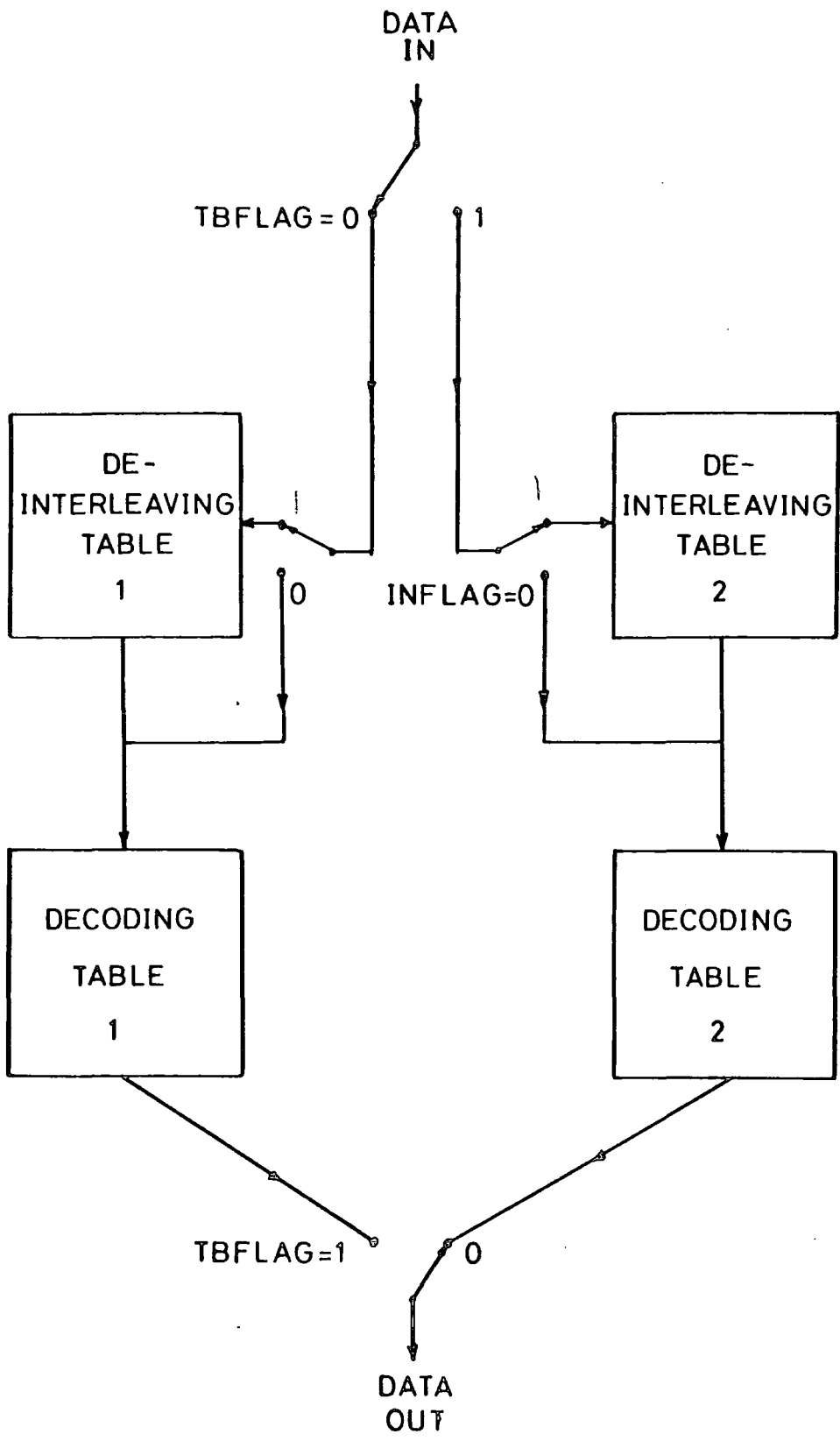


FIGURE 8.8 DIAGRAMMATIC REPRESENTATION OF DE-INTERLEAVING & DECODING PROCEDURES.

two-byte counter "ERRCOR".

While decoding and de-interleaving operations were performed by the slave, the newly received data was entered into the inactive pair of tables in slave RAM. Coded but non-interleaved data was entered directly into the decoding table. When the slave operations were complete, the decoded data was offloaded by the master and the tables were switched to allow the receiving process to continue.

8.6.4 Error counting

The number of errors in a decoded message was determined by cross-correlating the decoded words with a stored table of correct characters. This cross-correlation was performed by implementing an exclusive-OR operation between the decoded word and the corresponding stored character. The least significant 7 bits of the result were then shifted out into the carry flag, in turn, and if the flag was set, the bit was found to be erroneous. The total number of corrected errors was obtained from the slave processor on completion of the decoding procedure.

The total number of errors in the received bit stream (before decoding) was determined by a similar correlation procedure and the statistical properties of the error patterns were recorded as described in the next section. Results of the error counts and the number of errors corrected were stored on disc.

8.6.5 Error pattern recording

As each bit was received a test was made to determine whether or not the bit was erroneous. This was done by correlating the received data with the stored (correct) bit pattern. Three tables of stored patterns were required; one for each of the uncoded, coded and coded and interleaved sequences. A flag (PREV) was used to indicate if the bit immediately preceding the current bit had been received correctly, thereby allowing the number of consecutive erroneous bits to be recorded. A count of the number of consecutive correctly received bits was recorded as a positive number (1 or 2 bytes) on the disc and a count of the number of consecutive errors was recorded as a 2's complement negative number. A complete record of the error pattern information for one received sequence therefore consisted of alternate positive and negative values which could be later retrieved for analysis. A 2-byte "start-of-message" marker (\$FFFF) was recorded at the beginning of each data set to distinguish the records. Because it was often necessary to record on disc a large number of bytes for each received sequence, 2 disc drives were used to store the information. When the disc in drive 1 was full, the file (ERRPAT.000) was closed and a new file (with the same name) was opened on drive 2. The file capacity was therefore increased from 80 kbytes to 160 kbytes and allowed a useful quantity of data to be collected (> 24 hours) without exceeding the disc storage capacity.

8.6.6 Disc management

The disc file management software allowed sequential access disc files to be created, written, read and destroyed by the system user and is described in reference (76). Two files were used to store information on the received data; therefore two file control blocks (FCBs) were required. A file on disc drive 0 (called 'ERRDAT.000') was used to record the following parameters in sequential order:

- (a) time of day (three 2-digit BCD numbers)
- (b) message type (1-uncoded, 2-coded, 3-interleaved)
- (c) no. of errors in decoded message (2 bytes)
- (d) total no. of errors in received bit stream (2 bytes)
- (e) total no. of corrected errors (2 bytes)

A second file (called 'ERRPAT.000') was created, initially on drive 1, to store error pattern data on the incoming bit stream.

8.6.7 Real-time clock

A real time clock was kept within the processor system memory by using the M6800 interrupt capability. A 1Hz timing reference was derived from a digital clock unit installed in the receiver rack to display the current time of day. The 1Hz TTL output from this unit was connected to the CA1 control line of a PIA which was configured to cause an interrupt to be generated on detection of a negative edge on this line. If bit 0 of the control register is set to a '1' and bit 1 is set to a '0', a negative edge on CA1 will cause an interrupt to be generated by setting the IRQA1 flag (bit 7), which causes the IRQ line on the processor to be brought low. The system monitor interrupt routine then causes a jump to the address stored at the lowest two locations in the monitor RAM.

This address is the start of the clock routine. The clock routine updates the time of day which is stored in BCD format in four bytes located on page 0, one each to hold seconds, minutes, hours and days. The penultimate instruction in the clock routine clears the IRQA1 flag by reading peripheral data register A. A 'return from interrupt' instruction then restores the stack and execution of the interrupted program is resumed. The fastest execution time for the clock interrupt routine is 31 machine cycles; the slowest is 84 cycles.

8.7 Data analysis programs

The experimental results were based on data collected during the week of 10/7/80 to 16/7/80. Propagation conditions were found to be similar from day to day. Two programs were written in BASIC to analyse the data. The first was used to obtain the following information from the file ERRDAT.000, averaged over each hour of the observation period:

- (1) BER averaged over all received messages. *(each hour)*
- (2) BER averaged for each of the three received sequences.
- (3) BER after decoding for each of the 2 coded sequences.
- (4) Proportion of errors corrected for serial and interleaved codeword sequences. *etc*

The second program was used to obtain statistical information on the error patterns stored in the file ERRPAT.000. The following information was obtained from this program:

- (1) Frequency of occurrence of consecutive errors.
- (2) Frequency of occurrence of error free intervals.
- (3) Frequency of occurrence of error bursts.

Assembler-written routines were called by the BASIC user function, USR(X), to control the reading of data blocks from the disc files.

8.8 Experimental Results

The results section of this chapter is divided into two subsections:

- (1) Statistical analysis of error patterns.
- (2) Analysis of coding performance.

The coding performance results will be seen to correlate with the error pattern analysis.

8.8.1 Analysis of error patterns

Distributions of consecutive errors, consecutive error-free intervals, and burst error occurrence were obtained from the recorded results.

For a random error distribution, the theoretical probability, P_e , that n consecutive errors occur when the average probability of a bit error is p , is given by:

$$P_e(n) = p^n(1-p)$$

This function is tabulated in table 8.1 for two chosen values of BER. For a BER of 10^{-2} it can be deduced that 99% of the errors should occur singly; for BER = 2×10^{-2} the figure is 98%.

n	1×10^{-2}	2×10^{-2}
0	9.9×10^{-1}	9.8×10^{-1}
1	9.9×10^{-3}	1.96×10^{-2}
2	9.9×10^{-5}	3.92×10^{-4}
3	9.9×10^{-7}	7.84×10^{-6}
4	9.9×10^{-9}	1.57×10^{-7}

Table 8.1

The error performance of the HF link is plotted in figure 8.9 for a 24 hour observation period. The averaged raw BER (without coding) per hour was observed to vary between 3.2×10^{-3} and 1.94×10^{-2} . Two hourly observation periods were chosen having BERs close to those used to compute the theoretical distributions; these were the periods 2300-0000 hrs. and 1400-1500 hrs. which corresponded to average bit error rates of 0.98×10^{-2} and 1.94×10^{-2} respectively. (figures 8.10(a) and (b)). The results indicated that around 75% of the errors were single, and 15% occurred in pairs. It can be seen that the measured distributions deviated substantially from the theoretical random estimation, but that the measured distributions appeared similar for both values of BER. In other words the consecutive error distribution appeared to be independent of error rate. It was also noted that the distribution appeared to be exponential. A regression analysis showed that a function conforming closely to the measured distribution is given by:

$$f_o(n) = 74 e^{(-6.2(n-1))}$$

where $f_o(n)$ is the frequency of occurrence (%) of n consecutive errors. The results indicated that the errors were tending to cluster and that the size of the clusters was independent of the average error density.

The cumulative distribution of error-free intervals was plotted for the two hourly periods analysed above and compared with the theoretical random distributions. The theoretical probability, $P_f(n)$, of an n -bit error-free interval assuming random error statistics is given by:

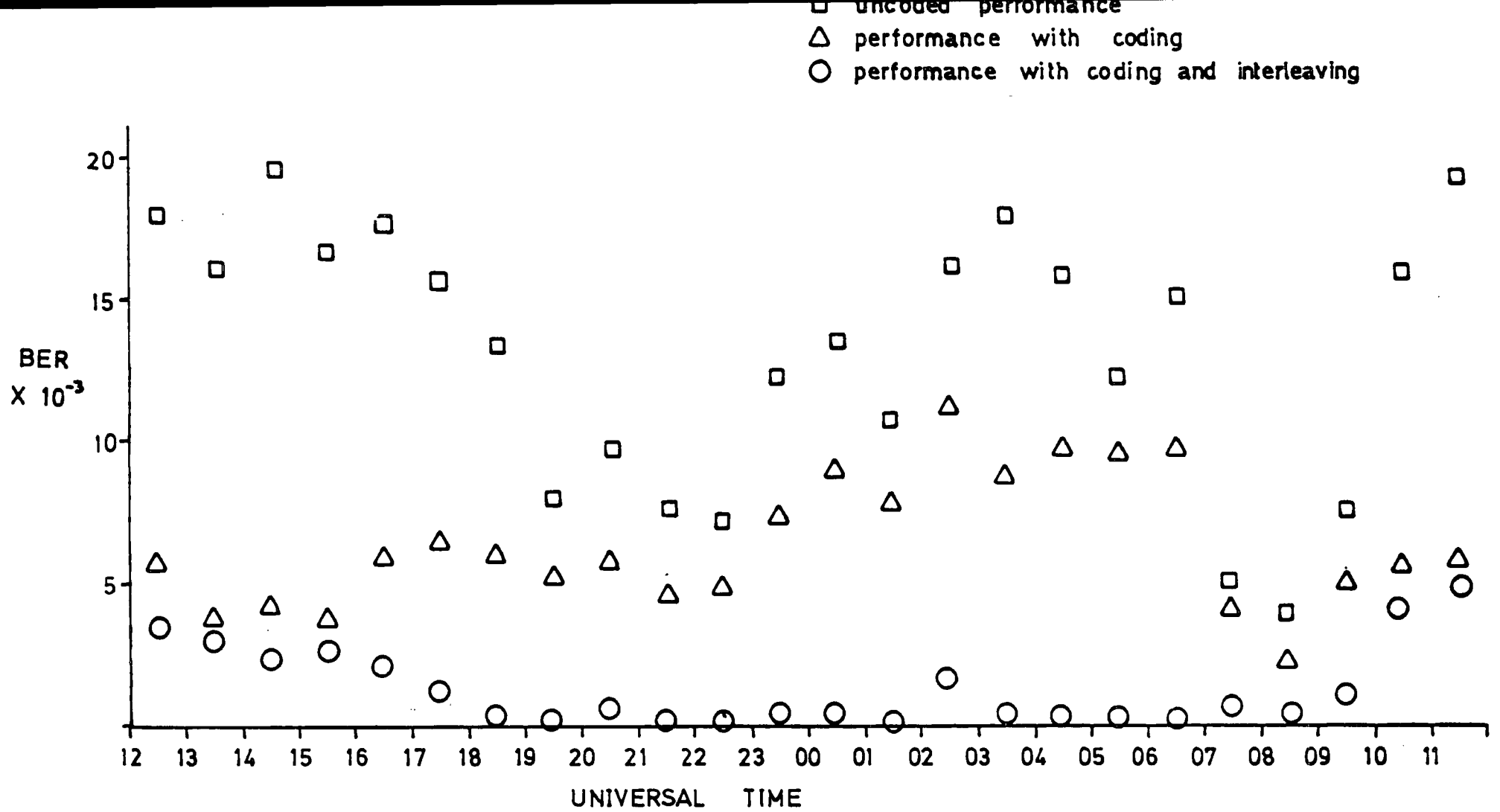


Fig. 8.9. TYPICAL ERROR PERFORMANCE OF HF LINK

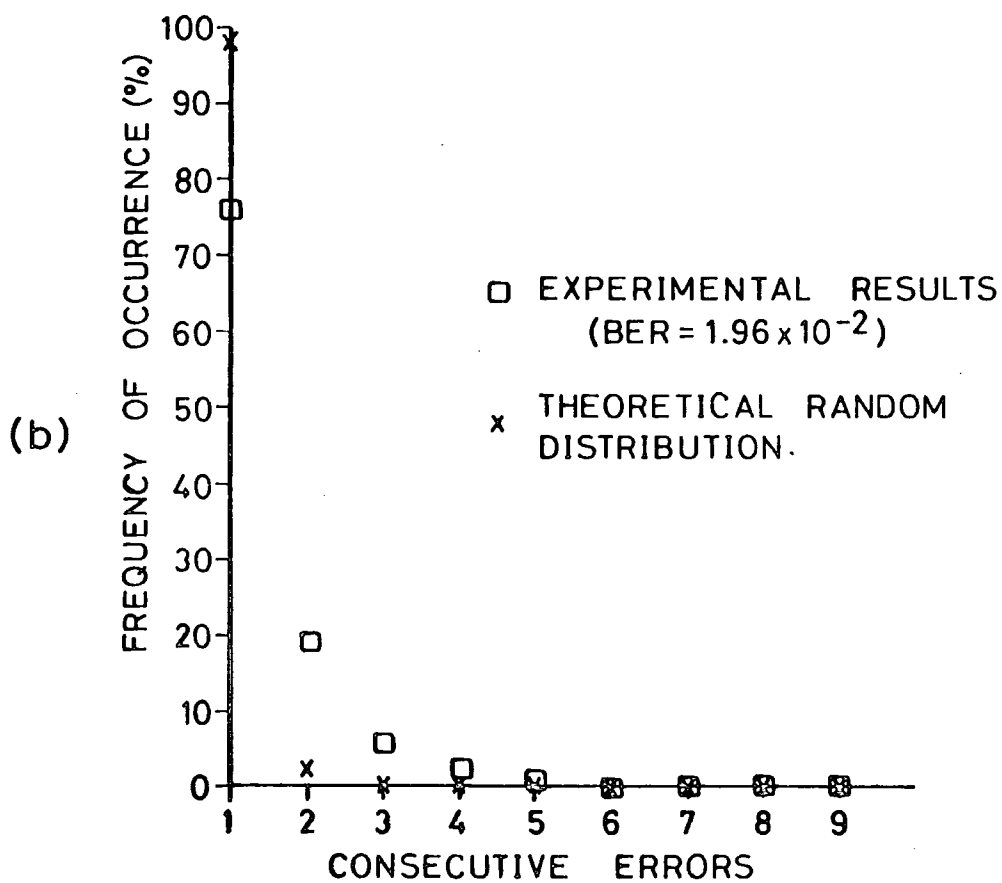
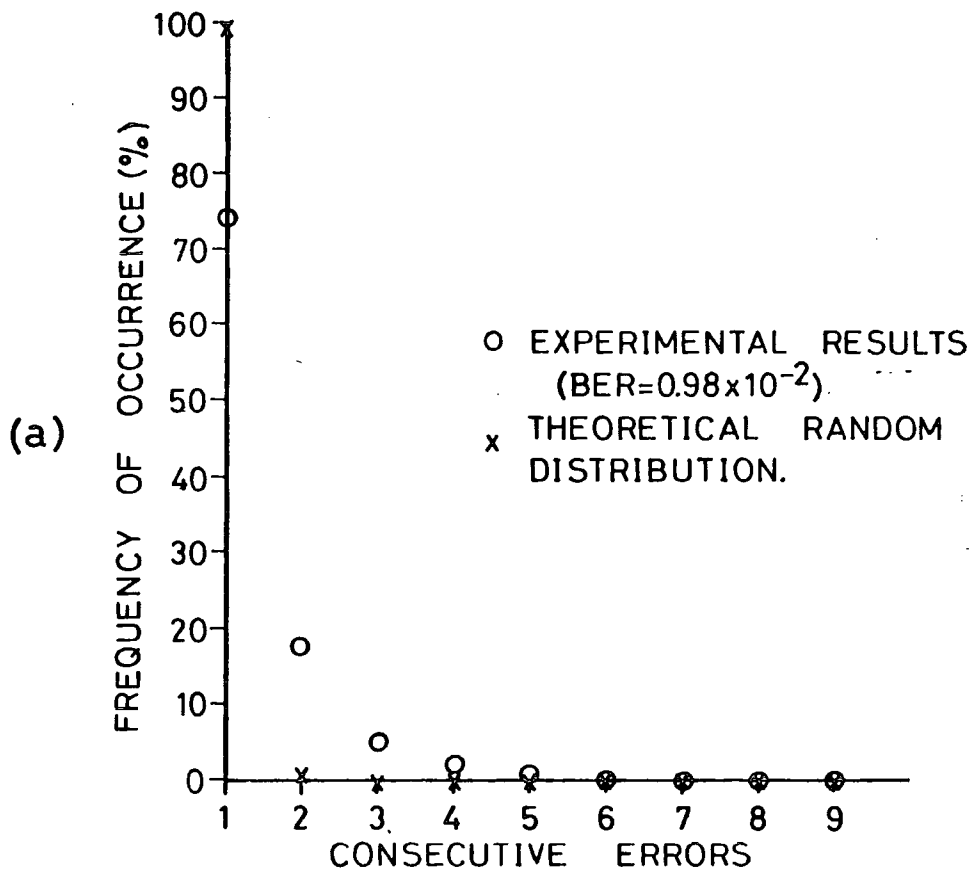


FIGURE 8.10. DISTRIBUTIONS OF CONSECUTIVE ERRORS.

$$P_f(n) = p(1-p)^n$$

where p is the probability of a bit error. Again, the results (figs 8.11 (a) and (b)) indicate substantial deviations from the random distribution, and appear to be independent of error rate. For a BER of 10^{-2} , the random distribution predicts that only 10% of the errors should be separated by a gap of 10 bits or less, whereas the measured figure was 76%.

An error "burst" is defined in a paper by Brayer (77) to be a region of the serial data stream where the following properties hold. A minimum of 2 errors are contained in the region and the minimum density of errors in the region is Δ . The burst error must always begin with a bit in error and end with a bit in error, and must be immediately preceded and followed by an interval in which the density of errors is less than Δ . The burst density criterion Δ in this paper was chosen to be 0.05. This definition of an error burst did not appear to be entirely satisfactory as the following example illustrates.

Suppose that the first 5 and the last 1 of 120 consecutive bits are received erroneously. The error density is therefore $6/120 = 0.05$ which is exactly the minimum density required to define a burst. The data would be logged as containing a burst of errors of length 120 even though there is an error-free interval of 114 bits. It would intuitively seem more reasonable to suppose that the errors were distributed as a burst of length 5 followed by a single random error occurring 114 bits later.

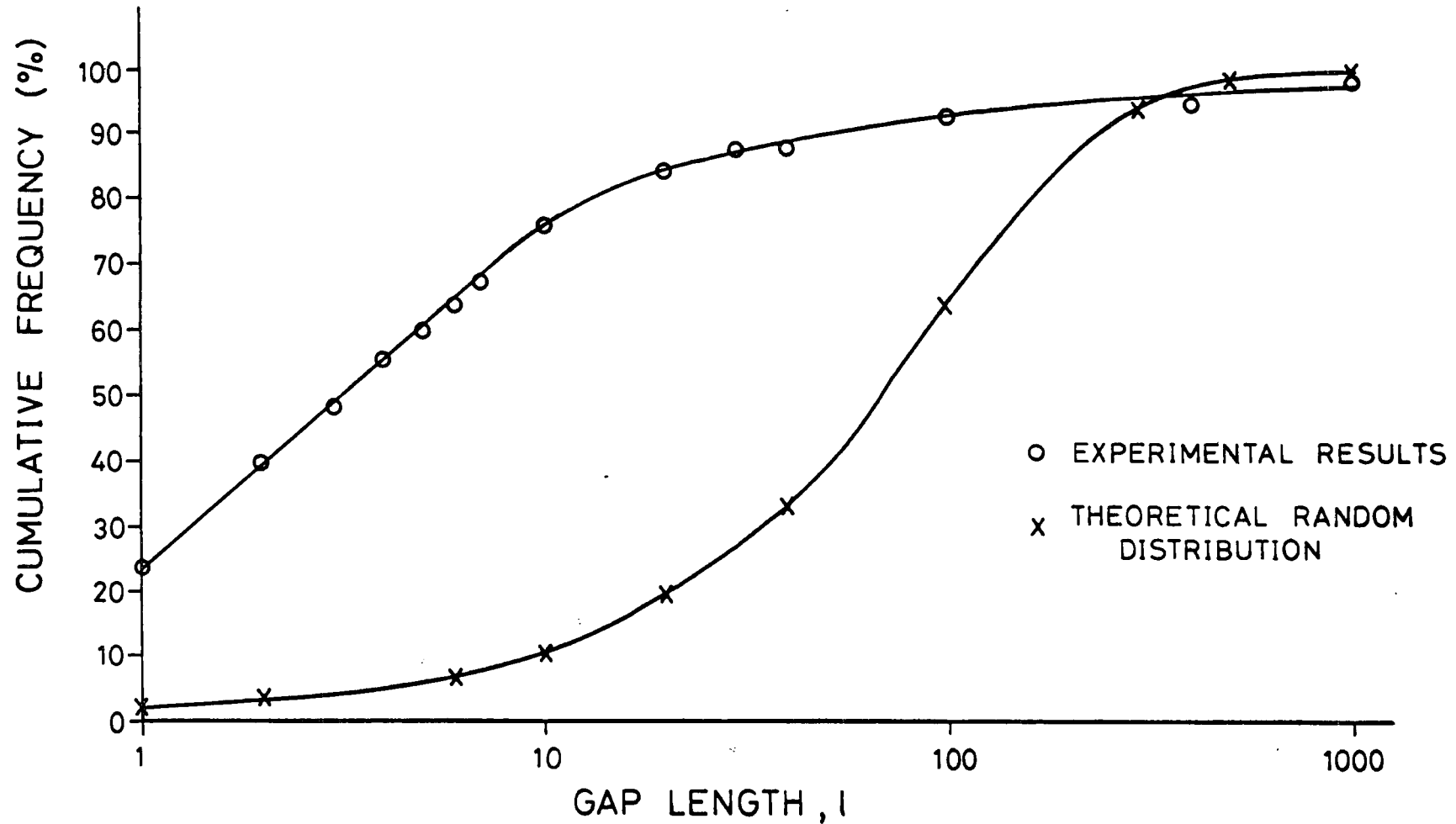


FIGURE 8.11(a). DISTRIBUTION OF ERROR-FREE INTERVALS.
 (BER = 0.98×10^{-2})

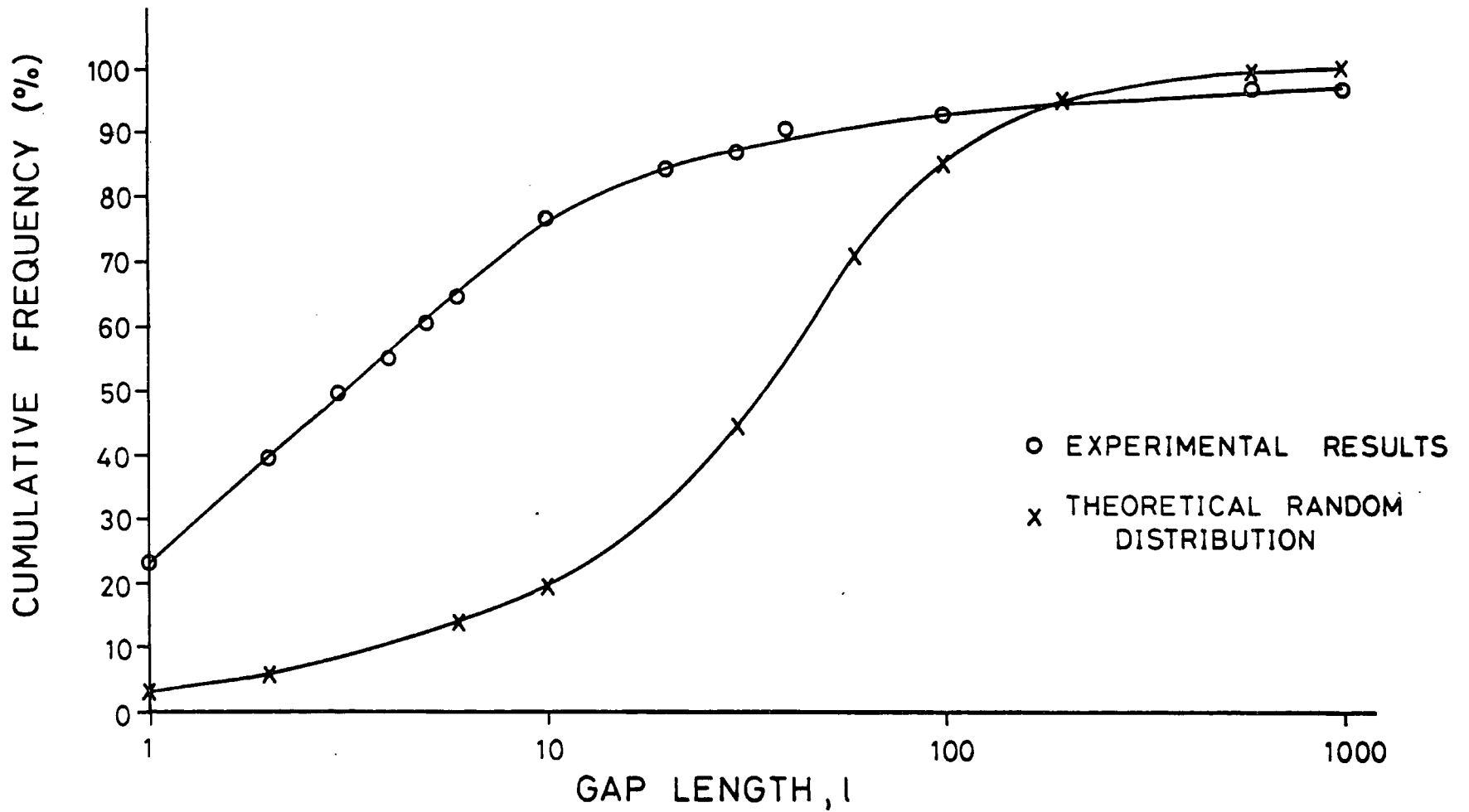


FIGURE 8.11(b). DISTRIBUTION OF ERROR-FREE INTERVALS.
 (BER = 1.96×10^{-2})

It is proposed that the following constraint be appended to the above definition: All errors contained within the burst must be separated by an error-free interval of w bits or less where $w = 2/\Delta - 1$. This ensures a more even distribution of errors within the burst. In this experiment, Δ was chosen such that an error burst would cause consecutive character errors to occur in an uncoded 7-bit ASCII character stream. For characters preceded by a single start bit and terminated with a single stop bit the burst error density criterion is $2/9 = 0.22$.

The distribution of error bursts complying with the amended definition was plotted for data averaged over a 24 hour observation period and is shown in figure 8.12. It can be seen that the distribution is an exponential decay apart from a noticeable peak at $b=10$. The unusually high proportion of error bursts of this length was attributed to regular noise bursts from locally situated machinery which were consistently observed throughout the period. Observations on an oscilloscope indicated that these were of 0.14s in duration, ie. approximately ten times the signal element duration. The averaged measured BER over the observation period was 1.35×10^{-2} .

8.8.2 Coding performance

The performance of the HF data link in terms of bit error rate has been plotted in figure 8.9 for a 24 hour observation period. The results shown illustrate the performances obtained with no coding, with forward error correction (FEC) and with FEC and interleaving of the codewords to a depth of 16. It can be seen that the improvement with interleaving exceeds the

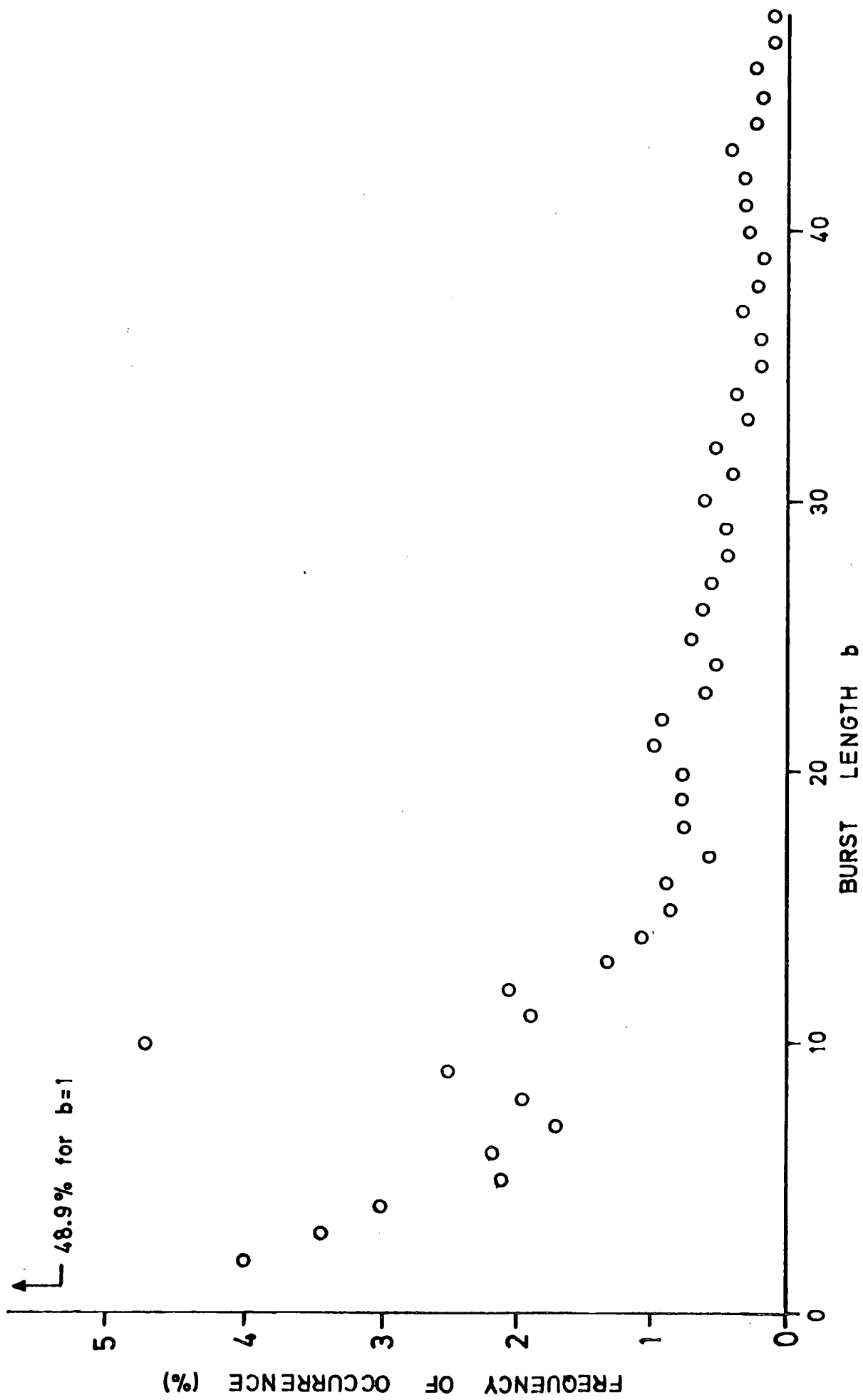


Fig.8.12. DISTRIBUTION OF ERROR BURSTS ($b > 1$)

improvement with coding alone. Figure 8.13 shows the time distribution of the proportion of total errors corrected, with and without codeword interleaving, for a different observation period. The performance with interleaving always exceeds the performance with coding alone, but the relative improvement gained is not constant. This is also evident from fig 8.9. We shall define the "interleaving gain" as:

$$20 \log_{10}(c_i(t)/c_c(t))$$

where $c_i(t)$ and $c_c(t)$ are the percentages of total errors corrected with and without interleaving respectively. A plot of this variable against time together with a plot of BER (figure 8.14) shows that it is not necessarily dependent on error rate. The interleaving gain varies between 0.2dB and 7.2dB. It appears from these results that the nature of the error structure varies with time and, for the observation period chosen, the errors tend to cluster during the night and are of a more random nature during the day. The apparent increase in the burst nature of the error structure during the night is attributed to an increase in the skip distance which exposes the receiver to static bursts and interference from further afield. During the day, short noise spikes were observed which gave rise to an increase in the number of random errors.

Plots were made of the proportion of errors corrected against bit error rate (figure 8.15), (a) for no interleaving of codewords and, (b) for an interleaving depth of 16. The same data (over 24 hours) was used for each plot. It can again be seen that the performance with coding and interleaving exceeds the performance

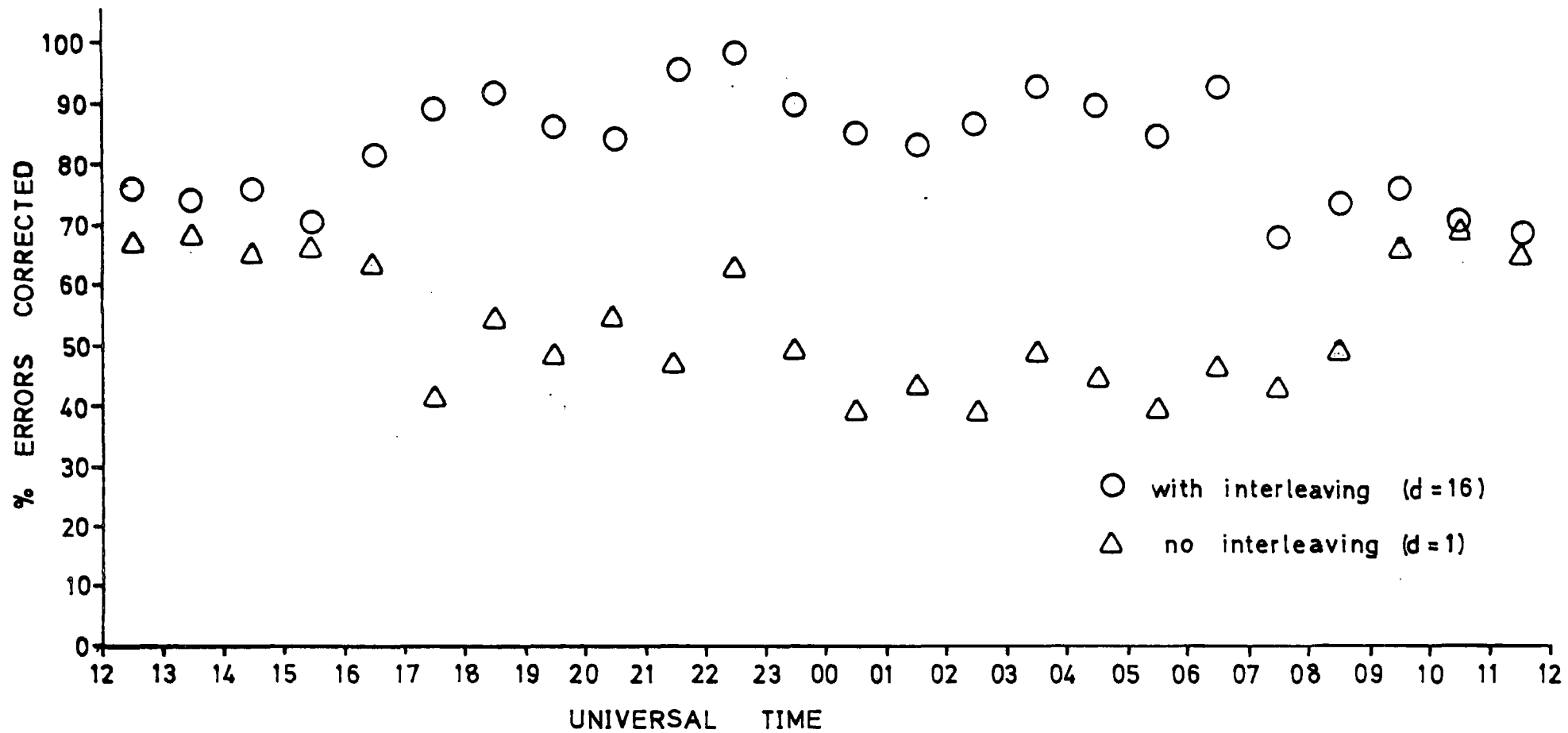


Fig. 8.13. TYPICAL DISTRIBUTION OF ERRORS CORRECTED

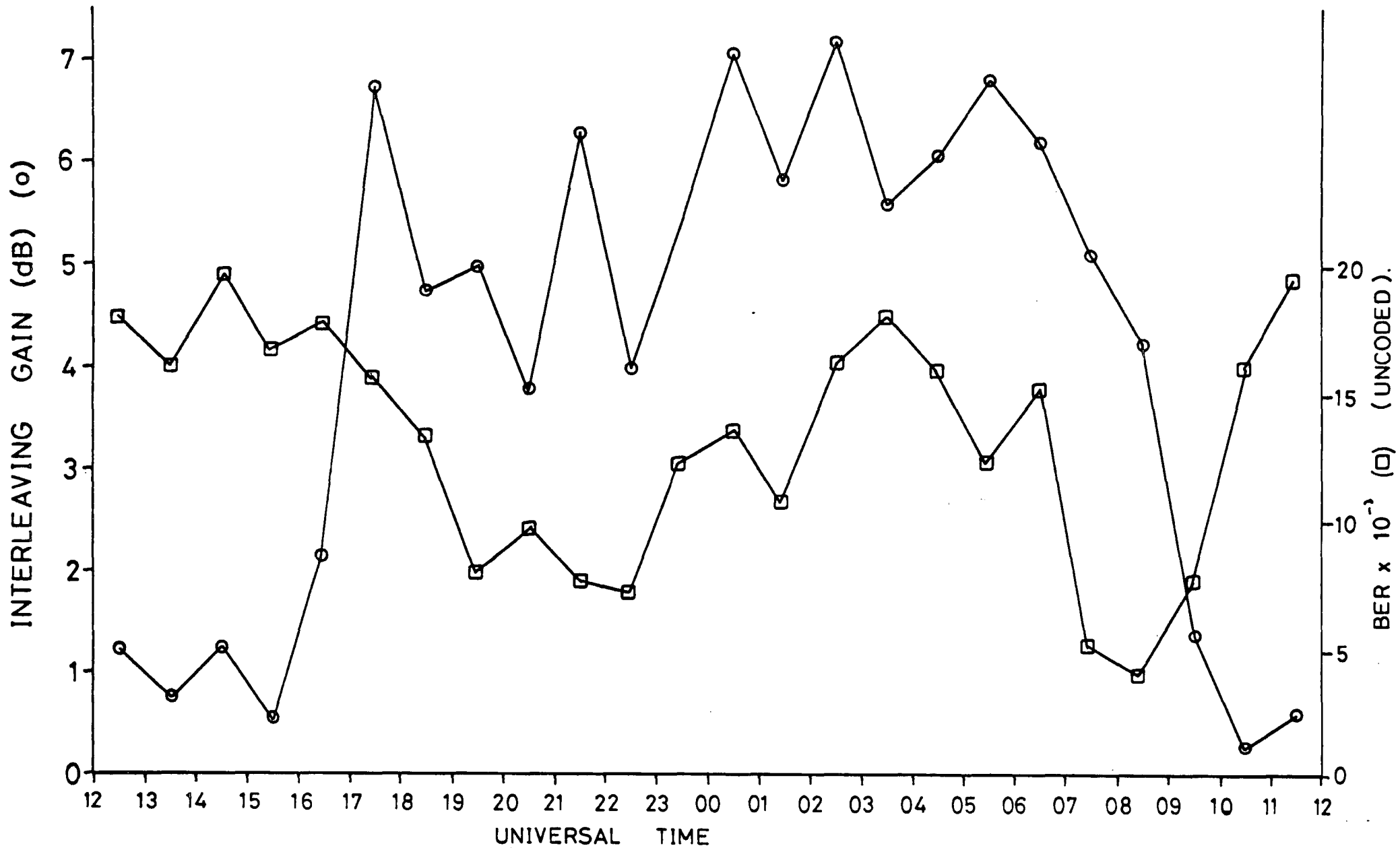
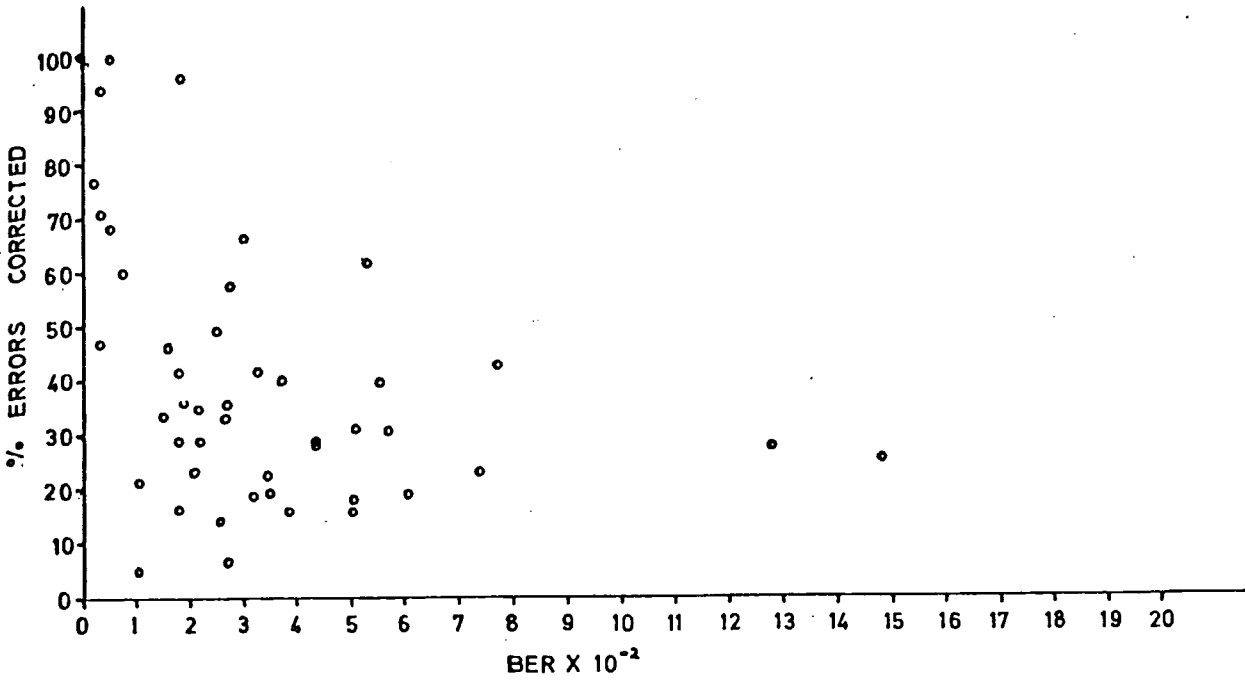
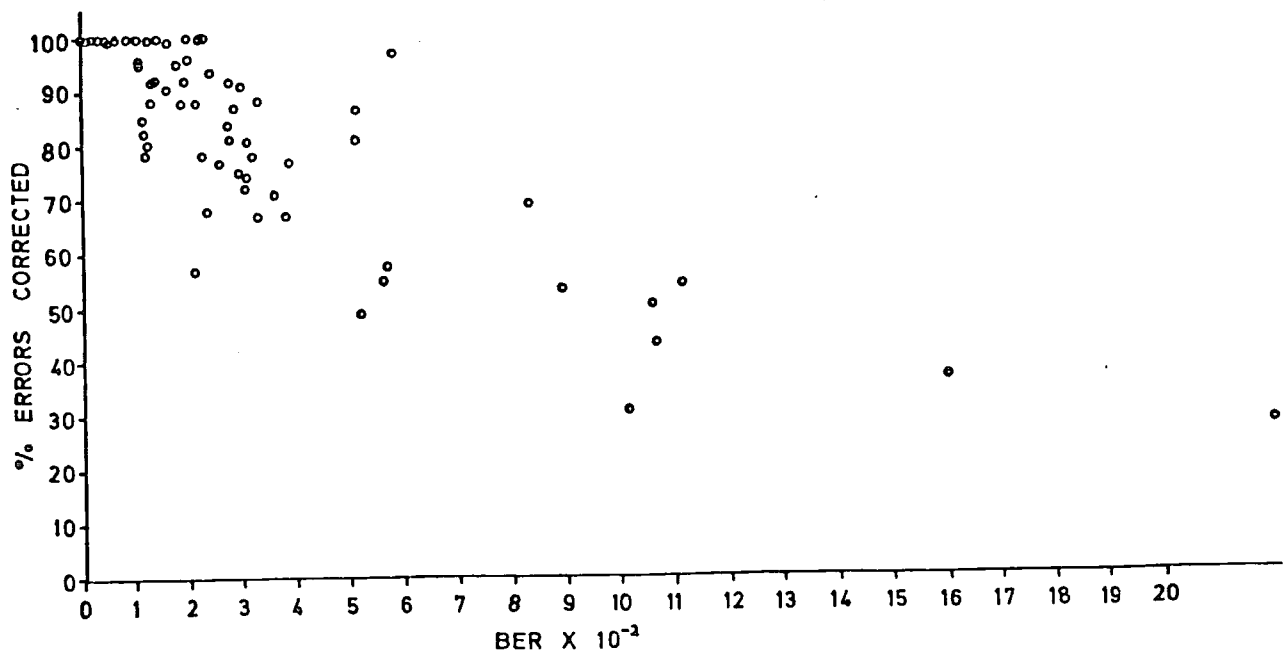


FIGURE 8.14 TIME VARIATION OF INTERLEAVING GAIN AND BER.



