Self-healing concepts involving fine-grained redundancy for electronic systems

PHILIPP SCHIEFER

SCHOOL OF ENGINEERING AND COMPUTING SCIENCES

South Road, Durham, DH1 3LE, UK

Self-healing concepts involving fine-grained redundancy for electronic systems

by

Philipp Schiefer

A thesis submitted for the

Degree of Doctor of Philosophy

2016

Abstract

The start of the digital revolution came through the metal-oxide-semiconductor field-effect transistor (MOSFET) in 1959 followed by massive integration onto a silicon die by means of constant down scaling of individual components. Digital systems for certain applications require fault-tolerance against faults caused by temporary or permanent influence. The most widely used technique is triple module redundancy (TMR) in conjunction with a majority voter, which is regarded as a passive fault mitigation strategy. Design by functional resilience has been applied to circuit structures for increased fault-tolerance and towards self-diagnostic triggered self-healing. The focus of this thesis is therefore to develop new design strategies for fault detection and mitigation within transistor, gate and cell design levels.

The research described in this thesis makes three contributions. The first contribution is based on adding fine-grained transistor level redundancy to logic gates in order to accomplish stuck-at fault-tolerance. The objective is to realise maximum fault-masking for a logic gate with minimal added redundant transistors. In the case of non-maskable stuck-at faults, the gate structure generates an intrinsic indication signal that is suitable for autonomous self-healing functions. As a result, logic circuitry utilising this design is now able to differentiate between gate faults and faults occurring in inter-gate connections. This distinction between fault-types can then be used for triggering selective self-healing responses.

The second contribution is a logic matrix element which applies the three core redundancy concepts of spatial- temporal- and data-redundancy. This logic structure is composed of quad-modular redundant structures and is capable of selective fault-masking and localisation depending of fault-type at the cell level, which is referred to as a spatiotemporal quadded logic cell (QLC) structure. This QLC structure has the capability of cellular self-healing. Through the combination of fault-tolerant and masking logic features the QLC is designed with a fault-behaviour that is equal to existing quadded logic designs using only 33.3% of the equivalent transistor resources. The inherent self-diagnosing feature of QLC is capable of identifying individual faulty cells and can trigger self-healing features.

The final contribution is focused on the conversion of finite state machines (FSM) into memory to achieve better state transition timing, minimal memory utilisation and fault protection compared to common FSM designs. A novel implementation based on content-addressable type memory (CAM) is used to achieve this. The FSM is further enhanced by creating the design out of logic gates of the first contribution by achieving stuck-at fault resilience. Applying cross-data parity checking, the FSM becomes equipped with single bit fault detection and correction.

Declaration

No part of the work described in this thesis has been submitted in support of an application for another degree or qualification to this or any other university or institute of learning.

Statement of Copyright

Acknowledgments

I would like to thank those who made this thesis possible and truly believed in me for the last few years during the course of my PhD. First I would like to say thank you to my doctoral supervisor Professor Alan Purvis. He gave me a great opportunity for conducting my research with freedom and flexibility. I would like to thank my doctoral co-supervisor Doctor Richard McWilliam for his good advice, experiences and friendship. This has been invaluable on both levels academic and personally.

I would like to thank my parents Peter (who could not see the end of this, but whose memory has accompanied me and has given me inspirations) and Lieselotte Schiefer for supporting my vision and being there for me, even in times we do not see eye to eye. My sister Katrin and my uncle Fritz Großmann for their help and support they have given me for fulfilling my ambition. A special thanks towards Dr Friedrich and Ruth Schiefer for helping me in getting a great start into my professional life. Thank you to Bernhard and Else Gaul for leading by example for changing their life to fulfil a life dream despite of the risks it involved.

I would like to thank my friends for their understanding and support through the time of working on my research. The following persons I would like to specially thank: Heiko Jausel for being a friend over the years and his encouragement, even in hard times; Dr Albrecht Carver for showing me that you can conquer the hardest obstacles, personal or professional; Jürgen Böhm for being the same great person, unchanged as I have known you since school.

Dedicated to my wife Iris and my Son Max for their love and understanding they have given me throughout this time and beyond.

"Growing old is mandatory. Growing up is optional." – Carroll Bryant

List of Abbreviations:

µC - micro-controller
ABS - anti-lock break systems (ABS)
ADC - analogue-to-digital converter
ALU - arithmetic logic unit
ASIC - application specific integrated circuit
BB - building block
BISR - built-in self-repair
BIST - built-in self-test
CAD - computer aided design
CAM - content-addressable memory
ccfuse - current sensing and conversion fuse
CCU - central control unit
CISC - complex instruction set computing
CLB - configurable logic blocks
CLS - control logic section
CME - coronal mass ejection
CMOS - complementary metal-oxide-semiconductor
COTS - component of the shelf
CPU - central processing unit
CSP - combined single process
DAC - digital-to-analogue converter
ECC - error correction codes
ECU - electronic control unit
FF - flip-flop
FLB - functional logic block
FPGA - field programmable gate array
FR - fault rate
FSM - finite state machine
IC - input condition
JK-FF - JK-flip-flop
LET - linear energy transfer
LET_{TH} - LET threshold
LUT - look-up table
MBU - multibit upset
MCU - multiple-cell upset

MEMS - micro-electro-mechanical systems
MML - memory-mapped logic
PAL - programmable array logic
PCB - printed circuit board
PLA - programmable logic array
PLD - programmable logic device
PROM - programmable read-only memory
QLC - quadded logic cluster
RAM - random access memory
RISC - reduced instruction set computing
ROM - read-only memory
RTR - run-time reconfiguration
SAFR - stuck-at fault resilient
SAH - stuck-at high
SAL - stuck-at low
SBU - single bit upset
SCO - separated combinatorial outputs
SEB - Single event burnout
SEE - single event effect
SEFI - Single event interrupt
SEL - Single event latch up
SET - Single event transient
SEU - Single event upsets
SMD - surface mount devices
SoC - System-on-Chip
SOI - silicon on insulator
SRAM - static random access memory
SRO - separated registered outputs
TMR - triple module redundant
TRS - temporal-redundant systems
VHDL - Verilog hardware description language

Table of Contents:

List of Figures and Tables:

Chapter 1: Introduction and Overview

1.1. Introduction

In the late 1950s the first bipolar junction transistor was invented at the AT&T Bell laboratories in the United States of America. This invention paved the way for the electronic revolution which subsequently followed by application of this development something which could not have been imagined at that time. In these applications the bulky electric tubes or electro mechanical relays, which were previously used for building all necessary electronic systems, were replaced by a bipolar junction transistor. This invention opened the way for smaller systems and the increased system uptime over the old systems made it a universal part of modern lifestyle. The real push into changing our lives pushed us into the digital age through the next big invention in 1959 of the metal-oxide-semiconductor field-effect transistor (MOSFET). In this thesis the MOSFET transistor is referred as the transistor. The electronic parameters of the bipolar junction transistor worked within the analogue functionality and the transistor was geared towards digital functionality. The newly developed chip industry designed integrated digital circuits on a planar silicon die surface in massive numbers and with standardised logic functionality placed in standardised packages. Since then the driving factor of the chip industry is to reduce the required silicon area per given logic function and, therefore, for that, as a result every 18 months the number of transistors per fixed area doubles. This was defined as a law in 1965 by Moore [39]. The continuous feature size reduction pushes the individual component or transistor into the nano-structure regions allowing even more integration of more individual logic functionality into a single chip. Because of the integration of even more logic functionality into one chip, this made them less likely to experience faults in the overall electronic system. These chips are not insusceptible against faults caused through a number of reasons based on their nano-structure feature size. In this thesis the main focus of faults which are going to be investigated is limited to radiation-induced effects causing temporary and permanent faults within the logic circuit.

In the case of an error affecting the behaviour of the electronic system for counteracting the effects caused by the fault, the system needs to be fault-tolerant or self-healing. Any user of this particular system will experience this circumstance, that, using this electronic system, he wishes that the system can “mysteriously” repair or fix itself. Nature has equipped specimens with the capability of self-healing. Even humans are capable of self-healing of minor cuts through the skin. Due to the requirement of electronic system users the area of fault-tolerant, self-repairing or self-healing electronic systems was originated. Novel electronic system-level concepts were introduced and designed to meet this user requirement for certain applications.

The research work of this thesis is focused on creating a matrix logic structure which is capable of self-maintaining required logic functionality through autonomous fault detection and evaluation with minimum logic hardware overheads.

1.2. Problem definition

The motivation for this research work arose out of an increased requirement of equipping electronic logic systems designed for and implemented on Field Programmable Gate Arrays (FPGAs) platforms with self-maintaining capabilities to counteract the effects of radiation-induced faults in terrestrial systems. Radiation-induced faults in certain electronic components have been a known problem in space application using FPGAs-based systems as system platforms [40]. Continued efforts of the chip manufacturer to increase the amount of configurable logic circuit per square mm of their FPGA chips, are pushing the feature size into even smaller component structure dimensions. Feature sizes of individual components are created out of less than 10 atoms [41]. By reducing the feature sizes of individual components the sensitivity for radiation-induced faults increased dramatically. The increased sensitivity to radiation-induced faults was not only noticeable in space applications, but also in increased numbers of functional upset at terrestrial-level systems [42-44]. The most sensitive areas for radiation-induced faults are the memory chips due to their dense structure and way of storing data [45]. Due to the increase of radiation-induced faults on terrestrial-level electronic systems, these systems are required to be designed with the same fault-tolerance mechanics as space-based electronic systems. This counter action, arising from the type of faults the systems experienced, included fault-masking or system reconfiguration initiated by system-independent checker structures. Both approaches require a trustworthy checker structure which in all circumstances must be able to detect faults and constructed to be per design fault-tolerant. Due to the fact that these logic systems are artificial logic structures mostly created out of the same components as the one which they are checking makes it harder to be fault-tolerant. Also both structures are running on the same die likely to have the same individual transistor fault characteristic. The functionality and task of a system-checker remains a philosophical question and is not part of this research work. The research work of this thesis is focused on the realisation of logic structure with built-in autonomous self-maintenance, minimal checker logic and limited hardware overheads.

Radiation-induced faults on any integrated circuits are the result of activities on the sun, which is the centre of our cosmos and not a planet like the Earth, more like a ball of gas which is less cohesive [1]. Because of this the sun does not rotate like a solid planet, it is more like a process of rotating gas mass generating coronal mass ejection (CME) or ejecting solar flares into space (see Figure 1.1).

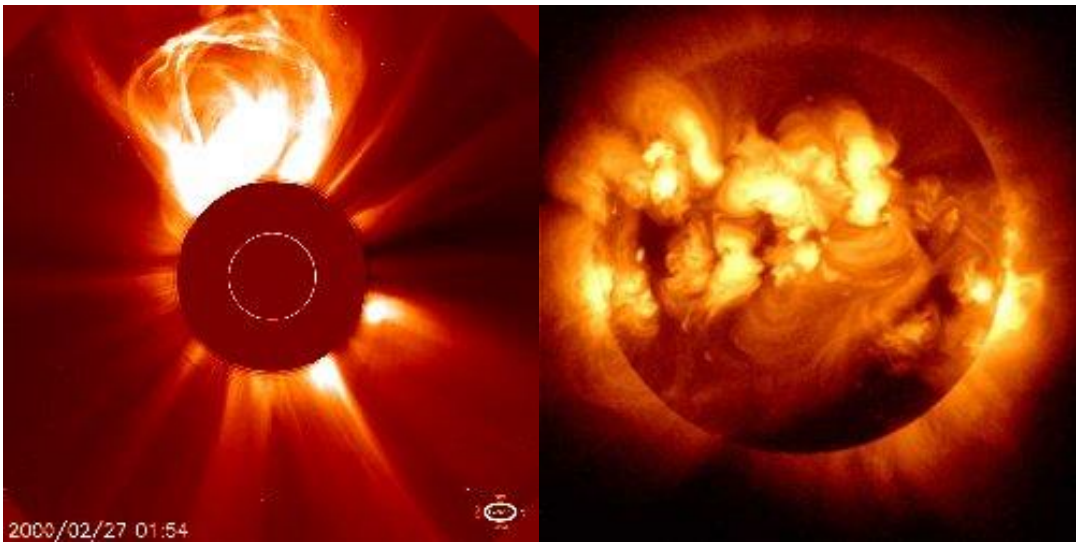


Figure 1.1: (left) Coronal mass ejection and (right) multiple solar flares [1]

CME consists of massive amounts of electrons and protons, which are ejected into space. Released into space they are travelling long distances having an impact on anything they meet, noticeable in integrated circuits as random information corruption is one possibility. The effect can have the nature of a temporary or permanent hardware or data fault inside a digital logic circuit. Solar flares contain a massive amount of photons of all wavelengths, but not all have an impact on everything they meet [1]. The scaling of the complementary metal-oxide-semiconductor (CMOS) into the region of feature sizes close to single numbers atoms structures is resulting out of the continuous efforts of the chip industry over recent decades. The ongoing increase of components per certain die area was predicted by Moore's law which was hypothesized in 1965 [39]. Modern integrated circuit structures are in the region of nanoscale dimensions making them even more susceptible to radiation-induced effects [5, 44]. Faults caused by radiation inside electronic logic systems made out of nanoscale components will then be relevant at ground level and no longer the only fault conditions at high altitude applications [44]. The shrinking transistor structure has been the driving force over recent decades for producing more logic functionality into a given chip. This trend of increasing the logic functionality per given die area was driven by customer demand for better calculation performance of applications. FPGAs offer more active logic components than other chips and give the system designer more possibilities for creating their required System On a Chip (SOC) design. Computer aided design (CAD) tools are available to help the designers programming their required logic functionality into the FPGA chip. By having this flexibility and the capability of constantly reconfigurable logic structure inside an FPGA this made it unnecessary to produce an Application Specific Integrated Circuit (ASIC) with fixed combinational logic. Research done on the effects of transient-induced faults caused by radiation showed that combinational logic is much less susceptible than memory elements [46]. This shows that the logic

functionality controlled within an FPGA by memory elements could be altered with potentially critical effects on the overall system behaviour. These systems require a type of checker for maintaining the integrity of the electronic system or logic structures which can mask faults inside boundaries and fix the effects of any fault.

1.3. Objectives

System-fault identification in regards to temporal or permanent ones requires the means in some cases for a more enhanced system than the supervised system. If the system-checker is required to identify faults down to a gate, intra-gate-connection or interconnect level, these types of system-checker require advanced test capabilities and broad system specific functional knowledge. Each of these requirements can be accomplished with state of the art dedicated digital circuits structures creating logic overhead. This logic circuit is required to have fast response timing for keeping the impact on the overall system behaviour to a minimum or even completely unnoticeable. For some type of systems a pre-defined response time is required to maintain system integrity. This fixed response times; for instance in an automotive safety-critical system, is that the system is supervised and governed by a required alteration of the watch-dog signal within a given time frame. Custom chips are available to be configured through external components for monitoring the required toggling of certain logic signals within a system specific time frame. This is an established method within fault tolerant systems. With this research work the focus is set beyond this established fault tolerant logic structures.
This thesis research has the following objectives:

- The design of a functional logic unit, which combines all of the three redundancy concepts (spatial, temporal and data) to show their combined capability for fault masking and correction.

- Through altering the logic gate transistor level design, the goal was to design a logic gate with fault-masking and intrinsic fault-indication in case of the presence of a non-maskable fault. By constructing logic circuits out of this type of logic gate, a distinction between gate level and interconnect faults can be realised.

- Design a majority voter structure, which is insusceptible to stuck-at faults.

- Self-healing logic structures triggered by autonomous fault detection within given logic cluster boundaries and eliminated by a self-initiated repair process utilising dedicated spare logic units.

- Altering the description of the behaviour of a finite state machine such that it can be transferred into memory-only based hardware platform. This hardware platform offers the advantage of including fault tolerant features, allowing it to be used as a system checker for interconnection faults of a given logic structure.

1.4. Framework of this research work

This PhD thesis is organised as follows:

Chapter 2 introduces the basic concept of an electronic system and its different types of central logic chips which are capable of governing its system behaviour. Comparisons between these different types with regard to radiation tolerance are drawn. Detailed information of different types of field programmable gate arrays (FPGAs) are illustrated and their development shown.

Chapter 3 focuses on the effects of radiation onto electronic systems. Radiation effects are defined as single event effects (SEEs) and within this chapter a range of different types of SEEs are explored. Also their impact on static random access memory (SRAM) based FPGAs. The diverse simulation variety of fault-injection possibilities which can cause effects within electronic systems is discussed in detail.

Chapter 4 introduces the impact of permanent and wear-out related faults within integrated circuits in future chip generation with smaller structure dimensions. By means of even smaller individual component sizes the likelihood of manufacturing fault-free chips will diminish and counter responses with regard to novel fault-tolerant designs are required. Due to their nano-size feature size of individual components chips are going to be more susceptible to radiation-induced faults of a temporal or permanent nature. These radiation-induced faults require fault-masking techniques for avoiding system errors.

Chapter 5 focuses on fault-tolerant systems which are designed for avoiding the propagation of fault beyond system boundaries and its manifestation as a system error noticeable to the user of the system. This can be done by the use of selected logic structures which are capable of masking faults and providing correction at the same time. Applying these logic structures onto a given logic design increases the hardware overhead.

Chapter 6 analyses the different fault-tolerant structures centred on hardware redundancy and spatiotemporal redundant structures. A novel concept of spatiotemporal redundancy for achieving

fault-tolerance and identification is demonstrated which is based on a time-triggered reconfigurable matrix cell.

Chapter 7 introduces the difference between functional and fine-grained redundancy within an electronic logic system. Functional redundancy works on N-sets of functional blocks in this regard in information redundancy. Fine-grained redundancy is working on the gate level by using transistor redundancy. Fault-masking in functional redundancy is being done by majority-voting. Fine-grained redundancy offers the possibility of masking and correcting faults at individual gates. Fault rate analysis of this fine-grained structure shows the fault-tolerance capabilities and by fault occurrence optimisation distinguished fault-behaviour discovered.

Chapter 8 focuses on the approach of mapping finite state machines (FSM) into memory for the benefit of elimination of programmable logic devices or combinational logic. Memory-based systems offer the advantage of better fault detection and correction, due to error-correcting coding of the data stored inside of these memory elements.

Chapter 9 deals with the concept of self-healing within electronic logic systems. The concept of self-healing within any given logic system relies on the adding of spare or redundant logic elements. These elements are used in the case of a fault detected by the system-checker of this system. Electronic systems rely on trying to mimic self-healing on spare elements and a system-checker identifying faulty behaviours. Nature realises self-healing without spare elements and external intervention. Logic gates with altered internal structure are capable of intrinsically indicating non-maskable faults and trigger reconfiguration without outer involvement.

Chapter 10 outlines the final conclusion of this research work and indicates possible ongoing postgraduate research work from the work which has been performed to date.

The appendix includes the simulation programmes written for the different simulations and fault-behaviour analysis.

Chapter 2: Design of electronic systems

2.1. Introduction

In today's world, electronic systems are part of our daily life and they change the way we do things due to their way of offering us more sophisticated solutions. The electronic system application versatility covers all parts of low to highly sophisticated systems. Safety-critical systems, which are highly sophisticated systems like medical systems, are required to function without faults or noticeable impact to the user, or in this case, for the patient. This trend of dependability and system uptime requires novel concepts of system structures and trustworthy system components. Ongoing trends within the chip industry for increasing the transistor count per silicon die area are only possible with ever shrinking size dimensions of the individual chip components. This trend of transistor count increase had been predicted and is reflected in Moore's law from 1965 that every 18 months the transistor count doubles per equal area [39, 47]. Through the reduction in active silicon material forming individual transistors the intrinsic variations of the doping atoms becomes more abundant. Due to this doping variation this will be reflected in higher faults rates during production and over the life-time [47]. This phenomenon makes the functionality of an entire chip dependent on the performance of a single transistor and, in this way, the whole functionality of an electronic system. The internal structure of an electronic system or electronic control unit (ECU) is centred on a type of application-specific logic chip or micro controller. The type to be selected for an ECU depends on the level of complexity of the application.

This central logic chip governs the behaviour of the ECU and it can also be described as a processing engine. It accords with a Ford Motor Company document [48], which identified that 70% of the overall project costs are going to be allocated for the design process of the ECU. This indicates the importance of the correct selection of the processing engine. The choice of correct processing engine or microcontroller for a given project has a direct impact on design process, verification, test and production. Costs for the total life cycle management are also included in this amount of project costs [48]. Depending on the complexity and nature of the usage of the ECU, the design and development is governed by the application specification and environmental conditions.

2.2. Basic structure of an electronic system

Every electronic system is built out of functional units and the level of granularity is defining what fine-grained or coarse-grained functional units are, in accordance to [49] defining the level of fine-grained and coarse-grained functional units as follows. Fine-grained functional units are capable of performing a single logic function on small numbers of bits whereas the coarse-grained functional unit is much bigger than the fine-grained level and contains, for instance, an arithmetic and logic unit (ALU) and, if required, memory. In this regard an electronic system is a coarse-grained functional unit. The basic structure of a coarse-grained electronic system or ECU is demonstrated in Figure 2.1 as a block diagram. Within the block diagram the central block containing the four functional elements of control logic section, memory, input/output section and logic unit is identified. The main block regarding logic complexity is the logic unit or processing engine. The total transistor count of the control unit is less than that of the memory block of the same system. The memory block contains the highest transistor density of all the blocks within the ECU. The blocks controlling logic and input/output sections are both required for data transfer in between different blocks.

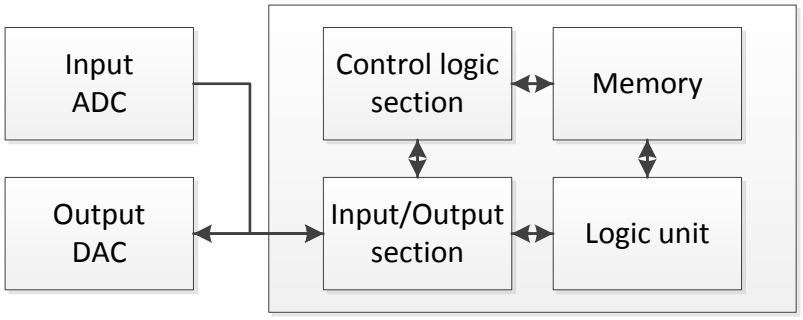


Figure 2.1: Basic block diagram of an ECU with the central block of an ECU containing control logic section, memory, input/output section and logic unit

Through the nature of the digital system interacting with the system in the outside world every electronic system requires an input and output block. Within the input block the analogue signals are being converted into digital signals and in the output block digital signals are being converted into analogue signals if required. It is also possible that purely digital based information is being used. Research done in the area of radiation effects on analogue-to-digital converter (ADC) shows that faults are possible and ADC conversion results are being altered by radiation [50, 51]. The digital-to-analogue converter (DAC) research showed that radiation is capable of altering the results due to bit flips [51]. Due to this radiation-hardened version for space application of the ADC and DAC is available and in use. If the radiation effects have an impact on electronic systems

at ground level on ADC or DAC is not part of this thesis and the information of each converter will be seen as fault-free in this work if required. Nowadays the production of ADC and DAC chips is moving away from being produced with precision-resistor networks in favour of CMOS-type structures. With this alteration of the production method these chips can be produced without costly resistor laser trimming or resistor paste printing. This will make this type of chips cheaper. By the use of CMOS based structures designing converter chips they could now be susceptible to radiation effects. Whether or not these chip types are prone to show effects due to radiation-induced faults at ground level or space is ongoing research work and the effects will depend on the actual feature-size structure of the components. Also such research work is not part of this thesis due to the level of complexity for doing radiation injection into converter chips.

The central block of the block diagram structure of an ECU demonstrated in Figure 2.1 is a logic unit of variable nature which controls the behaviour of the electronic system. The controlling of the behaviour can be done by variable or fixed application description. The variable solution is based on a µC, which is administered by an algorithm-based process description converted into µC direct executable commands. Alteration of the system behaviour can be done through modifying the process description and programming into the memory block of the µC. Another variable solution can be done by logic synthesis into memory-only. By the use of memory-only the electronic system can be designed without a µC or complex combinational logic circuit. The memory-mapped solution of an application is a self-governing memory block controlled by a single addressing register. Within this system structure the outputting of the required output functions depends on the input stimulus. The fixed application description is based on combinational or sequential logic executed by digital logic circuits. The logic circuits can be implemented on a custom-made chip like an ASIC or on a chip capable of creating the desired logic function by programmable logic structures like an FPGA. All of these diverse logic units are susceptible to radiation effects at variable conditions if the unit is not made as a radiation-hardened version of the used chip design. Radiation-hardening can also be performed with the help of logic functionality.

2.3. Central logic unit variation for electronic systems

The behaviour of the electronic systems is defined through the application requirements specified in the system specification for any electronic system. These requirements form the basis of the system action at required times including output release or the specific action on certain inputs. This dependency of input-controlled behaviour changes and generates predefined output following the input stimulus as a FSM. The description of an FSM regarding state processing is defined as if at any given time only one active state in processing exists. Two types of FSMs can be specified concerning the output response, a Moore and a Mealy FSM [52-54]. For a Moore FSM it is defined that the values of the outputs are released only by the state itself and not triggered by the input. The

Mealy FSM output release is described in conjunction of the state and input stimulus [55, 56]. This implementation of a Moore or Mealy FSM can be done on different central logic units. All of the different logic units are based on digital circuit theory [57] and it can be distinguished by working in sequential or combinational logic, where sequential logic means that the logic circuit output not only depends on the input stimulus, but also includes the history of the input stimulus and for this case this type of circuit design requires memory. Sequential logic is also divided into synchronous and asynchronous types. The output of the combinational logic only depends on the current input stimulus.

2.3.1. Microcontroller

Any type of µC or central processing unit (CPU) is a programmable integrated circuit for a multipurpose digital data application and is controlled by stored executable memory information. The memory attached to a CPU supplies executable CPU-specific instructions and data for certain instructions. A CPU contains the general blocks register, control logic section (CLS) and the arithmetic logic unit (ALU). In some cases the ALU is described as logic unit. A basic block diagram structure of a CPU is shown in Figure 2.1. The function of the CPU register is to be temporary data storage. This data within the register can be variable information for current or later use during execution and memory addresses for storing programme-specific execution sequences. The CLS translates the executable instructions out of the memory into commands to control the operation of the ALU, data handling, addressing of the memory and in-/output function.
The embedded functionality of the CPU is hardwired by logic gates in the CLS. This means that every single executable instruction of the CPU is hardwired within the CLS. These logic circuits control the behaviour of the CPU and the logic hardware size depends on the number of instructions of the CPU. There are two types of CLS execution styles, which are utilised within different CPU designs, the complex instruction set computing (CISC) and the reduced instruction set computing (RISC). The difference between these two CLS types is in the logic circuit complexity of the CLS. The logic functionality of CISC-type CPUs requires more logic circuits within the CLS for creating instruction specific low-level operations sequences. CISC based CPUs are designed in a way that a single instruction executes several low-level operations in a given sequence to perform a specific function of one instruction. This can take usually several clock cycles of the central CPU clock until execution has been finished. In contrast, RISC based CPU executes a single function with one instruction in one clock cycle. This is due to the less complex structure of each instruction of an RISC-CPU. The functional complexity to perform a certain task is put into the program, which is stored in memory, then into complex decoding and controlling logic hardware within the CPU.

The ALU of a CPU performs arithmetic and logic operations. For performing these operations the ALU reads and writes registers which are controlled through the CLS. The complexity and transistor count has increased since the development of the very first produced CPU in silicon. Following this trend of increased transistor count Moore's law, published in 1965, predicted that every 18 months the transistor count for a fixed die area doubles [39]. Due to the constant increase of transistor density this is only being accomplished by individual feature size reduction. These dimensional reductions of the individual transistors make the CPUs more susceptible to radiation-induced faults. CPU chip producers had to alter the design of their products to make them resilient against radiation-injected faults [43, 44]. The most vulnerable components within a CPU are any memory elements, e.g. are registers, cache memory or memory-based pipelines. Radiation effects can cause a bit flip or latch-up altering of the stored information and this can result in system lock-ups or incorrect system responses [43]. CPU supplier in the past had problems with radiation-induced faults and they had to alter their chip design. For instance the 5^{th} generation SPARC64 from Fujitsu had its design altered in a way that 80% of the 200,000 latches had been converted to have parity checking to protect the CPU against radiation faults at ground level [43, 58]. This processor type had been fabricated in 130nm silicon on insulator (SOI) CMOS [58] and today's CPUs are fabricated in even smaller feature size.

Radiation hardened versions of CPUs are available for specific customers and applications. Since certain logic circuits of CPUs are protected against radiation-induced fault effects on memory circuits, the capabilities of built-in self-repairing is not part of any CPU. Today's CPUs advance into multi-processor application or multi-cores on a single chip, which enables the core to be deactivated if, within one of these multi cores, a hardware-related fault condition occurs.

2.3.2. Application specific integrated circuit

Application specific integrated circuits (ASIC) are customised chips for a single purpose only. The functionality of the logic function is tailored for the customer's need and is fixed by means of a design freeze. By using a custom chip for this particular application means that the use of industry-standard integrated circuits for the customer has been excluded. This offers the advantage of cost reduction at the size and complexity of the printed circuit board (PCB) and individual component quantity. Another advantage of an ASIC is that it is optimised for a single purpose only and this will reduce the ASIC chip parameter area, delay and power consumption against a FPGA by ~21 times, ~4 times and ~12 times respectively [2]. ASICs due to their optimised solution can be faster than CPU-based solutions. Because an ASIC is a customised chip, a combination of digital, analogue, and micro-electro-mechanical systems (MEMS) is possible. This possibility of combining different subsystems within one chip offers solutions otherwise not achievable as a component of the shelf (COTS). The main costs for utilising an ASIC as a solution within an

electronic system are the design costs which are due to uniqueness of the chip and compared against the production costs. Because of this the amortisation is only given in high volume production. Radiation hardened versions of ASICs can be designed on customer request and defined by their specification. Due to the fact that ASICs are produced on similar material and production steps as COTS chips, this makes any type of memory element within the design susceptible to radiation-induced faults. ASICs require the same techniques for radiation-hardening by design like other COTS chip-based applications. If self-healing is required from the customer or their application these capabilities within the ASIC logic circuit have to be conceived during the design phase and the logic structure cannot be altered after manufacturing by configuration by means of programming. The longest design phase for an ASIC is the full-custom design, because every circuit is designed for the specific customer application and no industry standard blocks can be used [3]. The shortest design phase is with a gate-array design. A gate-array design approach uses pre-fabricated gate-array structures where the final metallisation mask for the interconnection links between individual components is missing. The design of this ASIC only requires the generation of the different final metallisation masks on top of the gate array structure.

2.3.3. Field programmable gate array

FPGAs are pre-fabricated silicon chips offering a sea of logic functionality, which can be by means of electronic programming, transformed into any kind of digital circuit or system [3]. The internal structure of today's static memory-based FPGAs (commonly specified as SRAM-based FPGA [3]) is demonstrated in Figure 2.2. The structure of an FPGA is equally balanced between functional blocks and interconnection blocks. Interconnection blocks establish the connection between functional blocks for the application design and will be regarded as interconnection throughout this thesis. The contrary connection definition is the intra-gate-connection, which establishes the connection between individual logic gate transistors placed in close proximity. The configuration of the FPGA functionality by programming is controlled by SRAM bits and divided into configuration bits for the interconnection and selection of the logic functionality. The functional block of RAM within an FPGA is part of the logic structure and the total chip area of memory cells for a given FPGA can be 50% to 90% of the total chip dies area [2, 6]. Modern FPGAs are transformed in complexity and logic functionality from the first programmable array logic (PAL) or programmable logic array (PLA). Both PLA and PAL internal logic structures are demonstrated in Figure 2.3. The PLA structure is presented in Figure 2.3a and the PAL structure in Figure 2.3b. Both are using programmable input selection sections, which are then feeding into an AND plane. For the PAL structure all the output signals of the AND section are being fed into an OR gate. Whereas for the PLA the selection of the AND gate output signal at the AND section is being realised by programming the connection or selection of the required input digital signal feeding

into an OR gate [3, 5]. In comparison the structure of the SRAM-based FPGA is of a matrix-type layout where at the cross points alternating functional blocks have been placed. Also today's FPGAs are equipped with freely associated input and output pins in accordance with the needs of the circuit design and PCB design [3].

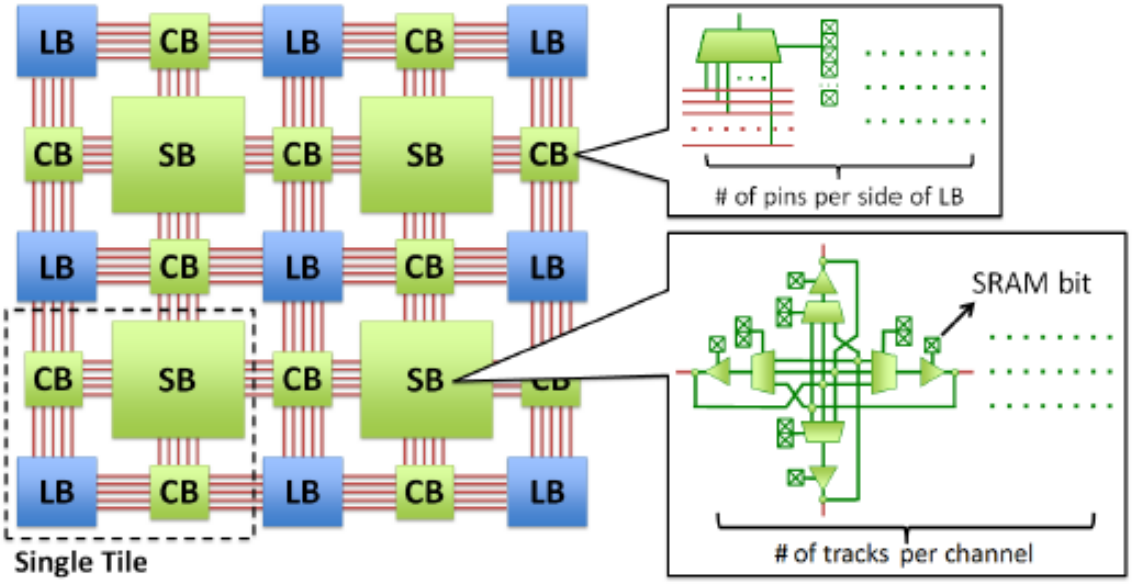
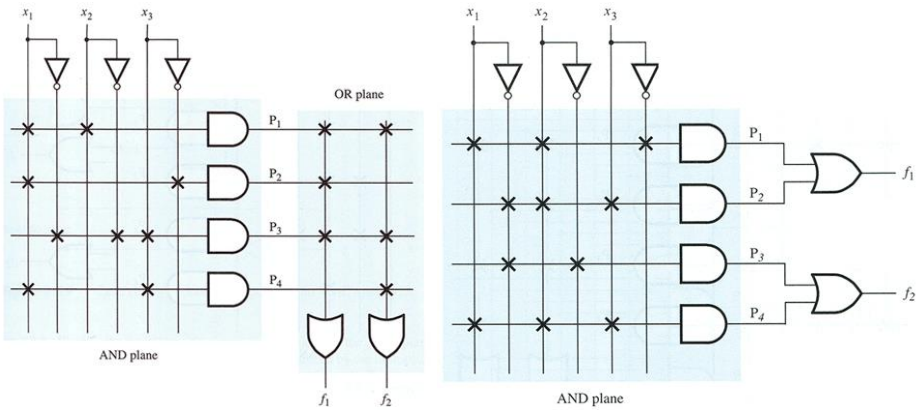


Figure 2.2: SRAM-based FPGA with two connection blocks (CB), one switch block (SB), one logic block (LB) forming a single tile [2]



(a) (b)

Figure 2.3: (a) PLA and (b) PAL architectures of the internal section structure [3, 4]

The different functional blocks are designed for a specific functionality which is flexible enough to cover a wide range of logic alteration done by programming alteration, due to this wide range of logic versatility within a functional block of a common FPGA. This logic versatility gives the

FPGA the flexibility needed to fabricate almost any specified digital logic circuit envisaged by its user.

The functionality of the switching block, which is placed between a set of connecting blocks, is to establish the application required routing of the interconnection between different connection blocks [2, 3]. All the interconnection routing done on an FPGA chip is done by the switching block and connecting block. Both are controlled by means of SRAM elements and are essential for creating logic circuits on an FPGA chip. In the case of faults within the routing blocks, or interconnection structure, an alteration of the logic outputs will reflect this. Distinction if the fault has been caused by a logic block or any interconnection block is limited. A test pattern applied onto the interconnecting block would reveal the existence of a faulty condition. The same external testing of the functionality of a logic block has to be executed to identify faulty behaviour. External testing is required for this chip structure to reveal and find faults within its structure. Would it not be better to have an FPGA or other type of logic structures with intrinsically built-in fault detection capability as nature offers for their efforts in regard to self-healing?

Similar routing circuitry like the switching-block structure for generating the interconnection is in use for the flexible connection of the external input and output pins to matrix-style internal access style structures within any given FPGA chip [3]. The connection block is located around the logic block and makes the necessary connection between needed input/output pins and between logic blocks. The rule of generating interconnections between logic blocks is set to link logic blocks together which are located within close proximity to each other. Every logic block of an FPGA contains a cluster of basic logic elements and memory look-up tables, which can be used to provide customised logic functions [2]. Historically the first FPGAs designed were based on erasable programmable read-only memory (EPROM) and electronically EPROM (EEPROM) [59]. Today the most commonly used memory types in modern technology FPGAs are flash RAM, static RAM and antifuse approaches [3]. All of these different memory concepts are used for having the configuration data stored of the required logic functionality and interconnection setting with the internal FPGA structure in mind.

Some of the modern day FPGAs are designed in a way that during operation of the chip, the configuration of the logic functionality and interconnection setting can be altered without interfering with the running operation and execution. This is called run-time reconfiguration (RTR) [60-63]. With this type of FPGAs the application designer is in a position to modify the active logic structure to perform a different application or alter the structure because a fault in a block requires a logic structure reconfiguration during operation. This flexibility offers the possibility to reconfigure a faulty FPGA logic structure during logic operation to continue working correctly and the exterior does not notice a change. The actual reconfiguration of the configuration data programmed into the FPGA requires an external device where alternative configuration settings are stored or a remapping system.

2.3.4. Mapping logic into memory

Any individual or subsidiary electronic systems behaviour follows the rules of an FSM and is defined by its state diagram of this application. The state diagram can be transferred into a digital input/output sequence, which is then transferable into a truth table. This truth table represents all possible digital input stimuli with associated output responses. The information stored inside the truth table can be transferred into a memory unit. An example for transferring a JK-flip-flop into memory is demonstrated in Figure 2.4. The state transition table displayed in Figure 2.4(a) shows all the different possible states of the JK-FF which can exist and the associated output data. The coding and replacing of the state labels has been done in Figure 2.4(b) and can be seen as a truth table. Within Figure 2.4(c) a data reduction and combination in matching memory structures, produces the final memory data representing the memory-mapped JK-FF.

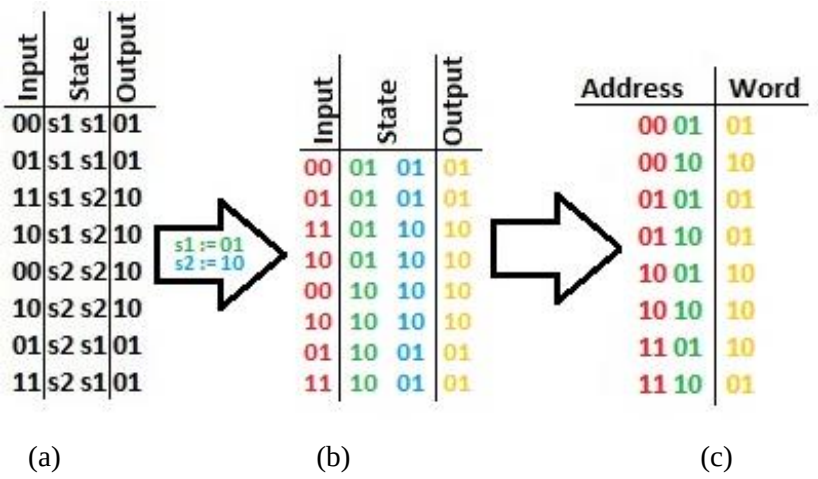


Figure 2.4: JK-flip-flop state transition table transformation into memory; (a) state transition table; (b) state transition table including coded replacement of states and can be seen as a trues table; (c) memory data created out of data from (b)

With this step of transferring the system behaviour converted over into digital sequences, the mapping into an appropriate memory unit can be done. The memory-mapped state transfer offers the advantages of minimal control logic and an error-correctable memory block. The use of an error-correctable memory block is because of the effects that radiation can have on memory of causing bit flips and in this way state transition alteration. Error-correcting memory can detect this type of data alteration and fix it. The block diagram of a memory-mapped controller can be seen in Figure 2.5. By comparing this to a µC structure (see Figure 2.1) similarities can be identified. Both comprise a memory block and a logic block and in these two parts both are comparable. But the logic circuit amount is different for both solutions. The µC calculates the required transition out of the input stimulus using executable code stored in memory. In contrast, the memory logic

application adaptation contains all the necessary information within its unique memory-addressing structure, which is triggered by input stimulus. The memory for the memory-mapped solution can be linear addressable memory [64, 65] or content-addressable memory [33]. Linear addressable memory has the disadvantage that undefined input related addresses can upset the sequence of the state transition or retaining the system in one state.

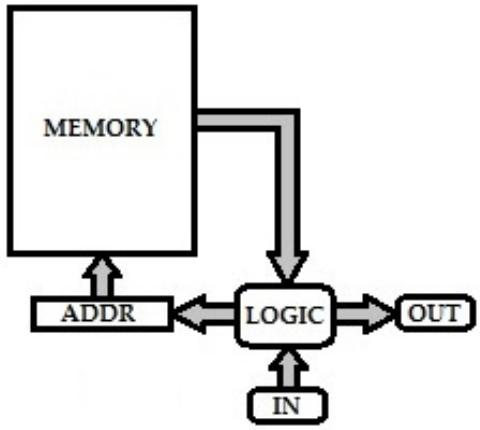


Figure 2.5: Block diagram of a memory-mapped FSM

For accessing of the data stored in memory and in this way the simulation of the state transition behaviour is done through unique addresses. The addressing of a specific memory location, as part of the FSM state transition, is a combination of a unique state counter and input stimulus, which forms the unique address-pointer. The data stored at this location contains the information for the next state transition and output information. Fault-tolerance with regard to logic faults in the logic unit can be handled by redundancy. Faults within the memory data require error-correction hardware and in the case of non-fixable faults, rearrangement of the unique memory information structure. This cannot be done by the system itself and requires external offline rearranging if possible.

2.3.5. Comparison of the different logic units

Four different central logic units usable for an electronic system are CPU, ASIC, FPGA and memory-mapped logic, and they can be compared against each other. The main focus of the comparison will be on the possibility of self-healing in the case of permanent hardware faults and the capability to handle radiation-induced faults. The other key factors such as power consumption, signal delay and chip size are not part of this comparison done in this research work forming the foundation of this thesis. This is because fault-tolerance and self-healing capability are relevant for electronic systems, which are exposed to radiation-induced system alterations.

General comparison of the different controller types:

- CPU: COTS standard types in most cases do not possess memory cell protection such as error correction codes (ECC) or designed-in radiation hardened circuit design. Highly specialised CPUs for example like the 5th generation SPARC64 from Fujitsu [6, 43] ,the LEON processor [66] (which is an open source implementation of a SPARC V8 processor adapted in an FPGA) or the IBM Power6 [67] are three examples of processors which are custom-made chips and produced in low numbers and which contain fault-tolerant solutions with regard to ECCs or redundancies. Fault-tolerance regarding permanent hardware faults can only be accomplished by redundancy of the whole CPU or at fine-grain redundancy at gate level. Modern multicore processors handle faults within one core in the way that this core gets deactivated. This approach eliminates the need for redundancy at any level within the chip.
- ASIC: the whole circuit design has to be done in radiation-hardened design and ECC has been applied to memory elements of the processor. This type of ASIC is only produced in low numbers because it will only be used in low volume applications like satellites. Any subsequent necessary circuit changes are not possible after design freeze. For handling permanent hardware faults, the ASIC needs to be equipped with redundant structures at functional or gate level. The redundant structure is put in place during the design phase of the chip and every fault possibility has to be envisaged at this state. If a fault occurs within the switching circuit between redundant elements the approach for fault repair cannot be done and the fault cannot be fixed.
- FPGA: can be described as a sea of logic. This sea of logic can be configured by means of programming in accordance to the specification, which governs the logic structure programmed inside the FPGA. During the entire design phase the intended logic structure can be altered because a fixed and final hardware structure will not be produced. The design is embedded inside a programming file, which can be also altered during the life-time. This offers flexibility to the designer to alter the logic circuit layout to incorporate radiation hardened logic circuit structures at the appropriate locations throughout the chip design phase. Or even afterwards in uptime of the electronic system by reprogramming the target FPGA chip on the fly. This is possible through run-time reconfiguration of the FPGA configuration [60-63]. The Xilinx Virtex-6 contains ECC capabilities for the configuration data programmed into the configuration memory [68]. With this feature of the Virtex-6 alteration of radiation-induced faults can be detected and corrected.
- Memory-mapped logic: the whole logic performance regarding input dependant output and system transition is mapped into memory. A comparatively small control logic circuit is governing the input and output activation of the memory block by creating the unique

address-pointer accessing the stored data. Due to the reduced logic count and low level complexity of the control logic redundancy for fault-tolerance can be applied. Fault-tolerance at the memory level as for instance memory addressing logic faults can be coped with on a reduced scale by expanding the necessary memory and addressing register. The fault-tolerance is limited due to the unique memory addressing in conjunction with the input stimulus.

In Table 2.1 an evaluation of the different controller types against the system requirements for creating a fault-tolerant system is shown. The different system requirements are:

- Application fixed: with this point the capability of alteration of the application created within each controller is evaluated. The adaptation of alteration even after design freeze or during the life-time is needed to maintain an up-to-date system with can meet customer requirements.
- Reconfiguration: the capability of altering the logic circuit structure during operation. This point shows how the system can be adapted in case of hardware faults.
- Hardware requirement: the evaluation of hardware structure present within the evaluated controller type. The key is to have flexibility within a given logic structure offering the required logic functionality without having too much unused hardware resources.
- Memory requirement: how much memory is required for storing the application specific code data, configuration data for hardware arrangement and general data storage during runtime.
- SEU tolerant: indicates if the controller type has SEU tolerant features present for fixing radiation-induced bit alteration in memory.
- Logic interconnection complexity: is a general evaluation of the way the individual logic functions are linked together. The key is short interconnection links between logic functions without too much unused hardware overhead.
- I/O flexibility: evaluates the flexibility of the input and output connection with regard to designing a PCB with this controller type. The key is to give the PCB designer the possibility of arranging the chip interconnection with the best routing arrangement.

By evaluation of the different points of Table 2.1 the best controller type for fault-tolerant systems can be found. Each point of this table is evaluated for each controller type, including finding the possible optimum fit for fulfilling the system requirement for each point and these points are coloured green within the table. For finding the best match of controller type meeting the system requirements the total number of fulfilments within the table are counted.