(a) interleaving depth = 1 (no interleaving)



(b) interleaving depth = 16

FIG. 8.15. DISTRIBUTIONS OF ERRORS CORRECTED PER MESSAGE

with coding alone, as the points tend to cluster towards the top of the plot in the latter case, and are more widely dispersed for the former.

Theoretical distributions were plotted, using the recorded data, of the proportion of errors that would be corrected for different depths of codeword interleaving. Three curves were plotted from data obtained from three intervals of 1 hour, each of which exhibited a different BER. The results were based on the assumption that if two or fewer errors occurred in a 15-bit codeword the errors would be corrected otherwise none would be corrected. Points were plotted for five values of bit interleaving. It can be seen that the relative improvement gained by using greater interleaving depths tends to decrease with increasing depth and that the curves tend to flatten for low values of BER (figure 8.16). As the interleaving depth is increased, the delay before decoding is increased. An interleaving depth of 16 appears to present a reasonable compromise between error-correcting ability, decoding delay and implementation complexity.

8.9 Conclusion

This chapter has described an experiment using microprocessor techniques undertaken to investigate the error patterns occurring over a medium-haul HF data link. The results have confirmed the view that the error statistics are significantly non-random in nature and a comparison has been made indicating precisely the deviation from the theoretical random distributions. It has also been shown that the distribution of consecutive errors and error

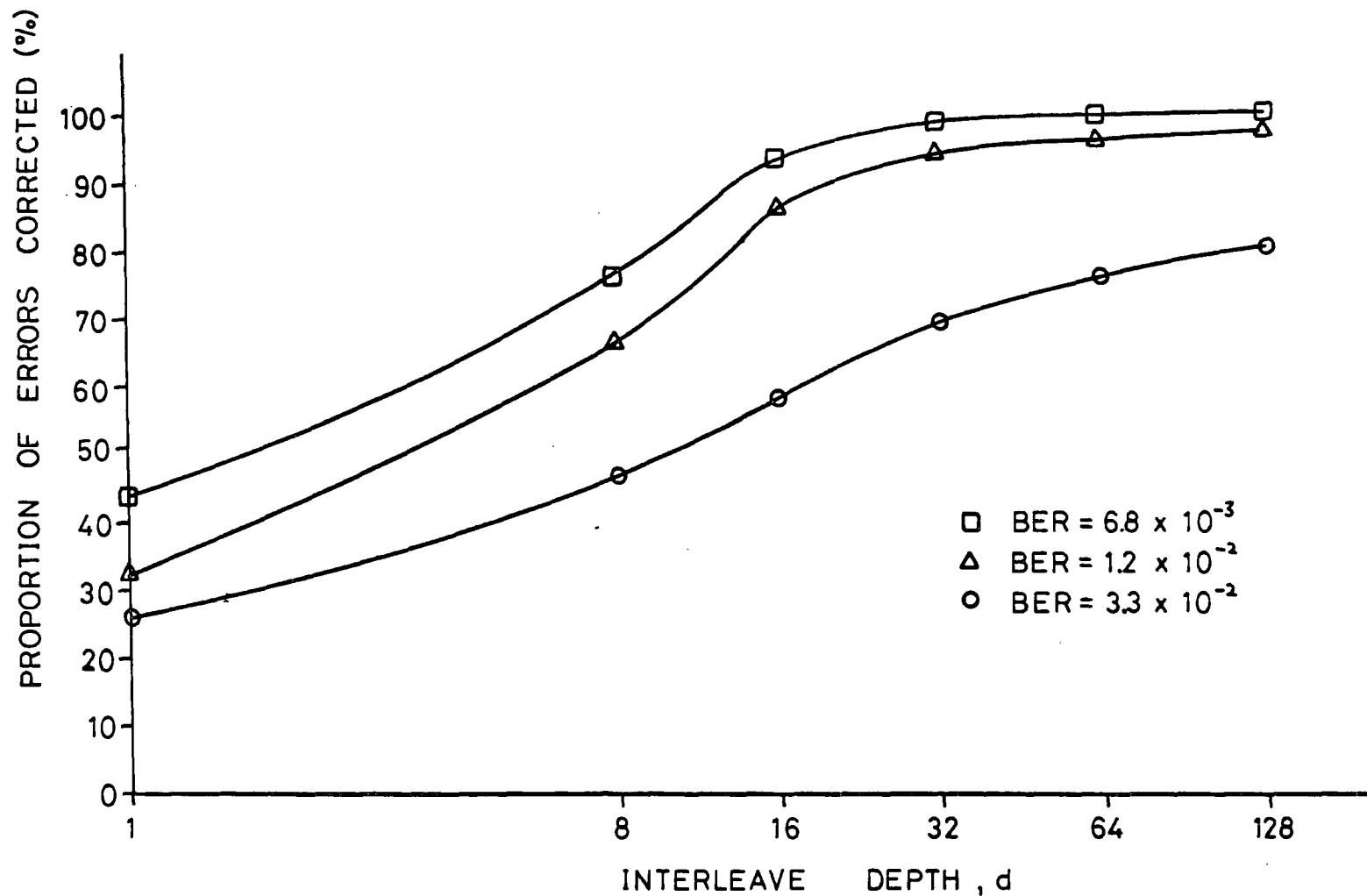


FIGURE 8.16. PROPORTION OF ERRORS CORRECTED FOR VARYING DEPTH OF CODEWORD INTERLEAVING.

free intervals appears to be independent of error rate. An investigation of error burst statistics for this experiment has shown that the distribution of burst lengths is exponential but that bursts of a fixed length may occur more frequently than expected.

An error-correcting coding scheme has been investigated using real-time encoding and decoding procedures implemented on a low-cost conventional 8-bit microprocessor. A substantial improvement in error performance was achieved by interleaving successive codewords, but the improvement in error rate was not proportional to interleaving depth. Also, the benefit gained by interleaving codewords may not be constant, indicating that the error structure is time-varying. The (15,7) BCH code interleaved to depth 16 was found to be a good compromise between error-correcting ability over the HF channel, decoding delay, and implementation complexity. In view of this, the code was chosen as the forward error correction scheme for use with the HF data modem described in chapter 7.

Shortly after the advent of digital communications over wire links, investigations were carried out into the possibility of using the refractive properties of the ionosphere as a medium for data transmission over long distances. By the early 1950's, telegraphy and telephony were well established methods for communication via HF radio links, but considerable difficulties were encountered when attempts were made to use the ionospheric medium for digital data transmission.

Acceptable error rates could be achieved only if the transmission rate was severely restricted to avoid the intersymbol interference caused by long differential multipath delays. Even so, it was found that the effects of noise, often tolerated by telephony and telegraphy systems (which are highly redundant), were to introduce large numbers of errors, occasionally of such severity as to render the channel useless for many communications purposes.

Attempts were made to find techniques which could, to some extent, overcome the detrimental effects of the HF channel. Coding, diversity, time-to-frequency division multiplexing and equalisation were all found to be effective in reducing the error rate under certain conditions, and are techniques which have been adopted in a variety of systems. Development work on systems which exhibit a certain degree of tolerance to the channel distortions was carried out in the 1950's and 60's, mainly in the U.S., and resulted in the production of a number of HF

communications terminals for military applications, notably Kineplex (5), Kathryn (6) and Codep (15). These systems were highly complex, bulky and extremely expensive to produce, and their performance was not entirely satisfactory.

Following the introduction of satellite communications during the 1960's, work in the HF field was largely abandoned in favour of this more attractive medium, which could provide reliable, high-speed data communications over long distances, albeit at high cost. However, the subsequent development of sophisticated weaponry and jamming systems led to a realisation that the satellite medium was an extremely vulnerable one, and attention was again turned to ionospheric data communications, which was often the only alternative to satellites for mobile communication over long distances. Much of the new work was concentrated on sounding techniques (11), in order to gain a better understanding of the effects of the HF channel on a digital signal. Adaptive techniques were investigated (60) to cope with the time-varying properties of the transmission medium.

The revolutionary developments in electronics technology over the last decade, notably the advances in digital electronics, have considerably eased the burden of designing complex systems for data communications (1,2). Additionally, the economic benefits to be gained using VLSI technology are often considerable. It is only in the last five years that attention has been concentrated on the possibility of implementing real-time signal processing techniques using microprocessors although such techniques have been applied mainly to line and to

VHF/UHF transmission systems (56-58). Relatively little work has been published on the direct applications of microprocessors in the field of HF data communications, which is suprising, as the inherently flexible nature of microprocessor systems make them extremely suitable for HF radio systems, where a high level of adaptability is often required.

This thesis has described an investigation into the applications of microprocessors in a variety of HF radio communications techniques. A real-time channel evaluation system has been described whereby an HF radio voice channel may be assessed as to its suitability for communication over a link. The implementation of forward error correction schemes suitable for overcoming the effects of burst errors occurring over a link have been investigated and demonstrated. This has led to the development of an adaptive modem, which uses a technique of adaptive time-to-frequency division multiplexing, together with forward error correction, to implement a low-cost system in which all the signal processing techniques have been implemented digitally. The real-time processing requirement has resulted in the development of a novel distributed processing arrangement in which a number of subsidiary ("slave") processors are under control of a central ("master") processor; the overall processing power of the system is thereby increased. Some considerable attention has been paid to the signal processing requirements of HF systems, for spectral analysis, modulation, and demodulation. This has included investigations into Fourier transformations and matched filter detection using microprocessor techniques, in addition to a study of the chirp-z transformation using a charge

coupled device.

It is hoped that this thesis has illustrated the potential of the applications of VLSI technology to the implementation of effective, low-cost systems in the field of digital HF radio communications. The complex signal processing techniques required for channel evaluation and communication over the HF medium may be realised at low cost using primarily software-based techniques. It is anticipated that further advances in the field will result in a considerable resurgence of interest in long-distance communications over HF links.

Note on publications by the author:-

"Software Cassette Interface for the SWTP 6800 Microprocessor System". Isaac, D.R. Personal Computer World, August 1979.

"Real-time Adaptive Coding of Ionospherically Propagated Data". Isaac, D.R. and Spracklen, C.T. IEEE Zurich International Seminar on Digital Communications, March 1980.

to be published:-

"A Microprocessor Implemented Trackside Recorder for British Rail". Isaac, D.R., Spracklen, C.T. & Manning, J. Journal of the Permanent Way Institution. December, 1981.

"Error Patterns and Real-time Correction Procedures for Data Transmission over HF Radio Links". Isaac, D.R. and Spracklen, C.T. IEE Conference on HF Comm. Systems and Techniques, 15-16 February, 1982.

APPENDIX 1

REFERENCES

- (1) "Digital Transmission for Radio Systems". Colavito, C. Telecomm. journal. vol. 45, July 1978.
- (2) "LSIs in Data Networks". van der May, J.E., Forney, G.D. & Stearns, R.W. Data Processing journal. April 1977, pp. 30-31.
- (3) "Experimental Confirmation of an HF Channel Model". Watterson, C.C., Juroshek, J.R., & Bensema, W.D. IEEE Trans. Comm. Tech. Vol. COM-16, no. 6. December 1970.
- (4) "Real-time Updating of MUF Predictions". Jones, T.B., Spracklen, C.T., & Stewart, C.P. AGARD Conference Proceedings no. 238, pp. 17-1 to 17-10.
- (5) "Kineplex. A Bandwidth Efficient Binary Transmission System". Mosier, R.R. & Clabough, R.G. AIEE Trans. Comm. & Electron. vol. 76 pp. 723-728. 1958.
- (6) "The AN/GSC-10 (KATHRYN) Variable Data Rate Modem for HF Radio". Zimmerman, M.S. & Kirch, A.L. IEEE Trans. Comm. Tech. vol. COM-15 part 2. pp. 197-204. 1967.
- (7) "Equalisation of Radiolink Channels with Multipath Propagation". Castel, A., Henry, M., Rolland, P., Proc. International Seminar on on Digital Comm. IEEE cat. no. 80CH1521-4 COM. Zurich, 1980.
- (8) "Telecommunications". Brown, J., & Glazier, E.V.D. Chapman & Hall, 1974.
- (9) "Radio Communication Handbook". RSGB publications, 1979.

(10) "The Improvement of Slow Rate FSK by Frequency Agility and Coding". Gott, G.F. & HILLAM, B. Proc. IEE Conf. of Recent Advances in HF Communications Systems & Techniques, Feb. 1979, pp. 19-24.

(11) "CW Sounding and its use for control of HF Adaptive Systems". Betts, J.A., Ellington, R.P., Jones, D.R.L. Proc IEE, vol 117 no. 12, December 1970. pp. 2209-2215.

(12) "Error Correcting Codes". Peterson, W.W. & Weldon, E.J.

(13) "Introduction to the Analysis and Processing of Signals". Lynn, P.A. Macmillan Press, 1973.

(14) "Medium-speed Digital Data Transmission over HF Channels". Darnell, M. Loughborough Conference on Digital Processing of Signals in Communications. Sept. 1977. pp. 293-402.

(15) "A Combined Coding and Modulation Approach for Communication over Dispersive Channels". Chase, D. IEEE Trans. Comm. Tech. Vol. COM-21, no. 3, March 1973.

(16) "Digital Signal Processing". Peled, Liu. Wiley Inc., 1976.

(17) "Motorola M6800 Microprocessor Programming Manual". Motorola Publications, 1978.

(18) "SWTP Computer Systems". South-West Technical Products, San Antonio, Texas.

(19) "High Speed Software Cassette Interface for the SWTP 6800 System". Isaac, D.R. Personal Computer World, August 1979. pp. 39-43.

- (20) "Microcomputer Components". Motorola Publications, 1979.
- (21) "Engineering Note 100". Motorola Publications, 1978.
- (22) "Digital Signal Processing". Oppenheim, A.V. & Schafer, R.W. Prentice-Hall Inc. 1975.
- (23) "Digital Signal Processing". Stanley. Reston, 1975.
- (24) "Introduction to Digital Filters". Bogner, R.E. & Constantinides, A.G. Wiley Press, 1975.
- (25) "An Algorithm for the Machine Calculation of Complex Fourier Series". Cooley, J.W. & Tukey, J.W. Math. Computation, Vol. 19, 1965, pp. 297-301.
- (26) "The Chirp-Z Transform Algorithm". Rabiner, L.R., Schafer, R.W., Roder, C.M. IEEE Trans. Audio Electroacoust., Vol. AU-17, June 1969.
- (27) "A Linear Filtering Approach to the Computation of the Discrete Fourier Transform". Bluestein, L.I. IEEE Trans. Audio Electroacoust., Vol. AU-18, December, 1970.
- (28) "M6800 Microprocessor Applications Manual". Motorola Publications, 1975.
- (29) "Comparison between the CCD Chirp-Z Transform and the Digital FFT". Buss, D.D., Broderon, R.W., Hewes, C. Proc. 1975 Naval Electronics Lab. Center Int. Conf. on Applications of CCDs.

- (30) "A 500-stage CCD Transversal Filter for Spectral Analysis". Broderson, R.W., Hewes, C.R., Buss, D.D. IEEE Trans. Electronic Devices vol ED-23 pp. 143-152. Feb. 1976.
- (31) "The ABCs of CCDs". Kosonocky, W.F. & Sauer, J. Electronic Devices, vol. 23 pp. 58-63, April 1975.
- (32) "Real Time Spectral Analysis using Hardware Fourier and Chirp-Z Transformations". Benjamin, R. "Radio & Electronic Engineer". vol. 49, no. 2. pp. 101-107. Feb. 1979.
- (33) "Communications Applications of CCD Transversal Filters". Buss, D.D., Broderson, R.W., Hewes, C.R., Tasch, A.F. IEEE Nat. Telecomm. Conf. Record, vol. 1. Dec. 1-3, 1975. pp. 1.1-1.5.
- (34) "Interfacing a Hardware Multiplier to a General-Purpose Microprocessor". Davies, A.C. & Fung, Y.T.
- (35) "A High-Speed FFT Processor". Zaheer, M. Ali. IEEE Trans. Comm. Vol. COM-26, no. 5. May 1978.
- (36) "The Fadeogram, a sonogram-like display of the time-varying frequency response of HF SSB radio channels". Filter, J.H.J, Arazi, B., Thomson, R.G.W. IEEE Trans. Comm. June 1978 pp. 913-917.
- (37) "The Fadeogram - Applied to analysing HF data modem performance". Thomson, R.G.W. & Filter, J.H.J. Proc. IEE Conf. on Recent Advances in HF Communications Systems & Techniques. Feb. 1979. pp. 19-24.

- (38) "M6800 Microprocessor Applications Manual". Motorola Publications, 1975.
- (39) "The M6800 Microprocessor Unit". Motorola Data Sheet.
- (40) "Cyclic Codes for Error Detection". Peterson, W.W., Brown, D.T. Proc. IRE Jan. 1961. pp. 228-235.
- (41) "Principles of Data Communications". Lucky, R.W., Salz, J., Weldon, E.J. McGraw-Hill, 1968.
- (42) "Further Results on Error Correcting Binary Group Codes". Bose, R.C. & Ray-Chaudhuri, D.K. IEEE Trans. Inf. & Control. Vol. IC-3, 1960. pp. 279-90.
- (43) "Encoding and Error Correction Procedures for the BCH Codes". Peterson, W.W. IRE Trans. Inf. Theory. September, 1960. pp. 459-470.
- (44) "Cyclic Decoding Procedures for BCH Codes". Chien, R.T. IEEE Trans. Inf. Theory. October, 1964. pp. 357-363.
- (45) "On Decoding Binary BCH Codes". Berlekamp, E.R. IEEE Trans. Inf. Theory, vol. IT-11, no. 4. October, 1965.
- (46) "Computer Implementation of Decoders for Several BCH Codes". Michelson, A.M. Polytech. Inst. of Brooklyn int. symposium on computer proc. in comm. April, 1969.
- (47) "Practical Algebraic Decoders". Berlekamp, E.R. IEEE Trans. Inf. Theory.

- (48) "Processor Implemented Decoders for Block & Convolutional Codes". Michelson, A.H. & Boyd, T.H. Proc. EASCON '74. pp. 353-360.
- (49) "Generalised Burst-Trapping Codes". Burton, H.O., Sullivan, D.D., Shih Yung Tong. IEEE Trans. Inf. Theory. Vol. IT-7, no. 6, Nov. 1971.
- (50) "Burst-Trapping Techniques for a Compound Channel". Shih Yung Tong. IEEE Trans. Inf. Theory. Vol. IT-15, no. 6. Nov. 1969.
- (51) "Improving the Reliability of HF Data Transmissions". Sloggett, D.R. Proc. Conf. on Recent Advances in HF Comm. Systems & Techniques. Feb. 1979. pp. 74-78.
- (52) "Improved Signal Design for Fading Channels by Coding". Chase, D. CNR Inc. Newton, Mass..
- (53) "Characteristics of Interfering Signals in Aeronautical HF Voice Channels". Gott, G.F., Staniforth, M.J.D. Proc. IEE vol. 125, no. 11. Nov. 1978. pp. 1208-1212.
- (54) "An HF Data Modem with in-band Frequency Agility". Darnell, M. IEE Proc. Conf. on Recent Advances in HF Communications Systems & Techniques. Feb. 1979. pp. 19-24.
- (55) "Data at Twice the Speed Eases HF Traffic Jam". Porter, G.C., Gray, M.B., Perkett, C.E. "Electronics" journal. October 1967. pp. 115-120.

(56) "A 4800 bps Microprocessor Data Modem". Watanabe, K., Inoue, K., Sato, Y. IEEE Trans. Comm. Tech. vol. COM-26 no. 5. May 1978.

(57) "An Experimental Microprocessor-Implemented 4800 bps Limited Distance Voiceband PSK Modem". Stroh, R.W. IEEE Trans. Comm. Tech. vol. COM-26, no. 5. May 1978. pp. 507-512.

(58) "Microprocessor Implementation of Tactical Modems for Data Transmission over VHF Radio". Davies, B.H. & Davies, T.R. "Radio & Electronic Engineer", vol. 49, no. 4. pp. 204-210. April, 1979.

(59) "Microprogrammed HF Modem". Smith, S.D., Dimond, K.R., Farrell, P.G. IEE Conference on the Impact of new LSI techniques on Comm. Systems., Oct. 1980.

(60) "Data Transmission with Variable-Redundancy Error Control over an HF Channel". Goodman, V.M.F. & Farrell, P.G. Proc. IEE. Vol. 122, no. 2. February, 1975.

(61) "Real-time Adaptive Coding of Ionospherically Propagated Data." Isaac, D.R. & Spracklen, C.T. Proc. International Seminar on Digital Communications. IEEE cat. no. 80CH1521-4 COM. pp. G5.1 - G5.6. March 1980.

(62) "Selective Fading Limitations of the Kathryn Modem and some Design Considerations". Bello, P.A. IEEE Trans. Comm. Tech. vol. 13, no. 3. September 1965.

(63) "Field Test Results of the AN/GSC-10 (KATHRYN) Digital Data Terminal". Kirsch, A.L., Gray, P.R. Hanna, D.W. IEEE Trans. Comm. Tech. Vol. COM-17, no. 2. April, 1969.

(64) "Voiceband 4-Phase Data Set for Global Communications". Horlacher, R.L. & Schroeder, H.C. IEEE Trans. Cmm. Tech. Vol. COM-17 no. 3. June 1969.

(65) "Error Performance of Quadrature Pilot-tone Phase Shift Keying". Bussgang, J.J. & Leiter, M. IEEE Trans. Comm. Tech. Vol. COM-16. no. 4, August 1968.

(66) "Digital Angle Modulation". Thompson, R. & Clouting, D.R. "Wireless World", December 1976. pp. 71-76.

(67) "An Integrated HF Error-Controlled Modem". deLellis, J. & Michelson, A. Proc. 11th International Conference of Communications. San Fransisco, Calif. June 1975. pp. 13-6 to 13-10.

(68) "A Program Controlled HF Modem Implementation". deLellis, J., Fast, D., Michelson, A., Carmichael, W.. EASCON '74 Record, Washington D.C. October 1974. pp. 349-352.

(69) "Design and Performance of a new Adaptive Serial Modem on a Simulated Time-variable Multipath HF Link". IEEE Annual Conference Record, Boulder, June 1975, pp. 769-773.

(70) "Use of Pilot Tones for Real Time Channel Estimation of HF Data Circuits". Betts, J.A., Broom, R.S., Cook, S.J., Clark, J.G. Proc. IEE, vol. 122, no. 9, Sept. 1975.

(71) "Principles of Digital Data Transmission". Clark, A.P. Pentech Press, 1976.

(72) "The Improvement of Digital HF Communications through Coding". Brayer, K. IEEE Trans. Comm. Tech. Vol. COM-16., no. 6. Dec. 1968.

(73) "Application of Forward-Error Correction to a Rayleigh Fading Communication Channel". Treciokas, R. Proc. IEE, Vol. 125, no. 3. March, 1978.

(74) "Effective Application of Forward Error Control Coding to Multichannel HF Data Modems". Pierce, A.W., Barrow, B.B., Goldberg, B., Tucker, J.R. IEEE Trans. Comm. Tech. Vol. COM-18, No. 4, August, 1970.

(75) "Soft-decision Error Control for HF Data Transmission". Farrell, P.G. IEE Proc. Vol. 127, pt. F. no. 5 October 1980.

(76) "Disk Systems" manual. Smoke Signal Broadcasting Corporation, Westlake Village, Calif., USA.

(77) "Error Patterns Measured on Transequatorial HF Communications Links". Brayer, K. IEEE Trans. Comm. Tech., Vol. COM-16, no. 2. April, 1968. pp. 215-221.

APPENDIX 2

PROGRAM LISTINGS

listings for chapter 2

(1) BASIC FFT

(2) Assembler FFT

371 PRINT "TRANSFORMED DATA":GOSUB 1990:PRINT "N. REAL. IMAGE"

```

372 PRINT "-----"
373 PRINT
380 N=20
390 FOR I=1 TO N
400 PRINT I,A(1),A(2),I
410 NEXT I
420 X=11:GOSUB 1990
430 INPUT "OUTPUT DATA PLOTOUT:":A$
432 IF A$="YES" THEN 435
433 IF A$="NO" THEN GOSUB 2410
434 PRINT "INVALID REPLY":GOTO 430
435 M1=15:M2=15
436 M3=M1*M2
437 FOR I=1 TO M
440 IF A(1) < M1 THEN M1=A(1),I
441 IF A(1) < M2 THEN M2=A(1),I
442 IF A(2) < M3 THEN M3=A(2),I
443 IF A(2) < M4 THEN M4=A(2),I
444 NEXT I
450 X=5:GOSUB 1990
451 GOSUB 3000
460 GOSUB 1000
470 GOSUB 1500
480 M=200
490 L1=20
500 L2=10
510 L3=10
520 L4=10
530 L5=10
540 L6=10
550 L7=10
560 A(2,1)=A(1),L2
570 A(2,1)=A(1),L2
580 A(2,1)=A(1),L2
590 FOR I=2 TO L1
600 A(2,I)=A(1),I
610 A(2,I)=A(1),I
620 A(2,I)=A(1),I+2
630 A(2,I)=A(1),I+2
640 A(1,I)=A(1),I+2
650 A(2,I)=A(1),I+2
660 A(1,N-I+2)=A(1),I+2
670 A(2,N-I+2)=A(1),I+2
680 NEXT I
690 FOR I=2 TO L3
700 A(1,I)=A(1),I+1
710 A(2,I)=A(2),I+1
720 NEXT I
730 A(1,N)=A1
740 A(2,N)=A2
750 RETURN
1010 FOR I=1 TO N
1020 A(1,I)=A(1),I/N
1030 A(2,I)=A(2),I/N
1040 NEXT I
1050 RETURN
1500 M=20
1510 L1=N-1
1520 L2=N-2
1530 %M INPLACE SHUFFLE OF DATA BESSING
1540 J=1
1550 FOR I=1 TO L1
1560 IF I >= J THEN 1530
1570 A(1,I),A(1,J)

```

```

11507 MD. OF DATA POINTS AS POWER OF 2:11
11508 M=20
11509 DIM A(1),A(2),I=1 TO M
11510 PRINT "ENTER 2-DIMENSIONAL DATA"
11511 FOR I=1 TO 2
11512 FOR J=1 TO M
11513 A(I,J)=RND(1)*255-0.5
11514 NEXT J
11515 NEXT I
11516 INPUT "OUTPUT DATA PLOTOUT:":A$
11517 IF A$="YES" THEN 11520
11518 IF A$="NO" THEN GOTO 11530
11519 PRINT "INVALID REPLY: YES AND ONLY: YES AND ONLY: GOTO 30"
11520 INPUT "EAST FOURIER TRANSFORM":A$
11521 IF A$="YES" THEN GOSUB 1990
11522 PRINT "ORIGINAL DATA: POINT"
11523 PRINT "N. REAL. IMAGE"
11524 PRINT "-----"
11525 PRINT
11526 FOR I=1 TO M
11527 FOR J=1 TO M
11528 A(1,I)=A(1,I)+A(2,I):J=J+2:NEXT J
11529 GOTO 11530
11530 M=200
11531 L1=20
11532 L2=10
11533 L3=10
11534 L4=10
11535 L5=10
11536 L6=10
11537 L7=10
11538 A(2,1)=A(1),L2
11539 A(2,1)=A(1),L2
11540 A(2,1)=A(1),L2
11541 FOR I=2 TO L1
11542 A(2,I)=A(1),I
11543 A(2,I)=A(1),I
11544 A(2,I)=A(1),I+2
11545 A(2,I)=A(1),I+2
11546 A(1,I)=A(1),I+2
11547 A(2,I)=A(1),I+2
11548 A(1,N-I+2)=A(1),I+2
11549 A(2,N-I+2)=A(1),I+2
11550 NEXT I
11551 FOR I=2 TO L3
11552 A(1,I)=A(1),I+1
11553 A(2,I)=A(2),I+1
11554 NEXT I
11555 A(1,N)=A1
11556 A(2,N)=A2
11557 RETURN
11558 FOR I=1 TO N
11559 A(1,I)=A(1),I/N
11560 A(2,I)=A(2),I/N
11561 NEXT I
11562 RETURN
11563 M=20
11564 L1=N-1
11565 L2=N-2
11566 %M INPLACE SHUFFLE OF DATA BESSING
11567 J=1
11568 FOR I=1 TO L1
11569 IF I >= J THEN 11566
11570 A(1,I),A(1,J)

```

```

1520 A2=A(2,I)
1530 A(1,I)=A(1,I)
1540 A(2,I)=A(2,I)
1550 A(1,I)=A1
1560 A(2,I)=A2
1570 L=L2
1580 IF L=J THEN 1590
1590 J=J+1
1600 L=L+2
1610 GOTO 1540
1620 J+L-NEXT I
1630 REM SHUFFLE COMPLETE
1640 REM TRANSFORM BY DECIMATION IN TIME BEGINS
1650 FOR I=1 TO N
1660 L1=2*(I-1)
1670 L2=2*(N-I)
1680 A(3)=2*A1+L2-N
1690 C1=1
1700 C2=0
1710 C3=0
1720 C4=0
1730 C5=0
1740 C6=0
1750 C7=0
1760 C8=0
1770 C9=0
1780 C10=0
1790 FOR M=1 TO L1
1800 FOR L=L2 TO L2
1810 L3=L-1+(L-1)*2*L1+1
1820 S1=A(1,L2)
1830 S2=A(2,L2)
1840 S1=A(1,L3+L1)
1850 S2=A(2,L3+L1)
1860 A1=S1+C1+C2+C3+S1
1870 A2=S2+C1+C2+C3+S2
1880 A(1,L3)=A1+A1
1890 A(2,L3)=A2+A2
1900 A(1,L3+L1)=S1-A1
1910 A(2,L3+L1)=S2-A2
1920 NEXT L
1930 C1=C1+C2+C3+S1+S2
1940 C2=C1+C2+C3+S1+S2
1950 C3=C2
1960 C4=C3
1970 NEXT M-NEXT I
1980 RETURN
1990 FOR J2=1 TO N:PRINT:NEXT J2:RETURN
2000 PRINT "TRANSFORM PLOTOUT"
2010 PRINT "-----"
2020 N=2:GOSUB 1990
2030 PRINT "SCALING DATA"
2040 PRINT
2050 PRINT "REAL MAX=";M1
2060 PRINT "REAL MIN=";M2
2070 PRINT "IMAG MAX=";M3
2080 PRINT "IMAG MIN=";M4
2090 PRINT
2100 PRINT "SYMBOLS USED...REAL>>>>(*)"
2110 PRINT "          IMAG>>>>(.)"
2120 PRINT "          AXIS>>>>(.)"
2130 PRINT "REAL DATA"
2140 L1=M1-M2:L2=M3-M4
2150 FOR I=1 TO N:A(1,I)=(A(1,I)-M2)*64/L1:A(2,I)=(A(2,I)-M4)*64/L2:NEXT
2160 FOR I=1 TO 64:PRINT ".":NEXT I:PRINT
2170 M2=ABS(M2/L1):M4=ABS(M4/L2):M2=M2*64:M4=M4*64
2180 FOR I=1 TO N
2190 IF INT(M2)>INT(A(1,I)) THEN PRINT TAB(M2);"+":GOTO 2220
2200 IF INT(M2)>INT(A(1,I)) THEN PRINT TAB(A(1,I));"+";TAB(M2);".":GOTO 2220

```

```

2210 PRINT TAB(M2);".":TAB(A(1,I));"+*
2220 NEXT I
2225 XX=5:GOSUB 1990
2230 PRINT "IMAG DATA"
2240 FOR I=1 TO 64:PRINT ".":NEXT I:PRINT
2250 FOR I=1 TO N
2260 IF INT(M4)>INT(A(2,I)) THEN PRINT TAB(M4);"+":GOTO 2300
2270 IF INT(M4)>INT(A(2,I)) THEN PRINT TAB(A(2,I));"+";TAB(M4);".":GOTO 2300
2280 PRINT TAB(M4);".":TAB(A(2,I));"+*
2290 NEXT I
2300 GOSUB 1990
2310 INPUT "POWER SPECTRUM PLOTOUT";A3
2320 IF A3="YES" THEN 2450
2330 IF A3="NO" THEN 2450
2340 PRINT "INVALID REPLY":GOTO 2410
2350 FOR I=1 TO N
2360 A(1,I)=A(1,I)^2+A(2,I)^2
2370 NEXT I
2375 IF A3="NO" THEN 2590
2380 GOSUB 1990
2390 PRINT "POWER SPECTRUM PLOTOUT"
2400 PRINT "-----"
2410 PRINT "N","POWER","N","POWER"
2420 PRINT "-","-----","-","-----"
2430 PRINT
2440 K=1
2450 PRINT K,A(1,K),K+1,A(1,K+1)
2460 K=K+2
2470 IF K>N THEN 2590
2480 GOTO 2450
2490 PRINT
2500 INPUT "POWER SPECTRUM PLOTOUT";A3
2510 IF A3="YES" THEN 2625
2520 IF A3="NO" THEN 3000
2530 PRINT "INVALID REPLY":GOTO 2590
2540 GOSUB 1990
2550 PRINT "POWER SPECTRUM PLOTOUT"
2560 PRINT "-----"
2570 PRINT
2580 M1=-1E6:M2=1E6
2590 FOR I=1 TO N
2600 IF A(1,I)>M1 THEN M1=A(1,I)
2610 IF A(1,I)<M2 THEN M2=A(1,I)
2620 NEXT
2630 PRINT "SCALING DATA"
2640 PRINT
2650 PRINT "MAX=";M1
2660 PRINT "MIN=";M2
2670 PRINT
2680 PRINT "SYMBOLS USED...POWER>>>>(L)"
2690 PRINT "          AXIS>>>>(.)"
2700 PRINT "POWER"
2710 FOR I=1 TO 64:PRINT ".":NEXT I:PRINT
2720 L1=M1-M2
2730 FOR I=1 TO N:A(1,I)=(A(1,I)-M2)*64/L1:NEXT
2740 M2=ABS(M2/L1):M2=M2*64
2750 FOR I=1 TO N
2760 IF INT(M2)>INT(A(1,I)) THEN PRINT TAB(M2);"+":GOTO 2870
2770 IF INT(M2)>INT(A(1,I)) THEN PRINT TAB(A(1,I));"+";TAB(M2);".":GOTO 2870
2780 PRINT TAB(M2);".":TAB(A(1,I));"+L"
2790 NEXT I
2800 GOSUB 1990
2810 RETURN

```

```

1:      NAM      FAST FOURIER TRANSFORM
2:      OPT      NOS+LIS+PAG
3:      *-----*
4:      * FAST FOURIER TRANSFORM
5:      *-----*
6:      *
7:      * THIS PROGRAM COMPUTES AN N-POINT DFT OF THE COMPLEX
8:      * SEQUENCE STORED IN THE DATA TABLE AS FOLLOWS:
9:      * DBASE FCOY REAL (MSB)
10:     * DBASE+1 FCOY REAL (LSB)
11:     * DBASE+2 FCOY IMAG (MSB)
12:     * DBASE+3 FCOY IMAG (LSB)   ETC.
13:     * N IS AN INTEGRAL POWER OF 2 BETWEEN 8 AND 256 INCLUSIVE.
14:     * R = LOG2(N)
15:     *-----*
16:     *
17:     * VARIABLE STORAGE AREA:
18:     *
19:     *   ORG 0
20:     *   RMB 1          LOG2(N)
21:     *   RMB 1          NO. OF POINTS/2
22:     *   RMB 1
23:     *   RMB 2          DATA TABLE INDEX
24:     *   RMB 2          DATA INDEX
25:     *   RMB 2          BUTTERFLY DATA INDEX
26:     *   RMB 2          LOOKUP TABLE INDEX
27:     *   RMB 2          LOOKUP TABLE POINTER
28:     *   RMB 1
29:     *   RMB 1
30:     *   RMB 1
31:     *   RMB 1
32:     *   RMB 1
33:     *   RMB 2          TEMP REAL STORE
34:     *   RMB 2          TEMP IMAG STORE
35:     *   RMB 2          MULTIPLIER
36:     *   RMB 2          MULTIPLICAND
37:     *   RMB 4          PRODUCT
38:     *   RMB 1          TEST BYTE
39:     *
40:     * DATA STORAGE AREA:
41:     *
42:     *   ORG $0400
43:     *   RMB $0400
44:     *
45:     * LOOKUP TABLE AREA:
46:     *
47:     *   ORG $5000
48:     *   RMB 512
49:     *
50:     * PROGRAM AREA:
51:     *
52:     *   ORG $0100
53:     *   LDB DBASE
54:     *   LDR BINDEX
55:     *   JCR SHUF
56:     *   JCR FFT
57:     *   STD
58:     *-----*
59:     * BIT-REVERSAL SHUFFLING ROUTINE

```

```

59:     *-----*
60:     *
61:     * SHUF LDA B R
62:     * DEC B          R-1
63:     * LDA A L1
64:     * FINDN2 ASL A
65:     * DEC B          FIND N/2
66:     * SNE FINDN2
67:     * STA A N2
68:     * SUB A L1      N/2-1
69:     * ASL A        N-2 N-2
70:     * STA A N21
71:     * CLR A
72:     * STA A I      I=0
73:     * STA A J      J=0
74:     * SHUF0 LDA B J
75:     * CMP B I
76:     * BLE SHUF1
77:     * JCR INDX4    SCALE UP (FACTOR 4)
78:     * ADD B BINDEX+1
79:     * ADD A BINDEX
80:     * STA B BINDEX+1
81:     * STA A BINDEX
82:     * LDX BINDEX
83:     * LDA A K      GET DATA
84:     * STA A REAL
85:     * LDA B I
86:     * JCR INDX4
87:     * ADD B BINDEX+1
88:     * ADD A BINDEX
89:     * STA B BINDEX+1
90:     * STA A CINDEX
91:     * LDX CINDEX
92:     * LDA A K
93:     * LDX BINDEX
94:     * STA A K
95:     * LDA A REAL
96:     * LDX CINDEX
97:     * STA A K
98:     * SHUF1 LDA A N2
99:     * STA A K      K=N/2
100:    * LDA A J
101:    * SHUF2 CMP A K
102:    * BLT SHUF3
103:    * SUB A K
104:    * LDR K
105:    * BRR SHUF2
106:    * SHUF3 ADD A K
107:    * STA A J
108:    * LDA A I
109:    * INC A
110:    * STA A I
111:    * CMP A N21
112:    * BLE SHUF0
113:    * RTS
114:    *-----*
115:    * FFT KERNEL:
116:    *-----*
117:    * FFT LDX DBASE INITIALISE DATA INDEX
118:    * STX BINDEX
119:    * LDX LKUP AND LOOKUP INDEX