	Application fixed	Reconfigurable	Hardware requirement	Memory requirement	SEU tolerant	Logic interconnection complexity	I/O flexibility	COTS
micro-controller	no	no	high	optimised	no	fixed	fixed	yes
FPGA	no	yes	high	high	yes	design specific	flex	yes
ASIC	yes	design specific	optimised	optimised	design specific	design specific	design specific	no
memory mapped logic	no	no	optimised	optimised	design specific	low	fixed	no

Table 2.1: Evaluation of the different controller types against system requirements

Concluding by evaluation of Table 2.1 shows that for this comparison the FPGA can fulfil the requirements of fault-tolerance. This is because it covers a broad spectrum of attributes for a fault-tolerant system design by its general chip design and makes it a perfect platform for fault-tolerant systems.

Table 2.1 also shows that memory-mapped logic (MML) is the second best solution for a fault-tolerant system design. MML indicates in some points a better solution than an FPGA chip. The two main points which make the MML controller second are COTS and SEU tolerance. The COTS point is because of the hardware requirement of the address-pointer. The fulfilment of the built-in SEU tolerance depends on the application specification. If ECC memory is used for an MML the system has built-in SEU tolerance.

2.4. Development of FPGAs

The development of the FPGAs started in the middle of the 20th century because of the demand for generating logic designs within a chip with a faster turnaround [5]. The first programmable logic was the logic mapping into memory with the help of read-only memory (ROM) and followed by programmable read-only memory (PROM). Both types are one time programmable logic array normally used for storing micro-controller executable instructions. The EPROM evolved out of this as the next generation of programmable logic array. All of this adaptation used N number of address inputs to implement a logic function stored in memory and through the number N it also defines the required and addressable memory size. The size of the addressable memory was the disadvantage of this application and the next developments were PAL and PLA with AND and OR gate arranged in alternating specific logic gate sections or planes (see Figure 2.3). In some chip

designs this logic structure also included D-type flip-flops. This gate arrangement offered the flexibility to programme combinational and sequential logic structures [5].

Both chip designs PLA and PAL were limited by their internal structure only to allow fixed connection between input pins and logic gates. The demand of flexibility within the internal connection of a logic chip required an alternative and programmable interconnection arrangement. By adding the typical crossbar design for the interconnection to the chip structure the flexibility regarding connection was resolved. The added crossbar to the current logic chip significantly expanded the size requirements for the die [3]. The introduction of the static memory-controlled interconnection switches and logic configuration reduced the die size and increased the flexibility of this type of logic chip. Xilinx was the first company introducing the FPGA design, which is still used today. It was built around configurable logic blocks (CLB) [3] (demonstrated in Figure 2.2 and Figure 2.8). These type of devices used bit stream programming to configure logic or interconnection structures, comparable to the devices with static memory [5]. The development of the FPGA is centred on the capability of programming the appropriate configuration into the memory controlled switches. Historically, the development included EPROM, EEPROM, flash, static RAM and antifuse configuration structures [5]. In modern day FPGA designs only the memory technology flash, static RAM and antifuse is applied [3]. Out of these three the static RAM is amongst the most used technology there is. All the current types of FPGAs are fabricated on CMOS technology and all developments of scaling can be utilised.

2.4.1. SRAM-based FPGAs

The SRAM programming technology is used by Xilinx, Lattice and Altera in their devices [5]. The advantages of SRAM or static RAM technology lies in the capability of indefinite re-programmability [3]. SRAM or static RAM cells are designed in the way demonstrated in Figure 2.6 and they are used in interconnection and implementing logic functionality throughout the entire FPGA chip structure.

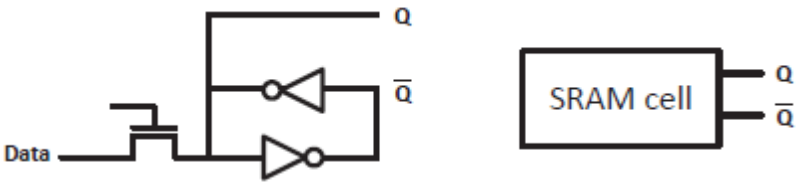


Figure 2.6: SRAM or static RAM cell structure for programming one bit [5]

With one SRAM cell the interconnection switch gets controlled to connect the required crossbar lines together. So each crossbar within an FPGA contains for each possible connection capability an SRAM cell. This means that a high number of SRAM cells are within one die location and SRAMs are susceptible to radiation effects. An alteration of a single SRAM cell affects the switch and in this way the interconnection of logic functions or logic gate intra-gate-connection. The gate level is constituent out of the individual transistors and intra-gate-connection. Intra-gate-connection in this regards is associated with connecting the individual transistors oriented in close proximity together for the required logic functionality. The design of one interconnection switch is demonstrated in Figure 2.2 (right-hand side bottom small figure) and an example of possible connection creatable with this switching structure is demonstrated in Figure 2.7 [6].

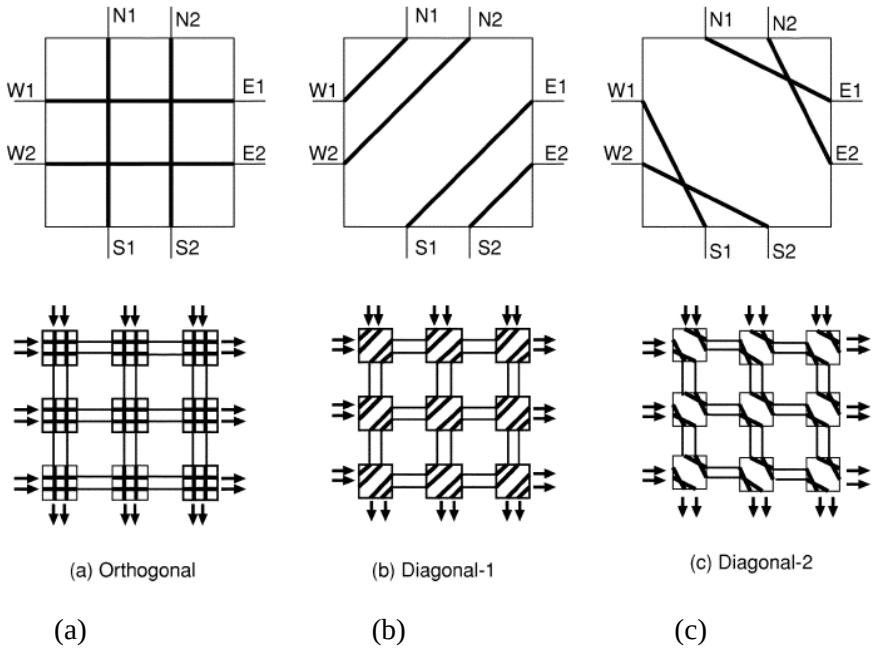


Figure 2.7: Example of possible interconnection switching configuration; (a) orthogonal, (b) one type of diagonal, (c) another type of diagonal interconnection [6]

For implementing logic structures in an FPGA the use of look-up tables (LUT), multiplexer (MUXs) and FFs are configured in the way for simulating the required logic function. These elements can be found in each of the CLBs within an FPGA. A block diagram of an FPGA capability of a switching element is described and the internal structure of a CLB illustrated.

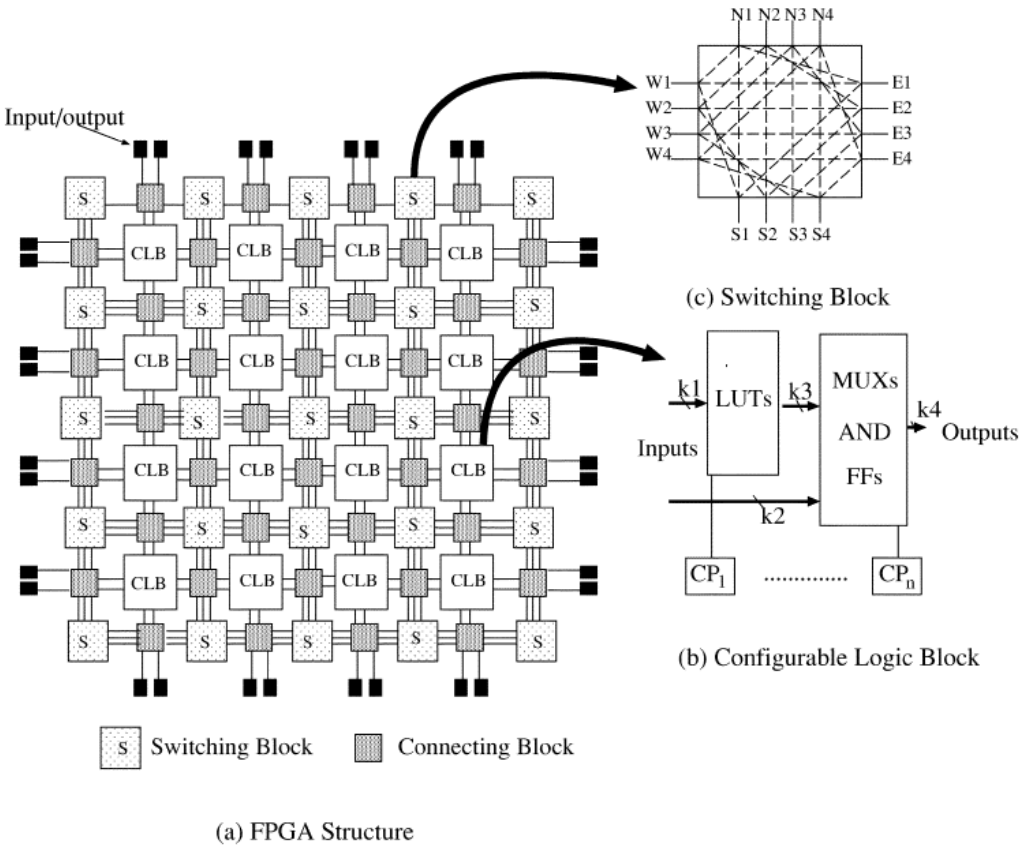


Figure 2.8: (a) Block diagram of a SRAM based 4x4 CLB element structure with interconnection elements building the FPGA structure; (b) block diagram of the inside of a configurable logic block (CLB) [6]

Due to the nature of the SRAM based memory the information stored is not permanent and has to be reprogrammed every time at power up of the FPGA-based system. This makes it necessary to have permanent storage alongside the FPGA or modern chip containing permanent storage on the chip, like flash memory, inside the chip. This type of permanent storage makes SRAM based FPGAs inefficient [3]. SRAM based FPGAs are fabricated in CMOS technology and this technology is susceptible to radiation-induced faults. The ongoing reduction in transistor scaling increases the sensitivity to terrestrial related radiation-induced faults.

2.4.2. Antifuse-based FPGAs

Antifuse-based programmable switches can be implemented in FPGAs. The advantage of this technology lies in the positioning of the fuse underneath the gate electrode at each of the transistor as a programmable controlled element of an FPGA. This technology does not require additional circuity added to each programmable switch. Two methods of creating the fuse are possible. One is based on using oxide nitride [3] and the other is a metal-to-metal-based [5] antifuse. The metal-to-

metal-based approach can be done by positioning an insulation material like amorphous silicon or silicon oxide inside two metal layers [5]. A comparison between the SRAM and antifuse-based programmable switch regarding of chip structure is demonstrated in Figure 2.9. In Figure 2.9(a) the normal transistor layer structure is shown and in Figure 2.9(b) the one with the antifuse layer underneath the gate is illustrated. This technology could not be done with a standard CMOS process due to the need of additional process steps and masks. The mechanism of programming or altering the conductance of the fuse requires significant changes within the material of the fuse. This makes the adaptation of scaling within new chip designs a challenging and costly undertaking [5]. Because of this the newest CMOS advantages cannot be utilised in antifuse-based FPGAs. The technology "kilopass" changed the way the required antifuse process steps became part of the standard CMOS production if the 2T bitcell design is being used [69].

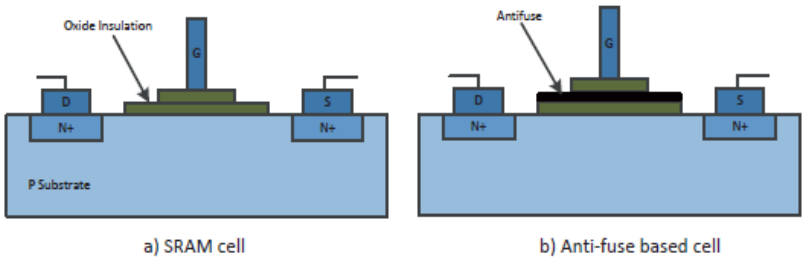


Figure 2.9: (a) SRAM vs. (b) Antifuse-based programmable switch of an FPGA [5]

For programming the antifuse transistor a high voltage is needed for breaking down the antifuse and forming a conductive connection. This approach requires large programming transistors on the die for handling the high programming voltage. Also the antifuse programming requires a special programming device and programming has to be done before the chip gets mounted on the PCB. The chip production yield of antifuse-based FPGAs chips can be expected to be successfully programmable with confidence in the order of 90% yield [3]. This programming yield number indicates that a manufacturing test cannot detect every possible defect in a given chip [5]. Due to the only one-time programmable fuse the programmed design in the FPGA cannot be changed. This makes this type of FPGA insusceptible too radiation-induced faults altering the information on any programmable switch. Because of the non-volatile stored switching information, the device can function directly after power-up and no external non-volatile memory is required for reading the programming stream.

2.4.3. Flash-based FPGAs

Flash-based programmable switches of an FPGA belong to the family of non-volatile memory and on power-on the FPGA system is already configured. This is similar to the antifuse-based FPGA devices. Due to the all-time constant programming state of the switches an external flash memory is not required as for the SRAM-based FPGA. The main difference between antifuse-based and flash-based FPGAs is in the number of re-programmability cycles. Antifuse-based programmable switches are only programmable one time and the flash-based switches can be reprogrammed a limited number of times. For instance the Actel ProAsic3 can be re-programmed 500 times [3]. In comparison the SRAM based programmable switches can be programmed an infinite numbers of times [5]. The functionality of the flash-based or EPROM based programmable switch is based on a gate that floats above the transistor. Onto this floating gate a charge can be stored and as long it stays above the threshold voltage level of the distinct high level this switching transistor will remain in the programmed state [7]. A valid high level can be maintained on the floating gate for up to 10 years [7]. Two types of flash-based memory structure can be distinguished, NOR and NAND gate type structure and both are illustrated in Figure 2.10 [7].

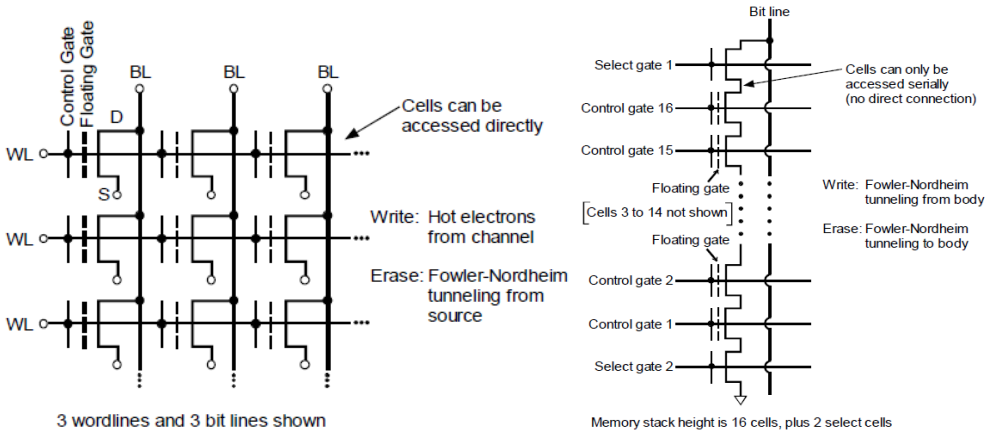


Figure 2.10: Cell architecture for NOR (a) and NAND (b) gate design [7]

Research done on the influence of radiation effects on the floating gate of a single flash-based programmable switch showed that no effect can be noticed for the low level. Effects on the high level can be noticed and a drop of the stored charge can lead to a drop below the threshold voltage of the high level. This would lead to alteration of the configuration of the stored design within the FPGA [7, 70]. In the case of the use of flash-based programmable FPGA systems the external flash based memory is no longer required and in these systems is eliminated. A reprogramming of an apparently faulty system by self-initiation is not possible. The system needs an external initiation

bit stream for reprogramming. FPGA manufacture offers chips where a designated flash storage is located next to the SRAM-based FPGA die within the same chip package. This combination allows the storage of the bit stream information of the specific design to be read at power-up initiation and the chip can function according to the required application [5].

2.5. Summary of chapter

In this chapter the function of an electronic system was analysed in a way to identify the key functional block of it. The key functional block of every electronic system is the central control unit (CCU). This is due to the fact that the control of the application is governed by its logic structure and represents a significant amount of hardware. This central hardware needs to be insusceptible to faults. Faults of any type within this hardware affect the behaviour of this system and the selection of the fault tolerant strategy is significant. System fault tolerance is achieved through redundancy concepts, which are described in detail in chapter 4. Hardware level strategies for increased fault tolerances in regards to configuring the logic design of an FPGA are investigated with this chapter. The individual logic gates are not included within these different approaches and instead require alternative solutions.

Different types of CCUs are being used throughout electronic systems worldwide and the four main types of CCUs in use are defined within this chapter for further examination. The focus of this examination was to reveal the best possible CCU platform for a fault-tolerant design of an electronic system. Through Table 2.1 the evaluation identified the FPGA to be the optimum match for these requirements. An FPGA offers the most requirement matches for a fault-tolerant system. This is due to the fact that the sea of logic and the ability to be reconfigured during execution are useful features of the FPGA for designing this type of electronic system. The second optimum CCU was the memory-mapped logic and this is due to the fact that the origin of the FPGA was memory-based logic adaptation. The origin of mapping logic into a flexible logic structure began with the use of memory replacing discrete logic gate structures by using memory-based platforms for the first attempt to design circuit logic behaviour in a quicker and more compact hardware structure. Before the use of memory each logic circuit had been designed out of discrete logic gates.

Chapter 3: Radiation effects on electronic system components

3.1. Introduction

Introducing and utilising electronic systems in a wider spectrum of applications also exposes these systems to a broader range of environmental conditions. One of these conditions is high energy particles generated from the suns radiation-induced fault caused by one high energy neutron particle can strike a single transistor of a logic gate within a chip. This particle strike of the transistor can alter the logic state on this transistor. In this case a soft error has occurred in this particular part of a chip and this could cause an upset of the behaviour of the electronic system. This upset could manifest itself in system malfunction behaviour or the system can mask the fault at a functional boundary. Fault-masking has to be within a defined logic block dependence on the logic circuit design of the electronic system. In the past, this type of system upset was associated with high altitude electronic systems such as satellites orbiting around the Earth or passing through space. In this application, specific logic circuit design solutions were applied to cope with soft errors within defined circuit system boundaries. The chip industry has continued to scale back the individual components of a given die into even more minor dimensions and this has triggered a negative effect on increased susceptibility against radiation-induced upsets within electronic systems even at terrestrial levels. Now soft errors can be experienced at terrestrial level similar to high altitude systems. This effect is especially noticeable in these high-density circuit chip structures within static or dynamic memory [71]. FPGA contains large number of memory cells used for configuration or data storage, which are used for logic function simulation or as memory banks. All of these memory cells are at risk of being altered by radiation-induced faults. Soft errors in combinational logic have not been of great concern so far with the current level of technology. But the ongoing trend of size reduction of individual transistors will make the combinational logic structure on a given chip susceptible to soft errors. In this regard, the whole chip and so the trustworthiness of the electronic system decreases and advancements to the logic design has to be put in place to regain it.

3.2. The sun as source of the radiation effects in electronic systems

The sun in our galaxy represents the centre planet. But in fact the sun is not a solid planet as the Earth is or the other planets surrounding the sun. It is more a ball of hot gases with a nuclear fusion reactor in its centre. In the core of the sun the temperature is 15 million degrees Celsius [72] sufficient enough to maintain this fusion process running for billions of years. The energy generated in the core of the sun needs 179000 years to get to the surface. The temperature drops below 2 million degrees [5] and the final surface temperature is around 5505 degrees Celsius [73].

This surface temperature is still sufficient for particles to escape the gravity force of the sun and travel through the outer space. The released particles of the sun are protons, electrons, alpha ions and heavy ions. Also the sun ejects millions of tonnes of material during a CME into outer space, which creates solar winds. All these particles are bombarding the planets surrounding the sun and satellites within space.

The magnetic field of the Earth is formed from the inner core of the Earth into outer space until it encounters the effects of the solar winds. This magnetic field protects the Earth against the solar winds generated from the sun. When the Earth's magnetic field comes in contact with the solar winds, the magnetic field is compressed by the effect of the high energy particles. The magnetic field of the Earth, which is not facing the sun, is being elongated into space. Both of these effects on the Earth's magnetic field are being illustrated in Figure 3.1. The magnetic field of the Earth has the shape of belts around the Earth inner core and these magnetic belts extend into outer space. The American astrophysicist James Van Allen was the first to predict their existence in 1958. In Figure 3.2 both belts of the Earth's magnetic field are shown. The outer belt of the magnetic field of the Earth is capable of trapping high energy (0.1-10 MeV) electrons, the inner belt of the Earth's magnetic field traps high concentrations of low energy (range of hundreds of keV) electrons and high energetic protons with energies exceeding 100 MeV [74].

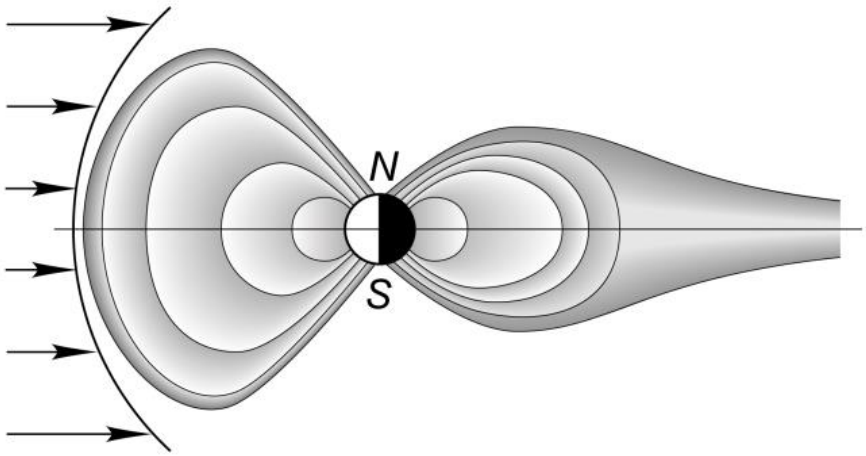


Figure 3.1: Solar wind and Earth's magnetic field interaction [8]

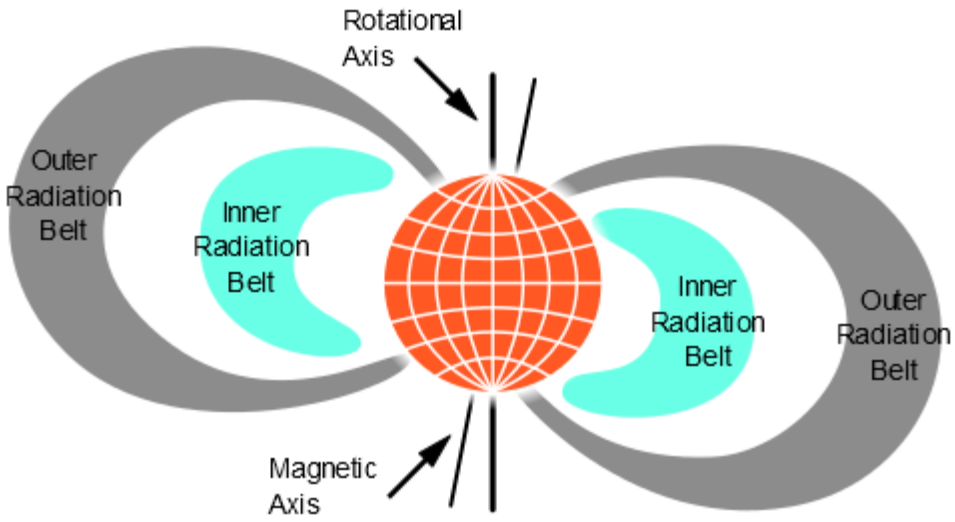


Figure 3.2: Van Allen radiation belts of the Earth magnetic field [9]

3.3. History and impact of single event upset effects on electronic systems

A single event upset (SEU) happens when a high energy particle or electro-magnetic radiation collides with a sensitive component of an electronic system and is capable of altering its data condition. Historically the first effects of high energy particles altering electronic equipment happened through the detonation of nuclear bombs above ground level around the world between 1954 and 1957. During these nuclear tests the first effects of unexplainable anomalies on electronic monitoring equipment happened and could not be explained. The equipment indicated faulty behaviour but no hardware fault could be identified and the term soft-error was associated with this phenomenon. Soft-errors were also encountered during the first satellite space explorations. In 1978, the phenomenon of altered behaviour within electronic systems due to soft-errors in integrated circuits were explained with the presence of alpha particles in the packaging material emitted by traces of uranium and thorium impurities of these chips [44, 75]. This soft-error effect caused by a chip housing contamination was first reported by Timothy C. May and M.H. Woods. The material of the integrated circuits had been modified in a way that no more radiation was eradiated and the phenomenon of soft-errors caused by contaminated packaging material was dissolved. James Ziegler described in 1979 the mechanism that high energy particles from space can cause a soft-error within an electronic system at sea level [44, 76].

The range of effects caused from soft-errors can be of transient and permanent manifestation in a chip structure of an electronic system after a hit by a high energy particle. The impacts of SEUs on a given chip normally are of a transient nature and randomly distributed over the chip die area. Permanent impacts to the affected circuit structure are possible in some cases. SEUs can occur within a memory cell or a logic latch [10] and according to [10, 77] the SRAM soft-error rate will

increase by 8% per chip generation. Figure 3.3 shows the graph of the soft-error rate per chip generation. The current design of FPGAs is primarily designed with programmable SRAM based switches and LUTs. Soft-errors are playing a significant role in the fault sensitivity of FPGAs. The radiation-induced soft-error can cause a bit flip within a memory cell. The detection and fixing of this type of memory fault can be done with the help of ECC. The FPGA chip manufacturer Altera offers in some of their FPGA designs an automatic cyclical redundancy check for correcting configuration bit alteration [78]. Soft-error introduced alteration of information on a flip-flop is harder to detect and because of this is impossible to correct.

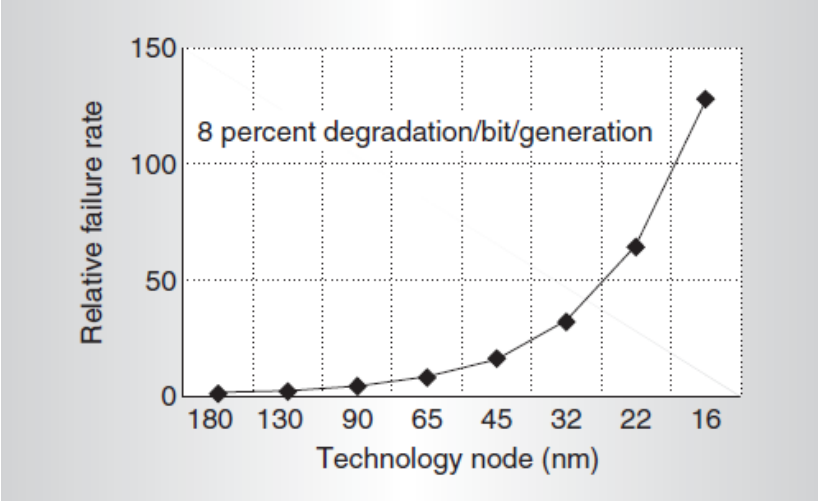


Figure 3.3: Soft-error rate per chip generation (logic and memory structure included) [10]

3.4. Definition of single event effect

A single event effect (SEE) is caused by a single radiation event, like a high energy particle, striking the silicon die of a chip. At the die location where the particle hits the silicon die a charge is generated along the track of the high energy particle. This charge created within the silicon die can affect the chip structure in close proximity and alter the stored conditions on transistors or only a single transistor [16]. In this case a soft-error at this single transistor has occurred and this is defined as single bit upset (SBU). If no permanent damage has occurred and in the case of new data getting written to this individual transistor of a memory cell, the transistor is capable of storing the new data. If no permanent alteration to this particular transistor happens it will continue working correctly after the incident. In the case of a collision of a high energy particle with a high density transistor structure, the created charge at the track can affect multiple transistors in close proximity. This case represents a multibit-upset (MBU) within a high density transistor structure which can be found in memory elements of systems. SBUs and MBUs can happen in an electronic system located in space, like a satellite or within an electronic system at terrestrial level such as an ECU of

a car. The radiation sources affecting the two electronic systems at these altered altitudes are different. Radiation-induced SEEs in electronic systems in space can be due to three variation of particle types: ionised particles (which are part of the natural galactic background), solar particles and high energy protons trapped in the Earth's Van Allen belts (see Figure 3.2). The terrestrial SEEs are caused by neutrons and protons created through the collision of cosmic particles with an atom in the Earth's atmosphere.

The likelihood of a fault happening in a silicon chip structure due to an SEE depends on the chip technology used and the radiation intensity. Different chip technologies have different susceptibility to SEEs and this susceptibility also depends on the linear energy transfer (LET) of each particular particle [16]. The susceptibility of the different chip technologies is specified by the LET threshold (LET_{TH}). The total amount of radiation over time can cause long-term damage or degradation effects to integrated circuits. This type of radiation-induced fault is relevant for space application, where electronic systems are exposed for years to continuous striking by radiation particles. An example of this is a satellite on a mission to Mars which takes many years to complete. The constant injection of radiation particles into a chip of the system will show degradation effects on the silicon based structural elements of the chip over time in space, which is due to the nature of total radiation effects in relation to time and space. This aspect of long-term radiation-induced fault-types is not part of this research work.

3.4.1. Types of SEEs

SEEs can be divided into two categories; transient and permanent causing faults within a given electronic circuit. A transient or non-destructive SEE is a fault where the information is stored or passed through a type of component in which it can be stored and the information is altered in a way that it is changed until new information updates this altered information. A permanent or hard SEE is a fault affecting an active component in a way that the SEE changes the information on an active component in such a means that the new information cannot be altered by any stimulation [16].

Transient SEEs within an electronic system are the following ones according to [76]:

- SEU or SBU affects the information stored on an active electronic component in a temporal means and not as permanent information alteration. Any new information can be stored on the affected component afterwards.
- Multiple-cell upset (MCU) means that at least two or more memory cells of latches are affected by the event.

- MBU indicates that at least two or more bits of a data word are altered by the radiation event.
- Single event transient (SET) affects the signal level for a short time within a combinational logic signal path. This can be an interconnection between two logic gates. If the SET gets stored within a memory element like an FF at the right time an SEU has occurred within this electronic system.
- Single event interrupt (SEFI) occurs if the system malfunctions due to a bit flip within the system critical memory element.

Permanent SEEs within an electronic system are the following ones according to [79]:

- Single event latch-up (SEL) occurs by means of turning on the parasitic bipolar transistors between n-well/p-well and substrate within a silicon die. In a CMOS intrinsic bipolar junction transistors are being created due to their manufacturing process, forming n-well/p-well combinations inside the die. If these formations are forming a parasitic n-p-n-p structure in this way a PNP and NPN transistor are structurally stacked next to each other. Through this stacking a thyristor-like device between V_{cc} and GND rail has been created and with a satisfactory voltage level affecting both transistors can be turned on and maintain this condition until a power-cut. A high energy particle can trigger this thyristor-like device and will create a short circuit between V_{cc} and GND inside the chip. This effect occurs with significant current flow. The current flow usually results in the destruction of the chip and only a power-down of the chip can resolve this condition. Latch-up resistant design alterations for CMOS chips are in place to prevent SELs from happening.
- Single event burn-out (SEB) is caused through an increased current flow between Drain-Source paths of a Power-MOSFET. This current flow will destroy the component. If the power of the component gets discontinued or interrupted in time the component or chip can be saved from burning out.
- Single event gate rupture (SEGR) happens through a higher gate current level than the one specified for a Power-MOSFET. This current flow could cause the destruction of the gate-dielectric of the Power-MOSFET and it can be cleared by means of a component power interruption.

3.4.2. Linear energy transfer function

If a SEE occurs within affected silicon chip is depending on the LET level caused by the particle, which is affecting the silicon chip. The level of LET within a given material depends on the mass and energy of the radiation particle and within the type of material it is travelling [16].

The level of the LET can be calculated as follows:

$$LET = \frac{1}{\rho}\frac{dE}{dx} \qquad [5, 16] \qquad \text{(Equation 3.1)}$$

In equation 3.1 the expression E is the energy of the radiation particle, dx the unit of the material and ρ is the density of the material. The LET unit is defined in $MeV\frac{cm^2}{mg}$. The LET threshold (LET_{TH}) defines the minimum level of LET created within a certain material by a certain type of radiation particle, which will create enough energy, that it has an effect on the components. The cross section (σ) defines the number of upsets within a given area based on the number of particles the chip device gets exposed to.

3.4.3. SEU in relation to sea-level

When a radiation particle enters the Earth's atmosphere it can collide with Earth's atmospheric atoms and will produce a cascade of secondary radiation particles. These secondary particles produced by this collision are pions, muons and neutrons. The average timespan before decaying of these pions and muons are in the region of nanoseconds and microseconds. Where the neutrons average lifespan before decaying is in the region of 10 to 11 minutes and, in the case of a collision with another atmospheric atom, another cascade of secondary radiation particles are created [17].
The flux of these secondary radiation particles fluctuates with the altitude and location to the Earth. Due to the small thickness of the Earth's atmosphere within the outer stratosphere, the flux of the secondary radiation particles is small and increases to its maximum value at 13km altitude against sea-level. This point is also known as the Pfotzer point. Thereafter the flux of the secondary particles decreases until sea-level. A rough approximation of the flux level at a given altitude can be calculated with the following equation:

$$Flux\ increase\ over\ sea\ level = e^{\left(\frac{119.685H - 4.585H^2}{136}\right)} \quad [17] \quad \text{(Equation 3.2)}$$

With equation (3.2) the flux level can be calculated and H defines the altitude over sea-level in kilometres. The level of flux of secondary particles for Denver Colorado, USA elevation is 3.5 times higher than at sea-level. A typical airplane which flies at an altitude of 10km will expose the electronic systems to a 228 times higher-level of flux than at sea-level [17].

3.5. SEE impacts on SRAM-based FPGAs

The key advantage of an FPGA is the possibility to configure the interconnection and logic resources freely and as often as required for the application and even during operation alteration of these FPGA resources is possible. This configuration of the FPGA happens by means of programming bit sequences into configuration memory. The configuration memory within an FPGA can be of an SRAM, antifuse or EEPROM memory function and their functionality has been described in a previous chapter. Each of these different memory types reacts differently to radiation-induced upsets. FPGAs-based on SRAM-type configuration memory are the most commonly ones used as application platforms for today's electronic systems. SRAM-based FPGAs are amongst the most susceptible to radiation-induced upsets among the three memory types used in FPGAs. In this dissertation the primary focus of FPGA-specific configuration type memories lies on the SRAM-based configuration controlled FPGAs. This is because they are the most commonly used FPGAs in today's electronic systems due to the reconfiguration capability and the application of choice for fault-tolerant systems.

The number of configuration memory cells of an FPGA represents the vast majority of the total number of memory cells implemented on a given FPGA chip [80]. SEEs effects in an SRAM-based FPGA can affect the data within the configuration memory or the data within the user memory of the logic circuit (in these cases flip-flops, look-up tables or memory cells). These two types of memory-related faults due to SEEs causing effects within an FPGA are the primary focus of this dissertation. SEEs within the clocking logic can be possible and the effect can be that the entire FPGA design is turned off [80]. The effects on the clocking logic will not be further investigated in this research work because the logic and memory part of the FPGA is the primary focus of this thesis.

3.5.1. SEE impact on configuration data stored in SRAMs

Radiation-induced faults on the configuration information stored in the SRAM of an FPGA can be affected in a way that information is altered. This altered configuration information will affect the intended routing and logic resources of the design programmed inside the FPGA. Due to the size of

the configuration memory of an FPGA it is possible that the radiation-induced faults are within a part of the configuration memory, which is not being used by the application. In this case the fault has no effect on the logic structure running on this FPGA. The SEEs effect on the routing part affects the interconnection between different logic blocks of the design created within an FPGA and a bit flip within this sensible part of the FPGA has a severe impact on the interconnection between logic functions. The fault in the interconnection configuration memory part of the FPGA can manifest itself as disconnection of a logic interconnection, creating a new interconnection between logic blocks or bridging two interconnections together [5, 80]. An example of an altered interconnection within the switching matrix is demonstrated in Figure 3.4. In this case a different signal coming from another logic part is routed to the same logic unit instead of the intended logic signal.

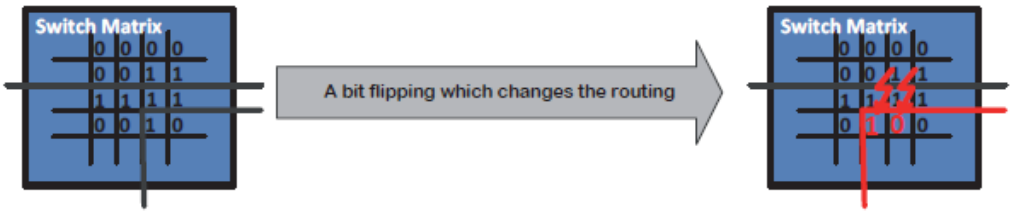


Figure 3.4: SEE-induced alteration of the interconnection within a switching matrix [5]

3.5.2. SEE impact on user data stored in SRAM

Today's modern FPGAs contain two types of memory elements within the logic part of the user application. The first memory type is a standard memory structure, which is based on SRAMs to store data of the user application. The second memory type is a memory-based LUT in which output values of logic function are transferred and stored in LUT memory and read upon request. By the use of this step the logic functionality is transferred into SRAM-based LUT memory eliminating the need of implementing every possible combinational logic functions within this logic block. In the case of a bit flip within the LUT the intended logic function is altered to give a result from an alternative and incorrect logic function. This fault condition is demonstrated in Figure 3.5 where the output value of an AND gate stored in the LUT gets altered by means of an SEE into a NAND gate output value.

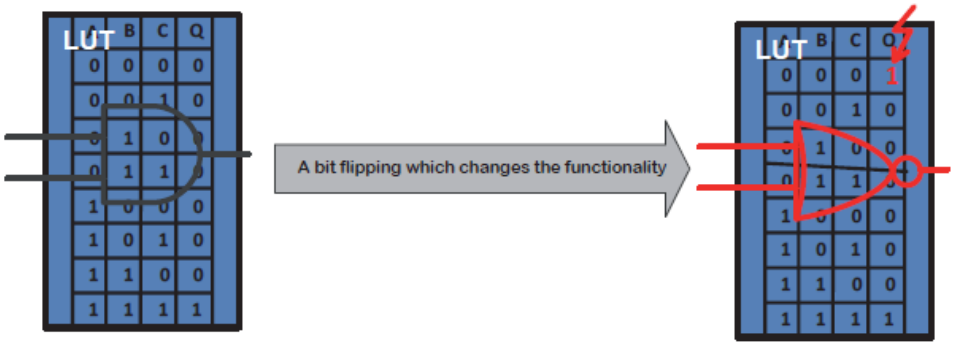


Figure 3.5: SEE alteration of the stored logic function data to another logic functionality [5]

3.5.3. SEE impact on the user logic

How the configurable logic within an FPGA gets used depends on the logic design requirements and specification of the user. In general the logic can be used as combinational or sequential logic design and both designs are likely to be affected by SEEs, most likely by SEUs and SETs. The effect of an SET manifest in combinational and sequential logic if the glitch is caused by the SET is captured in a memory element. Sequential logic designs contain embedded flip-flops, which are acting as a memory element in the logic circuit for storing past values. The manifestation of an upset caused by an SEU or SET within a sequential logic circuit requires a type of in circuit memory element in which it is getting stored but only if the enabling line of the memory element gets activated to store the current data at the timing around the glitch. The storing of the altered information in the memory element needs to be coinciding with the temporary glitch affecting the logic circuit otherwise it will be without any effect on the system [75]. In this way the circuit timing and the delay caused by the combinational logic are important features for the possible manifestation of SEEs altered information having an effect within this system. Memory elements within a combinational or sequential logic circuit can be altered by an SEU if it is hit by a high energy particle directly.

3.6. Simulation of SEE faults in an electronic system

During its life-time it is quite possible that a given electronic system will be exposed to any number of SEEs, which can be noticed or can happen unnoticed by this system. For the verification of the fault-handling capability of an electronic system by putting it into space or waiting for naturally caused upsets is not timing and budged wise. Within both natural test set-ups the fault-causing conditions are unpredictable and the amount of possible upsets cannot be controlled or predicted. By the use of natural radiation sources of any type the test coverage cannot be predicted. A possible simulation of using a radiation source for directly bombarding a chip with naturally caused

radiation is to use Californium-252 (Cf-252). By using Californium-252 a constant flux rate for a given time is guaranteed [81]. The disadvantage of using Californium-252 is that the chip has to be bare die without the housing for being subjected to the radiation from the source in close proximity to the die.

Impacts of SEEs onto an electronic system need to be appropriately simulated for testing the fault-tolerance of a given electronic system for a defined and controllable impact on this system. Different test set-ups are possible for simulation of an SEE on electronic systems which is illustrated in Figure 3.6. Most of the shown fault-injection methods in Figure 3.6 work on a coarse-grained level by injecting a fault in a way that a logic function by itself gets altered or inputs are being changed. The fault-injection method based on logic equation simulation can work on fine-grained simulation in which the individual transistor of a gate gets simulated and a fault gets applied onto an individual transistor. In this way the effects of radiation on single transistors are possible and the fault effects on the whole system can be evaluated. Simulation of the whole logic gate is also possible. But for this research work the fine-grained simulation of individual logic gate transistors has been chosen because, by this method, the impact of redundant transistors in the case of transistor faults can be better evaluated.

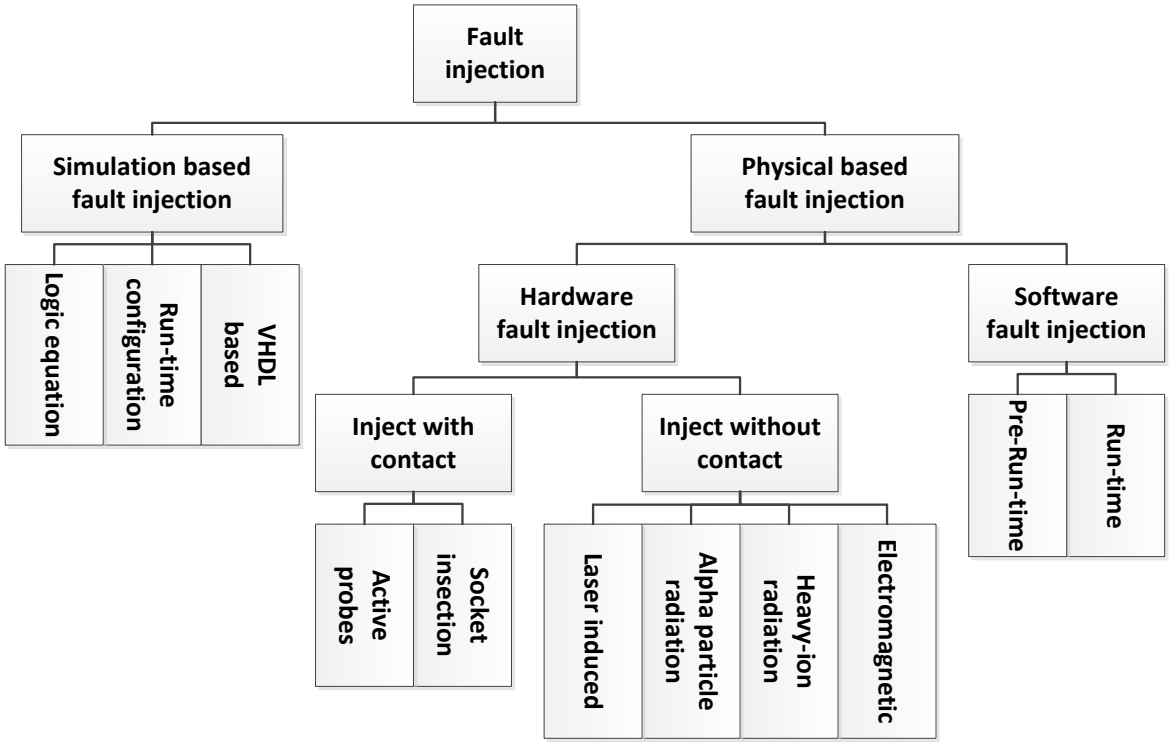


Figure 3.6: Overview of fault-injection methods [11]

For the analysis of the fault-tolerant efficiency designed inside in an electronic system the ability to repeat and reproduce the two factors is required for certification testing. By repetition the key focus

is put onto the ability of repeating the experiments exactly or with a very high level of precision over and over again. The main target of reproducing is the capability to regenerate the same results over and over again. This can be achieved by exactly controlling the experiment and in this way the radiation effects on the whole system.

3.6.1. Simulation-based fault-injection

The use of simulation tools during development and design of an electronic system offers the possibility of injection of faults into the application model and the simulation of the application within a computer reveals the response of the injected fault. The injection of faults into a simulated application works in altering logical values during application simulation without having any target hardware available. This injection approach works on system-model simulation or on emulating hardware.

3.6.1.1. VHDL-based fault-injection

The fault-injection simulation on Verilog hardware description language (VHDL) can be done by using two approaches for simulating fault-injection onto a target circuit. The first approach for the fault-injection technique is to use the simulator command tools. By using the command tools it is possible that during runtime of the simulation signals and variables of the model can be manipulated. This technique does not alter the VHDL code of the application circuit. The second fault-injection technique uses direct VHDL code manipulation in a way that it alters the model by adding saboteurs or diversifies the individual model of a single component.

3.6.1.2. Fault-injection with means of run-time configuration manipulation

The fault-injection approach, which is done by run-time configuration, takes advantage of hardware prototyping. This hardware prototyping is normally done on an FPGA-based hardware emulator. This offers all the advantages of run-time reconfiguration needed for this fault-injection approach. The use of an emulator for the synthesis of each fault-injected design description has to be synthesised, placed and routed. The disadvantage of this type of approach is that the simulation time increases with the size of the design and number of faults which have to be injected. By using the approach of bit-stream modification after the synthesis, placement and routing of the design the test time can be reduced. For performing a fault-injection simulation only some of the bits in the bit-stream have to be altered.

3.6.1.3. Fault-injection into logic equation

For the fault-injection approach into logic equation each of the individual logic gates of the logic structure used for the electronic system are descripted with the help of individual logic equation. Each of the logic gates are being split into pull-up and pull-down network as demonstrated in Figure 3.7 for a NAND gate. In this case the pull-up network represents the functional side of the gate, which creates a connection to the high-side or V_{cc} rail and the opposite for the pull-down network. The operation of the standard NAND gate circuit illustrated in Figure 3.7 is described by the logic expressions $\overline{X1} + \overline{X2}$ for pull-up network and $X1 \cdot X2$ for pull-down network. Out of these two expressions the overall output signal is determined according to [36] out of four possible logic states. Logic state one is the low output active, which in this case defines that the pull-down network logic equation is the true one. The logic state two is the high output active, which means that the pull-up network logic equation is the true one. Both these logic states are producing valid output results and are the normal working states with regards to accurate output signal. The third logic state is where pull-up and pull-down networks are off and the output is in an undefined state. For this state both logic equation pull-up and pull-down are not true. In contrast the fourth state is where pull-down and pull-up networks are turned on and create a short circuit between Vcc and GND rails. All these definitions of the different logic gate states are defined within Table 3.1 [36].