```



```

340: 0299 32 12      300: 39C A REAL
341: 0298 0E 07      301: LDX CINDX
342: 0297 A7 00      302: BUTTERFLY ADDRESS
343: 0296 E7 01      303: STA A X
344: 0295 9E 19      304: STA B 1+X
345: 0294 06 15      305: GET OLD IMAG DATA
346: 0293 9E 15      306: LDA B 2+1
347: 0292 9E 14      307: SUB B IMAGS+1
348: 0291 9E 02      308: STA A IMAG
349: 0290 E7 02      309: SUB B 3+X
350: 0289 32 02      310: STA B 3+X
351: 0288 79 001C      311: 0
352: 0287 79 001B      312: 0
353: 0286 79 001A      313: 0
354: 0285 E7 02      314: 0
355: 0284 79 001H      315: 0
356: 0283 E7 02      316: 0
357: 0282 79 001H      317: 0
358: 0281 79 001H      318: 0
359: 0280 79 001H      319: 0
360: 0279 79 001H      320: 0
361: 0278 79 001H      321: 0
362: 0277 79 001H      322: 0
363: 0276 79 001H      323: 0
364: 0275 79 001H      324: 0
365: 0274 79 001H      325: 0
366: 0273 79 001H      326: 0
367: 0272 79 001H      327: 0
368: 0271 79 001H      328: 0
369: 0270 79 001H      329: 0
370: 0269 79 001H      330: 0
371: 0268 79 001H      331: 0
372: 0267 79 001H      332: 0
373: 0266 79 001H      333: 0
374: 0265 79 001H      334: 0
375: 0264 79 001H      335: 0
376: 0263 79 001H      336: 0
377: 0262 79 001H      337: 0
378: 0261 79 001H      338: 0
379: 0260 79 001H      339: 0
380: 0259 79 001H      340: 0
381: 0258 79 001H      341: 0
382: 0257 79 001H      342: 0
383: 0256 79 001H      343: 0
384: 0255 79 001H      344: 0
385: 0254 79 001H      345: 0
386: 0253 79 001H      346: 0
387: 0252 79 001H      347: 0
388: 0251 79 001H      348: 0
389: 0250 79 001H      349: 0
390: 0249 79 001H      350: 0
391: 0248 79 001H      351: 0
392: 0247 79 001H      352: 0
393: 0246 79 001H      353: 0
394: 0245 79 001H      354: 0
395: 0244 79 001H      355: 0
396: 0243 79 001H      356: 0
397: 0242 79 001H      357: 0
398: 0241 79 001H      358: 0
399: 0240 79 001H      359: 0
400: 0239 79 001H      360: 0

```

```

399: 0299 32 12      300: 39C A REAL
400: 0298 0E 07      301: LDX CINDX
401: 0297 A7 00      302: BUTTERFLY ADDRESS
402: 0296 E7 01      303: STA A X
403: 0295 9E 19      304: STA B 1+X
404: 0294 06 15      305: GET OLD IMAG DATA
405: 0293 9E 15      306: LDA B 2+1
406: 0292 9E 14      307: SUB B IMAGS+1
407: 0291 9E 02      308: STA A IMAG
408: 0290 E7 02      309: SUB B 3+X
409: 0289 32 02      310: STA B 3+X
410: 0288 79 001C      311: 0
411: 0287 79 001B      312: 0
412: 0286 79 001A      313: 0
413: 0285 E7 02      314: 0
414: 0284 79 001H      315: 0
415: 0283 E7 02      316: 0
416: 0282 79 001H      317: 0
417: 0281 79 001H      318: 0
418: 0280 79 001H      319: 0
419: 0279 79 001H      320: 0
420: 0278 79 001H      321: 0
421: 0277 79 001H      322: 0
422: 0276 79 001H      323: 0
423: 0275 79 001H      324: 0
424: 0274 79 001H      325: 0
425: 0273 79 001H      326: 0
426: 0272 79 001H      327: 0
427: 0271 79 001H      328: 0
428: 0270 79 001H      329: 0
429: 0269 79 001H      330: 0
430: 0268 79 001H      331: 0
431: 0267 79 001H      332: 0
432: 0266 79 001H      333: 0
433: 0265 79 001H      334: 0
434: 0264 79 001H      335: 0
435: 0263 79 001H      336: 0
436: 0262 79 001H      337: 0
437: 0261 79 001H      338: 0
438: 0260 79 001H      339: 0
439: 0259 79 001H      340: 0
440: 0258 79 001H      341: 0
441: 0257 79 001H      342: 0
442: 0256 79 001H      343: 0
443: 0255 79 001H      344: 0
444: 0254 79 001H      345: 0
445: 0253 79 001H      346: 0
446: 0252 79 001H      347: 0
447: 0251 79 001H      348: 0
448: 0250 79 001H      349: 0
449: 0249 79 001H      350: 0
450: 0248 79 001H      351: 0
451: 0247 79 001H      352: 0
452: 0246 79 001H      353: 0
453: 0245 79 001H      354: 0
454: 0244 79 001H      355: 0
455: 0243 79 001H      356: 0
456: 0242 79 001H      357: 0
457: 0241 79 001H      358: 0
458: 0240 79 001H      359: 0
459: 0239 79 001H      360: 0

```

```

340: 0299 32 12      300: 39C A REAL
341: 0298 0E 07      301: LDX CINDX
342: 0297 A7 00      302: BUTTERFLY ADDRESS
343: 0296 E7 01      303: STA A X
344: 0295 9E 19      304: STA B 1+X
345: 0294 06 15      305: GET OLD IMAG DATA
346: 0293 9E 15      306: LDA B 2+1
347: 0292 9E 14      307: SUB B IMAGS+1
348: 0291 9E 02      308: STA A IMAG
349: 0290 E7 02      309: SUB B 3+X
350: 0289 32 02      310: STA B 3+X
351: 0288 79 001C      311: 0
352: 0287 79 001B      312: 0
353: 0286 79 001A      313: 0
354: 0285 E7 02      314: 0
355: 0284 79 001H      315: 0
356: 0283 E7 02      316: 0
357: 0282 79 001H      317: 0
358: 0281 79 001H      318: 0
359: 0280 79 001H      319: 0
360: 0279 79 001H      320: 0
361: 0278 79 001H      321: 0
362: 0277 79 001H      322: 0
363: 0276 79 001H      323: 0
364: 0275 79 001H      324: 0
365: 0274 79 001H      325: 0
366: 0273 79 001H      326: 0
367: 0272 79 001H      327: 0
368: 0271 79 001H      328: 0
369: 0270 79 001H      329: 0
370: 0269 79 001H      330: 0
371: 0268 79 001H      331: 0
372: 0267 79 001H      332: 0
373: 0266 79 001H      333: 0
374: 0265 79 001H      334: 0
375: 0264 79 001H      335: 0
376: 0263 79 001H      336: 0
377: 0262 79 001H      337: 0
378: 0261 79 001H      338: 0
379: 0260 79 001H      339: 0
380: 0259 79 001H      340: 0
381: 0258 79 001H      341: 0
382: 0257 79 001H      342: 0
383: 0256 79 001H      343: 0
384: 0255 79 001H      344: 0
385: 0254 79 001H      345: 0
386: 0253 79 001H      346: 0
387: 0252 79 001H      347: 0
388: 0251 79 001H      348: 0
389: 0250 79 001H      349: 0
390: 0249 79 001H      350: 0
391: 0248 79 001H      351: 0
392: 0247 79 001H      352: 0
393: 0246 79 001H      353: 0
394: 0245 79 001H      354: 0
395: 0244 79 001H      355: 0
396: 0243 79 001H      356: 0
397: 0242 79 001H      357: 0
398: 0241 79 001H      358: 0
399: 0240 79 001H      359: 0
400: 0239 79 001H      360: 0

```

```

399: 0299 32 12      300: 39C A REAL
400: 0298 0E 07      301: LDX CINDX
401: 0297 A7 00      302: BUTTERFLY ADDRESS
402: 0296 E7 01      303: STA A X
403: 0295 9E 19      304: STA B 1+X
404: 0294 06 15      305: GET OLD IMAG DATA
405: 0293 9E 15      306: LDA B 2+1
406: 0292 9E 14      307: SUB B IMAGS+1
407: 0291 9E 02      308: STA A IMAG
408: 0290 E7 02      309: SUB B 3+X
409: 0289 32 02      310: STA B 3+X
410: 0288 79 001C      311: 0
411: 0287 79 001B      312: 0
412: 0286 79 001A      313: 0
413: 0285 E7 02      314: 0
414: 0284 79 001H      315: 0
415: 0283 E7 02      316: 0
416: 0282 79 001H      317: 0
417: 0281 79 001H      318: 0
418: 0280 79 001H      319: 0
419: 0279 79 001H      320: 0
420: 0278 79 001H      321: 0
421: 0277 79 001H      322: 0
422: 0276 79 001H      323: 0
423: 0275 79 001H      324: 0
424: 0274 79 001H      325: 0
425: 0273 79 001H      326: 0
426: 0272 79 001H      327: 0
427: 0271 79 001H      328: 0
428: 0270 79 001H      329: 0
429: 0269 79 001H      330: 0
430: 0268 79 001H      331: 0
431: 0267 79 001H      332: 0
432: 0266 79 001H      333: 0
433: 0265 79 001H      334: 0
434: 0264 79 001H      335: 0
435: 0263 79 001H      336: 0
436: 0262 79 001H      337: 0
437: 0261 79 001H      338: 0
438: 0260 79 001H      339: 0
439: 0259 79 001H      340: 0
440: 0258 79 001H      341: 0
441: 0257 79 001H      342: 0
442: 0256 79 001H      343: 0
443: 0255 79 001H      344: 0
444: 0254 79 001H      345: 0
445: 0253 79 001H      346: 0
446: 0252 79 001H      347: 0
447: 0251 79 001H      348: 0
448: 0250 79 001H      349: 0
449: 0249 79 001H      350: 0
450: 0248 79 001H      351: 0
451: 0247 79 001H      352: 0
452: 0246 79 001H      353: 0
453: 0245 79 001H      354: 0
454: 0244 79 001H      355: 0
455: 0243 79 001H      356: 0
456: 0242 79 001H      357: 0
457: 0241 79 001H      358: 0
458: 0240 79 001H      359: 0
459: 0239 79 001H      360: 0

```

```

0011 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0012 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0013 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0014 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0015 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0016 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0017 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0018 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0019 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0020 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0021 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0022 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0023 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0024 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0025 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0026 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0027 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0028 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0029 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0030 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0031 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0032 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0033 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0034 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0035 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0036 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0037 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0038 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0039 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0040 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

```

PC1: REGISTER TRANSDRM

CLEAR TEST BYTE

REG SHIFT COUNT

listings for chapter 3

(1) The HF Spectrogram

```

1:      NAM  HF SPECTROGRAM
2:      OPT  NOS,LIS,PAG
3:      *****
4:      *
5:      *      HF SPECTROGRAM SOFTWARE
6:      *      *****
7:      *
8:      * THIS PROGRAM IS USED TO EVALUATE AND DISPLAY THE
9:      * TIME-VARYING SPECTRAL PROPERTIES OF AN HF VOICE
10:     * CHANNEL.
11:     *
12:     * X AXIS:  TIME DOMAIN,  Q POINTS (Q <= 256).
13:     * Y AXIS:  FREQ DOMAIN,  M/2 POINTS (M <= 256).
14:     *
15:     * DATA IS SAVED ON DISC FOR SUBSEQUENT ANALYSIS
16:     * IF REQUIRED.
17:     *
18:     * *****
19:     *
20:     * I/O ADDRESSES:
21:     *
2010   22:  XAXIS  EQU  $8010      TIME AXIS PORT
2012   23:  XAXIS  EQU  $8012      FREQ AXIS PORT
2013   24:  YAXIS  EQU  $8013      POWER SPECTRUM O/P.
201A   25:  INDATA EQU  $801A      INPUT DATA
2003   26:  CLOCK  EQU  $8003      REAL-TIME CLOCK
27:  *
28:  * MONITOR ROUTINES:
29:  *
E09A   30:  INHEX  EQU  $E09A      IN 1 HEX CHAR
E073   31:  INCH  EQU  $E073
E08F   32:  OUT2H  EQU  $E08F      OUT 2 HEX CHARS
E1D1   33:  OUTCH  EQU  $E1D1      OUT 1 ASCII CHAR
E0E3   34:  CNTRL  EQU  $E0E3      RESTART ADDRESS
A110   35:  DR5   EQU  $A100
A000 05 00 36:  FDB   CLK      INTERRUPT VECTOR
37:  *
38:  * DISC OPERATING SYSTEM ENTRY POINTS:
39:  *
7786   40:  DFM   EQU  $7786      DISC FILE MANAGEMENT
7293   41:  TYPE  EQU  $7293      TYPE DISC ERROR
7783   42:  CDAM  EQU  $7783      CLOSE ACTIVE FILES
7283   43:  DWARM  EQU  $7283      WARM START
44:  *
45:  * MAIN PROCEDURE VARIABLE STORAGE:
46:  *
0000   47:  DR5   DR5   0
0000   48:  N     RMB   2      NO. OF POINTS
0002   49:  M2    RMB   1
0003   50:  R     RMB   1      LOG2(N)
0004   51:  Q     RMB   2      NO. OF LINES
0005   52:  I     RMB   1
0007 03 53:  PERIOD FDB   3      SAMPLE PERIOD CONST
0000   54:  DELT  FDB   1      DELAY BETWEEN LINES
0000   55:  HOUR  FDB   1      CLOCK VARIABLES
0000   56:  MIN  FDB   1
0000   57:  SECT  FDB   1
0000   58:  TEMP  FDB   2      WRES TEMP STORE
0000   59:  TEMP1 FDB   2

```

```

0010   60:  QSD   RMB   1      DFM FUNCTION CODE
0011   61:  XINC  RMB   1      X INCREMENT
0012   62:  YINC  RMB   1      Y INCREMENT
63:  *
64:  * FFT ROUTINE VARIABLE STORAGE:
65:  *
0013   66:  M     RMB   1      BUTTERFLY INDICES
0014   67:  L     RMB   1
0015   68:  I     RMB   1
0016   69:  K     RMB   1
0017   70:  NH    RMB   1
0018 00 26 71:  CLKUP  FDB  0BASE      COSINE LOOKUP POINTER
001A 01 26 72:  SLKUP  FDB  0BASE      SINE LOOKUP POINTER
001C 02 26 73:  WLKUP  FDB  WBASE      WINDOW LOOKUP POINTER
001E   74:  BINDX  RMB   2      DATA POINTER
0020   75:  CINDX  RMB   2      LOOKUP POINTER
0022   76:  REAL  RMB   2      REAL COEFFICIENT
0024   77:  IMAG  RMB   2      IMAG COEFFICIENT
0026   78:  CBASE  RMB  256
0126   79:  SBASE  RMB  256
0226   80:  WBASE  RMB  512
0426   81:  DBASE  RMB  512
82:  *
83:  * MULTIPLICATION ROUTINE VARIABLES:
84:  *
0626   85:  Y     RMB   2      MULTIPLIER
0623   86:  Z     RMB   2      MULTIPLICAND
062A   87:  P     RMB   4      PRODUCT
062E   88:  TB    RMB   1      TEST BYTE
062F   89:  TEMP  RMB   1
90:  *
91:  * DIVISION & SQRT ROOT VARIABLES:
92:  *
0630   93:  KKDVSR RMB   2      DIVISOR
0632   94:  KKDVND RMB   4      DIVIDEND
0636   95:  KKQUOT RMB   4      QUOTIENT
063A   96:  FLAG  RMB   1
0638   97:  G     RMB   4
98:  *
99:  * DISC FILE CONTROL BLOCK
100:  *
063F   101:  FCB   RMB   2      FUNCTN CODE/ ERR STATUS
0641 01 102:  FCB   RMB   1      DRIVE UNIT NUMBER
0642 53 103:  FCB   RMB  154      FILE NAME
0648   104:  FCB   RMB  154      REMAINDER OF FCBLOCK
105:  *
106:  * *****
107:  * MAIN PROCEDURE BEGINS
108:  * *****
0100   109:  DR5   DR5   $0100
0100 8D 0120 110:  JCR  PIASET      INITIALISE PIASC
0100 8D 013A 111:  JCR  PAPARM      PRINT MESSAGES ON TERMINAL ETC.
0106 8D 0259 112:  JCR  INDC      CALCULATE X & Y INCREMENTS
0109 8D 031A 113:  JCR  OPNFIL      OPEN FILE FOR WRITE
010C 8D 037B 114:  SLOOP JCR  ORIGIN      RETURN BEAM TO ORIGIN
010F 8D 0271 115:  JCR  PLOT      PLOT A FRAME
0112 8D 02D7 116:  JCR  PTIME      PRINT TIME
0115 8D 0301 117:  JCR  PREQ      ANOTHER PLOT ?
0118 26 F2 118:  SNE  SLOOP      YES: LOOP BACK
011A 8D 0321 119:  JCR  CLOSFL      CLOSE FILE

```

```

0110 7E 7233 120:      JMP  ZWARMS      JUMP TO DOS
121:      *****
122:      * INITIALISE I/O PORTS:
123:      *****
0120 0E FF04 124:  PIRSET LDX  E1FF04
0123 FF 8010 125:      STX  XAKIS
0126 FF 8012 126:      STX  YAKIS
0129 FF 8018 127:      STX  ZAKIS
012C 0E 0904 128:      LDX  E10004
012F FF 801A 129:      STX  INDATA
0132 0E 0905 130:      LDX  E10005
0135 FF 800B 131:      STX  CLOCK
0138 0E      132:      CLI
013B 3F      133:      RTS
134:      *****
135:      * PRINT MESSAGES AND GET PARAMETERS FROM TERMINAL
136:      *****
013A 0E 012E 137:  PARAM LDX  EBANNR      PRINT BANNER
013D 8D 023D 138:      JSR  PRINT
0140 0E 0153 139:      LDX  ETREQ      REQUEST FOR TIME
0143 8D 023D 140:      JSR  PRINT
0146 0E 01D9 141:      LDX  EHMS      REQUEST HOURS
0149 8D 023D 142:      JSR  PRINT
014C 8D 023D 143:      JSR  BYTE      GET HOURS
014F 37 03      144:      STA  A HOURS
0151 0E 01E3 145:      LDX  EMMES      REQUEST MINS
0154 8D 023D 146:      JSR  PRINT
0157 8D 023D 147:      JSR  BYTE      GET MINS
015A 37 04      148:      STA  A MINS
015D 0E 01E0 149:      LDX  ESMES      REQUEST SECS
015F 8D 023D 150:      JSR  PRINT
0162 8D 023D 151:      JSR  BYTE      GET SECS
0165 37 03      152:      STA  A SECS
0167 0E 01F5 153:      LDX  ERREQ      GET LOG(2)N
016A 8D 023D 154:      JSR  PRINT
016D 8D 024A 155:      JSR  INHEX
0170 37 03      156:      STA  A R
0172 8D 024A 157:      JSR  EXP      FIND N
0175 37 00      158:      STA  A N
0177 37 01      159:      STA  B N+1
0179 44      160:      LSP  A
017A 56      161:      RDR  B
017B 37 02      162:      STA  B N2      N*2
017D 36 02      163:      LDA  A ESREQ      GET LOG(2)Q
017F 8D 023D 164:      JSR  PRINT
0182 8D 024A 165:      JSR  INHEX
0185 37 05      166:      STA  A S
0187 8D 024A 167:      JSR  EXP
018A 37 04      168:      STA  A Q
018C 37 05      169:      STA  B Q+1
018E 0E 020F 170:      LDX  EIREQ      GET DELAY
0191 8D 023D 171:      JSR  PRINT
0194 8D 023D 172:      JSR  BYTE
0197 37 03      173:      STA  A DLY
0199 8D 023D 174:      JSR  BYTE
019C 37 03      175:      STA  A DLY+1
176:      *****
177:      * MESSAGES:
178:      *****
019E 0D 179:  BANNR  AFB  $D,$A

```

```

01A0 2A      180:      FCC  '*** HF SPECTROGRAM ***'
01B3 04      181:      FCB  4
01B9 0D      182:      TREQ  FCB  $D,$A
01BB 45      183:      FCC  'ENTER TIME OF DAY AS FOLLOWS:'
01D3 04      184:      FCB  4
01D9 0D      185:      HMES  FCB  $D,$A
01DB 43      186:      FCC  'HOURS ?'
01E2 04      187:      FCB  4
01E3 0D      188:      MMES  FCB  $D,$A
01E5 4D      189:      FCC  'MINS ?'
01EB 04      190:      FCB  4
01EC 0D      191:      SMES  FCB  $D,$A
01EE 53      192:      FCC  'SECS ?'
01F4 04      193:      FCB  4
01F5 0D      194:      RREQ  FCB  $D,$A,$A
01F8 4C      195:      FCC  'LOG(2)N ?'
0201 04      196:      FCB  4
0202 0D      197:      DREQ  FCB  $D,$A,$A
0205 4C      198:      FCC  'LOG(2)Q ?'
020E 04      199:      FCB  4
020F 0D      200:      IREQ  FCB  $D,$A,$A
0212 44      201:      FCC  'DELAY UNITS ?'
021F 04      202:      FCB  4
0220 0D      203:      PMES  FCB  $D,$A
0222 41      204:      FCC  'ANOTHER PLOT ?'
0231 04      205:      FCB  4
0232 0D      206:      TMES  FCB  $D,$A,$A
0235 54      207:      FCC  'TIME = '
023C 04      208:      FCB  4
209:      *****
210:      * PRINT A MESSAGE:
211:      *****
023D A6 00      212:      PRINT  LDA  A K
023F 81 04      213:      CMP  A E4      DELIMITER ?
0241 27 06      214:      BEQ  PRINI
0243 8D E1D1      215:      JSR  DUTCH      PRINT IT
0245 08      216:      INX
0247 20 F4      217:      BRA  PRINT
0249 39      218:      PRINI  RTS
219:      *****
220:      * FIND 2*(ACCA)
221:      *****
024A 87 062F      222:      EXP  STA  A TEMP
024D 06 01      223:      LDA  B E1
024F 4F      224:      CLR  A
0250 58      225:      EXP1  ASL  B
0251 49      226:      ROL  A
0252 7A 062F      227:      DEC  TEMP
0255 26 F9      228:      BNE  EXP1
0257 39      229:      RTS
230:      *****
231:      * FIND X AND Y INCREMENTS:
232:      *****
0258 06 01      233:      INCS  LDA  B E1
025A 96 05      234:      LDA  A Q+1
025C 27 04      235:      BEQ  INCD
025E 58      236:      INCI  ASL  B
025F 48      237:      ASL  A
0260 26 FC      238:      BNE  INCI
0262 07 12      239:      INCD  STA  B YINC

```

```

0264 06 01 240: LDA B #1
0265 06 01 241: LDA A N+1
0266 07 04 242: BEO INC4
0267 08 243: INC3 ASL B
0268 09 244: ASL A
0269 0A FC 245: BNE INC3
0270 07 11 246: INC4 STA B XINC
0271 09 247: RTS
.....
248:
249: * PLOT A FRAME
.....
250:
251: PLOT LDX Q SAVE NO. OF LINES
252: STX XTEMP1 IN TEMP STORE
253: LDA A #2
254: STA A QSD
255: LDA A #FFF 'NEW PLOT' TO DISC
256: JSR FLOCTL
257: LDA A HOURS
258: JSR FLOCTL
259: LDA A MINS
260: JSR FLOCTL
261: LDA A S
262: JSR FLOCTL
263: LDA A P
264: JSR FLOCTL
265: PLOT1 LDX LDBASE
266: JSR SAMPLE GET DATA FROM A/D
267: JSR WINDOW MULTIPLY BY TIME WINDOW
268: JSR SHUF BIT-REVERSAL SHUFFLE
269: JSR FFT FAST FOURIER TRANSFORM
270: JSR PSPEC FIND POWER SPECTRUM
271: LDA B N2 GET N/2
272: LDX LDBASE+1 BOTTOM OF TRANSFORMED DATA
273: IPLOT PSW B SAVE POINTS COUNT
274: LDA A X GET DATA BYTE
275: COM A
276: STA A ZAXIS OUTPUT TO SCOPE
277: JSR DELAY WAIT TO SETTLE
278: LDA A YINC
279: ADD A YAXIS INCREMENT Y AXIS
280: STA A YAXIS MOVE BEAM UP
281: INX MOVE TO NEXT DATA
282: INX
283: INX
284: INX
285: PUL B COUNTER OFF STACK
286: DEC B
287: BNE DPLOT GET NEXT POINT
288: JSR SAVED SAVE DATA ON DISC
289: CLR YAXIS MOVE BEAM TO AXIS
290: LDA A XINC AND ACROSS
291: ADD A XAXIS TO RIGHT
292: STA A XAXIS
293: LDX XTEMP1 ANY MORE LINES ?
294: DEK
295: BNE PLOT1
296: RTS
.....
297:
298: * PRINT TIME OF DAY:
.....
299:

```

```

02D7 0E 0232 300: PTIME LDX LTMES
02D8 0D 023D 301: JSR PRINT
02D9 0E 0009 302: LDX LHOURS
02E0 0D E0BF 303: JSR OUT2H
02E1 06 3A 304: LDA A #1
02E2 0D E1D1 305: JSR OUTCH
02E3 0E 000A 306: LDX LMIN3
02E4 0D E0BF 307: JSR OUT2H
02E5 06 3A 308: LDA A #1
02E6 0D E1D1 309: JSR OUTCH
02E7 0E 000B 310: LDX LSECS
02E8 0D E0BF 311: JSR OUT2H
02E9 09 312: RTS
.....
313:
314: * DELAY LOOP:
.....
315:
02FA 06 08 316: DELAY LDA A DLY
02FB 01 317: DLY1 NOP
02FC 0A 318: DEC A
02FD 06 FC 319: BNE DLY1
0300 09 320: RTS
.....
321:
322: * REQUEST NEW PLOT:
.....
323:
0301 0E 0220 324: PRED LDX LPMES
0302 0D 023D 325: JSR PRINT
0303 0D E078 326: JSR INCH
0304 01 59 327: CMP A #1Y
0305 09 328: RTS
.....
329:
330: * INPUT A BYTE:
.....
331:
0306 0D E0AA 332: BYTE JSR INHEX
0307 01 48 333: ACL A
0308 01 48 334: ASL A
0309 01 48 335: ACL A
0310 01 48 336: ASL A
0311 01 16 337: TAB
0312 0D E0AA 338: JSR INHEX
0313 01 1B 339: ABA
0314 09 340: RTS
.....
341:
342: * OPEN DISC FILE FOR 'WRITE':
.....
343:
031A 06 01 344: OPNFIL LDA A #1 'WRITE OPEN' CODE
031B 07 10 345: STA A QSD
031C 0D 6E 346: BSR FLOCTL
0320 09 347: RTS FILE NOW OPEN
.....
348:
349: * CLOSE DISC FILE FOR 'WRITE':
.....
350:
0321 06 03 351: CLOSFL LDA A #3 'CLOSE' CODE
0322 07 10 352: STA A QSD
0323 0D 67 353: BSR FLOCTL
0327 09 354: RTS
.....
355:
356: * SAVE DATA ON DISC:
.....
357:
0328 06 02 358: SAVED LDA A #2 'WRITE TO DISC' CODE
0329 07 10 359: STA A QSD

```

```

0330 06 02 360: LDA B W2
0331 07 13 361: STA B M
0332 08 0426 362: LDX EDBASE
0333 0F 00 363: SAVE1 STX KTEMP
0334 06 00 364: LDA A X
0335 0D 039E 365: JSR FLOTRL
0336 0E 00 366: LDX KTEMP
0337 03 367: INX
0338 03 368: INX
0339 03 369: INX
0340 03 370: INX
0341 7A 0013 371: DEC M
0342 26 EE 372: BNE SAVE1
0343 59 373: RTS
374:
375: * COLLECT N SAMPLES FROM A-D
376:
377: SAMPLE LDA A N
378: LDA B N+1
379: SMPL1 PSH A SAVE COUNT
380: PSH B
381: LDA A E834 ZERO A/D
382: STA A INDATA+1
383: LDA A E83C 'CONVERT' COMMAND
384: STA A INDATA+1
385: LDA B PERIOD
386: SMPL2 DEC B WAIT LOOP
387: BNE SMPL2
388: CLR B
389: LDA A R
390: STA A TEMP
391: LDA A INDATA GET DATA
392: EOR A E830 2'S COMP
393: SMPL4 ROR A
394: ROR B SCALE DATA
395: DEC TEMP
396: BNE SMPL4
397: STA A X SAVE DATA
398: STA B 1*X
399: INX
400: INX
401: PUL A
402: PUL B
403: JSR DPDEC
404: BEQ SMPL1
405: RTS
406:
407: * SET X,Y, AND Z AXES TO ZERO
408:
409: ORIGIN CLR A
410: STA A XANIS
411: STA A YANIS
412: STA A ZANIS
413: RTS
414:
415: * DECREMENT A DOUBLE PRECISION NUMBER
416:
417: DPDEC SUB B I1
418: DEC A E0
419: BNE DPDI

```

```

0380 5D 420: TST B
0381 39 421: DPDI PTS
422:
423: * DISC FILE MANAGEMENT SUBROUTINE
424:
425: FLOTRL LDX EFCB CONTROL BLOCK ADDRESS
0391 06 10 426: LDA B 050 DFM FUNCTION CODE
0392 E7 00 427: STA B X
0393 0D 7796 428: JSR DFM CALL DFM
0394 27 09 429: BEQ OK NO DISC ERRORS ?
039A 0D 72A9 430: JSR ZTYPDE TYPE ERROR
039D 0D 7793 431: JSR CDFM CLOSE ACTIVE FILES
039E 7E 7283 432: JMP ZWARMS DOS
03A3 39 433: OK RTS
434:
435: * MULTIPLY INPUT DATA BY TIME WINDOW
436:
437: WINDOW LDA B E1
03A5 06 03 438: LDA A R
03A8 01 08 439: JMWLN CMP A E3
03AA 27 04 440: BEQ WMUL
03AC 58 441: RCL B
03AD 4C 442: INC A
03AE 20 F8 443: SRA JMWLN
03B0 58 444: WMUL RCL B FIND 2*256/2^R
03B1 07 13 445: STA B M
03B3 06 00 446: LDA A N
03B5 06 01 447: LDA B N+1
03B7 0E 0426 448: LDX EDBASE
03BA DF 1E 449: WLOOP STX BINDX DATA POINTER
03BC 37 450: PSH B SAVE COUNT
03BD 36 451: PSH A
03BE DF 1C 452: STX WLKUP
03C0 EE 00 453: LDX X GET LOOKUP
03C2 FF 0626 454: STX Y
03C5 0E 1E 455: LDX BINDX
03C7 EE 00 456: LDX X GET DATA
03C9 FF 0628 457: STX Z
03CC 0D 04F4 458: JSR MULT16 MULTIPLY TOGETHER
03CF 06 062B 459: LDA A P+1
03D2 F6 062C 460: LDA B P+2 RESULT
03D5 0E 1E 461: LDX BINDX
03D7 A7 00 462: STA A X RESTORE DATA
03D9 E7 01 463: STA B 1*X
03DB 08 464: INX
03DC 08 465: INX MOVE TO NEXT POINT
03DD 08 466: INX
03DE 08 467: INX
03DF DF 1E 468: STX BINDX
03E1 06 13 469: LDA B M
03E3 4F 470: CLR A
03E4 08 1D 471: ADD B WLKUP+1 FIND ADDRESS OF
03E6 09 1C 472: ADD A WLKUP NEXT LOOKUP
03E8 07 1D 473: STA B WLKUP+1
03EA 07 1C 474: STA A WLKUP
03EC 0E 1C 475: LDX WLKUP
03EE 32 476: PUL A
03EF 33 477: PUL B GET COUNTER BACK
03F0 0D 0386 478: JSR DPDEC
03F3 25 07 479: SCS WLOOP

```

```

03F5 39      480:      RTS
03F5 39      481:      *****
03F5 39      482:      * SHUFFLE INPUT DATA IN BIT-REVERSED ORDER:
03F5 39      483:      *****
03F6 36 02   484: SHUF  LDA A N2      GET N/2
03F6 36 01   485:      SUB A E1      N/2-1
03F6 36 01   486:      ROL A      N-2
03F6 36 01   487:      SUB A E1      N-3
03F6 36 14   488:      STA A L      L=N-3
03F6 36 01   489:      CLR A      I=0
0400 5F      490:      CLR B      J=0
0401 11      491: ILOOP CBA      (I-J)<0 ?
0402 30 33   492:      SGE  NOSWAP  NO: DON'T SWAP
0404 37      493: SWAP  PSH B      SAVE J
0405 36      494:      PSH A      AND I
0406 4F      495:      CLR A
0407 50 0450 496:      JSR  INDX2    GO FIND ADDRESSES
0408 50 21   497:      STA B CINDX+1
0408 50 20   498:      STA A CINDX  ADDRESS OF X(J)
0408 50 32   499:      PUL A      GET I
0408 50 36   500:      PSH A      KEEP IT
0410 16      501:      TAB
0411 4F      502:      CLR A
0412 50 0450 503:      JSR  INDX2    FIND ADDRESS OF X(I)
0415 57 1F   504:      STA B BINDX+1
0417 57 1E   505:      STA A BINDX
0418 5E 1E   506:      LDX  BINDX
0418 5E 00   507:      LDX  0*X      GET X(I)
0419 5F 0C   508:      STX  XTEMP    XTEMP=X(I)
041F 5E 20   509:      LDX  CINDX
0421 56 00   510:      LDA A X
0423 56 01   511:      LDA B 1*X
0425 56 1E   512:      LDX  BINDX
0427 57 00   513:      STA A X      X(I)=X(J)
0429 57 01   514:      STA B 1*X
0429 56 0C   515:      LDA A XTEMP
0429 56 0D   516:      LDA B XTEMP+1
042F 56 20   517:      LDX  CINDX
0431 57 00   518:      STA A X      X(J)=XTEMP
0433 57 01   519:      STA B 1*X
0435 5E 32   520:      PUL A
0435 5E 33   521:      PUL B
0437 56 36   522: NOSWAP PSH A
0439 56 02   523:      LDA A N2
0439 56 16   524:      STA A K      K=NH
0439 56 32   525:      PUL A
0440 01 16   526: TSTJ  CMP B K      J >= K ?
0440 5D 07   527:      BLT  NOCHNG
0441 5D 15   528: CHNG  SUB B K      J=J-K
0443 77 0015 529:      ABR  K      K=K/2
0445 5D 25   530:      SRA  TSTJ
0445 5D 16   531: NOCHNG ADD B K      J=J+K
0449 40      532:      INC A      I=I+1
0449 51 14   533:      CMP A L      I > N-2 ?
0449 5F 22   534:      BLE  ILOOP
044F 33      535:      RTS
0450 58      536:      *
0450 58      537: INDX2 ROL B
0451 49      538:      ROL A
0452 58      539:      ROL B

```

```

0453 49      540:      ROL A
0454 0E 0426 541:      LDX  LDBASE
0457 0F 0C   542:      STX  XTEMP
0459 08 0D   543:      ADD B XTEMP+1
045B 99 0C   544:      ADC A XTEMP
045D 39      545:      RTS
0456 54:      *****
0457: * EVALUATE SQUARE ROOT OF NUMBER IN ACCA,ACCB.
0458: * THIS SUBROUTINE EVALUATES THE NEWTON-RAPHSON
0459: * RECURSION FORMULA:
0459: * X0 = X0 - ((X0^2-X)/2*X0)
0459: * WHERE X0 IS A BETTER APPROXIMATION TO THE
0459: * SQUARE ROOT OF X
0459: *****
045E 7F 063A 554: SORT  CLR  FLAG    CLEAR PRECISION INDICATOR
0461 87 063D 555:      STA A G+2
0464 F7 063E 556:      STA B G+3
0467 7F 0638 557:      CLR  G
046A 7F 063C 558:      CLR  G+1
046D 87 0626 559: SORT1 STA A Y      PUT IN MULTIPLIER
0470 87 0628 560:      STA A Z      AND MULTIPLICAND
0473 F7 0627 561:      STA B Y+1
0476 F7 0629 562:      STA B Z+1
0479 5D 04F4 563:      JSR  MULTI16
047C B6 062C 564:      LDA A P+2
047F F6 062D 565:      LDA B P+3
0482 F0 063E 566:      SUB B G+3
0485 B2 063D 567:      SBC A G+2
0488 F7 0635 568:      STA B KKDVND+3 RESULT IN DIVIDEND
048B 87 0634 569:      STA A KKDVND+2
048E F6 0628 570:      LDA B P+1
0491 B6 062A 571:      LDA A P
0494 F2 063C 572:      SBC B G+1
0497 B3 0638 573:      SBC A G
049A F7 0633 574:      STA B KKDVND+1
049D 87 0632 575:      STA A KKDVND
04A0 F6 0629 576:      LDA B Z+1
04A3 B6 0628 577:      LDA A Z
04A6 58      578:      ROL B      FIND 2*X0
04A7 49      579:      ROL A
04A8 87 0630 580:      STA A KKDVDR  PUT IN DIVISOR
04AB F7 0631 581:      STA B KKDVDR+1
04AE 8D 0553 582:      JSR  DIVIDE  FIND ((X0*X0-X)/2*X0)
04B1 B6 0628 583:      LDA A Z
04B4 F6 0629 584:      LDA B Z+1
04B7 F0 0639 585:      SUB B KKQUOT+3
04BA B2 0638 586:      SBC A KKQUOT+2 X0-(X0*X0-X)/2*X0
04BD 81 0628 587:      CMP A Z      MSBYTE SAME ?
04C0 26 AB   588:      BNE  SORT1  NO: ITERATE AGAIN
04C2 36      589:      PSH A      SAVE MSBYTE
04C3 B6 0629 590:      LDA A Z+1
04C6 19      591:      SBA
04C7 81 01   592:      CMP A E1      X0(OLD)-X0(NEW)
04C9 27 08   593:      BEQ  SORT2  IF DIFFERENT BY 1
04CB 81 00   594:      CMP A E0      OR EQUAL THEN TEST
04CD 27 04   595:      BEQ  SORT2  FOR PRECISION LEVEL
04CF 32      596:      PUL A
04D0 7E 046D 597:      JMP  SORT1
04D3 32      598: SORT2 PUL A      RESTORE MSBYTE
04D4 7D 063A 599:      TST  FLAG

```

```

0407 36 1P 600: RNE DONE
0408 70 0639 601: INC FLAG
0409 85 063E 602: LDA A G+3
040F 87 0631 603: STA A G+1
040E 86 063D 604: LDA A G+2
0405 87 063B 605: STA A G
0403 7F 063D 606: CLR G+2
040B 7F 063E 607: CLR G+3
040E 17 603: TBA
040F 5F 609: CLR B
0400 7E 045D 610: JMP SORT1
0403 39 611: DONE RTS
612: *****
613: * SIGNED MULTIPLY ROUTINE:
614: * MULTIPLIES 2 16-BIT 2'S COMPLEMENT NUMBERS
615: * USING THE BOOTH ALGORITHM.
616: *
617: * MULTIPLIER: Y,Y+1
618: * MULTIPLICAND: Z,Z+1
619: * PRODUCT: P,P+1,P+2,P+3
620: *
621: * THE MULTIPLIER IS DESTROYED
622: * THE MULTIPLICAND IS UNCHANGED
623: *
624: *****
04F4 0E 0000 625: MULT16 LDN 10
04F7 FF 0629 626: STN P CLEAR REGISTERS
04FA FF 062C 627: STN P+2
04FD 7F 063E 628: CLR TB
0500 0E 0010 629: LDN 116
0508 56 0627 630: LOOP LDA A Y+1
0509 84 01 631: AND A 1
050B 16 632: TBA
0509 83 043E 633: EOR A TB Y(LSB)=Y(LSB-1) ?
0500 87 89 634: SED SHIFT YES? SHIFT
0508 5D 635: TST B
050F 87 14 636: SED ADD Y(LSB)=0: ADD.
0511 86 0634 637: SUBTR LDA A P
0514 86 0639 638: LDA A P+1
0517 80 0629 639: SUB B Z+1
0519 8D 0439 640: SBC A Z
051D 87 0634 641: STA A P
051D 87 0628 642: STA B P+1
0513 80 12 643: BRA SHIFT
0510 86 0634 644: ADD LDA A P
0509 86 0628 645: LDA B P+1
0508 85 0639 646: ADD B Z+1
0508 89 0639 647: ADC A Z
0501 87 0434 648: STA A P
0504 87 0628 649: STA B P+1
0507 7F 0628 650: SHIFT CLR TB
050A 76 0626 651: ROR Y
050D 76 0627 652: ROR Y+1
0540 79 062E 653: ROL TB
0541 77 0634 654: ROR P
0546 76 0626 655: ROR P+1
0549 76 0630 656: ROR P+2
0541 76 063D 657: ROR P+3
054F 09 658: JEX
0551 86 81 659: BNE LOOP

```

```

0552 39 660: RTS
661: *****
662: * DIVISION ROUTINE.
663: * DIVIDES A 32-BIT 2'S COMPLEMENT NUMBER BY A
664: * 16-BIT 2'S COMPLEMENT NUMBER TO GIVE A 16-BIT
665: * RESULT.
666: *
667: * DIVIDEND: XKDOND,XKDOND+1,XKDOND+2,XKDOND+3
668: * DIVISOR: XKDVR,XKDVR+1
669: * QUOTIENT: XKQOQT,XKQOQT+1,XKQOQT+2,XKQOQT+3
670: *
671: * FROM "MOTOROLA APPLICATIONS MANUAL", PP. 2.23-24.
672: *
673: *****
0553 674: DIVIDE EQU +
0553 06 10 675: XKDIVD LDA B 116 S=16
0555 7F 0636 676: CLR XKQOQT
0558 7F 0637 677: CLR XKQOQT+1
055B 7F 0638 678: CLR XKQOQT+2 CLEAR QUOTIENT BUFFER
055E 7F 0639 679: CLR XKQOQT+3
0561 5C 680: DVDLPO INC B S=S+1
0562 01 20 681: CMP B 132
0564 2E 61 682: BGT DVDERR IF S>32 ERROR
0566 78 0631 683: RSL XKDVR+1 LEFT SHIFT DIVISOR
0569 79 0630 684: ROL XKDVR
056C 24 F3 685: BCC DVDLPO IF C=0 DON'T LOOP
056E 76 0630 686: ROR XKDVR SHIFT DIVISOR BACK 1
0571 76 0631 687: ROR XKDVR+1
0574 37 688: PSH B SHIFT COUNT ON STACK
0575 86 0632 689: LDA A XKDOND
0578 81 0630 690: DVDLP1 CMP A XKDVR IF DIVIDEND<DIVISOR
057B 25 21 691: SBC DVNSUB DON'T SUBTRACT
057D 0D 692: DVDLP2 SEC OF DIVIDEND> OR =
057E 79 0639 693: ROL XKQOQT+3 DIVISOR THEN
0581 79 0638 694: ROL XKQOQT+2 SHIFT 0 LEFT
0584 79 0637 695: ROL XKQOQT+1 1 BIT.
0587 79 0636 696: ROL XKQOQT
058A 86 0632 697: LDA A XKDOND
058D 86 0633 698: LDA B XKDOND+1 Y(M)=Y(M)-X
0590 F0 0631 699: SUB B XKDVR+1
0593 82 0630 700: SBC A XKDVR
0596 87 0633 701: STA B XKDOND+1
0599 87 0632 702: STA A XKDOND
059C 20 0D 703: BRA DVSHFT
059E 0C 704: DVNSUB CLC SHIFT 0 LEFT
059F 79 0639 705: ROL XKQOQT+3
05A2 79 0638 706: ROL XKQOQT+2
05A5 79 0637 707: ROL XKQOQT+1
05A8 79 0636 708: ROL XKQOQT
05AB 33 709: DVSHFT PUL B
05AC 5A 710: DEC B
05AD 37 711: PSH B
05AE 27 1A 712: SED DIVDEND
05B0 0C 713: CLC IF S>0 SHIFT DIVIDEND
05B1 79 0625 714: ROL XKDOND+3
05B4 79 0634 715: ROL XKDOND+2
05B7 79 0633 716: ROL XKDOND+1
05BA 79 0632 717: ROL XKDOND
05BD 86 0632 718: LDA A XKDOND
05C0 86 0633 719: LDA B XKDOND+1