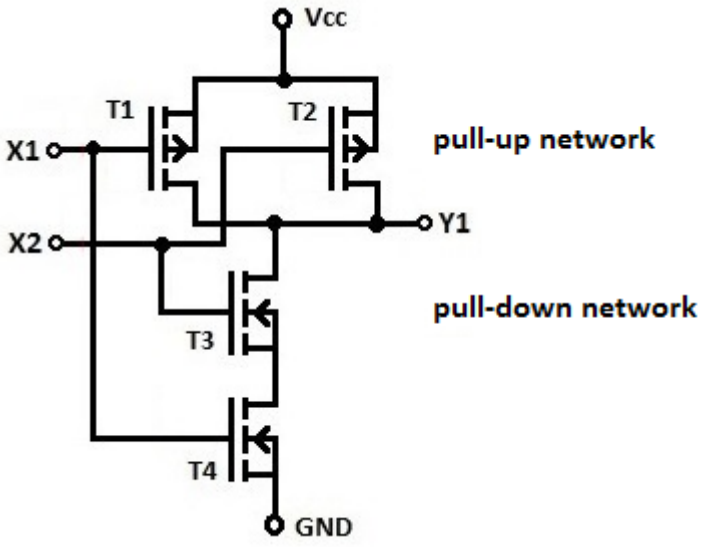


Figure 3.7: Circuit layout of a standard NAND gate with identification of pull-up and pull-down network

	Stage 1	Stage 2	Stage 3	Stage 4
pull-up network	FALSE	TRUE	FALSE	TRUE
pull-down network	TRUE	FALSE	FALSE	TRUE

Table 3.1: All four possible logic state for a NAND gate in accordance with [36]

This fine-grained evaluation of the logic functionality makes it possible to simulate individual transistor faults and their impact on the whole circuit. In Figure 3.8 some of the possible fault conditions in which an individual transistor can be functional are demonstrated. The fault conditions A, B and C of Figure 3.8 represent a disconnection of the transistor and conditions D, E and F are showing shorts between two of the transistor pins. Each transistor of the logic gate can now be put into different fault conditions and simulated with the help of the pull-up and pull-down network logic equation. By using the logic equation approach the overall impact of a single fault onto the whole logic system can be simulated and evaluated. In this thesis two types of fault simulation of individual transistors are being used for fault simulation on fine-grained-level logic gates simulation, stuck-at high (SAH) and stuck-at low (SAL) faults. This method has been chosen to study the response of a logic gate by affecting single transistors of it with faults for a set duration of time. By using this method, a logic gate output or behaviour can be put into different states than the normally permitted ones. All of the other fault injection methods described within this chapter affect the function of a logic system or specific input or output values. This can be by random distribution or at selected locations for fixed or variable time duration. The fault model used for this work is of stationary nature for a set time at selected locations in order to achieve comparable results amongst different logic systems.

The faults caused by effects happening to the intra-gate-connection, which can have the same effects as the one at the transistor level, will not be individually investigated. This is because these intra-gate-connections are dependent on the actual chip design and this is beyond the scope of this thesis. SAH represents the condition that the transistor is on all the time which will be represented in the logic equation with a high or one level. The SAL condition represents the condition that the transistor is off all the time and within the logic equation this case is simulated with a low or zero value.

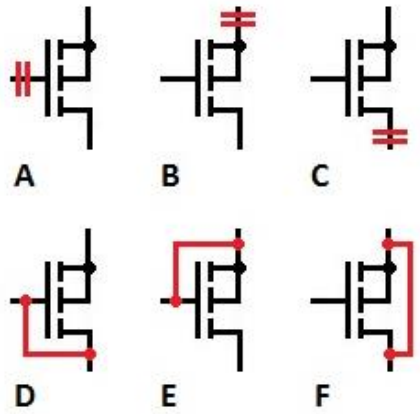


Figure 3.8: Some possible electronic faults in a transistor with regard to open connection or shorts between two pins [12]

3.6.2. Physical-based fault-injection

The physical simulation of fault-injection requires the final or a certain level of completion of the application hardware and software. It is possible that different physical fault-injection set-ups are possible for the simulation of fault-injection. One set-up can be done on the software level and the other can be done on hardware and software together. For the second set-up the final hardware and software design should be used. This is because variation of the target design can affect the behaviour response after fault-injection. The fault-injection can be done on the hardware or on the software of the second fault-injection set-up. If during the fault-injection test a problem regarding fault-masking occurs, the appropriate non fault-tolerant component of the application has to be exchanged or updated regardless of whether it is hardware or software. Reassessment of the altered target has to be performed. Different physical hardware fault-injection methods are available.

3.6.2.1. Hardware fault-injection

For hardware fault-injection the appropriate application hardware needs to be used. Two hardware test set-ups are possible, contact or contactless testing of the appropriate application hardware. The first test method of contacting the application hardware has certain limitations and advantages. The advantage of using contacting the hardware under test is that fault-injection can be repeated and altered in any possible way. The limitation of contact-based fault-injection on a given hardware lies in the access capability of the test coverage. Physical contact and fault-injection capability is limited by pins used to make interconnection possible between components. Access to internal circuit structures within a chip is not possible with this test method. The faults injectable into the circuits can be of voltage or current nature [11] or simulation of stuck-at conditions [82]. Two

contact-based fault-injection procedures on hardware level are feasible. The first method is based on using test pins to contact the suitable test pads on the PCB, which should not be done due the fact that probing on individual components, like surface mount devices (SMD), can damage the component due to the spring force of the probe. Because of this the layout of the application PCB needs to be considered with the suitable test pads in place. The second method uses a chip socket to act as an access point for injection of a fault into the circuit. Today more and more components on a PCB are of SMD nature; these components cannot be accessed through chip sockets.

Contactless testing of application hardware can be done with any type of radiation source capable of causing an upset within the hardware under test. By using radiation sources a duplication of the natural environmental, which exists in space, is performed and a contactless fault-injection into the chip design or structure can be performed. Radiation sources are producing high energy particles, which can be used for bombarding the chip structure to cause the same effects as in space but at a much higher rate. Through this a simulation of a life-time exposed to radiation particles can be performed within a short period of time. A heavy ion source used for SEU simulation is for instance Californium-252 (Cf-252) [82-84], which offers a good source of constant radiation output for bombarding the entire chip. For simulation of alpha particles Americium-241 (Am-241) [85] can be used which is widely available due to the use within smoke detectors. The Americium-241 is produced as a film and can be cut to the size required for causing SEE in parts or the whole chip. Another way of generating static proton and heavy ion radiation is with the help of cyclotron facilities producing a range of high energy particles which can be used to bombard the chip [86, 87]. With the help of the cyclotron facility it is possible to bombard only a small area of the chip if required or the entire chip. By only causing an SEE within an area of a chip, the unexposed chip area can be used for verification purposes. By utilising radiation sources, random bombardment of a chip can be performed similar to space-caused radiation effects. If every part of the entire logic structure has been exposed to high energy particles cannot be assured because of the randomness of the natural radiation distribution hitting the chip die. This method of using radiation sources is a method, which can be used on every type of silicon-based chip and it is not limited to FPGA-type chips only. Using natural radiation sources for simulation of SEU in high numbers means that the target chip has to be de-lidded otherwise the metal lid used for chip housing interferes with the radiation, which could affect the chip structure [81].

Another contactless way of injecting faults into an electronic system is by electromagnetic interference. This technique is the common disturbance in automotive vehicles, trains, airplanes or industrial plants due to high current flow in nearby electronic systems [82]. With the help of a burst generator an electromagnetic field can be generated, which then can affect the whole PCB, the whole system or only a single chip. Through this electromagnetic field, alteration within the data stored in individual transistors takes place and the fault-tolerant features of the system have to cope with them.

3.6.2.2. Software fault-injection

Software based fault-injection into the application simulation or system is a low-cost and easy-to-control method for testing the effects on a fault-tolerant system. Its approach is to change the contents of memory or register information in accordance to specific fault models. With alteration of information the emulation of hardware faults or injected software faults manifest within the software and the performance of fault-masking algorithm can be evaluated [11, 82]. Software-based fault-injection can be done at compile-time or at runtime. The method of fault-injection at compile-time introduces errors into the source code of the target programme and generates a modified application software [11]. This altered software gets downloaded into the target hardware and executed to verify its fault-tolerant capability. The fault-injection method of introducing faults at runtime works on the principle that at a trigger point alteration of memory or registering of information has taken place.

3.7 Summary of the chapter

This chapter is about the increased sensitivity against radiation effects on individual components of a given logic chip. The increased sensitivity of individual chip components in the case of individual transistors is increasing through ongoing downscaling of these components over past decades. Radiation-induced effects are impacting high-altitude applications but the smaller feature sizes of modern-day chips experience radiation upsets at terrestrial level now. In this work the focus of radiation effects was put onto the FPGA due to the selection as the best CCU platform for an electronic system. Due to the capability of freely configuring the logic structure interior of an FPGA is controlled by switches in conjunction with SRAM elements. Radiation particles bombarding a given FPGA chip die are able of altering the stored information written inside these SRAM elements. This alteration of the stored information in the SRAM-based configuration memory will, in a way, modify the intended logic design configured inside the FPGA for fulfilling the application requirements. Simulation of different fault-injection methods modelling the effects of radiation-induced faults within a given electronic systems were evaluated.

Different fault injection methods are used to simulate and evaluate the fault tolerant behaviour of a system under test. These methods needed to be evaluated, in order to find a suitable method for the work on making logic gates insusceptible against a specific type of faults. Fault injection into a system can be performed by temporal randomly or stationary applied fault types at fixed or randomly selected locations of the system. The selected method for this thesis for injection and simulating faults at each of the individual transistors of a logic gate is a stationary influence with a fixed digital level.

The simulation of breaking down logic gates into pull-up and pull-down networks including evaluation of individual transistors of this structural configuration showed that the logic gate responded within four feasible logic states. One of these logic gate states offers the condition required to be used as a uniquely identifiable signal in case of a fault presented within the logic gate. Detection and triggering on this signal could be used for the purpose of initiation of self-healing features, which is going to be investigated in chapter 7.

Chapter 4: Review of type of faults and their behaviour on an electronic system

4.1. Introduction

Every man-made electronic system can suffer from an electronic fault at any time during operation. Faults can be through undetected manufacturing hardware defects, which become faults or show wear-out characteristics of individual components within the integrated circuit. Both these types of permanent hardware faults are going to increase in future electronic systems, which are based on integrated circuits. This is because feature sizes of these components are being scaled down, due to process development of chip manufacturing moving down into the regions of nano-structures of individual components. The requirement of producing fault-free chips in the future will only be possible through capital investments of the chip manufacturers. Faults within a system with this feature size of individual components can also be possible because of radiation-induced effects. Radiation-induced effects which alter data within memory in most cases on a temporary basis within a given chip will only increase in the future. Both types of faults, permanent and temporary, can have different effects on the behaviour of the electronic system. Some take effect right away and alter the electronic system behaviour in a way that is noticeable to the user or the outside world. If the fault is able to propagate through the system passing to every functional boundary this fault become an error of the system. The opposite is that faults can also be masked within the system before they can effect or alter the required system responses. A system, which is capable of masking fault autonomies, has a system structure that is designed in a way to handle faults by masking them. The electronic system can also be equipped with self-healing circuit features, which can handle and correct faults within the circuit structure before it passes a system boundary. Certain techniques are proven concepts for fault-masking and they will be identified, described and analysed within this chapter.

4.2. Impact of chip feature-scaling development on fault-behaviour

Until very recently the driving factors for the microcontroller industry have been cost, performance and reduction of chip die size. Reducing the chip die size is the key figure for the overall chip price. The less silicon is required for a given chip the less is the price of the given chip. The reduction of transistor size was predicted by Moore in 1965 and he forecast that every 18 months the transistor count on a fixed silicon die area doubles. Even today this law still remains valid but by moving into the region of nano-structures the law will possibly no longer be valid in future. By increasing the amount of transistors produced on a given die area, this trend will go hand in hand

with new challenges for the chip industry. These challenges are going to change their objective. The reliability and yield of their product will see the biggest impact. In real terms a 5% loss of a typical 90-nm chip fab would be around $100 million per year due to permanent faults [88]. An overview of possible failure mechanisms of semiconductor devices is illustrated in Figure 4.1 [13-15]. The chip industry is working on counteraction like adding spare circuity on a chip, which can be patched in for a faulty chip component. For doing this the entire chip has to be tested thoroughly to detect any possible fault and trying to fix the fault with patching in spare components for keeping their yield numbers up. This process is a time consuming task and requires sophisticated test equipment. Another possible approach could be to equip chips with the capability to fix themselves autonomously in the event of a permanent fault presence within the logic structure of the chips. With this approach the reliability of chips will increase over their life-time due to counteraction taken by the chip itself against wear-out effects.

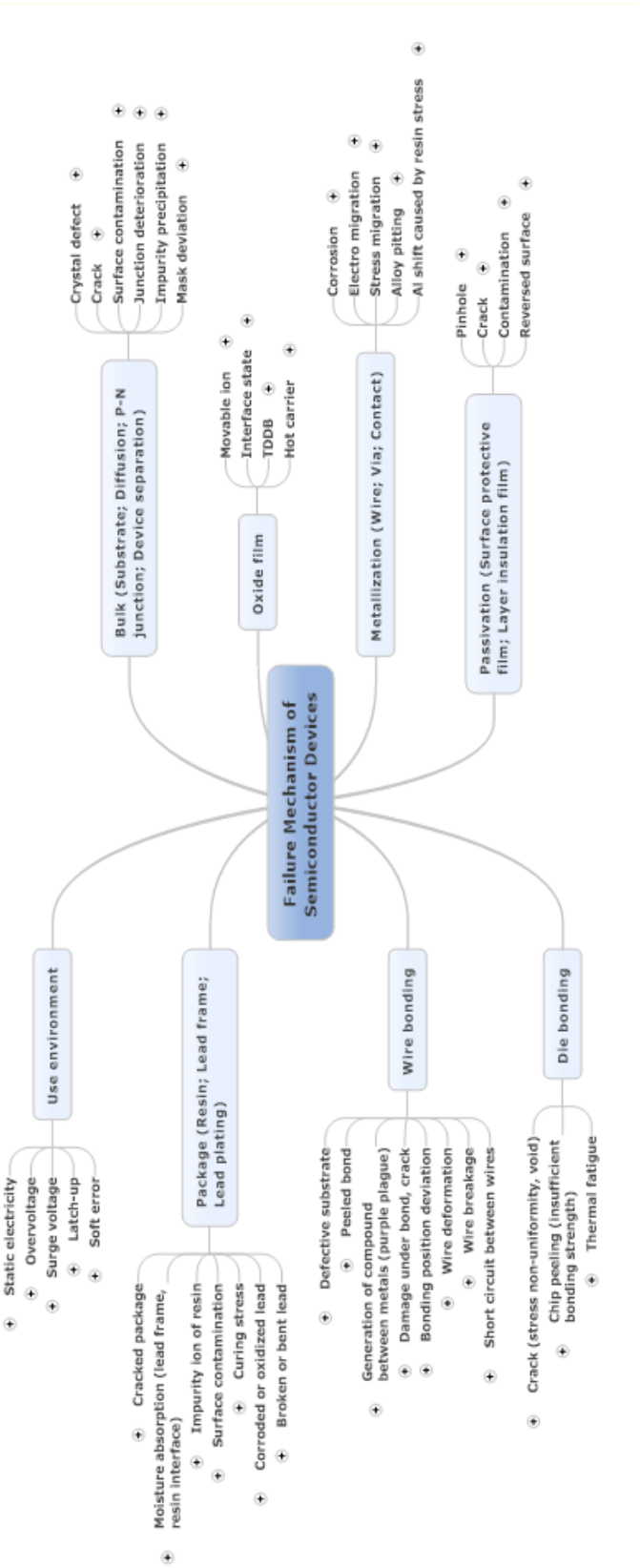


Figure 4.1: Overview of possible failure mechanisms of semiconductor devices [13-15]

Parameter degradation of single transistors due to the reduced active silicon feature structure can lead to permanent faults and will affect the system during its operation time. For these types of faults a chip cannot be tested during its manufacturing process. It can only be modelled to calculate the time frame in which the chip will work fault-free before wear-out effects take place. The dimensions of one transistor will gear towards single digit atom count used for their feature size. This will reduce the amount of dopant atoms present within its structure. By reducing the transistor size, with every generation by two, the dopant atoms decrease accordingly and the predicted trend is demonstrated in Figure 4.2 for the random dopant fluctuation.

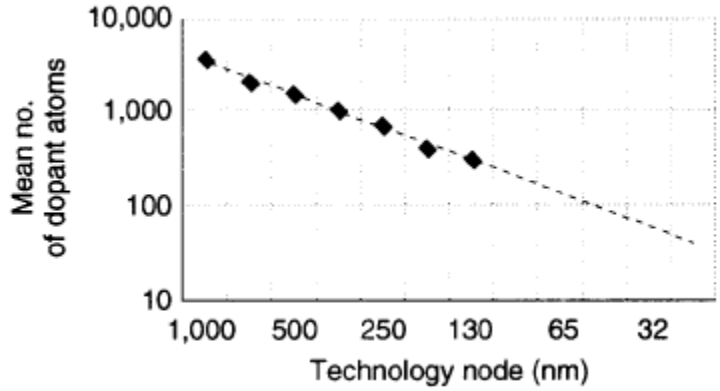


Figure 4.2: Graph of the random dopant fluctuation due to feature size reduction [16]

As demonstrated in Figure 4.2 the predictability of the reliability of the doping with a single transistor will be unpredictable. This indicates that two transistors produced side by side on the same die will have different electronic parameters. The performance of these two transistors with regard to operational behaviour will be different. The oxide thickness of each individual transistor will have a high level of impact on the overall performance of the entire chip. Due to the overall reduced transistor size the oxide thickness gets thinner and this increases the leakage current of this particular chip. In order to overcome this problem one possibility could be that the oxide thickness can be increased to counteract the leakage current. By increasing the oxide thickness the switching speed of the individual transistor is going to be reduced and in this way the logic performance of the chip [16, 41]. Finding the right balance of all the different parameters of producing a chip is the challenge for the chip industry. But a reduction of the oxide thickness increases the wear-out behaviour of each transistor differently during life-time use of this chip [89]. This increase in wear-out can lead to permanent faults, for instance that a transistor stays active at all times. That type of fault would represent an SAH fault of this particular transistor. Right after the production of the new chip it has a random number of imperfections within the oxide layer within different transistors and this is distributed across the chip die. At first the wear-out of the oxide will alter the

timing behaviour of the affected transistor and in this way the response time of this logic gate gets slower. In this case these types of fault are identified as soft breakdowns and this will lead to a permanent fault of the particular transistor. The definition of this type of fault condition is identified as stuck-at transistor faults behaviour [90]. A different failure can happen with reduced dimension size of the components of the chip die, this is the electro migration. Electro migration happens due to the reduced isolation gap between tracks. A smaller gap between two tracks will increase the electric field between them and this field can lead to electro migrational growth. Electro migration happens due to metal ions migrating due to an electronic field over time and could cause faults like short or open circuits [91]. These types of faults are considered as permanent hardware faults. Electronic shorts can be against other signals, V_{cc} or GND. Electro migration is a typical fault within the application system during life-time use and not during the production of the electronic system. This makes this type of fault within a given chip a concern for the manufacture of the electronic system and its end user.

As happens right now some IC manufacturers scrap products if they have a single fault or did not pass their manufacturing test and/or cannot be fixed. That is because the current manufacturing failure rate for producing conventional complementary metal oxide semiconductor (CMOS) devices is roughly $10^{-7} - 10^{-6}$ faults [92]. This failure rate will change in the future and chip manufacturers need to deal with these defective parts within their production. For the user of these chips another key figure is the failure-in-time (FIT) rate, measured in one failure in 10^9 device (chip) hours uptime. Applying a given FIT rate of 10 on a given number of one million components (e.g. transistors) operating for one thousand hours would mean to expect 10 components having failures. The chip user acceptance FIT rate for electronic devices in the year 2000 was 10 FITs for a certain chip type. But for the future the users are expecting a smaller FIT rate for a given chip family [89]. This expectation does not coincide with the demand for more functionality and speed out of a given chip area. These chip customer demands can only be achieved by increasing the stress on the chip because of higher current densities and higher electronic fields within smaller geometric transistor dimensions. Another industry-used failure definition for a single electronic component is the value of the mean time to failure (MTTF). Each single component (e.g. transistors) could have a MTTF of a billion years. But due to the fact that a single microcontroller has hundreds of millions of individual components with individual MTTF, the overall MTTF of the microcontroller could be just a few months. Today's electronic systems contain a number of chips and so the overall MTTF of this particular electronic system could be possibly months or weeks or days [93]. Figure 4.3 is showing the definition of MTTF within a system.

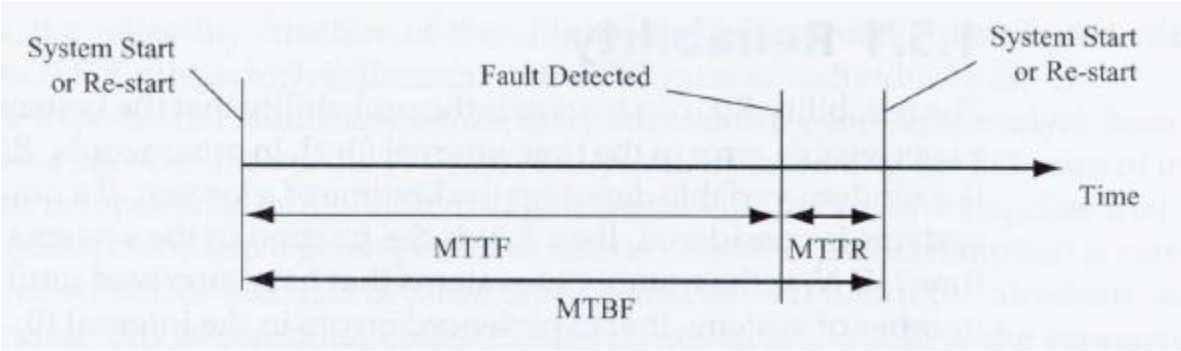


Figure 4.3: Mean time of failure-type definition within a system [17]

In Figure 4.3 two more industry definition for mean time of failure are demonstrated. The first is mean time between failure (MTBF) and the second one is mean time to repair (MTTR). MTTR defines the time between the detection of the fault and the time it takes to repair the fault. MTBF is the added time of MTTF and MTTR together and describes the time frame for how long it takes before the system works fault-free again [93]. Two more definitions are being used for the description of the ability of a system's fault-handling capability, which are mean time to manifest (MTTM) an error and mean time to detect (MTTD) a fault. These two types of fault definitions are linked to configuration bits used to define functionality within a flexible programmable logic system like an FPGA. MTTM defines the time an error is dormant within a system. A dormant fault is a fault which is not active at the time due to fact that it could be possible that the fault is present within a occasionally used or a spare part of the chip/circuit [94]. This time can vary depending on the functionality assigned to this faulty configuration bit. Only in the case of actually using this functional part of the chip or logic structure of the application where the fault is located, will the effects of the fault show up within the system. The MTTD defines the elapsed time between the corruption of the configuration bit and the detection of the faulty bit within the configuration [95].

4.3. Definition of fault and error in an electronic system

The definition of a fault being active within an electronic system is when the circuit or logic system produces an error, which represents a result or action that deviates from the correct service state of the equipment [95]. Because of this link between the different phases it takes for the manifestation a fault within an electronic system the phases have to be described in more detail. First, a fault occurs within a sub function of an electronic system, which triggers an error. The error sets off a failure within the subsystem or sub-function. A failure is the end product of a system-level or functional hardware block fault. This is only the case if the failure shows up on the boundary of the system otherwise the fault is dormant. Any type of dormant fault could be present within systems

parts, which are not in an active or used part within the system on a frequent count. Even they can be present in spare part, which are only be used in cases of reconfiguration and in this case are not sufficient for fixing a faulty system [94]. For the stability of any electronic system the propagation of any fault through the system has to be prevented and this is the area of fault-tolerant systems. If an electronic system shows an error or delivers incorrect results this happens because of a fault or failure within this system. In this way an error is more or less the manifestation of a fault for the user of this electronic system. Errors within an electronic system could manifest themselves during operation time, e.g. a changed memory bit within a memory cell. Only in the case of activation of these faulty logic parts of the chip will the fault show up and possibly traverse through an electronic system. The definition of a fault-tolerant system is that it is designed to deal with faults within given design limitations for maintaining the required system functionality. A fault within an electronic system could be caused by a hardware or software fault. The research work done for this thesis is focused only on the electronic hardware faults.

4.4. Faults and errors in an electronic system

With Figure 4.4 the possible fault propagation within an electronic system is demonstrated. In the case A of Figure 4.4 the fault is being masked within the inner scope of the electronic system with the help of fault-tolerant logic. The electronic system will not show an error at the outer system scope. So the user of this system will not know that a fault happens within his electronic system. In the case of a masked fault which happened at the inner scope of the system the system designer has designed an indication for this purpose and the user will be informed about it. In case B of Figure 4.4 the fault shows up in the outer scope visible to the user. This can be through an alteration of the required system response.

One approach to deal with faults in a system is to mask the fault within the inner scope of the system. In this case the outer system scope will not see the fault and will generate no error or system misbehaviour. The technique of fault-masking relies on the capability of detecting a fault which exists within the electronic system and is adapting the logic design of the system to cope with this type of fault. In some cases different masking capabilities have to be used. Miscellaneous masking schemes are available for electronic logic systems.

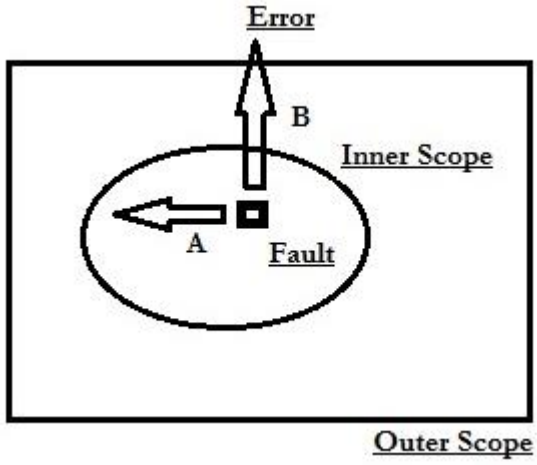


Figure 4.4: Fault propagation within system [17]

4.5. Types of faults in an electronic system

All these different types of electronic hardware faults within an electronic system can be classified into three categories: permanent, intermittent and transient faults. Besides these three main fault categories two more fault-types have to be mentioned which are benign and malicious type faults.
A benign fault of an electronic system is a fault condition when the system just goes dead during normal operation without any prior indication. This kind of fault-type would be easily detectable and repairable, but the impact of this fault happening during operation could be a real misfortune for the user of this system. If this kind of possible fault happens during a space flight of a satellite the whole project would be lost without the possibility of repairing the system. Malicious faults, which are also called Byzantine faults, are such as when a system will deliver reasonable looking results on request but these results are incorrect. For example an altitude sensor of an airplane reports 1000-feet altitude instead of the correct 8000-feet altitude [96]. These two types of faults are falling into the category of logic function-related faults or even design related. Because of the way benign faults happen within a system they cannot be resolved within this block or system. Due to that impact and the way these faults react they will not be part of this research work.

4.5.1. Transient faults in an electronic system

Transient faults occur and vanish within a system and manifest themselves in most cases in the nature of bit flips, which got stored or logic gate malfunction. The root cause of a transient fault is due to a high energy particle like a neutron or alpha particle hitting the silicon structure of the chip. This impact of this particle has to be near a transistor or capacitor of a static random access memory cell logic to cause a bit flip. This is due to the energy induced at this point of the chip

where the particle struck. The current ongoing dimension reduction of all components on a chip reduces the amount of charge stored within the capacitor of a static random access memory (SRAM) cell. Due to this reduced charge stored within the capacitor it makes it more susceptible to gamma particle radiation [25].

By hitting the silicon chip structure the high energy particle creates a charge that alters the voltage levels in this area and can flip a bit in a memory cell or a logic latch. Within the memory chip the effect of the flipped bit can be detected and corrected with the help of parity bits and ECC.

Researchers expect that per new generation of chip technology the soft-error rate per logic state will increase by 8%. In Figure 4.5 the soft-error rate in relation to the technology generation is demonstrated and is showing that the soft-error rate for future chip structures will increase. This is happening because of the reduced component size of a given chip into the nano regions and this will cause the likelihood of an increase of soft-errors at sea-level increases. These soft-error effects in a given next-generation chip will no longer only be a problem to high altitude applications and because of this it will require the same fault-tolerant approaches to be implemented for low altitude applications.

The detection and correction of flip bits caused by energy particles within a flip-flop of a chip is a much harder problem [16]. New chips are equipped with more functionality built-in and because of the reduced size the number of components within a given chip area is significantly increased. Both points lead to bigger chip sizes within the package and this means an increase of the target area for energy particles [25].

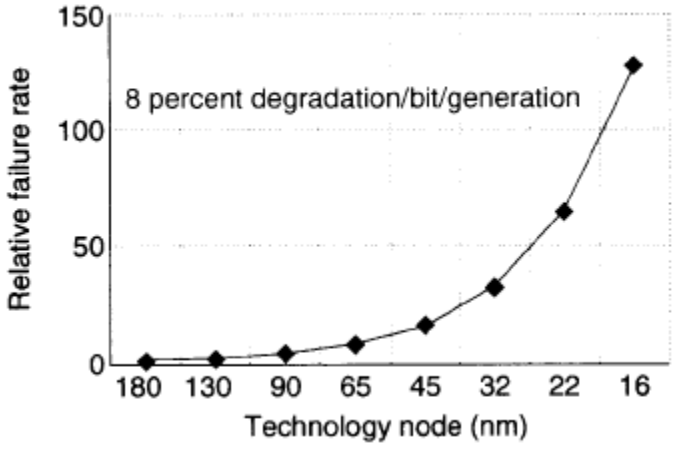


Figure 4.5: Soft-error failure-in-time of a chip (logic and memory) [10]

The second cause of this type of fault can be electromagnetic interferences. Transient faults cannot be fixed with the exchange of the hardware [97]. This type of fault gets described as single event upset (SEU). In the case that this upset happens in the same component with a certain frequency,

this is getting defined as a single event rate (SER) for this specific component. Because of the transistor structure dimensions being close to nano-style feature size on a chip it is possible to generate MBUs by a single radiation hit. It is much easier to generate MBUs within a memory chip due to the density of memory cells within a certain area. The SER gets commonly measured in FIT. Modern chips with their reduced structure dimensions of a single transistor will have soft-error rates, which are producing a failure rate that is higher than all the permanent hardware failures combined. Today's electronic devices have a typical failure rate of a gate oxide breakdown, metal electro migration for example of 1-50 FIT for a single device type. The overall FIT rate of a chip will be due to the critical reliability mechanisms of any chip, which are going to be more in the range of 50-200 FIT. By comparing the chip FIT rate against an easily exceeding SER driven FIT rate of possible 50000 FIT/chip the scope of the relevance of the FIT driving effect changes [16]. These numbers show that in the future the soft-error-induced FIT rate is going to be the dominant FIT rate within an electronic system of a given application in the future. According to [98] the FIT rate at sea level for latches and SRAM cells varies between 0.001 – 0.01 FIT/bit which increases with altitude. The combined FIT rate of a whole chip is the sum of all raw FIT rates multiplied by the soft error susceptibility factor of this individual component [98].

4.5.2. Permanent faults in an electronic system

A permanent fault can be described in this way; as that part of an electronic system that produces a fixed result permanently. This result in any digital system can be either correct or incorrect. For example if the permanent created result due to the fault is a constant digital high level and in the case the system is requiring a high level result, which means that the evaluation of the result will be seen as correct even if the result has not been generated. But in the case of a required zero level it is incorrect. Judging the correctness of the circuit only on the comparison against similar circuit output will not always reveal a faulty system. It could be possible that more output results for evaluation are required or another type of indication in the case of a fault is necessary.

A permanent fault reflects irreversible physical changes within a chip logic circuit of the system [97]. In this way a permanent fault will remain for an indefinite period within the electronic system until this device or component gets replaced. A permanent fault can be best described with the example of a defective light bulb. In the case of a fault the light bulb will not generate light. The fault will only be fixed in the event of replacing the light bulb. Within an electronic system this could mean for example that at a given chip an input or output of a logic gate is stuck-at high or zero permanently. This type of fault could be due to wear-out, migration, manufacturing issues or using the device out of specification. Latch-up effects within the chip can also act as a permanent fault. The difference between hardware related issues and latch-ups means that the latch-ups can be resolved with power cycling. But in some cases a burn-out of the particular logic circuit, which is

having a latch-up, can become a permanent hardware fault. In this case the hardware needs to be replaced with new equipment or the chip by itself is capable of altering the application-specific logic structure to be reconfigurable for avoiding this part of the chip.

4.5.3. Intermittent faults in an electronic system

Intermittent faults are faults, which could appear and disappear over time during the operational period of the electronic system. As the name indicates this kind of fault is not of a permanent nature, it will happen from time to time. Sometimes errors, which are intermittently affecting an electronic system, tend to occur within this system in bursts if the transient fault happens at the same location and activation [97]. Intermittent faults can be seen as an early device indication for permanent faults, which could manifest within this device as a certain individual component. An example of an intermittent fault could be a partial oxide wear-out of a single transistor of a chip. A study, which has been done, was based on fault data collected from a number of data servers for identifying intermittent faults and their effect on the operation of these data servers. This data represented the fault data of these data servers over 310 operational years. The data showed that the systems experienced 6% intermittent single-bit errors (SBE) within their memory during the time of observation. All these faults were corrected with the memory error correcting code (ECC) and therefore no service interruption happened. Failure analysis carried out when possible indicated that manufacturing residues on the contacts of the memory cards caused an intermittent contact problem [97]. This was seen as the root cause of the intermittent bit faults within these data servers.

4.6. Detection of fault or error occurrence in an electronic system

As Figure 4.4 demonstrated, the definition of a fault is that the fault stays within the limits of the functional block of the total electronic system. The fault gets identified at the boundary of the functional block and masked. Faults which are masked stay within the functional block unnoticeable to the outside world. Errors are manifestations of faults occurring within any system noticeable to the outside world. This indicates that the fault had passed through every boundary of each functional block. For masking a fault at the functional block boundary different approaches such as majority-voting or comparing can be applied within a logic based system.

4.6.1. Majority voter at the boundary of a functional block

The function of a majority voter can be seen as majority-voting the overall output result out of a set of individually created or a stored number of data bits. In this way the majority voter is working on the concept of data redundancy, which should all represent the same value. The functional block diagram of majority voter in general is demonstrated in Figure 4.6. According to von Neumann [99] at least more than $^{N}/_{2}$ of the inputs (Y_x) supplied from the identical circuits (M) have to carry the correct result for a majority-voted result. The identical circuits (M) are forming the main functional block of an electronic system and the number M represents the number of hardware overheads compared to a single structure. In principle the voter demonstrated in Figure 4.6 can only work on single digital results or data structure due to the direct comparability of the system results. The majority-voted result of single digital results can be done within one clock cycle. For a majority-voting based on data structures the voting has to majority vote on each individual data structure bit and this must accord to the number of data bits the structure contains. For creating a majority-voted output result of the data, each individual bit of the data word has to be majority-voted and has to be in accords to the requirement of more than $^{N}/_{2}$ bits have to match of the same data structures. It is also possible to create a majority voter, which will take the whole data structure and create a majority-voted output result by means of doing the comparison of the whole data within one clock by parallel majority-voting. This concept would require the number of data bits majority voter working in parallel.

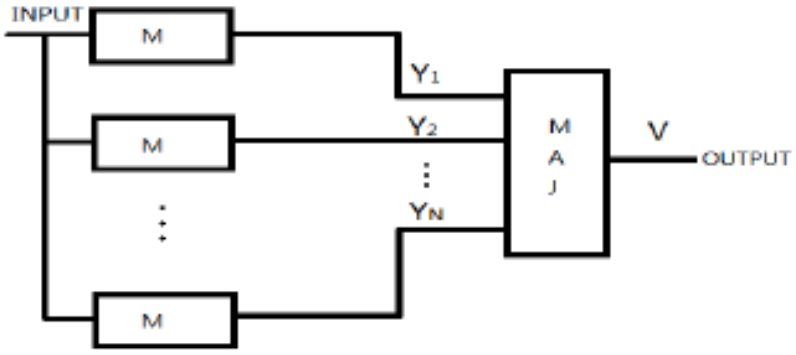


Figure 4.6: Majority voter block diagram for an NMR system [18]

The majority voter demonstrated in Figure 4.7 shows the logic circuit of a conventional triple module redundant (TMR) majority voter. The Boolean equation of this TMR voter is:

$$Y1 = (X1 \wedge X2) \vee (X1 \wedge X3) \vee (X2 \wedge X3) \qquad \text{(Equation 4.1)}$$

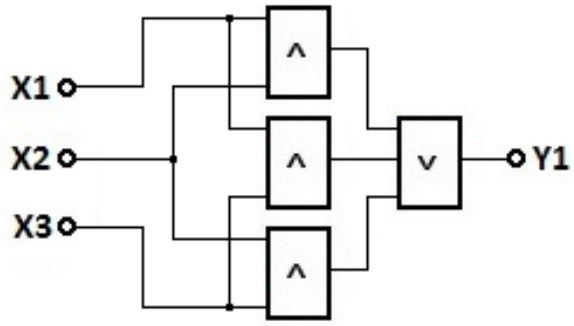


Figure 4.7: Conventional triple module redundant (TMR) majority voter logic circuit created out of single logic gates

For this example of a majority voter a TMR system gets chosen because it is the minimal redundant system, which is fulfilling the von Neumann rule. The resulting truth table of the majority voter for a TMR system is shown in Table 4.1. The principle of majority-voting can be observed within the data shown in Table 4.1 and the rule that $^{N}/_{2}$ of the inputs are required to have the same result for the majority vote to generate a valid output result. The results of the table reflect this rule for all the different input sequences. The data represented in Table 4.1 shows the output results of a fault-free majority voter for a TMR system. But how is the output result of a TMR majority voter affected by fault-injection of SAH or SAL faults at different injection points in accordance with Figure 4.8? The majority voter is the functional block within a fault-tolerant system based on fault-masking and the fault-behaviour affects the fault performance. The reliability calculation for a TMR-based fault-tolerant system demonstrates the impact on the overall reliability of a TMR system with majority voter. The general reliability equation for a TMR system with majority voter is for two out of three subsystems of a TMR system to be correct is:

$$R_{tmr\ with\ voter}(t) = R_{voter}(t) \sum_{i=0}^{1} \binom{3}{i} (1 - R(t))^{i} R(t)^{3-i} \qquad \text{(Equation 4.2)}$$

General TMR reliability equation with majority voter

$$R_{tmr\ with\ voter}(t) = R_{voter}(t) \sum_{i=2}^{3} \binom{3}{i} R(t)^{i} (1 - R(t))^{3-i} \qquad \text{(Equation 4.3)}$$

TMR reliability equation with majority voter where two out of three subsystems are correct

$$R_{tmr\ with\ voter}(t) = R_{voter}(t)(3R^{2}(t) - 2R^{3}(t)) \qquad \text{(Equation 4.4)}$$

The reliability equation of a TMR system with majority voter where two out of three subsystems are correct

As equation 4.4 shows the overall reliability of a TMR system with majority voter is determined by the reliability of the majority voter. This is due to the multiplication of the reliability of the majority voter with the overall reliability of the whole TMR system. In this regard the majority voter can be seen as the single point of failure within this fault-tolerant structure. Any deviation of a 100% reliable majority voter cannot be tolerated for the performance of the reliability of a fault-tolerant and reliable TMR system with majority voter. In this regard the question of the fault-tolerance of a majority voter will show how the fault-tolerance of a TMR system with majority voter affects the trustworthiness of this system.

X1	X2	X3	Y1
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	1
1	1	1	1

Table 4.1: Truth table of the TMR majority voter
demonstrated in Figure 4.7

For analysing the fault-behaviour of the conventional TMR majority voter regarding SAH or SAL faults injected into the circuit at the appropriate points in accordance with Figure 4.8, a fault rate analysis will reveal the fault-behaviour effects. This simulation was performed to reveal the fault rate (FR) of the circuit and the effect on the majority-voted output with regard to trustworthiness. For indicating a fault the comparison between the output result of the fault-free against a fault-injected one for a given input stimuli has been used. In this case for a TMR majority voter any deviation of the output value of the fault-free results (shown in Table 4.1) can be seen as an untrustworthy output result. Out of this type of results it can be seen as a system error generated and passing through a functional boundary caused through a fault within the majority voter. This example illustrates the impact the fault within the majority voter has on the fault-behaviour of the entire system and proves the point that the majority voter is the single point of failure. The method of FR was chosen because it offers the best comparability of fault-behaviour between different system structures. The calculation of the FR of a given circuit structure can be done by the following equation that is usable for different circuit structure set-ups:

$$FR = \frac{N_{Faults}}{N_{Input\_Stimuli}} \cdot 100\% \qquad \text{(Equation 4.5)}$$

The FR for the fault simulation of the TMR majority voter is shown in Table 4.2 with a total FR of 22.6% for injected stuck-at fault simulations at appropriate stimulus points (see Figure 4.8). Each of these results of the TMR majority voter in response of a stuck-at fault-injection is a deviation and resulting in an error (see Figure 4.4). This error revealed by means of stuck-at fault-injection shows that the trustworthiness for the TMR majority voter is given by an FR of 22.6%. With this FR the TMR majority voter will generate 77.4% correct results under the influence of a stuck-at fault and cannot be identified as a fault-tolerant or a 100% correctly working system under the influence of a stuck-at fault. Out of this fault-behaviour and the importance of the majority voter on the overall system behaviour the majority voter will require further analysis work for increasing the competence of this vital functional block of a fault-tolerant system. This further analysis and circuit alteration is carried out within Chapter 7.

The majority voter in general is placed at a boundary of a subsystem and supplies a result into another subsystem or to the outside of the system. Masking of a faulty generated output signal in this circuit set-up is shown in Figure 4.7 and is not part of the logic structure. So in the case of a fault within the TMR majority voter the faulty output signal will propagate through the system and will pass functional system boundaries. Inherent or designed into the circuit structure capabilities of fault-indication is not possible with the circuit demonstrated in Figure 4.7.

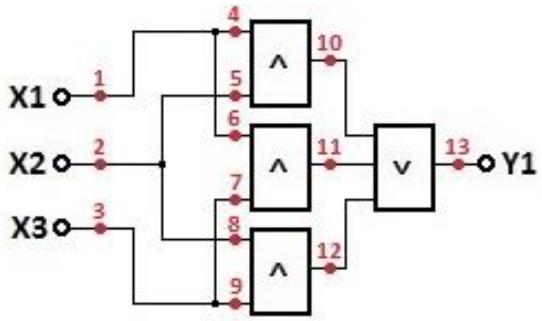
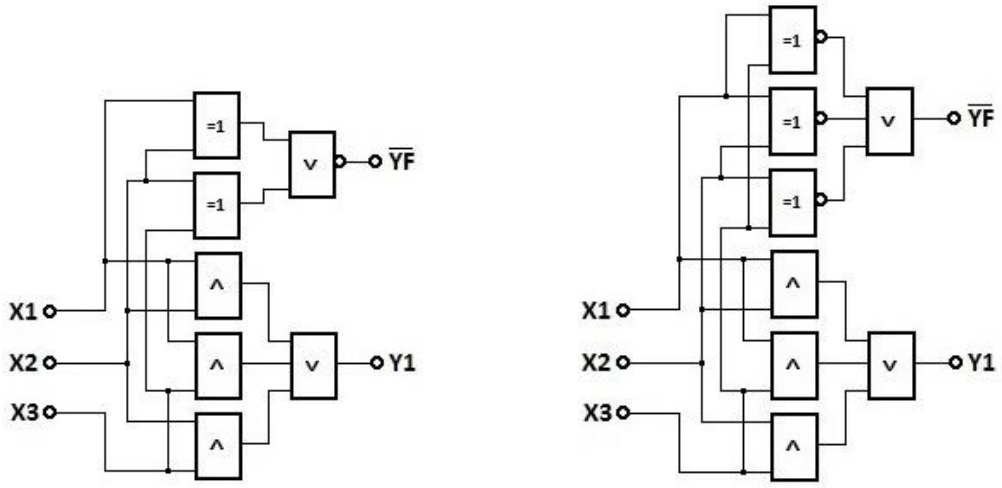


Figure 4.8: Conventional TMR majority voter logic circuit with stuck-at simulation points (1 to 13)

	1	2	3	4	5	6	7	8	9	10	11	12	13
SAH	2	2	2	1	1	1	1	1	1	4	4	4	4
SAL	2	2	2	1	1	1	1	1	1	1	1	1	4
# of faults	4	4	4	2	2	2	2	2	2	5	5	5	8
Fault rate	25	25	25	13	13	13	13	13	13	31	31	31	50

Table 4.2: Fault rate data of the stuck-at simulation at specified injection points indicated in Figure 4.8

If a fault-tolerant TMR majority voter (see Figure 4.7) is required the circuit needs an additional circuit part that indicates through a signal if not all input signals (Y_x) of the TMR majority voter do not have the same digital value. For this example the number of inputs for any type of logic gate used within a circuit is limited to two inputs. This has been chosen for evaluation purposes. The fault-indication circuit is demonstrated in Figure 4.9 where $\overline{YF}$ indicates the situation that not all input signals feeding into the majority voter are of the same logic level value.



(a) (b)

Figure 4.9: TMR majority voter with fault indicator circuit for the case that inputs are homogenous. (a) for homogenous of all inputs, (b) for homogenous of two out of three

The fault-indication solution demonstrated in Figure 4.9(a) is for the case that all inputs are homogenous and the logic equation is therefore:

$$\overline{YF} = \overline{(X1 \oplus X2) \vee (X1 \oplus X3)} \qquad \text{(Equation 4.6)}$$

The equation demonstrated in Equation 4.6 and the corresponding Figure 4.9(a) represent a circuit for indication of consistent input signals feeding into the TMR majority voter. This circuit solution is not in accordance with the von Neumann rule that $^{N}/_{2}$ inputs have to be the same and a matching circuit in accordance with this rule is shown in Figure 4.9(b). For this circuit the logic equation is:

$$\overline{YF} = \overline{(X1 \oplus X3)} \vee \overline{(X1 \oplus X2)} \vee \overline{(X2 \oplus X3)} \qquad \text{(Equation 4.7)}$$

With this circuit structure (shown in Figure 4.9(b)) this altered TMR majority voter is now able to indicate that a deviation of one input signal has occurred. For these different faults indication

signals of these altered TMR majority voters will indicate a fault condition to the higher controlling circuit. With these fault-indication signals the identification of the specific fault within a path is possible and with the help of another circuit structure the identification of the faulty input path is possible. The identification of the faulty input path is possible through an evaluation of the majority-voted result signal against the individual voter signals. This approach also discloses the cured influence of a fault affecting a single track of a TMR system to a system controller of this TMR system. A simple comparison circuit is delineated in Figure 4.10 and the associated logic equations of this circuit are the following ones:

$$YFX1 = X1 \oplus Y1 \qquad \text{(Equation 4.8)}$$

$$YFX2 = X2 \oplus Y1 \qquad \text{(Equation 4.9)}$$

$$YFX3 = X3 \oplus Y1 \qquad \text{(Equation 4.10)}$$

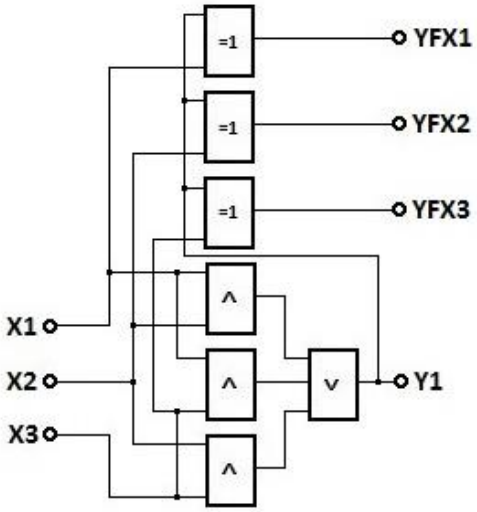


Figure 4.10: TMR majority voter with output fed-back comparator against inputs for identifying faulty input path

For a fault-tolerant TMR majority voter the total circuit would be a combination out of the following circuits illustrated in Figure 4.9(b) and Figure 4.10, which then would work side by side. This combination would indicate the presence of a fault and the fault creating input path of the TMR system. Comparing the overhead based on transistors the following logic gate transistor counts has been used:

AND = 6 Transistors, OR = 6 transistors, NOR = 4 transistors,
XOR = 12 transistors, XNOR = 14 transistors.

Total transistor count in accordance with the figures and comparison against the transistor count of the TMR majority voter (see Figure 4.7) is represented in Table 4.3. The overhead for the different circuit configuration is significant. By using two standard input logic gates for making the TMR majority-voter fault-tolerance the overall problem of faults is still maintained due to the fact that in the case of a faulty component within the circuit no indication of this is built-in.

	Transistor	Overhead
Fig. 4.6	24	0%
Fig. 4.8 (a)	52	117%
Fig. 4.8 (b)	72	200%
Fig. 4.9	60	150%
Fig 4.8 (b) + Fig. 4.9	108	250%

Table 4.3: Transistor count comparison against TMR voter
(see Figure 4.7) as overhead

4.6.2. Comparator at the boundary of a functional block

Instead of the majority-voting a comparison of the results produced from individual blocks can also be used for avoiding fault propagation through the system. The resulting circuit uses less individual components than the TMR majority voter, which in this case should result in a reduced FR. This comparator approach is mostly used for dual redundancy electronic systems displayed in Figure 4.11 in which a single AND gate is being used as the comparator.

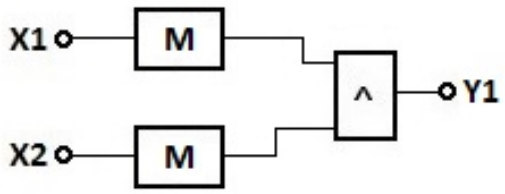


Figure 4.11: Dual redundancy electronic system with AND-gate
as a comparator at the output

This simple comparator solution done with the AND gate has a significant impact on the FR of this set-up. In the case of a mismatch between both output results an overall result of zero is produced and this correlates to a 50% FR. The circuit shown in Figure 4.11 needs a means of output mismatch indication to a hierarchical higher control system. A simple solution is described in Figure 4.12 where this comparison of the two outputs has been done with a single XOR gate [100].

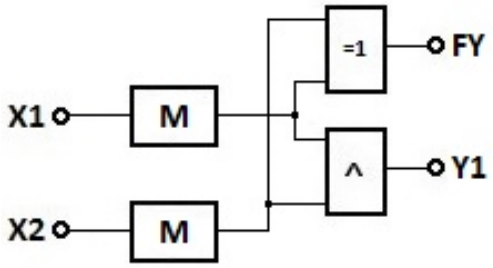


Figure 4.12: Dual redundancy electronic system with AND gate comparator and XOR gate as fault indicator

4.7. Summary of the chapter

Emphasis in this chapter was placed on temporary and permanent faults within a given logic structure and their effect on the whole electronic system. In general any electronic system is divided into different functional logic blocks and each functional logic block can be defined as surrounded by a boundary. A fault which passes through this boundary unnoticed and without masking/fixing is defined as an error. This behaviour puts the trustworthiness of the whole system into question. Measuring at the boundary of the functional logic subsystem is required to perform fault-tolerance and masking. The most commonly used logic structure to perform the task at the boundary is the TMR majority voter. TMR majority voter requires a triplication of the functional logic circuit for generating three independent output results. This by itself generates 200% hardware overhead. The trustworthiness of the functionality of the functional block depends on the fault-behaviour of the TMR majority voter. A fault-injection simulation performed on the input and output structure of the discrete voter structure reveals its fault response to stuck-at high/low faults. This simulation revealed that the FR of a TMR majority voter is 22.6%. Fault identification with regard to identification of outputs feeding into the voter requires additional logic circuit facility. The FR of the voter indicates that an altered logic structure or fault-tolerant logic gates are required for designing a fault-tolerant TMR majority voter. Within a fault-tolerant TMR system the majority voter can be seen as the single point of failure for the system. Because of the impact the majority voter has on the fault-tolerance research work with regard to increasing the fault-tolerance of the majority voter will be one focus of this thesis.

Chapter 5: Concepts for increasing dependability of logic systems

5.1. Introduction

Fault-tolerant systems are capable of preventing the propagation of a fault through their logic structure, which subsequently could manifest as a noticeable error to the world outside of this system. The fault-masking capabilities of these types of systems rely on certain logic structure designs created to equip logic-based systems with fault-tolerance. By applying these techniques onto a given logic circuit, the reliability of the resulting circuit regarding fault-tolerance will be increased. This is the direct benefit but in most cases it comes with a price to pay, which is logic overhead. This logic overhead affects all parameters of the logic circuit.

5.2. Fault-tolerant per system design

The concept of fault-tolerance per system design can be achieved with two different approaches. The first one is fault-masking and the second one is fault correction. The fault-masking principle works on the concept of using redundancy within the output result creating functional blocks so that through means of comparing or majority-voting a final output result can be generated. Reliable systems working with the approach of redundancy usually exploit one of the three possible redundancy forms: temporal (time), spatial (hardware) or pertaining to information [88, 101]. The information redundancy can also be defined as data redundancy where a set of the same value is generated independently or stored within different memory locations. A generalised block diagram of the majority-voting principle structure for an N-type design is shown in Figure 4.6, which is used for the spatial redundancy principle for generating a single output data out of redundant data. The original design principle was described in 1956 by von Neumann [99] for logic designs with a high number of redundant copies of the same logic structure. The most commonly used redundancy structure is the three-parallel electronic system or TMR working side-by-side, which also fulfils the requirement that more than half of the redundant systems produce the same output result. Dual redundancy systems cannot fulfil this requirement and an output result comparison can be utilised, in which case both output results have to be identical. Comparison cannot offer fault-masking due to the fact that there is no comparison in the case of a non-matching situation. The overall reference is missing by only having two output results. Another type of spatial redundancy can be applied onto the individual transistors creating the logic gate function, which has been proposed by El-Maleh et al in [22]. This paper proposed to replace every transistor of a logic gate with an N^2-transistor structure. Through this approach the altered logic gate is fault-tolerant against stuck-at faults. The principle of fault correction within a logic circuit defined as quadded logic structure was introduced in 1960 by Tryon for a certain set of logic gates and in 1963 Jenson expanded it by

another gate type [102]. The principle of fault correction can be done with replacing each two input logic gate by four individual four-input logic gates. The fault correction works on the principle of interwoven signal paths to these four logic gates and alteration of logic functionality throughout the circuit.

5.3. Fault-tolerant approaches based on fault elimination or masking

Fault-tolerance within an electronic system can be performed with the help of two entirely different methods. The first method is to mask the faulty output result out of N-numbers of output results, these outputs are created by independent system structures. By using N-numbers of identical system structures a single fault within one cannot propagate through the entire system because of the masking done by means of output signal comparator for a dual system or majority-voting for N-number of redundant systems. The impact of faults onto the behaviour of these two approaches has been analysed in detail within Chapter 4. For the creation of the N-number output results three different redundancy concepts can be used: spatial (hardware), temporal (time) and pertaining to information [88, 101]. In today's electronic systems the spatial redundancy is used in the majority of fault-tolerant systems and in the form of TMR circuit structure in connection with a majority voter. The second method is to detect, locate and repair the faulty part of the logic structure. This approach of fault-tolerance is achieved through logic structure reconfiguration within an appropriate device.

5.3.1. Redundancy concepts in a system

Redundancy concepts in a system can be broken down into two concepts. The first concept of system redundancy is looking in detail at the creation of a set of output results, from which the overall output result can be determined. To achieve this, three different system-based concepts can be utilised: spatial (hardware), temporal (time) and data (information) [88]. For the two redundancy concepts of spatial (hardware) and temporal (time) a valid output is generated by the use of majority-voting, which is working on the principle of data redundancy. The second concept focused on fine-grained redundancy centred on the transistors of each logic gate to perform fault-masking within the individual logic gates [22].

5.3.1.1. Spatial redundancy system structure

Spatial or hardware redundancy was originally postulated and described in 1956 by von Neumann for high numbers of redundancy [99]. It is also referred to as N-module redundancy (NMR) as displayed in Figure 4.6. N copies of the same logic design or functional block work side-by-side to generate N-numbers of output results, which fall into the category of data redundancy. The resulting N output results are fed into a decision-making circuit, thus creating a majority result out of the N-number output results. This majority result is valid if it is representing the value of more than $^{N}/_{2}$ output results which have the same value [99, 103]. In the case of N=2 this overall output result is generated by a comparator and for N>2 by a majority voter gets used. The original concept of von Neumann was designed for N-number redundant devices where N was a big enough number of duplicated copies of the original functional block design. This concept of using a big enough number of functional block redundancies has the problem of logic hardware complexity and overhead. The most commonly used adaptation of his concept is TMR and it is represented in Figure 4.6 with the setting of N=3. TMR is used in mission critical systems and it is a balance between circuit complexity and reliability. The number of redundancy blocks (R) required for creating an NMR system that can tolerate a required number of faults (E) feeding into the majority voter can be determined by the following equation:

$$R \geq 2 * E + 1 \qquad [104] \qquad \text{(Equation 5.1)}$$

In accordance with equation 5.1 a TMR system can tolerate one fault feeding into the majority voter. If the system is a dual module redundant (DMR) structure the equation 5.1 and $^{N}/_{2}$ can never both be achieved and no real majority-voted result can be generated within this system. Because of the equation $^{N}/_{2}$ the values of R within equation 5.1 will be odd numbers to fulfil the fault-tolerance for a certain number of faults (E). In a DMR system both modules have to generate the same result otherwise if one module is given an incorrect result the DMR system has a 50% chance of generating the correct output result. This means in this case that the DMR system holds and indicates the mismatch by means of a system-fault-flag.

Different types of NMR-based systems can be applied to make a system fault-tolerant. The basic version of an NMR system is the DMR system, which works in lock-step system configuration [101]. In some DMR-based applications the system repeats the execution until the results are matching or until a certain number of repeats have been reached. In this case, the DMR-based system is put on hold and this puts the system into a safe condition. The presence of a fault that cannot be resolved will be indicated by the use of a system-fault-flag. DMR-based systems are used in safety-critical automotive applications like anti-lock break systems (ABS) [79, 101].

The hardware or spatial TMR system is the system which uses three identical copies of the same functional circuit block working side-by-side to generate three output results in a lock-step approach [101]. TMR is the base for all the deviation set-ups applied for making logic designs radiation hardened by design and remains by far the most popular redundant system until today.
Variations of hardware TMR can be:

- Block TMR or BTMR [105] is an older methodology by triplicating the functional block and adding a majority voter.
- Local TMR or LTMR [105] is focused on triplication of the result storing elements within a BTMR and where the data path remains a single path. The resulting output majority-voted value gets fed back into every flip-flop (FF) to correct any incorrectly stored values within the output FFs of the LTMR. This fed-back loop can be seen as a self-correcting circuit.
- Global TMR or GTMR [105] uses the approach of triplicating everything throughout the system and due to this the upset rate regarding faults is very low. The triplication includes the clock and domain circuit of the system, so they are independent of each other. The use of the approach of GTMR can be seen in the use of large chip area overhead and power usage.
- Distributed TMR or DTMR [105] is a basic version of GTMR in which everything gets triplicated but this does not include the global clock-routing and reset. By not including the clock into the triplication like the GTMR it is susceptible against upsets causing faults.
- Selective TMR or STMR [105] only triplicates selective circuits within the system which can be identified as sensitive to SEU-induced faults. Due to the unique identification of sensitive circuits this method cannot be automated by tools [106]. In paper [107] two more different concepts for STMR have been proposed. The first one is coarse-grained TMR or CGTMR referred to in [107] using the method to triplicate large parts of the logic circuit of the system. The second one is fine-grained TMR (FGTMR) [107] which directly triplicates fine parts of the circuits and uses BTMR on these parts.
- Functional TMR or FTMR [105, 108] works on the principle that the functional blocks are triplicated and feed into a triple majority voter circuit. The resulting majority-voted values get stored into triple sequential logic where the output gets fed into a triple majority voter to generate the three independent overall results. These results are fed into the next FTMR block and get fed back into the functional blocks of the first FTMR system for correction of output values, if necessary.
- R-fold modular redundancy or RMR, where R is an odd number for the number of redundant system copies working side by side [109]. In the case of R=3 it represent a TMR system.

- Cascaded triple modular redundancy or CTMR [109] uses three individual BTMRs within its data path for creating a set of three fault-tolerant data paths. So each of these data paths has its own BTMR system. At the end all majority-voted values of the three BTMR systems are fed into a majority voter creating an overall majority-voted result out of these three BTMR subsystems.
- Xilinx TMR or XTMR [95, 107] is part of the Xilinx development platform and can be selected during design and compilation of the logic design. XTMR applies the following logic structural design onto the application design; it triplicates all the following functional blocks of the system: input/output, the throughput logic and inserts fed-back logic for register data correction. In comparison with CTMR, XTMR is more advanced in protecting data with the help of a feed-back register for checking if soft-error has occurred within this data path. Also XTMR is part of the Xilinx design library and will be in use for a number of applications where Xilinx FPGAs chips are used. This will make the XTMR approach most certainly the new industry standard of radiation-hardening for logic devices within FPGA devices. The XTMR solution is coming with a price in logic resources, performance limitations, power consumption and vulnerability of the voter.

All of the described versions above of TMR systems are working in lock-step approach centred on the individual output results. Without the lock-step approach the majority-voting of the different output results would not be possible without mismatches due to timing problems.

5.3.1.2. Temporal redundancy system structure

Temporal redundancy uses redundancy in time differently to the spatial redundancy of the N-type system redundancy approach [19]. The TMR system is the most common approach, which uses three copies of the same functional block to produce a set of output results out of these and with the help of a majority voter an overall output result gets voted. In the case of a transient fault or permanent defect within one functional block one output result of this set of output results will be different and by the use of majority voter will be excluded. The approach of spatial redundancy increases the hardware complexity and if a system is needed where timing is less important than hardware complexity, time redundancy can be utilised [110]. The method of time redundancy works on the concept of creating a set of output results with only one functional block by using it recurrently within a set time frame to create a set of output results. These output results are being stored within separate memory cells. If a similar set of output results comparable to a TMR system performance is required three memory cells are needed and each memory cell is filled after one clock cycle. Novel concepts of memory utilisation could be applied onto the part of the storing of the results generated after each temporal cycle. Within this research work a direct comparison

between TMR, quadded logic structure and QLC within their output result structure is based on the concept of using an N-number input majority voter logic circuit. This is because the majority voter is the logic circuit, which is handling the fault-masking for the feeding logic functional block.
Out of these stored output results a majority-voted result can be formed similar to the TMR system. The idea behind time redundancy is that if any type of intermittent or transient fault occurs it will only happen within one output result creation due to the duration of the effects of the SEU [111]. In the case of a permanent hardware fault in this functional block of the temporal-redundant system all results created at different time intervals will have the same error. The overall majority-voted output result will be affected and the incorrect output result gets chosen. Temporal redundancy systems are designed to handle transient faults and not permanent hardware faults [19]. For handling permanent faults within a temporal-redundant system an addition to the original structure has been proposed in paper [19]. This concept uses data encoding algorithms before execution of the functional block and inverse algorithm for data decoding afterwards for different execution time frames within one general cycle. The sequence of result creation for the logic structure illustrated in Figure 5.1 is working in accordance with a specified process flow. The first result gets generated without data coding, the second one with one type of algorithm and the third result gets created with an altered algorithm. Out of this set of output results an error-free result gets majority-voted and the block diagram for the data encoded temporal-redundant system following the flow defined beforehand is illustrated in Figure 5.1. Through this approach a single permanent fault within the single functional block can be compensated because of using two different coding approaches for the generation of the three output results for the data [19].

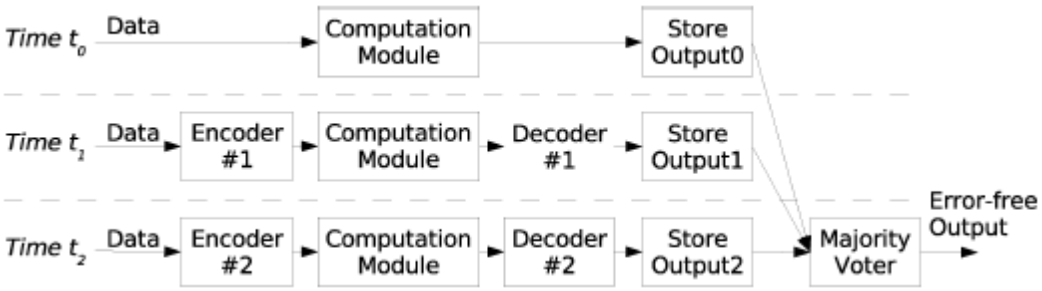


Figure 5.1: Timing sequence of the encoding/decoding approach of the permanent fault-masking temporal redundancy structure [19]

A different concept of using temporal redundancy instead of TMR was proposed in [19] as a time-shared TMR (TSTMR) concept and in [112] as a quadruple time redundancy (QTR) concept. The following chapter describes the concept of these two papers in more detail. Both concepts work with the principle of splitting the functional block into three individual sub-blocks, which are

getting their input data through MUXs. These MUXs are splitting the input data into three sub-data segments of the whole data structure. The resulting outputs out of these three sub-blocks are fed into a majority voter and by the use of a DeMUX unit the corresponding sub output result data gets generated. The TSTMR block diagram of an adder is delineated in Figure 5.2. With the TSTMR structure errors correcting adder and multipliers have been created accordingly. Similar to the concept of splitting the data into three blocks as done for the TSTMR concept, the QTR concept splits the data into four blocks. So instead of using three clock cycles for the TSTMR concept the QTR concept needs four clock cycles to generate a set of four output data result structures. These data result structures are fed into a majority voter for the generation of the output data structures. The disadvantage of TSTMR and QTR is to generate the suitable MUX and DeMUX units, which are, in this case, to be implemented within an FPGA, and are susceptible to SEUs. This would make this structure unreliable.

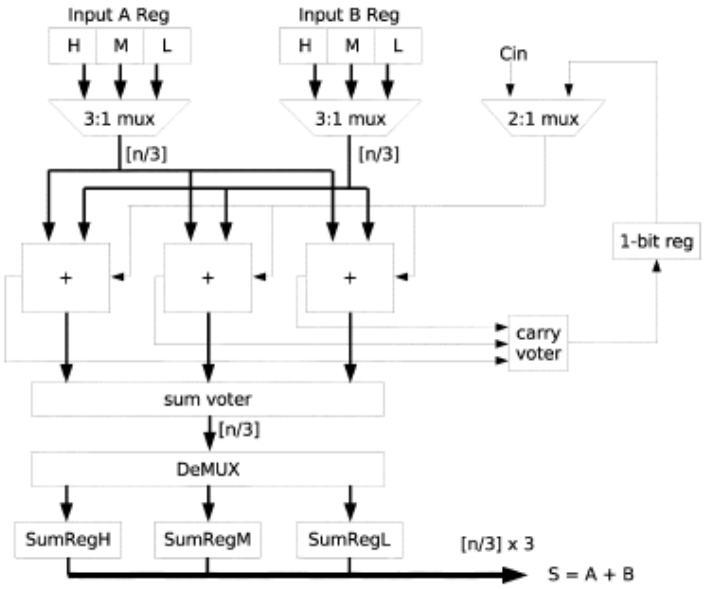


Figure 5.2: TSTMR error correcting adder [19]

5.3.1.3. Information redundancy data structures

The information redundancy works on the principle that additional data or information is being added to the information or to protected data stored or used within an electronic system. The added data facilitates detection and correction of faults within the information data. Information redundancy can also be used in an approach of storing redundant copies of the same data at different locations or memory units. This concept of working with multiple copies of the same data is data redundancy and out of this set of individual values a common value has to be generated. The

generation of this common value out of this set of values is performed by a majority voter. Information redundancy adds extra bits to the information data, which protects against transient errors or permanent faults within the memory cells used to store the data. This data, which got added to the original data, is reflecting in a way the content of the stored information in memory so that in the case of an alteration of the information it can be detected and regenerated. For storing this information data plus added protection data more memory cells per individual information word are required and because of this efficient compact protection concepts are needed.
The most commonly used error-correcting code (ECC) is based on the Hamming code, which is designed to detect a certain number of faults within the original data and is also able to correct them or indicate the presence of an un-correctable amount of them. Beyond a certain number of faults within the data ECC is capable of detection without fault correction. The fault-tolerance mechanism of the ECC is based on arithmetic equations or specific data structures. This principle of Hamming code was invented in 1950 by R.H. Hamming [113] and since then many variations tailored for a certain application, which are based on this mechanism, have been proposed and implemented over time.

5.3.1.4. Fine-grained redundancy on logic gate level

Fine-grained transistor redundancy is centred on the approach of adding redundant transistors within the logic gate. This adding of redundancy to a common gate is done with the focus of improving the fault-tolerant behaviour of this specific logic gate. The fault-tolerance enhancement done through the logic gate could empower the gate to mask certain types of faults or indicate the presence of non-maskable faults. The part of indicating of non-maskable faults is the area this thesis will focus on and can be seen as an innovated concept. With manufacturability in mind the adding of redundant transistors is best done in complements of two. This is because for manufacturing these redundant transistors can be created by adding only two parallel strips of p- and n-diffusion material to the existing design. It could be possible that the redundant transistors are using possibly the same poly-silicon input lines, which makes it easily addable to the common logic gate chip design [12]. Most effective combinations are based upon adding an odd number of transistors so that the original transistor is replaced by an even number of transistors. By using the redundancy rule of adding transistors only by even numbers of redundant transistors this rules out odd-based redundant transistor structures. The maximum number of transistors added as redundancy to a single original logic gate transistor was limited to three within this thesis. Thus in total a quadded transistor structure is replacing one logic gate transistor. Beyond this point the created fault-tolerant gate structures defeat the proposed target of this work of creating the smallest possible gate structure to cope with certain types of stuck-at faults.

Within [21] from Naran S. et al it was proposed that dual transistors as redundancy are getting added in parallel to the circuit structure and in [22] from El-Maleh A.H. et al quadded transistor structure are being proposed for replacing a single transistor of a given logic gate. By using dual transistor structure this type of gate will not be capable of masking one permanent fault. But it is capable of masking SAL faults, which has been verified within [20] from Djupdal A. et al with the help of finding the best redundant structure by using evolutionary principles to find the best transistor structures for enhancing fault-tolerance to a given logic gate function. An SAL resilient inverter logic gate was the result of this investigation, which has been found within the paper [20] and is described in Figure 5.3. The structure of this altered logic gate has a parallel redundant transistor structure for each original logic gate transistor.

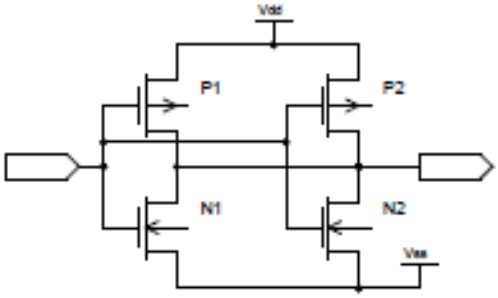
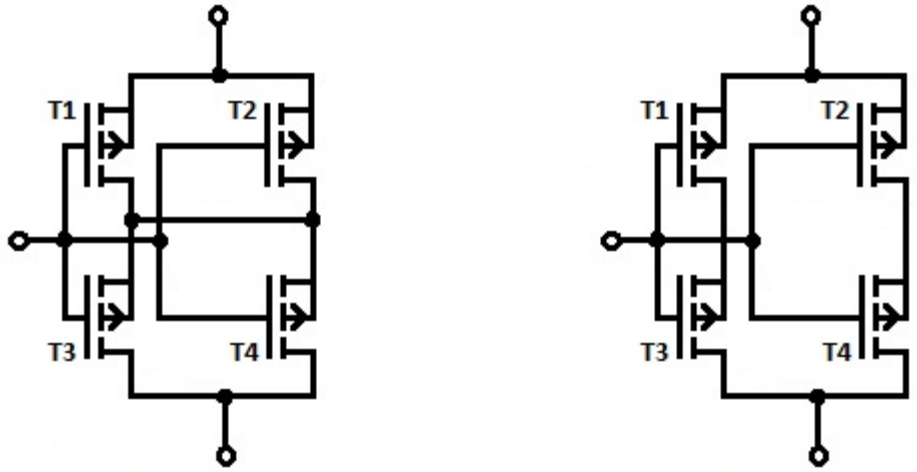


Figure 5.3: Best evolved SAL resilient inverter gate [20, 21]

The quadded transistor replacement structure proposed in [22] is represented in Figure 5.4 and can also be described as N^2 transistor structure. This structure is capable of handling $(N-1)$ permanent faults within each single quadded transistor replacement structure. This masking of an SAH fault within this transistor structure is possible as long as the fault is only one permanent SAH fault per replacement transistor structure of an original logic transistor. Investigation performed has shown that this design can tolerate certain combinations of two permanent SAH faults but this depends upon their locations. Also within Figure 5.4(a) the cross connection indicated between the centres of the quadded transistor structure has an impact on the fault-tolerance. Without the cross connection (see Figure 5.4(b)) two independent SAH faults within each signal path are possible, with the cross connection (see Figure 5.4(a)) only if both SAH faults are present within the top or bottom part of each of the signal paths.



(a) (b)

Figure 5.4: The two possible replacement quadded transistor structures for a single transistor of a common logic gate [22]; (a) with and (b) without cross bridge

5.3.2. Reconfiguration concepts in a system

The concept of fault-tolerance by reconfiguration of the logic design can only be achieved by using suitable chips composed of a uniform logic structure that are configurable by software. This uniform reconfigurable chip structure includes programmable logic, interconnection and everything associated with configuration capability through memory [114]. These chip structures can be found within commercial off-the-shelf (COTS) chips like FPGAs. FPGAs chips support two different reconfiguration options. The first one is the reconfiguration at run-time or dynamically and the second applying reconfiguration only to a defined part of the device, which is called partial reconfiguration of a logic block [115]. Customised chips that are also capable of offering reconfiguration features are mostly designed for a certain application and not for adapting a general application on the fly. In general, reconfiguration of a given logic circuit design requires the capability of altering the way the logic circuit design is implemented within a fine-grained logic elements structure provided within a given chip. Different methods of reconfiguration can be utilised for constructing fault-tolerant logic circuit designs.

5.3.2.1. Data scrubbing

The logic gate design configured within an FPGA is executed and stored within most modern-day FPGAs within SRAM elements. Chapter 2.3.3 shows that the assignment of SRAM within an FPGA can be 50% to 90% of the actual memory of a particular FPGA type. In [116] the allocation of configuration memory of a Xilinx Virtex XCV1000 chip was 97.4% of the total memory bits. Through these SRAM-type bits the configuration of the logic circuit gets set and alteration of this information changes the designed circuit structure. SEUs altering the memory information can only be detected through evaluation of the results generated through this circuit structure or through read back of the data stored in the configuration memory. After the read back a comparison against a golden copy reveals SEU-related bit alteration. The altered data can then be overwritten and this task is referred as data scrubbing [117, 118]. This process is also described as read-back scrubbing. Hardware related faults within the chip cannot be detected through this approach. The technique of data scrubbing is not directly an approach of reconfiguration by altering a given circuit structure due to a fault within a certain part of the chip structure. Data scrubbing, or better described as rewriting the original configuration information rather, is a re-establishing of the intended logic circuit design defined through the configuration data. Scrubbing can be divided into internal and external scrubbing. Internal is done with the help of ECC associated with configuration memory banks. In the case of a single-bit alteration through an SEU the altered bit within the configuration data can be detected and restored with the ECC controller. External data scrubbing of a device is divided into blind and read-back data scrubbing. Blind data scrubbing is writing the golden copy of the configuration data stored in an external memory into the FPGA regardless of whether a fault has occurred or has not occurred. The concept of read-back data scrubbing involves first reading back the entire data of the device and checking for data alterations. In case of a found data alteration this data and only the altered data gets written.

5.3.2.2. Reconfiguration with pre-defined data

In the case of a permanent fault within a logic circuit design configured data within an FPGA the approach of using reconfiguration with pre-defined configuration data requires that the general layout of the FPGA structure is divided into equal blocks, in this example into column-based blocks. Each block contains a certain part of the whole design. In the case of a fault within one function block, a predefined unused block is used to act as a replacement for this faulty block. The configuration data of the different function blocks can be assigned to the blocks to the right of this replacement block [23, 24]. This principle of logic design reconfiguration of a given logic design inside an FPGA by means of block-dependent rearrangement is shown in Figure 5.5. Within this figure is demonstrated the occurrence of a hardware fault within the functional block 3, which is

currently used for function D of the application's logic design. This block experiences a fault which requires logic structure reconfiguration. The internal structure inside the FPGA is rearranged due to the hardware fault present and detected in FPGA block 3 that the currently unused FPGA block 5 (see Figure 5.5(a)) is used for the functional block D reconfiguration (see Figure 5.5(b)). This reconfiguration involves the shift of functional block C into FPGA block 4 and the functional block D into FPGA block 5. After the reconfiguration the unused FPGA block 5 before the presence of the fault is now FPGA block 3 which contains the hardware fault.

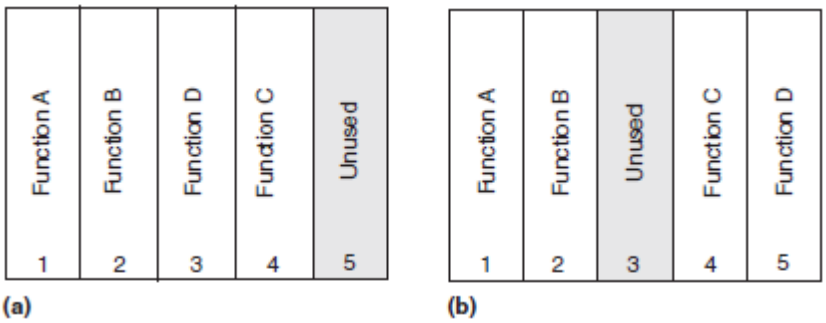


Figure 5.5: Column-based precompiled individual functional blocks. The fault-free configuration is displayed in (a) and an altered configuration after a fault is shown in (b) [23, 24]

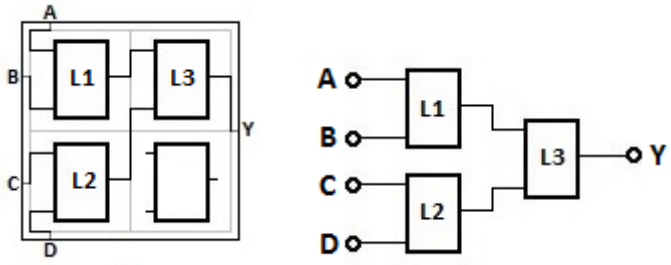
The division of the entire FPGA structure into homogeneous pre-compiled blocks can be done in any shape and size and this all depends on the application design of the system. The specification of certain block structures within the given application design has to be done during the design of the logic structure prior to the compilation of the configuration file data.

5.3.2.3. Tile approach with rotating reconfiguration

The four-tile approach within a given logic cell is defined in [25] from Lach J. et al. This logic structure has a fixed input/output interface and contains four-tiles. The structure of this tile approach is shown in Figure 5.6(a). The logic functionality of each tile is not pre-defined or fixed and hence the tiles may be regarded as a configurable logic unit and are implemented within the CLBs of the FPGA. Each logic cell can be a unique logic function selected or programmed into an LUT out of the functionality of a CLB. For example within the four-tile approach, which is illustrated in Figure 5.6(a), a fixed logic circuit has been defined as shown in Figure 5.6(b). The tiled logic structure implements the fixed logic circuit using three out of the four logic units: the remaining logic unit acts as a spare in the case of a hardware fault of another one. The interconnection between different logic cells is not part of the investigation and proposed solution of [25] for a fault-tolerant system solution. In the case of a fault within the interconnection

structure complex reconfiguration of this interconnection structure for replacing a faulty logic cell would require a set of pre-compiled configuration data for each possible arrangement. Another solution for this problem could be during run-time with the help of an embedded microcontroller on the FPGA chip or with the help of an external arrangement. All these approaches are complex and would have a noticeable impact on the availability of the system during interconnection reconfiguration. A possible solution for the reconfiguration problem of this logic cell matrix could be by creating a fixed line structure for routing input or output signals through this fixed line structure. This concept would require a fixed amount of lines between logic cells, which is given within an FPGA. But within an FPGA the routing is done by means of the interconnection and which is optimised during compilation of the logic design through the compiler tool. For creating this bus-type interconnection structure it would have to be forced during compilation or a unique chip design has to be created.

Reconfiguration of the interconnections between the internal logic units does not affect the input/output interface of the logic cell. Any type of reconfiguration of the internal structure follows a pre-defined arrangement, which is illustrated in Figure 5.7 subsections I to IV, in the form of a clockwise reconfiguration.



(a) (b)

Figure 5.6: (a) Logic cell with four logic units in accordance with [25];
(b) Internal logic structure created out of the three logic gates

The fixed logic circuit designed in each logic cell forms the logic circuit shown in Figure 5.6(b). With this logic circuit the following logic function for an example is taken out of [25] and has been created:

$$\text{Boolean function } Y = (A \wedge B) \wedge (C \vee D) \qquad \text{(Equation 5.2)}$$

The implementation within a logic cell of this Boolean function (Equation 5.2) from above is shown in Figure 5.7 Hardware fault detection within this logic cell has to be done by external functionality and in the case of a hardware fault, reconfiguration data is used, which is also stored

externally within a memory-based functional block. After the detection of a hardware fault within a logic cell a pre-defined altered configuration is applied, which alters the logic cell accordingly. The predefined reconfiguration data mimics a clockwise rotation of the configuration until a fault-free configuration has been detected by the external functional checker. The limitation of this approach arises by virtue of not directly identifying the logic unit that has a hardware fault. Through the clockwise rotational reconfiguration the identification of the faulty logic unit can be achieved because presumably the newly created spare logic unit is the one with the hardware fault. This cannot really be in the case of the concept of critical and non-critical logic gate alteration, which is illustrated in Table 5.1.

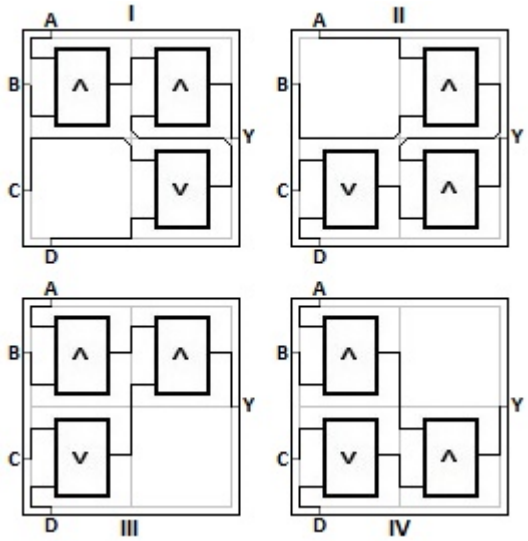


Figure 5.7: Clockwise reconfiguration of the internal circuit structure for maintaining the required Boolean function [25]

5.4. Fault-tolerant approach based on fault-masking

The concept of fault-masking gets used within an electronic system to prevent any propagation of faults through the electronic system and in the case of a fault it gets masked at a functional boundary. The first idea of using redundant information paths assessed by a majority voter had been first introduced by von Neumann in 1952 in an oral form and 1956 in paper form [99]. This concept is in use as a TMR system with a majority voter as the minimal solution of this concept. The approach of fault-masking within an electronic system can be done by knowing what is the correct output result or statistical evaluation of a set of output results of a given electronic system. The first method of knowing the resulting output values of an electronic system triggered by a certain input stimulus can make the whole logic system obsolete. The logic system is obsolete because why have a complex logic circuit if all the output results are matched to the corresponding

input stimulus and could be programmed into a memory accessible by means of addressing? By defining all this input/output information it is also possible to replace the logic circuit with an appropriate memory chip. The addressing of the memory chip is used for the input stimulus translation into the correct memory address. The stored data at this address is the corresponding output result for this input or address stimulus. In the case of using the memory-mapped solution as a logic circuit checker it has to work parallel to the logic circuit. In this case the assumption has to be that the checker system is fault-free throughout its operation. If a fault-detecting electronic system is required a combination of two systems in lock-step approach will create the required fault-detecting system. One of these systems is going to be the original logic circuit and the other one the checker system. This combination of two systems would be halted in the case of a detection of a mismatch between both systems, which also does coincide as fault-indication. For this case a supervisor checker system would be required to determine the correct response for this situation of the system at this point. The supervisor checker is also required to contain the correct results for the current input stimulus as the checker system. In this way the entire system transforms into a majority-voting system, which is more or less the same solution as the second method for fault-masking. This system can also be defined as a TMR system with majority-voting. Within this system or in any other system using a majority voter a fault-masking logic structure has been added to the original functional structure. This majority voter is a vital functional block where applied within any fault-tolerant system.

5.5. Fault-tolerant approach based on fault correction

The concept of creating an electronic system with the capability of fault correction deviates from the concept of fault-masking. Fault-masking within an electronic system is based on detection and correction of a fault at a functional boundary. A system with fault correction works on the concept of using logic circuits, which enables the logic structure to correct faults by means of its logic circuit arrangement. The concept of fault correction and performing the required logic functionality at the same time was introduced by Tryon in 1958 with the quadded gate logic structure [119, 120]. This original work focused on the logic gates AND, OR and NOT. In 1963 Jenson expanded the logic gate selection with the NOR gate [102] and with this the whole fundamental range of basic logic gate functionality was covered. The quadded logic gate structure cannot perform fault-masking, it is more likely the kind of failure correction by the use of a given logic arrangement and which performs the required logic function at the same time. Quadded logic gate structures require a majority voter for performing the fault-masking. It requires four times the logic circuit compared to a standard logic circuit design and each replacing logic gate becomes a four-input one [102, 119, 120]. Quadded logic circuits can correct all single faults within the structure through interwoven redundant logic structure [37]. The concept of interwoven redundant logic structure is applied onto

the input interconnection between the different layers of logic gates. With the help of replicated input signals to each logic gate in accordance with a specific pattern, the fault correction within the quadded gate structure can be achieved. In [37] Pierce also introduced with interwoven redundant logic structure the concept of critical and subcritical errors between logic gates. Interwoven redundant logic structures like the quadded logic design can fix permanent and transient faults until the last layer of logic gates. Faults appearing at the last interconnection layer will affect the output results in a way that an equal amount of zeros and ones are fed into a majority voter - (i.e. a majority-voted result does not exist).

For the investigation of the fault-correcting capability of the interwoven logic structure the type of faults will be limited to stuck-at faults i.e. SAH or SAL at the interconnection structure. Internal effects of stuck-at faults injected at the individual transistors will not be done for this fault investigation. The analysis of the fault-behaviour caused by the individual logic gate transistors and increasing their resilience against stuck-at faults through redundancy is part of Chapter 7. The results of this fault-handling capability of a quadded logic system will be based on fault-behaviour regarding SAL and SAH individual results and will be based on comparison of FR. The FR is calculated with the equation 4.2 for all the different simulation cases analysed within this chapter. An example for the FR calculation can be seen in appendix 2.

The resulting impact of stuck-at fault conditions for different types of logic gates is illustrated in Table 5.1 [37]. The definition of a critical fault is that a stuck-at fault at the input will lead to a stuck-at fault at the output of this logic gate. A subcritical fault for a logic gate is that a stuck-at fault at the input will not cause a stuck-at fault at the output of this logic gate.

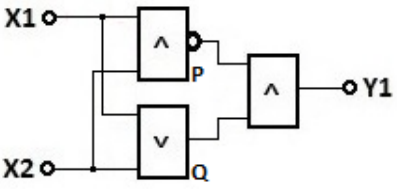
Function	Subcritical error in the input	Critical error in the input	Output error due to critical error
AND	0 → 1	1 → 0	1 → 0
OR	1 → 0	0 → 1	0 → 1
NAND	0 → 1	1 → 0	0 → 1
NOR	1 → 0	0 → 1	1 → 0

Table 5.1: Critical and subcritical faults within different logic gate types [37]

Applying quadded logic design structures to a given logic design means that every logic gate is replaced by four logic gates and each having four-inputs. An example of transforming an XOR logic gate built out of individual gates (see Figure 5.8(a)) into quadded logic gate structure is shown in Figure 5.8(b). With logic equation 5.3a to 5.3c the logic functionality of the XOR logic gate is described. With logic equation 5.4a*x* to 5.4c*x* the logic functionality of the quadded logic-based XOR logic gate is defined and the interwoven input arrangement can be observed. All these

equations are the foundation of the logic simulation under stuck-at fault-injection at specific points delineated in Figure 5.9 in subfigures (a) and (b).

(a)



(b)

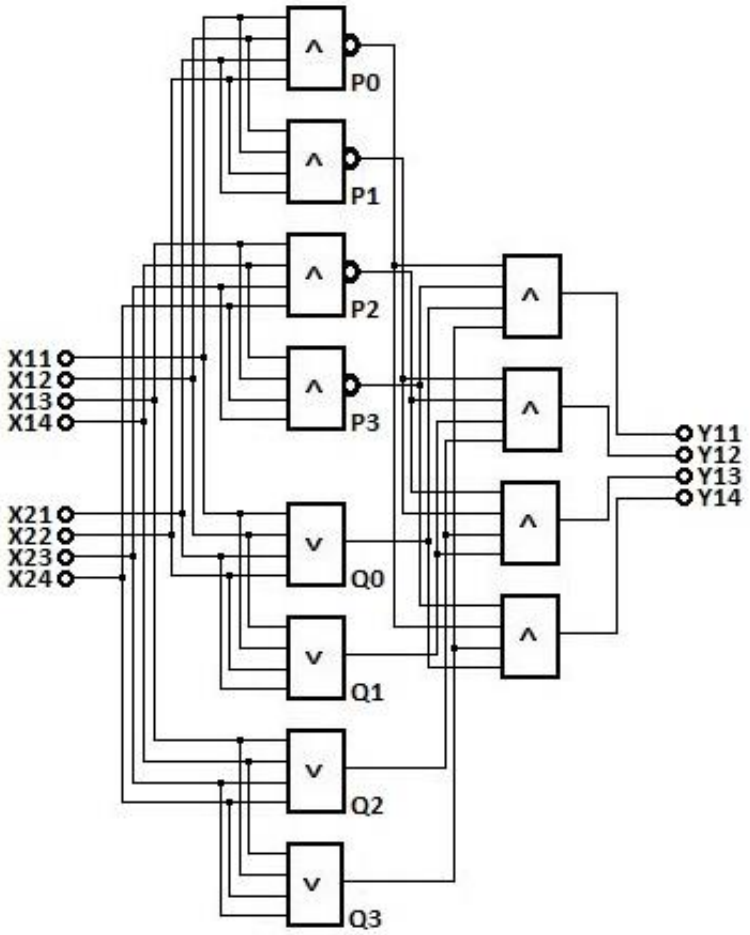


Figure 5.8: XOR logic gate design in (a) standard logic gate structure and (b) quadded logic gate structure

Logic equation describing the behaviour of the XOR logic gate designed out of three logic gates in accordance with circuit shown in Figure 5.8(a).

$$P = \overline{X1 \wedge X2} \qquad \text{(Equation 5.3a)}$$

$$Q = X1 \vee X2 \qquad \text{(Equation 5.3b)}$$

$$Y1 = P \wedge Q \qquad \text{(Equation 5.3c)}$$

The logic equations for the quadded logic design of the XOR logic gate (see Figure 5.8(b)) designed on the basis of the standard XOR logic gate shown in Figure 5.8(a).

$$P0 = \overline{X11 \wedge X12 \wedge X21 \wedge X22} \quad \text{(Equation 5.4a1)}$$
$$P1 = \overline{X12 \wedge X11 \wedge X22 \wedge X21} \quad \text{(Equation 5.4a2)}$$
$$P2 = \overline{X13 \wedge X14 \wedge X23 \wedge X24} \quad \text{(Equation 5.4a3)}$$
$$P3 = \overline{X14 \wedge X13 \wedge X24 \wedge X23} \quad \text{(Equation 5.4a4)}$$

$$Q0 = X11 \vee X12 \vee X21 \vee X22 \quad \text{(Equation 5.4b1)}$$
$$Q1 = X12 \vee X11 \vee X22 \vee X21 \quad \text{(Equation 5.4b2)}$$
$$Q2 = X13 \vee X14 \vee X23 \vee X24 \quad \text{(Equation 5.4b3)}$$
$$Q3 = X14 \vee X13 \vee X24 \vee X23 \quad \text{(Equation 5.4b4)}$$

$$Y11 = P0 \wedge P3 \wedge Q0 \wedge Q3 \quad \text{(Equation 5.4c1)}$$
$$Y12 = P1 \wedge P2 \wedge Q1 \wedge Q2 \quad \text{(Equation 5.4c2)}$$
$$Y13 = P2 \wedge P1 \wedge Q2 \wedge Q1 \quad \text{(Equation 5.4c3)}$$
$$Y14 = P3 \wedge P0 \wedge Q3 \wedge Q0 \quad \text{(Equation 5.4c4)}$$

The standard XOR logic gate design (Figure 5.9(a)) contains 9 fault-injection points and the quadded logic XOR logic gate design (Figure 5.9(b)) contains 68 fault-injection points. At each fault-injection point SAL or SAH faults are statically applied for the duration of altering each possible input combination at the circuit inputs, which is in this case four-input combination. The corresponding output values generated for each input stimulus have been evaluated against the known good value. Figure 5.9(b) shows four-inputs instead of the two inputs of the standard XOR logic gate design in accordance with Figure 5.9(a). At these four-inputs of the quadded logic gate structure a set of four equal input values is applied and no faults affecting these inputs are subject of this simulation. For the standard XOR logic gate the output value is a single bit and for the quadded logic XOR logic gate design a set of four output bits. The evaluations of the sets of bits are done by comparison of the individual bits against known good values. For the overall evaluation of the accuracy of the resulting output sets an evaluation by the use of a voter simulation indicates if in a case of a faulty output this fault can be masked or not.