```

```

0503 25 B3 720: BCS DVDLP2 IF C=1 GOTO LOOP2
0505 20 B1 721: BRA DVDLP1
0507 7E E0E3 722: DVDERR JMP CNTRL
050A 33 723: DVDEND PUL B
050B 39 724: RTS
725: *****
726: * INTERRUPT ROUTINE
727: * UPDATES REAL TIME CLOCK
728: *****
0500 96 08 729: CLK LDA A SECS
0502 88 01 730: ADD A #1
0504 19 731: DAA
0501 97 08 732: STA A SECS
0503 81 60 733: CMP A #60
0505 26 10 734: BNE CK1
0507 7F 000B 735: CLR SECS
0509 96 09 736: LDA A MINS
050C 88 01 737: ADD A #1
050E 19 738: DAA
050F 97 09 739: STA A MINS
0511 81 60 740: CMP A #60
0513 26 0E 741: BNE CK1
0515 26 09 742: LDA A HOURS
0517 88 01 743: ADD A #1
0519 19 744: DAA
051A 97 09 745: STA A HOURS
051C 81 24 746: CMP A #24
051E 26 03 747: BNE CK1
0520 7F 0009 748: CLR HOURS
0523 16 800B 749: CK1 LDA A CLOCK
0526 38 750: RTS
751: *****
752: * COMPUTE POWER SPECTRUM
753: * P=SOR(RE+RE+IM+IM)
754: *****
05F7 CE 0426 755: PSPEC LDX LDSBASE
05FA DF 00 756: STX XTEMP
05FC 26 02 757: LDA B #2
05FE 37 758: POWER1 PSH B SAVE COUNT
05FF DE 00 759: LDX XTEMP
0601 EE 00 760: LDX 0+X GET REAL DATA
0603 FF 0626 761: STX Y
0606 FF 062B 762: STX Z
0609 BD 04F4 763: JER MULT16 FIND RE+RE
060C EE 02 764: LDX 2+X
060E FF 0626 765: STX Y
0611 FF 062B 766: STX Z
0614 88 062A 767: LDA A P
0617 F6 062B 768: LDA B P+1
061A 37 769: PSH B
061B 36 770: PSH A
061C BD 04F4 771: JER MULT16
061F 38 772: PUL A
0620 39 773: PUL B
0621 F8 062B 774: ADD B P+1 R+R+I+I
0624 89 062A 775: ADD A P
0627 01 00 776: CMP B #0
0629 26 06 777: BNE SORROOT
062B 81 00 778: CMP A #0
062D 26 02 779: BNE SORROOT

```

```

062F 20 03 780: BRA HOP
0631 BD 945E 781: SORROOT JSR SORT
0634 DE 0C 782: HOP LDX XTEMP
0636 A7 00 783: STA A X
0638 E7 01 784: STA B 1+X
063A 96 0C 785: HXLOC LDA A XTEMP
063C D6 00 786: LDA B XTEMP+1
063E CB 04 787: ADD B #4
0640 89 00 788: ADC A #0
0642 97 0C 789: STA A XTEMP
0644 D7 0D 790: STA B XTEMP+1 NEW POINTER
0646 33 791: PUL B
0647 5A 792: DEC B
0648 26 B4 793: BNE POWER1
064A 39 794: RTS
795: *
796: * FFT ROUTINES:
797: *
064B 798: FFT EQU *
799: END
NO ERROR(S) DETECTED

```

DOS:

listings for chapter 4

- (1) Memory diagnostic
- (2) Vector diagnostic
- (3) Parallel processing diagnostic

```

NAM      CDAT1
*MEM DIAGNOSTIC (JOHN CHRISTENSEN'S)
OPT      0
EOE3     CONTRL EQU $EOE3
A002     ORG $A002
A002 0002 LOTEMP RMB 2      STARTING ADDRESS
A004 0002 HITEMP RMB 2     ENDING ADDRESS

A014     ORG $A014
A014 00   INIPAT FCB 0      INITIAL TEST PATTERN
A015 FF   TESPAT FCB $FF    TEST PATTERN
A016 0002 IXRTMP RMB 2      IXR TEMPORARY STORAGE
A018 FE A002 START LDX LOTEMP
A01B B6 A014 LDA A INIPAT
A01E A7 00 LOOP1 STA A 0,X
A020 A1 00 CMP A 0,X
A022 26 53 BNE ERPNT1
A024 BC A004 CPX HITEMP
A027 27 03 BEQ LOAFAT
A029 03 INX
A02A 20 F2 BRA LOOP1
A02C FE A002 LOAFAT LDX LOTEMP
A02F F6 A015 LDA B TESPAT
A032 E7 00 LOOP4 STA B 0,X
A034 20 14 BRA CHECK

A048     ORG $A048
A048 A018 FDB $A018
A04A E1 00 CHECK CMP B 0,X
A04C 26 2A BNE ERPNT2
A04E FF A016 CHKLOW STX IXRTMP
A051 BC A002 LOOP2 CPX LOTEMP
A054 27 07 BEQ CHCKHI
A056 09 DEX
A057 A1 00 CMP A 0,X
A059 26 1E BNE ERPNT3
A05B 20 F4 BRA LOOP2
A05D FE A016 CHCKHI LDX IXRTMP
A060 BC A004 CPX HITEMP
A063 27 16 BEQ END
A065 08 LOOP3 INX
A066 A1 00 CMP A 0,X
A068 26 10 BNE ERPNT4
A06A DC A004 CPX HITEMP
A06D 26 F6 BNE LOOP3
A06F FE A016 RESTRE LDX IXRTMP
A072 A7 00 STA A 0,X
A074 08 INX
A075 20 BB BRA LOOP4
A077 3F ERPNT1 SWI
A078 3F ERPNT2 SWI
A079 3F ERPNT3 SWI
A07A 3F ERPNT4 SWI
A07B 7E EOE3 END JMP CONTRL
END

```

```

ERROR ON INITIAL PATTERN
ERROR ON TEST PATTERN
DUAL ADDRESS ERROR LOW
DUAL ADDRESS ERROR HI

```

```

1:      NAM  SLAVE VECTOR TEST ROUTINE
2:      OPT  NOS,LIS,PAG
3:      *
4:      *      SLAVE PROCESSOR DIAGNOSTIC 2
5:      *      =====
6:      *
7:      * ROUTINE TO TEST SLAVE INTERRUPT VECTORS. AN INTERRUPT
8:      * SEQUENCE IS GENERATED BY THE MASTER, RESULTING IN THE
9:      * LOADING OF A TEST BYTE WITH $00, $DD, $EE, OR $FF,
10:     * DEPENDING ON THE TYPE OF INTERRUPT GENERATED.
11:     *
12:     *
13:     *
13:      TSTBYT EQU  $0001      TEST BYTE
14:      LATCH EQU  $C000      BASE OF SLAVE RAM
15:     * VECTOR AREA:
16:     *
16:      ORG  $C0F8
17:      IR0  FDB  IR0TST-LATCH
18:      SWI  FDB  SWITST-LATCH
19:      NMI  FDB  NMITST-LATCH
20:      RES  FDB  RESTST-LATCH
21:     *
22:     *
22:      ORG  $C100
23:      IR0TST LDA A $100
24:      LOOP  STA A TSTBYT
25:      BRA  LOOP
26:      SWITST LDA A $1DD
27:      BRA  LOOP
28:      NMITST LDA A $1EE
29:      BRA  LOOP
30:      RESTST LDA A $1FF
31:      BRA  LOOP
32:      END

```

NO ERROR(S) DETECTED

```

1:      NAM      PARALLEL PROCESSING TEST ROUTINE
2:      OPT      NOS,LIS,PAG
3:      *
4:      *
5:      *      SLAVE PROCESSOR DIAGNOSTIC 3
6:      *      =====
7:      *
8:      * ROUTINE TO DEMONSTRATE THE PARALLEL PROCESSING
9:      * CAPABILITY OF THE MASTER-SLAVE SYSTEM. THE
10:     * EXPRESSION (A1 X B1) + (A1 X D1) IS EVALUATED
11:     * BY EXECUTING THE MULTIPLICATIONS IN PARALLEL.
12:     * ALL OPERANDS ARE 2-BYTE 2'S COMPLEMENT NUMBERS.
13:     * RESULT IN "P".
14:     *
15:     *
16:     *
17:     * MASTER VARIABLE AREA:
18:     *
19:     *      ORG      0
20:     *      A1      RMB      2      OPERANDS
21:     *      B1      RMB      2
22:     *      C1      RMB      2
23:     *      D1      RMB      2
24:     *      F      RMB      4      RESULT
25:     *      Y1      RMB      2      MULTIPLIER
26:     *      Z1      RMB      2      MULTIPLICAND
27:     *      P1      RMB      4      PRODUCT
28:     *      TB1     RMB      1      TEST BYTE
29:     *
30:     * SLAVE VARIABLE AREA:
31:     *
32:     *      LATCH   EQU      $0000
33:     *      ORG      $0001
34:     *      TSTBYT  RMB      1
35:     *      TB2     RMB      1      TEST BYTES
36:     *      Y2      RMB      2      MULTIPLIER
37:     *      Z2      RMB      2      MULTIPLICAND
38:     *      P2      RMB      4      PRODUCT
39:     *
40:     * MASTER PROCESSOR ROUTINES:
41:     *
42:     *      ORG      $0100
43:     *      BEGIN   LDA      A,$FF
44:     *              STA      A,LATCH      HOLD SLAVE IN RESET
45:     *              LDX      A,1         GET OPERANDS
46:     *              STX      Y2+LATCH    INTO SLAVE RAM
47:     *              LDX      B,1
48:     *              STX      Z2+LATCH
49:     *              LDA      A,$FF
50:     *              STA      A,LATCH      START SLAVE.
51:     *              LDX      C,1         GET OPERANDS
52:     *              STX      Y1         INTO RAM
53:     *              LDX      D,1
54:     *              STX      Z1
55:     *              JSR      MULTI      FIND (C X D)
56:     *      LOOP    TST      TSTBYT     SLAVE FINISHED ?
57:     *              BNE      LOOP      IF NOT: WAIT.
58:     *              LDA      B,P1+2    FIND SUM OF PRODUCTS
59:     *              LDA      A,P1+2

```

```

60:      ADD      B,P2+LATCH+3
61:      ADC      A,P2+LATCH+2
62:      STA      A,P+2      SAVE PARTIAL RESULT
63:      STA      B,P+3
64:      LDA      B,P1+1
65:      LDA      A,P1
66:      ADC      B,P2+LATCH+1
67:      ADC      A,P2+LATCH
68:      STA      A,P      SAVE FULL RESULT
69:      STA      B,P+1
70:      SMI
71:      *
72:      * 2'S COMPLEMENT MULTIPLY ROUTINE:
73:      *
74:      MULTI1  LDX      $0          CLEAR REGISTERS
75:      STX      P1
76:      STX      P1+2
77:      CLR      TB1
78:      LDX      $16
79:      LOOP1   LDA      A,Y1+1
80:      AND      A,$1
81:      TAB
82:      EOP      A,TB1
83:      BEQ      $SHIFT1
84:      TST      B
85:      BEQ      $ADD1
86:      SUB1    LDA      A,P1
87:      LDA      B,P1+1
88:      SUB      B,Z1+1
89:      SBC      A,Z1
90:      STA      A,P1
91:      STA      B,P1+1
92:      BRA      $SHIFT1
93:      ADD1    LDA      A,P1
94:      LDA      B,P1+1
95:      ADD      B,Z1+1
96:      ADC      A,Z1
97:      STA      A,P1
98:      STA      B,P1+1
99:      SHIFT1  CLR      TB1
100:     ROR      Y1
101:     ROR      Y1+1
102:     ROL      TB1
103:     ROR      P1
104:     ROR      P1+1
105:     ROR      P1+2
106:     ROR      P1+3
107:     DEK
108:     BNE      $LOOP1
109:     ASL      P1+2
110:     RCL      P1+1
111:     ROL      P1
112:     RTS
113:     *
114:     * SLAVE PROCESSOR ROUTINES
115:     *
116:     *      ORG      $0100
117:     *      SLAVE   LDS      $03E0     SET UP STACK
118:     *              LDA      A,$FF     SET FLAG
119:     *              STA      A,TSTBYT

```

```

0107 8D 05 120: BSR MULT2
0108 7F 0001 121: WAIT CLR TSTBYT
0109 20 FB 122: BRA WAIT
123: *
124: * 2'S COMPLEMENT MULTIPLY ROUTINE:
125: *
010E CE 0000 126: MULT2 LDX L0
0111 DF 07 127: STX P2 CLEAR REGISTERS
0112 DF 03 128: STX P2+2
0115 7F 0002 129: CLR T82
0118 CE 0010 130: LDX L16 SHIFT COUNTER
011B 95 04 131: LOOP2 LDA A Y2+1
011D 84 01 132: AND A L1
011F 15 133: TAB
0120 93 02 134: EOR A T82
0122 27 1D 135: BEQ SHIFT2
0124 50 136: TST B
0125 27 0E 137: BEQ ADD2
0127 95 07 138: SUB2 LDA A P2
0129 95 03 139: LDA B P2+1
012B 90 06 140: SUB B Z2+1
012D 92 05 141: SEC A Z2
012F 97 07 142: STA A P2
0131 97 03 143: STA B P2+1
0133 20 0C 144: BRA SHIFT2
0135 95 07 145: ADD2 LDA A P2
0137 95 03 146: LDA B P2+1
0139 98 06 147: ADD B Z2+1
013B 99 05 148: ADC A Z2
013D 97 07 149: STA A P2
013F 97 03 150: STA B P2+1
0141 7F 0002 151: SHIFT2 CLR T82
0144 76 0003 152: ROR Y2
0147 75 0004 153: ROR Y2+1
014A 79 0002 154: ROL T82
014D 77 0007 155: ASR P2
0150 76 0008 156: ROR P2+1
0153 76 0009 157: ROP P2+2
0156 76 000A 158: ROR P2+3
0159 09 159: DEX
015A 28 BF 160: BNE LOOP2
015C 78 0009 161: RIL P2+2
015F 79 0003 162: ROL P2+1
0162 79 0007 163: ROL P2
0165 39 164: RTS
165: *
166: * SLAVE RESET VECTOR
167: *
03F8 168: ORG 103F8
03F9 01 00 169: FDB SLAVE-LATCH
170: END

```

NO ERROR(S) DETECTED

listings for chapter 5

(1) BCH Encoder

(2) BCH Decoder

```

1: 1: NAM (15.7) BCH ENCODER
2: 2: OPT M3S.LIS
3: 3: *****
4: 4: (15.7) BCH ENCODING SUBROUTINE
5: 5: *****
6: 6: *****
7: 7:
8: 8: * SUBROUTINE TO MAP 9 7-BIT INFORMATION BLOCK INTO
9: 9: * 4 (15.7) CODEWORD.
10: 10: * DATA BLOCK IS DIVIDED BY THE GENERATOR POLYNOMIAL
11: 11: * TO FIND CHECK BITS. DATA IN PCCR, CHECK BITS
12: 12: * RETURNED IN POCB.
13: 13: *
14: 14: *
15: 15: *****
16: 16: * DATA STORAGE AREA:
17: 17: *
18: 18: *
19: 19: 005 0
20: 20: GENPOL PGR 110011000 M1CK)
21: 21: GCB 111111000 M3CK)
22: 22: SHECNT 9M8 1
23: 23: 00100
24: 24: CODE 924 9
25: 25: COME1 L04 3 L3
26: 26: STA 3 SHECNT
27: 27: CLR 3
28: 28: EXOROP EOR 8 GENPOL+1
29: 29: EOR 9 GENPOL
30: 30: BITTST BIT 9 L880
31: 31: 3M5 EXOROP
32: 32: 001 3
33: 33: 90L 9
34: 34: DEC SHECNT
35: 35: 3M5 BITTST
36: 36: 3M5 TAB
37: 37: PUL 9
38: 38: 0T3
39: 39: 0T3
40: 40: 0T3
41: 41: 0T3
42: 42: 0T3
43: 43: 0T3
44: 44: 0T3
45: 45: 0T3
46: 46: 0T3
47: 47: 0T3
48: 48: 0T3
49: 49: 0T3
50: 50: 0T3
51: 51: 0T3
52: 52: 0T3
53: 53: 0T3
54: 54: 0T3
55: 55: 0T3
56: 56: 0T3
57: 57: 0T3
58: 58: 0T3
59: 59: 0T3
60: 60: 0T3
61: 61: 0T3
62: 62: 0T3
63: 63: 0T3
64: 64: 0T3
65: 65: 0T3
66: 66: 0T3
67: 67: 0T3
68: 68: 0T3
69: 69: 0T3
70: 70: 0T3
71: 71: 0T3
72: 72: 0T3
73: 73: 0T3
74: 74: 0T3
75: 75: 0T3
76: 76: 0T3
77: 77: 0T3
78: 78: 0T3
79: 79: 0T3
80: 80: 0T3
81: 81: 0T3
82: 82: 0T3
83: 83: 0T3
84: 84: 0T3
85: 85: 0T3
86: 86: 0T3
87: 87: 0T3
88: 88: 0T3
89: 89: 0T3
90: 90: 0T3
91: 91: 0T3
92: 92: 0T3
93: 93: 0T3
94: 94: 0T3
95: 95: 0T3
96: 96: 0T3
97: 97: 0T3
98: 98: 0T3
99: 99: 0T3
100: 100: 0T3

```

```

1:      NAM      (15.7) BCH DECODER
2:      OPT      NOS,LIS
3:      *
4:      *
5:      *      (15.7) BCH DECODING SUBROUTINE
6:      *      *
7:      *
8:      * SUBROUTINE TO CORRECT UP TO 2 ERRORS IN A
9:      * 15-BIT RECEIVED WORD. RECEIVED WORD
10:     * CONTAINED IN R0, R0+1
11:     *
12:     *
13:     *
14:     * VARIABLE STORAGE AREA:
15:     *
16:     ORG      0
17:     R0      RMB      2      RECEIVED WORD R(X)
18:     M1      FCB      %10011000  MINIMUM POLYNOMIAL M1(X)
19:     M3      FCB      %11111000  MINIMUM POLYNOMIAL M3(X)
20:     M        RMB      1      DIVISOR
21:     TB1PTR  RMB      2      LOOKUP TABLE POINTERS
22:     TB2PTR  RMB      2
23:     S1      RMB      1      POWER SUM SYMMETRIC FUNCTION S1
24:     S3      RMB      1      POWER SUM SYMMETRIC FUNCTION S3
25:     SIGMA2  RMB      1      ELEMENTARY SYMMETRIC FUNCTION 2
26:     SHFCON  RMB      1      SHIFT COUNTER
27:     SLENP   RMB      1      EXPONENT OF S1
28:     TEMP1   RMB      1      TEMPORARY DATA STORE
29:     I        RMB      1      GALOIS FIELD EXPONENT COUNT
30:     GENPOL  FCB      %11101000,%10000000
31:     *
32:     ORG      $0300      EXPONENTS >> FIELD ELEMENTS
33:     TBL1    FCB      $1,$2,$4,$8,$3,$6,$C
34:     TBL2    FCB      $B,$5,$A,$7,$E,$F,$D,$9,$1
35:     *
36:     ORG      $0400      FIELD ELEMENTS >> EXPONENTS
37:     TBL3    FCB      0,$10,$1,$4,$2,$8,$5,$A,$3
38:     TBL4    FCB      $E,$9,$7,$6,$D,$B,$C
39:     *
40:     * BEGIN DECODING PROCEDURE
41:     *
42:     ORG      $0100
43:     DECODE  LDX      LTEL1
44:     STX     TB1PTR      INITIALISE LOOKUP POINTERS
45:     LDX     LTEL2
46:     STX     TB2PTR
47:     *
48:     * FIND POWER SUM SYMMETRIC FUNCTIONS S1 & S3
49:     *
50:     *
51:     * DIVISION ROUTINE TO FIND RESIDUE
52:     * OF R(X)/M :
53:     *
54:     SYND   LDA      B $11      INITIALISE SHIFT COUNT
55:           STA      B SHFCON
56:           LDA      A R0
57:           LDA      B R0+1
58:           EOR      A M1
59:           BIT      A $180      LEFT JUSTIFIED ?

```

```

60:           BNE      DIV1      YES: EXOR AGAIN
61:           ASL      B          NO: SHIFT DIVIDEND LEFT
62:           POL      A
63:           DEC      SHFCON
64:           BNE      TST1
65:           LSR      A          RIGHT JUSTIFY REMAINDER
66:           LSR      A
67:           LSR      A
68:           LSR      A
69:           STA      A S1
70:     *
71:     * COMPUTE S3 BY USING LOOKUP TABLES TO SUM
72:     * POWERS OF R(ALPHA**3).
73:     *
74:     SYND3  CLR      B $3      ZERO S3
75:           LDA      B $14      EXPONENT COUNT
76:           STA      B SHFCON
77:           LDA      A R0      GET WORD
78:           LDA      B R0+1
79:     SYND30 BIT      A $180      LEFT JUSTIFIED ?
80:           BNE      BIT      YES: FIND FIELD ELEMENT
81:     SYND31 ASL      B          SHIFT WORD
82:           POL      A
83:           DEC      SHFCON      NEXT EXPONENT
84:           BGE      SYND30     BRANCH IF >=0
85:           TST      S1         S1=0?
86:           BNE      SIG2
87:           TST      S3         S3=0?
88:           BNE      SIG2
89:     NOERR  RTS              ERROR-FREE WORD
90:           BIT              SAVE ADDS ON STACK
91:           PSH      B
92:           PSH      A
93:           LDA      B SHFCON   GET CURRENT EXPONENT
94:           TBA
95:           ASL      A          MULTIPLY BY 3
96:           ABA
97:     SYND32 CMP      A $15     ADD MODULO-15
98:           BLT      SYND33
99:           SUB      A $15
100:          BRA      SYND32     GO TEST AGAIN
101:          STA      A TB1PTR+1  SAVE EXP IN PTR
102:          LDX      A X         GET FIELD ELEMENT
103:          EOR      A S3        ADD IN TO S3
104:          STA      A S3        AND RETURN NEW VALUE
105:          PUL      A          RESTORE STACK
106:          PUL      T
107:          BRA      SYND31
108:     *
109:     * FIND SIGMA2=S1+S3/S1 (GF(2**4))
110:     *
111:     SIG2   LDA      A S1      GET S1
112:           BSR      GFSQR      FIND S1*S1
113:           STA      A TEMP1    SAVE RESULT
114:           LDA      A S3
115:           BEQ      ZERO      BRANCH IF ZERO
116:           LDA      B S1
117:           BSR      GFDIV     FIND S3/S1
118:           EOR      A TEMP1    S1*S1+S3/S1
119:           STA      A SIGMA2

```

```

0175 00 2E 120: BRA CONT CONTINUE WITH MAIN PROCEDURE
121: *****
122: * GALOIS FIELD SQUARING ROUTINE:
123: * FINDS ACCA=ACCA*ACCA IN 5F(2**4)
124: *****
0177 97 09 125: GFSQR STA A TBPTR+1 FIELD ELEMENT >>POINTER
0179 0E 07 126: LDX TBPTR FETCH POINTER
017B 46 00 127: LDA A X FETCH EXPONENT OF ROOT
017D 97 00 128: STA A SIENP SAVE EXPONENT
0177 4B 129: ASL A MULTIPLY BY 2
0180 91 0F 130: CMP A #15 MODULO-15
0182 00 02 131: BLT S01
0184 00 0F 132: SUB A #15
0186 97 06 133: SQ1 STA A TBPTR+1 NEW EXPONENT >>POINTER
0188 0E 05 134: LDX TBPTR FETCH POINTER
018A 46 00 135: LDA A X FETCH NEW FIELD ELEMENT
018C 39 136: RTS
137: *****
138: * GALOIS FIELD DIVISION ROUTINE:
139: * FINDS ACCA = ACCA/ACCB IN 5F(2**4)
140: *****
0190 97 03 141: GFDIV STA A TBPTR+1 FIELD ELEMENT >>POINTER
0192 0E 07 142: LDX TBPTR FETCH POINTER
0194 46 00 143: LDA A X FETCH EXPONENT
0196 07 03 144: STA B TBPTR+1 DIVISOR FIELD ELEMENT
0198 0E 07 145: LDX TBPTR
0197 06 00 146: LDA B X DIVISOR EXPONENT
0199 10 147: SBA SUBTRACT EXPONENTS
019A 2A 02 148: BPL QUOT
019C 05 0F 149: ADD A #15 MODULO-15
019E 97 06 150: QUOT STA A TBPTR+1 NEW EXPONENT >>POINTER
01A0 0E 05 151: LDX TBPTR GET POINTER
01A2 46 00 152: LDA A X QUOTIENT FIELD ELEMENT
01A4 39 153: RTS
154: *****
155: * BEGIN SEARCHING FOR ROOTS
156: *****
01A5 4F 157: CONT CLR A CLEAR "I"
01A6 07 0F 158: TEST STA A I SAVE CURRENT "I"
01A8 48 159: ASL A EXPONENT*2
01A9 91 0F 160: CMP A #15 MODULO-15
01AB 00 02 161: BLT S02
01AD 00 0F 162: SUB A #15
01AF 97 06 163: SQ2 STA A TBPTR+1
01B1 0E 05 164: LDX TBPTR
01B3 46 00 165: LDA A X GET SQUARED FIELD ELEMENT
01B5 97 05 166: STA A TEMP1 AND SAVE IT
01B7 04 00 167: LDA B SIENP GET EXPONENT OF S1
01B9 08 0F 168: ADD B I
01BB 01 0F 169: CMP B #15 MODULO-2 ADD
01BD 00 02 170: BLT MUL1
01BF 00 0F 171: SUB B #15
01C1 07 06 172: MUL1 STA B TBPTR+1 SAVE EXPONENT
01C3 0E 05 173: LDX TBPTR GET POINTER
01C5 06 00 174: LDA B X GET FIELD ELEMENT
01C7 0B 0E 175: EOR B TEMP1 FIND PARTIAL SUM (5F(2**4))
01C9 08 08 176: EOR B SIGMA2 ADD IN SIGMA2
01CB 07 04 177: BEQ ROOT ROOT FOUND ?
01CD 04 0F 178: RTI LDA A I FETCH LOCATOR
01CE 91 0F 179: CMP A #14

```

```

01D1 27 03 180: BEQ STOP STOP IF SEARCH DONE
01D3 4C 181: INC A INCREMENT LOCATOR
01D4 20 00 182: BRA TEST AND TEST AGAIN
01D6 39 183: STOP RTS SEARCH COMPLETE
01D7 5F 184: ROOT CLR B CLEAR SHIFT COUNTER
01D8 74 0000 185: LSR RX RIGHT JUSTIFY R(X)
01DB 76 0001 186: ROR RX+1
01DE 01 0F 187: TST2 CMP B I IS SHIFT COUNT = 1?
01E0 27 0F 188: BEQ BTXWAP YES; CORRECT ERRONEOUS BIT
01E2 06 189: ROOT1 TAP GET CARRY BIT
01E3 76 0000 190: ROR RX ROTATE R(X) RIGHT
01E6 76 0001 191: ROP RX+1
01E9 07 192: TPA SAVE CARRY
01EA 5C 193: INC B BUMP COUNTER
01EB 01 10 194: CMP B #16 IS SHIFT DONE ?
01ED 27 0E 195: BEQ RTI YES; EXIT
01EF 20 0D 196: BRA TST2 LOOP AGAIN
01F1 37 197: BTXWAP PSH B SAVE COUNTER ON STACK
01F2 08 01 198: LDA B RX+1
01F4 08 01 199: EOR B #1 SWITCH THE BIT IN ERROR
01F6 07 01 200: STA B RX+1 AND RESTORE R(X)
01F8 33 201: PUL B RESTORE STACK
01F9 20 0E 202: BRA ROOT1
203: END

```

NO ERROR(S) DETECTED

listings for chapter 7

(1) Transmitter software

```

1:      NAM   MODEM TRANSMITTER SOFTWARE
2:      OPT   MOD,LID,PAR5
3:      *
4:      *
5:      *      HF MODEM TRANSMITTER SOFTWARE
6:      *      *
7:      *
8:      * TRANSMISSION FORMAT:
9:      *
10:     * (1) NOISE ESTIMATION (NO TRANSMISSION)
11:     * (2) SYNC BLOCK (PHASE REVERSALS (179 ELS.))
12:     * M-SEQUENCE (31 ELS.)
13:     * ADVISORY SEQUENCE (30 ELS.) (FIXED FREQ.)
14:     * (3) 53 MESSAGE BLOCKS (14380 ELS.)
15:     *
16:     * BLOCK FORMAT:
17:     * 240 ELEMENTS, 16 DATA SUBCHANNELS,
18:     * 2 DATA SUBCHANNELS PER SUBCHANNEL FREQUENCY
19:     * (DIFFERENTIAL 4-PHASE SHIFT KEYING).
20:     *
21:     *
22:     *
23:     * BASE EQU $F000      EPROM OFFSET
24:     * LATCH EQU $C000     SLAVE CONTROL LATCH
25:     *
26:     * ORG $OFF8
27:     * FDB IRDRTN+BASE    IRQ VECTOR
28:     * ORG $0AFFE
29:     * FDB START+BASE    MASTER RESET VECTOR
30:     *
31:     * I/O ADDRESSES:
32:     *
33:     * INPIA EQU $6000     MODULATOR INPUT PIA
34:     * OUTPIA EQU $6C04    MODULATOR OUTPUT PIA
35:     * ADC EQU $3014      RECEIVER AUDIO
36:     * TXRY EQU $3010     TX/RX RELAY
37:     * ACIA EQU $300C     SERIAL INPUT TO MODEM
38:     * ACIAC EQU $300D    ACIA CONTROL REGISTER
39:     *
40:     * MASTER PROCESSOR VARIABLE STORAGE
41:     *
42:     * ORG 0
43:     * TBL1 RMB 32        CHANNEL SPECTRUM TABLE
44:     * TBL2 RMB 3         3 OPTIMUM SUBCHANNEL SLOTS
45:     * XTEMP RMB 2        INDEX REG TEMP STORES
46:     * XTEMP1 RMB 2
47:     * XTEMP2 RMB 2
48:     * FROM RMB 2
49:     * TO RMB 2
50:     * LKINDX RMB 2      LOOKUP TABLE POINTER
51:     * BINDX RMB 2       DATA POINTERS FOR FFT
52:     * CINDX RMB 2
53:     * REAL RMB 2        REAL FOURIER COEFFICIENT
54:     * IMAG RMB 2        IMAG FOURIER COEFFICIENT
55:     * Y RMB 2           MULTIPLY PARAMETERS
56:     * Z RMB 2
57:     * P RMB 4
58:     * TB RMB 1
59:     * I RMB 1           INDICES

```

```

0046      50: M      RMB 1
0047      51: L      RMB 1
0048      52: MIN     RMB 2
0049      53: COUNT1 RMB 1
004A      54: SHFCNT RMB 1      SHIFT COUNT
004B      55: CNCTR  RMB 2      SUBCHANNEL COUNTER
004C      56: PHPTR  RMB 2      PHASE POINTER
004E      57: ELCNT  RMB 1      SIGNAL ELEMENT COUNTER
0050      58: BITPTR  RMB 1      BIT POINTER
0051      59: BYTPTR  RMB 1      BYTE POINTER
0052      60: DATR   RMB 2      DATA POINTER
0053      61: TABPT  RMB 1
0055      62: BUFPTR  RMB 2      BUFFER POINTER
0056      63: DIBIT  RMB 1      DIBIT
0058      64: DTFLAG RMB 1      DATA TABLE FLAG
0059      65: ADVSEQ  RMB 16     ADVISORY SEQUENCE
005A      66: DTBL1  RMB 30
005B      67: DTBL2  RMB 30
005C      68: ORG   $0100
005D      69: DBASE  RMB 256
005E      70: CHN000 RMB 30      DATA SUBCHANNEL TABLES (1)
005F      71: CHN001 RMB 30
0060      72: CHN002 RMB 30
0061      73: CHN003 RMB 30
0062      74: CHN004 RMB 30
0063      75: CHN005 RMB 30
0064      76: CHN006 RMB 30
0065      77: CHN007 RMB 30
0066      78: CHN008 RMB 30
0067      79: CHN009 RMB 30
0068      80: CHN010 RMB 30
0069      81: CHN011 RMB 30
006A      82: CHN012 RMB 30
006B      83: CHN013 RMB 30
006C      84: CHN014 RMB 30
006D      85: CHN015 RMB 30
006E      86: CHN100 RMB 30      DATA SUBCHANNEL TABLES (2)
006F      87: CHN101 RMB 30
0070      88: CHN102 RMB 30
0071      89: CHN103 RMB 30
0072      90: CHN104 RMB 30
0073      91: CHN105 RMB 30
0074      92: CHN106 RMB 30
0075      93: CHN107 RMB 30
0076      94: CHN108 RMB 30
0077      95: CHN109 RMB 30
0078      96: CHN110 RMB 30
0079      97: CHN111 RMB 30
007A      98: CHN112 RMB 30
007B      99: CHN113 RMB 30
007C      100: CHN114 RMB 30
007D      101: CHN115 RMB 30
007E      102: BUFF  RMB 512     INPUT DATA BUFFER
007F      103: *
0080      104: * SLAVE VARIABLE STORAGE
0081      105: *
0082      106: ORG   $0001
0083      107: FLAG  RMB 1      MASTER HANDSHAKE
0084      108: LU    RMB 2      LOOKUP POINTER
0085      109: XTEMP3 RMB 2      XREG TEMP STORE

```



```

0997 20 B7 240: BRA SLAVES
241: *
242: * SLAVE PROCESSOR INTERRUPT ROUTINE:
243: *
0999 244: SLIRQ EQU *
0999 F6 6006 245: LDA B OUTPIA+2 GET CONTROL LINES
099C 03 10 246: EOR B E%00010000 SWAP SHIFT REGISTERS
099E F7 6006 247: STA B OUTPIA+2 AND RESTORE LINES
09A1 56 5000 248: LDA B INPIA CLEAR IRQ FLAG
09A4 39 249: RTI
250: *
251: * SLAVE VECTORS
252: *
09A5 00 07 253: SLVEC FDB SLIRQ-SLVRST+LATCH
09A7 00 07 254: FDB SLOG6
09A9 00 07 255: FDB SLOG6
09AB 00 07 256: FDB SLOG6
257: *****
258: * MAIN PROCEDURE BEGINS:
259: *****
09AD 0E 0004 260: START LDX E%0004 INITIALISE I/O
09B0 FF 3014 261: STX ADC
09B3 0E FF20 262: LDX E%FF20
09B6 FF 3016 263: STX ADC+2
09B9 0E FF04 264: LDX E%FF04 TX/RX RELAY CONTROL
09BC FF 3010 265: STX TXRX
09BF 96 FF 266: LDA A E%FF TXMIT
09C1 37 3010 267: STA A TXRX
09C4 26 80 268: LDA A E%80
09C6 37 3000 269: STA A ACIAC
09C9 96 1000 270: LDC E%1000 SYSTEM STACK
09CC 36 FE 271: LDA A E%FE ENSURE SLAVE IS HALTED
09CE 37 2000 272: STA A LATCH
09D1 7F 0061 273: CLR FLAG+LATCH
09D4 3D F849 274: JSR BOOT+BASE BOOTSTRAP LOADER
09D7 0E 0500 275: LDX E%0000
09DA 0F 56 276: STX BUFPTR INITIALISE BUFFER POINTER
09DC 0E 277: CLI
09DD 3D FA47 278: NMES JSR SFRAME+BASE BUILD START FRAME
09DE 0E 56 279: MDATA LDX BUFPTR
09E2 3C 07BF 280: ORX E%07BF+511 BUFFER FULL ?
09E5 26 69 281: BNE MDATA NO: WAIT TIL FULL
09E7 0E 0500 282: LDX E%0000 RESTORE BUFFER POINTER
09EA 0F 56 283: STX BUFPTR
09EC 0F 2E 284: STX FROM
09EE 0E 03E0 285: LDX E%03E0 SELECT TABLE
09F1 0F 30 286: STX TO
09F3 3D FA91 287: JSR INTLVE+BASE AND INTERLEAVE DATA
09F6 3D FC1E 288: JSR ROVE+BASE ASSESS NOISE IN CHANNEL
09FA 3D FC2E 289: JSR ADV+BASE CONSTRUCT ADVISORY SEQUENCE
09FC 4F 290: CLR A
09FD 37 55 291: STA A TABPT POINT TO TABLE 0
09FF 37 52 292: STA A BYTPTR BYTE 0
0A01 40 293: INC A
0A02 37 51 294: STA A BITPTR BIT 1
0A04 3D FD09 295: JSR OFRESD+BASE INITIALISE FREQUENCIES
0A07 7F 0A59 296: CLR ELCNT CLEAR ELEMENT COUNTER
0A09 36 FF 297: LDA A E%FF START TRANSMISSION
0A0C 37 2000 298: STA A LATCH
0A0E 2D FA3C 299: TX1 JSR PHASE+BASE

```

```

0A12 7D 0050 300: TST ELCNT ALL ELEMENTS TRANSMITTED ?
0A15 26 FA 301: BNE TX1 NO: LOOP AGAIN
0A17 3D FCE7 302: JSR NFREQ+BASE JUMP TO NEW FREQUENCIES
0A1A 06 1F 303: LDA B E%1 CODEWORD FRAME COUNTER
0A1C 37 304: NXFRAM PSH B
0A1D 0E 0200 305: LDX E%0200 BACK TO TABLE 0
0A20 3D 13 306: BSR TX0
0A22 0E 03E0 307: LDX E%03E0 NOW TABLE 1
0A25 3D 0E 308: BSR TX0
0A27 33 309: PUL B MORE FRAMES ?
0A29 5A 310: DEC B
0A2B 26 FA 311: BNE NXFRAM
0A2E 3D FA8C 312: TX5 JSR PHASE+BASE
0A32 7D 0050 313: TST ELCNT ALL ELEMENTS TRANSMITTED ?
0A31 26 FA 314: BNE TX5
0A33 20 AB 315: BRA NMES START NEW MESSAGE
316: *
317: * INTERLEAVE NEXT BLOCK. THEN WAIT FOR PREVIOUS
318: * BLOCK TO BE TRANSMITTED.
319: *
0A35 0F 30 320: TX0 STX TO
0A37 0E 05C0 321: LDX E%05C0
0A3A 0F 2E 322: STX FROM
0A3C 3D FB91 323: JSR INTLVE+BASE INTERLEAVE A BLOCK
0A3F 3D FA8C 324: TX2 JSR PHASE+BASE
0A42 7D 0050 325: TST ELCNT
0A45 26 FA 326: BNE TX2
327: *****
328: * CONSTRUCT "START" FRAME:
329: * PHASE REVERSAL SEQUENCE (179 ELEMENTS).
330: * SYNCHRONISATION SEQUENCE (31 ELEMENTS).
331: * FREQUENCY ADVISORY SEQUENCE (30 ELEMENTS).
332: *****
0A47 0E 0200 333: SFRAME LDX E%0200
0A4A 06 10 334: LDA B E%10 DATA SUBCHANNEL COUNTER
0A4C 37 335: SFRAM0 PSH B
0A4D 0F 28 336: STX XTEMP
0A4F 3D 12 337: BSR SFRAM1
0A51 96 29 338: LDA A XTEMP
0A53 06 29 339: LDA B XTEMP+1
0A55 0B 1E 340: ADD B E%10 NEXT DATA SUBCHANNEL
0A57 39 00 341: ADC A E%0
0A59 97 28 342: STA A XTEMP
0A5B 07 29 343: STA B XTEMP+1
0A5D 0E 28 344: LDX XTEMP
0A5F 33 345: PUL B
0A60 5A 346: DEC B
0A61 26 E9 347: BNE SFRAM0
348: *
0A63 0E 28 349: SFRAM1 LDX XTEMP
0A65 06 16 350: LDA B E%16
0A67 36 FF 351: LDA A E%FF
0A69 A7 00 352: SFRAM2 STA A X FILL WITH 1'S
0A6B 03 353: INX
0A6C 5A 354: DEC B
0A6D 26 FA 355: BNE SFRAM2
0A6F 0F 2A 356: STX XTEMP1 SAVE POINTER
0A71 06 03 357: LDA B E%3
0A73 0E FA89 358: LDX E%FA89+BASE
0A76 A6 00 359: SFRAM3 LDA A X

```



```

0858 DE 3C
0859 CE F99C
0860 DE 3E
0861 CE C9E8
0862 CE C9E8
0863 DE 10
0864 CE F882
0865 DE 3C
0866 CE F801
0867 DE 3E
0868 CE C007
0869 DE 01
0870 DE 33
0871 DE 40
0872 DE 30
0873 DE 30
0874 DE 3E
0875 DE 3E
0876 DE 0E
0877 DE 00
0878 DE 3C
0879 DE 40
0880 DE 00
0881 DE 03
0882 DE 3C
0883 DE 40
0884 DE 00
0885 DE 03
0886 DE 40
0887 DE 00
0888 DE 03
0889 DE 40
0890 DE 3E
0891 DE 3E
0892 DE 2A
0893 DE 2A
0894 DE 4A
0895 DE 4A
0896 DE 2A
0897 DE 4A
0898 DE 23
0899 DE 21
0900 DE 2E
0901 DE 2A
0902 DE 4A
0903 DE F1
0904 DE 2E
0905 DE 03
0906 DE 09
0907 DE 4A
0908 DE 00
0909 DE 2E
0910 DE 08
0911 DE 2E
0912 DE 02
0913 DE 2A
0914 DE 4A
0915 DE 09
0916 DE F5
0917 DE 16
0918 DE 32
0919 DE 39
0920 DE 03
0921 DE 30
0922 DE 56
0923 DE 0E
0924 DE 30
0925 DE 39
0926 DE 03
0927 DE 30
0928 DE 30
0929 DE 39
0930 DE 30
0931 DE F2
0932 DE F99C
0933 DE 05C0
0934 DE 2A
0935 DE 00
0936 DE 56
0937 DE 7F
0938 DE 05
0939 DE 30
0940 DE 30
0941 DE 03
0942 DE 03
0943 DE 03
0944 DE 30
0945 DE 30
0946 DE 30
0947 DE 30
0948 DE 30
0949 DE 30
0950 DE 30
0951 DE 30
0952 DE 30
0953 DE 30
0954 DE 30
0955 DE 30
0956 DE 30
0957 DE 30
0958 DE 30
0959 DE 30
0960 DE 30
0961 DE 30
0962 DE 30
0963 DE 30
0964 DE 30
0965 DE 30
0966 DE 30
0967 DE 30
0968 DE 30
0969 DE 30
0970 DE 30
0971 DE 30
0972 DE 30
0973 DE 30
0974 DE 30
0975 DE 30
0976 DE 30
0977 DE 30
0978 DE 30
0979 DE 30
0980 DE 30
0981 DE 30
0982 DE 30
0983 DE 30
0984 DE 30
0985 DE 30
0986 DE 30
0987 DE 30
0988 DE 30
0989 DE 30
0990 DE 30
0991 DE 30
0992 DE 30
0993 DE 30
0994 DE 30
0995 DE 30
0996 DE 30
0997 DE 30
0998 DE 30
0999 DE 30
1000 DE 30