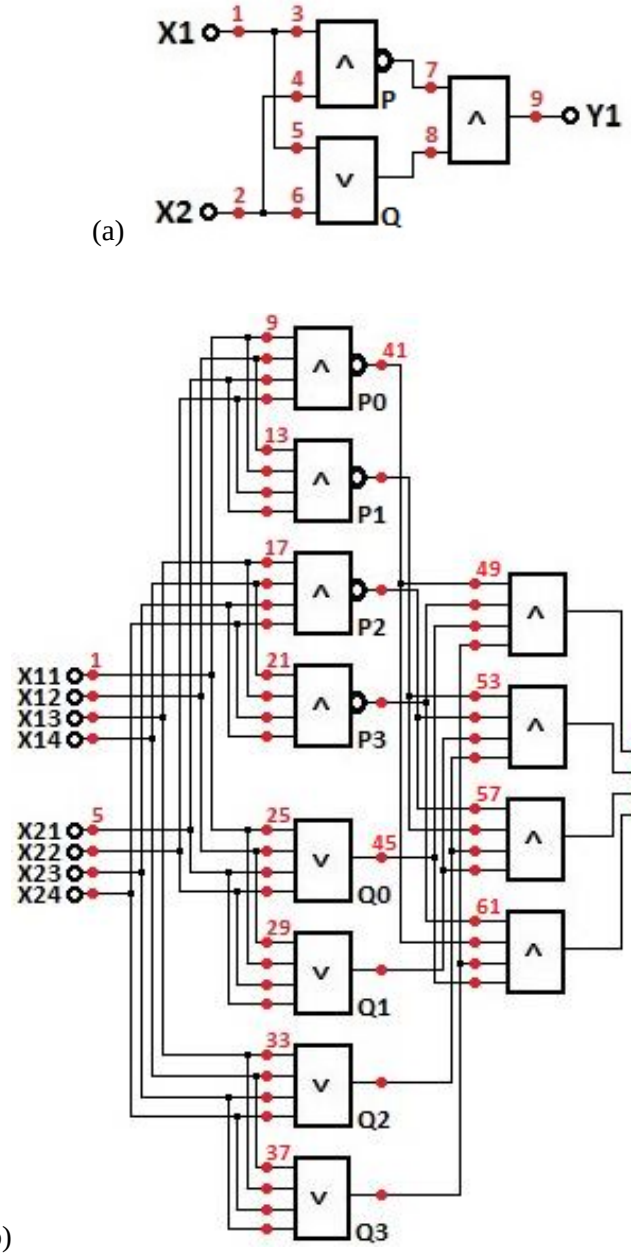


Figure 5.9: XOR gate design in (a) standard gate structure and (b) quadded gate structure both with specific defined stuck-at fault-injection points

Table 5.2 represents the results of the stuck-at fault-injection simulation for the standard XOR gate represented. The results indicate that by means of injecting a fault at every injection point certain faults are being corrected and others show an impact on the output values of the logic structure. The total FR for all nine fault-injection points and all possible fault stimuli for the standard XOR logic gate is 36.1%. Due to the single bit nature of the output value, masking of this fault is not possible and the results will have an effect on the overall circuit. A standard XOR logic gate by itself cannot be identified as fault-free under the influence of stuck-at faults injected at the defined injection points. In accordance with the simulation results shown in Table 5.2 it is shown that the

most impact on the combined fault-behaviour effect onto this logic structure are the injection points 1, 2 and 9. Including these points within the simulation has been important because of the nature of comparing a logic unit against another type of logic unit structure and the interface points are the central points of every unit. Points 1 and 2 are the corresponding input pins and a stuck-at fault at this point will affect the input stimulus pattern. Point 9 affects the output behaviour of this logic gate construction in a way that a permanent output value is present. A fault-tolerant version of the standard XOR logic gate is only possible with the help of additional checker hardware or by creating a TMR-style XOR logic gate. As found with the analysis of the standard XOR logic gate the fragile points are both inputs and the output. This fault condition of faulty central inputs can be applied onto the TMR version of the XOR logic gate. The central input, feeding into the TMR structure, is the corresponding weakest point similar to the inputs of the standard XOR logic gate. The majority voter in this regard is sharing the same fault-behaviour as the standard XOR logic gate and this can be expanded onto any logic structure. Central inputs and outputs of logic circuits in this regards are the main weak points for influences by faults.

Fault point	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Fault SAL	2	2	1	1	1	1	2	2	2
Fault SAH	2	2	1	1	1	1	1	1	2

Table 5.2: Breakdown of the different fault results of the fault-injection at the different injection points of the standard XOR logic gate displayed in Figure 5.9(a)

The quadded logic design of the XOR logic gate (see Figure 5.8(b)) has 68 fault-injection points (see Figure 5.9(b)) where stuck-at fault-injections are going to reveal the fault-behaviour of this logic structure. The individual result evaluation of the output responses after stuck-at fault-injection for the quadded logic gate structure is illustrated in Table 5.3 in a way that each single bit deviation of the output result set of the quadded logic structure is counted as an individual fault.

Injecting a stuck-at fault-type at the fault-injection points 1 to 40 (see Figure 5.9(b)) can be corrected within the interwoven quadded logic circuit design and no alteration of the output result set deviates from the defined good output values. For injection points 41 to 68 each injected SAL fault has an effect on the output result set. The output values creating logic gates for this design are AND logic gates and according to Table 5.1 the critical fault condition, which alters the output is an SAL fault at the input of this type of logic gate. This SAL fault simulation is equivalent to a logic gate output stuck-at low feeding into the AND logic gate. This is a possible fault condition for a logic gate and the effect on the performance of the quadded logic circuit is tremendous. Injecting an SAL fault into the injection points 41 to 48 of the quadded logic circuit (see Figure 5.9(b)) alters the output result set in such a way that an equal distribution of zeros and ones in the output result set is generated. By means of this output result distribution a majority voter circuit is connected to

the outputs of the quadded logic circuit, which is shown in Figure 5.9(b). The shown logic circuit is not capable of determining the correct majority-voted output value and defaults to a zero output value. This fault effect on the resulting majority-voted output value is caused through the combined injection of the SAL fault into the interwoven redundant signals feeding into the inputs of two output-creating logic gates at the same time. The incorrect resulting output sets corresponding with the stuck-at fault-type injection at points 49 to 68 (see Table 5.3) can all be masked through a majority voter circuit and will not have a negative effect on the following logic circuit of the whole system.

Injection point	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Fault SAL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fault SAH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Maskable	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Injection point	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
Fault SAL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fault SAH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Maskable	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Injection point	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
Fault SAL	0	0	0	0	0	0	4	4	4	4	4	4	4	4	2	2	2
Fault SAH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Maskable	--	--	--	--	--	--	no	no	no	no	no	no	no	no	yes	yes	yes

Injection point	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68
Fault SAL	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Fault SAH	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2
Maskable	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes

Table 5.3: Breakdown of the different fault results of fault-injection at the different injection points of quadded logic XOR logic gate in accordance with Figure 5.8(b)

Altering the circuit components of the standard XOR logic gate, which is shown in Figure 5.8(a), into a version with a NOR logic gate as the output logic gate changes the sensitivity of it to another output dependent critical fault. In accordance with Table 5.1 this critical fault for the NOR logic gate is the SAH condition. This can be translated into an SAH fault, which has been injected into the fault-injection points creating similar fault-behaviour like the SAL fault affecting the logic circuit shown in Figure 5.8(a), with AND logic gate creating the output. For the comparison between fault-behaviour the quadded logic design of the XOR logic gate displayed in Figure 5.8(b), was adapted in accordance with the logic gate configuration that is shown in Figure 5.10. The similar fault-injection test which was used to create Table 5.3 was applied onto this circuit and the resulting fault-behaviour is illustrated in Table 5.4.

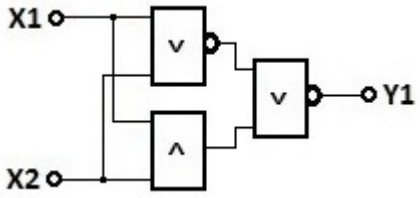


Figure 5.10: XOR logic gate design in standard gate structure with altered output logic gate different from figure 5.8(a)

The equal fault-behaviour in response to the stuck-at fault-injection similar to the one illustrated in Table 5.3 is delineated in Table 5.4 for the quadded logic circuit designed in accordance with the logic gate definition displayed in Figure 5.10. The same non-correctable output condition is present within this data and triggered through SAH fault injected at points 41to 48. This behaviour follows the definition defined within Table 5.1 and affects the output result set of the quadded logic structure. These output result sets cannot be fixed by means of the interwoven interconnect structure or masked through a majority voter. These faults are fault cases where the quadded logic design cannot fix a stuck-at fault and these faults generate an output result at the majority voter which defaults to a given value. This value can be correct or not but this is indeterminate by the standpoint of fault-tolerance. Calculating the FR for these discovered fault cases of the quadded logic reveals that it is 5.9%. This FR is taking only the faults where the majority voter is not capable of masking the fault present at the output of the quadded logic circuit. The FR for the ones, which can be masked by the use of a majority voter, is 8.8%. By taking all the faults present at the output as either maskable or non-maskable the total FR is 14.7% and this is, for a fault-tolerant concept not an expected value, especially that all the faults are related to the last logic gate set of the quadded logic structure. The corresponding fault-injection points are 41 to 68. All of these fault-injection points affect the resulting output values. The output-creating logic gates of the quadded logic structure are the most vulnerable ones and would need a logic structural enhancement to become fault-tolerant.

Injection point	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Fault SAL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fault SAH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Maskable	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Injection point	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
Fault SAL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fault SAH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Maskable	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Injection point	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
Fault SAL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fault SAH	0	0	0	0	0	0	4	4	4	4	4	4	4	4	2	2	2
Maskable	--	--	--	--	--	--	**no**	**no**	**no**	**no**	**no**	**no**	**no**	**no**	**yes**	**yes**	**yes**

Injection point	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68
Fault SAL	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2
Fault SAH	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Maskable	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**

Table 5.4: Breakdown of the different fault results of fault-injection at the different injection points of quadded logic XOR logic gate transformed out of Figure 5.10

5.6. Summary of the chapter

The focus in this chapter was placed on the different concepts of fault-masking or correcting within a given logic circuit. Fault-masking works on the concept of majority-voting by the use of a set of output results to generate the majority-voted output value. The set of output results can be generated by the use of spatial (hardware), temporal (time) or data (information) redundancy. Spatial and data redundancy are the most frequently applied concepts within logic systems to mask a fault. Temporal redundancy comes with the disadvantage of the required timing to generate a set of results if used in time-critical applications. The advantages of temporal redundancy are reduced hardware requirements and good at handling transient effects causing data alteration within the hardware. Another disadvantage of this concept is that in the case of a permanent hardware fault within the functional logic block all the output results of the result set are altered in the same way. Due to this constant alteration of all output results it will become the majority-voted output result. For overcoming this effect in temporal-redundant logic system hardware reconfiguration or other approaches can be utilised for creating hardware alteration, which create unique hardware set-ups for each output result creation. As found within this chapter each of the three redundancy concepts has disadvantages in at least one area of logic circuit structure. A novel concept would be, if it was possible, to combine all three redundancy concept within one fault-tolerant systems approach and through the combination disadvantages of one redundancy concepts could be resolved by another redundancy concept.

Fault correction within a given logic structure requires a specific logic design to perform the required logic functionality and fault correction at the same time. The concepts of quadded logic structure fulfil both of these proposed requirements and have been published in associated papers. For evaluation of the fault-correction capability of a quadded logic circuit a given circuit was injected with stuck-at faults and the output results were compared against the known good ones. The FR generated from these incorrect output results had been used as an indication of the fault-correction capability of quadded logic circuits. In general any quadded logic circuit requires a majority voter for generating a single majority-voted output result and at the same time for masking of a certain percentage of faults present within the output results. The quadded logic circuit is relying on the majority voter regarding masking faults which are being generated within the circuit structure and because of this are present within the output results at the output of the quadded logic structure. A certain set of faults generated within the quadded logic circuit cannot be masked with the help of the majority voter or by the interwoven interconnection structure of the quadded logic structure. These types of faults show that the quadded logic circuit is not completely capable of correcting all faults within its logic structure.

In all cases of fault-masking the majority voter is the central functional block to make a logic system fault-tolerant by the means of fault-masking. As shown in the previous chapter the FR of a

majority voter requires hardware alteration to the logic circuit of a common majority voter to be more fault-tolerant. These hardware modifications are needed to make the voter fault-free regardless of the fault happening within its circuit structure. The impact of the majority voter in terms of the FR requires further investigation with the goal of creating a fault-tolerant majority voter logic circuit for stuck-at faults by the use of altered logic gates.

The following question arose out of this chapter. Is it possible to alter the fine-grained transistor structure of a logic gate to be better equipped against stuck-at faults at the transistor level with a minimal hardware overhead and what impact on a given logic circuit can be achieved? Does this altered logic gate design offer a feature, which could be utilised for an intrinsic built-in feature for initiation of circuit alteration without the influence of external logic circuitry? Would it be possible to combine the three redundancy concepts spatial (hardware), temporal (time) or data within one overall redundant concept and what kind of impact has this concept on the FR compared against quadded logic structure? Can it be done to create an FSM with minimal fault-tolerant hardware fulfilling the task of fault location identification within a given logic structure?

Chapter 6: Design of a fault-tolerant temporal-redundant matrix element

6.1. Introduction

Within this chapter the question stated in Chapter 5 concerning combining the three redundancy concepts spatial (hardware), temporal (time) or data within one overall redundant concept is going to be further investigated. The combination of the three redundancy concepts will originate a new logic structure within a fixed functional block. This logic structure will contain an overlap of the fault-handling capabilities of these redundancy theories and its ability will be evaluated by fault-injection. The effectiveness of the new concept will be evaluated against the quadded logic structure. This direct comparison has been selected for the fact that both structures comprise the feature of fault correction and generate equal number of output results. The quadded logic is achieving this through spatial redundancy and a distinct fixed gate interconnection. In contrast the newly created logic structure utilises temporal triggered logic gate rearrangements out of a fixed number of logic gates for achieving the same fault behaviour.

A logic system designed for accomplishing certain functionality cannot by itself be fault-tolerant without increasing the logic complexity and hardware of the desired logic functionality. The increased complexity of the fault-tolerant logic system reflects the fault-tolerant approach chosen by the system designer required to meet the specification of the system. Fault-tolerance by masking a fault of a logic system requires a set of results, out of which a majority voter can generate the majority-voted output result, under the assumption that more than half of the output results are valued ones. Temporal-redundancy reuses the same logic hardware for a specified number of times to generate an independent set of output results from each other. Permanent hardware faults within this logic hardware system will generate consistent faulty output results. By overcoming this constant effect of a permanent hardware fault within a part of the logic system it can be addressed by using temporal-dependent hardware reconfiguration. This temporal-dependent logic hardware reconfiguration requires a newly designed logic structure, which can be time-triggered and altered accordingly to the necessary logic functionality. The newly designed logic structure is based on the concept of a matrix structure due to the reconfigurable requirement. This matrix structure is designed with the capability of using a defined logic overlapping for every output result generation out of its given matrix structure. This logic overlap is altered with each timing cycle and it also excludes some logic functions for this duration. Through this non-fixed and overlapping hardware logic usage identification of faults within this matrix structure can be achieved and reacted on.

6.2. A fault-tolerant temporal-redundant structure

The concept of fault-tolerance is based on the level of functional complexity which is involved and how it can be distinguished between the levels of functional complexity. The functional complexity can be broken down into fine-grained and coarse-grained functional complexity. Fine-grained complexities specify the functional complexity to be a single functional one and through combining several of these fine-grained units a higher-level functional complexity can be achieved. Coarse-grain complexity specifies the functional complexity as an ALU and if required a memory circuit. Coarse-grained structures can perform functionalities on their own [49]. Out of this a fault-tolerant system is a coarse-grained functional unit. The coarse-grained fault-tolerant logic systems work on the principle of fault-masking or fault correction for containing a fault within a given functional block and preventing the fault from propagating through the system to become an error. The concept of fault correction requires a certain type of logic structure, like the quadded logic structure. This logic circuit structure involves a robust design of logic and interwoven interconnection for the logic functionality. The generation of the single-valued output result of a quadded logic system is generated by the use of a majority voter. The concept of fault-masking is working on the principle of using a defined number of redundant functional logic blocks to produce a set of output results independently of each other. In this regard it is working with the data redundancy concept for the set of output values. These sets of results are processed by a majority voter to vote on the majority result. Both concepts require a majority voter and this is why the voter is a vital functional block within any fault-tolerant system. Faults affecting the majority voter are altering the majority-voting of the overall output result and counteract any fault-tolerance put in place for generating the set of output results feeding into the majority voter.

A fault-tolerant electronic logic system, which is based on majority-voting, requires a set of N-number of individually generated and stored output or results, out of which more than $^{N}/_{2}$ of the output results represent the same value [99, 103]. The most commonly used fault-tolerant logic design is the TMR structure feeding into a majority voter, (see 4.6.1. majority voter at the boundary of a functional block) which masks single faulty output results of one subsystem. In this regard a TMR system is based on a spatial redundant concept to produce a set of three output values, which are data redundant. TMR-based systems are designed for time-critical logic designs due to the simultaneous generation of the three output results within the same time frame. This task of generation of three independently produced logic system output results requires three identical logic circuits working side by side and this increases the logic hardware requirement by 200% overhead without the majority voter logic circuit. The reduction of the hardware overhead of a fault-tolerant system can be done with temporal redundancy reusing one set of logic hardware a given number of times to generate by temporal difference independent output results from each other. These results are stored in separate memories, one per each generated output result. These

stored output results are evaluated by a majority voter, which polls on the common output result. A system, which is using temporal redundancy, can be described as a temporal-redundant system (TRS). A TRS is designed for handling transient faults affecting its logic hardware and preventing these faults from propagating beyond functional block boundaries. The effect of a transient upset onto any logic structure can only become a fault if the deviation of the logic value coincides with the storage of the altered output result or the intermittent internal value of this logic circuit. In this way transient faults within a temporal-redundant logic system can, in the best case, modify only one value of the set of output results and this depends on the occurrence of the frequency of the transient upsets. A permanent fault within the logic hardware used by the TRSs alters the output result sets in a consistent way during the generation of the individual output results. Due to this consistent fault within the set of the output results the majority-voted result will reflect it as a common factor amongst them. In this way permanent hardware faults within the functional logic block (FLB) of a TRS require a similar redundant FLB to generate independent output results, which can be evaluated against the other output results in a lock-step approach. By expanding the TRS with a redundant FLB a hardware-redundant structure has been assembled similar to a dual hardware-redundant system. This system increases the hardware requirement and takes the TRS away from reducing unnecessary hardware overhead.

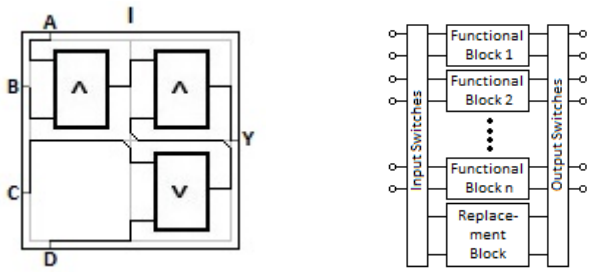
How can a logic structure be based on the three redundancy theories and show what kind of impact this combination will have on the fault-handling ability? This question can and will be answered within this chapter through the creation of a temporal-dependent reconfigurable ‘‘round-robin’’ matrix element for creating a set of data redundant output results. The logic structure design will be based on the three redundancy theories and is combining their fault-handling capabilities into forward-thinking features. In this chapter the capability of this matrix element of handling faults is the goal of this thesis. The concept of making a TRS resilient against stuck-at faults is based on the idea of temporal-depending alteration of the logic gate structure generating for the output value during one clock cycle. By using a fixed number of clock cycles a set of independently generated output results will be generated. This set of output results can be seen as data redundancy concept. Altering the logic circuit structure of the FLB of the TRS in accordance with the generation of each output result bears the approach of not having a permanent fault affecting logic gate functionality constantly present in the used logic circuit. This temporal-depending reconfiguration of a logic circuit is embedded into a defined matrix cell, which can be used to build a matrix element. The fault-handling capability of this matrix element gets evaluated against known fault-tolerant logic circuit structures. Can a temporal-dependent reconfigurable matrix element be as good as or even better than a quadded logic circuit structure performing the same logic functionality?

6.3. Design of a fault-tolerant temporal-dependent reconfigurable round-robin element

For creating a fault-tolerant logic system two different approaches can be applied within a logic system, fault-masking with the help of majority-voting or fault-correcting within a given logic structure. A TMR system uses fault-masking by the use of a majority voter and a quadded logic-based system combines both fault-tolerant approaches. Influence on the output value of a fault-tolerant system based on fault-masking by majority-voting requires that more than $^{N}/_{2}$ of the data redundant output results have to be created under the influence of a given fault or faults before the voted output results are affected. In this chapter, the focus of the number of faults present in a logic system, which is going to be analysed is limited to one stuck-at fault only within a given logic structure. Because of this a comparison of the fault-handling capability of the temporal-dependent reconfigurable round-robin matrix element against a TMR-based system is not possible. For the creation of a noticeable majority-voted output result alteration at the voter of a TMR system it requires two FLB of this TMR system to be under the influence of at least one fault at the appropriate logic circuit location, which is capable of altering the output value. The only time a TMR-based system can be altered by one fault only, is when the fault happens at one of the common inputs feeding into the three FLBs of the TMR system and altering an input signal all the time. A quadded logic structure generates four independent output results coming out of four individual logic gates at the output layer with interwoven interconnection between each gate layer. By design a quadded logic-based system should be fault-tolerant through fault-masking and correcting against faults happening at its interwoven interconnection network. In Chapter 5 the analysis of the behaviour of a quadded-based logic circuit revealed through Table 5.3 and Table 5.4 that by applying stuck-at faults at certain fault-injection points at the interwoven interconnection structure the creation of faults at the output of the circuit occurs, which are non-maskable faults.

The objective of this concept of temporal-dependent reconfigurable round-robin matrix element was to create a logic structure similar in output results numbers and equal or better fault-tolerant behaviour like a quadded logic system structure. The matrix element incorporates the three redundancy concepts spatial (hardware), temporal (time) and data (information) for achieving its fault-tolerant behaviour. The matrix element, which can also be seen as cluster, has to be designed with fever logic gates and interconnections between logic gates than required for a quadded logic circuit. For the achieving of these objectives within a matrix element, a combination of the tile-reconfigurable matrix structure proposed in [25] by Lach J. et al and the reconfigurable logic block proposed in [26] by Koal T. et al has been utilised within this matrix element for providing configurable logic functionality within this matrix element. Within Figure 6.1(a) the principle of the tile-based reconfigurable matrix structure has been developed and is outlined in [25] by Lach J. et al with the focus of limited localised reconfiguration through pre-defined reconfiguration data for this cell divided into tiles in case of a fault within one tile. This matrix structure with its general

structure has been used as the central component for designing the temporal-dependent reconfigurable round-robin matrix element. The alteration applied to its behaviour was to use time dependent reconfiguration of the internal four-tile structure instead of fault triggered reconfiguration performed by an external system. The temporal reconfiguration is performed through adding a switchable interconnection structure between the four-tile elements. These interconnection switches are controlled by a programmable time-triggered shift-register. By doing so the concept of temporal and spatial redundancy of achieving fault-tolerance has been applied and can be identified as one functional principle. The reconfigurable logic block proposed in [26] by Koal T. et al was developed with the focus of maintaining a required logic functionality within given and fixed access points regardless of a fault present within its block. The block diagram of the reconfigurable logic block is illustrated in Figure 6.1(b). This concept of reconfigurable logic functionality has been designed into each of the four-tiles of the matrix element without deleting the proposed replacement block for internal fault-tolerance. Also the proposed functional blocks within its structure were replaced by fixed logic gate functionality. As defined within the concept of [26] the internal switches remain unchanged but they are controlled by the same time-triggered shift-register controlling the initial functional principle. Due to these changes to the internal tile structure they had more logic units and represented the second functional principle. The second functional principle is applying spatial redundancy within the logic unit. The logic unit can be seen as fine-grained logic granularity and through this the level of logic complexity has been defined for this research work. Coarse-grained logic functionality has not been selected due to the fact that this research work focused on demonstrating a fault-tolerant concept possible through this approach. Also this work is limited to the analysis of the fault-tolerance of a single matrix element and not a multidimensional array of these matrix elements performing elaborate logic functions. These array structures can be achieved in principle with this matrix element, but further research work has to be performed on the design of these array structures, which are beyond the research objective of this thesis.



(a) (b)

Figure 6.1: (a) Matrix structure divided into tiles which can be localised reconfigured in the case of a fault within a single tile [25];
(b) A reconfigurable logic block between fixed interconnection points for maintaining a logic functionality in the case of a fault within a functional block [26]

The temporal-dependent reconfigurable round-robin matrix element combines both functional principles and the resulting functional block diagram combining both principles is illustrated in Figure 6.2. The central part of this matrix element is within the four logic units, which provide the necessary flexibility for fulfilling the logic function alteration triggered by the clock cycle. Before each time-triggered alteration the current specified logic structure will perform the required logic functionality by using the input stimulus for generating an output value. The number of time-triggered alterations defines the number of output values and due to the comparison against the quadded logic structure, four output values will be generated. This set of output values represents the concept of data redundancy and the last of the three redundancy concepts utilised within the matrix element to achieve fault-tolerance.
Due to the fact that this matrix element contains four logic units it can also be described as quadded logic cluster (QLC). Each of the different functional blocks of a QLC which are shown in Figure 6.2 has the following functionality:

- register block
- switching unit
- logic unit

The register block of the QLC is shown in Figure 6.2 as the central controlling block and is realised in the logic circuit as a loop-back shift-register. The function of the loop-back shift-register within the QLC is to control each switch within the QLC elements switch and logic unit. The required logic functionality can be configured for the logic circuit design within the individual logic units by the programming of the configuration data into the shift-register. With every clock cycle the configuration data within the shift-register gets shifted by one position and the logic functionality

for the associated logic units is altered accordingly. Through the loop-back of the shift-register the configuration data rotates around and this can be seen as temporal-controlled round-robin reconfiguration of the matrix element or QLC. The other block of the QLC functional block diagram shown in Figure 6.2 is also controlled by the means of this shift-register and this is the switching block for controlling the inter-block connection.

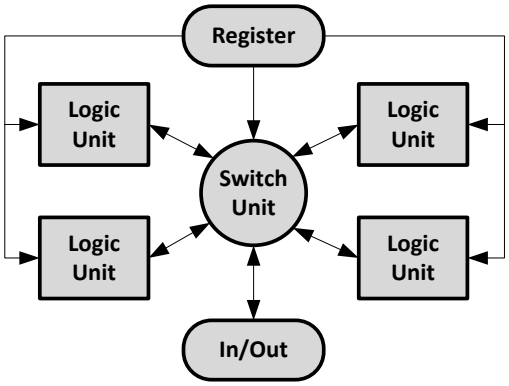


Figure 6.2: Functional block diagram of the temporal-dependent reconfigurable round-robin matrix element

The functional block diagram of the QLC is transformed into a general block diagram of the QLC, which is illustrated in Figure 6.3. The clockwise orientation pointing arrow in the centre of the general block diagram of the QLC represents the temporal-dependent loop-back shift-register, which is controlled by a central clock and is illustrated in Figure 6.4(a). The shift-register is divided into four sections SR1 to SR4 and each section is linked to the corresponding logic unit. For instance the SR1 section is linked and controlling the logic unit 1 or is also defined as A (see Figure 6.6(b)). As shown in Figure 6.2 the shift-register is controlling the switch unit and the configuration of the logic unit. Through the switch unit the shift-register is controlling the selection of a defined number of logic units within one clock cycle to be used for performing required logic functionality. The selection of the logic unit is done through the control line SU1 to SU4 of the associated shift-register section SR1 to SR4 and is shown in Figure 6.4(a). The choice of the selected logic function has been done by means of switches inside each logic unit and is shown in Figure 6.4(b). The logic function required is done by means of selecting the required logic gate through the switches S1.x of the logic unit 1, which are controlled through the corresponding shift-register section for example. All the required reconfiguration of the QLC is done by means of switches and not by reconfiguration through reprogramming a section of a configurable chip, like a FPGA. Because of the use of switches for the temporal-triggered reconfiguration the implementation of the QLC within a COTS chip like an FPGA is not feasible and for further fault-tolerant behaviour analysis software simulation has to be used.

The four logic units of the general block diagram of the QLC represent the reconfigurable logic blocks needed to alter the logic functionality of the single logic unit of the QLC. The alteration of the logic functionality can be done out of a set of logic functions and in accordance with the designed and required overall logic circuit. The logic functionality within a logic unit is altered as a physical logic gate controlled through switches and not as a memory-based look-up table done in FPGAs-based logic circuit designs. The internal physical logic gate structure of a single logic unit is delineated in Figure 6.4(b). By choosing switchable physical logic gates at this state of the QLC instead of memory-based look-up tables the logic circuit design at stuck-at simulation at the different interconnection can be compared to other logic circuits. Also using physical logic gates eradicates the susceptibility against SEUs and eliminates a lock-step redundant checker system working parallel to the QLC for fault checking. The resulting fixed logic gate configuration within a QLC per each clock cycle uses only three out of the four logic units. The generic fixed logic gate configuration is outlined in Figure 6.5(a). The table described in Figure 6.5(b) shows the different selectable physical logic gate functionalities within a single logic unit of a QLC. The table also contains the relevant coding information for the selection of the logic functionality. This coding or configuration information is written to the shift-register of a QLC and is shifted by one clock cycle. After four clock signals the full round-robin cycle has finished and the relevant output results stored within memory. Because of the four clock signals the QLC will generate four output results comparable to the number of output results of a quadded logic structure and these four independent output results can be seen as data redundancy. The selection of this four basic logic gate functionality and the resulting internal logic unit circuit makes it possible to adapt other logic gate functionality like XOR or XNOR with the help of an entire QLC.

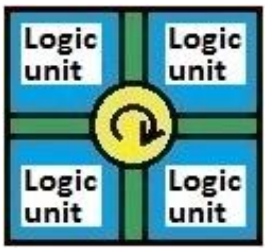


Figure 6.3: General block diagram of the quadded logic cluster

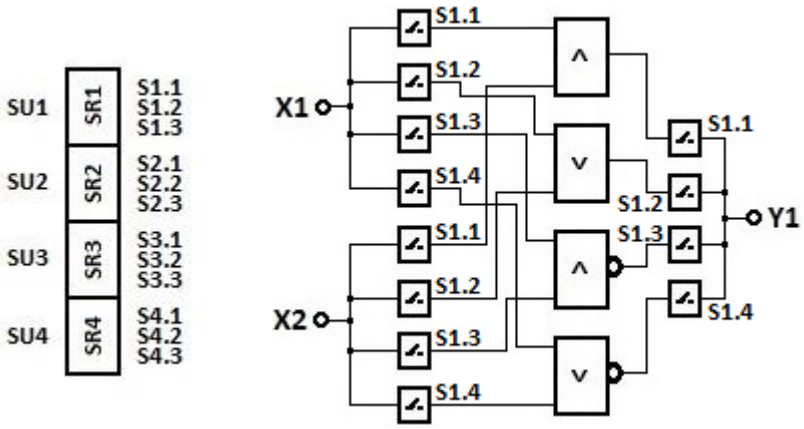
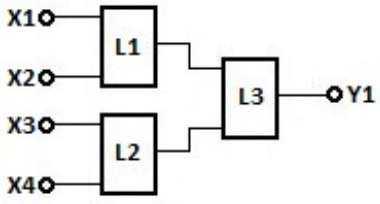


Figure 6.4: Functional blocks of the QLC matrix element; (a) the shift-register which controls the selection of logic units and the selection of the logic gate functionality; (b) internal structure of logic unit with switches for selecting logic gate functionality



Select	Coding	Logic Fkt
1	001	AND
2	010	NAND
3	011	OR
4	100	NOR

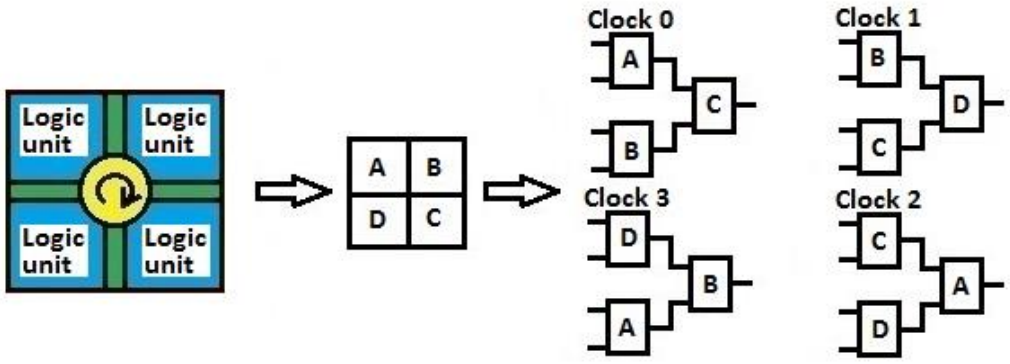
(b)

Figure 6.5: (a) Internal logic gate combination of the QLC per one clock cycle; (b) Logic function corresponding to the required selection

The functionality of the QLC is based on the concept of altering a fixed amount of logic units per clock cycle, which is present within a QLC as logic unit structure for creating and maintaining the pre-defined logic gate circuit structure (see Figure 6.5(a)). The associated logic units of the QLC used for the creation of the pre-defined circuit structure are exchanged in accordance with the clock cycle. The required logic functionality specified for each logic unit of the pre-defined circuit structure will be maintained through the data linked to each logic unit by the shift-register data. By only utilising three out of the four possible logic units within a QLC it is guaranteed that between each clock cycle an overlay of $^{2}/_{3}$ of the logic units through the fixed logic circuit exists (see Figure 6.5(a)). By using this overlay between each clock cycle a faulty logic unit rotates through the pre-defined circuit structure and for one clock cycle it will not be used. Through this concept of using different logic gate functionality for each of the logic unit of the pre-defined fixed logic structure, a faulty logic gate within a logic unit will only be used within one clock cycle throughout the four clock cycles. By applying this approach the faulty logic gate within one logic unit will only affect

one of the four output results and so can be identified. This approach of using different sets of logic units with a defined overlay will have an effect on dormant faults, which can in the best case only be unnoticed for one clock cycle.

During each clock cycle alteration only one logic unit is exchanged out of the fixed logic circuit in accordance with the pre-defined circuit schema. The resulting different pre-defined logic circuits configurations, which are going to be created out of the four logic units for the four matrix clock cycles (defined and shown in Figure 6.5(a)) are illustrated in Figure 6.6(b). Within Figure 6.6(a) the four logic units are labelled for reference purposes with the letter A to D. Figure 6.6(b) shows the different pre-defined logic configurations utilising the appropriate logic units labelled with these letters. The configuration is utilising three out of four logic units in a round-robin approach altered per matrix clock cycle. The matrix clock cycle defines the internal matrix count and it is triggered by a central clock. By comparing the used logic units at two different succeeding matrix clock cycles, for instance matrix clock cycles 2 and 3, represented in Figure 6.6(b) the utilisation of the logic units can be seen. Matrix clock cycle 2 uses logic units A, C, D and matrix clock cycle 3 uses logic units A, B, D. The logic unit overlay of this fixed logic configuration between these two matrix clock cycles is A and D. Both remaining logic units B and C are only used during one matrix clock cycle in this example and a more detailed example for the function of the shift register is outlined in Figure 6.7. Within this example the adaptation of a XOR logic gate function is performed through the fixed logic configuration. The time triggered round-robin function through the shift-register for the four clock cycles is illustrated. For each clock cycle the data within the shift-register and the associated used logic units within the QLC element are outlined. This concept of defined logic unit utilisation within a reconfigurable matrix per clock cycle will be used for fault identification within a QLC.



(a) (b)

Figure 6.6: (a) Block diagram of QLC with labelled logic units,
(b) configuration of logic units in conjunction to round-robin clock

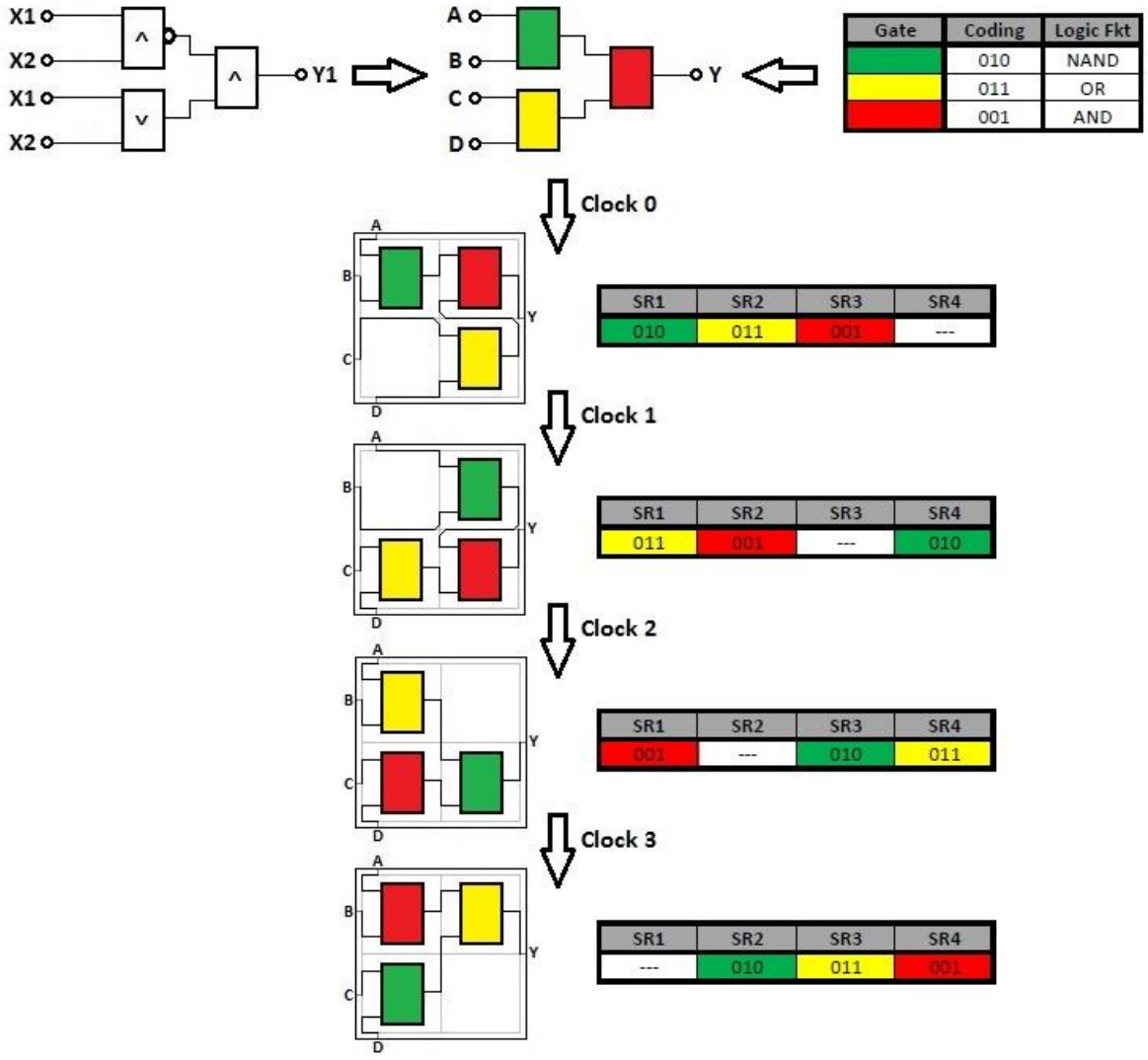


Figure 6.7: Detailed example of the mapping of a XOR logic function onto the QLC elements and shift-register details for the full round-robin cycle

6.4. Fault-handling capability of QLC compared against quadded logic structures

Fault-handling within a logic system per definition can be based on two concepts. The use of these concepts can be done within the logic structure as fault-correcting or at the boundary between functional blocks as fault-masking. The internal fault-tolerance of a logic system is based on the concept of redundancy and usually uses one of these three redundancy forms: temporal (time), spatial (hardware) or data [88, 101]. The most applied approach of spatial redundancy is applied onto logic circuit designs and commonly utilised as N-type identical copies of hardware working in parallel. This structure can be seen as a redundant system and the generation of N-number of outputs can be seen as data redundancy. All the N-number output results of the redundant systems have to be majority-voted to get a single overall output result. This is the concept of boundary-

based fault-tolerance by means of fault-masking without fault correction. The quadded logic circuit structure offers the fault-handling capability of fault correction within its logic structure and fault-masking at the output logic gate interface by the use of a majority voter. The boundary fault-handling applied at the functional block outputs requires a decision-making device, which in most cases is fulfilled through a majority voter.

The evaluation of the fault-handling capability of quadded logic structure vs. QLC is split into two parts. Part one is based on evaluation of fault correction performed by the use of the internal logic structure and the second part is based on the effect of adding a majority voter to the logic structure. For comparing the fault-handling capability of both logic circuits the investigation will be aligned on the fault-rate analysis for each logic circuit.

6.4.1. Fault-handling evaluation of quadded logic vs. QLC, both without voter

The first investigation of the fault-tolerance of quadded logic vs QLC will be done on the basis of fault-tolerance of the logic circuit by itself without using a majority voter for fault-masking. Quadded logic circuits per design are capable of performing fault correction by the use of interwoven interconnection and the use of four logic gates with four-inputs. The QLC works on the concept of temporal-triggered reconfiguration by using a set of logic functionality, which is altered by $^1/_3$ for each clock cycle. Both logic concepts are designed to generate a set of four independently generated output results, which can be seen as data redundancy. But how independent is the generation of this set of output results in the presence of stuck-at fault-injection at the inputs and outputs of the individual logic gates of each logic structure? The stuck-at faults are going to be injected into the inputs and outputs of each logic gate within each logic structure. This investigation will show the fault-tolerance capability of both these logic circuits. Fault-behaviour investigation of the impact of interconnection between logic gates of the quadded logic and QLC structure has not been done and has not been specified for this research work as fault-free. In this analysis work all the used switches within the QLC matrix element performing logic circuit alteration are defined as fault-free. This is because of the fact that their fault-behaviour would create erratic logic structures, which is beyond the set scope of this thesis.

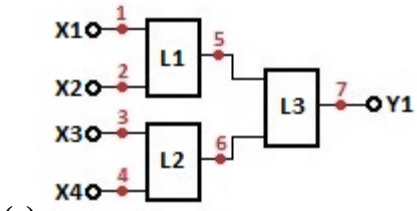
For the evaluation a fair comparison of the fault-handling capabilities for these two different logic circuits a common logic structure must be used. For this analysis the pre-defined logic structure that is defined in Figure 6.5(a) is going to be used. This circuit structure is created within the QLC matrix element at each matrix clock cycle with the help of interchanged use of logic units. The fault-handling capability of the QLC is compared against the quadded logic structure performing the pre-defined logic structure with alteration of the logic functionality within the logic units by using a defined set of logic gate functionality. Both fault-injection evaluations of the logic circuit are done within MATLAB simulations. The MATLAB simulation performed the required logic

function as a true logic gate function. No memory mapping was performed. The faults injected into the inputs or outputs of a logic function were done by altering the required variable before the logic function evaluates the input data for generating the output. In the case of a fault of the output of logic gate the output value was altered accordingly after the logic function evaluation. For generating the possible faulty outputs the entire input range was evaluated one by one and the resulting outputs out of the pre-defined logic structure with the selected logic function combination were stored in an array. This data array was compared against the same output sequence generated by a fault-free version of the pre-defined logic structure. Each deviation of the comparison was counted and the FR was calculated with the equation 6.3. The MATLAB code and an example of the logic structure evaluation can be found within appendix 3.3.

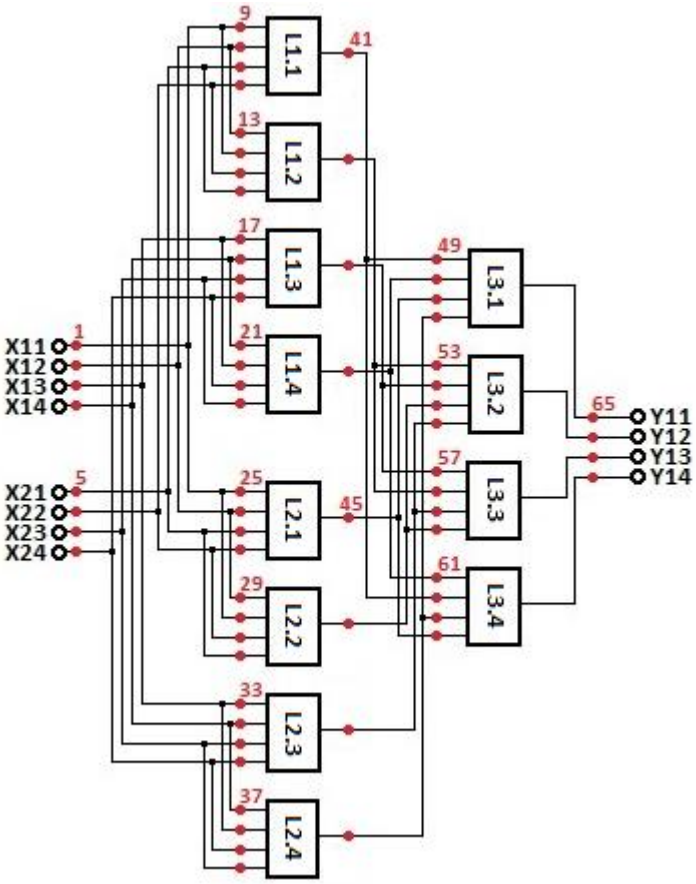
This fault-handling evaluation is performed by applying all the different logic gate combinations possible at each logic gate specified within the table of Figure 6.5(b). The resulting output values of the circuit under influence of the injected stuck-at faults are compared against the known good output value of the fault-free logic circuit one. By applying this method of fault-injection into both logic circuits a distinction between maskable faults (M), output values which deviated from the correct value as faults (F) and non-maskable faults (NM) can be made. The sum of faults (F) and NM faults of one type of logic circuit under the influence of injected stuck-at faults is the total number of faults. These types of faults are the deviation from the correct output value of the fault-affected logic circuit and these types of faults can propagate throughout the functional boundary into the next functional block of a complex system. The definition for maskable faults (M) means that $^{N}/_{2}$ of the output values at the majority voter contain the same value and these output values match the correct output value compared with the logic structure without a fault being injected. Non-maskable faults are faults where the output value set, which are going into the majority voter, are equally distributed between zeros and ones. In this case the majority voter will generate a zero output value as a majority-voted result due to the internal logic circuit structure (see Figure 6.10 for a four-input majority voter). In some cases the majority-voted output value of zero is the correct value expected for this input stimulus. This fault-behaviour condition is not given in all possible cases of this logic circuit.

The logic circuit structure for the fault-tolerance evaluation of QLC vs. quadded logic structure is based on the pre-defined logic structure outlined in Figure 6.5(a). For this pre-defined logic structure design N=64 different logic gate combinations are possible based on the logic functionality defined within the table of Figure 6.5(b). The resulting FR of each logic gate combination after the SAH and SAL fault-injection has been evaluated and the resulting FR has been determined. The resulting FRs is shown within a table for each design. The structure of these tables is that each column of this table is identifiable through the variance of the logic functionality of the pre-defined logic structure. Instead of the logic functionality the selection information out of the table, which is displayed in Figure 6.5(b), has been used for writing the selection number into

the column fields of the resulting FR of both logic structures within their analysis result tables. This concept is also applied on the result table of the reference result table to make the three tables comparable based on their resulting FR.