```

```

430: STX Y
431: LDX Y
432: LSHR Y+BASE
433: STX Z
434: LSHR Y+LATCH
435: BSR
436: BSR
437: LDX
438: LSHR Y+BASE
439: STX Z
440: BSR
441: LSHR Y+LATCH
442: BSR
443: STX
444: LSHR Y+BASE
445: STX Z
446: BSR
447: BSR
448: STX
449: BSR
450: BSR
451: STX
452: LSHR Y+BASE
453: STX Z
454: BSR
455: BSR
456: STX
457: LSHR Y+BASE
458: STX Z
459: BSR
460: BSR
461: STX
462: LSHR Y+BASE
463: STX Z
464: BSR
465: BSR
466: STX
467: LSHR Y+BASE
468: STX Z
469: BSR
470: BSR
471: STX
472: LSHR Y+BASE
473: STX Z
474: BSR
475: BSR
476: STX
477: LSHR Y+BASE
478: STX Z
479: BSR
480: BSR
481: STX
482: LSHR Y+BASE
483: STX Z
484: BSR
485: BSR
486: STX
487: LSHR Y+BASE
488: STX Z
489: BSR
490: BSR
491: STX
492: LSHR Y+BASE
493: STX Z
494: BSR
495: BSR
496: STX
497: LSHR Y+BASE
498: STX Z
499: BSR
500: BSR
501: STX
502: LSHR Y+BASE
503: STX Z
504: BSR
505: BSR
506: STX
507: LSHR Y+BASE
508: STX Z
509: BSR
510: BSR
511: STX
512: LSHR Y+BASE
513: STX Z
514: BSR
515: BSR
516: STX
517: LSHR Y+BASE
518: STX Z
519: BSR
520: BSR
521: STX
522: LSHR Y+BASE
523: STX Z
524: BSR
525: BSR
526: STX
527: LSHR Y+BASE
528: STX Z
529: BSR
530: BSR
531: STX
532: LSHR Y+BASE
533: STX Z
534: BSR
535: BSR
536: STX
537: LSHR Y+BASE
538: STX Z
539: BSR
540: BSR
541: STX
542: LSHR Y+BASE
543: STX Z
544: BSR
545: BSR
546: STX
547: LSHR Y+BASE
548: STX Z
549: BSR
550: BSR
551: STX
552: LSHR Y+BASE
553: STX Z
554: BSR
555: BSR
556: STX
557: LSHR Y+BASE
558: STX Z
559: BSR
560: BSR
561: STX
562: LSHR Y+BASE
563: STX Z
564: BSR
565: BSR
566: STX
567: LSHR Y+BASE
568: STX Z
569: BSR
570: BSR
571: STX
572: LSHR Y+BASE
573: STX Z
574: BSR
575: BSR
576: STX
577: LSHR Y+BASE
578: STX Z
579: BSR
580: BSR
581: STX
582: LSHR Y+BASE
583: STX Z
584: BSR
585: BSR
586: STX
587: LSHR Y+BASE
588: STX Z
589: BSR
590: BSR
591: STX
592: LSHR Y+BASE
593: STX Z
594: BSR
595: BSR
596: STX
597: LSHR Y+BASE
598: STX Z
599: BSR
600: BSR
601: STX
602: LSHR Y+BASE
603: STX Z
604: BSR
605: BSR
606: STX
607: LSHR Y+BASE
608: STX Z
609: BSR
610: BSR
611: STX
612: LSHR Y+BASE
613: STX Z
614: BSR
615: BSR
616: STX
617: LSHR Y+BASE
618: STX Z
619: BSR
620: BSR
621: STX
622: LSHR Y+BASE
623: STX Z
624: BSR
625: BSR
626: STX
627: LSHR Y+BASE
628: STX Z
629: BSR
630: BSR
631: STX
632: LSHR Y+BASE
633: STX Z
634: BSR
635: BSR
636: STX
637: LSHR Y+BASE
638: STX Z
639: BSR
640: BSR
641: STX
642: LSHR Y+BASE
643: STX Z
644: BSR
645: BSR
646: STX
647: LSHR Y+BASE
648: STX Z
649: BSR
650: BSR
651: STX
652: LSHR Y+BASE
653: STX Z
654: BSR
655: BSR
656: STX
657: LSHR Y+BASE
658: STX Z
659: BSR
660: BSR
661: STX
662: LSHR Y+BASE
663: STX Z
664: BSR
665: BSR
666: STX
667: LSHR Y+BASE
668: STX Z
669: BSR
670: BSR
671: STX
672: LSHR Y+BASE
673: STX Z
674: BSR
675: BSR
676: STX
677: LSHR Y+BASE
678: STX Z
679: BSR
680: BSR
681: STX
682: LSHR Y+BASE
683: STX Z
684: BSR
685: BSR
686: STX
687: LSHR Y+BASE
688: STX Z
689: BSR
690: BSR
691: STX
692: LSHR Y+BASE
693: STX Z
694: BSR
695: BSR
696: STX
697: LSHR Y+BASE
698: STX Z
699: BSR
700: BSR
701: STX
702: LSHR Y+BASE
703: STX Z
704: BSR
705: BSR
706: STX
707: LSHR Y+BASE
708: STX Z
709: BSR
710: BSR
711: STX
712: LSHR Y+BASE
713: STX Z
714: BSR
715: BSR
716: STX
717: LSHR Y+BASE
718: STX Z
719: BSR
720: BSR
721: STX
722: LSHR Y+BASE
723: STX Z
724: BSR
725: BSR
726: STX
727: LSHR Y+BASE
728: STX Z
729: BSR
730: BSR
731: STX
732: LSHR Y+BASE
733: STX Z
734: BSR
735: BSR
736: STX
737: LSHR Y+BASE
738: STX Z
739: BSR
740: BSR
741: STX
742: LSHR Y+BASE
743: STX Z
744: BSR
745: BSR
746: STX
747: LSHR Y+BASE
748: STX Z
749: BSR
750: BSR
751: STX
752: LSHR Y+BASE
753: STX Z
754: BSR
755: BSR
756: STX
757: LSHR Y+BASE
758: STX Z
759: BSR
760: BSR
761: STX
762: LSHR Y+BASE
763: STX Z
764: BSR
765: BSR
766: STX
767: LSHR Y+BASE
768: STX Z
769: BSR
770: BSR
771: STX
772: LSHR Y+BASE
773: STX Z
774: BSR
775: BSR
776: STX
777: LSHR Y+BASE
778: STX Z
779: BSR
780: BSR
781: STX
782: LSHR Y+BASE
783: STX Z
784: BSR
785: BSR
786: STX
787: LSHR Y+BASE
788: STX Z
789: BSR
790: BSR
791: STX
792: LSHR Y+BASE
793: STX Z
794: BSR
795: BSR
796: STX
797: LSHR Y+BASE
798: STX Z
799: BSR
800: BSR
801: STX
802: LSHR Y+BASE
803: STX Z
804: BSR
805: BSR
806: STX
807: LSHR Y+BASE
808: STX Z
809: BSR
810: BSR
811: STX
812: LSHR Y+BASE
813: STX Z
814: BSR
815: BSR
816: STX
817: LSHR Y+BASE
818: STX Z
819: BSR
820: BSR
821: STX
822: LSHR Y+BASE
823: STX Z
824: BSR
825: BSR
826: STX
827: LSHR Y+BASE
828: STX Z
829: BSR
830: BSR
831: STX
832: LSHR Y+BASE
833: STX Z
834: BSR
835: BSR
836: STX
837: LSHR Y+BASE
838: STX Z
839: BSR
840: BSR
841: STX
842: LSHR Y+BASE
843: STX Z
844: BSR
845: BSR
846: STX
847: LSHR Y+BASE
848: STX Z
849: BSR
850: BSR
851: STX
852: LSHR Y+BASE
853: STX Z
854: BSR
855: BSR
856: STX
857: LSHR Y+BASE
858: STX Z
859: BSR
860: BSR
861: STX
862: LSHR Y+BASE
863: STX Z
864: BSR
865: BSR
866: STX
867: LSHR Y+BASE
868: STX Z
869: BSR
870: BSR
871: STX
872: LSHR Y+BASE
873: STX Z
874: BSR
875: BSR
876: STX
877: LSHR Y+BASE
878: STX Z
879: BSR
880: BSR
881: STX
882: LSHR Y+BASE
883: STX Z
884: BSR
885: BSR
886: STX
887: LSHR Y+BASE
888: STX Z
889: BSR
890: BSR
891: STX
892: LSHR Y+BASE
893: STX Z
894: BSR
895: BSR
896: STX
897: LSHR Y+BASE
898: STX Z
899: BSR
900: BSR
901: STX
902: LSHR Y+BASE
903: STX Z
904: BSR
905: BSR
906: STX
907: LSHR Y+BASE
908: STX Z
909: BSR
910: BSR
911: STX
912: LSHR Y+BASE
913: STX Z
914: BSR
915: BSR
916: STX
917: LSHR Y+BASE
918: STX Z
919: BSR
920: BSR
921: STX
922: LSHR Y+BASE
923: STX Z
924: BSR
925: BSR
926: STX
927: LSHR Y+BASE
928: STX Z
929: BSR
930: BSR
931: STX
932: LSHR Y+BASE
933: STX Z
934: BSR
935: BSR
936: STX
937: LSHR Y+BASE
938: STX Z
939: BSR
940: BSR
941: STX
942: LSHR Y+BASE
943: STX Z
944: BSR
945: BSR
946: STX
947: LSHR Y+BASE
948: STX Z
949: BSR
950: BSR
951: STX
952: LSHR Y+BASE
953: STX Z
954: BSR
955: BSR
956: STX
957: LSHR Y+BASE
958: STX Z
959: BSR
960: BSR
961: STX
962: LSHR Y+BASE
963: STX Z
964: BSR
965: BSR
966: STX
967: LSHR Y+BASE
968: STX Z
969: BSR
970: BSR
971: STX
972: LSHR Y+BASE
973: STX Z
974: BSR
975: BSR
976: STX
977: LSHR Y+BASE
978: STX Z
979: BSR
980: BSR
981: STX
982: LSHR Y+BASE
983: STX Z
984: BSR
985: BSR
986: STX
987: LSHR Y+BASE
988: STX Z
989: BSR
990: BSR
991: STX
992: LSHR Y+BASE
993: STX Z
994: BSR
995: BSR
996: STX
997: LSHR Y+BASE
998: STX Z
999: BSR
1000: BSR

```

TRANSFER LDA B L3

```

0015 2F 3010 600: RCVE CLR TXRX TRANSMITTER OFF.
0021 0E FFFF 601: LDX E3FFFF DELAY TO SETTLE
0024 09 602: RCVE1 DEK
0025 26 FD 603: BNE RCVE1
0027 3D FD34 604: JBR TRANS+BASE DO 3 TRANSFORMS
0029 3D FD13 605: JBR FIND+BASE FIND CLEAREST SLOTS
0030 39 606: RTS
007: *****
008: * CONSTRUCT SUBCHANNEL ADVISORY SEQUENCE
009: *****
002E 0E 0020 610: ADV LDX ETBL2 TABLE OF NUMBERS
0031 2F 2A 611: STX KTEMP1
0033 4F 612: CLR A
0034 0E 0000 613: LDX ETBL1 NEW TABLE
0037 2F 2C 614: ADV0 STX KTEMP2
0039 0E 2A 615: LDX KTEMP1
003B 36 616: RSH A SAVE COUNT
003C 49 617: RSL A
003D 43 618: RSL A
003E 49 619: RSL A
003F 42 620: RSL A
0040 49 00 621: ADD A X
0043 19 622: RSL A LEFT JUSTIFY INFO BITS
0043 3D BF 623: BSR CODE
0045 33 624: INK
0046 2F 2A 625: STX KTEMP1
0048 2E 2C 626: LDX KTEMP2
004A 47 00 627: STA A X
004C 57 01 628: STA B 1*X
004E 08 629: INK
004F 08 630: INK
0050 32 631: PUL A RESTORE COUNT
0051 4C 632: INC A NEXT SUBCHANNEL
0052 31 03 633: CMP A E3
0054 26 E1 634: BNE ADV0
0056 3D 23 635: BSR ADV00 INTERLEAVE TO DEPTH 2
0059 0E 021A 636: LDX ECHN000+26 ADD IN TO START FRAME
005B 2F 2A 637: STX KTEMP1
005D 26 04 638: LDR B E4 FOUR-FOLD SPECTRAL REDUNDANCY
005F 37 639: ADV23 RSH B
0060 0E 005A 640: LDX EADVSEQ
0063 2F 2C 641: STX KTEMP2
0065 06 04 642: LDR B E4 4 DATA SUBCHANNELS
0067 37 643: ADV22 RSH B
0069 06 04 644: LDR B E4 4 BYTES PER DATA SUBCHANNEL
006A 2E 2C 645: ADV21 LDX KTEMP2
006C 46 00 646: LDR A X
006E 03 647: INK
006F 2F 2C 648: STX KTEMP2
0071 2E 2A 649: LDX KTEMP1
0073 47 00 650: STA A X
0075 03 651: INK
0077 2F 2A 652: STX KTEMP1
0079 5A 653: DEC B
007B 26 EF 654: BNE ADV21
007D 46 2A 655: LDR A KTEMP1 MOVE TO NEXT SUBCHANNEL
007F 26 23 656: LDR B KTEMP1+1
0081 26 1A 657: ADD B E26
0083 39 00 658: RDC A E0
0085 07 29 659: STA B KTEMP1+1

```

```

0085 97 2A 660: STA A KTEMP1
0087 33 661: PUL B
0089 5A 662: DEC B
0089 26 DC 663: BNE ADV22
008B 33 664: PUL B
008C 5A 665: DEC B
008D 26 D0 666: BNE ADV23
008F 39 667: RTS
668: *
669: *
0090 06 04 670: ADV00 LDA B E4
0092 0E 0000 671: LDX ETBL1
0095 2F 2A 672: STX KTEMP1
0097 0E 005A 673: LDX EADVSEQ
009A 2F 2C 674: STX KTEMP2
009C 5F 03 675: CLR 3*X
009E 5F 07 676: CLR 7*X
00A0 5F 0B 677: CLR 11*X
00A2 5F 0F 678: CLR 15*X
00A4 37 679: ADV10 RSH B SAVE DATA CHANNEL COUNTER
00A5 06 07 680: LDA B E7
00A7 3D 1B 681: BSR ADV3
00A9 2E 2A 682: LDX KTEMP1
00AB 03 683: INK
00AC 06 03 684: LDA B E3 DO CHECKBITS
00AE 3D 14 685: BSR ADV3
00B0 2E 2A 686: LDX KTEMP1
00B2 03 687: INK
00B3 03 688: INK
00B4 03 689: INK
00B5 03 690: INK
00B6 2F 2A 691: STX KTEMP1
00B8 2E 2C 692: LDX KTEMP2
00BA 03 693: INK
00BB 03 694: INK
00BC 03 695: INK
00BD 03 696: INK
00BE 2F 2C 697: STX KTEMP2
00C0 33 698: PUL B
00C1 5A 699: DEC B
00C2 26 E0 700: BNE ADV10
701: *
702: * INTERLEAVE BITS:
703: *
00C4 2E 2A 704: ADV3 LDX KTEMP1
00C6 59 00 705: ROL X
00C8 09 706: INK
00C9 03 707: INK
00CA 2F 2A 708: STX KTEMP1
00CC 3D 0E 709: BSR ADV4
00CE 2E 2A 710: LDX KTEMP1
00D0 59 00 711: ROL X
00D2 09 712: DEK
00D3 09 713: DEK
00D4 2F 2A 714: STX KTEMP1
00D6 3D 04 715: BSR ADV4
00D8 5A 716: DEC B
00D9 26 E9 717: BNE ADV3
00DB 39 718: RTS
719: *

```

```

0000 0E 3C      720: ADVA  LDX  KTEMP2
000E 59 03      721:      ROL  3,K
00E0 59 02      722:      ROL  2,K
00E2 59 01      723:      ROL  1,K
00E4 59 00      724:      ROL  0,K
00E6 39          725:      RTS
726: *****
727: ◆ SET SUBCHANNELS TO NEW FREQUENCIES
728: *****
00E7 0E 0057    729: NFREQ LDX  ECHNTBL+LATCH
00EA 0F 2C      730:      STX  KTEMP2
00EC 0E 0020    731:      LDX  ETBL2
00EE 0F 2A      732:      STX  KTEMP1
00F1 0E 2A      733: NFREQ1 LDX  KTEMP1
00F3 06 00      734:      LDA  A  K
00F5 48          735:      ASL  A
00F6 28 06      736:      ADD  A  E6
00F8 09          737:      INX
00FA 0F 2A      738:      STX  KTEMP1
00FB 0E 3C      739:      LDX  KTEMP2
00FD 07 00      740:      STA  A  K
00FF 09          741:      INX
0000 0F 3C      742:      STX  KTEMP2
0002 3C 005F    743:      CPX  ECHNTBL+LATCH+8
0005 26 EA      744:      BNE  NFREQ1
0007 39          745:      RTS
746: *****
747: ◆ INITIALISE SUBCHANNEL FREQUENCIES TO
748: ◆ ALTERNATE SLOTS:
749: *****
0008 0E 0057    750: CFREQ LDX  ECHNTBL+LATCH
0009 26 06      751:      LDA  A  E6
000D 07 00      752: CFREQ1 STA  A  K
000F 28 04      753:      ADD  A  E4
0011 09          754:      INX
0012 3C 005F    755:      CPX  ECHNTBL+LATCH+8
0015 26 E6      756:      BNE  CFREQ1
0017 39          757:      RTS
758: *****
759: ◆ ROUTINE TO SELECT 8 SMALLEST 2-BYTE NUMBERS FROM
760: ◆ SET OF 16
761: *****
0018 0E 0020    762: FIND  LDX  ETBL2
0018 0F 28      763: FIND1 STX  KTEMP
0019 3D 1A      764:      BSR  TEST      FIND INDEX OF SMALLEST
001F 0E 3E      765:      LDX  Z          REMAINING NUMBER.
0031 26 FF      766:      LDA  A  E3FF    REPLACE MSB WITH FF
0033 07 00      767:      STA  A  K      SO DON'T GET IT AGAIN
0035 26 3F      768:      LDA  A  Z+1    DIVIDE INDEX BY 2
0037 24 1F      769:      AND  A  E%00011111 DON'T WANT BITS 5-7
0039 47          770:      ASR  A
003A 0E 28      771:      LDX  KTEMP      % STORE IN TABLE OF
003C 07 00      772:      STA  A  K      SUBCHANNEL NOS.
003E 3C 0027    773:      CPX  ETBL2+7    ALL DONE ?
0031 27 03      774:      BEO  FIND2      YES: SORT INTO ASCENDING ORDER
0033 09          775:      INX
0034 20 E5      776:      BSA  FIND1      NO: GET NEXT NO.
0036 3D 26      777: FIND2 BSR  CHSORT    SORT INTO ASCENDING ORDER
0038 39          778:      RTS
779: *****

```

```

780: ◆ FIND THE SMALLEST REMAINING 2-BYTE NUMBER:
781: *****
0039 36 FF      782: TEST  LDA  A  E3FF    GIVE MIN, MIN+1, A
003B 07 48      783:      STA  A  MIN    HIGH VALUE AT FIRST
003D 06 10      784:      LDA  B  E16    NUMBER COUNTER
003F 0E 001E    785:      LDX  ETBL1+30    TABLE OF NUMBERS
0042 37          786: TEST4 PSH  B          SAVE ON STACK
0043 06 01      787: TEST1 LDA  B  1,K    FETCH LSB
0045 06 00      788:      LDA  A  K      % MSB
0047 91 48      789:      CMP  A  MIN    MSB <= MIN ?
0049 22 0C      790:      BHI  TEST3    NO: IGNORE
004B 26 04      791: TEST2 BNE  TEST5    IF NOT =, MUST BE <
004D 01 49      792:      CMP  B  MIN+1  IF MSB=MIN, TEST LSB
004F 22 06      793:      BHI  TEST3    IF LSB>MIN+1, IGNORE
0051 97 48      794: TEST5 STA  A  MIN    MIN=MSB
0053 07 49      795:      STA  B  MIN+1  MIN+1=LSB
0055 0F 3E      796:      STX  Z          SAVE POSITION IN Z
0057 09          797: TEST3 DEK          GO TO NEXT NUM
0059 09          798:      DEK
0059 33          799:      PUL  B
005A 5A          800:      DEC  B          ALL DONE ?
005B 26 E5      801:      BNE  TEST4    NO: TEST NEXT NUM
005D 39          802:      RTS
803: *****
804: ◆ 8 BYTES FROM ETBL2->ETBL2+7 ARE SORTED INTO
805: ◆ ASCENDING ORDER.
806: *****
005E 0E 0020    807: CHSORT LDX  ETBL2
0061 0F 29      808: SORT1 STX  KTEMP
0063 06 00      809:      LDA  B  K      GET A BYTE
0065 09          810: SORT2 INX
0066 06 00      811:      LDA  A  K      GET ANOTHER
0068 11          812:      CBA          A>B ?
0069 22 0B      813:      BHI  NOSMAP    YES: DON'T SWAP 1EM
006B 07 00      814:      STA  B  K      YES: SWAP 1EM ROUND
006D 0F 3C      815:      STX  Y          SAVE K
006F 0E 29      816:      LDX  KTEMP
0071 07 00      817:      STA  A  K
0073 16          818:      TAB          NEW VALUE FOR ACCB
0074 0E 3C      819:      LDX  Y          RESTORE NREG
0076 3C 0027    820: NOSMAP CPX  ETBL2+7  ALL DONE ?
0079 26 EA      821:      BNE  SORT2    NO: CONTINUE
007B 0E 28      822:      LDX  KTEMP
007D 09          823:      INX          MOVE ALONG THERE
007E 3C 0027    824:      CPX  ETBL2+7  ALL SORTED OUT ?
0081 26 DE      825:      BNE  SORT1    NO: GO AGAIN
0083 39          826:      RTS
827: *****
828: ◆ DO 8 TRANSFORMS, SUMMING POWER SPECTRA
829: *****
0084 3D 0C      830: TRANS BSR  CLRtbl  CLEAR TABLE
0086 06 09      831:      LDA  B  E8
0088 37          832: LP1  PSH  B
0089 3D 44      833:      BSR  P3PEC    GET A POWER SPECTRUM
008B 3D 14      834:      BSR  CHSUM    SUM POINTS
008D 33          835:      PUL  B
008E 5A          836:      DEC  B
008F 26 F7      837:      BNE  LP1
0091 39          838:      RTS
839: *****

```

```

340: * CLEAR THE TABLE
341: *****
0092 06 10 342: CLR TBL LDA A E16
0094 0E 0000 343: LDX L TBL1
0097 0F 00 344: LP2 CLR X
0099 0F 01 345: CLR 1-X
0099 03 346: INX
0090 08 347: INX
0090 49 348: DEC A
0095 26 F7 349: BNE LP2
0090 39 350: RTS
351: *****
352: * SUM THE 16 POINTS FROM 450 HZ. TO 2700 HZ.
353: *****
0091 0E 0100 354: CHSUM LDX LDBASE+12
0094 0F 30 355: STX Y USE Y AS TEMP STORE
0096 0E 0000 356: LDX L TBL1
0099 0F 3E 357: STX Z USE Z AS TEMP STORE
0099 06 10 358: LDA B E16
0090 37 359: CHSUM1 PSH B SAVE POINT COUNTER
0095 0E 30 360: LDX Y
0090 06 04 361: LDA B E4
0082 03 30 362: ADD B Y+1 POINT TO NEW ADDRESS
0084 07 30 363: STA B Y+1
0085 06 00 364: LDA A X GET SPECTRAL DATA
0083 06 01 365: LDA B 1-X
008A 0E 3E 366: LDX Z
0080 03 01 367: ADD B 1-X
008E 09 00 368: ADD A X ADD IN NEW VALUES
0080 07 00 369: STA A K
0082 07 01 370: STA B 1-X
0084 06 02 371: LDA B E2 POINT TO NEW ADDR
0086 03 3F 372: ADD B Z+1
0082 07 3F 373: STA B Z+1
008A 38 374: PUL B
0083 0A 375: DEC B
0080 26 0F 376: BNE CHSUM1
008E 39 377: RTS
378: *****
379: * FIND POWER SPECTRUM OF INPUT SIGNAL
380: *****
008F 30 FE0F 381: PSPEC JSR SEMPL+BASE GET DATA
0082 30 38 382: BCR FFT PERFORM TRANSFORM
0084 0E 0100 383: LDX LDBASE
0087 0F 39 384: STX KTEMP POINTER
0089 06 20 385: LDA B E32 COUNTER
0089 37 386: PSPEC1 PSH B
0080 0E 20 387: LDX KTEMP
008E 0E 00 388: LDX 0-X GET REAL COMPONENT
0080 0F 30 389: STX Y
0082 0F 3E 390: STX Z
0084 30 FF85 391: JSR MULT16+BASE FIND R+R
0087 06 40 392: LDA A P
0089 06 41 393: LDA B P+1
0089 07 394: PSH B SAVE PARTIAL RESULT ON STACK
0080 3E 395: PSH A
0080 0E 20 396: LDX KTEMP
008E 0E 02 397: LDX 2-X GET IMAG COMPONENT
0081 0F 30 398: STX Y
0083 0F 3E 399: STX Z

```

```

00F5 80 FF85 900: JSR MULT16+BASE FIND I+I
00F9 32 901: PUL A FETCH R+R
00F9 33 902: PUL B
00FA 0B 41 903: ADD B P+1 R+R+I+I
00FC 09 40 904: ADD A P
00FE 0E 23 905: LDX KTEMP
0E00 07 00 906: STA A X RESTORE TO TABLE
0E02 07 01 907: STA B 1-X
0E04 06 29 908: LDA B KTEMP+1
0E06 03 04 909: ADD B E4 BUMP POINTER
0E09 07 29 910: STA B KTEMP+1
0E0A 33 911: PUL B FETCH COUNTER
0E0B 0A 912: DEC B
0E0C 26 CD 913: BNE PSPEC1
0E0E 39 914: RTS
915: *****
916: * FFT KERNEL: FIXED LENGTH: 64 POINTS *
917: *****
0E0F 0E 0100 918: FFT LDX LDBASE
0E12 0F 34 919: STX BNDX INITIALISE POINTERS
0E14 0F 36 920: STX CINDX
0E16 0E F900 921: LDX LKUP+BASE
0E19 0F 32 922: STX LKINDX
0E1B 4F 923: CLR A
0E1C 07 46 924: STA A M M=0
0E1E 07 47 925: STA A L L=0
0E20 40 926: INC A
0E21 07 45 927: STA A I I=1
0E23 30 FE09 928: BEGIN JSR INDX+BASE LOCATE PAIR OF POINTS
0E26 06 46 929: LDA B M
0E29 27 00 930: BEQ NOMULT M=0?
0E2A 80 FEBC 931: JSR NN+BASE NO: FIND ADDRESS OF LOOKUPS
0E2D 07 33 932: STA B LKINDX+1 SAVE LSBYTE
0E2F 0E 36 933: LDX CINDX ADDRESS OF BUTTERFLY
0E31 0E 00 934: LDX X (REAL PART)
0E33 0F 30 935: STX Y PUT IN MULTIPLY
0E35 20 07 936: BRA ISLE1
0E37 80 FF39 937: NOMULT JSR NOMLT+BASE DON'T MULTIPLY
0E3A 20 4E 938: BRA TSTM CONTINUE
0E3C 20 E5 939: BEGIN1 BRA BEGIN
0E3E 0E 32 940: ISLE1 LDX LKINDX
0E40 0E 00 941: LDX X GET COSINE LOOKUP
0E42 0F 3E 942: STX Z SAVE IN MULTIPLICAND
0E44 30 FF85 943: JSR MULT16+BASE FIND R1+R2
0E47 0E 41 944: LDX P+1 GET RESULT
0E49 0F 39 945: STX REAL AND SAVE
0E4B 0E 36 946: LDX CINDX
0E4D 0E 02 947: LDX 2-X IMAG BUTTERFLY DATA
0E4F 0F 30 948: STX Y SAVE IN MULTIPLIER
0E51 80 FF85 949: JSR MULT16+BASE FIND I1+I2
0E54 0E 41 950: LDX P+1 GET RESULT
0E56 0F 3A 951: STX IMAG AND SAVE
0E58 0E 36 952: LDX CINDX
0E5A 0E 02 953: LDX 2-X IMAG BUTTERFLY DATA
0E5C 0F 30 954: STX Y IN MULTIPLIER
0E5E 0E 32 955: LDX LKINDX
0E60 0E 40 956: LDX 64-X GET -SINE LOOKUP
0E62 0F 3E 957: STX Z PUT IN MULTIPLICAND
0E64 30 FF85 958: JSR MULT16+BASE FIND I1+I2
0E67 06 38 959: LDA A REAL

```



```

0F16 5F 1030: CLR B ACCB IS MSB OF SCALED DATA
0F17 49 1031: ASL A ROTATE DATA LEFT
0F18 59 1032: ROL B MSBIT TO ACCB
0F19 49 1033: ASL A
0F1A 59 1034: ROL B
0F1B 47 01 1035: STA A 1,X SAVE LSB OF SCALED DATA
0F1C 05 02 1036: BIT B 12000000010 -VE ?
0F1E 36 02 1037: SNE NEG YES: BRANCH
0F21 29 03 1038: BRA SCL1 NO: DO NOTHING
0F23 09 FD 1039: NEG EOR B 1X111111100 GET REMAINING BITS
0F25 01 1040: NOP WAIT EXTRA TIME
0F26 07 00 1041: SCL1 STA B X NOW STORE MSB
0F27 07 02 1042: CLR 2,X CLEAR IMAG COMPONENT
0F28 07 03 1043: CLR 3,X
0F2C 06 07 1044: LDA 6 17 TIMING LOOP
0F2E 5A 1045: SEMPL3 DEC B WAIT FOR FULL PERIOD
0F2F 25 FD 1046: SNE SEMPL3 MAKE UP TO 136 CYCLES
0F31 33 1047: ROL B GET COUNTER OFF STACK
0F32 50 1048: INC B AND BUMP IT
0F33 01 40 1049: CMP B 164 FINISHED ?
0F35 26 09 1100: SNE SEMPL1 NO: GET NEXT SAMPLE
0F37 39 1101: RTS YES: RETURN: BUFFER FULL.
1102: *****
1103: * DONT MULTIPLY BY LOOKUPS
1104: *****
0F38 0E 36 1105: NDMLT LDX CINDX BUTTERFLY
0F3A 0E 00 1106: LDX X DATA GOES DIRECTLY
0F3C 0F 33 1107: STX REAL INTO "REAL"
0F3E 0E 36 1108: LDX CINDX AND "IMAG"
0F40 0E 02 1109: LDX 2,X WITHOUT NEED FOR
0F42 0F 3A 1110: STX IMAG MULTIPLYING
1111: *****
1112: * FIND 2 NEW POINTS FROM 2 OLD
1113: *****
0F44 0E 34 1114: NWDATA LDX BINDX GET CURRENT DATA PTR
0F46 06 00 1115: LDA A X REAL COMPONENT
0F48 06 01 1116: LDA B 1,X
0F4A 07 30 1117: STA A Y Y IS TEMP STORE
0F4C 07 3D 1118: STA B Y+1
0F4E 09 39 1119: ADD B REAL+1 ADD TO REAL PRODUCT
0F50 09 38 1120: ADD A REAL
0F52 07 00 1121: STA A X SAVE IN CURRENT POSITION
0F54 07 01 1122: STA B 1,X
0F56 06 02 1123: LDA A 2,X GET IMAG DATA
0F58 06 03 1124: LDA B 3,X
0F5A 07 3E 1125: STA A Z Z IS TEMP STORE
0F5C 07 3F 1126: STA B Z+1
0F5E 08 38 1127: ADD B IMAG+1 ADD TO IMAG PRODUCT
0F60 09 3A 1128: ADD A IMAG
0F62 0E 34 1129: LDX BINDX
0F64 07 02 1130: STA A 2,X SAVE IN CURRENT POSITION
0F66 07 03 1131: STA B 3,X
0F68 06 30 1132: LDA A Y GET OLD DATA (REAL)
0F6A 06 3D 1133: LDA B Y+1
0F6C 06 39 1134: SUB B REAL+1 SUBTRACT REAL PRODUCT
0F6E 06 38 1135: SBC A REAL
0F70 0E 36 1136: LDX CINDX BUTTERFLY DATA POINTER
0F72 07 01 1137: STA A X SAVE REAL RESULT
0F74 07 01 1138: STA B 1,X
0F76 06 3E 1139: LDA A Z GET OLD DATA (IMAG)

```

MODEM TRANSMITTER SOFTWARE

```

0F78 06 3F 1140: LDA B Z+1
0F7A 00 3B 1141: SUB B IMAG+1 SUBTRACT IMAG PRODUCT
0F7C 02 3A 1142: SBC A IMAG
0F7E 0E 36 1143: LDX CINDX BUTTERFLY
0F80 07 02 1144: STA A 2,X SAVE IMAG RESULT
0F82 07 03 1145: STA B 3,X
0F84 39 1146: RTS
1147: *****
1148: * 16 BY 16 MULTIPLICATION ROUTINE USING BOOTH'S
1149: * ALGORITHM.
1150: * MULTIPLIER: Y,Y+1
1151: * MULTIPLICAND: Z,Z+1
1152: * PRODUCT: P,P+1,P+2,P+3,P+4
1153: * MULTIPLIER DESTROYED
1154: * MULTIPLICAND PRESERVED
1155: *****
0F85 0E 0000 1156: MULT16 LDX 10 CLEAR REGS.
0F88 0F 40 1157: STX P
0F8A 0F 42 1158: STX P+2
0F8C 7F 0044 1159: CLR TB CLEAR TEST BYTE
0F8F 0E 0010 1160: LDX 116
0F92 96 3D 1161: LOOP LDA A Y+1
0F94 34 01 1162: AND A 11
0F96 16 1163: TAB
0F97 98 44 1164: EOR A TB Y(LSB)=Y(LSB-1) ?
0F99 27 1D 1165: BEQ SHIFT YES: SHIFT
0F9B 5D 1166: TST B
0F9C 27 0E 1167: BEQ ADD Y(LSB)=0
0F9E 96 40 1168: SUBTR LDA A P
0FA0 06 41 1169: LDA B P+1
0FA2 00 3F 1170: SUB B Z+1
0FA4 92 3E 1171: SBC A Z
0FA6 97 40 1172: STA A P
0FA8 07 41 1173: STA B P+1
0FAA 20 0C 1174: BRA SHIFT
0FAC 96 40 1175: ADD LDA A P
0FAE 06 41 1176: LDA B P+1
0FB0 08 3F 1177: ADD B Z+1
0FB2 39 3E 1178: ADC A Z
0FB4 97 40 1179: STA A P
0FB6 07 41 1180: STA B P+1
0FB8 7F 0044 1181: SHIFT CLR TB CLEAR TEST BYTE
0FBB 76 0030 1182: ROR Y
0FBE 76 003D 1183: ROR Y+1
0FC1 79 0044 1184: ROL TB
0FC4 77 0040 1185: ADR P
0FC7 76 0041 1186: ROR P+1
0FCA 76 0042 1187: ROR P+2
0FCD 76 0043 1188: ROR P+3
0FD0 09 1189: DEK DEC SHIFT COUNT
0FD1 26 3F 1190: SNE LOOP
0FD3 39 1191: RTS
1192: END

```

NO ERROR(S) DETECTED

listings for chapter 8

(1) Transmitter software

(2) Receiver software

TRANSMITTER SOFTWARE

SSB MNEMONIC ASSEMBLER PAGE 3

```

0351 87 8000 120: STA A INPIA SHOVE IN OTHER END
0354 57 8002 121: STA B INPIA+2 AND SHIFT
0357 05 84 122: LDX XTEMP1 RAM TABLE POINTER
0359 47 00 123: STA A X
035B 57 01 124: STA B 1*X
035D 03 125: INX BUMP TABLE POINTER
035E 03 126: INX
035F 5F 04 127: STX XTEMP1 NEW VALUE
0361 26 05 128: LDA A STEP GET STEP
0363 3B 03 129: ADD A LU+1 AND COMPUTE NEW
0365 91 57 130: CMP A #37 POINTER VALUE MOD-40
0367 2D 02 131: BLT NOSUB2 AS BEFOR
0369 80 50 132: SUB A #30
036B 37 03 133: NOSUB2 STA A LU+1
036D 33 134: PUL B GET COUNTER OFF STACK
036E 59 135: DEC B DONE 40 ?
036F 26 01 136: BNE SLAVE3 IF NOT, DO NEXT
0371 0E 0047 137: LDX #BASE
0374 15 23 138: LDA B #40
0376 46 00 139: SLAVE4 LDA A X
0378 57 8000 140: STA A INPIA
037B 46 01 141: LDA A 1*X
037D 57 8002 142: STA A INPIA+2
037F 03 143: INX
0381 03 144: INX
0383 54 145: DEC B
0385 26 01 146: BNE SLAVE4
0388 39 27 147: BRA SLAVE5
148: *
149: * SLAVE PROCESSOR INTERRUPT ROUTINE:
150: *
0387 151: SLIRD EDU *
0387 46 8006 152: LDA B OUTPIA+2 GET CONTROL LINES
0389 03 10 153: EDR B #00010000 SWAP SHIFT REGISTERS
038B 57 8006 154: STA B OUTPIA+2 AND RESTORE LINES
038E 46 8000 155: LDA B INPIA CLEAR IRQ FLAG
0390 38 156: RTI
157: *
158: * SLAVE VECTORS
159: *
0393 01 0E 160: SLVEC FDB SLIRD-SLVRST+SLORG
0395 01 07 161: FDB SLVRST-SLVRST+SLORG
0397 01 07 162: FDB SLVRST-SLVRST+SLORG
0399 01 07 163: FDB SLVRST-SLVRST+SLORG
164: *
165: * BEGIN MAIN PROCEDURE
166: *
039B 0E 8F04 167: INITLSE LDX #8FF04 INITIALISE SWITCHES
039E 4F 8010 168: STX KEYS
03A1 3E 8030 169: LDC #10300 MASTER STACK POINTER
03A4 26 FF 170: LDA A #8FF
03A6 57 8010 171: STA A KEYS TX OFF/ ON KEY OFF.
03A9 36 FE 172: LDA A #8FE
03AB 57 0000 173: STA A LATCH BRING RST ON SLAVE LOW
03AD 50 80E9 174: JSR #000+BASE BOOT IN SLAVE PROCEDURES
03AF 36 FE 175: START LDA A #8FE
03B1 57 0000 176: STA A LATCH
03B4 36 07 177: LDA A #7 INITIALISE TO ZERO PHASE
03B7 50 80D0 178: JSR #040+BASE
03B9 36 14 179: LDA A #20 SET UP TONE (N 1500 HZ.

```

TRANSMITTER SOFTWARE

SSB MNEMONIC ASSEMBLER PAGE 4

```

094D 8D 80AF 180: JSR #SLOTS+BASE
0950 36 FF 181: LDA A #8FF
0952 87 0000 182: STA A LATCH
0955 36 FD 183: LDA A #8FD TX ON/ ON KEY OFF
0957 87 8010 184: STA A KEYS
095A 8D 809C 185: JSR #MORSE+BASE ANNOUNCE STATION I.D.
095D 7F 8010 186: CLR KEYS TX ON/ KEY ON
0960 36 FF 187: LDA A #8FF
0962 87 0000 188: STA A LATCH
0965 0E 0800 189: LDX #2048
0968 8D 80D4 190: TONE JSR #WAIT+BASE SEND TONE
096B 09 191: DEX
096D 26 FA 192: BNE TONE
193: *****
194: * SEND 3 LOTS OF THE FOLLOWING:
195: * MESSAGE 1: PREAMBLE, 4 CHARACTER SEQUENCES, NO CODING.
196: * MESSAGE 2: PREAMBLE, 4 CHARACTER SEQUENCES, 3CH
197: * CODING, NO INTERLEAVING.
198: * MESSAGE 3: PREAMBLE, 4 CHARACTER SEQUENCES, 3CH
199: * CODING, INTERLEAVING TO DEPTH 16.
200: *****
096E 06 09 201: MSEN LDA B #3
0970 37 202: MSENS PSH B SHOVE COUNTER ON STACK
0971 8D 80CF 203: JSR #MSEN1+BASE DO 1ST MESSAGE
0974 8D 80CF 204: JSR #MSEN1+BASE DO IT AGAIN
0977 8D 80E5 205: JSR #MSEN2+BASE DO 2ND MESSAGE
097A 8D 80E5 206: JSR #MSEN2+BASE DO IT AGAIN
097D 8D 80D0 207: JSR #MSEN3+BASE DO 3RD MESSAGE
0980 8D 80D0 208: JSR #MSEN3+BASE DO IT AGAIN
0983 32 209: PUL B GET COUNTER
0984 5A 210: DEC B DO WHOLE LOT 3 TIMES
0985 26 E9 211: BNE MSENS
0987 7E 8041 212: JMP #START+BASE REPEAT FROM START
213: *
214: * MORSE CODE TRANSMISSION ROUTINE:
215: *
098A 02 216: MMES FCB #3, #A, #1, #4, #3A, #2A, #11
0991 45 217: FCB #45, #14, #54, #22, #3A, #11, #51, #55 REPRESENTATION D
F MESSAGE
0993 03 218: FCB #3, #A, #30 'OE 598LD'
0996 86 8010 219: MORSE LDA A KEYS
099F 8A 01 220: ORA A #00000001 ENSURE KEY IS OFF
09A1 87 8010 221: STA A KEYS
09A4 0E 809A 222: LDX #MMES+BASE GET START OF MORSE MESSAGE
09A7 06 03 223: MRS0 LDA B #3 BIT COUNT
09A9 46 00 224: LDA A X GET A BYTE
09AB 37 225: MRS1 PSH B
09AC 43 226: PSH A GET OUT BIT
09AD 24 08 227: BCC DLY IF NOT SET, NO CHANGE
09AF 46 8010 228: LDA B KEYS
09B2 03 01 229: EDR B #1 SWITCH MORSE KEY
09B4 57 8010 230: STA B KEYS
09B7 8D 0B 231: DLY BR #SPACE GO TO WAIT LOOP
09B9 33 232: MRS2 PUL B
09BA 5A 233: DEC B GET NEXT BIT
09BB 26 EE 234: BNE MRS1
09BD 03 235: INX GET NEXT BYTE
09BE 9C 809C 236: CPX #MMES+BASE+13
09C1 26 E4 237: BNE MRS0 GO TO END OF MESSAGE
09C3 33 238: RTS
09C4 0F 00 239: SPACE STX XTEMP SAVE KRES

```

```

0415 1E 3339 240: LDX L9000 LDDP PARAMETER
0416 09 241: SOCI DEY
0417 09 242: SVE SOCI
0418 09 243: LDX XTEMP RESTORE XREG
0419 39 244: RTS

```

```

*****
245: * MESSAGE 1: SEND PREAMBLE, 4 CHAR SEQUENCES.
246: *****
247: *****
248: MSEN1 SVE PBLE SEND PREAMBLE
249: LDR A L11101000 1ST MESSAGE I.D. BYTE
250: LDR B L7
251: JCR SIBITS+BASE SEND AS SINGLE CHAR
252: LDR B L4 CHARACTER SEQUENCE COUNTER
253: MSEN10 PCH B EMSEB+BASE BASE OF 1ST MESSAGE
254: LDX STRINGS SEND AS A STRING
255: PUL B
256: DEC B
257: SVE MSEN10
258: RTS

```

```

*****
259: * MESSAGE 2: SEND PREAMBLE, 4 CHAR SEQUENCES CODED WITH
260: NO INTERLEAVINGS:
261: *****
262: *****
263: MSEN2 SVE PBLE SEND PREAMBLE
264: LDR A L11010110 2ND MESSAGE I.D.
265: LDR B L7
266: JCR SIBITS+BASE SEND AS SINGLE CHAR
267: LDR B L4 SEQUENCE COUNTER
268: MSEN20 PCH B EMSEB+BASE
269: LDX STRINGS
270: PUL B
271: DEC B
272: SVE MSEN20
273: RTS

```

```

*****
274: * MESSAGE 3: PREAMBLE, 4 CHAR SEQUENCES BOH ENCODED,
275: INTERLEAVED TO DEPTH 15
276: *****
277: *****
278: MSEN3 SVE PBLE SEND PREAMBLE
279: LDR A L110011100 3RD MESSAGE I.D.
280: LDR B L7
281: JCR SIBITS+BASE
282: LDR B L4
283: MSEN30 PCH B EMSEB+BASE
284: LDX STRINGS
285: PUL B
286: DEC B
287: SVE MSEN30
288: RTS

```

```

*****
289: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
290: *****
291: *****
292: MSEN4 SVE PBLE SEND PREAMBLE
293: LDR A L110011100 4TH MESSAGE I.D.
294: LDR B L7
295: JCR SIBITS+BASE
296: LDR B L4
297: MSEN40 PCH B EMSEB+BASE
298: LDX STRINGS
299: PUL B
300: DEC B
301: SVE MSEN40
302: RTS

```

```

*****
303: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
304: *****
305: *****
306: MSEN5 SVE PBLE SEND PREAMBLE
307: LDR A L110011100 5TH MESSAGE I.D.
308: LDR B L7
309: JCR SIBITS+BASE
310: LDR B L4
311: MSEN50 PCH B EMSEB+BASE
312: LDX STRINGS
313: PUL B
314: DEC B
315: SVE MSEN50
316: RTS

```

```

*****
317: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
318: *****
319: *****
320: MSEN6 SVE PBLE SEND PREAMBLE
321: LDR A L110011100 6TH MESSAGE I.D.
322: LDR B L7
323: JCR SIBITS+BASE
324: LDR B L4
325: MSEN60 PCH B EMSEB+BASE
326: LDX STRINGS
327: PUL B
328: DEC B
329: SVE MSEN60
330: RTS

```

```

*****
331: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
332: *****
333: *****
334: MSEN7 SVE PBLE SEND PREAMBLE
335: LDR A L110011100 7TH MESSAGE I.D.
336: LDR B L7
337: JCR SIBITS+BASE
338: LDR B L4
339: MSEN70 PCH B EMSEB+BASE
340: LDX STRINGS
341: PUL B
342: DEC B
343: SVE MSEN70
344: RTS

```

```

*****
345: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
346: *****
347: *****
348: MSEN8 SVE PBLE SEND PREAMBLE
349: LDR A L110011100 8TH MESSAGE I.D.
350: LDR B L7
351: JCR SIBITS+BASE
352: LDR B L4
353: MSEN80 PCH B EMSEB+BASE
354: LDX STRINGS
355: PUL B
356: DEC B
357: SVE MSEN80
358: RTS

```

```

0422 05 08 300: LDR B L8
0423 2D 4E 301: BCR SIBITS
0424 08 302: INX
0425 08 303: BRR MSEN31
0426 2D 03 304: MSEN32 PUL B
0427 2D 03 305: MSEN32 PUL B
0428 2D 03 306: DEC B
0429 2D 03 307: SVE MSEN30
0430 39 308: RTS

```

```

*****
309: * SEND PREAMBLE AND/OR STRING:
310: *****
311: *****
312: PBLE LDR A L132
313: PBLE1 PCH B WAIT+BASE
314: LDR B L20
315: BRR SLOTS
316: JCR WAIT+BASE
317: CLR A
318: BRR SLOTS
319: PUL B
320: DEC B
321: SVE PBLE1
322: LDX EMSEB+BASE SEND PREAMBLE
323: LDR A X
324: INX SIBITS
325: BRR SIBITS
326: LDR A X
327: BRR SIBITS
328: RTS

```

```

*****
329: * SEND STRING OF ASCII CHARS
330: *****
331: *****
332: STRING LDR A X
333: CMP A L7L
334: BRR STRNG0
335: BRR CHAR
336: INX
337: BRR STRNG0
338: STRNG0 RTS
339: *
340: FOR BRR59
341: * SEND AN ASCII CHARACTER:
342: *****
343: *****
344: CHR SVE TEMP
345: PCH A
346: BRR WAIT
347: LDR A L20
348: BRR SLOTS
349: PUL A
350: ADD A
351: LDR B L7
352: BRR SIBITS
353: BRR WAIT
354: CLR A
355: BRR SLOTS
356: BRR TEMP
357: RTS

```

```

*****
358: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
359: *****
360: *****
361: MSEN9 SVE PBLE SEND PREAMBLE
362: LDR A L110011100 9TH MESSAGE I.D.
363: LDR B L7
364: JCR SIBITS+BASE
365: LDR B L4
366: MSEN90 PCH B EMSEB+BASE
367: LDX STRINGS
368: PUL B
369: DEC B
370: SVE MSEN90
371: RTS

```

```

*****
372: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
373: *****
374: *****
375: MSEN10 SVE PBLE SEND PREAMBLE
376: LDR A L110011100 10TH MESSAGE I.D.
377: LDR B L7
378: JCR SIBITS+BASE
379: LDR B L4
380: MSEN100 PCH B EMSEB+BASE
381: LDX STRINGS
382: PUL B
383: DEC B
384: SVE MSEN100
385: RTS

```

```

*****
386: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
387: *****
388: *****
389: MSEN11 SVE PBLE SEND PREAMBLE
390: LDR A L110011100 11TH MESSAGE I.D.
391: LDR B L7
392: JCR SIBITS+BASE
393: LDR B L4
394: MSEN110 PCH B EMSEB+BASE
395: LDX STRINGS
396: PUL B
397: DEC B
398: SVE MSEN110
399: RTS

```

```

*****
399: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
400: *****
401: *****
402: MSEN12 SVE PBLE SEND PREAMBLE
403: LDR A L110011100 12TH MESSAGE I.D.
404: LDR B L7
405: JCR SIBITS+BASE
406: LDR B L4
407: MSEN120 PCH B EMSEB+BASE
408: LDX STRINGS
409: PUL B
410: DEC B
411: SVE MSEN120
412: RTS

```

```

*****
413: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
414: *****
415: *****
416: MSEN13 SVE PBLE SEND PREAMBLE
417: LDR A L110011100 13TH MESSAGE I.D.
418: LDR B L7
419: JCR SIBITS+BASE
420: LDR B L4
421: MSEN130 PCH B EMSEB+BASE
422: LDX STRINGS
423: PUL B
424: DEC B
425: SVE MSEN130
426: RTS

```

```

*****
427: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN BOH INDICATED
428: *****
429: *****
430: MSEN14 SVE PBLE SEND PREAMBLE
431: LDR A L110011100 14TH MESSAGE I.D.
432: LDR B L7
433: JCR SIBITS+BASE
434: LDR B L4
435: MSEN140 PCH B EMSEB+BASE
436: LDX STRINGS
437: PUL B
438: DEC B
439: SVE MSEN140
440: RTS

```

```

360: * BY COUNT IN ACCB
361: *****
362: SNBITS STX XTEMP      SAVE XREG
363: ROTAT  PSH A           GET OUT BIT
364:        PSH A           SAVE REST
365:        PSH B           SAVE COUNT
366:        BCC CON        CHANNELS ON IF '1'
367: COFF   CLR A           CHANNELS OFF IF '0'
368:        BRR NXBIT
369: CON    LDA A E20
370: NXBIT  BSR WAIT      WAIT FOR ELEMENT
371:        BSR SLOTS6
372:        PUL B
373:        PUL A           RESTORE STACK
374:        DEC B
375: BNE   ROTAT
376: LDY   XTEMP          RESTORE XREG
377: RTS
378: *****
379: * SET UP STEP LENGTHS IN SLAVE RAM
380: *****
381: CSET   LDY ECHNTBL+LATCH
382:        LDA B E36      FIRST STEP
383:        STA B STEP
384:        LDA B E4       CHANNEL COUNTER
385: CSET1  PSH B
386:        LDA B STEP
387:        PSH A           GET DATA BIT
388:        BCC CSET2     '0' ?
389:        STA B N        NO CHANNEL NEEDED
390:        CLR 4*X
391:        BRR CSET3
392: CSET2  CLR X          NOT NEEDED, SO
393:        CLR 4*X        CLEAR SLOTS
394: CSET3  SUB B E8      NEXT STEP LENGTH
395:        STA B STEP
396:        INX
397:        PUL B
398:        DEC B
399: BNE   CSET1
400: RTS
401: *****
402: * SET UP A SINGLE CHANNEL
403: *****
404: SLOTS  LDY ECHNTBL+LATCH
405:        LDA B E4
406: SLOTS6 STA A X
407:        CLR 4*X
408:        INX
409:        DEC B
410: BNE   SLOTS6
411: RTS
412: *****
413: * SUBR TO CONVERT BINARY PHASE INDICATOR TO
414: * LOOKUP START INDEX
415: *****
416: SINLKP TST A
417:        BNE TST1      00?
418:        LDA A E7      YES: 00->07
419:        RTS

```

```

0A03 81 01 420: TST1  CMP A E1
0A05 26 03 421: BNE TST2      01?
0A07 26 18 422: LDA A E27    YES: 01->27
0A09 39 423: RTS
0A0A 81 02 424: TST2  CMP A E2
0A0C 26 03 425: BNE TST3    10?
0A0E 26 43 426: LDA A E67    YES: 10->67
0A10 39 427: RTS
0A11 26 2F 428: TST3  LDA A E47    11->47
0A13 39 429: RTS
430: *****
431: * WAIT TIL SLAVE FINISHED TASKS:
432: *****
0A14 7D C001 433: WAIT  TST FLAG+LATCH
0A17 27 FB 434: BSR  WAIT
0A19 7F C001 435: CLR  FLAG+LATCH
0A1C 39 436: RTS
437: *****
438: * LOAD ALL CHANNELS WITH SPECIFIED PHASES:
439: *****
0A1D CE C05F 440: CLOAD  LDY ECHNTBL+LATCH+3
0A1E A7 00 441: CLOAD1 STA A X
0A20 03 442: INX
0A23 3C C057 443: CPX  ECHNTBL+LATCH+16
0A26 26 FB 444: BNE  CLOAD1
0A28 39 445: RTS
446: *
447: *****
448: * BOOTSTRAP LOADER FOR SLAVE PROCESSOR ROUTINES:
449: *****
0A29 CE F850 450: BOOT  LDY E$LVRO3+BASE
0A2C DF 05 451: STX  Y      BASE OF SOURCE
0A2E CE F922 452: LDY  E$LVRO3-1+BASE
0A31 DF 07 453: STX  Z      TOP OF SOURCE
0A33 CE C107 454: LDY  E$LDRO6+LATCH BASE OF DESTINATION
0A36 3D 1F 455: BSR  BOOT10  BOOT ROUTINES IN.
0A39 CE F923 456: LDY  E$LVRO3+BASE
0A3C DF 05 457: STX  Y      BASE OF SOURCE
0A3E CE F92A 458: LDY  E$LVRO7+BASE
0A41 DF 07 459: STX  Z      TOP OF SOURCE
0A43 CE C3F8 460: LDY  E$03A8+LATCH BASE OF DESTINATION
0A46 3D 10 461: BSR  BOOT10  BOOT VECTORS IN.
0A49 CE F800 462: LDY  E$LVRO3+BASE
0A4C DF 05 463: STX  Y      BASE OF SOURCE
0A4E CE F84F 464: LDY  E$LVRO3+79+BASE
0A51 DF 07 465: STX  Z      TOP OF SOURCE
0A53 CE C007 466: LDY  E$KUP1+LATCH BASE OF DESTINATION
0A56 3D 01 467: BSR  BOOT10  BOOT LOOKUP TABLES IN.
0A59 39 468: RTS
469: *****
470: * BOOT CONTENTS OF ADDRESSES Y->Z INTO RAM AREA
471: * BEGINNING WITH XREG.
472: *****
0A5B 09 473: BOOT10 STX  P      SAVE TARGET BASE
0A5D 7C 0008 474: INC  Z+1
0A5F CE 05 475: BOOT11 LDY  Y
0A62 3C 07 476: CPX  Z
0A65 27 0E 477: BSR  BOOT12
0A68 46 00 478: LDA  A X      SET 1 BYTE
0A6B 03 479: INX

```

TRANSMITTER SOFTWARE

```

0825 DE 05 430: STX Y
0826 DE 08 431: LDX P
0827 DE 08 432: STA A X
0828 DE 00 433: INX
0829 DE 08 434: STX P
0830 DE 00 435: SRA R00T11
0831 DE 00 436: R00T12 RTS
437: * SEQUENCE 1:
438: * ASCII CHARACTER STRING ONLY
439: *
440: *
441: *
442: *
443: *
444: *
445: *
446: *
447: *
448: *
449: *
450: *
451: *
452: *
453: *
454: *
455: *
456: *
457: *
458: *
459: *
460: *
461: *
462: *
463: *
464: *
465: *
466: *
467: *
468: *
469: *
470: *
471: *
472: *
473: *
474: *
475: *
476: *
477: *
478: *
479: *
480: *
481: *
482: *
483: *
484: *
485: *
486: *
487: *
488: *
489: *
490: *
491: *
492: *
493: *
494: *
495: *
496: *
497: *
498: *
499: *
500: *
501: *
502: *
503: *
504: *
505: *
506: *
507: *
508: *
509: *
510: *
511: *
512: *
513: *
514: *
515: *
516: *
517: *
518: *
519: *
520: *
521: *
522: *
523: *
524: *
525: *
526: *
527: *
528: *
529: *
530: *
531: *
532: *
533: *
534: *
535: *
536: *
537: *
538: *
539: *
540: *
541: *
542: *
543: *
544: *
545: *
546: *
547: *
548: *
549: *
550: *
551: *
552: *
553: *
554: *
555: *
556: *
557: *
558: *
559: *
560: *
561: *
562: *
563: *
564: *
565: *
566: *
567: *
568: *
569: *
570: *
571: *
572: *
573: *
574: *
575: *
576: *
577: *
578: *
579: *
580: *
581: *
582: *
583: *
584: *
585: *
586: *
587: *
588: *
589: *
590: *
591: *
592: *
593: *
594: *
595: *
596: *
597: *
598: *
599: *
600: *
601: *
602: *
603: *
604: *
605: *
606: *
607: *
608: *
609: *
610: *
611: *
612: *
613: *
614: *
615: *
616: *
617: *
618: *
619: *
620: *
621: *
622: *
623: *
624: *
625: *
626: *
627: *
628: *
629: *
630: *
631: *
632: *
633: *
634: *
635: *
636: *
637: *
638: *
639: *
640: *
641: *
642: *
643: *
644: *
645: *
646: *
647: *
648: *
649: *
650: *
651: *
652: *
653: *
654: *
655: *
656: *
657: *
658: *
659: *
660: *
661: *
662: *
663: *
664: *
665: *
666: *
667: *
668: *
669: *
670: *
671: *
672: *
673: *
674: *
675: *
676: *
677: *
678: *
679: *
680: *
681: *
682: *
683: *
684: *
685: *
686: *
687: *
688: *
689: *
690: *
691: *
692: *
693: *
694: *
695: *
696: *
697: *
698: *
699: *
700: *
701: *
702: *
703: *
704: *
705: *
706: *
707: *
708: *
709: *
710: *
711: *
712: *
713: *
714: *
715: *
716: *
717: *
718: *
719: *
720: *
721: *
722: *
723: *
724: *
725: *
726: *
727: *
728: *
729: *
730: *
731: *
732: *
733: *
734: *
735: *
736: *
737: *
738: *
739: *
740: *
741: *
742: *
743: *
744: *
745: *
746: *
747: *
748: *
749: *
750: *
751: *
752: *
753: *
754: *
755: *
756: *
757: *
758: *
759: *
760: *
761: *
762: *
763: *
764: *
765: *
766: *
767: *
768: *
769: *
770: *
771: *
772: *
773: *
774: *
775: *
776: *
777: *
778: *
779: *
780: *
781: *
782: *
783: *
784: *
785: *
786: *
787: *
788: *
789: *
790: *
791: *
792: *
793: *
794: *
795: *
796: *
797: *
798: *
799: *
800: *
801: *
802: *
803: *
804: *
805: *
806: *
807: *
808: *
809: *
810: *
811: *
812: *
813: *
814: *
815: *
816: *
817: *
818: *
819: *
820: *
821: *
822: *
823: *
824: *
825: *
826: *
827: *
828: *
829: *
830: *
831: *
832: *
833: *
834: *
835: *
836: *
837: *
838: *
839: *
840: *
841: *
842: *
843: *
844: *
845: *
846: *
847: *
848: *
849: *
850: *
851: *
852: *
853: *
854: *
855: *
856: *
857: *
858: *
859: *
860: *
861: *
862: *
863: *
864: *
865: *
866: *
867: *
868: *
869: *
870: *
871: *
872: *
873: *
874: *
875: *
876: *
877: *
878: *
879: *
880: *
881: *
882: *
883: *
884: *
885: *
886: *
887: *
888: *
889: *
890: *
891: *
892: *
893: *
894: *
895: *
896: *
897: *
898: *
899: *
900: *
901: *
902: *
903: *
904: *
905: *
906: *
907: *
908: *
909: *
910: *
911: *
912: *
913: *
914: *
915: *
916: *
917: *
918: *
919: *
920: *
921: *
922: *
923: *
924: *
925: *
926: *
927: *
928: *
929: *
930: *
931: *
932: *
933: *
934: *
935: *
936: *
937: *
938: *
939: *
940: *
941: *
942: *
943: *
944: *
945: *
946: *
947: *
948: *
949: *
950: *
951: *
952: *
953: *
954: *
955: *
956: *
957: *
958: *
959: *
960: *
961: *
962: *
963: *
964: *
965: *
966: *
967: *
968: *
969: *
970: *
971: *
972: *
973: *
974: *
975: *
976: *
977: *
978: *
979: *
980: *
981: *
982: *
983: *
984: *
985: *
986: *
987: *
988: *
989: *
990: *
991: *
992: *
993: *
994: *
995: *
996: *
997: *
998: *
999: *

```

NO ERROR(S) DETECTED

```

1:      NAM  RECEIVER SOFTWARE
2:      OPT  NOS,LIS,PAG
3:
4:  * SYNCHRONISATION, DE-INTERLEAVING, AND DECODING
5:  * ROUTINES.
6:
7:  * ANALYSIS IS PRINTED ON TERMINAL AS FOLLOWS:
8:  * TIME OF DAY & DATA TYPE (UNCODED, CODED, INTERLEAVED)
9:  * NO. OF ERRORS IN DECODED DATA STREAM
10: * BURST ERROR DISTRIBUTION
11: * IF DATA IS CODED, THEN:
12: * NO. OF ERRORS IN RECEIVED BIT STREAM
13: * TOTAL NO. OF CORRECTED ERRORS IN RECEIVED BIT STREAM
14:
15: *****
16:
17: * INTERRUPT VECTOR:
18:
19:      ORG  $F000
20:      FDS  I$0ARD
21:
22: * MONITOR LINKAGES:
23:
24: OUTCH EQU  $E101
25: PDATA EQU  $E07E
26: OUT4MS EQU  $E0C3
27: OUT2M EQU  $E09E
28: INCH  EQU  $E07B
29: INHEN EQU  $E0B9
30:
31: * DISK FILE MANAGEMENT LINKAGES:
32:
33: ODFM EQU  $77B0  OPEN DFM
34: CDFM EQU  $77B3  CLOSE DFM
35: DFM  EQU  $77B6
36: DWARM EQU  $72B3  WARM START
37: DTYPE EQU  $7E49  TYPE DISK ERROR
38:
39: * VARIABLE STORAGE AREA:
40:
41:      ORG  0
42: HOURS RMB 1      REAL-TIME CLOCK PARAMETERS
43: MIN  RMB 1
44: SECS RMB 1
45: DAY  RMB 1
46: COUNT RMB 2     SECONDS COUNTER
47: XRES  RMB 2     XRES TEMPORARY STORAGE
48: XTEMP1 RMB 2
49: XTEMP2 RMB 2
50: XTEMP3 RMB 2
51: XTEMP4 RMB 2
52: XTEMP5 RMB 2
53: XTEMP6 RMB 2
54: XTEMP7 RMB 2
55: XTEMP8 RMB 2
56: XTEMP9 RMB 2
57: XTEMP0 RMB 2
58:
59: * I/O ADDRESSES

```

```

F532 60: *
F530 61: PIA  EQU  $F532
62: PIAT EQU  $F530
63:
64: * BEGIN MAIN PROCEDURE:
65: *****
66:      ORG  $0100
67:      LDX  $0004  INITIALISE PIAD
68:      STX  PIA    RECEIVED DATA PORT
69:      INX
70:      STX  PIAT   TIMER INTERRUPT PORT
71:      BEI
72:      JSR  ODFM   OPEN DFM
73:      LDX  EFCB0
74:      JSR  ODFNFI OPEN FILE ON DRIVE 0
75:      LDA  A  E1
76:      STA  A  FCB1+2
77:      LDX  EFCB1
78:      JSR  ODFNFI OPEN FILE ON DRIVE 1
79:      JSR  FETTIM AND REQUEST TIME FROM TERMINAL
80:      CLI    ENABLE 1HZ. CLOCK
81:      BSR  INTLSE INITIALISE VARIABLES
82:      CMP  A  IDAT1 1ST MESSAGE ?
83:      BNE  BEGIN1 NOT START AGAIN
84:      JSR  MSGE1 YES: READ IT
85:      JSR  STDATA AND ANALYSE IT
86:      BSR  INTLSE DO SAME FOR 2ND MESSAGE
87:      CMP  A  IDAT2
88:      BNE  BEGIN2
89:      JSR  MSGE2
90:      JSR  STDATA
91:      BSR  INTLSE 3RD MESSAGE SEARCH
92:      CMP  A  IDAT3
93:      BNE  BEGIN3
94:      JSR  MSGE3
95:      JSR  STDATA
96:      BRR  BEGIN1  START ALL OVER AGAIN
97:
98: * INITIALISE VARIABLES ETC. :
99: *****
100: INTLSE LDA  A  E3FE  STOP BOTH SLAVES
101:      STA  A  LATCH
102:      STA  A  LATCH1
103:      JSR  ELOAD  LOAD TIMER SLAVE
104:      STA  A  FLAG  SET FLAG IN TIMER SLAVE
105:      LDX  E0
106:      STX  ERRCNT  ZERO THE ERROR COUNT
107:      STX  MERR   ZERO MESSAGE ERROR COUNT
108:      STX  ERRCOR+LATCH ZERO NO. OF CORRECTED ERRORS
109:      LDX  ERRBUF  INITIALISE ERROR STATS BUFFER
110:      STX  STPTR  POINTER
111:      CLR  X
112:      CLR  1,X
113:      LDA  A  E3FA  'END OF TABLE' FLAG
114:      STA  A  2,X
115:      CLR  PREV   NO ERRORS ON PREVIOUS
116:      JSR  SYNC  LOOK FOR SYNC PATTERN
117:      JSR  IDPAT GET I.D. NUMBER
118:      RTS
119: *****

```



```

0295 80 0300 240: JSR WDISCO
0299 80 0320 241: JSR WDISC1
029E 86 02 242: LDA A SECS
029E 80 0300 243: JSR WDISCO
02A1 80 0329 244: JSR WDISC1
02A4 89 245: RTS
245: *****
247: * WRITE ERROR PATTERN DATA TO DISC
248: *****
02A5 0E 0807 249: ERRPAT LDX IERRBUF
02A8 0F 06 250: STX KTEMP
02AA 86 AA 251: ERST0 LDA A IERR 'GOOD BITS' MARKER
02AC 80 06 252: BSR ERST10
02AE 86 B8 253: LDA A IERR 'BAD BITS' MARKER
02B0 80 02 254: BSR ERST10
02B3 80 56 255: SRA ERST0
255: *
02B4 0E 06 257: ERST10 LDX KTEMP
02B6 80 00 258: TST X
02B8 28 10 259: SMI ERST3 OUT IF TABLE END
02BA 80 0829 260: JSR WDISC1 WRITE MARKER
02BD 80 00 261: TST X
02BF 27 05 262: BEQ ERST2
02C1 86 00 263: LDA A X WRITE MSB IF NEEDED
02C3 80 0829 264: JSR WDISC1
02C6 0E 06 265: ERST2 LDX KTEMP
02C8 83 266: INX
02CA 0F 04 267: STX KTEMP
02CB 86 00 268: LDA A X WRITE LSB
02CD 80 0829 269: JSR WDISC1
02D0 0E 06 270: LDX KTEMP
02D2 83 271: INX
02D4 0F 06 272: STX KTEMP
02D6 89 273: RTS
02D8 81 274: ERST3 INX
02DA 81 275: INX
02DC 89 276: RTS
276: *****
278: * OPEN FILE FOR WRITE
279: *****
02D9 0F 09 280: OPNFIL STX KTEMP1
02DB 86 01 281: LDA A E1
02DD 87 00 282: STA A X
02DF 86 03 283: LDA A E2 FILE TYPE
02E1 87 00 284: STA A I2+X (BINARY SEQUENTIAL)
02E3 80 7786 285: JSR DFM
02E5 27 09 286: BEQ FILOPN
02E7 80 72A9 287: JSR ZTYPDE
02E8 80 7783 288: JSR CDFM
02EA 80 7383 289: JSR ZWARMS
02F1 86 03 290: FILOPN LDA A E2
02F3 0E 03 291: LDX KTEMP1
02F5 87 00 292: STA A X
02F7 89 293: RTS
293: *****
295: * CLOSE FILE FOR WRITE:
296: *****
02F9 86 03 297: CLIFIL LDA A E3
02FB 87 00 298: STA A X
02FD 80 7016 299: JSR DFM

```

```

02FF 27 09 300: BEQ CLSOK
0301 80 72A9 301: JSR ZTYPDE
0304 80 7783 302: JSR CDFM
0307 7E 7283 303: JMP ZWARMS
030A 0E 304: CLSOK CLI
030B 89 305: RTS
305: *****
307: * WRITE TO DRIVE 0
308: *****
030C 86 309: WDISCO PSH A
030D 87 310: PSH B
030E 0F 14 311: STX KTEMP2
0310 0E 0A97 312: LDX EFCB0
0313 80 7786 313: JSR DFM
0315 27 0B 314: BEQ WRITOK
0318 80 72A9 315: JSR ZTYPDE
031B 80 7783 316: JSR CDFM
031E 7E 7283 317: JMP ZWARMS
0321 0E 14 318: LDX KTEMP2
0323 83 319: WRITOK PUL B
0324 83 320: PUL A
0325 0E 14 321: LDX KTEMP2
0327 0E 322: CLI
0328 89 323: RTS
323: *****
325: * WRITE TO UNITS 1 OR 2
326: *****
0329 86 327: WDISC1 PSH A
032A 87 328: PSH B
032B 0F 14 329: STX KTEMP2
032D 0E 0B2F 330: LDX EFCB1
0330 80 7786 331: JSR DFM TRY TO WRITE BYTE
0333 27 28 332: BEQ WROK
0335 86 0B30 333: LDA A FCB1+1 GET ERROR STATUS
0338 81 07 334: CMP A E7 DISC FULL ?
033A 27 09 335: BEQ DISFUL YES
033C 80 72A9 336: DERR JSR ZTYPDE NOT MUST BE ANOTHER ERROR
033F 80 7783 337: JSR CDFM
0342 80 7283 338: JSR ZWARMS
0345 86 0B31 339: DISFUL LDA A FCB1+2 ON DRIVE 2 ?
0348 81 02 340: CMP A E2
034A 27 11 341: BEQ WROK BOTH FULL, DON'T WRITE
034C 0E 0B2F 342: LDX EFCB1
034F 80 0BFB 343: JSR CLSFIL CLOSE FILE ON UNIT 1
0352 86 02 344: LDA A E2 NOT CHANGE UNIT 1
0354 87 0B31 345: STA A FCB1+2
0357 0E 0B2F 346: LDX EFCB1
035A 80 02D9 347: JSR OPNFIL AND OPEN FILE ON UNIT 2
035D 0E 348: WROK CLI
035E 0E 14 349: LDX KTEMP2
0360 83 350: PUL B
0361 83 351: PUL A
0362 89 352: RTS
352: *****
354: * FIRST MESSAGE:
355: *****
0363 0E 0382 356: MSGE1 LDX EMSG1
0365 0F 00 357: STX ERRPTR
0368 0E 0A06 358: LDX IERR
036B 80 0794 359: MSGE11 JSR DFM

```

0385 97 01 340: STR A X
0375 85 0035 341: XOR LBUF+127
0370 73 03 342: BRD MSSE12
0365 63 03 343: INV
0360 50 03 344: BRD MSSE11
0355 40 04 345: MSSE12 LDR A L1
0350 30 03 346: LDR A X
0345 20 01 347: LDR A L1
0340 10 11 348: STR A MESSUM
0335 00 00 349: MESSUM MESSAGE IDENTIFIER
0330 00 00 350: STR A X
0325 00 00 351: MESSI
0320 00 00 352: LDR A L1
0315 00 00 353: LDR A L1
0310 00 00 354: LDR A L1
0305 00 00 355: LDR A L1
0300 00 00 356: LDR A L1
0295 00 00 357: LDR A L1
0290 00 00 358: LDR A L1
0285 00 00 359: LDR A L1
0280 00 00 360: LDR A L1
0275 00 00 361: LDR A L1
0270 00 00 362: LDR A L1
0265 00 00 363: LDR A L1
0260 00 00 364: LDR A L1
0255 00 00 365: LDR A L1
0250 00 00 366: LDR A L1
0245 00 00 367: LDR A L1
0240 00 00 368: LDR A L1
0235 00 00 369: LDR A L1
0230 00 00 370: LDR A L1
0225 00 00 371: LDR A L1
0220 00 00 372: LDR A L1
0215 00 00 373: LDR A L1
0210 00 00 374: LDR A L1
0205 00 00 375: LDR A L1
0200 00 00 376: LDR A L1
0195 00 00 377: LDR A L1
0190 00 00 378: LDR A L1
0185 00 00 379: LDR A L1
0180 00 00 380: LDR A L1
0175 00 00 381: LDR A L1
0170 00 00 382: LDR A L1
0165 00 00 383: LDR A L1
0160 00 00 384: LDR A L1
0155 00 00 385: LDR A L1
0150 00 00 386: LDR A L1
0145 00 00 387: LDR A L1
0140 00 00 388: LDR A L1
0135 00 00 389: LDR A L1
0130 00 00 390: LDR A L1
0125 00 00 391: LDR A L1
0120 00 00 392: LDR A L1
0115 00 00 393: LDR A L1
0110 00 00 394: LDR A L1
0105 00 00 395: LDR A L1
0100 00 00 396: LDR A L1
0095 00 00 397: LDR A L1
0090 00 00 398: LDR A L1
0085 00 00 399: LDR A L1
0080 00 00 400: LDR A L1
0075 00 00 401: LDR A L1
0070 00 00 402: LDR A L1
0065 00 00 403: LDR A L1
0060 00 00 404: LDR A L1
0055 00 00 405: LDR A L1
0050 00 00 406: LDR A L1
0045 00 00 407: LDR A L1
0040 00 00 408: LDR A L1
0035 00 00 409: LDR A L1
0030 00 00 410: LDR A L1
0025 00 00 411: LDR A L1
0020 00 00 412: LDR A L1
0015 00 00 413: LDR A L1
0010 00 00 414: LDR A L1
0005 00 00 415: LDR A L1
0000 00 00 416: LDR A L1
0000 00 00 417: LDR A L1
0000 00 00 418: LDR A L1
0000 00 00 419: LDR A L1
0000 00 00 420: LDR A L1
0000 00 00 421: LDR A L1
0000 00 00 422: LDR A L1
0000 00 00 423: LDR A L1
0000 00 00 424: LDR A L1
0000 00 00 425: LDR A L1
0000 00 00 426: LDR A L1
0000 00 00 427: LDR A L1
0000 00 00 428: LDR A L1
0000 00 00 429: LDR A L1
0000 00 00 430: LDR A L1
0000 00 00 431: LDR A L1
0000 00 00 432: LDR A L1
0000 00 00 433: LDR A L1
0000 00 00 434: LDR A L1
0000 00 00 435: LDR A L1
0000 00 00 436: LDR A L1
0000 00 00 437: LDR A L1
0000 00 00 438: LDR A L1
0000 00 00 439: LDR A L1
0000 00 00 440: LDR A L1
0000 00 00 441: LDR A L1
0000 00 00 442: LDR A L1
0000 00 00 443: LDR A L1
0000 00 00 444: LDR A L1
0000 00 00 445: LDR A L1
0000 00 00 446: LDR A L1
0000 00 00 447: LDR A L1
0000 00 00 448: LDR A L1
0000 00 00 449: LDR A L1
0000 00 00 450: LDR A L1
0000 00 00 451: LDR A L1
0000 00 00 452: LDR A L1
0000 00 00 453: LDR A L1
0000 00 00 454: LDR A L1
0000 00 00 455: LDR A L1
0000 00 00 456: LDR A L1
0000 00 00 457: LDR A L1
0000 00 00 458: LDR A L1
0000 00 00 459: LDR A L1
0000 00 00 460: LDR A L1
0000 00 00 461: LDR A L1
0000 00 00 462: LDR A L1
0000 00 00 463: LDR A L1
0000 00 00 464: LDR A L1
0000 00 00 465: LDR A L1
0000 00 00 466: LDR A L1
0000 00 00 467: LDR A L1
0000 00 00 468: LDR A L1
0000 00 00 469: LDR A L1
0000 00 00 470: LDR A L1
0000 00 00 471: LDR A L1
0000 00 00 472: LDR A L1
0000 00 00 473: LDR A L1
0000 00 00 474: LDR A L1
0000 00 00 475: LDR A L1
0000 00 00 476: LDR A L1
0000 00 00 477: LDR A L1
0000 00 00 478: LDR A L1
0000 00 00 479: LDR A L1
0000 00 00 480: LDR A L1

0443 CE 0406 420: LDX LBUF
0446 DF 08 421: STX XTEMP1
0448 86 FF 422: LDR A XTEMP
0449 B7 C001 423: STR A TEMPLG+LATCH
044D 7F C003 424: CLR TEMPLG+LATCH
0450 7F C002 425: CLR TEMPLG+LATCH
0453 CE C07C 426: LDX TBCDD0+LATCH
0455 BD 030C 427: JDR MSSE20
0458 8F C07C 428: STR A LATCH
0459 8F C07C 429: LDR A LATCH
0462 CE C07C 430: LDX TBCDD1+LATCH
0464 8F C07C 431: JDR MSSE20
046A 05 03 432: LDR A L1
046B 37 433: MSSE22
046C 0F 434: SET
046D 0F 435: JDR MSSE22
046E 87 C003 436: STR A TEMPLG+LATCH
046F 87 C002 437: STR A LATCH
0471 CE C07C 438: LDX TBCDD0+LATCH
0474 0F 06 439: STX XTEMP
0475 8D 0532 440: JDR MSSE24
0479 CE 04BD 441: LDX XTEMP
047C DF 0D 442: STX ERPTLR
047E CE C05C 443: LDX TBCDD0+LATCH
0481 BD 050C 444: JDR MSSE20
0484 0F 06 445: SET
0485 3D 75 446: BRD MSSE23
0487 7F C003 447: CLR TEMPLG+LATCH
0489 87 C000 448: STR A LATCH
048D CE C07C 449: LDX TBCDD1+LATCH
0490 DF 06 450: STX XTEMP
0492 BD 0532 451: JDR MSSE24
0495 CE C07C 452: LDX TBCDD1+LATCH
0498 3D 75 453: BRD MSSE20
049A 33 454: STR B
049C 36 08 455: DEC B
049E 56 456: BRD MSSE22
04A0 87 C003 457: STR A TEMPLG+LATCH
04A2 87 C000 458: STR A LATCH
04A6 CE C05C 459: LDX TBCDD0+LATCH
04A8 0E 06 460: STX XTEMP
04A9 BD 0532 461: JDR MSSE24
04AB 8D 053C 462: LDX TBCDD1+LATCH
04AD DF 06 463: STX XTEMP
04B3 3D 74 464: BRD MSSE24
04B5 3D 02 465: LDR A L1
04B9 37 11 466: STR A MESSUM
048C 32 467: MSSE
048D 54 470: MESSI
04C7 49 471: FCB \$44, \$34, \$38, \$F5, \$A5, \$D5, \$51, \$03, \$55, \$55, \$55
04D1 48 472: FCB \$49, \$24, \$43, \$48, \$48, \$57, \$42, \$39, \$55, \$41
04D8 58 473: FCB \$44, \$8B, \$57, \$49, \$4E, \$60, \$45, \$17, \$3E, \$58
0455 53 474: FCB \$53, \$0F, \$48, \$86, \$55, \$55, \$49, \$E, \$50, \$52
0457 54 475: FCB \$53, \$70, \$4E, \$81, \$5C, \$47, \$45, \$50, \$50, \$50
0459 54 476: FCB \$54, \$34, \$48, \$55, \$45, \$3E, \$40, \$45, \$41, \$45
0459 54 477: FCB \$54, \$B0, \$59, \$4E
478: *
479: *
480: *
481: *
482: *
483: *
484: *
485: *
486: *
487: *
488: *
489: *
490: *
491: *
492: *
493: *
494: *
495: *
496: *
497: *
498: *
499: *
500: *
501: *
502: *
503: *
504: *
505: *
506: *
507: *
508: *
509: *
510: *
511: *
512: *
513: *
514: *
515: *
516: *
517: *
518: *
519: *
520: *

```

04FD 00 0001 430: MSGE23 TST STFLAG+LATCH
0500 06 08 431: BNE MSGE23
0502 06 0E 432: LDA R 18FF
0504 07 0000 433: STA A LATCH
0507 07 434: INC A
0509 07 0001 435: STA A STFLAG+LATCH
050B 08 436: RTS
437: *****
438: * GET 16 CODEWORDS INTO BUFFER POINTED AT
439: * BY X REGISTER:
440: *****
050D 08 10 441: MSGE20 LDA B 116 CODEWORD COUNTER
050E 08 442: MSGE20 PSH B
050F 08 443: CLR A
0510 08 07 444: LDA B 17 INFORMATION BIT COUNT
0512 0D 13 445: BIR MSGE21
0514 08 07 446: LDA B 17
0516 0D 0875 447: JIR ERROR FIND IF ERRORS IN INFOBITS
0518 08 08 448: LDA B 18 CHECKBIT COUNT
051B 0D 04 449: BIR MSGE21
051D 08 08 500: LDA B 18
051F 0D 0875 501: JIR ERROR FIND IF ERRORS IN CHECKBITS
0521 08 502: PUL B
0523 08 503: DEC B
0524 06 08 504: BNE MSGE29
0526 08 505: RTS
506: *****
507: * GET BITS INTO ACCA. (NO. OF BITS IN ACCB)
508: * STORE ACCA AT X+ BUMP X
509: *****
0527 0D 0788 510: MSGE21 JIR BIT GET A BIT
0528 08 511: PUL A INTO ACCA
0529 08 512: DEC B BIT COUNT
052C 06 09 513: BNE MSGE21
052E 07 00 514: STA A X STORE BYTE
0530 08 515: INX
0531 08 516: RTS
517: *****
518: * TRANSFER BYTES BETWEEN BUFFERS
519: *****
0532 06 10 520: MSGE24 LDA B 116
0534 0E 08 521: MSGE26 LDX XTEMP FIRST POINTER
0536 06 00 522: LDA A X 'FROM' ADDRESS
0538 08 523: INX
0539 0F 08 524: STX XTEMP
053B 0E 08 525: LDX XTEMP1 2ND POINTER
053D 07 00 526: STA A X 'TO' ADDRESS
053F 08 527: INX
0540 0F 08 528: STX XTEMP1
0542 08 529: DEC B
0543 06 0F 530: BNE MSGE26
0545 08 531: RTS
532: *****
533: * THIRD MESSAGE: CODED AND INTERLEAVED TO DEPTH 16:
534: *****
0546 0E 0504 535: MSGE3 LDX 1ME13
0548 0F 00 536: STX ERRPTR
054B 0E 0406 537: LDX 1BUFF
054E 0F 08 538: STX XTEMP1
0550 06 0F 539: LDA R 18FF

```

```

0552 07 0001 540: STA A STFLAG+LATCH SET SLAVE STATUS FLAG
0555 07 0002 541: STA A INFLAG+LATCH INDICATE INTERLEAVING REQUIRED
0558 07 0003 542: CLR STFLAG+LATCH INDICATE TABLE 0
055B 0E 0020 543: LDX 1TBINT0+LATCH
055E 0D 0400 544: JOP MSGE30 FILL TABLE 0 WITH DATA
0561 06 0F 545: LDA R 18FF
0563 07 0000 546: STA A LATCH TELL SLAVE TO DECODE IT
0566 0E 003E 547: LDX 1TBINT1+LATCH
0569 0D 0400 548: JOP MSGE30 FILL TABLE 1 WITH DATA
056C 06 08 549: LDA B 18 GROUP COUNTER
056E 07 550: MSGE32 PSH B
056F 0F 551: SEI
0570 0D 08 552: BDR MSGE23 CHECK IF SLAVE FINISHED DC.
0572 07 0003 553: STA A TBFLAG+LATCH INDICATE TABLE 1 TO SLAVE
0575 07 0000 554: STA A LATCH THEN RESTART SLAVE
0578 0E 0050 555: LDX 1TBODD0+LATCH
057B 0F 08 556: STX XTEMP
057D 0E 0504 557: LDX 1MES3
0580 0F 00 558: STX ERRPTR RE-INITIALISE POINTER
0582 0D 0E 559: BOP MSGE24 EXTRACT DATA FROM TABLE 0
0584 0E 0020 560: LDX 1TBINT0+LATCH
0587 0D 07 561: BDR MSGE30 GET 30 BYTES OF DATA
0589 0F 562: SEI
058A 0D 04FD 563: JDR MSGE23 CHECK SLAVE STATUS DC.
058D 07 0003 564: CLR TBFLAG+LATCH INDICATE TABLE 0 TO SLAVE
0590 07 0000 565: STA A LATCH RESTART SLAVE
0593 0E 0070 566: LDX 1TBODD1+LATCH
0596 0F 08 567: STX XTEMP
0598 0D 08 568: BDR MSGE24 EXTRACT DATA FROM TABLE 1
059A 0E 003E 569: LDX 1TBINT1+LATCH
059D 0D 01 570: BDR MSGE30 GET 30 BYTES OF DATA
059F 03 571: PUL B
05A0 08 572: DEC B
05A1 06 08 573: BNE MSGE32
05A3 0D 04FD 574: JDR MSGE23 CHECK SLAVE STATUS
05A6 07 0003 575: STA A TBFLAG+LATCH
05A9 07 0000 576: STA A LATCH
05AC 0E 0050 577: LDX 1TBODD0+LATCH
05AF 0F 08 578: STX XTEMP
05B1 0D 0532 579: JDR MSGE24 GET OUT DATA
05B4 0D 04FD 580: JDR MSGE23
05B7 0E 0070 581: LDX 1TBODD1+LATCH
05BA 0F 08 582: STX XTEMP
05BC 0D 0532 583: JDR MSGE24
05BF 06 03 584: LDA R 18
05C1 07 11 585: STA A MESNUM MESSAGE I.D. NUMBER
05C3 08 586: RTS
05C4 0F 587: MES3 PCB 1FF, 1FE, 100, 100, 198, 151, 145, 128, 149, 13E
05C6 08 588: PCB 103, 1FE, 13F, 132, 169, 1F3, 16E, 10D, 10C, 16E, 1E1
05C9 08 589: PCB 185, 122, 185, 1ED, 115, 133, 191, 179, 1E7, 1FF, 1FF
05CB 00 590: PCB 100, 100, 15A, 103, 1A4, 12B, 167, 159, 13E, 132, 16D
05CE 15 591: PCB 115, 1F5, 1B2, 17B, 130, 14C, 1E6, 11D, 17F, 101, 117
05FA 03 592: PCB 103, 173, 1B3, 119, 147, 18C
593: *****
594: * GET 30 BYTES OF INTERLEAVED CODEWORDS:
595: *****
0600 06 1E 596: MSGE30 LDA B 130
0602 07 597: MSGE31 PSH B
0603 08 598: CLR A
0604 06 08 599: LDA B 18