(a)



(b)

Figure 6.8: (a) Shows the logic gate configuration for logic function alteration and fault-injection points at the inputs and outputs of each logic gate; (b) shows the same as (a) but for the quadded logic structure

For the evaluation of the fault-tolerance of both logic structures under the influence of SAH and SAL injected faults the FR for each variation of the logic configuration applied onto the pre-defined logic structure has been established. The pre-defined logic structure is demonstrated in Figure 6.5(a) and is built out of individual logic gates without any fault-tolerant hardware features

and this logic structure is the reference logic gate structure for fault-tolerance evaluation. Figure 6.8(a) shows the fault-injection points for this generic logic gate structure and these points are going to be used in principle for the two other logic circuit structures with fault-tolerance. The QLC internal logic structure, which is performing the same logic structure defined within Figure 6.8(a), has the same fault-injection points at inputs and outputs of each logic gate as the reference logic structure. The generic quadded logic structure is defined within Figure 6.8(b) and the corresponding fault-injection points located at each input and output of all logic gates are also defined within this illustrated figure.

The FR results of the SAH and SAL fault-injection simulation applied onto the reference logic gate structure performing the pre-defined logic circuit are represented in Table 6.1. This table represents the logic combinations possible by using the four logic gates selectable within each logic unit. The total number of combinations can be calculated by:

$$N_{var} = N_{logic\ gates}^{N_{logic\ units}} \qquad \text{(Equation 6.1)}$$

$$N_{var} = 4^3 = 64$$

The number of possible logic variations within this set-up is 64 and the different variations are represented within the table through the columns *L1, L2 and L3*. Each logic set-up is colour coded in accordance to the definition of Figure 6.5(b). Each row of this table represents the applied variation of one of the 64 possible logic gate functionalities in accordance with the logic gate selection defined within the table of Figure 6.5(b). The FR of each row of Table 6.1 represents the sum of all individual FRs after applying all possible input stimuli at the logic structure, while being under the influence of stuck-at faults at one of the defined injection points. The values of these faults are shown in the table within the *F* column. For the calculation of the FR of one of these logic gate variations the total number of possible output variations had to be defined. This value can be calculated with:

$$N_{output\ var} = N_{Fault\ types} \cdot N_{input\ var} \cdot N_{fault\ injection\ points} \qquad \text{(Equation 6.2)}$$

$$N_{output\ var} = 2 \cdot 16 \cdot 7 = 224$$

For this logic gate structure, which is defined in Figure 6.8(a), the number of possible output variations is 224. This reference logic gate structure has no fault-handling capability in regards of fault-masking or correcting due to the lack of a reference output value feeding into a majority voter or comparator or input signal redundancy, which are feeding into a set of redundant logic gates. Because of these missing fault-handling capabilities each of the faults is a fault that is a deviation to the correct output result and will propagate through the system. The propagation of this fault

through the single logic gate structure will be seen as an error of this system due to the lack of fault-tolerant circuit features as illustrated in Figure 4.4.

Within this reference logic gate constellation, each of the 64 logic gate variations show deviation of the output value under the influence of a stuck-at fault injected at the injection points defined at Figure 6.8. Comparable evaluation can be done on the basis of FR numbers. FR of a logic system can be calculated in the following way:

$$FR = \frac{total\ number\ of\ faults}{total\ number\ of\ possible\ output\ logic\ results} \cdot 100 \qquad \text{(Equation 6.3)}$$

The fault range defined by minimum and maximum of the FR for the reference logic gate structure evaluation shows the following values taken out of Table 6.1. The minimum $FR_{min} = 14.3\%$ and the maximum of $FR_{max} = 28.6\%$ have been evaluated for the reference logic structure under the influence of stuck-at faults. Both values are common results for a set of logic gate variations shown in Table 6.1. Table 6.2(a) displays all the logic gate variations for the minimum FR_{min} and Table 6.2(b) displays the same for the maximum FR_{max} logic gate variations taken out of Table 6.1. Within both tables the breakdown of the faults is done by the causing fault-injection point. The data reveals that the majority of the faults causing injection points are around the logic gate L3 for the reference logic gate structure, which is the output-producing logic gate. Due to the similarity of the FRs documented within Table 6.1 for the different logic gate variations the Table 6.2(c) shows the faults per injection point breakdown as an example for the other sub-tables. As shown in Table 6.2(a) and Table 6.2(b) the fault-injection points, which are causing the most faults are around the output-generating logic gate L3 for all the different logic gate variations. The fault-causing injection points are affecting the inputs and the output of the L3 logic gate.

This evaluation of the fault-handling capability of a reference logic gate design forms the basis of this comparison and each one of these fault-tolerant logic gate designs needs to show better FR results.

L1	L2	L3	F	%FR
1	1	1	32	14,3
1	1	2	32	14,3
1	1	3	64	28,6
1	1	4	64	28,6
1	2	1	56	25,0
1	2	2	56	25,0
1	2	3	40	17,9
1	2	4	40	17,9
1	3	1	56	25,0
1	3	2	56	25,0
1	3	3	40	17,9
1	3	4	40	17,9
1	4	1	32	14,3
1	4	2	32	14,3
1	4	3	64	28,6
1	4	4	64	28,6

L1	L2	L3	F	%FR
2	1	1	40	17,9
2	1	2	40	17,9
2	1	3	56	25,0
2	1	4	56	25,0
2	2	1	64	28,6
2	2	2	64	28,6
2	2	3	32	14,3
2	2	4	32	14,3
2	3	1	64	28,6
2	3	2	64	28,6
2	3	3	32	14,3
2	3	4	32	14,3
2	4	1	40	17,9
2	4	2	40	17,9
2	4	3	56	25,0
2	4	4	56	25,0

L1	L2	L3	F	%FR
3	1	1	40	17,9
3	1	2	40	17,9
3	1	3	56	25,0
3	1	4	56	25,0
3	2	1	64	28,6
3	2	2	64	28,6
3	2	3	32	14,3
3	2	4	32	14,3
3	3	1	64	28,6
3	3	2	64	28,6
3	3	3	32	14,3
3	3	4	32	14,3
3	4	1	40	17,9
3	4	2	40	17,9
3	4	3	56	25,0
3	4	4	56	25,0

L1	L2	L3	F	%FR
4	1	1	32	14,3
4	1	2	32	14,3
4	1	3	64	28,6
4	1	4	64	28,6
4	2	1	56	25,0
4	2	2	56	25,0
4	2	3	40	17,9
4	2	4	40	17,9
4	3	1	56	25,0
4	3	2	56	25,0
4	3	3	40	17,9
4	3	4	40	17,9
4	4	1	32	14,3
4	4	2	32	14,3
4	4	3	64	28,6
4	4	4	64	28,6

Table 6.1: Results of fault simulation in accordance of logic gate alteration applied onto Figure 6.6 (a) reference logic gate circuit performing the fixed logic structure of Figure 6.5(a)

L1	L2	L3	F	%FR	1	2	3	4	5	6	7
1	1	1	32	14,3	2	2	2	2	4	4	16
1	1	2	32	14,3	2	2	2	2	4	4	16
1	4	1	32	14,3	2	2	2	2	4	4	16
1	4	2	32	14,3	2	2	2	2	4	4	16
2	2	3	32	14,3	2	2	2	2	4	4	16
2	2	4	32	14,3	2	2	2	2	4	4	16
2	3	3	32	14,3	2	2	2	2	4	4	16
2	3	4	32	14,3	2	2	2	2	4	4	16
3	2	3	32	14,3	2	2	2	2	4	4	16
3	2	4	32	14,3	2	2	2	2	4	4	16
3	3	3	32	14,3	2	2	2	2	4	4	16
3	3	4	32	14,3	2	2	2	2	4	4	16
4	1	1	32	14,3	2	2	2	2	4	4	16
4	1	2	32	14,3	2	2	2	2	4	4	16
4	4	1	32	14,3	2	2	2	2	4	4	16
4	4	2	32	14,3	2	2	2	2	4	4	16

(a)

L1	L2	L3	F	%FR	1	2	3	4	5	6	7
1	1	3	64	28,6	6	6	6	6	12	12	16
1	1	4	64	28,6	6	6	6	6	12	12	16
1	4	3	64	28,6	6	6	6	6	12	12	16
1	4	4	64	28,6	6	6	6	6	12	12	16
2	2	1	64	28,6	6	6	6	6	12	12	16
2	2	2	64	28,6	6	6	6	6	12	12	16
2	3	1	64	28,6	6	6	6	6	12	12	16
2	3	2	64	28,6	6	6	6	6	12	12	16
3	2	1	64	28,6	6	6	6	6	12	12	16
3	2	2	64	28,6	6	6	6	6	12	12	16
3	3	1	64	28,6	6	6	6	6	12	12	16
3	3	2	64	28,6	6	6	6	6	12	12	16
4	1	3	64	28,6	6	6	6	6	12	12	16
4	1	4	64	28,6	6	6	6	6	12	12	16
4	4	3	64	28,6	6	6	6	6	12	12	16
4	4	4	64	28,6	6	6	6	6	12	12	16

(b)

L1	L2	L3	F	%FR	1	2	3	4	5	6	7
1	1	1	32	14,3	2	2	2	2	4	4	16
1	1	2	32	14,3	2	2	2	2	4	4	16
1	1	3	64	28,6	6	6	6	6	12	12	16
1	1	4	64	28,6	6	6	6	6	12	12	16
1	2	1	56	25,0	6	6	6	6	12	4	16
1	2	2	56	25,0	6	6	6	6	12	4	16
1	2	3	40	17,9	2	2	2	2	4	12	16
1	2	4	40	17,9	2	2	2	2	4	12	16
1	3	1	56	25,0	6	6	6	6	12	4	16
1	3	2	56	25,0	6	6	6	6	12	4	16
1	3	3	40	17,9	2	2	2	2	4	12	16
1	3	4	40	17,9	2	2	2	2	4	12	16
1	4	1	32	14,3	2	2	2	2	4	4	16
1	4	2	32	14,3	2	2	2	2	4	4	16
1	4	3	64	28,6	6	6	6	6	12	12	16
1	4	4	64	28,6	6	6	6	6	12	12	16

(c)

Table 6.2: Fault breakdown per fault-injection point for the reference logic gate structure; (a) shows all the logic gate variations for the minimum FR; (b) shows all the logic gate variations for the maximum FR; (c) shows the breakdown in regards to fault injection point of the first table of Table 6.1

The next FR analysis of stuck-at high/low faults injected into a logic structure at defined injection points is performed onto the quadded logic circuit without a majority voter circuit evaluating the generated output results. The adaptation of the base circuit structure of this quadded logic circuit adapting the fixed logic structure is displayed in Figure 6.8(b). Each of the individual logic gates of the circuit, shown in Figure 6.8(b) will be generalised in a way that the logic functionality illustrated in this figure is going to be replaced with Lx.y replacements. These Lx.y replacements are going to be used for the logic alteration specified for this simulation in accordance with the table, which is shown in Figure 6.5(b). As determined in Chapter 5 the alteration of the interwoven signals between the different logic gate levels of the quadded logic structure shows that there is no impact on the fault-tolerance of this logic structure. Due to this evaluation it had been found that the alteration of the interwoven signal structures of a quadded logic structure is not required and the reference quadded logic circuit stays the way as shown in Figure 6.8(b). The fault-injection points specified for the quadded logic structure are also represented in Figure 6.8(b) and these points are going to be utilised for this stuck-at fault-injection simulation. The FR results for this fault-injection simulation are illustrated in Table 6.3 in the same way for all possible logic function variations as for the single logic gate reference structure in Table 6.1. In addition to the labels and definition of Table 6.1 the Table 6.3 has more of the following columns. The column *M* shows the number of faults, which are maskable through a voter due to the fact that only one single output value is incorrect. The column *NM* illustrates the number of faults, which are non-maskable with a majority voter due to the fact that the output set contains 50% ones and 50% zeros. Because of this value distribution no majority voter can vote on a majority output value. The total number of output results is calculated for this logic structure with equation 6.2:

$$N_{output\ var} = N_{Fault\ types} \cdot N_{input\ var} \cdot N_{fault\ injection\ points}$$
$$N_{output\ var} = 2 \cdot 4 \cdot 68 = 544$$

For establishing the fault-tolerance capability of the quadded logic structure a comparison between the reference logic gate structure and quadded logic structure will detail this. The overall FR performance of these two logic gate structures will show that the quadded logic structure has an overall much lower FR value for the different logic variations than the reference logic gate structure. The quadded logic structure under the influence of stuck-at faults injected at the fault-injection points defined at Figure 6.8(b) has the following FR range of minimal $FR_{min} = 0.0\%$ value and the maximum $FR_{max} = 9.2\%$. value. For any logic gate alteration having a FR_{min} of zero value indicates that for this logic gate combination definition realised within the fixed logic structure has created a fault-free or completely fault-tolerant logic circuit. Analysing the fault data regarding where these fault-injection points are triggering a fault and a non-maskable fault is located, the data did not show a clear pattern about where these injection points are. The most

common fault location points have been 41 to 48 throughout the data but these are not the majority ones. These injection points have been identified within Table 5.3 to cause non-maskable faults within a quadded logic structure. Figure 6.8(b) illustrates why these fault-injection points are central points of impact due to the fact that a stuck-at fault injected at these locations simulates a faulty output of a logic gate feeding into the output value-generating logic gates. The one stuck-at fault affects two output-generating logic gates at the same time and this causes the fact that two of the four output values can be affected.

A direct comparison between the reference logic gate structure and the quadded logic structure can be done on the basis of calculation of the average FR out of the 64 logic gate alteration cases. The average FR for the reference logic gate structure is 21.43% and for the quadded logic designs it is 3.49%. This comparison of the average FR shows that the quadded logic design has a significant impact on the numbers of faults present at the outputs of this logic circuit before feeding it into the majority voter. The quadded logic structure has a 6.14 times better fault-handling performance than the reference logic gate structure.

L1	L2	L3	M	F	NM	%FR
1	1	1	32	8	40	8,8
1	1	2	32	8	40	8,8
1	1	3	64	0	24	4,4
1	1	4	64	0	24	4,4
1	2	1	16	0	0	0,0
1	2	2	16	0	0	0,0
1	2	3	16	0	0	0,0
1	2	4	16	0	0	0,0
1	3	1	32	8	24	5,9
1	3	2	32	8	24	5,9
1	3	3	32	8	24	5,9
1	3	4	32	8	24	5,9
1	4	1	16	0	0	0,0
1	4	2	16	0	0	0,0
1	4	3	48	0	16	2,9
1	4	4	48	0	16	2,9

L1	L2	L3	M	F	NM	%FR
2	1	1	16	0	0	0,0
2	1	2	16	0	0	0,0
2	1	3	16	0	0	0,0
2	1	4	16	0	0	0,0
2	2	1	64	0	24	4,4
2	2	2	64	0	24	4,4
2	2	3	32	8	40	8,8
2	2	4	32	8	40	8,8
2	3	1	48	0	16	2,9
2	3	2	48	0	16	2,9
2	3	3	16	0	0	0,0
2	3	4	16	0	0	0,0
2	4	1	32	8	24	5,9
2	4	2	32	8	24	5,9
2	4	3	32	8	24	5,9
2	4	4	32	8	24	5,9

L1	L2	L3	M	F	NM	%FR
3	1	1	32	8	24	5,9
3	1	2	32	8	24	5,9
3	1	3	32	8	24	5,9
3	1	4	32	8	24	5,9
3	2	1	48	0	16	2,9
3	2	2	48	0	16	2,9
3	2	3	16	0	0	0,0
3	2	4	16	0	0	0,0
3	3	1	64	0	24	4,4
3	3	2	64	0	24	4,4
3	3	3	32	8	40	8,8
3	3	4	32	8	40	8,8
3	4	1	16	0	0	0,0
3	4	2	16	0	0	0,0
3	4	3	16	0	0	0,0
3	4	4	16	0	0	0,0

L1	L2	L3	M	F	NM	%FR
4	1	1	16	0	0	0,0
4	1	2	16	0	0	0,0
4	1	3	48	0	16	2,9
4	1	4	48	0	16	2,9
4	2	1	32	8	24	5,9
4	2	2	32	8	24	5,9
4	2	3	32	8	24	5,9
4	2	4	32	8	24	5,9
4	3	1	16	0	0	0,0
4	3	2	16	0	0	0,0
4	3	3	16	0	0	0,0
4	3	4	16	0	0	0,0
4	4	1	32	8	40	8,8
4	4	2	32	8	40	8,8
4	4	3	64	0	24	4,4
4	4	4	64	0	24	4,4

Table 6.3: Results of fault simulation in accordance with logic gate alteration applied onto Figure 5.9 quadded logic gate circuits without voter

The third analysis of the fault-handling capability of a QLC is the last one for this evaluation. For the simulation of the fault-behaviour affected under the influence of injected SAH and SAL faults into the logic circuit the internal structure of logic unit (see Figure 6.4) requires fault-injection points. As indicated within this figure, stuck-at faults are only injected at inputs or outputs of this logic structure. Fault effects, caused by the switches, which are not able to close or stay closed continuously, are excluded from this analysis. The effects of interconnect faults are also not part of

this analysis. The exclusion of these two analysis points is due to the simulation and data analysis work involved, which should be extensive.

The location of these injection points within the logic unit is outlined in Figure 6.9. The fault-injection is only required to be performed on one of the four logic units of the QLC. Applying only the fault-injection points within one logic unit is because of the temporal-dependent reconfiguration with a round-robin utilisation of each logic unit within the QLC. Due to the round-robin approach of logic unit utilisation the one faulty logic unit will be used within every possible arrangement within the pre-defined logic gate structure. Because of this utilisation, injection faults in each logic unit do not get other results as with those only using one logic unit. The main difference is going to be that a set of four identical results has been created without gaining more fault-behaviour information. The set-up for this fault-injection for creating the simulation data of the FR table, the logic unit C of a QLC is the one where all the stuck-at faults are being injected. The selection of the fault-injection points within the fixed logic structure has been defined in accordance with Figure 6.9. The creation of the FR has been done in the same way as for the two other fault-injection simulations by utilisation of all possible logic gate variations and the resulting FR is displayed in Table 6.4. Within this table the same labels are being used with the same definition as Table 6.3. The total number of output results of this logic structure is calculated with equation 6.2:

$$N_{output\ var} = N_{Fault\ types} \cdot N_{input\ var} \cdot N_{fault\ injection\ points}$$
$$N_{output\ var} = 2 \cdot 16 \cdot 15 = 480$$

The QLC structure has the following range of FR with minimal $FR_{min} = 0.8\%$ value and the maximum $FR_{max} = 9.2\%$ value. When breaking down the fault data into the fault-causing injection points it does not show a clear pattern about where these injection points are. The most common fault-injection points within the data are the points 1, 2 and 15 due to their central functionality for the input and output of the logic unit. The average FR for the QLC structure is 3.97% over the 64 diverse simulation cases. In comparison to the average FR of the reference logic gate structure with 21.43% and of the quadded logic structure at 3.49%, the QLC average FR is only 0.48% higher than the value of the quadded logic design.

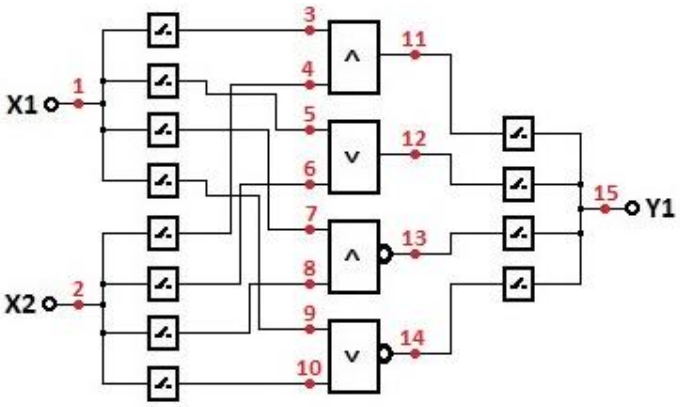


Figure 6.9: Fault-injection points at the logic structure of a logic unit excluding the switches and interconnection between the logic gates

L1	L2	L3	M	F	NM	%FR
1	1	1	30	6	16	4,6
1	1	2	62	2	6	1,7
1	1	3	81	11	31	8,8
1	1	4	102	2	34	7,5
1	2	1	39	3	40	9,0
1	2	2	104	0	12	2,5
1	2	3	93	3	13	3,3
1	2	4	116	0	6	1,3
1	3	1	35	5	39	9,2
1	3	2	100	2	11	2,7
1	3	3	35	5	39	9,2
1	3	4	100	2	11	2,7
1	4	1	45	1	16	3,5
1	4	2	72	0	4	0,8
1	4	3	119	9	15	5,0
1	4	4	140	0	18	3,8

L1	L2	L3	M	F	NM	%FR
2	1	1	39	3	40	9,0
2	1	2	104	0	12	2,5
2	1	3	93	3	13	3,3
2	1	4	116	0	6	1,3
2	2	1	111	9	19	5,8
2	2	2	132	0	22	4,6
2	2	3	55	1	11	2,5
2	2	4	68	0	6	1,3
2	3	1	119	9	15	5,0
2	3	2	140	0	18	3,8
2	3	3	45	1	16	3,5
2	3	4	72	0	4	0,8
2	4	1	95	3	12	3,1
2	4	2	118	0	5	1,0
2	4	3	95	3	12	3,1
2	4	4	118	0	5	1,0

L1	L2	L3	M	F	NM	%FR
3	1	1	35	5	39	9,2
3	1	2	100	2	11	2,7
3	1	3	35	5	39	9,2
3	1	4	100	2	11	2,7
3	2	1	119	9	15	5,0
3	2	2	140	0	18	3,8
3	2	3	45	1	16	3,5
3	2	4	72	0	4	0,8
3	3	1	81	11	31	8,8
3	3	2	102	2	34	7,5
3	3	3	30	6	16	4,6
3	3	4	62	2	6	1,7
3	4	1	93	3	13	3,3
3	4	2	116	0	6	1,3
3	4	3	39	3	40	9,0
3	4	4	104	0	12	2,5

L1	L2	L3	M	F	NM	%FR
4	1	1	45	1	16	3,5
4	1	2	72	0	4	0,8
4	1	3	119	9	15	5,0
4	1	4	140	0	18	3,8
4	2	1	95	3	12	3,1
4	2	2	118	0	5	1,0
4	2	3	95	3	12	3,1
4	2	4	118	0	5	1,0
4	3	1	93	3	13	3,3
4	3	2	116	0	6	1,3
4	3	3	39	3	40	9,0
4	3	4	104	0	12	2,5
4	4	1	55	1	11	2,5
4	4	2	68	0	6	1,3
4	4	3	111	9	19	5,8
4	4	4	132	0	22	4,6

Table 6.4: Results of fault simulation in accordance with logic gate alteration applied onto Figure 6.6 QLC in accordance with injection points indicated in Figure 6.9 without voter

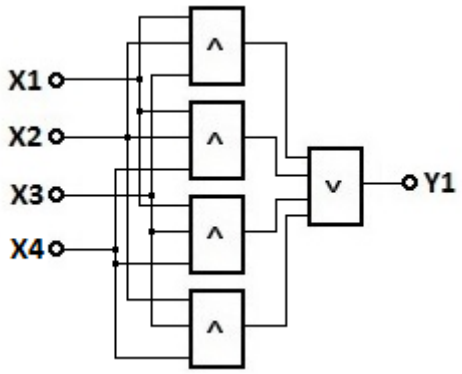
6.4.2. Fault-handling evaluation of quadded logic vs. QLC, both with voter

Digital systems, which use spatial redundancy, [88, 101] are required to reduce the N-output results supplied from the N-time redundant digital systems back down into one overall digital output result of this system. The generation back into one output result is done with the help of majority-voting in almost every case of an N-time redundant digital system. The impact of injected SAH or SAL faults onto the correctness of a majority voter has been evaluated within Chapter 4.6.1 and is not relevant for this evaluation of the fault-handling capability of these two different logic structure

designs of this chapter. In this chapter only the impact of the majority voter on the total fault-handling capability for these two designs is going to be evaluated and the central requirement for this evaluation is the fault-tolerance of the majority voter in regard to fault-masking. For this chapter the majority voter is fault-free and works without faults in accordance with the following equation for a majority voter done on four-inputs:

$$Y1 = (X1 \wedge X2 \wedge X3) \vee (X1 \wedge X2 \wedge X4) \vee (X1 \wedge X3 \wedge X4) \vee (X2 \wedge X3 \wedge X4) \qquad \text{(Equation 6.4)}$$

The logic circuit design of this four-input majority voter is outlined in Figure 6.10(a) and this voter is added to the output of each of these two logic structures under evaluation. In Figure 6.10(b) the truth table of the four-input majority voter is defined and this truth table data is used for the fault-masking evaluation for both logic structures. The QLC design requires three memory elements for the first three output values to be stored until the last output value has been generated. Also for this fault-injection simulations regarding stuck-at high of low fault evaluation these memory elements are fault-free and do not create faults by altering information stored inside them in any way.



(a)

X1	X2	X3	X4	Y1
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
1	1	0	0	0
0	0	1	0	0
1	0	1	0	0
0	1	1	0	0
1	1	1	0	1
0	0	0	1	0
1	0	0	1	0
0	1	0	1	0
1	1	0	1	1
0	0	1	1	0
1	0	1	1	1
0	1	1	1	1
1	1	1	1	1

(b)

Figure 6.10: (a) four-input voter circuit; (b) truth table of the four-input majority voter

The evaluation of the fault-handling capability of the combined circuit of quadded logic and majority voter under the influence of stuck-at high or low faults injected at the relevant injection points specified at Figure 6.8(b), is delineated in Table 6.5. This table represents only the total number of faults per logic gate alteration due to the fact that the output result of the voter is the majority-voted result. No other output values are being generated by this logic circuit. A

comparison of this majority-voted output result against a known good output value determines if this output value is correct or faulty. If the comparison determines that the output value is faulty it will be counted as a fault. This quadded logic structure with majority voter has the FR range of minimal $FR_{min} = 0.0\%$ value and the maximum $FR_{max} = 8.8\%$ value. The average FR over the 64 resulting FR cases is 2.02% for the quadded logic structure. Compared against the other average FR found so far, shows that this is the lowest average FR so far and indicates the significance of the impact of a majority voter on the overall system FR. A quadded logic structure without a majority voter has in accordance with Table 6.3 an average FR of 3.49% and by adding a majority voter to the same logic structure it reduces the average FR by 1.47% to 2.02%. The cases of fault-free logic gate variance for a majority voter less quadded logic system are 24 cases or 37.5% of all cases outlined in Table 6.3. This ratio of fault-free cases improves by adding a majority voter and increases in this way to 32 cases, which are now been seen as fault-free indicated. This rise in fault-free cases represents an increase of 12.5% to be 50% of all cases.

L1	L2	L3	F	%FR
1	1	1	48	8,8
1	1	2	8	1,5
1	1	3	0	0,0
1	1	4	24	4,4
1	2	1	0	0,0
1	2	2	0	0,0
1	2	3	0	0,0
1	2	4	0	0,0
1	3	1	32	5,9
1	3	2	8	1,5
1	3	3	8	1,5
1	3	4	32	5,9
1	4	1	0	0,0
1	4	2	0	0,0
1	4	3	0	0,0
1	4	4	16	2,9

L1	L2	L3	F	%FR
2	1	1	0	0,0
2	1	2	0	0,0
2	1	3	0	0,0
2	1	4	0	0,0
2	2	1	24	4,4
2	2	2	0	0,0
2	2	3	8	1,5
2	2	4	48	8,8
2	3	1	16	2,9
2	3	2	0	0,0
2	3	3	0	0,0
2	3	4	0	0,0
2	4	1	32	5,9
2	4	2	8	1,5
2	4	3	8	1,5
2	4	4	32	5,9

L1	L2	L3	F	%FR
3	1	1	32	5,9
3	1	2	8	1,5
3	1	3	8	1,5
3	1	4	32	5,9
3	2	1	16	2,9
3	2	2	0	0,0
3	2	3	0	0,0
3	2	4	0	0,0
3	3	1	24	4,4
3	3	2	0	0,0
3	3	3	8	1,5
3	3	4	48	8,8
3	4	1	0	0,0
3	4	2	0	0,0
3	4	3	0	0,0
3	4	4	0	0,0

L1	L2	L3	F	%FR
4	1	1	0	0,0
4	1	2	0	0,0
4	1	3	0	0,0
4	1	4	16	2,9
4	2	1	32	5,9
4	2	2	8	1,5
4	2	3	8	1,5
4	2	4	32	5,9
4	3	1	0	0,0
4	3	2	0	0,0
4	3	3	0	0,0
4	3	4	0	0,0
4	4	1	48	8,8
4	4	2	8	1,5
4	4	3	0	0,0
4	4	4	24	4,4

Table 6.5: Results of fault simulation in accordance with logic gate alteration applied onto Figure 5.9 quadded logic gate circuits with voter

The impact of adding a majority voter to the QLC structure has been evaluated in the same way as the evaluation of the fault-handling capability of the quadded logic structure and is illustrated in Table 6.4. The resulting FR under the influence of stuck-at high or low faults injected at the

appropriate fault-injection points (see Figure 6.9) and the same assumption of only using one logic unit to do so is displayed in Table 6.6. This table shows the faults counted per logic alteration where the deviation of the majority-voting output result against a known good output result exists. This is done in the same way as for the quadded logic structure evaluation. The QLC structure with added majority voter has the range of FR of minimal $FR_{min} = 0.0\%$ value and the maximum $FR_{max} = 6.5\%$ value. The average FR of these 64 cases for this logic system is 2.25%. In comparison to the majority voter less QLC system a reduction of the average FR of 1.72% has been achieved. It can be observed in Table 6.6 that four fault-free logic alteration set-ups are within the whole simulation range and this represents 6.3% of all cases. This result is in contrast to no fault-free cases within the simulation data of the system without majority voter. This is also for the quadded logic structure by adding a majority voter, which is masking faults within the result data and because of this a majority voter is a vital functional block within the fault-handling capability of any system.

L1	L2	L3	F	%FR
1	1	1	6	1,3
1	1	2	4	0,8
1	1	3	23	4,8
1	1	4	30	6,3
1	2	1	18	3,8
1	2	2	3	0,6
1	2	3	14	2,9
1	2	4	5	1,0
1	3	1	18	3,8
1	3	2	6	1,3
1	3	3	31	6,5
1	3	4	9	1,9
1	4	1	6	1,3
1	4	2	1	0,2
1	4	3	18	3,8
1	4	4	15	3,1

L1	L2	L3	F	%FR
2	1	1	18	3,8
2	1	2	3	0,6
2	1	3	14	2,9
2	1	4	5	1,0
2	2	1	22	4,6
2	2	2	0	0,0
2	2	3	7	1,5
2	2	4	6	1,3
2	3	1	15	3,1
2	3	2	3	0,6
2	3	3	12	2,5
2	3	4	3	0,6
2	4	1	5	1,0
2	4	2	0	0,0
2	4	3	13	2,7
2	4	4	5	1,0

L1	L2	L3	F	%FR
3	1	1	18	3,8
3	1	2	6	1,3
3	1	3	31	6,5
3	1	4	9	1,9
3	2	1	15	3,1
3	2	2	3	0,6
3	2	3	12	2,5
3	2	4	3	0,6
3	3	1	30	6,3
3	3	2	8	1,7
3	3	3	22	4,6
3	3	4	6	1,3
3	4	1	5	1,0
3	4	2	1	0,2
3	4	3	28	5,8
3	4	4	9	1,9

L1	L2	L3	F	%FR
4	1	1	6	1,3
4	1	2	1	0,2
4	1	3	18	3,8
4	1	4	15	3,1
4	2	1	5	1,0
4	2	2	0	0,0
4	2	3	13	2,7
4	2	4	5	1,0
4	3	1	5	1,0
4	3	2	1	0,2
4	3	3	28	5,8
4	3	4	9	1,9
4	4	1	6	1,3
4	4	2	0	0,0
4	4	3	15	3,1
4	4	4	22	4,6

Table 6.6: Results of fault simulation in accordance with logic gate alteration applied onto Figure 6.6 QLC in accordance with injection points indicated in Figure 6.9 with voter

6.4.3. Overview of simulation results of the different systems

For comparing the fault-handling capabilities of both logic structures a defined set of results has to be used. The design of the reference logic gate performing the fixed logic structure will be excluded because of the lack of fault-handling capabilities. The comparison of the fault-handling capability of this different multi output result system set-ups will be based on average FR, min/max value of FR and the number of faults deviating from correct output values. Table 6.7 shows the defined selection of results of the fault-injection simulation for quadded and QLC logic gate structure of the fixed logic structure. For a comprehensive analysis of the fault-handling capability the total number of faults deviating from the correct output value is divided into incorrect output values, which are indicated as a fault (F) and non-maskable faulty outputs (NM). Both these values are added to Table 6.7 because of their nature of the behaviour of the system. Faults (F) shown at the central output of each logic structure are passing through into the next logic system without identification or an external checker running side-by-side generating fault-free results. The external checker could be used for checking the correctness of the output value independent and unrestricted. By using an external checker the hardware overhead is going to be increased and the trustworthiness of the checker has to be assured. So the total number of faults (F) caused by stuck-at high/low fault-injection is an important fault-handling indication of a system. Non-maskable (NM) faults can on the other hand, be identified through two possible ways with some additional logic structure. First by checking that the not majority-voted output results of the logic system fulfil the majority voter rule, which is $^{N}/_{2}$ of the number of outputs containing the same value [99]. The second solution is the majority-voted output result feedback for comparison against each not majority-voted logic output result for identifying the number of deviations. In the case of two deviations a non-maskable fault has altered the majority-voted output result. If one of these solutions has been applied onto a logic structure the identification of this non-maskable condition is possible and this can be indicated to prevent the fault transition through the system unnoticed.

	avr %FR	min %FR	max %FR	F	NM	# Faults
quadded w/o voter	3,49	0,0	9,2	192	1024	1216
quadded w voter	2,02	0,0	8,8	192	512	704
QLC w/o voter	3,97	0,8	9,2	166	1054	1220
QLC w voter	2,25	0,0	6,5	166	527	693

Table 6.7: Overview of different results of quadded and QLC logic design including with (w) or without (w/o) majority voter

Based on the average FR for these two logic structures the quadded logic design has the lowest average FR of 2.02% in comparison to the QLC structure with added majority voter. Even that the QLC structure with majority voter has a lower maximum FR for a single logic gate alteration the average FR over all cases is still higher than the quadded logic structure. That the quadded logic structure has a lower average FR over all cases is due to the fact that it has more fault-free logic alteration set-ups as the QLC structure. For both logic structures the number of incorrect output values indicated as a fault (F) remain at the same level regardless of the presence of a majority voter or without one. This is due to the fact that the injected stuck-at high or low fault creates an altered output result affecting all individual output values or more than $^{N}/_{2}$ of them for creating a correct majority-voted output result. The numbers of faults (F) for the different logic structures are illustrated in Table 6.7. The total fault numbers of both logic structures are 192 for the quadded logic structure and 166 for the QLC structure. Putting these numbers of faults (F) of both logic structures into perspective to the total number of stuck-at fault-injection simulation runs, the percentage of the quadded logic structure is 0.55% and 0.54% for the QLC structure. Both output values are of almost similar percentage value and only the absolute value difference between both numbers of fault (F) values reveals which of the logic structures requires an external checker. In this case the quadded logic structure has the higher number of faults (F), which means that a system-checker would be required for this structure.

This value difference is 26 between both total fault (F) values from the two logic structure designs. If the logic structure is going to be equipped with an external checker the logic design for the quadded logic structure has to be at least 15.66% covering more fault cases than the one for a QLC structure. The number of non-maskable (NM) faults for both logic structures is reduced by 50% and this is due to the added majority voter. The majority voter behaviour is defined in Figure 6.10(b) and the fault condition of both logic structures regarding non-maskable faults is defined for an equally distribution of zeros and ones within the direct output results. For the two logic structures an evenly allocation of zeros and ones means that the direct output result contains two zeros and two ones randomly orientated. The majority-voted result for this input sequence is as per the definition of Figure 6.10(b) in all cases zero. This output result of the majority voter, which does not reflect a majority-voted result is more likely a default value. That output value is now used to compare it against the correct output value and in 50% of the non-maskable output results it is the correct value of zero. This condition is not given per design of the logic structure in conjunction with a majority voter. It is more to do with a 50% chance of being correct. If the whole system is required to indicate the presence of this condition existing for a generated directly produced output set, the logic structure has to be expanded with an external majority-voted output feedback comparator for each individual output signal. In the case of the presence of two deviating single output values compared to the majority-voted output value an indication can be generated and then be indicated through the system-checker. The system-checker on the other hand cannot alter the

output set into the correct output set and due to the nature of the output value distribution it is also only in 50% of the cases correct. If the system-checker needs to indicate this fault correctly, the system-checker has to be designed in a way that the system-checker looks for three of the same kind of values present within the logic structure output set. In the case of an equal distribution of ones and zeros the checker indicates this as a fault condition for this particular functional block.

6.5. Summary of the chapter

The questions answered within this chapter are: would it be possible to combine the three redundancy concepts of spatial (hardware), temporal (time) or data (information) within one overall redundancy concept and what kind of impact will this concept cause on the FR compared against quadded logic structure?

The central question about combining all three redundancy concepts into one concept can be answered with the logic structure that fulfils the request to be a QLC structure. The QLC structure is based on a matrix structure divided into four logic units similar to the tile concept shown in Figure 6.1(a) and each unit contains a range of configurable logic circuits similar to the one shown in Figure 6.1(b) for a range of logic functions. This logic structure is using spatial redundancy and the functional logic circuit inside a QLC has a fixed three out of four logic unit's structure which is defined in Figure 6.5(a).

The novelty of the QLC structure has been done by the use of time-triggered round-robin reconfiguration of a fixed functional logic circuit shown in Figure 6.6(b). Through this approach the temporal and data redundancy is fulfilled. By using the data redundancy the N-numbers of independent output results are being generated. Each output result out of this set of output results has been generated where ⅔ of the logic units of the fixed logic circuit overlap for every time-trigger. With this concept of logic units overlap the identification of a logic unit with a permanent fault can be achieved through the alternating utilisation of the logic unit by the fixed logic circuit. The time-triggered round-robin reconfigurable QLC structure is designed for being fault-tolerant. The final fault-tolerance of any N-number-based generating logic structure is achieved by the use of a majority voter. As analysed within Chapter 4 the majority voter is capable of masking faults but as a logic circuit it cannot be considered as fault-tolerant.

The second question was about the comparison between the newly created logic structure vs. quadded logic has been answered in the chapter through an FR analysis. The analysis was performed over a wide range of logic functions injected with faults, by applying each possible variation feasible regarding logic functionality onto the individual logic gates creating the fixed logic structure. A range of 64 different logic set ups under the influence of stuck-at high or low faults had been analysed for generating FR for each logic configuration. The fault-handling capability of QLC vs. quadded logic structure of the FR of each logic configuration has been compared and the results are represented in Table 6.8. The results reveal that the fault-handling capability of the QLC structure is not as good as the quadded logic structure. The FR for quadded logic without (w/o) majority voter is 12.1% and with (w) majority voter 10.2% better than the QLC structure. Comparing the individual FR for each logic configuration shows for the maximum FR that the QLC is 26.1% better than the quadded logic structure. This indicates that the QLC structure fault-handling performance is not comparable with the quadded logic structure, but the absolute

number of faults within a variable logic configuration is smaller and therefore the QLC is the better logic circuit in terms of fault sensitivity.

	avr %FR	min %FR	max %FR
quadded w/o voter	3,49	0,0	9,2
quadded w voter	2,02	0,0	8,8
QLC w/o voter	3,97	0,8	9,2
QLC w voter	2,25	0,0	6,5

Table 6.8: Comparison result of FR analysis for QLC vs. quadded logic under the influence of stuck-at high or low faults injected

The main concept developed and proven within this chapter is centred on the design of the temporal-dependant reconfigurable round-robin matrix or QLC matrix element. Within the QLC matrix element the three redundancy concepts of spatial, temporal and data are combined in a way that a unique fault-localisation and discrimination is inherent within the logic structure. Through the correct combination of the three redundancies within the QLC matrix element fault-masking and correcting has been achieved close to the capabilities of a quadded logic structure, but without the use of interwoven interconnection structure and not by using quadded module redundancy. By design, the quadded logic structure is capable of performing fault-tolerance by means of fault-masking and correcting. The QLC logic structure has an inherent concept of fault-localisation, which is beyond the two fault-tolerant approaches of the quadded logic structure. This fault-localisation concept is accomplished by the utilisation and mixing of temporal-triggered reconfiguration and partial overlapping hardware structure used during each time slot. A detailed description of the QLC logic structure fault-localisation feature is part of Chapter 9. The impact of the majority voter on N-number redundantly generated output results has been analysed. This analysis revealed the significance of the majority voter for the overall fault-tolerance of these type of logic systems and required further research work in fine-grained logic structures. The majority voter needs to be by definition fault-tolerant and equipped with fundamental fault detection, which requires logic gate alteration. Both requirements towards the logic gates are investigated further within Chapter 7. Beyond these requirements a concept of intrinsic triggered self-healing of a given logic function within a fine-grained logic structure has been developed and analysed within the following chapter. The localisation and distinction between logic gate and interconnect faults is another essential requirement towards a fault-tolerant system. A concept of achieving this requirement is going to be analysed and fulfilled within the Chapter 9 of this thesis.

Chapter 7: Design of a fault-tolerant logic gate

7.1. Introduction

This chapter deals with the question stated within Chapter 5 about whether it would be possible to alter the fine-grained transistor structure of a logic gate to be better equipped against stuck-at faults at the transistor level with a minimal hardware overhead and what impact on a given logic circuit can be achieved? Does this altered logic gate design offer a feature which could be utilised for intrinsic built-in feature for initiation of circuit alteration without the influence of external logic circuitry? Both questions are going to be answered within this chapter by the means of the conducted research. The developed redundancy fine-grained transistor structure has been applied onto standard logic circuits to show their usefulness for increasing the fault-tolerance of these gates. The altered logic gates are resilient against SAL faults and for the SAH faults it indicated the influence on the logic gate through an intrinsically built-in indication signal. Self-healing of faulty logic gates can be designed out of these altered logic gates.

Fault-tolerances of electronic systems are achieved through the usage of redundancy applied onto the logic structure at a functional or fine-grained level. Functional level redundancy uses N-number of the same logic circuit design to generate N-number of output results. Out of them a majority voter generates a single majority output result. Fine-grained redundancy is applied onto the transistor structure of the individual logic gates increasing the insensibility against faults affecting the individual transistors of this logic gate. The main difference between both approaches is that the functional approach requires a functional block performing majority output voting versus the fine-grained transistor structure, which is an intrinsic fault-tolerant part of each logic gate.

The aim of fine-grained transistor level redundancy is a combination of fault-masking and fault detection strategy applied to logic gates to achieve immunity to any single stuck-at high or low fault conditions affecting a logic gate transistor. In the case that the logic gate cannot perform fault-masking or detection for a stuck-at fault this logic gate structure contains a built-in feature of clear indication detectable by a higher-level system. Through a number of FR analyses performed on various alterations of the fault-tolerant redundant transistor design, the optimum design was revealed fulfilling this set of requirements. The requirements for this redundant transistor level logic gate are fault-masking and fault detection coverage of this logic gate with minimum transistor redundancy overhead. It will be shown that with this logic gate structure it is possible to achieve both 100% immunity to SAL faults and detection of non-correctable SAH faults. The logic gate's capability of clearly indicating a non-correctable SAH faults within its transistor structure is demonstrated and is set in contrast to simulation results. The combination of fault-masking and detection within a logic circuit is used within self-reconfiguring logic circuit design.

7.2. A fault-tolerant logic gate

The constant increase in transistor density occurs every 18 months within a given chip achieved by the chip manufacturer predicted by Moore's law from 1975 [39, 47], is pushing the feature size of individual transistors into the region of being built out of only a few atoms. This small structure sizes require tighter process control at each level of fabrication and even higher sophisticated production test facilities. Due to the increased levels of defects present within a given chip, better built-in self-test (BIST) functionality is required by meeting the right balance between test coverage and test time. Test time during production of a chip can be seen as a non-value adding feature to the chip and must be kept to a bare minimum.

The current 90 nm chip fabrication technology however involves only 20 to 30 layers of atoms and the resulting gate oxide thickness of a single transistor is reduced down to 5 nm or less [12]. Beyond current technology nodes, uniformity of transistor parameters within a chip cannot be sustained without major fabrication innovations and as a result transistor-level shorts, which account for the most common fault in chip fabrication, must be considered [89, 93]. Besides production test yield enhancement, fluctuation in transistor operation will be affecting in-service chip performance over the life-time and will increase the likelihood of transient and permanent single transistor faults [88].

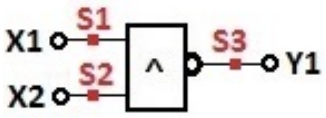
A common method of testing the fault-handling or testing the functionality of a circuit is to force inputs or output pins or access points to a stuck-at high or low level. The total number of responses with regard to stuck-at high or low faults injected is dependent on the accessibility to the logic gate structure. This technique has been used for the fault-handling capability of the quadded logic and QLC structure within Chapter 6 as a tool for generating FR numbers, which can be used to compare fault-tolerance of logic structures. It is also used in production related logic circuit testing to determine if a system is fault-free or not. Logic structures within a chip by itself due to their inaccessibility to individual logic gates are making it hard to perform stuck-at high or low fault-injection to all logic elements. Due to the limited accessibility to all logic functionality, simulation of the logic structure had to be used for evaluation of the fault-handling capability of the entire system.

7.2.1. Comparing of logic gates responses under the influence of fault-injection

The injection of stuck-at high or low faults into a logic gate can be done from the outside at the pins or at the individual transistors forming the logic gate function. The accessibility of the individual transistors of a logic gate is, per design, limited because logic circuit feature sizes on a silicon die are becoming smaller and test/contact pads within the design in most cases do not exist. So to perform stuck-at high or low fault-injection at the individual transistors an alternative to real

transistors can be achieved through the simulation of the circuit in use. The output responses of a logic gate differ under the influence of where the fault is being injected at the individual access points throughout the circuit.

The first fault-injection analysis is performed on the interface pins of a logic gate. This simulation will reveal the fault-behaviour of a logic gate as an entire structure. By injecting a stuck-at fault at the input or output pins of any logic gate, the resulting output value under the influence of a stuck-at fault will be of a clear output of a one or a zero value. Both these output results represent the two functional states of any given logic gate, state one and two in accordance with [36]. Figure 7.1(b) is representing the result of applying stuck-at high or low faults at a NAND logic gate at the indicated fault-injection points shown in Figure 7.1(a). The results found within the fault-injection simulation have been verified with a NAND logic gate on an experimental breadboard and the voltage levels have been checked with a voltage multi-meter.



(a)

X1	X2	No Fault Y1	S1=SAL Y1	S1=SAH Y1	S2=SAL Y1	S2=SAH Y1	S3=SAL Y1	S3=SAH Y1
0	0	1	1	1	1	1	0	1
0	1	1	1	0	1	1	0	1
1	0	1	1	1	1	0	0	1
1	1	0	1	0	1	0	0	1

(b)

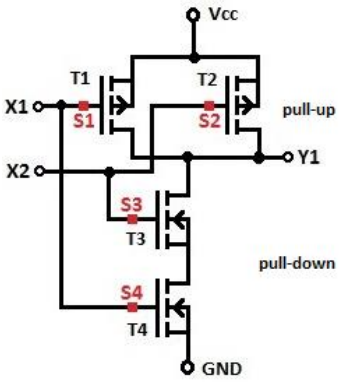
Figure 7.1: Analysing the behaviour of a NAND gate under the influence of stuck-at fault
(a) definition of the fault-injection points at input and output pins;
(b) output results of the NAND gate under the influence of stuck-at faults

The second analysis is performed on the individual transistors of a logic gate to reveal the fault-behaviour of the logic gate on the fine-grained level. This analysis is carried out for analysing the difference between entire versus fine-grained fault-injection behaviour of a logic gate. Applying a single stuck-at high or low fault at an individual transistor of a logic gate is the same as applying a specified voltage level, which is representing the equivalent of a high or low digital levels, at the gate pin of a transistor. By doing so the channel between drain and source of this particular transistor is activated or de-activated. This means that this particular transistor is either turned on or turned off. Within this thesis work the meaning of SAH affecting a transistor is representing an active or switched-on transistor regardless what kind of gate voltage has to be applied to make this

happen. The definition for the SAL, which is used in this thesis, is that it will never turn on regardless of the gate voltage or de-activate it. The simulation results of a spice simulation of injected stuck-at high or low faults at individual transistors of a NAND logic gate built out of four transistors is outlined in Figure 7.2(b). The required fault-injection points of the four NAND logic gate transistors are defined in Figure 7.2(a). For the spice-based simulation of the NAND logic gate generated out of individual transistors the selection of the transistors had been done out of the generic phil_fet library of the spice simulator software. The final design of the fault-tolerant logic gate design has been built out of power MOSFETs on an experimental breadboard and a PCB design to prove the logic gate behaviour. For this hardware analysis the following MOSFETs have been used: on the pull-up network the IRFD9024 MOSFET [121] and for the pull-down network the IRFD020 MOSFET [122]. The conversion of the voltage output levels into digital high or low representation has been based on the voltage vales defining the CMOS digital levels in accordance with [38] and is represented in Table 7.1 accordingly. The spice simulation results of the stuck-at high or low fault-injection simulation at the individual transistors of a NAND gate indicates that the logic gate can get into two more logic state conditions, which are defined within [123] as logic gate state three and state four respectively. This logic gate state behaviour has been verified with the experimental breadboard and with the PCB design (see appendix 6 & 7). On both platforms the logic gate state behaviour, defined with states three and four, could be confirmed.

The definition of the third logic state of a logic gate defining the output of the gate has been isolated from V_{cc} (pull-up network) and GND (pull-down network) (see Figure 7.2(a) for definition of networks). This condition is also referred as tristate condition or isolation of the output through the deactivation of the pull-up and pull-down networks of a logic gate. If at this state the output is floating and it is remaining in the logic level of the previous state [36], this can be seen as memory condition and this type of fault is shown in Figure 7.2(b) for the fault-injection points S1 and S2 under the influence of an SAL injected fault indicated through *mem* within the table. The logic gate state three can also be seen as a high-impedance condition of the logic gates output.

The definition of the fourth logic state is that at the same time the pull-up and pull-down network is conductive and a short circuit between the V_{cc} and GND rail has been created. Through this path an increase of the I_{ddq} current will show the presences of the state at which the logic gate is currently in. The I_{ddq} current within CMOS represents the supply current (I_{dd}) in the quiescent state after all the transistor switching and stable inputs. This type of logic gate behaviour under the influence of a fault is shown in Figure 7.2(b) for fault-injection points S3 and S4 under the influence of an injected SAH fault. This short-circuit path between the V_{cc} and GND rail creates an increase of the I_{ddq} current for the duration of the presence of this fourth logic gate state. Because of the direct connection of V_{cc} and GND rail a significant current flow is occurring within the transistor structure of the logic gate and due to this value of the current level it can be harmful to the circuit structure.



(a)

			S1 SAH_T1 Y1	S2 SAH_T2 Y1	S3 SAH_T3 Y1	S4 SAH_T4 Y1	S1 SAL_T1 Y1	S2 SAL_T2 Y1	S3 SAL_T3 Y1	S4 SAL_T4 Y1
X1	X2	Y1								
0	0	1	1	1	1	1	1	1	1	1
1	0	1	1	1	0	1	1	mem	1	1
0	1	1	1	1	1	0	mem	1	1	1
1	1	0	0	0	0	0	0	0	1	1

(b)

Figure 7.2: Simulation results of Spice simulation of NAND gate with stuck-at fault-injection at individual transistors; (a) definition of the fault-injection points at each transistor; (b) output results of the NAND gate under the influence of stuck-at faults

Level definition CMOS 5V			
Input		Output	
Low	High	Low	High
<= 1.5V	>= 3.5V	<=0.5V	>= 4.4V

Table 7.1: CMOS definition of input and output voltage levels representing high and low digital conditions [38]

Evaluating the fault-behaviour regarding fault-injection of these two different NAND logic gate set-ups has revealed that a direct correlation between both performances is not possible, because it must be known where the fault has been injected and how it affects the logical evaluation of the input stimulus. A fault at the interface structure of a logic gate affects the input stimulus going into the logic gate. By altering the input sequences into an altered version the generated output value will follow this change. Direct fault-injection at the individual transistor of a logic gate puts the logic gate into another altered logic gate state than the two valued ones. These two logic states are three and four for certain types of stuck-at fault-injection in combination with a defined input sequence. The method of fault-injection at the individual logic gate transistors indicates the

required analysis approach needed for this thesis research work to optimise a logic gate structure in terms of fault-tolerance. The additional logic gate state could be utilised in terms of designing a logic gate with only three of the four general inherent logic gate states. The correct selection out of these two logic gate states will equip the newly designed logic gate beyond fault-tolerance. Especially the fourth logic gate state where both gate networks are active at the same time and in conjunction with a certain input stimulus it could be a possible arrangement for equipping this altered logic gate design to indicate non-maskable faults affecting its performance.

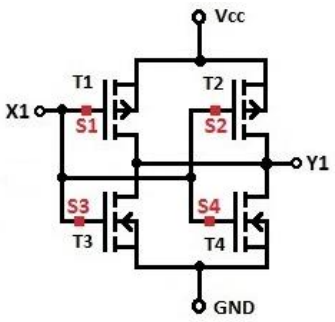
7.2.2. Identifying the functionality of a fault-tolerant logic gate

The designing of a fault-tolerant logic gate, which is able to tolerate a single stuck-at high or low fault affecting one of the logic gate transistors will need a structure, in which a certain number of redundancies through transistors can mask or correct each type of these faults within the logic gate transistor structure. In the event that this transistor structure cannot mask or correct a specific type of stuck-at fault, an intrinsic signalling capability needs to indicate this fault condition to the outside of this logic gate structure. The design specifications for the fault-tolerant logic gate are the following ones:

- The total number of redundant transistors used to fulfil the fault-tolerant requirement has to be minimised under the consideration of maintaining a symmetrical geometrical transistor arrangement within the pull-up and pull-down network.
- If masking or correcting of a stuck-at fault is not possible for this logic gate structure it has a built-in clear indication signal for indicating the occurrence of this fault condition. This indication signal can be used by any higher controlling structure governing the logic gate structure for initiation of self-healing effects embedded into the circuit structure.

In addition to the above points the fault-tolerant design of the logic gate has to achieve the generic requirements of not generating incorrect output results and the appearance of logic state three to manifest at the logic gate output. A fault-tolerant generic structure, which can be applied onto any type of logic gate transistor structure was proposed in [22] in which every single logic gate transistor was replaced with a matrix style 2x2 or generically defining N^2 transistor structure. In this proposal N defines the matrix dimensions and according to the paper also defines with $N-1$ the number of faults this logic gate structure is capable of masking or correcting as a minimum fault number. This fault-tolerant logic gate has a 300% hardware overhead compared to a standard logic gate, which is based on the transistor count of the logic gate.
The optimisation of a logic gate structure has been done in a way that all incorrect output values, state three and fourth state faults are masked or corrected, with the help of an overhead of

redundant transistors within this logic gate structure. A fine-grained SAL fault-tolerant inverter has been proposed in paper [20] and the circuit structure is displayed in Figure 7.3(a). Within Figure 7.3(a) the fault-injection points at the transistor gates of the individual transistors of the SAL fault-tolerant inverter have been defined. The stuck-at high or low fault-injection analysis reveals that the fault-tolerant inverter is capable of handling a single SAL fault without a noticeable alteration of the logic gate output value deviating from the correct output value. Through the injection of SAH at the two points S3 and S4 defined in Figure 7.3(a) two effects on the fault-tolerant inverter can be absorbed, which is an incorrect output value and the state four condition of this logic gate. By applying SAH faults at injection points S1 and S2 of the same inverter shown in Figure 7.3(a) the correct output values are generated in conjunction with logic state four. Each SAH fault causes the presences of the fourth logic state within the SAL fault-tolerant inverter gate. Per design this gate is tailored to handle a certain type of single stuck-at fault and in this case SAL faults. The data illustrated in Figure 7.3(b) also highlights the capability of this gate of indicating the existence of an SAH fault within the individual transistor structure, which cannot be masked. During the influence of the SAH fault the logic gate switches into logic gate state four and the short circuit between V_{cc} and GND rail causes an increase of the I_{ddq} current. In this case this design concept of the fault-tolerant inverter resilient against SAL faults matches the requirements of designing a fault-tolerant logic gate focused on in this chapter.



(a)

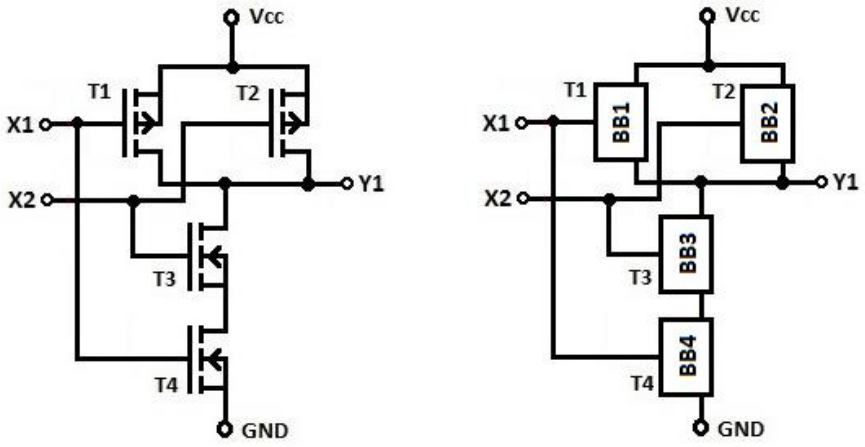
		S1 SAH_T1 Y1	S2 SAH_T2 Y1	S3 SAH_T3 Y1	S4 SAH_T4 Y1	S1 SAL_T1 Y1	S2 SAL_T2 Y1	S3 SAL_T3 Y1	S4 SAL_T4 Y1
X1	Y1								
0	1	1	1	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0

(b)

Figure 7.3: SAL fault-tolerant inverter proposed in [20]; (a) circuit structure of SAL fault-tolerant inverter with injection points; (b) output results of the INV gate under the influence of stuck-at faults

7.2.3. Design of a fault-tolerant NAND logic gate

As specified in the previous chapter the fault-tolerant NAND logic gate needs to fulfil the two requirements specified with the use of minimum redundant transistor count and clear indication of non-maskable faults. The maximum solution for the design of this fault-tolerant logic gate has been outlined in [22] as a quadded transistor-type logic gate (see Figure 5.4) and in [20] the design of a SAL fault-tolerant inverter (see Figure 7.3(a)). For finding the optimised fault-tolerant NAND gate the general structure of a NAND gate is displayed in Figure 7.4(a) and is transferred into a building block (BB) structure, which is shown in Figure 7.4(b). The NAND gate altered into a generic structure is needed for the analysis of finding the optimised logic gate design by systematically altering the content within the BB blocks.



(a) (b)

Figure 7.4: (a) Standard NAND gate structure; (b) NAND gate with replaced transistor with building blocks (BB)

Within the BBs of the NAND gate structure outlined in Figure 7.4(b) a defined variety of transistor structures is going to be inserted. These different transistor structures are being created up to a certain number of transistors and all possible circuit variations. For each of these created NAND gates, a stuck-at high and low faults injection simulation at all the possible fault-injection points specified to be at the transistor gate has been performed for the evaluation of the fault-tolerance of this structure. The transistor structures are made by increasing the number of transistors until the quadded logic design structure has been reached, which represents a total number of four transistors within one BB. Within Figure 5.4 the two possible general BBs for a quadded logic gate are displayed and in Figure 7.7 one of these general BBs is used to show the structure of a quadded transistor NAND logic gate. For this analysis each possible circuit variation will be generated to be put into the BB of the NAND gate structure displayed within Figure 7.4(b). For each increased

transistor to the BB a set of transistor structural circuit combinations has been generated and this is illustrated in Figure 7.5. Up to the maximum number of transistors each logic circuit structure variations have been generated and the total number of variations is 21. In Figure 7.5 with C1 until C21 an overview of all these different transistor structures is outlined and it also defines which of them are being used in this analysis for finding the logic structure fulfilling the relevant requirements. All the different transistor combinations have been done with p-channel MOSFET types for uniformity required for the creation of pull-up networks logic gate path. The selection of the p-channel MOSFET has been done at this analysis point to illustrate the only combinational variety. The required complementary n-channel MOSFET configurations are used for the simulation of the logic gate functionality for the fault-injection simulation. All these different structures shown in Figure 7.5 are going to be applied into BB1 and BB2 and the complementary configuration with n-channel transistor into BB3 and BB4 of Figure 7.4(b). The configuration transistor variations shown in Figure 7.5 are going to be applied onto the structure shown in Figure 7.4(b) in a way that both BBs of the pull-up network are containing one configuration setting until all the different configuration variations have been interchanged in the two BBs of the pull-down network. For each transistor structure configuration within the different BBs a complete stuck-at high and low fault-injection simulation at each transistor has been carried out to evaluate the fault-handling capability of this particular NAND gate design. These simulations have been analysed within MATLAB in a way that the both networks of the NAND logic gate are being described by logic equations individually. The advantage of breaking the NAND logic gate function into two descriptive logic equations means that each variable of these equations represents an individual transistor. In this way the simulation of individual fault-injection into transistors can be analysed. The injection of stuck-at faults into these logic equations has been done by direct altering of the input data at the necessary data location accordingly for simulating a fault. A SAH will result in a high value when being altered and the contrasting value for an SAL. For example the pull-up and pull-down logic equations for a standard NAND logic gate which is shown in 7.3(a) are the following:

$$Y_{pull-up} = \overline{X1} + \overline{X2} \qquad \text{(Equation 7.1)}$$

$$Y_{pull-down} = X1 \cdot X2 \qquad \text{(Equation 7.2)}$$

$$Y1 = \begin{cases} 0 : X1 \cdot X2 \\ 1 : \overline{X1} + \overline{X2} \end{cases} \qquad \text{(Equation 7.3)}$$

By the use of splitting the logic gate function into $Y_{pull-up}$ and $Y_{pull-down}$ and injecting SAH or SAL faults at each transistor of the created logic gate structure equivalent in the logic gate equations, the two logic states three and four can be identified and recorded. The optimum for the

fault-tolerant NAND gate is reached when the number of logic state three-causing faults is zero and the state four causing faults are the only ones causing faulty output behaviour of this altered logic gate by means of added transistor redundancy.

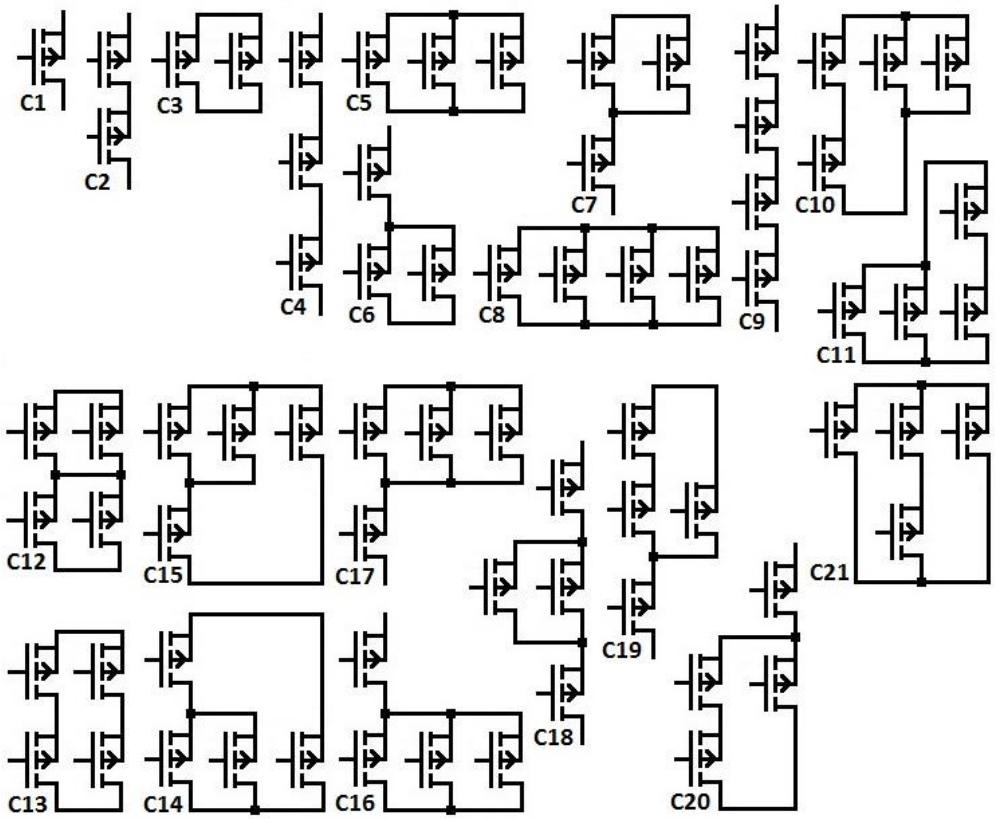


Figure 7.5: All variations of transistor redundancy structures done for incremental increase of transistors performed up to quadded transistor structure

The fault-handling evaluation of the different NAND gate structures created out of the different transistor arrangements C1 to C21 displayed in Figure 7.5 has been based on fault count analysis. Each NAND gate set-up had been exposed to single SAH or SAL at a time applied onto each individual transistor of this gate construction. After each simulation the corresponding fault count for this gate set-up has been established and the results are displayed in Table 7.2 and Table 7.3 accordingly. Within Table 7.2 the number of faults ($F_{\#SAH}$) caused under the influence of a single SAH at each individual transistor is displayed and Table 7.3 displays the numbers of faults ($F_{\#SAL}$) for single SAL influence on each individual transistor of the NAND gate set-up. As a reference point for both tables the combination pull-up network C12 and C13 with corresponding pull-down network represents the two generic quadded transistor-style NAND gates with are fault-tolerant for single SAH and SAL fault-injection. Both Tables show for these four design cases of the NAND gates each zero total numbers of faults, which makes these designs fault-tolerant by design.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21
C1	4	2	4	2	4	2	2	4	2	2	4	2	2	2	2	2	2	2	2	2	4
C2	2	0	4	0	4	0	0	4	0	0	4	0	0	0	0	0	0	0	0	0	2
C3	4	4	8	4	8	4	4	8	4	4	8	4	4	4	4	4	4	4	4	4	6
C4	2	0	4	0	6	0	0	6	0	2	4	0	0	0	0	0	0	0	0	0	2
C5	4	4	8	6	12	6	6	12	6	8	10	6	6	6	6	6	6	6	6	6	8
C6	2	0	4	0	6	0	0	6	0	2	4	0	0	0	0	0	0	0	0	0	2
C7	2	0	4	0	6	0	0	6	0	2	4	0	0	0	0	0	0	0	0	0	2
C8	4	4	8	6	12	6	6	16	8	12	12	8	8	10	10	8	8	8	8	8	12
C9	2	0	4	0	6	0	0	8	0	4	4	0	0	2	2	0	0	0	0	0	4
C10	2	0	4	2	8	2	2	2	4	8	8	4	4	6	6	4	4	4	4	4	8
C11	4	4	8	4	10	4	4	12	4	8	8	4	4	6	6	4	4	4	4	4	8
C12	2	0	4	0	6	0	0	8	0	4	4	0	0	2	2	0	0	0	0	0	4
C13	2	0	4	0	6	0	0	8	0	4	4	0	0	2	2	0	0	0	0	0	4
C14	2	0	4	0	6	0	0	10	2	6	6	2	2	4	4	2	2	2	2	2	6
C15	2	0	4	0	6	0	0	10	2	6	6	2	2	4	4	2	2	2	2	2	6
C16	2	0	4	0	6	0	0	8	0	4	4	0	0	2	2	0	0	0	0	0	4
C17	2	0	4	0	6	0	0	8	0	4	4	0	0	2	2	0	0	0	0	0	4
C18	2	0	4	0	6	0	0	8	0	4	4	0	0	2	2	0	0	0	0	0	4
C19	2	0	4	0	6	0	0	8	0	4	4	0	0	2	2	0	0	0	0	0	4
C20	2	0	4	0	6	0	0	8	0	4	4	0	0	2	2	0	0	0	0	0	4
C21	4	2	6	2	8	2	2	12	4	8	8	4	4	6	6	4	4	4	4	4	8

Table 7.2: Simulation results of the fault count ($F_{\#SAH}$) per NAND gate configuration under the influence of a single SAH at each individual transistor of the gate set-up

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21
C1	4	4	2	4	2	4	2	2	4	2	2	2	2	2	2	4	2	4	2	4	2
C2	4	8	4	8	8	6	4	4	8	4	4	4	4	4	4	6	6	6	4	6	4
C3	2	4	0	4	0	2	0	0	4	0	0	0	0	0	0	2	2	2	0	2	0
C4	4	8	4	12	6	8	8	6	12	6	6	6	6	6	6	8	8	10	8	8	6
C5	2	4	0	6	0	2	2	0	6	0	0	0	0	0	0	2	2	4	2	2	0
C6	4	6	2	8	2	4	4	2	8	2	2	2	2	2	2	4	4	6	4	4	2
C7	2	4	0	8	2	4	4	2	8	2	2	2	2	2	2	4	4	6	4	4	2
C8	2	4	0	6	0	2	2	0	8	0	0	0	0	0	0	2	2	4	2	2	0
C9	4	8	4	12	6	8	8	8	16	8	8	8	8	8	8	10	10	12	10	10	8
C10	2	4	0	6	0	2	2	0	8	0	0	0	0	0	0	2	2	4	2	2	0
C11	2	4	0	6	0	2	2	0	8	0	0	0	0	0	0	2	2	4	2	2	0
C12	2	4	0	6	0	2	2	0	8	0	0	0	0	0	0	2	2	4	2	2	0
C13	2	4	0	6	0	2	2	0	8	0	0	0	0	0	0	2	2	4	2	2	0
C14	2	4	0	6	0	2	2	0	8	0	0	0	0	0	0	2	2	4	2	2	0
C15	2	4	0	6	0	2	2	0	8	0	0	0	0	0	0	2	2	4	2	2	0
C16	4	6	2	8	2	4	4	2	10	2	2	2	2	2	2	4	4	6	4	4	2
C17	2	6	2	8	2	4	4	2	10	2	2	2	2	2	2	4	4	6	4	4	2
C18	4	6	2	10	4	6	6	4	12	4	4	4	4	4	4	6	6	8	6	6	4
C19	2	4	0	8	2	4	4	2	10	2	2	2	2	2	2	4	4	6	4	4	2
C20	4	6	2	8	2	4	4	2	10	2	2	2	2	2	2	4	4	6	4	4	2
C21	2	4	0	6	0	2	2	0	8	0	0	0	0	0	0	2	2	4	2	2	0

Table 7.3: Simulation results of the fault count ($F_{\#SAL}$) per NAND gate configuration under the influence of a single SAL at each individual transistor of the gate set-up

The optimal fault-tolerant NAND gate structure had been found by the means of the analysis of Table 7.3, which shows the number of faults caused by SAL fault-injection into each NAND logic gate variation. By finding the NAND logic gate in which SAL faults have no noticeable effect on the output value it fulfils, is the requirement set for the optimal fault-tolerant NAND gate. Within the data of the table each zero result of a NAND gate design represents this logic gate arrangement. Evaluating Table 7.3 to find the first zero entry from the upper left-hand corner of this table, it will indicate the NAND gate design accomplishing the minimum transistor count. The first zero entry within Table 7.3 happens at the configuration arrangement of pull-up network configuration C3 and pull-down network configuration C3. The resulting NAND gate design is illustrated in Figure 7.6 and has a 100% hardware overhead compared against a standard NAND gate design shown in Figure 7.4(a). The data of the table proves that this logic gate design is fault-tolerant against single SAL faults like the SAL fault-tolerant inverter of Figure 7.3(a). The internal design of the transistor structure of the SAL fault-tolerant inverter also shows parallel redundancy of the individual transistors.

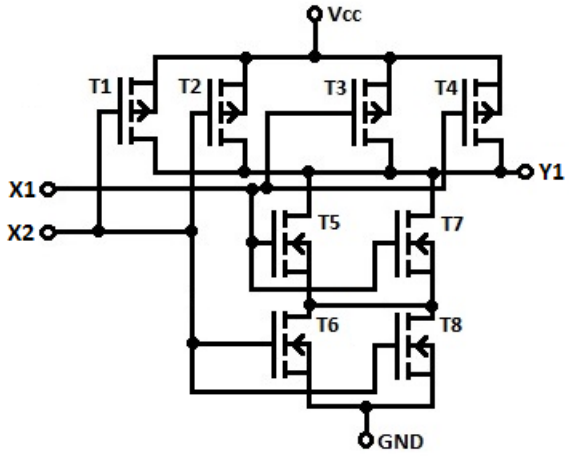


Figure 7.6: Single stuck-at fault resilient (SAFR) NAND gate design as a result of the single SAL fault-injection simulation data is displayed in Table 7.3

In Table 7.2 the total numbers of faults caused by a single SAH-injected fault are being entered into the table in relationship to the configuration variation of the pull-up and pull-down network. For the optimal fault-tolerant NAND gate with the configuration C3 & C3 identified out of Table 7.3 data is used for finding the SAH corresponding total number of faults for the same configuration. The total number of faults within the optimised NAND gate is eight SAH-related faults. These faults are caused through single SAH faults and the effect on the functionality of the NAND gate means that for certain single input stimulus the logic state four occurs for each of these faults. Each of these faults within the fault-tolerant optimised NAND gate creates a short-circuit connection between V_{cc} and GND rail through the pull-up and pull-down network transistors. This direct path

created through the transistors is the lowest possible direct connection imaginable within this logic gate circuit. Because of this the I_{ddq} current increases significantly and is wanted per design specification as a clear indication signal intrinsically built-in into the optimised NAND gate shown in Figure 7.6 as indication of a non-maskable SAH fault affecting the logic gate. The logic gate changed into logic state four, which also changes the output behaviour from digital into analogue signal behaviour and this means that the output can be any voltage between V_{cc} and GND. The voltage level depends on the internal resistor value of the transistors within the pull-up and pull-down network. The output voltage level has to be translated with the help of Table 7.1 to get a digital indication if required. These results have been validated through transferring the gate design onto a PCB design (see appendix 8) including fault-injection points. The resulting behaviour matched the behaviour found through the simulation and the results have been published in the following conference paper [124].

In Table 7.4 for each individual transistor of the optimised NAND gate the corresponding input condition (IC) is listed, which produces the fourth logic state condition within the logic gate under the influence of a single SAH fault-injection at one transistor. For the transistors T1 to T4 of the optimised NAND gate the same IC applies to put the gate into the fourth logic state condition. This means for the logic gate that the injected SAH fault happens within the pull-up network and this alters the output status into constantly connecting Vcc to the output of the logic gate. For creating the current path the pull-down network has to become conductive through IC4. This pattern also generates logic zero condition at the output of the logic gate which is an incorrect value. That happens when the analogue output voltage of 0.88V is translated with the help of Table 7.1 into digital values.

Transistor	T1	T2	T3	T4	T5	T6	T7	T8
IC	*IC4*	*IC4*	*IC4*	*IC4*	*IC2*	*IC2*	*IC3*	*IC3*

IC = input condition *IC1 = 00* *IC2 = 01* *IC3 = 10* *IC4 = 11*

Table 7.4: Results of SAH fault-injection at each individual transistor of the two input NAND gate and the corresponding IC where the fourth logic state occurs

7.2.4. Validation of the optimised fault-tolerant NAND logic gate

The optimised NAND gate design found is displayed in Figure 7.6 and needs to be validated for consistency of the fault numbers specified within Table 7.2 and Table 7.3. This approach is the first verification of the correctness of the logic gate design found. The second approach was done through building the optimised NAND gate design on a breadboard and measuring the circuit behaviour under the influence of injected stuck-at faults. A cross check on this type of fault-tolerant

NAND logic gate design under the influence of single SAH or SAL injected faults needs to confirm the recorded faults numbers, which are displayed in Table 7.2 and Table 7.3. The simulation validation has to be performed in a way that the test is starting with a standard NAND gate where each of the individual transistors are affected by injection of a single SAH and SAL fault into each transistor of the gate. For the evaluation the pull-up and pull-down network transistor structure has been modelled with individual logic equation for each network. The equations are of the same structure as equations 7.1 and 7.2. These logic equations are being evaluated within MATLAB and both results for pull-up and pull-down network produce the selection of which network will produce the output state. Through the evaluation of the logic equation for pull-up and pull-down network the relevant logic states can be identified. In case both logic equations are producing logic high conditions for each equation, which will put the logic gate into logic state four. For the condition that both equations are generating low condition, this will result in logic state three. After each simulation run, a single transistor has been added as a redundant transistor to the standard NAND gate transistor. The flow of adding one redundant transistor starts with the top left transistor and continues in a clockwise direction through the individual standard NAND gate transistors. This approach of adding one transistor as a redundancy continues until the full quadded transistor NAND gate design has been formed. The whole sequence of adding single transistors to the standard NAND is illustrated in Figure 7.7 and the incrementally added transistors are labelled NAND+1 to NAND+12. The label NAND+1 means, for example, that this is the first added transistor to the standard NAND gate transistors.

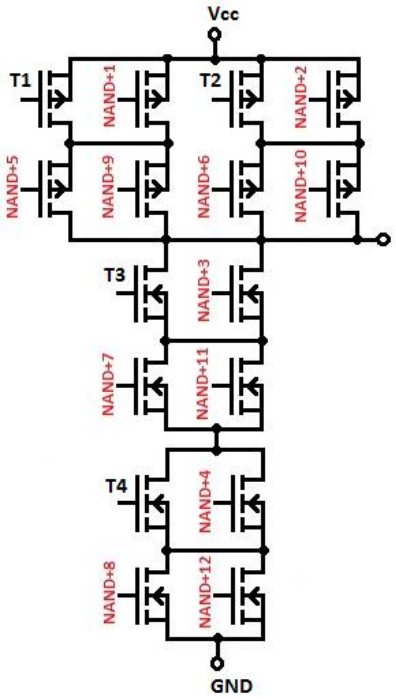
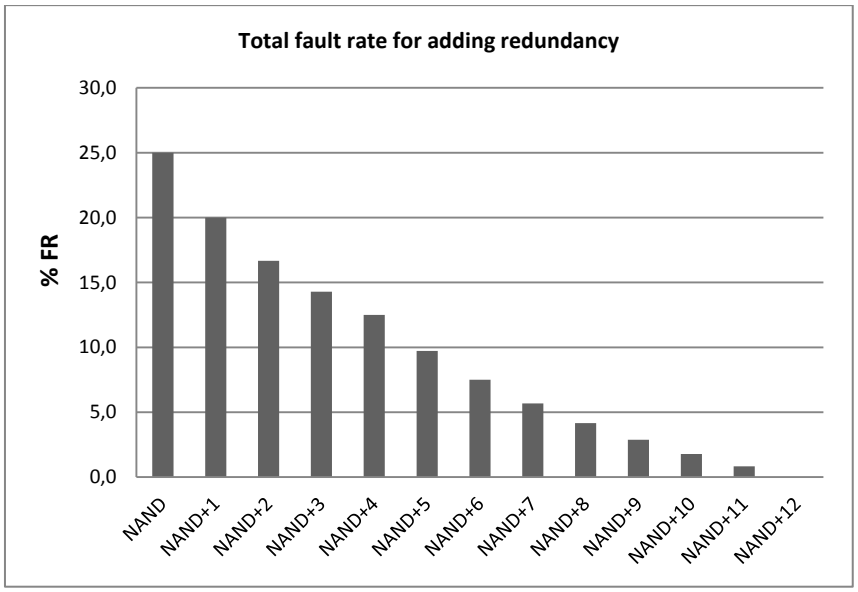


Figure 7.7: NAND gate with increased redundancy by NAND+1 until NAND+12

The evaluation of the fault-handling capability has been done in the same way as in chapter 7.2.3 by simulation of the pull-up and pull-down network in logic equations within MATLAB and the results of these two logic equations have been compared against known values. In the case of a deviation against the correct output value, the FR per incremental added transistor to the logic gate design can be established. In Figure 7.8(a) the FR per incremental added redundant transistor to the standard NAND gate in accordance with Figure 7.7 is plotted. The bar plot of the FR shows continued decline of the FR due to the added redundant transistor. Because of the added redundant transistor the fault-handling capability of the NAND gate increases with each added redundant transistor. As per [22] stated the quadded transistor structure defined by N^2 is capable of handling $N - 1$ faults. The value N for the quadded logic transistor structure for the finding of a minimum fault-handling capability is $N = 2$. This can be observed through the plot shown in Figure 7.8(a) by the downward trend until zero FR for quadded transistor-style structure indicated by NAND+12. In this case the transistor count of the logic gate is sixteen. This makes it 300% hardware overhead for creating a logic gate which is fault resilient for a single stuck-at fault.

In Figure 7.8(b) the total number of faults plotted against the incremental added redundant transistor to the standard NAND gate reflects that at a certain level of added redundant transistors the number of faults declines down to zero. Within this plot of Figure 7.8(b) the total number of faults illustrated per individual result bar has been separated into logic states three and four contributing faults. This has been done to determine the NAND gate configuration where the logic state three indicates zero faults and only logic state four is affecting the fault-behaviour of this NAND gate design. This effect can be observed for the NAND gate design at graph point NAND+4. At this point the design indicates the same number of redundant transistors with the same internal circuit structure as found in chapter 7.2.3 for the optimal NAND gate. At this point the total number of faults affecting the logic gate is eight, which is the same value as defined with Table 7.2 and Table 7.3 for the similar logic gate design. Through this the simulation-based verification of the optimum NAND gate design shows that the resulting design defined with Figure 7.6 represents the optimum design fulfilling the requirements.

(a)



(b)

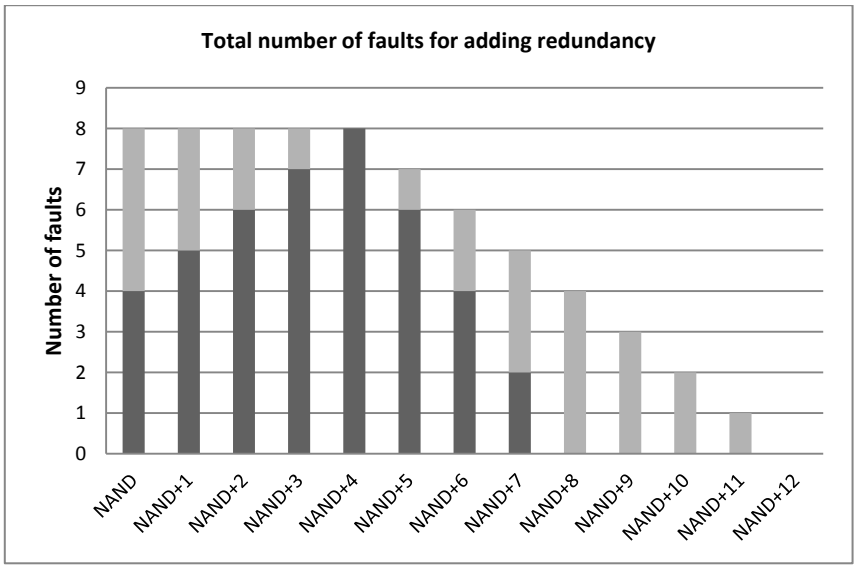


Figure 7.8: (a) FR analysis for the standard NAND gate with increased added transistor redundancy.(b) Total number of faults broken down into state three and fourth per increased added transistor redundancy (The bar is split into top part logic state three and bottom part logic state four)

7.2.5. Scalability of optimised fault-tolerant NAND logic gate

The current research work for the optimised fault-tolerant NAND logic gate investigation was based on a two input logic gate type. The fundamental structure for this type of logic gate is outlined in Figure 7.4(b) and the different configuration transistor arrangements of Figure 7.5 were applied in a systematic way to find the optimum solution for a two input NAND logic gate. The solutions shown in Table 7.2 and Table 7.3 with C3 for pull-up network and C3 for pull-down

network are not limited to two input logic gates. The three input version of the NAND gate with transistors being replaced with BB is outlined in Figure 7.9. Due to the symmetric increase of the BB within the three input NAND gate the fault-handling capability found for the two input NAND gate version (see Figure 7.6) will maintain the same fault conditions for stuck-at faults. The fault count for the single SAH fault-injection of the two input logic gate is shown in Table 7.2. For the pull-x networks variation C3/C3 will increase from eight to twelve for the three input logic gate with the same effect in I_{ddq}.

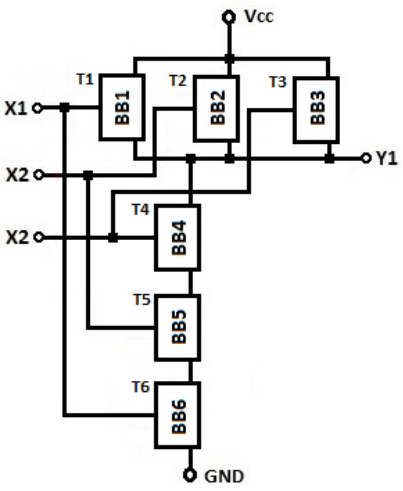


Figure 7.9: Increasing two input NAND gate with BB to a three input version

Because of the symmetric structure of the internal logic gate transistor structure the upscaling to any number of inputs can be done through increasing the number of BB per pull-x network. The behaviour simulation of the standard three input NAND gate on the injection of SAH and SAL faults at each of the individual transistors is represented in Table 7.5. The gate responses match the responses of a two input NAND gate with the perception of having three identical faults where beforehand only two faults had been noticed. This is due to the fact that the three input gate is containing two more transistors than the standard NAND gate per pull-x network. The transforming of the standard three inputs NAND into a SAL fault-tolerant logic gate can be done by using the NAND gate structure outlined in Figure 7.9 with configuration unit C3 of Figure 7.5. The resulting three inputs NAND gate is like the two input optimised NAND single injected SAL fault-tolerant and for single SAH faults the gate is working in logic state four. In Figure 7.10 the internal transistor structure is outlined with all transistors labelled for the single fault SAH or SAL injection at each individual transistor. The corresponding ICs which are required to get this NAND gate into logic state four are indicated in Table 7.6.

				S1 SAH_T1	S2 SAH_T2	S3 SAH_T3	S4 SAH_T4	S5 SAH_T5	S6 SAH_T6	S1 SAL_T1	S2 SAL_T2	S3 SAL_T3	S4 SAL_T4	S5 SAL_T4	S6 SAL_T5
X1	X2	X3	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	0	1	1	1	1	mem	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	0	1	1	mem	1	1	1	1
0	1	1	1	1	1	1	1	1	0	mem	1	1	1	1	1
1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1

Table 7.5: Simulation results of Spice simulation of three input NAND gate with stuck-at fault-injection at individual transistors (mem represents memory effect)

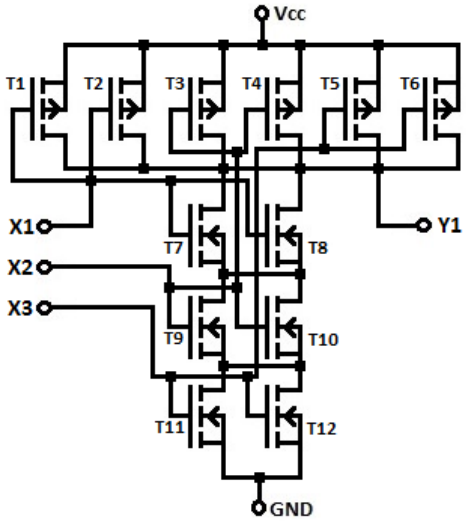


Figure 7.10: Three input optimised NAND gate resilient to SAL faults

Transistor	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
IC	*IC8*	*IC8*	*IC8*	*IC8*	*IC8*	*IC8*	*IC4*	*IC4*	*IC6*	*IC6*	*IC7*	*IC7*

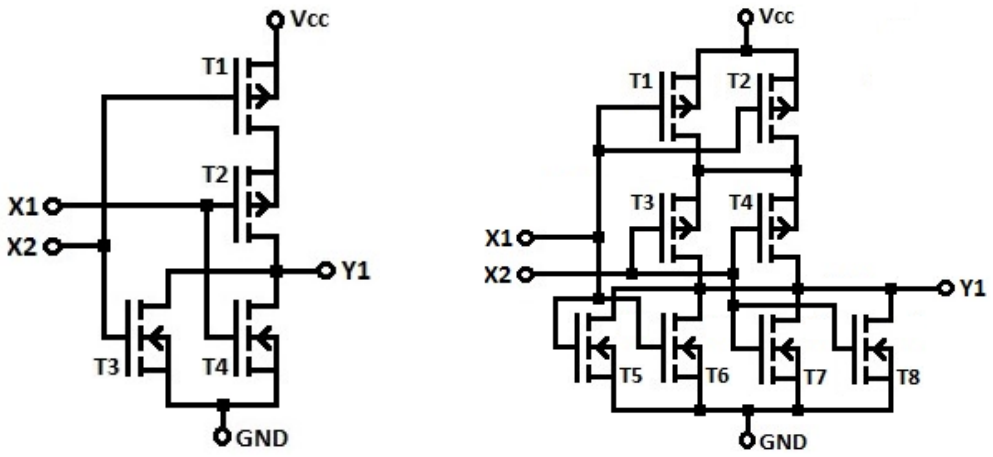
IC = input condition
IC1=000 IC2=100 IC3=010 IC4=110 IC5=001 IC6=101 IC7=011 IC8=111

Table 7.6: Results of SAH fault-injection at each individual transistor of the two input NAND gate and the corresponding IC where the fourth logic state occurs

7.3. Alteration of other fundamental logic gates according to design specification

Within the previous chapters the optimum solution for the NAND gate has been found and evaluated. This NAND gate structure (see Figure 7.6) is fulfilling the specification for a fault-tolerant logic gate with the help of Table 7.2 and Table 7.3 and cross evaluated with Figure 7.8(b). The design structure of the NAND gate is similar to the proposed logic structure for the SAL single fault-tolerant inverter introduced in [20] and shown in Figure 7.3(a). This single SAL fault-tolerant inverter fulfils the specification set in this thesis for the design of a fault-tolerant logic gate. The last fundamental logic gate out of the family of the logic gates is the NOR logic gate. The range of fundamental logic gates are the following types inverter, NAND and NOR. These gates are defined as fundamental logic gates due to the fact that these types of logic gates are designed out of the minimal numbers of individual transistors. The Figure 7.11(a) shows the standard NOR gate structure.

For finding the optimum NOR gate, which meets the same specification defined for the NAND gate, it requires the same analysis strategies to be used. The main difference between the NAND and the NOR gate concerns to the internal transistor circuit structure. The internal structure of the NOR gate (see Figure 7.11(a)) is in reference to a NAND gate (see Figure 7.4(a)) swapped around. This means that the pull-up and pull-down transistor network is swapped. The test follows the same strategy performed at the NAND logic gate by applying single SAH and SAL faults at the individual transistors of the NOR gate for finding the deviating results at the output of the NOR gate. The analysis for finding the optimum NOR gate design has been performed in the same manner evaluated for the NAND gate. This evaluation had been performed by interchanging the different transistor configurations (which are defined within Figure 7.5) within the BBs of the altered NOR gate injecting single SAH and SAL faults at the individual transistors of the logic gate design. Through this the total number of faults has been revealed and the analysis will indicate the optimum NOR gate design. Performing the single SAH and SAL fault-injection on the NOR gate portrays the same total fault numbers as the NAND gate had and which are represented in Table 7.2 for SAH faults being injected and Table 7.3 for SAL faults being injected. With regards to absolute fault count there is no difference between the NAND and the NOR gate. Because of the fault numbers being similar, the same configuration resulting out of Table 7.2 and Table 7.3, which is C3 for the pull-up network and C3 for the pull-down network, can be revealed. The internal circuit structure varies between NAND and NOR in a way that structurally the pull-up and pull-down transistor networks are switched (see Figure 7.2) for the NOR logic gate compared to the NAND logic gate. The optimal NOR gate design fulfilling the specification is shown in Figure 7.11(b).



(a) (b)

Figure 7.11: (a) Standard NOR logic gate; (b) optimised NOR gate resilient to SAL faults

By having these three essential logic gate functionalities of inverter, NAND and NOR gate with the inherent capability of being single stuck-at fault resilient (SAFR) for SAL faults and indicating an SAH fault, all the other logic gates can now be substituted through a structure of the fundamental SAFR type logic gate versions. Through this the fault-handling capability of logic circuits can be improved.

7.4. Converting standard logic circuits into fault-tolerant logic circuits

The effectiveness of the SAFR type logic gates can be evaluated by the means of replacing standard logic circuits with SAFR-logic gates. A direct comparison between the faults-handling performances of a standard logic circuit against the transformed fault-tolerant logic circuit is based on FR comparison of the circuits and this will show the improvement in fault immunity. For this comparison three standard logic circuits were chosen. The first circuit is the full digital 2 bit adder, which is one of the fundamental logic circuits within digital systems and can be scaled for higher bit numbers, if required. The second is the C17 circuit out of the ISCAS-85 benchmark circuit [27] and the last circuit is a three input majority voter.

7.4.1. Comparing a 2-bit full adder design implementation

The full adder is one of the central logic circuit functions in many given digital applications and is used in a vast spectrum of systems. Each microcontroller contains an adder for the required bit size needed for fulfilling the required operational calculations. Optimisation and fault-tolerance for different designs can be found throughout the literature. Concepts of fault-tolerance require a logic checker for identification of faulty output results or a concept of adder redundancy. For this

comparison the full 2-bit adder has been designed only out of NAND logic gates and the circuit is displayed in Figure 7.12. The standard design of the full 2-bit adder uses 11 individual NAND gates and therefore uses in total 44 individual transistors. The SAFR-NAND gate design of the same adder type requires 88 individual transistors within also 11 individual SAFR-NAND gates. The individual FR for the different circuit designs is evaluated by applying single SAH and SAL faults injected at each individual transistor of the individual logic gates of the adder circuit. Table 7.7(a) and Table 7.7(b) represent the FR of these two dissimilar designs, which are using the different logic gate types. Table 7.7(a) shows the simulation results of the standard gates and Table 7.7(b) for the SAFR-gate design. The FR improvement of the circuit design done in SAFR-NAND gates over the standard NAND gates is 3.11 times better. The SAFR-NAND gate improves the FR by 67.9% compared to a non-fault-tolerant logic gate circuit like the one displayed in Figure 7.12. All faults of the SAFR-NAND gate based design are related to injection of SAH faults into the fault-tolerant version of the adder. Each of these faults also increases the I_{ddq} current for the duration of the presence of the fault within the circuit and the required IC. Monitoring of the current by means of an external current monitoring circuit makes it possible for flagging faulty output results or triggering self-repair functionality designed into the circuit. The hardware overhead compared on the basis of individual transistors is 100% due to the redundant structure within the SAFR-NAND gate. This hardware overhead is universally applicable for the use of a SAFR-type gate in comparison to standard gates, which will always be 100% hardware overhead.