```

```

0606 80 0537 600: JSR MSGE21
0609 16 08 601: LDA B E8
060B 80 0375 602: JSR ERROR FIND ERRORS, IF ANY
060E 33 603: PUL B
060F 5A 604: DEC B
0610 26 F0 605: SNE B MSGE31
0612 39 606: RTS
607: *****
608: * MESSAGE AREA:
609: *****
0613 41 610: STRN51 FOC 'ANALYSIS AT TIME '
0614 04 611: FCB 4
0615 80 612: STRN52 FOC ' OF DATA WITHOUT CODING'
0617 04 613: FCB 4
0618 80 614: STRN53 FOC ' OF DATA WITH BCH CODING'
0619 04 615: FCB 4
0620 80 616: STRN54 FOC ' OF DATA WITH BCH CODING & INTERLEAVING'
0621 04 617: FCB 4
0622 45 618: STRN55 FOC 'ERROR COUNT IN DATA STREAM = '
0623 04 619: FCB 4
0624 4E 620: STRN57 FOC 'NO. OF ERRORS IN TOTAL BIT STREAM = '
0625 04 621: FCB 4
0626 4E 622: STRN58 FOC 'NO. OF ERRORS CORRECTED = '
0627 04 623: FCB 4
0628 52 624: STRN59 FOC 'READY....'
0629 04 625: FCB 4
062A 2A 626: STRN5A FOC '*****'
627: *****
062B 04 627: FCB 4
062C 80 628: CRLF FCB 'D+3A.4 CR, L/F STRING'
629: *****
630: * SYNCHRONISE WITH START PATTERN:
631: *****
0632 06 08 632: SYNC LDA B E8 MUST GET 8 BITS
0633 37 633: PCH B
0634 80 4B 634: SYNCA BCR SYNCO SYNC ON BIT
0635 33 635: PUL B
0637 5A 636: DEC B
0638 26 FA 637: SNE SYNCA
0639 08 0530 638: LDX E1408 WAIT TIL CENTRE
OF BIT
063A 09 639: SYNCA BCR SYNCA
063B 26 FD 640: SNE SYNCA
063C 36 FF 641: LDA A E3FF START SLAVE
063D 87 0007 642: STA A FLAG
063E 87 0000 643: STA A LATCH1
644: *
645: * GET FRAME SYNC:
646: *
0647 06 80 647: LDA B E125 128 TRIES ALLOWED
0648 75 0019 648: CLR INPAT
0649 75 0019 649: CLR INPAT+1
064A 37 650: SYNCC PCH B GET A BIT
064B 80 78 651: BCR BIT
064C 79 0019 652: PUL INPAT+1
064D 79 0013 653: PUL INPAT
064E 35 13 654: LDA A INPAT
064F 16 13 655: LDA B INPAT+1
0650 83 0792 656: EOR A M5E0 CORRELATE WITH
STORED SEQUENCE
0651 78 0792 657: EOR B M5E0+1
0652 37 16 658: STA A TEMP
0653 07 17 659: STA B TEMP+1
    
```

```

0757 05 0F 660: LDA B E15
0759 4F 661: CLR A
075A 76 0016 662: SYNC PDR TEMP
075D 76 001E 663: RDR TEMP1
0760 25 01 664: BCC SYNCE
0762 40 665: INC A
0763 5A 666: SYNCE DEC B
0764 26 F4 667: BNE SYNCD
0766 81 03 668: CMP A E8
0768 2E 05 669: BGT SYNCF GOT SYNC ?
076A 33 670: PUL B NO; TRY AGAIN
076B 5A 671: DEC B
076C 26 D2 672: BNE SYNCD
076E 20 81 673: BRA SYNCD REDD BIT SYNC
0770 39 674: SYNCF RTS
675: *
676: * GET BIT SYNC:
677: *
0771 7D F532 678: SYNCO TST PIA '0' ?
0774 26 FB 679: BNE SYNCO
0776 7D F532 680: SYNCO TST PIA '1' ?
0779 27 FB 681: BEQ SYNCO
077B 0E 1686 682: SYNCD LDX E5B14
077E 09 683: SYNCD DEX
077F 26 FA 684: BNE SYNCD WAIT.
0781 7D F532 685: TST PIA SHOULD BE '0' HERE
0784 26 EB 686: SNE SYNCO TRY AGAIN IF NOT
0785 0E 0062 687: LDX E99
0788 09 688: SYNCD DEX
078A 26 FD 689: BNE SYNCD WAIT AGAIN
078C 7D F532 690: TST PIA SHOULD BE '1' HERE
078F 26 E0 691: BNE SYNCD TRY AGAIN IF NOT
0791 39 692: RTS
0792 9E B2 693: M5E0 FDB E9E32 SYNC PATTERN
694: *****
695: * GET A CHARACTER (SLAVE PROCESSOR TIMING):
696: *
0794 8D 25 697: CHAR BCR BIT GET A BIT
0796 25 05 698: BCC BITOK
0798 8D 089B 699: JSR BITERR ERROR IN START BIT
0799 20 03 700: BRA STRT
079D 8D 08DB 701: STTOK JSR NBITER NO ERROR IN START BIT
079A 4F 702: STRT CLR A
0791 06 07 703: LDA B E7 BIT COUNT
0793 8D 16 704: BCR BIT
0795 49 705: PUL A
0796 5A 706: DEC B
0797 26 FA 707: BNE DBIT
0799 06 07 708: LDA B E7
079E 8D 0375 709: JSR ERRDR CHECK FOR ERRORS
079E 8D 08 710: STOP0 BCR BIT
078D 24 05 711: BCC STOPOK
0782 8D 089B 712: JSR BITERR
0785 20 03 713: BRA STOP1 NO ERROR IN STOP BIT
0787 8D 08DB 714: STOPK JSR NBITER
078A 39 715: STOP1 RTS
716: *****
717: * GET A BIT:
718: *
078B 0F 719: BIT SET MASK INTERRUPT
    
```

```

0780 70 0000 720: TST FLAG
0781 28 40 721: BNE BIT1
0782 08 071E 722: LDX ECRLF
0783 8D E00E 723: JSR PDATA
0784 08 07E3 724: LDX LEARNES
0785 8D E07E 725: JSR PDATA
0786 08 0A72 726: LDS LRF072 RELOAD STACK POINTER
0787 06 11 727: LDA A MESNUM
0788 01 01 728: CMP A L1
0789 07 07 729: BEQ B1
0790 01 02 730: CMP A L2
0791 07 06 731: BEQ B2
0792 7E 0100 732: JMP BEGINS
0793 7E 0100 733: B1 JMP BEGINS1
0794 7E 0100 734: B2 JMP BEGINS2
0795 0A 035: ERRMES FCB ***** BIT TIMING ERROR - SLAVE RELOADED *****
0796 04 036: FCB 4
0797 70 0000 737: BIT1 TST FLAG SLAVE TIMER FLAG
0798 04 08 738: BNE BIT1
0799 74 0000 739: DEC FLAG RESET FLAG
0800 70 0500 740: TST PIR
0801 07 03 741: BEQ ZERO
0802 00 742: ONE DEC
0803 20 01 743: BRA NKBIT
0804 00 744: ZERO CLC
0805 0E 745: NKBIT CLI
0806 09 746: RTS
0807 747: *****
0808 748: * FIND MESSAGE IDENTITY CODE:
0809 749: *****
0810 7A 0015 750: IDPAT CLR TEMP USE TEMP FOR RECEIVED
0811 06 07 751: LDA B L7 DATA
0812 07 752: IDPATO PSH B SAVE BIT COUNTER
0813 00 03 753: BR BIT SET BIT
0814 79 0016 754: ROL TEMP DO 7 TIMES
0815 03 755: PUL B
0816 04 756: DEC B
0817 26 06 757: BNE IDPATO
0818 06 16 758: LDA A TEMP GET 7-BIT CODE
0819 03 0372 759: EOR A IDPT1 CORRELATE WITH 1ST
0820 00 00 760: BR IDCOR PATTERN
0821 07 17 761: STA A TEMP+1 STORE RESULT
0822 06 0372 762: LDA A IDPT1
0823 07 13 763: STA A SNPAT
0824 06 16 764: LDA A TEMP SET CODE
0825 03 0373 765: EOR A IDPT2 CORRELATE WITH 2ND
0826 00 1E 766: BR IDCOR
0827 01 17 767: CMP A TEMP+1 COMPARE WITH PREVIOUS
0828 07 07 768: BLE IDPAT1 IF LESS, IGNORE.
0829 07 17 769: STA A TEMP+1 IF GREATER, SAVE.
0830 06 0373 770: LDA A IDPT2
0831 07 13 771: STA A SNPAT
0832 06 16 772: IDPAT1 LDA A TEMP SET CODE
0833 03 0374 773: EOR A IDPT3 CORRELATE WITH 3RD
0834 00 00 774: BR IDCOR
0835 01 17 775: CMP A TEMP+1 COMPARE WITH PREVIOUS
0836 07 05 776: BLE IDPAT2 IF LESS, IGNORE
0837 06 0374 777: LDA A IDPT3
0838 07 13 778: STA A SNPAT
0839 06 13 779: IDPAT2 LDA A SNPAT RETURN PATTERN
    
```

```

0861 09 780: RTS
0862 791: *
0863 792: * CORRELATE 7 BITS:
0864 793: *
0865 06 07 784: IDCOR LDA B L7
0866 07 19 785: STA B SNPAT+1 USE THIS BYTE AS COUNTER
0867 0F 786: CLR B
0868 44 787: IDCR1 LSR A
0869 25 01 788: SCD IDCR2
0870 50 789: INC B
0871 7A 0015 790: IDCR2 DEC SNPAT+1
0872 26 07 791: BNE IDCR1
0873 17 792: TBA CORR COEFF IN ACCA
0874 09 793: RTS
0875 794: *
0876 795: * MESSAGE IDENTITY PATTERNS:
0877 796: * (SHIFTED 7-BIT M-SEQUENCES)
0878 797: *
0879 74 798: IDPT1 FCB %01110100
0880 53 799: IDPT2 FCB %01010011
0881 4E 800: IDPT3 FCB %01001110
801: *****
802: * CORRELATE 8 BITS IN ACCA WITH BYTE POINTED
803: * TO BY ERRPTR. (DATA RIGHT JUSTIFIED)
804: *****
0885 36 805: ERROR PSH A SAVE ACCA
0886 0F 06 806: STX XTEMP
0887 0E 00 807: LDX ERRPTR
0888 0A 00 808: EOR A X
0889 07 809: PSH B BIT COUNTER
0890 01 03 810: ERRO CMP B L3
0891 27 04 811: BEQ ERR1
0892 43 812: ASL A LEFT JUSTIFY RESULT
0893 50 813: INC B
0894 20 08 814: BRA ERRO
0895 33 815: ERR1 PUL B
0896 49 816: ERR3 ROL A
0897 24 05 817: BCC BITOK
0898 00 039B 818: JSR BITERR BIT IN ERROR
0899 20 03 819: BRA ERR2
0900 00 08DB 820: BITOK JIR NBITER BIT NOT IN ERROR
0901 5A 821: ERR2 DEC B
0902 26 02 822: BNE ERR3
0903 08 823: ERR4 INX
0904 0F 00 824: STX ERRPTR
0905 0E 06 825: LDX XTEMP RESTORE XREG
0906 32 826: PUL A RESTORE DATA BYTE
0907 09 827: RTS
0908 828: *****
0909 829: * BIT IN ERROR:
0910 830: *****
0911 36 831: BITERR PSH A SAVE REGISTER3
0912 07 832: PSH B
0913 0F 14 833: STX XTEMP2
0914 0E 0F 834: LDX STTPTR
0915 70 000C 835: TST PPREV ERROR ON PREVIOUS BIT ?
0916 26 0E 836: BNE BITERR1
0917 08 837: INX NO; START A NEW COUNTER
0918 08 838: INX
0919 4F 839: CLR A
    
```

RECEIVER SOFTWARE

```

03A9 A7 00 340: STA A X
03AB 40 341: INC A
03AD A7 01 342: STA A 1.X
03AF 36 FF 343: LDA A $FF
03B1 A7 02 344: STA A 2.X
03B3 20 0E 345: BRA BITER2
03B5 A6 01 346: BITER1 LDA A 1.X
03B7 36 01 347: ADD A E1
03B9 19 348: DAA
03BB A7 01 349: STA A 1.X
03BD A6 00 350: LDA A X
03BF 39 00 351: ADD A E0
03C1 19 352: DAA
03C3 A7 00 353: STA A X
03C5 0F 0E 354: BITER2 STX BITPTR
03C7 36 0E 355: LDA A ERRCNT+1
03C9 36 01 356: ADD A E1
03CB 19 357: DAA
03CD 37 0E 358: STA A ERRCNT+1
03CF 36 0A 359: LDA A ERRCNT
03D1 39 00 360: ADD A E0
03D3 19 361: DAA
03D5 37 0A 362: STA A ERRCNT
03D7 36 FF 363: LDA A $FF
03D9 37 0C 364: STA A PREV
03DB 0E 14 365: LDX XTEMP2
03DD 33 366: PUL B
03DF 32 367: PUL A
03E1 39 368: RTS
03E3: *****
03E5: ◆ BIT NOT IN ERROR:
03E7: *****
03E9: NBITER1 PSH A
03EB: PSH B
03ED: STX XTEMP2
03EF: LDX BITPTR
03F1: TST PREV
03F3: BEQ NBITER1
03F5: INX
03F7: INX
03F9: CLR A
03FB: STA A X
03FD: INC A
03FF: STA A 1.X
0401: LDA A $FF
0403: STA A 2.X
0405: BRA NBITER2
0407: NBITER1 LDA A 1.X
0409: ADD A E1
040B: DAA
040D: STA A 1.X
040F: LDA A X
0411: ADD A E0
0413: DAA
0415: STA A X
0417: NBITER2 STX BITPTR
0419: CLR A
041B: STA A PREV
041D: LDX XTEMP2
041F: PUL B

```

RECEIVER SOFTWARE

```

090A 32 900: PUL A
090B 39 901: RTS
090C: *****
090E: ◆ REAL TIME CLOCK
0910: *****
0912: IROPRC LDX SCOUNT
0914: INX
0916: STX SCOUNT
0918: LDA A SECS
091A: ADD A E1
091C: DAA
091E: STA A SECS
0920: CMP A $B0
0922: BNE IROPRN
0924: CLR SECS
0926: LDA A MINS
0928: ADD A E1
092A: DAA
092C: STA A MINS
092E: CMP A $B0
0930: BNE IROPRN
0932: CLR MINS
0934: LDA A HOURS
0936: ADD A E1
0938: DAA
093A: STA A HOURS
093C: CMP A $B4
093E: BNE IROPRN
0940: CLR HOURS
0942: INC DAY
0944: IROPRN LDA A PIAT
0946: RTI
0948: *****
094A: ◆ FETCH TIME
094C: *****
094E: FETTIM LDX LOCALF
0950: JCR PDATA
0952: LDX ITHMES
0954: JCR PDATA
0956: JCR DECNM
0958: STA A HOURS
095A: JCR INCH
095C: CMP A E1
095E: BNE FETTIM
0960: JCR DECNM
0962: STA A MINS
0964: JCR INCH
0966: CMP A E1
0968: BNE FETTIM
096A: JCR DECNM
096C: STA A SECS
096E: LDX LOCALF
0970: JCR PDATA
0972: CLR DAY
0974: RTS
0976: TMMS FOC ENTER TIME OF DAY (HH:MM:SS)
0978: FCS 4
097A: *****
097C: ◆ FETCH DECIMAL NO.
097E: *****

```

```

0003 00 E039 950: DECNM JSR INHEX
0006 43 951: ASL A
0007 43 952: ASL A
0008 43 953: ASL A
0009 43 954: ASL A
000A 13 955: TAB
000B 00 E039 956: JSR INHEX
000C 13 957: ASA
000D 3A 958: RTS
959: *****
970: * CORRELATE DATA:
971: *****
0000 0E 0A06 972: CDSP LDR LBUFF RECEIVED DATA POINTER
0003 0F 08 973: STX NTEMP
0005 06 04 974: LDA B L4 MESSAGE COUNTER
0007 0E 0A03 975: L1 LDR LDATA
000A 0F 08 976: STX NTEMP1
000C 0E 08 977: LOOP2 LDR NTEMP1
000E 0E 00 978: LDA R X
0010 01 23 979: CMP R L4E
0012 07 10 980: BEQ ENDATA
0014 08 981: INX
0015 0F 08 982: STX NTEMP1
0017 0E 06 983: LDR NTEMP
0019 01 17 984: CMP R X
001B 07 02 985: BEQ MERR
001D 00 2A 986: BR MERR
001F 08 987: INX
0021 0F 08 988: STX NTEMP
0023 00 28 989: BR LOOP2
0025 0A 5A 990: ENDATA DEC B
0027 0E 20 991: BNE L1
0029 00 39 992: RTS
002B 0A 993: DATA FCB 'THEQUICKBROWNFOXJUMPSOVERTHELAZYL'
994: *****
995: * CORRELATE 3 CHARS:
996: *****
0000 00 997: MERR RCH B
0003 00 998: BR R X
0006 06 07 999: LDA B L7
0009 06 1000: MERR1 ROP A
000B 06 1001: RCH A
000D 04 0E 1002: BEQ MERR2
000F 06 13 1003: LDA A MERR+1
0011 08 01 1004: ADD A L1
0013 06 1A 1005: DAA
0015 07 13 1006: STA A MERR+1
0017 06 12 1007: LDA A MERR
0019 08 00 1008: ADD A L0
001B 07 19 1009: DAA
001D 07 12 1010: STA A MERR
001F 08 1011: MERR2 PUL A
0021 0A 5A 1012: DEC B
0023 06 0A 1013: BNE MERR1
0025 0A 38 1014: PUL B
0027 06 3A 1015: RTS
0029 06 1016: BUFF RMB 129
002B 07 1017: FCB0 RMB 2
002D 00 1018: FCB 0
002F 05 1019: FCB 'ERRDAT000'
    
```

```

0A93 1020: RMB 156
0B2F 1021: FCB1 RMB 2
0B31 01 1022: FCB 1
0B32 45 1023: FCB 'ERRPAT000'
0B3B 1024: RMB 156
0BD7 1025: ERRBUF RMB 256
1026: *****
1027: * SLAVE PROCESSOR ROUTINES FOR DE-INTEPLEAVING AND DECODING:
1028: * INFLAG=0 -> NO DE-INTEPLEAVING REQUIRED; DECODE DATA
1029: * IN FLAG INDICATED BY TFLAG
1030: * INFLAG<>0 -> DE-INTEPLEAVE DATA FROM ONE OF TWO
1031: * TABLES:
1032: * TFLAG=0 -> DATA TO BE DE-INTEPLEAVED: TBINT0
1033: * DATA TO BE DECODED: TB0000
1034: * TFLAG<>0 -> DATA TO BE DE-INTEPLEAVED: TBINT1
1035: * DATA TO BE DECODED: TB0001
1036: *****
C3FE 1037: ORS $C3FE
C3FE 01 12 1038: FDB SLAVE0-LATCH
1039: *
1040: * SLAVE VARIABLES:
1041: *
0000 1042: ORS $0000
0000 1043: LATCH RMB 1
0001 1044: ORS 1
0001 1045: STFLAG RMB 1 STATUS FLAG
0002 1046: INFLAG RMB 1
0003 1047: TFLAG RMB 1
0004 1048: BRCCOR RMB 2 NO. OF CORRECTED ERRORS
0006 1049: PK RMB 2 RECEIVED WORD P(K)
0008 1050: STEMP RMB 2 KREG TEMP STORE
000A 1051: STEMP1 RMB 2
000C 1052: STEMP2 RMB 2
000E 1053: STEMP3 RMB 2
0010 1054: TBDECD RMB 2
0012 1055: TBDECD1 RMB 2
0014 1056: M RMB 1
0015 1057: TB1PTR RMB 2
0017 1058: TB2PTR RMB 2
0019 1059: S1 RMB 1
001A 1060: S3 RMB 1
001B 1061: SIGMA2 RMB 1
001C 1062: SHFCNT RMB 1
001D 1063: S1EXP RMB 1
001E 1064: TEMP1 RMB 1
001F 1065: I RMB 1
0020 1066: TBINT0 RMB 30
002E 1067: TBINT1 RMB 30
005C 1068: TB0000 RMB 32
007C 1069: TB0001 RMB 32
1070: *
1071: * SLAVE MAIN PROCEDURE:
1072: *
C100 1073: ORS $C100
C100 01 1074: TBL1 FCB 1,2,4,8,3,6,$C GALOIS FIELD LOG TABLE
C107 08 1075: FCB 18,5,1A,7,1E,$F,$D,9,1
C110 98 1076: M1 FCB '10011000 MINIMUM POLYNOMIAL M1(K)
C111 F8 1077: M3 FCB '111111000 MINIMUM POLYNOMIAL M3(K)
C112 0E 00FE 1078: SLAVE0 LDS $00FE STACK POINTER
C115 7D 0003 1079: TBT TFLAG WHICH TABLES REQUIRED ?
    
```

```

1118 07 05 1030: SED SLAVE1
1119 08 0000 1031: LDX ETBCDD1 IF <> 0 THEN SELECT
1120 09 03 1032: BRR SLAVE2
1121 0E 0057 1033: SLAVE1 LDX ETBCDD0 OTHERWISE SELECT
1122 0F 10 1034: SLAVE2 STX TBDECD TBCCDD
1123 00 0000 1035: TST INFLAS DE-INTERLEAVING REQUIRED ?
1124 07 11 1036: SED SLAVE3 SKIP NEXT SECTION IF NOT
1125 00 0000 1037: TST TBFLAS TEST TABLE FLAS AGAIN
1126 07 05 1038: SED SLAVE0
1127 0E 005E 1039: LDX LTSINT1
1128 01 02 1040: BRR SLAVE1
1129 0E 0000 1041: SLAVE0 LDX LTSINT0
1130 0F 12 1042: SLAVE1 STX TSINT
1131 30 07 1043: BRR DINTLY DE-INTERLEAVE DATA
1132 00 45 1044: SLAVE3 BRR DECODE DECODE DATA
1133 0F 0001 1045: CLR DTFLAS TELL MASTER WE'RE DONE
1134 20 5E 1046: LOOP BRR LOOP
1097: *****
1098: * DE-INTERLEAVE 30 BYTES OF DATA FROM ADDRESS CONTAINED
1099: * IN TSINT, AND ENTER AS (7-8) BITS IN TABLE WHOSE
1100: * ADDRESS IS CONTAINED IN TBDECD:
1101: *****
1102: DINTLY LDX TSINT ADDRESS OF INTERLEAVED BLOCK
1103: STX STEMP2
1104: LDA B E7 DO INFORMATION BITS
1105: LDX TBDECD
1106: STX STEMP
1107: BRR DINTL1
1108: LDA B E8 DO PARITY CHECK BITS
1109: LDX TBDECD
1110: INX
1111: STX STEMP
1112: BRR DINTL1
1113: RTS
1114: *****
1115: DINTL1 PSH B
1116: LDX STEMP
1117: STX STEMP1
1118: BRR DINTL2
1119: PUL B
1120: DEC B
1121: BNE DINTL1
1122: RTS
1123: *****
1124: DINTL2 LDA B E2 INTERLEAVE DEPTH = 2 * 8
1125: DINTL3 PSH B
1126: LDA B E3
1127: DINTL4 LDX STEMP2
1128: ROL X EXTRACT A BIT
1129: LDX STEMP1 AND ENTER INTO
1130: ROL X NEW TABLE
1131: INX
1132: INX
1133: STX STEMP1
1134: DEC B BIT COUNTER
1135: BNE DINTL4
1136: LDX STEMP2
1137: INX
1138: STX STEMP2
1139: PUL B

```

```

1140: DEC B
1141: BNE DINTL2
1142: RTS
1143: *****
1144: * DECODE WORDS IN SELECTED TABLE
1145: *****
1146: DECODE LDX TBDECD GET SELECTED TABLE ADDRESS
1147: STX STEMP INTO XTEMP
1148: STX STEMP1
1149: LDA B E16
1150: PSH B
1151: LDX STEMP
1152: LDA A K GET INFO BITS
1153: AND A E37F MAKE SURE ITS ASCII
1154: LDA B 1*X GET CHECKBITS
1155: ROL B LEFT-JUSTIFY CODEWORD
1156: ROL A
1157: STA A RY PUT IN DECODING PARAMETERS
1158: STA B RY+1
1159: INX
1160: INX
1161: STX STEMP PSHY FOR NEXT
1162: BRR DECDD FIND CORRECT WORD
1163: LDX STEMP1
1164: LDA A RY
1165: LCP A CORRECTED CHARACTER
1166: STA A X
1167: INX
1168: STX STEMP1
1169: PUL B DONE 16 ?
1170: DEC B
1171: BNE DDD1
1172: RTS
1173: *****
1174: * GALOIS FIELD LOGARITHM/ANTI-LOGARITHM TABLES:
1175: *****
1176: *****
1177: * BEGIN DECODING PROCEDURE
1178: *****
1179: DECDD LDX ETBL1-LATCH
1180: DF 15 1180: STX TB1PTR INITIALISE LOOKUP POINTERS
1181: CE 0200 1181: LDX ETBL2-LATCH
1182: DF 17 1182: STX TB2PTR
1183: *****
1184: * FIND POWER SUM SYMMETRIC FUNCTIONS G1 & G3
1185: *****
1186: *****
1187: * DIVISION ROUTINE TO FIND REMAINDER
1188: *****
1189: SWHD LDA B E11 INITIALISE SHIFT COUNT
1190: D7 10 1190: STA B SHFNT
1191: 96 06 1191: LDA A RY GET RECEIVED WORD MOD
1192: 06 07 1192: LDA B RY+1 & LCB
1193: B8 0110 1193: DIV1 EOR A M1-LATCH
1194: 85 80 1194: TST1 BIT A E180 LEFT JUSTIFIED ?
1195: 26 59 1195: BNE DIV1 YES! EXOR AGAIN
1196: 58 1196: ROL B NO! SHIFT DIVIDEND LEFT
1197: 49 1197: ROL A
1198: 7A 0010 1198: DEC SHFNT
1199: 26 55 1199: BNE TST1

```

```

0100 44      1200:   LSR A           RIGHT JUSTIFY REMAINDER
010E 44      1201:   LSR A
0114 44      1202:   LSR A
0119 44      1203:   LSR A
0121 37 19    1204:   STA A S1
1205:
1206: * COMPUTE S3 BY USING LOOKUP TABLES TO SUM *
1207: * POWERS OF R (ALPHA**3). *
1208:
0128 75 001A 1209:   SYNDS3 CLR S3           ZERO S3
012E 0E 0E    1210:   LDA B #14           EXPONENT COUNT
0133 07 10    1211:   STA B S3FONTR
013A 26 06    1212:   LDA A R#           GET WORD
0141 26 07    1213:   LDA B R#+1
0148 35 00    1214:   SYNDS3 BIT A #130    LEFT JUSTIFIED ?
0150 36 3E    1215:   BNE FELMNT         YES: FIND FIELD ELEMENT
0158 53      1216:   SYNDS3 RCL B       SHIFT WORD
015B 49      1217:   ROL A
0164 7A 0010 1218:   DEC S3FONTR       NEXT EXPONENT
0167 20 55    1219:   BGE SYNDS3        BRANCH IF >=0
0169 70 0019 1220:   TST S1           S1=0?
016E 26 3F    1221:   BNE SIG2
016E 70 001A 1222:   TST S3           S3=0?
0171 26 1A    1223:   BNE SIG2
0173 39      1224:   NOERR RTS         ERROR-FREE WORD
1225:
1226:   JMS #0200
1227:   TBL2 FCB 0,0,1,4,2,8,5,1A,3 GALOIS FIELD ANTILOGS
1228:   FCB #E,3,7,6,5D,1B,1C
1229:   FELMNT RSH B     SAVE ACCS ON STACK
1230:   RSH A
1231:   LDA B S3FONTR   GET CURRENT EXPONENT
1232:   TBA
1233:   RCL A           MULTIPLY BY 3
1234:   RRA
1235:   SYNDS3 CMP A #15  ADD MODULO-15
1236:   BLT SYNDS3
1237:   SNE A #15
1238:   BRA SYNDS3     GO TEST AGAIN
1239:   SYNDS3 STA A T#1PTR+1  SAVE EXP IN PTR
1240:   LDX T#1PTR     GET POINTER
1241:   LDA A #K       GET FIELD ELEMENT
1242:   EOR A #3       ADD IN TO S3
1243:   STA A #3       AND RETURN NEW VALUE
1244:   RUL A         RESTORE STACK
1245:   RUL B
1246:   BRA SYNDS3
1247: * FIND SIGMA2=S1+S1+S3*S1 (GF(2**4)) *
1248:
0220 35 19    1249:   SIG2 LDA A S1       GET S1
022A 00 10    1250:   BR GFSOR         FIND S1*S1
0231 37 1E    1251:   STA A TEMP1     SAVE RESULT
0233 36 1A    1252:   LDA A #3
0235 27 04    1253:   BEQ ORZERO     BRANCH IF ZERO
0237 26 19    1254:   LDA B S1
0239 30 10    1255:   BR GFDIV        FIND S3*S1
023B 32 1E    1256:   ORZERO EOP A TEMP1  S1*S1+S3*S1
023D 37 19    1257:   STA A SIGMA2
023F 20 3E    1258:   BRA CONT        CONTINUE WITH MAIN PROCEDURE
    
```

```

1260: * GALOIS FIELD SQUARING ROUTINE: *
1261: * FINDS ACCA=ACCA*ACCA IN GF(2**4) *
1262:
0241 97 19    1263:   GFSOR STA A T#2PTR+1  FIELD ELEMENT >>POINTER
0243 DE 17    1264:   LDX T#2PTR       FETCH POINTER
0245 A6 00    1265:   LDA A #K         FETCH EXPONENT OF ROOT
0247 97 1D    1266:   STA A S1EXP      SAVE EXPONENT
0249 48      1267:   RSL A           MULTIPLY BY 2
024A 31 0F    1268:   CMP A #15        MODULO-15
024C 2D 02    1269:   BLT S01
024E 30 0F    1270:   SUB A #15
0250 97 16    1271:   S01 STA A T#1PTR+1   NEW EXPONENT >>POINTER
0252 DE 15    1272:   LDX T#1PTR       FETCH POINTER
0254 A6 00    1273:   LDA A #K         FETCH NEW FIELD ELEMENT
0256 39      1274:   RTS
1275:
1276: * GALOIS FIELD DIVISION ROUTINE: *
1277: * FINDS ACCA = ACCA/ACCB IN GF(2**4) *
1278:
0257 97 18    1279:   GFDIV STA A T#2PTR+1  FIELD ELEMENT >>POINTER
0259 DE 17    1280:   LDX T#2PTR       FETCH POINTER
025B A6 00    1281:   LDA A #K         FETCH EXPONENT
025D D7 18    1282:   STA B T#2PTR+1  DIVISOR FIELD ELEMENT
025F DE 17    1283:   LDX T#2PTR       DIVISOR EXPONENT
0261 E6 00    1284:   LDA B #K         SUBTRACT EXPONENTS
0263 10      1285:   SBA
0264 2A 02    1286:   SBL QUOT
0266 08 0F    1287:   ADD A #15        MODULO-15
0268 97 16    1288:   QUOT STA A T#1PTR+1  NEW EXPONENT >>POINTER
026A DE 15    1289:   LDX T#1PTR       GET POINTER
026C A6 00    1290:   LDA A #K         QUOTIENT FIELD ELEMENT
026E 39      1291:   RTS
1292:
1293: * BEGIN SEARCHING FOR ROOTS *
1294:
026F 4F      1295:   CONT CLR A         CLEAR "I"
0270 97 1F    1296:   TEST STA A I       SAVE CURRENT "I"
0272 49      1297:   RSL A           EXPONENT*2
0273 31 0F    1298:   CMP A #15        MODULO-15
0275 2D 02    1299:   BLT S02
0277 30 0F    1300:   SUB A #15
0279 97 16    1301:   S02 STA A T#1PTR+1  GET SQUARED FIELD ELEMENT
027B DE 15    1302:   LDX T#1PTR
027D A6 00    1303:   LDA A #K
027F 97 1E    1304:   STA A TEMP1     AND SAVE IT
0281 D6 1D    1305:   LDA B S1EXP     GET EXPONENT OF S1
0283 0B 1F    1306:   ADD B I
0285 C1 0F    1307:   CMP B #15        MODULO-2 ADD
0287 2D 02    1308:   BLT MUL1
0289 C0 0F    1309:   SUB B #15
028B D7 16    1310:   MUL1 STA B T#1PTR+1  SAVE EXPONENT
028D DE 15    1311:   LDX T#1PTR       GET POINTER
028F E6 00    1312:   LDA B #K         GET FIELD ELEMENT
0291 D8 1E    1313:   EOR B TEMP1     FIND PARTIAL SUM (GF(2**4))
0293 D8 1D    1314:   EOR B SIGMA2   ADD IN SIGMA2
0295 27 0A    1315:   BEQ ROOT       ROOT FOUND ?
0297 96 1F    1316:   RTI           FETCH LOCATOR
0299 31 0E    1317:   CMP A #14
029B 27 03    1318:   BEQ STOP5      STOP IF SEARCH DONE
029D 4C      1319:   INC A         INCREMENT LOCATOR
    
```


APPENDIX 3

PUBLICATIONS

- (1) "High Speed Software Cassette Interface for the SWTP 6800 System"
- (2) "Real-time Adaptive Coding of Ionospherically Propagated Data"
- (3) "Microprocessor Controlled Trackside Recorder for British Rail"

HIGH-SPEED SOFTWARE CASSETTE INTERFACE FOR THE SWTP 6800 SYSTEM

David R. Isaac

Dept. of Applied Physics & Electronics, University of Durham

This article describes a software approach to the problem of interfacing a microprocessor system (in this case the SWTP 6800 system) to an audio cassette used for data storage. This alternative to the more conventional hardware methods offers several distinct advantages. Since the system is software based (the only hardware requirements being the cassette player and two ports of a PIA), its most obvious attraction is low cost. Unlike dedicated hardware cassette interfaces, total control over speed is available and the system is capable of data transfer rates up to about 2000 baud. With the parameters given in the listings, a rate of 1600 baud is obtainable.

Figure 1 illustrates the hardware configuration. A PIA is plugged into row 4 in the motherboard and output to the mic. socket of the cassette recorder is taken directly via a screened lead from the most significant bit of the 'A' side of the PIA. Incoming data from the cassette player is received via another screened lead connected from the extension speaker socket to the most significant bit of the 'B' side. Good results were obtained using medium quality audio cassettes and with the volume and tone controls on the cassette player both set to maximum.

Data is stored on the cassette as a frequency shift keyed signal. Logic levels '0' and '1' are stored as single cycles of 2800 and 1400 Hz respectively which represents an average bit rate of 1600 baud. Data is sent as a MIKBUG formatted tape headed by a string of 200 ASCII nulls and terminated with an ASCII 'S9' end of file pattern. The 'save' and 'read' programs are shown in figures 2 and 3 respectively, and have been given arbitrary origins, although both programs are fully relocatable. By storing the programs in PROM, the risk of overwriting them is eliminated.

To save an area of data on cassette tape, first enter the beginning address of the area in locations A002 and A003 and the final address in locations A004 and A005. Set locations A048

and A049 to the start address of the SAVE program, switch the cassette player on to record and enter G. When the transfer of data is complete, the system automatically returns to MIKBUG.

To read data from the cassette, first set locations A048 and A049 to the start address of the READ program, start the cassette rolling a little before the data begins (to allow it to pick up speed) then enter G. When the end of file pattern is read from the tape the system returns to MIKBUG. Should an error be encountered during playback, an 'E' will be printed on the teletype; the tape should be stopped, rewound slightly and a G should again be entered. Note that there is no need to reset the program counter before doing this.

This cassette system has been used for several months and has been found to be very reliable. The data transfer rate is more than five times that of Kansas City standard system which results in a considerable time saving for long programs. A change of speed can be effected by changing the frequencies of the two FSK tones. This is done by altering the timing loop parameter at location 50D0 in the SAVE program, and the mean number of samples per half-cycle at location 600B in the READ program.

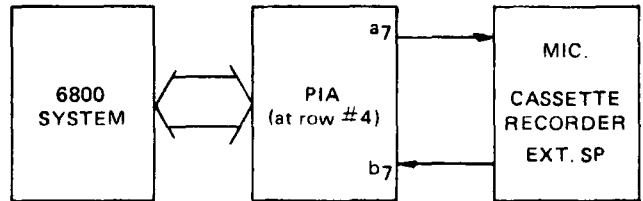


Figure 1. Hardware Configuration

```

TITLE: SAVE

LOC OBJECT CODE SOURCE STATEMENTS
*
* NAM SAVE
*
* TRANSMIT ROUTINES
*
* This program sends mikbug formatted data as
* an FSK signal through one port of a PIA.
* A logic '1' is sent as one cycle of 1400 Hz.
* A logic '0' is sent as one cycle of 2800 Hz.
* Bega, Bega + 1 and Enda, Enda + 1 contain the
* first and last addresses respectively of the
* data area to be saved.
*
901G PIAAD EQU $8010 PIA ADDRESS
* MIKBUG ROM ADDRESSES:
A002 BEGA EQU $A002 DATA START ADDRESS
A004 ENDA EQU $A004 DATA END ADDRESS
A00E TEMP EQU $A00E
A00F TW EQU $A00F
A011 MCOUNT EQU $A011
A012 XTEMP EQU $A012
* END-OF-FILE PATTERN:
A020 ORG $A020
A020 OD MTAPE2 FCB $D,$A,0,0,0,0,$53,$39,4
A021 OA
A022 OO
A023 OO
A024 OO
A025 OO
A026 53
A027 39
A028 04
* MIKBUG ROM ADDRESSES
EOE3 CONTRL EQU $EOE3
E134 MTAPE1 EQU $E134
5000 ORG $5000 PROGRAM ORIGIN
5000 CE FFO4 START LDX #FF04 INITIALISE PIA
5003 FF 8010 STX PIAAD
* SEND 200 NULLS
5006 C6 C8 LDA B #200
5008 4F NULLS CLR A NULL
5009 8D 67 BSR SEND
500B 5A DEC B
500C 26 FA BNE NULLS
* MIKBUG "PUNCH" ROUTINE:
500E FE A002 PUNCH LDX BEGA START ADDRESS
5011 FF A00F STX TW SAVE IT
5014 B6 A005 SEN11 LDA A ENDA+ 1
5017 B0 A010 SUB A TW+1
501A F6 A004 LDAB ENDA
501D F2 A00F SBCB TW
5020 26 04 BNE SEN22
5022 81 10 CMP A #16
5024 25 02 BCS SEN23
5026 B6 0F SEN22 LDA A #15
5028 B8 04 SEN23 ADD A #4
502A B7 A011 STA A MCOUNT FRAME COUNT THIS RECORD
502D 80 03 SUB A #3
502F B7 A00E STA A TEMP BYTE COUNT THIS RECORD
* SEND C/R, L/E, NULLS, S, 1 (START OF RECORD):
5032 CE E134 LDX #MTAPE1
5035 8D 14 BSR PDATAL
5037 5F CLR B ZERO CHECKSUM
* SEND FRAME COUNT
5038 CE A011 LDX #MCOUNT
503B 8D 38 BSR SEND2
* SEND ADDRESS
503D CE A00F LDX #TW
5040 8D 33 BSR SEND2
5042 8D 31 BSR SEND2
* SEND DATA
5044 FE A00F LDX TW
5047 8D 2C BSR SEND2 SEND ONE BYTE (2 FRAMES)
5049 7A A00E DEC TEMP DEC BYTE COUNT
504C 26 F9 BNE SEN32
504E FF A00F STX TW
5051 53 COM B
5052 37 PSH B
5053 30 TSX
* SEND CHECKSUM
5054 8D 1F BSR SEND2
5056 33 PUL B RESTORE STACK
5057 FE A00F LDX TW
505A 09 DEX
505B BC A004 CPX ENDA
505E 26 B4 BNE SEN11
* IF FINISHED, SEND END-OF-FILE
5060 CE A020 LDX #MTAPE2
5063 8D 06 BSR PDATAL
5065 7E E0E3 JMP CONTRL RETURN TO MIKBUG
5068 8D 24 PDATA2 BSR SENDIT
506A 08 INX
506B A6 00 PDATAL LDA A X
506D 81 04 CMP A #4
506F 26 F7 BNE PDATA2
5071 39 RTS
5072 8D 1A SEND BSR SENDIT
5074 39 RTS
* SEND 2 HEX CHARS: UPDATE CHECKSUM
    
```

Continued on page 42

```

5 EB 00 SEND2 ADD J O,X UPDATE CHECKSUM          50B7 86 FF          LDA A #1FF PIA---> HIGH
7 A6 00 OUT2H LDA A O,X SEND 2 HEX CHARS         50B9 8D 0F          BSR DELAY
9 8D 05 OUT2HA BSR OUTHL OUT LEFT CHAR         50BB 8D 0D          BSR DELAY WAIT AGAIN
B A6 00 LDA A O,X                                50BD 32              PUL A RESTORE STACK
D 08 INX                                          50BE 39              RTS
E 20 04 BRA OUTHR OUT RIGHT CHAR & RETURN        50BF 36              * BIT=0; TRANSMIT CYCLE OF HIGH FREQUENCY
O 44 OUTHL LSR A                                50C0 86 00          ZERO PSH A SAVE ACCA
1 44 LSR A                                        50C2 8D 06          BACK LDA A #0 PIA---> LOW
2 44 LSR A                                        50C4 86 FF          BSR DELAY TIMER
3 44 LSR A                                        50C6 8D 02          LDA A #1FF PIA---> HIGH
4 84 0F OUTHR AND A #1F OUT RIGHT BCD DIGIT    50C8 32              BSR DELAY
6 8B 30 ADD A #30                                50C9 39              PUL A RESTORE STACK
8 81 39 CMP A #39                                50CA 37              RTS
A 23 02 BLS SENDIT                              50CB C6 08          * FREQUENCY CONTROL DELAY LOOP
C 8B 07 ADD A #7                                50CD B7 8010        DELAY PSH B SAVE ACCB
* GENERATE FSK SIGNAL AND SEND THROUGH PIA    50DD 5A              LDA B #8 DELAY COUNTER
*                                          50DE 26 FA          STA A PIAAD OUTPUT TO PIA
*                                          50DF 33              DEC B DEC LOOP COUNT
*                                          50E0 39              BNE LOOP1 LOOP AGAIN
*                                          50E1 33              PUL B RESTORE STACK
*                                          50E2 39              RTS RETURN
E 37 SENDIT PSH B                                50E3 39              END
F FF A012 STX XTEMP SAVE XREG
2 C6 08 LDA B #0 BIT COUNT
4 8D 29 BSR ZERO SEND START BIT
6 0D SEC
7 49 NEXBIT ROL A GET NEXT BIT
8 24 04 BCC SKIP JUMP IF ZERO
A 8D 14 BSR ONE SEND A '1'
C 20 02 BRA OVER
E 8D 1F SKIP BSR ZERO SEND A '0'
O 0D SEC
1 5A DEC B
2 26 F3 BNE NEXBIT NEXT BIT
4 8D 0A BSR ONE SEND STOP BIT
6 86 00 LDA A #0 PIA---> LOW
8 B7 8010 STA A PIAAD
AB FE A012 LDX XTEMP RESTORE XREG
AE 32 PUL A AND STACK
AF 39 RTS
* BIT=1; TRANSMIT CYCLE OF LOW FREQUENCY
O 36 ONE PSH A SAVE ACCA
1 86 00 AGAIN LDA A #0 PIA---> LOW
3 8D 15 BSR DELAY TIMER
5 8D 13 BSR DELAY AND AGAIN

```

SYMBOL	VALUE	OVER	50AO
AGAIN	50B1	PDATA1	506B
BACK	50C0	PDATA2	5068
BEGA	AE02	PIAAD	8010
CONTRL	EOE3	PUNCH	500E
DELAY	50CA	SEND	5072
ENDA	A004	SENDIT	508E
LOOP1	50CD	SEND2	5075
MCOPT	A011	SEN11	5014
MTAPE1	E134	SEN22	5026
MTAPE2	A020	SEN23	5028
NEXBIT	5097	SEN32	5047
NULLS	5008	SKIP	509E
ONE	50B0	START	5000
OUTHL	5080	TEMP	A00E
OUTHR	5084	TW	A00F
OUT2H	5077	XTEMP	A012
OUT2HA	5079	ZERO	50BF

```

TITLE: READ
603A 8D E1D1 JSR OUTCH
603D 7E E0E3 LOAD21 JMP CONTRL RETURN TO MIKBUG
* BUILD ADDRESS:
BADDR BSR BYTE GET A BYTE
STA A XHI MSBYTE
BSR BYTE GET A BYTE
STA A XLOW LSBYTE
LDX XHI ADDRESS WE BUILT
RTS
* INPUT 1 BYTE (2 FRAMES):
BYTE BSR INHEX READ HEX CHAR
ASL A
ASL A
ASL A
ASL A
TAB
BSR INHEX READ HEX CHAR
ABA
TAB
ADD B CKSM
STA B CKSM
RTS
* INPUT HEX CHAR:
INHEX BSR READ1
SUB A #130
BMI LOAD19 NOT HEX; ERROR ?
CMP A #109
BLE INIHG
CMP A #111
BMI LOAD19 NOT HEX; ERROR ?
CMP A #116
CMP A #116
BGT LOAD19 NOT HEX; ERROR ?
SUB A #7
RTS
* SAMPLING AND TIMING ROUTINES:
READ1 PSH B SAVE ACCB
STX XTEMP AND XREG
START CLR A
CLR B
SEC
SBIT BSR LOOK LOOK FOR START BIT
BCS SBIT NOT FOUND; LOOK AGAIN
NEXBIT PSH A SAVE ACCS
PSH B
BSR LOOK
PUL B RESTORE ACCS
PUL A
ROL A GET RECEIVED BIT
INC B 8 BITS FOUND ?
CMP B #8
BNE NEXBIT NO; FETCH NEXT BIT
LDX XTEMP RESTORE INDEX REG
PUL B
RTS
* FETCH A BIT:
LOOK CLR CNTR SAMPLE COUNT
LDA A #180 SELECT INPUT LINE
CMP A PIAAD START OF CYCLE ?
BNE LOOP1 NO; LOOK AGAIN
INC CNTR YES; INC SAMPLE COUNT
NOP
CMP A PIAAD INPUT STILL HIGH ?
BEQ LOOP2 YES; KEEP LOOPING.
LDA A CNTR TOTAL SAMPLE COUNT
CMP A MEAN COMPARE WITH MEAN

```


D.R. Isaac and C.T. Spracklen
Department of Applied Physics and Electronics
Durham University
ENGLAND

Abstract

High frequency radio links are time - varying channels which can introduce considerable levels of attenuation and delay distortions. These effects, together with interference from natural and man - made sources, result in the high error rates observed over the HF channel. This paper describes a novel microprocessor implementation of a data transmission system for use over HF radio, where 16 data channels are used to modulate 8 frequency - agile subcarriers contained within the voice channel. The system attempts to locate the subchannels within the quiet portion of the channel to avoid narrow band interference. Broad - band effects are overcome by using bit - interleaved block codes. The software - orientated implementation renders the system both cost - effective and flexible.

1. Introduction

Information transmitted via high frequency (HF) radio signals reflected from the ionosphere is frequently degraded due to the characteristics of the propagation medium. The high error rates encountered can be attributed to the effects of multipath propagation and interference, both natural and man - made.

This paper describes an experiment being undertaken by the Department of Applied Physics and Electronics, Durham University, intended to demonstrate that with the aid of modern electronics technology in the form of microprocessors, it is possible to effect a significant improvement in the error rate.

The simplest, and often the most effective, way of reducing these errors is to implement some form of channel evaluation system to assess the suitability of a number of channels for transmission over the radio link /1,2/, and to choose the channel yielding the best signal - to - noise ratio. This approach may often be unrealistic since a wide selection of frequencies may not be available to the communicator and he may often have to make the best possible use of a frequency far removed from the optimum. However, even for a single assigned frequency, substantial improvements can be obtained by using adaptive techniques based on a 'microscopic' analysis of the channel /3/.

Over recent years there has been a considerable resurgence of interest in communications using the HF part of the radio spectrum. Many users faced with the problems of cost and vulnerability of satellite communications, have begun to look again at the prospect of achieving reliable data communications using HF radio links. Indeed, the HF path is the only alternative for long distance circuits involving mobiles such as ships and aircraft. However, the problems are considerable. If we consider signals propagated to outside the skip zone (where ionospheric propagation predominates) then we have the difficulty of communicating via an anisotropic time - varying medium with noise levels greatly exceeding those found in other communications systems. However, there has been considerable progress in the state of electronics technology in recent years, particularly in the digital field, and it seems likely that such a communications system would need to take full advantage of such advances.

2. Design Considerations

Multipath propagation causes time dispersion of the received signal /4/ which results in severe intersymbol interference if the signal element duration is of insufficient length. It has been shown /5/ that the optimum frame length for transmission over a dispersive medium is equal to $\sqrt{L/B}$, where L is the time spread introduced by the medium and B is the frequency spread. The time spread caused by multipath propagation, and the Doppler shift introduced by ionospheric perturbations, result in a near - optimum frame rate of 75 Hz. Serial data transmission systems are therefore limited to a data rate of 75 baud, which makes inefficient use of the 3 kHz. voice channel. To utilise the channel more efficiently the data can be time division multiplexed for transmission over a number of parallel sub - channels, orthogonally spaced within the voice channel.

Several such parallel sub - channel modems have been developed in the past, notably Kineplex /6/, Kathryn /7,8/, and Codem /9/, for medium speed data transmission over HF radio /10,11/.

A macroscopic investigation of the HF spectrum below the M.U.F. /1,12/ reveals that it is very difficult to find a 3 kHz. slot which is completely free of interference and it has also been shown /3/ that much of this interference is narrow - band. It is therefore unwise to attempt to use the channel to its full capacity. Codem uses a spectral redundancy technique to overcome narrow - band interference and fading by using only 16 of its 25 subchannels to carry information, the remainder being used for redundant parity bits. This is effective but inefficient in that the available transmitter power is distributed amongst all 25 channels which may present a disadvantage when working from a mobile transmitter. Gott /3/ and Betts /12/ have shown that interference measurements made on the voice channel are often valid for several minutes and occasionally for an hour or more. This suggests that an adaptive system might be used which avoids those subchannels exhibiting poor error performance. Ideally, a reverte link would be used to assess the signal - to - noise ratio on each subchannel at the receiving site by transmitting a sounding signal from the receiver back to the transmitter. In practice, reverte links are costly and often difficult to implement, especially from mobile sites, and the best that can be done is for the transmitting site to evaluate the spectral distribution of interference within the channel in the absence of a signal. It has been shown /3/ that over distances of several hundred miles the interference pattern at the transmitter site is similar to that observed at the receiver and provides a reasonable criterion for sub - channel selection. Results have also shown /13,14/ that, for a 2 - tone F.S.K. system, the error performance of an ideal frequency agile system greatly exceeds that of a system using fixed frequency allocation.

The system currently being developed uses 16 available equally (orthogonally) spaced sub - channels which accommodate 8 quaternary phase - modulated sub - carriers. One possible distribution of the sub - carriers is illustrated in Figure 1. A 4 - phase modulated signal occupies the same bandwidth as a bi - phase one but carries twice as much information, and was therefore chosen in preference /15,16/. Systems employing pilot tone phase references have been shown to exhibit poor performance on HF /17,18/, so a differential encoding technique is chosen to eliminate the need for an absolute phase reference. The differential phases correspond to dibits arranged in a

de around the unit circle (Figure 2) such that the probable phase error causes only a single bit error. Selection of the sub - channels to be used for transmission is based on the results of a measurement of the noise present in each of the 16 available sub - channels over an observation period of 3.5 seconds. This is a compromise arrived at by consideration of the major factors. Firstly, an observation period that is too long necessitates a large input data buffer at the transmitter if overflow is to be avoided. Secondly, a period that is too short may coincide with the fadeout of any narrow - band interfering signals. These fadeouts have been observed [19,20] to occur over periods of up to 0.5 seconds. Thirdly, as the measurements are based on the results of a spectral analysis using software - implemented Fast - Fourier Transforms, sufficient time must be allowed for the spectrum to be evaluated. After each observation period, the subchannels exhibiting the lowest noise levels are selected for transmission of the next message block. A preamble of phase reversals used to regain synchronization, the receiver is advised as to which sub - channels to be used for subsequent transmission of the data. This advisory sequence has a duty cycle level four times greater than that used for transmission of the message. Results [12] indicate that a reasonable interval between observation periods is of the order of several minutes and each message block consists of 3 minutes of data.

Errors observed on the HF channel predominantly occur in bursts and it has been shown [20,21,22,23] that interleaved binary block coding provides the best protection against these types of errors. As the code is to be implemented in real - time at the transmitter, a code must be used which is a reasonable compromise between burst - correcting ability and implementation complexity. The (15,7) dual - error - correcting binary BCH code [25] is reasonably simple to implement in real time using software [26], and groups of 15 bits interleaved to a depth of 16 in serial along each of the 16 data sub - channels can tolerate wide - band noise bursts or noise bursts of up to 0.43 seconds. This means that a block length (of $16 \times 16 \times 15 = 3840$ bits) in which a total number of 512 errors can be corrected.

Implementation of a parallel sub - channel system is not easily implemented using banks of narrow - band filters [26]. These are often costly and are not suitable in an in - band frequency agile environment where a small number of filters are in use at any one time. For this reason it was decided to use the Discrete Fourier Transform (D.F.T.) for sub - channel selection and phase decoding. Because the time to implement the F.F.T. algorithm using software exceeds the signal element duration, a software approach is adopted by evaluating the chirp - transform algorithm using charge - coupled - device transversal filters [27]. D.F.T. processors using charge - coupled devices are capable of evaluating a 512 - point transform in less than 6 μ S.

Implementation

The primary requirement for system flexibility, together with the falling cost of microprocessor technology, dictated that a primarily 'software - based' implementation was preferable to a purely hardware - based approach [28]. Coding / decoding and modulation / demodulation are controlled almost entirely by the microprocessor system, changes in system performance being effected by simply modifying the software level programs. This ensures that the system costs are kept to a minimum.

A block diagram of the complete system is shown in Figure 3. At the transmitter incoming serial data is buffered and encoded by the Motorola 6800 - based microprocessor system prior to generation of the frequencies and phases required to construct the multi - channel baseband signal. The data is frequency division multiplexed for transmission over eight differential quadrature phase shift keyed (DQPSK) sub - channels and the resulting composite waveform is low - pass filtered to remove components above the signal band which then forms the modulating signal to the single side - band (SSB) suppressed - carrier HF transmitter. During breaks in transmission, the baseband output from the HF receiver at the transmitting site is sampled by the A/D converter and an FFT is performed by the microprocessor to determine the optimum sub - channels for subsequent transmission.

At the receiving station, samples of the filtered baseband signal are processed by an FFT processor whose output coefficients in the frequency domain are used by the microprocessor system to demodulate the composite waveform. Decoding and re - formatting of the demodulated signal then results in the output serial data stream.

For real - time operation, it was found that single processor systems at the transmitter and receiver sites were insufficient to handle all the required tasks. For this reason, a multi - processor configuration has been developed in which one or more 'slave' processor units (Figure 4) are assigned tasks by a master processor which oversees operation of the entire system. Each slave processor unit includes a single 6800 central processing unit (CPU) and 1Kbyte of read / write or random access memory (RAM), which is used to contain data and program areas. The master processor accesses the slave control lines via a 4 - bit latch located at the base address of the slave processor. Once initialised, the slave processor unit is capable of performing tasks completely independently of the master thereby increasing the overall processing power of the complete system.

The transmitter (Figure 5) uses one slave unit for generation of the multitone modulating signal. Serial data at 550 bps is accepted into the serial interface and is loaded into the input data buffer in RAM via an interrupt service routine. Groups of seven data bits are extracted from the buffer and mapped into 15 - bit BCH codewords. A group of seven data bits can be represented as a sixth order modulo - 2 polynomial $d(x)$ which is encoded by finding

$$r(x) = x^6 d(x) + \text{rem}(x^6 d(x)/g(x))$$

where $g(x)$ is the generator polynomial

256 of the resulting codewords are assembled into a 16×240 bit matrix to enable interleaving to depth 16 along each of the 16 data sub - channels. On request from the slave processor, pairs of bits are extracted from the matrix and loaded into the slave RAM, each dibit corresponding to a single frequency division multiplexed sub - channel. For a single sub - channel, the absolute phase is determined by the result of an exclusive - OR operation on the current dibit with the immediately preceding dibit for that sub - channel, the result being used as an index on an 80 - point cosine lookup table in the slave processor - RAM. Samples for each sub - channel are then added in to one of two 80×12 bit multiplexed shift registers; the complete waveform for a single element is thus formed after eight complete rotations. When one shift register is full, its contents are output to the low pass filter at the appropriate sample rate of 6 kHz.

A message consists of 16,384 codewords, a total transmission time of 3.4 minutes. Between messages, an estimate of the interference present in each subchannel is made by evaluating a 64 - point radix - 2 butterfly Fast Fourier Transform algorithm on the sampled data signal. The 9600 Hz. sampling frequency provides a frequency - domain resolution of 150 Hz. Selection of the sub - channels to be used for transmission of the next message sequence is made by finding the eight sub - channel slots exhibiting the lowest power spectral density averaged over eight sample periods. Transmission is then resumed by sending a preamble of phase reversals over the eight sub - carriers used for the previous message. This is followed by an advisory sequence of 8 15 - bit codewords transmitted with a four - fold spectral redundancy to ensure a high probability of correct reception. The first three information bits of each codeword relate to a sub - channel number in the range 0 to 7, the last four indicating a frequency slot in the range 0 - 15. Transfer to the new frequency slots occurs after the final element of the advisory sequence.

The receiver (Figure 6) uses one slave processor to control element synchronisation of the received signal and one for decoding of the demodulated data. Demodulation is accomplished using a charge coupled device (CCD) to evaluate the chirp - Z transform /29/. The chirp - Z transform can be derived from the expression for the discrete Fourier Transform as follows /30/:

$$F_k = \sum_{n=0}^{N-1} f_n \exp(-i2\pi nk/N) \quad (\text{DFT})$$

$n, k = 0, 1, 2, \dots, N-1$

using the identity $2nk = n^2 + k^2 - (k-n)^2$

we obtain

$$F_k = \sum_{n=0}^{N-1} \left[f_n \exp(-i\pi n^2/N) \exp(i\pi(k-n)^2/N) \right] \exp(-i\pi nk^2/N)$$

(CZT)

The implementation involves three operations

- (i) the complex sequence gn is generated by the product of f_n with $\exp(-i\pi n^2/N)$
- (ii) a discrete convolution is performed between gn and $\exp(i\pi(k-n)^2/N)$
- (iii) the resulting sequence is multiplied by $\exp(-i\pi k^2/N)$

These steps are illustrated in Figure 7. Step (ii) involves four convolutions which are performed by the CCD transversal filters. Steps (ii) and (iii) are implemented by using multiplying digital to analog converters and lookup tables in programmable read - only memories which contain sampled sine and cosine 'chirp' waveforms.

For purely real inputs, the analysis band extends from zero to the Nyquist frequency. The CCD device with its associated circuitry implements a fixed length transform of 512 points which, for a frequency resolution of 75 Hz. requires a sampling frequency of 38.4 kHz. The total time taken to acquire 512 samples is thus $512/38400 = 13.3$ mS, which is the element duration. The required signal band extends from 300 to 2550 Hz. which is available in 885 microseconds. The remaining time of 12.4 mS is thus available for processing.

Synchronisation is achieved by observing the behaviour of the phasor. For a sequence of phase reversals, the locus of the phasor as we traverse along the signal element is a spiral, and its magnitude is a triangular wave (Figure 8), whose maxima correspond to the correct sampling instants. The magnitude of the phasor can be computed from the Fast Fourier Transform by observing the Fourier coefficients at the discrete sub - channel frequencies. An estimate of the mean value of the phasor over all sub - channels can be evaluated by finding

$$|\bar{F}| = \frac{1}{8} \sum_{k_c} \sqrt{F_{k_c}^2(\text{Re}) + F_{k_c}^2(\text{Im})}$$

where K_c is the sample number corresponding to the sub - channel frequency. The values of $|\bar{F}|$ thus obtained indicate the position of the primary sampling instant within the frame and the slave processor then instructs an appropriate change to synchronise the frame.

The demodulated data is entered into a 480 - byte matrix in the slave decoder for de - interleaving and decoding. Peterson's decoding algorithm /31/ is used to recover the seven information bits from each received codeword. Arithmetic operations are implemented in the Galois field of (2^8) elements. Addition and subtraction are simply performed modulo - 2, while multiplication and division operations are performed using logarithmic lookup tables. The algorithm used is shown in Figure 9 and can be described as follows

If $r(x)$ is the received word and $g(x) = m_1(x)m_2(x)$ then the two partial syndromes can be found by calculating

$$S_1 = \text{rem} \frac{r(x)}{m_1(x)} \quad S_2 = \text{rem} \frac{r(x)}{m_2(x)}$$

If errors have occurred, the syndromes will be non - zero and the elementary symmetric functions can be found directly from Newton's identities:

$$S_1 + \sigma_1 = 0$$

$$S_2 + S_1\sigma_1 + S_1\sigma_2 + \sigma_2 = 0$$

in matrix notation

$$S = M_t \Sigma$$

where

$$S = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} \quad \Sigma = \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} \quad M_t = \begin{bmatrix} 1 & 0 \\ S_1 & S_1 \end{bmatrix}$$

It is then possible to find the $(\sigma_j)_{j=1,2}$ by computing

$$\Sigma = M_t^{-1} S$$

It is then possible to solve /32/ for the error locator numbers (β_j) by substituting elements of Galois field (2^8) in the equation

$$\begin{aligned} \sum(x) &= (x + \beta_1)(x + \beta_2) \\ &= x^2 + \sigma_1 x + \sigma_2 \end{aligned}$$

This completes the decoding algorithm. The decoded data is stored in the output data buffer and is then converted to serial format by a parallel - to - serial converter.

Conclusion

A novel implementation of a multi-channel modem for data communications over HF radio links has been described. The use of microprocessor technology reduces complexity and increases system flexibility by using a software-based approach. The effects of fading and distortion are overcome by using a parallel channel system to extend the duration of the signal. Narrow-band interference is combated by using in-band frequency agility to locate the carriers within the quiet regions of the voice spectrum. Because of the time-varying nature of the fading, the subcarrier locations are updated at short intervals; thus the system adapts itself to the prevailing interference pattern. Subchannel synchronization and demodulation are accomplished by using frequency-coupled devices to evaluate the Discrete Fourier Transform, thereby alleviating the need for banks of narrow-band filters which are traditionally used in narrow-band carrier modems. Broad band noise effects are overcome by using bit-interleaved binary block

In view of the flexibility of the system, it is possible, if a revertive link were available, to overcome fluctuations in the error patterns observed at the receiver, and to modify the redundancy in the transmitted signal to suit the types of errors received. The system then becomes fully adaptive, the redundancy in the signal always being sufficient to maintain error-free communication. There is much scope for future work!

References

'An automatic HF channel monitoring system', Darnell, R.A.; Proc. Conf. on Advances in HF Communications Systems, IEE, Feb. 79, pp. 57 - 60.

'Use of pilot tones for real-time channel estimation of HF data circuits', Betts, J.A.; Thomson, R.S.; Cook, S.J.; Clark, J.G.; Proc. IEE, Vol. 122, No.9, Sept. 75.

'Characteristics of Interfering signals in nautical HF voice channels', Gott, G.F.; Darnell, M.J.D.; Proc. IEE, Vol. 125, No.11, Nov. 1978, pp. 1208 - 1212.

'Medium-speed digital data transmission over HF channels', Darnell, M.; Proc. of Conf. on Digital Processing of Signals in Communications, Southampton, Sept. 77, pp. 393 - 402.

'Minimum rate waveforms for dispersive channels', Newman, M.S.; Thomson, D.N.; General Atomics Corp., Philadelphia, Pa., Report. 1329 - 98 - 1, May 1964.

'A bandwidth-efficient binary transmission system', Mosier, R.R.; Claibough, R.; IEEE Trans. on Communications and Electronics, Vol. 76, 1958, pp. 723 - 728.

'AN/GSC - 10 (KATHRYN) variable rate data modem for HF radio', Zimmerman, M.S.; Kirsh, A.L.; IEEE Trans. Comm. Tech., Vol. COM - 15, Part 2, pp. 197 - 204.

'Field test results of the AN/GSC - 10 (KATHRYN) data terminal', Kirsh, A.L.; Gray, P.R.; Thomson, D.W.; IEEE Trans. Comm. Tech., Vol. COM - 16, 1969.

'Combined coding and modulation approach for communication over dispersive channels', Chase, R.; IEEE Trans. Comm. Tech., Vol. COM - 21, pp. 159 - 174.

'Program controlled HF modem implementation', de Groot, J.; Fast, D.; Michelson, A.; Carmichael, E.; EASCON '74, pp. 349 - 352.

'Integrated HF error-controlled modem', de Groot, J.; Michelson, A.; GTE Sylvania, Eastern Division, Needham Heights, Mass., 02194.

'Sounding and its use for control of HF (3 - 30

MHz.) adaptive systems for data transmission', Betts, J.A.; Ellington, R.P.; Jones, D.R.L.; Proc. IEE, Vol. 117, No. 12, Dec. 1970.

/13/ 'Improvement of slow-rate FSK by frequency agility and coding', Gott, G.F.; Hillam, B.; Proc. Conf. on Advances in HF Communication Systems and Techniques, Feb. 79, pp. 45 - 49.

/14/ 'An HF data modem with in-band frequency agility', Darnell, M.; Proc. Conf. on Advances in HF Communication Systems and Techniques, Feb. 79, pp. 50 - 54.

/15/ 'Voiceband 4 - phase data set for Global Communication', Horlacher, R.L.; Schroeder, H.C.; IEEE Trans. Com. Tech., Vol. COM - 17, No. 3, June 69.

/16/ 'Error performance of quadrature pilot-tone phase-shift keying', Bussgang, J.; Leiter, M.; IEEE Trans. Comm. Tech., Vol. COM - 16, No. 4, August 68, pp. 526 - 529.

/17/ 'The Fadeogram: An ionogram-like display of the time-varying frequency response of HF SSB radio channels', Filter, J.H.J.; Arazi, B.; Thomson, R.G.W.; IEEE Trans. Comm. Tech., Vol. COM - 26, No.6, June 78.

/18/ 'The Fadeogram - Applied to analysing HF data modem performance', Thomson, R.G.W.; Filter, J.H.J.; National Electrical Engineering Research Institute of the South African Council for Scientific and Industrial Research, Johannesburg.

/19/ 'Effective Application of Forward-Acting Error-Control Coding to Multichannel HF data modems', Pierce, A.W.; Barrow, B.B.; Goldberg, B.; Tucker, J.; IEEE Trans. Comm. Tech., Vol. COM - 18, No. 4, August 1970.

/20/ 'Error patterns measured on transequatorial HF communication links', Brayer, K.; IEEE Trans. Comm. Tech., Vol. COM - 16, No. 2, April 1968.

/21/ 'Improved signal design for fading channels by coding', Chase, D.; CNR Inc., Newton, Mass.

/22/ 'Application of forward-error correction to a Rayleigh fading communication channel', Treciokas, R.; Proc. IEE, Vol. 125, No. 3, March 1978, pp.173 - 178.

/23/ 'The Improvement of Digital HF Communication through coding', Brayer, K.; IEEE Trans. Comm. Tech., Vol. COM - 16, No.6, December 1968.

/24/ 'HF FSK error rate measurements', Ames, J.W.; IEEE Trans. Comm. Tech., Vol. COM - 17, August 69, pp. 438 - 442.

/25/ 'On a class of Error-Correcting Binary Group Codes', Bose, R.C.; Ray - Chaudhuri, D.K.; IEEE Trans. on Information and Control, Vol. IC - 3, 1960, pp. 279 - 290.

/26/ 'Computer Implementation of decoders for several BCH codes', Michelson, A.M.; Polytech. Inst. of Brooklyn International Symposium on Computer Processing in Communications, April 69, pp. 401 - 413.

/27/ 'A 500-stage CCD Transversal Filter for Spectral Analysis', Brodersen, R.W.; Hewes, C.; Buss, D.D.; IEEE Trans. Electronic Devices, Vol. ED - 23, Feb. 1976, pp. 143 - 152.

/28/ 'Microprocessor Implementation of Tactical Modems for Data Transmission over VHF radio', Davis, B.H.; Davis, B.A.; Radio and Electronic Engineer, Vol.49, No. 4, April 79, pp. 204 - 210.

/29/ 'Communications applications of CCD Transversal Filters', Buss, D.D.; Brodersen, R.W.; Hewes, C.; Tasch, A.F.; IEEE Nat. Telecomm. Conf. Rec., Vol. 1, Dec. 1975, pp. 1.1 - 1.5.

/30/ 'Comparison between the CCD CZT and the digital FFT', Buss, D.D.; Veenkant, R.L.; Brodersen, R.W.; Hewes, C.; Proc. 1975 Naval Electronics Lab. Center Int. Conf. on the Application of CCD's, October 1975, pp. 267 - 281.

/31/ 'Encoding and Error Correction Procedures for the Bose-Chaudhuri-Hocquenghem Codes', Peterson, W.W.; IRE Trans. on Information Theory, September 1960, pp. 459 - 470.

/32/ 'Cyclic Decoding Procedures for the Bose-Chaudhuri-Hocquenghem Codes', Chien, R.T.; IEEE Trans. on Information Theory, September 1964.

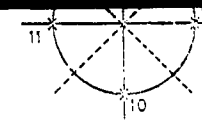
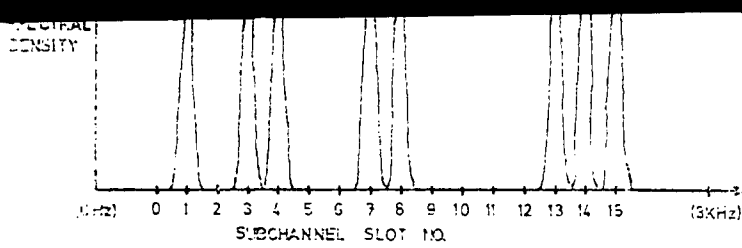


Figure 2. Phase encoding

Figure 1. One possible distribution of subchannels within the voice channel

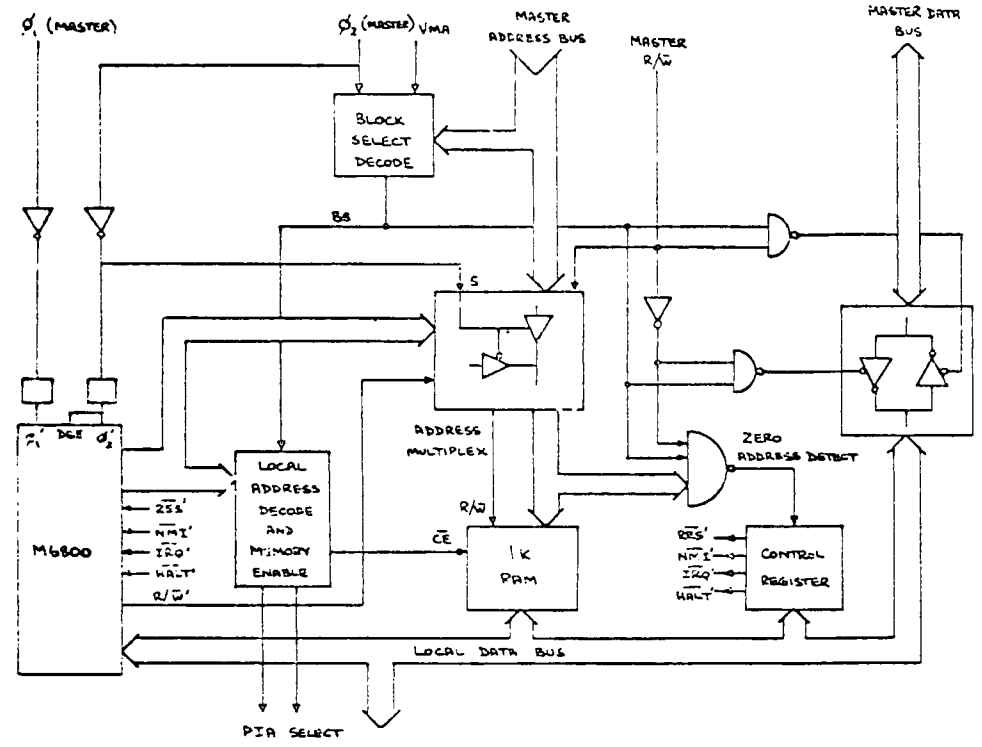
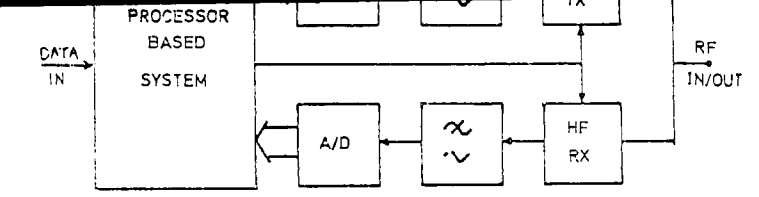
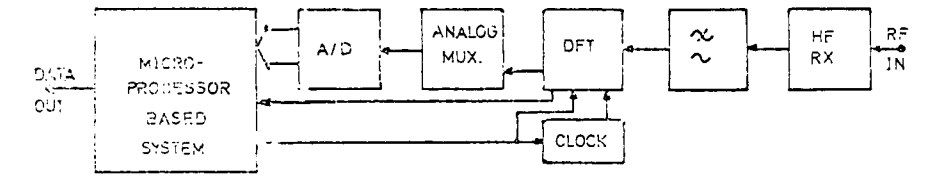


Figure 4. Slave processor system



(a)



(b)

Figure 3. Transmitter (a) and Receiver (b) systems

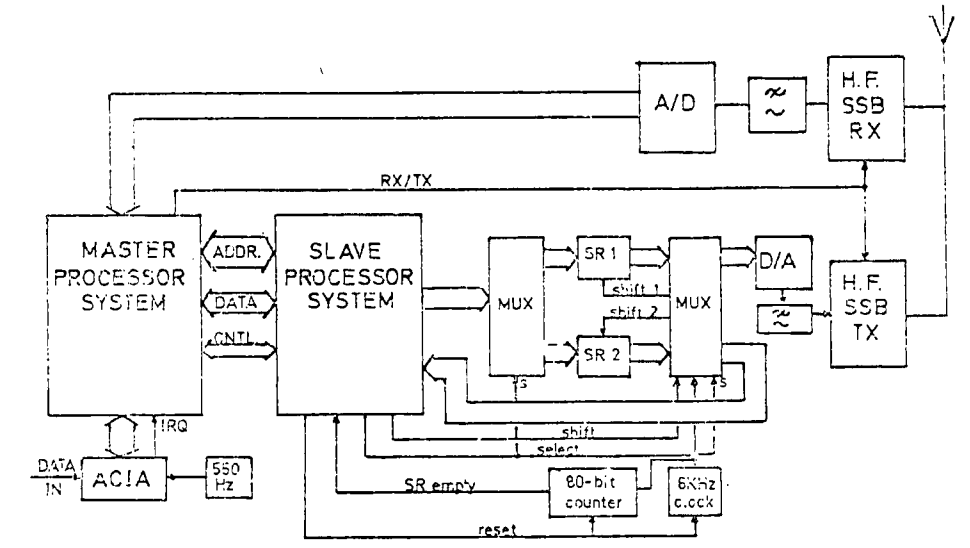


Figure 5. Transmitter system

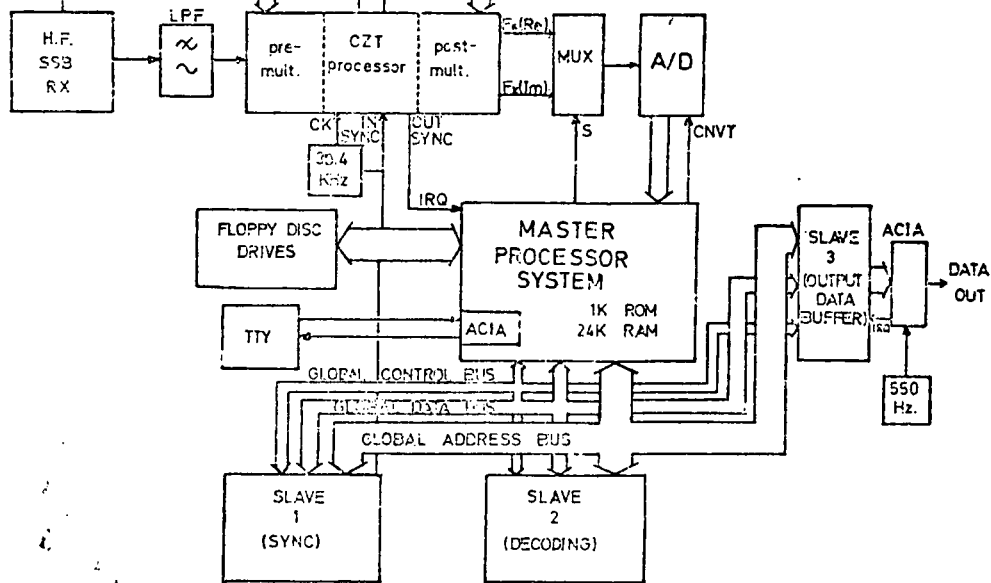


Figure 6. Receiver System

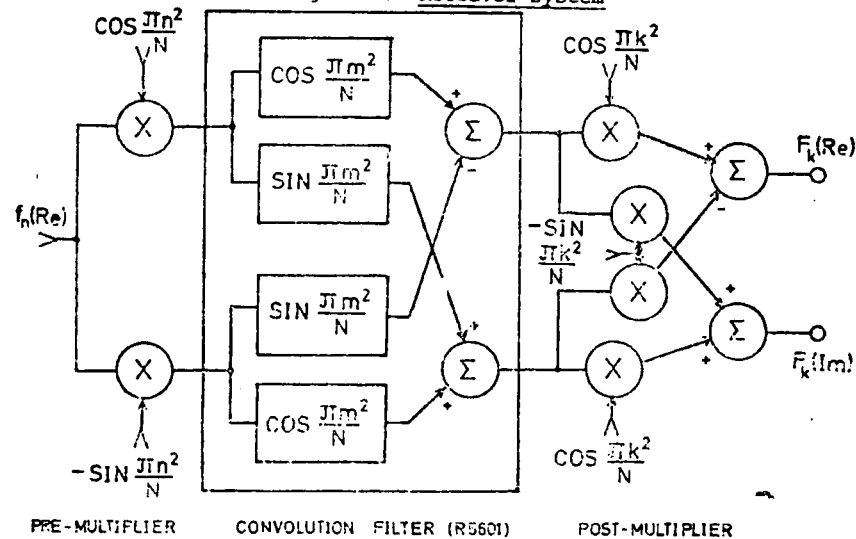


Figure 7. Implementation of Chirp-Z transform

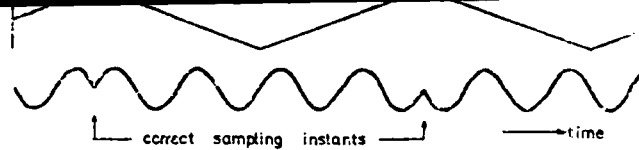


Figure 8. Variation of phasor magnitude over phase reversal sequence

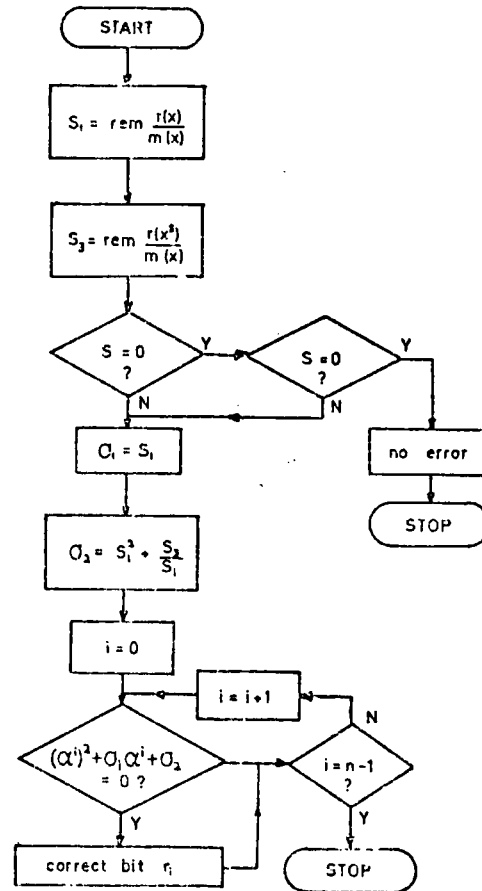


Figure 9. Implementation of decoding algorithm

Microprocessor Controlled Trackside Recorder

D.R. Isaac, C.T. Spracklen,
J.R. Manning

Abstract

This paper describes a new trackside train speed recorder designed at Durham University Applied Physics Department for the Chief Civil Engineer, British Rail Southern Region. The new recorder is controlled by a microprocessor unit and is capable of recording train speed, direction, wheel count, and time of day in an electrical memory contained within the system. The recorder is later transported to a microcomputer base station where the data is offloaded and displayed on a line printer. The new unit is entirely electronic in operation and is powered by rechargeable dry cells. The system is lighter, smaller, and more flexible than any previous designs.

History

The interest in the speeds which trains achieve extends beyond railway staff to some sections of the general public. Usually the interest is in the performance of particular trains over various routes and a record of speeds achieved can easily be obtained on board the train. Speed records in this form are of considerable value to railway operators. To be of use to permanent way engineers it is often desirable to obtain the speed of all trains passing over a particular piece of track. A man stood at the trackside using a stopwatch could obtain this information but such a practice is seldom justified.

The use of automatic unattended recording systems would be ideal for permanent way needs. The limited commercial scope for such equipment has tended to restrict development to the adaptation to train speed recording of standard commercial data recorders. The few recorders that have been developed tend to suffer from the trackside environment leading to frequent mechanical failures.

Specifications

A small, battery operated unit was required which could be placed at the track side to record data over a period of several days. The following parameters were to be recorded for each train passing during that period: time of day, train speed, wheel count and direction of travel. The recorder unit would then be transported to a base computer for offloading the data accumulated during the recording period and for recharging the dry cells. The unit would then be returned to the track side for further recording.

The unit was required to measure train speeds in the range 1 to 150 mph, and to record the speed rounded down to the nearest 1 mph. Trains travelling at speeds not exceeding 1 mph were to be regarded as stationary and not recorded. The time of day was to be recorded at the instant of arrival of a train, to the nearest minute at least, and the system was to be capable of recording up to 200 axles per train. The overall storage capacity of the system memory was to be sufficient to hold data relating to 1000 trains (based on an estimate of a maximum of 250 trains per day passing over a period of 4 days).

To operate the existing recorder, two switches are mounted on the rail head, spaced 66 feet apart (figure 1). The front wheel of an approaching train causes these switches to close in turn. The train speed is recorded by measuring the time interval between switch closures. The new system was required to operate correctly using the existing track switches but should be adaptable to other spacings.

A single base station may serve a number of trackside units, the overall cost being considerably lower than any previous system.

Design and Construction

Recent advances in silicon integrated circuit technology have allowed complex logic operations to be implemented using very small amounts of hardware. Instructions specifying the operations to be performed are stored sequentially in a "read only" memory. These instructions are read from the memory by a microprocessor and are obeyed or "executed" accordingly. Changes in the operation of the system may be effected by simply modifying the instructions (the "software") in the memory. The new trackside recorder is based on such a system; a block diagram is shown in figure 2.

The system is controlled by the microprocessor unit (a Motorola M6802), whose program is contained in a 2 kbyte programmable read only memory. The Peripheral Interface Adapter (PIA) is used to interface the microprocessor to the track switches, the control pushbuttons, and the base station. A crystal controlled clock IC is used to maintain the time of day. Train data is stored in four CMOS read/write memory ICs, which are powered by a backup battery supply in the event of main battery failure. A Liquid Crystal Display, with its associated interface IC, is included to display the current time, or the speed of the last train recorded.

The system software continuously monitors the track switches for a closure. When a closure is detected, the train direction is known, and the second switch is monitored for closure. The time between closures is measured to an accuracy of 1ms, and the train speed is computed by dividing a fixed number (which is proportional to the switch spacing) by the measured time. The number of axles is then found by counting the number of successive closures of the second switch (a 15ms "debounce" time

is allowed for each closure). It is assumed that the train has passed if no switch closures are detected within a time which is inversely proportional to the train speed. After the last axle has passed, the data is stored in memory in the format to be described, and a delay of exactly 15s is allowed before commencing the search for a new train. If a switch closure is detected within this time, the axle count is regarded as erroneous and is stored as a count of '0'. The search for a new train will not begin until both switches have returned to the rest position.

Data for each train is stored in four or five bytes (8 bits) of read/write memory. The fifth byte is used only if there has been a change in hours since the previous train was recorded. Otherwise the hours are not recorded (to conserve memory). Minutes and seconds are recorded using 7 bits each, direction is recorded as a single bit, speed and axle count occupy one byte each. Up to 254 axles per train are permitted. The maximum memory capacity is 8 kbytes, which allows nearly 1000 trains to be recorded. If a smaller memory capacity is sufficient, the number of memory ICs may be reduced accordingly to save cost.

Eight data lines and two control lines are used to interface the recorder unit to the base station for data offloading. A logic '0' on one of these lines (from the base station) indicates that the offloading procedure is to begin. Successive bytes of data are then passed from the recorder to the base station on command until the end of the data table is encountered. The batteries may then be changed, and the system time adjusted by depressing one of two buttons located on the inside of the recorder housing for "slow" or "fast" time advance (the time is monitored on the LCD mounted on the front panel).

The major elements of the recorder unit are mounted on a dual-sided printed circuit board, which also contains the backup battery supply. Several pushbutton controls are available inside the housing: Reset, to restart the recording operation; Test buttons, for simulation of the trackside switches; Slow and fast clock advance, for presetting system clock. A single control marked "time" is available outside the unit. Ordinarily, the speed of the last passing train is presented on the liquid crystal display; if the "time" button is depressed, the current system time is displayed until the button is released.

Base station

The system base station was required to offload, print, and if necessary, recall the data from the trackside unit. It was centred around an AIM-65, a small, inexpensive, 8-bit microcomputer system, with an integral printer. The trackside recorder base station program prompts the user to connect the recorder unit to the computer, then prints out the results in the format shown in figure 3. This format may be altered by simple software changes. A change of batteries will normally be made after the data has been offloaded. It is, however, possible to change the batteries before, even while the recorder is in the

field, as the battery backup will protect any existing data.

Field Trials

Laboratory tests have shown the accuracy of the new recorder to be 0.002% at 1 mph and 0.33% at 150 mph with a 66 ft. switch spacing. A comparison with an existing recorder over the full range of speeds showed the new recorder to be considerably more accurate, especially at higher speeds. accurate then the GMT at higher speeds. Tests on a stretch of (third rail) electrified line have shown the system to be immune to the transient effects often produced in a high-intensity electric field environment. Field tests have been carried out using a variety of switch mechanisms. The Silec oil-damped treadles, normally used with the existing recorders, gave good speed results but spurious axle counts due to the long time required for the treadles to return to rest after closure. Removal of the oil damping cures this problem but considerably shortens the life of the treadles. A magnetic proximity-detector type mechanism again gave good speed measurements but generally produced an axle count greater than the true count due to excessive contact bouncing. A pair of pressure-sensitive microswitches advantages of small size and low cost. Work is now in progress to fully develop this type of track switch.

Future Development

Automatic train identification should be possible if a measurement of axle spacing is made in addition to the axle count. The measured data can be correlated with stored data relating to a number of train types. The train type may then be identified with a high degree of certainty. This information would then be printed at the base station, together with the speed, direction and time.

A multiple switch facility would allow a number of switch pairs to be connected to a single recorder unit in order to monitor several tracks simultaneously. This might be especially useful for monitoring train movements over a number of routes at complex rail junctions. A maximum of four switch pairs would be tolerable without major hardware modifications to the current system. However, considerable software modifications would be necessary for this option.

The power consumption of the current trackside recorder system is approximately 1W; most of this is due to the M6802 microprocessor. A CMOS plug-in replacement for this device has been announced, which should reduce power consumption, and hence battery drain, by approximately 66%. It will shortly be possible to replace the other major components, namely the PIA and the EPROM, with their CMOS equivalents which should then allow the unit to operate continuously for several months without the need for battery recharging.

Conclusion

A trackside train speed recorder based on microprocessor technology has been described which presents many advantages over previous designs. A cost effective system is achieved, together with an overall performance improvement. The new recorder unit can be used with a variety of switch mechanisms and switch spacings, and may be operated from a set of dry batteries for several days. Very favourable results have been obtained from tests in the laboratory and in the field. It is anticipated that the falling cost of microelectronics technology will lead to further improvements and cost reductions in the near future.