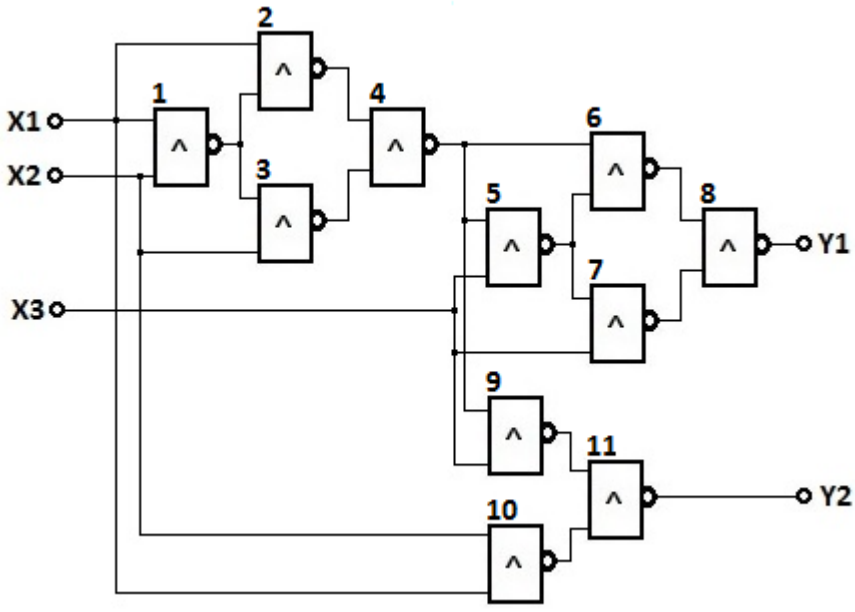


Figure 7.12: Logic gate circuit of a full 2-bit adder constructed only out of NAND logic gate designs

Gate	%FR
1	12,50
2	14,06
3	17,19
4	21,88
5	14,06
6	15,63
7	18,75
8	12,50
9	7,81
10	25,00
11	23,44
avr %FR	16,62

(a)

Gate	%FR
1	6,25
2	6,25
3	6,25
4	6,25
5	6,25
6	6,25
7	6,25
8	4,69
9	3,13
10	6,25
11	6,25
avr %FR	5,34

(b)

Table 7.7: (a) FR analysis of the standard 2-bit full adder.(b) shows the FR of the same logic gate structured using only SAFR-NAND gates for the full adder design

7.4.2. Comparing a C17 circuit design implementation

The C17 test circuit was chosen out of the ISCAS-85 benchmark circuit [27] because the entire logic circuit is created only with NAND logic gates. The C17 circuit is displayed in Figure 7.13, and contains six NAND gates. Converting the C17 circuit into using only SAFR-NAND gates is as easy as exchanging the standard logic NAND gates with SAFR-NAND gates without circuit modifications. The resulting circuit looks the same as shown before in Figure 7.13. The fault-handling performance of both circuits has been done through the injection of single SAH and SAL faults at each individual transistor of each logic gate. With the resulting faults the FR per gate and the overall FR had being calculated. The results of these simulations are displayed in Table 7.8(a) for the standard gate version and Table 7.8(b) for the SAFR-gate design. The results show that the fault-handling capability is 3.09 times better for the C17 circuit, created only out of SAFR-NAND gates. The FR for the only SAFR-NAND gate circuit of the C17 application is improved by 67.7%. All faults are related to SAH-injected faults, which also increases the I_{ddq} current at the same time. Identification of these faults and flagging them is possible through an external current monitoring device. No SAL faults alter the output behaviour of the C17 circuit and if an SAL fault is affecting a single transistor of the circuit it is masked.

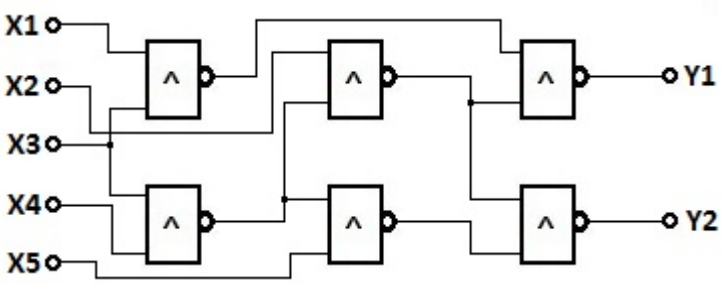


Figure 7.13: C17 test circuit out of the ISCAS-85 benchmark circuit library [27]

Gate	%FR
1	9,38
2	9,38
3	17,97
4	12,50
5	22,66
6	18,75
avr %FR	15,11

(a)

Gate	%FR
1	3,91
2	4,69
3	5,86
4	3,91
5	6,25
6	4,69
avr %FR	4,89

(b)

Table 7.8: (a) FR for the standard NAND gate implementation; (b) FR for the SAFR-NAND gate implementation

7.4.3. Comparing a three input majority voter circuit design implementation

The most commonly used implementation of a majority voter is outlined in Figure 4.7 and the circuit is constructed out of standard AND and OR logic gates. The logic equation of this majority voter design is defined as follows:

$$Y1 = (X1 \wedge X2) \vee (X1 \wedge X3) \vee (X2 \wedge X3) \qquad \text{(Equation 7.4)}$$

The adaption of this majority voter into only using NAND gates is possible and the circuit is outlined in Figure 7.14. The logic equation for this majority voter is the following one:

$$Y1 = \overline{\overline{X1 \wedge X2} \wedge \overline{X1 \wedge X3} \wedge \overline{X2 \wedge X3}} \qquad \text{(Equation 7.5)}$$

The transformation of the majority voter, designed out of only NAND gates, can easily be altered into a SAFR-NAND gate only version. All the required SAFR-NAND gates sizes are available due

to the scaling described in chapter 7.2.5. In this chapter the three input version of a SAFR-NAND gate had been realised with the same behaviourally aspects defined in the specification for the two input gate version. The different FRs for both majority voter implementations has been done by means of injecting single SAH and SAL faults into the individual transistors of the logic gates of both designs. This method has been used already for all the other implementations of circuit adaptations with SAFR-NAND gates. The resulting FR out of the simulation is shown in Table 7.9. The results in this table show that the fault-handling capability of the SAFR-NAND gate based majority voter is 3.5 times better than the standard logic gate implementation. This configuration of the majority voter, designed out of SAFR-NAND gate improves the FR by 71.5%. All of the remaining faults are SAH-related faults and these faults also increase the I_{ddq} current at the same time. This increase of the I_{ddq} current is a clear indication of a fault happening within the majority voter circuit designed, which cannot be masked. The design of the majority voter only out of SAFR-NAND gates makes it possible for identifying these kinds of faults affecting the circuit. The SAFR-NAND gate based majority voter got a fault-tolerant design, which overcomes the fault-related shortcomings of the standard logic gate design pointed out throughout this research work.

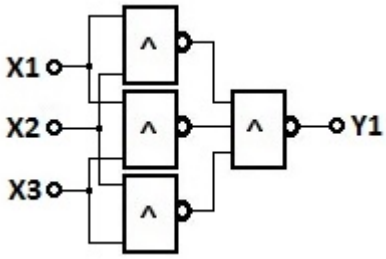


Figure 7.14: Majority voter constructed
out of NAND gate

Gate	%FR
1	6,25
2	6,25
3	6,25
4	17,71
avr %FR	9,12

(a)

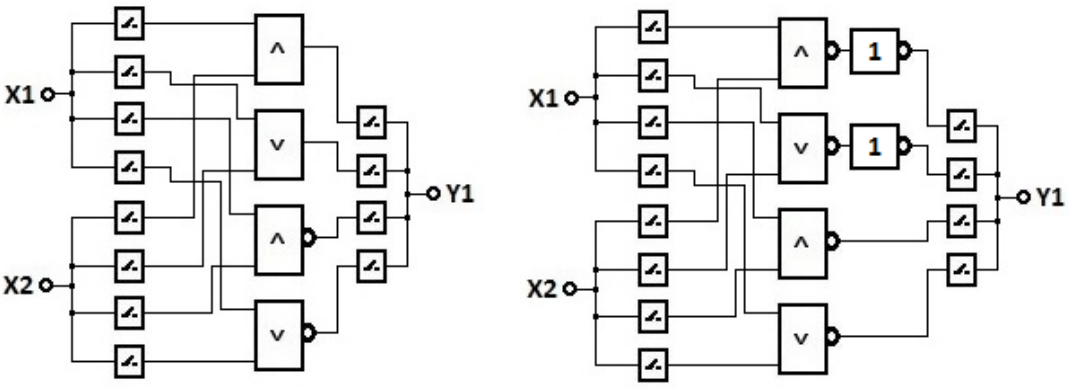
Gate	%FR
1	3,13
2	3,13
3	3,13
4	1,04
avr %FR	2,61

(b)

Table 7.9: (a) FR for the standard NAND gate implementation;
(b) FR for the SAFR-NAND gate implementation

7.5. Converting the logic unit of the QLC into using SAFR type logic gates only

The original design of the logic unit of a QLC had been done in a discrete circuit structure, in which each logic function had been utilised by means of its specific logic gate function and is displayed in Figure 7.15(a) as a copy of Figure 6.4. This means that four logic gates are working side-by-side will increasing the needed chip area unnecessarily. By looking into the transistor count of this design and excluding the switching transistors of this logic unit design done for the QLC shows that it needs to alter the logic unit design. This original design of a logic unit requires 20 signal transistors. The adaptation of the logic unit to work with SAFR-type logic gates only is described in Figure 7.15(b). This design of the logic unit contains all the fault-handling capabilities described through the adaptation of other types of standard logic circuits. The price for the fault-handling performance is that this design out of SAFR-NAND gates of the logic unit needs 40 signal transistors. With a 100% hardware overhead the logic unit has a better fault-handling capability. This type of logic unit design also has another disadvantage through the isolation of all the individual logic gates at default. The unused logic gate is still powered up during the entire power-up time. This increases the power consumption of the logic structure and will make the original design almost not useable. The functionality of the unaltered logic unit requires more coding information defined in Figure 6.8(b). This means in this case 3 bits per logic unit without fault-tolerant information protection is required. In general the switching structure for both designs remains the same and requires in total 24 transistors.



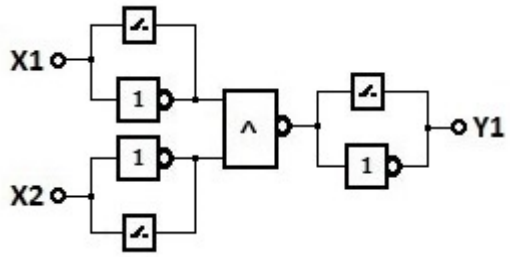
(a) (b)

Figure 7.15: (a) Logic unit design done out of standard logic gates;
(b) Logic unit adapted to work with SAFR-type logic gates

Altering the design of the logic unit away from the discrete circuit structure towards a less hardware requiring version of the logic unit is essential for hardware reduction at the coding level and the signal transistor count. The design of the minimal hardware requiring logic unit by

maintaining equal logic functionality is represented in Figure 7.16(a). This design of the logic unit is formed from SAFR-type logic gates only and it requires only 10 signal transistors without switching transistors. This is 50% fewer signal transistors than the original discrete solution of the logic unit design. The logic unit designs of Figure 7.15 require 24 switching transistors and the one of Figure 7.16(a) only 12 transistors. The total transistor count number for the original design (see Figure 7.15(a)) is 44 and the optimised version (see Figure 7.16(a)) only requires 22, which is a reduction by 50% on the total transistor count.

The fault-handling in regards to single SAH and SAL injected faults is limited to the SAFR logic unit design. This is per SAFR-type logic gate design used in the logic structure single SAL fault-tolerant and indication of SAH faults. Certain types of single SAH faults are masked within the circuit structure and, if not possible, a clear indication through an increase of I_{ddq} current flags it to an external hardware checker. The coding of the logic selection of this design is much more compact and requires for all the altered logic units of a QLC only 2 bits per logic unit configuration. The default logic functionality is AND logic function and through this logic functionality is always connected between input and output pins.



(a)

Select	Coding	Logic Fkt
1	00	AND
2	01	NAND
3	10	OR
4	11	NOR

(b)

Figure 7.16: (a) Optimised logic unit towards minimal logic gate use and minimal coding bits; (b) Coding table for the selection of required logic function of the minimal hardware requiring logic unit

7.6. Summary of the chapter

This chapter focused on answering the following research question: would it be possible to alter the fine-grained transistor structure of a logic gate to be better equipped against stuck-at faults at the transistor level with a minimal hardware overhead and what fault-handling impact on a given logic circuit can be achieved? For answering this question a detailed analysis of the entire different possible transistor redundancy alteration within a certain range had been done and applied into generating defined logic functionality. The upper range was defined by a known fault-tolerant fine-grained approach of the quadded transistor structure where every transistor is replaced by a network of four transistors. The quadded transistor structure is resilient against single SAH and SAL faults by having 300% hardware overhead. The basic logic function for this research is the NAND gate and all the different transistor structural variations of these logic gate configurations were exposed to fault-injection of stuck-at high or low at the individual transistor forming these logic gates. The FR data found indicated that for a certain minimum number of added redundant transistors done in a certain way the newly created NAND logic gate was masking SAL faults. Due to this feature this type of gate is referred to as SAFR-logic gate. The SAFR-gate responded for all SAH faults injected into the gate in conjunction with certain input pattern stimulation with a short between V_{cc} and GND rail. Through this short the I_{ddq} increased significantly and this increase of the I_{ddq} current is usable as an indication signal for non-maskable SAH faults presented within the gate. The resulting form of adding redundant transistors to a given logic gate was applied onto the NOR gate with the same behavioural responses as the altered SAFR-NAND gate. Validation of the approach can be used for upscaling the number of inputs of a given logic gate and maintains the same gate behaviour showing that this was feasible for any number of added inputs to a gate.

The question about equipping the newly created SAFR-logic gate with an intrinsic built-in feature for indicating that a non-maskable fault is affecting the gate can be answered. The newly created SAFR-logic gate increases the I_{ddq} considerably through logic state four and because of this increase it can be monitored and can be reacted upon. Due to the level of this current increase it is a unique indication of this fault condition. The part of the question about the possibility of triggering a self-repairing or self-healing based on this current signal will be answered within Chapter 9.

The comparison between FR of logic structures with and without this SAFR-logic gate is summarised in Table 7.10. The data shown in Table 7.10 compares a set of logic circuits designed out of standard and SAFR-logic gates on the basis of FR data. The FR data has been created through injection of SAH and SAL faults into the individual transistors of each logic gate of the circuit. Each logic circuit created out of SAFR-logic gates has a lower FR by far than the one created out of standard logic gates. For all the non-maskable faults of the SAFR-logic gate circuits the occurrence is indicated through the increased I_{ddq} current to the system outside. In this way the

circuit designed out of SAFR-logic gates are fault-tolerant through fault-masking and indication of non-maskable faults.

	avr %FR Stand.	avr %FR SAFR
Full adder	16,62	5,34
C17	15,10	4,88
Majority voter	9,11	2,60

Table 7.10: Comparison between the standard and SAFR logic gate-created logic circuits

The fine-grained redundancy at the individual transistor level increases the hardware requirement by 100%. If the SAFR-logic gates are going to be used for example, for creating a TMR system, the hardware requirement for this logic structure will be at 300% besides the hardware overhead of a TMR system. This would make the hardware overhead of the entire TMR system at 500%. The hardware evaluation indicates that at this moment the usefulness of creating an entire system out of SAFR-logic gates would be questionable. As a general concept and because of the evaluation of the fault-behaviour of a majority voter the design of the entire majority voter out of SAFR-logic gates offers a fault-tolerant benefit. Because of this fault-tolerance benefit the newly designed majority voter will mask all SAL faults and indicate the presence of SAH to a monitoring system.
In accordance with equation 4.4 the reliability of a TMR system with majority voter depends on the reliability of the majority voter because of the directly multiplication with the reliability of the TMR system. Any improvement done to the majority voter increases the reliability of the entire system by far. A majority voter created out of SAFR-logic gates will increase the fault-tolerance and the analysis of the FR is shown within Table 7.10.

Chapter 8: Mapping FSM functionality into memory

8.1. Introduction

FSMs are, in our electronic system world, the central type of logic application in use for a wide range of different applications. These applications can be a simple vending machine or an engine controller of an airplane. This extensive spectrum of applications is due to the fact that within a FSM only one process state is active and an alteration out of this state can only be done through defined ICs and stored information. The common approach of the implementation of FSMs is done based on a programmable logic device (PLD) or combinational logic done on a programmable platform or individual logic elements. All of these FSM application platforms have advantages and disadvantages in regarding hardware and software design. PLD-based FSMs are controlled by means of a solution specific programmed state flow, which can be done with a wide range of programming languages. Combinational based FSMs are fixed within their state flow due to their fixed interconnection of the individual logic elements, and it can be done on a programmable logic platform like an FPGA alteration by means of altering the logic structure and reprogramming. For a hardwired individual logic elements solution of an FSM any alteration can only be done with redesign of the hardwired interconnection and if needed by the changing of logic elements. In the case of adding fault-tolerance or self-healing capabilities to any of these different types of FSMs the complexity and hardware overhead will be significant.

An alternative solution for creating an FSM can be done by mapping the state flow into individual unique binary sequence information, which then can be stored in a memory unit. For accessing this information stored inside the memory a simple addressing logic circuit is required. The input and output data requires another simple circuit for extracting the data out of the stored memory data. All this combined into one system creates a memory-based FSM. Due to its simple hardware structure in terms of ‘memory-mapping’ of logic inside a uniform memory element makes adding self-healing concepts feasible. This is because of the use of only memory and data stored within it. Including fault-tolerance concepts to this memory-based FSM can be done regardless of the FSM state flow.

Within this chapter the first part of the research question, if it can be possible to create an FSM with minimal fault-tolerant hardware fulfilling the task of fault location identification within a given logic structure, is going to be investigated. Due to the use of SAFR type logic gates, the fault finding focus can be placed on the detection of interconnection faults. Test and detection concepts for interconnection faults require a sophisticated system checker and the system’s behaviour is governed by the FSM principle. This system checker should be based on a fault tolerant approach whilst requiring minimal hardware structure. For this case, the minimal fault-tolerant hardware is based on using a memory-only based platform in which the FSM logic functionality is mapped into

memory data entries. The state transition of this FSM is based on only on reading memory locations in which the required state behaviours are encoded. This novel FSM platform concept is used as the functional platform for the design of a system-checker identifying fault interconnections within a given logic structure. This research work is part of Chapter 9. Because of the minimal hardware requirement the combination of FSM functionality mapped into memory-only will build the base of a system-checker. The advantage of this approach is that fault-tolerance based on data protection does not require sophisticated hardware. By including fault-tolerance in the system-checker competence makes any type of self-checking concepts obsolete.

8.2 Principle of FSM architecture

In our lives we are using digital systems in a vast variety of applications, the concept of having only one single state or instruction active at one time within these systems is the unique behaviour of an FSM. Another feature of an FSM is that it contains a fixed mapping between input stimulus and output pattern stored within one state. Because of this fixed behaviour the concept of an FSM is the central and most common application controlling concept inside electronic systems. We can find it within a vending machine, a turn cross, an ATM machine and a safety-critical system inside an automobile. In all these applications an FSM concept is governing the controller [125]. The transition from active state to another state can only be done through specific input stimulus and stored data. The basic block diagram of an FSM includes logic structures, which contain decisions and memory element to store events [55]. Another example of the use of the FSM principle is the "Turing machine" of Alan Turing, which makes it possible to model the behaviour of a computer within a mathematical model. The basic principle of the "Turing machine" is the idea of an infinite memory tape, on which a read/write head controlled by a programme manipulates symbols [126]. The central control for the "Turing machine" is the programme, which is controlling the behaviour of the head. A programme is in general being described as an FSM implementation controlling the input and output activities of the machine.

A formal and general description of the input/output mapping of an FSM can be done through equations. In these equations the logic design of the FSM with a certain number of inputs *I*, outputs *O* and states *S* can be specified as a 5-tuple (I,O,S,δ,ω). The state transition function of the FSM is defined as $\delta : I \times S \rightarrow S$ and the FSM output function is defined as $\omega : I \times S \rightarrow O$ [31, 56].

How the output response is controlled within the FSM defines the type of FSM to be either a Moore or a Mealy state machine. An FSM-based on the Moore concept define that the output function *(O)* is controlled by the current state *(S)* only and not by the input stimulus *(I)*. The formal definition is defined as $\omega : S \rightarrow O$ and a block diagram of a Moore-based FSM is shown in Figure 8.1(a) [28, 54, 55]. The Mealy concept applied onto an FSM defines that the output function *(O)* depends on the state *(S)* and the input stimulus *(I)* and is formally defined as $\omega : I \times S \rightarrow O$. The

block diagram of the Mealy-based FSM is illustrated in Figure 8.1(b) [28, 53, 55]. Amongst these two concepts there is a uniquely important dissimilarity within their state structure. This is that a Mealy FSM requires less state transition compared to a Moore FSM. Within a Mealy-based FSM the state structure is of a more compact state transition structure then the Moore state transition structure [127].

The FSM is controlled through the state transitions, which are defining the required input stimulus for a transition and, if applicable, the required output information. The entire state transition flow is defined within the state diagram and all the information of the state diagram can be transferred into a state transition table. An example of a state diagram of a JK-flip-flop (JK-FF) is illustrated in Figure 8.2(a). The truth table of the example JK-FF is displayed in Figure 8.2(b) and represents the general behaviour specified as the user of this type of flip-flop (FF) and will experience if it is used as a black box or part of a logic circuit. Figure 8.2(a) shows the state transition within the JK-FF and at each state transition indicated through an arrow at which the required input stimulus and corresponding output sequence is defined. This information along the arrows has the format x_1x_2/y_1y_2 in which x_1x_2 represents the input sequence and y_1y_2 the output sequence. The input sequence defines the required input stimulus for the state transition change and for the example x_1 is representing the J-input, which makes x_2 to be the K-input. The corresponding output sequence defines the data produced at the output of the JK-FF and has the following format where y_1 is representing the Q-output, which makes y_2 the inverse of the Q-output and therefore it is possible to eliminate y_2 from the arrow in a general way. The transformation of all this information of each state transition is specified by the truth table (see Figure 8.2(b)) [128].

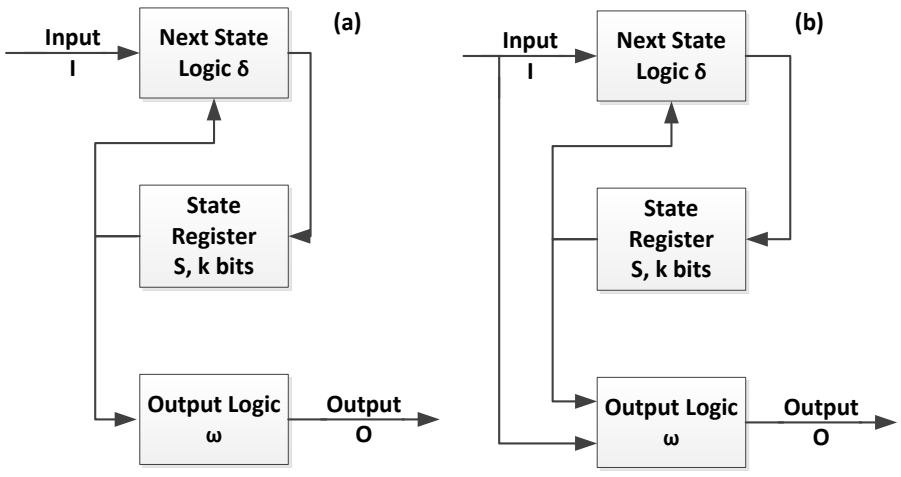
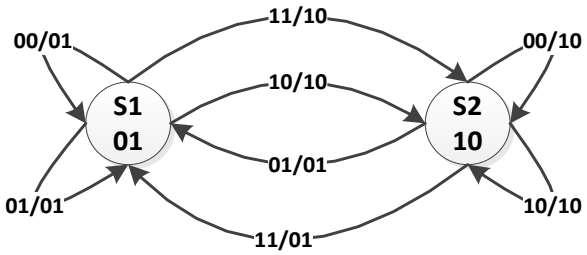


Figure 8.1: (a) Block diagram of a Moore-based FSM;
(b) Block diagram of a Mealy-based FSM [28]



(a)

J	K	Q
0	0	unchanged
0	1	0 (FF reset)
1	0	1 (FF set)
1	1	toggle

(b)

Figure 8.2: (a) state diagram of a JK-FF; (b) truth table of the JK-FF

By filling the state table for special cases where the output function *(O)* is independent of the previous state, a 'don't care' condition is possible within the state transition table and the functional representation of this condition is λ. Due to this the logic definition of this FSM changes from a 5-tuple definition into a 6-tuple one, which is defined as $(I,O,S,\delta,\omega,\lambda)$ [127]. The state transition of the Moore and Mealy concepts defines the basic functionality of an FSM. In the case that an FSM is coded with the help of a higher-level descriptive programming language there are three commonly used coding methods applicable [29]. These different coding styles are illustrated in Figure 8.3(a to c) [30].

The coding method of a combined single process (CSP) uses a single state, which controls both the state transition and the output functionality of the FSM coded in this programming style. The output information is stored in a register and is maintained as long as the register information does not get altered. A block diagram of the CSP-FSM programming structure is displayed in Figure 8.3(a). The coding style state-separated combinatorial output (SCO) uses two states for a state transition and the output function is directly generated out of combinational logic. The output function is not stored in a register of the CSP style. This is because it is generated through combinational logic, which requires no memory. A block diagram of the SCO-FSM programming structure is shown in Figure 8.3(b). The coding style state-separated registered output (SRO) is of the same programming style as the SCO style, but only an output register is added to this FSM coding structure. Within the state flow there is a single-state delay in the generation of the output signals after the decoding through the output combinational logic. A block diagram of the SRO-FSM programming structure is illustrated in Figure 8.3(c) [29].

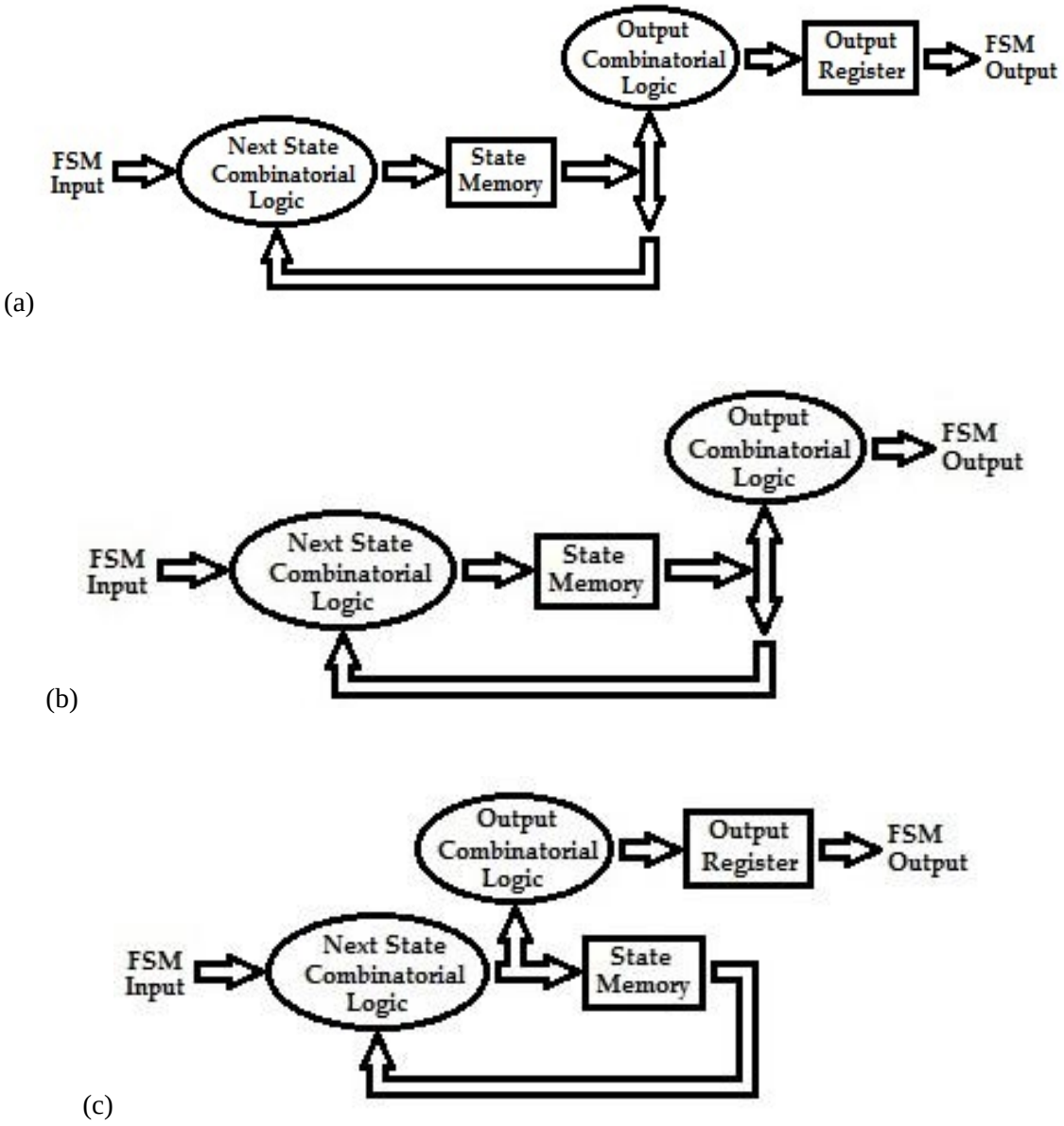


Figure 8.3: Demonstration of the three different coding style within block diagrams
(a) Coding style combined single process (CSP);
(b) Coding style state-separated combinatorial outputs (SCO);
(c) Coding style state-separated registered outputs (SRO) [29, 30]

8.3. Objective of mapping FSM logic functionality into memory

FSM-based electronic systems are used within a wide range of applications. This is because of the unique way FSMs are performing through a given process description by only allowing one active state at one time and a state transition can only be triggered through a unique input stimulus. Output functionality is linked to the state transition and follows a defined sequence. This unique processing sequence performed by an FSM can be converted into unique digital information, which

can be stored inside a memory-based LUT. This transformation of the state transition description into unique digital data can be performed for a Moore or Mealy FSM. The difference between both state machine styles is in the way the output information is generated, which has to be taken into account.

The elementary principle for the transfer of the state transitions of an FSM into an LUT memory structure requires that it is essential that for each state transition a memory row is allocated, which is uniquely addressable. It is vital that for the stored data within the LUT memory for each LUT memory row the same data structure has to be used [129]. The minimum numbers of rows within the LUT memory is defined by the number of state transition entries used for the FSM. By using a linear addressable LUT the number of addresses is defined by 2^{N_addr} in which N_addr is the binary number of uniquely necessary addresses for the access of the LUT memory. The address of the LUT is a combination of the number of inputs stimuli and the number of state transitions encoded within binary bits [125]. Because of this the size of the LUT can be quite big with not all LUT memory rows containing state information of the FSM state transitions. Also the number of addressable rows grows exponentially due to 2^x. By transforming the state transition information of an FSM into a uniquely addressable LUT memory containing the state information the LUT memory can be accessed through a simple address register. The hardware requirement for this memory-based FSM design is reduced compared against a PLD-based FSM.

The block diagram of this memory-based FSM is displayed in Figure 8.4 with the central deviation compared against the three different coding styles, which are illustrated in Figure 8.3(a) to Figure 8.3(c). This deviation is the state memory pointer used for accessing the required state transition within the LUT and also contains the output information of this state transition. The state pointer contains the following information's input stimulus, state transition and output information within binary-coded form. By combining these data they are forming a unique address for accessing the required LUT row which is necessary for reading the required state transitions information.

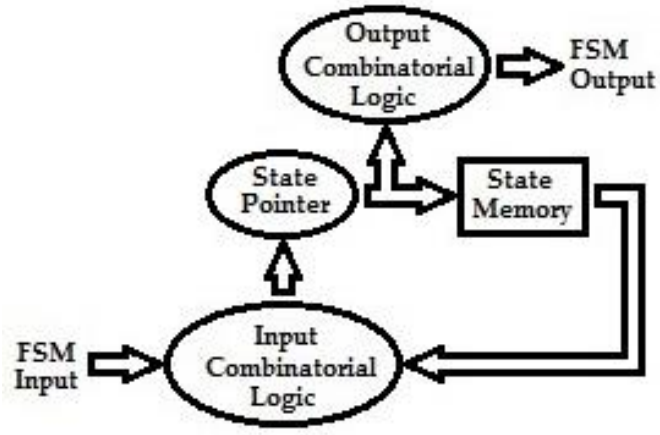


Figure 8.4: Block diagram of the memory-based FSM structure [30]

8.4. Mapping of a FSM logic functionality into memory

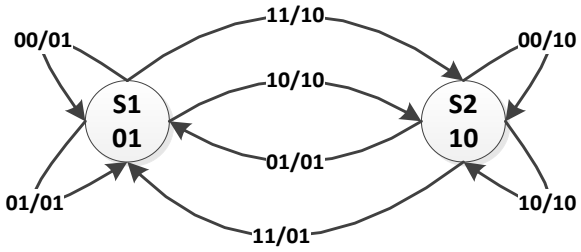
Mapping logic-based FSM functionality into a memory-based FSM requires access to the state transition information. The state transition description of any FSM can be done within a state transition diagram or a state transition table. Both define the state transition behaviour of the FSM and for creating the memory-based LUT the entries of the state transition table contain in general the necessary starting point information for the transfer of the state table into LUT memory. The state transition table can be created out of the state transition diagram if the state table does not exist for this FSM. The first step of transforming the state transition table information into unique LUT memory addresses is already achieved through unique binary coding of the individual state transitions and including the binary-coded input stimulus within the table. In most cases the second step is the optimisation of the input and output data including the arrangement of the data structure within the memory entries. With the help of the following examples the application of this transfer process will be demonstrating.

8.4.1. Mapping a JK-flip-flop into memory

A JK-FF is a simple FSM because it can only be in one state at a time and this is the definition of an FSM. The state diagram of a JK-FF is specified within Figure 8.5(a). Within this state diagram for each state transition the required input stimulus is defined at each state transition. Another possible way to define the state transition is to use a state transition table and the corresponding state table for a JK-FF, which is displayed in Figure 8.5(b). The structure of the state table is as follows. The column labelled with *input* defines the required input stimulus, (indicated by red numbers) which is necessary for altering the current state transitioning to another state. The state transition is defined within the column *state* where the left slide represents the current state (indicated by green numbers) and the right side indicates the next state where the FSM is going to be transitioned (indicated by blue numbers) if the corresponding input stimulus of this row are applied to the FSM. In the case of the JK-FF FSM which has two states *s1* and *s2*, the *output* column of the table specified in Figure 8.6(b) contains the associated output information (indicated by yellow numbers) for the current state transition.

For transforming the state table into LUT memory information the states *s1* and *s2* of this table, which are displayed in Figure 8.6(a) have to be uniquely binary-coded. This binary-coded information is used for replacing the states within the state table accordingly. For the JK-FF FSM *s1* is coded with '*01'* and *s2* with '*10'* in the state table of the JK-FF FSM example. The adaptation of the coding of the state table for the JK-FF FSM example is represented in Figure 8.6(b). Figure 8.6(a) shows the JK-FF state table without state substitution. In Figure 8.6(b) the state table is displayed with the uniquely binary-coded state information replacing the states of the state table.

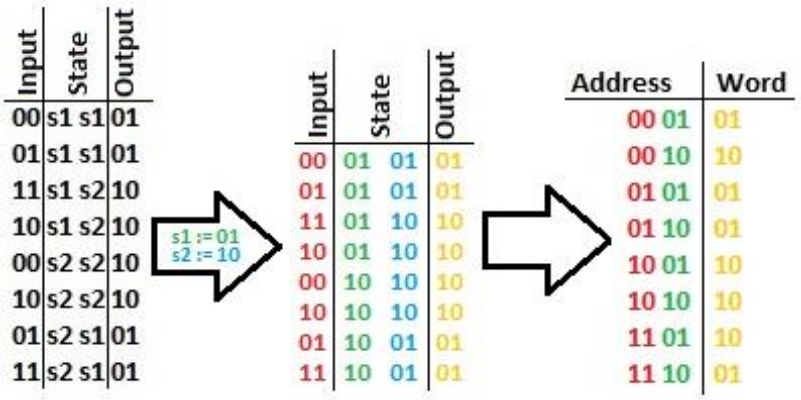
Out of the table described in Figure 8.6(b) the required memory LUT information can be derived from and is illustrated in Figure 8.6(c). The information within this table represents the following memory information. On the left side of the table the unique memory address information for the current FSM state can be found and on the right side the next state of the FSM is defined after transition triggered by input stimulus. In this special case for the JK-FF FSM the next state and the output information are of the same binary value. Due to this information can be combined by using only one data word within the memory.



Input	State	Output
00	s1 s1	01
01	s1 s1	01
11	s1 s2	10
10	s1 s2	10
00	s2 s2	10
10	s2 s2	10
01	s2 s1	01
11	s2 s1	01

(a) (b)

Figure 8.5: Shows the state information required for the JK-FF FSM
(a) state diagram of a JK-FF; (b) state table of a JK-FF



(a) (b) (c)

Figure 8.6: The state transition table of the JK-FF is transformed into a memory-usable table for a memory-based FSM adaptation [31]; (a) state table; (b) state table with binary-coded state replacement; (c) memory LUT information

As indicated within Figure 8.6(c) the address length of the input stimulus is four bits long and this makes the state pointer also four bits long. The state pointer is split into two sections. The upper two bits are for the data of the input stimulus and the lower two bits are for the coded states. This information shown in Figure 8.6(c) can now be transferred into the memory LUT block of the memory-based FSM shown in Figure 8.4. The memory LUT-based state transition through the different states works as follows and defined by the block diagram. The state pointer is loaded with the reset or power-up address-pointer *"0000"* and with the word information *"01"* into the lower two-bit side of the state pointer. The memory-based word information *"01* is the data of the memory with the address *"0000"*. The upper two-bit side of the state pointer is reserved for the input stimulus. After the input stimulus has occurred and is stored inside the state pointer register, a memory-read signal is generated for reading out the next state information out of the memory labelled by the state pointer data. The storing of the new data in the lower two-bits of the state pointer happens after the read-out of the memory LUT.

The size of the memory is defined by the length of the data word stored within the memory row and the number of necessary addressable memory rows. The necessary addressable rows within the memory LUT can be calculated with:

$$N_{row} = 2^{N\_SP\_bit} \qquad \text{(Equation 8.1)}$$

In this equation the N_SP_bit stands for number of bits of the state pointer, which are used for the JK-FF implementation, which is four bits and in this case N_{row} is sixteen. The example for the JK-FF has only eight states, which are defined within Figure 8.5(b) and does not include the reset entry requirement. In this case for the JK-FF it means a total of nine rows or addresses used of the sixteen memory rows of the memory-only-based LUT FSM. The remaining seven rows are considered out of the state transiting prospective as 'don't care' addresses, which are part of this memory LUT-addressing range due to the linear addressing. These 'don't care' addresses in the LUT memory are, if retrieved by the state pointer, undesirable states of the JK-FF FSM. Filling these 'don't care' rows with a defined state to avoid the fact that the JK-FF FSM alters its state into undefined state condition does not solve the problem. The JK-FF has two stable state conditions and, by only using one of these defaults state conditions; the applied state as default will have a 50% chance of being the incorrect default state. For avoidance of having a memory LUT with 'don't care' rows would be the better solution and this can be achieved with elimination of the linear memory against a content-addressable memory (CAM).

8.4.2. Mapping of an FSM into memory LUT

The mapping of an FSM into memory LUT will be shown with the example of a vending machine, which is designed to release a can of soda after the price of 30 cent is paid. In the case of over payment the vending machine will release a can of soda and the money which has been overpaid. The vending machine works as follows: A can of soda has the price of 30 cents and the machine accepts 5 cents (N), 10 cents (D) and 25 cents (Q) in any order until the price of 30 cents is reached. This example has been published in [32] with a minimal state diagram, which is shown in Figure 8.7. Within this example of [32] for the state marked with *'15'* the input stimulus *'D'* with the state transition to *'25'* is missing. Within [30] an altered version of the soda-vending FSM is illustrated, which includes the missing state transition and an optimisation regarding number *'return* D' has been performed. The state diagram of [30] is shown in Figure 8.7. Both state diagrams, which are displayed in Figure 8.7 and Figure 8.8, are lacking the correct input stimulus definition at each state. The missing state transition is the loop-back state transition until at each state another coin has been applied into the vending machine. For both figures the state transition through the vending process has been pre-defined instead of a randomly selected sequence.

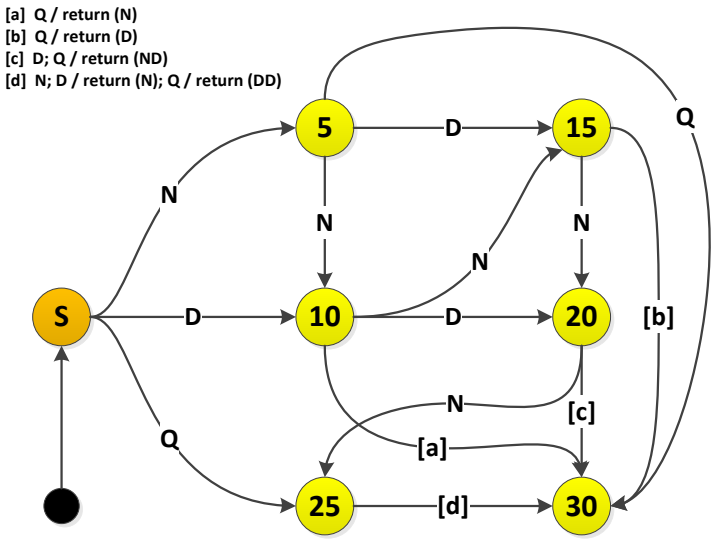


Figure 8.7: FSM state diagram of the soda machine [32]

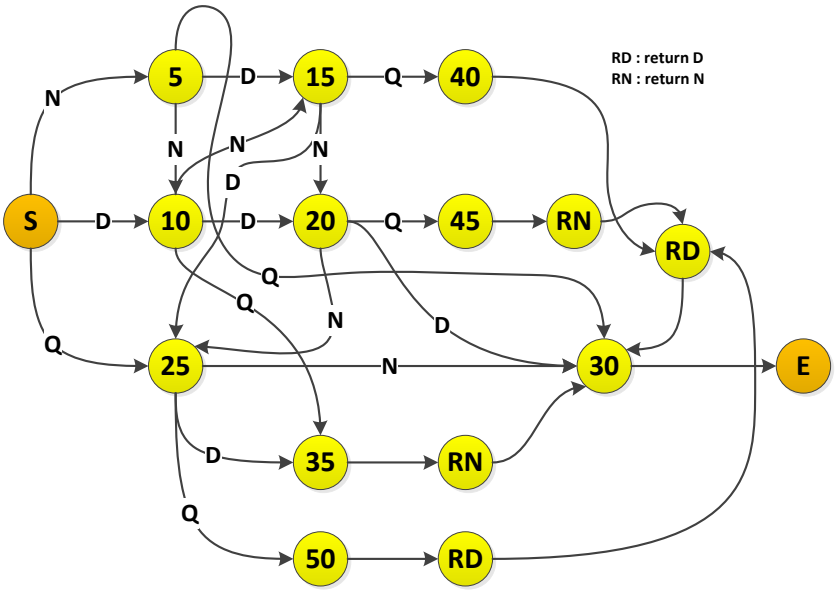


Figure 8.8: Optimised FSM state diagram of the soda machine [30]

The vending machine FSM is defined with all input stimulus and output functions per state transition and is described in Figure 8.9. For this state diagram the assumption for making this FSM-based machine work is the following one. There has to be at all times sufficient coins for change and also sufficient numbers of cans of soda inside the machine. If a check of both of these minimum levels is necessary to be added to the state diagram, it has to be checked right after the start for one transition run through the state diagram. The first check is going to be the defined minimum levels of coins inside the machine at this time and is necessary because the number of coins with variable nomination is unknown when starting. The second check is going to be the level of soda cans and will be done right after the finished run through the state diagram as part of the loop-back to the start.

The definition of the input stimulus per state transition, for example, has been done in this way that *'I:D'* represents *I* for input stimulus with a 10 cents (D) coin. At each state the loop-back transition has been done with the input stimulus defined as *'I:~C'* where as long as no coin *(C)* is inserted into the vending machine, this state transition is true and will be maintained. The *'~'* represents in this case the no function. The output function is defined at the state transition triggered by an input stimulus and is defined as *'O'* as output with the corresponding coin selection in accordance with the coin definition. The *'soda'* state is incorrect due to the fact that a dispensing function should be in accordance with the input trigger after reaching 30 cents or more as part of the normal output function. But this is a final state generated out of a state transition triggered by an input stimulus. This output state transition will be performed in conjunction with this reaching the 30 cents and an input stimulus will be generated within the state transition flow. This state has been added to the state diagram to indicate the end of the FSM state transition and the loop-back to the beginning.

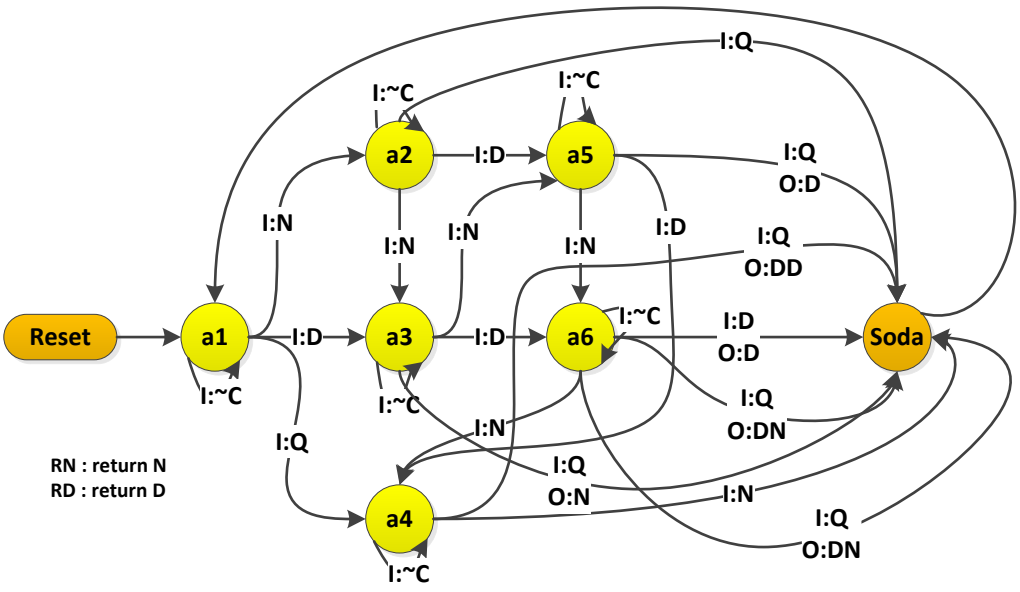


Figure 8.9: State diagram of the soda machine with definition of all input stimuli and output functions per state transactions as required for the state diagram

Transferring the state diagram, which is shown in Figure 8.9, into a data structure that can be used within a memory LUT-based FSM is illustrated in Figure 8.6 and applied onto the soda machine FSM. The result of this transfer into memory LUT is illustrated in Table 8.1. Table 8.1(a) is illustrating the results of the transformation of the state diagram information into the state transition table. Each state transition is represented with at least one line in the table. The table entry for the loop-back transition at each state waiting for the insertion of a coin is transferred as *'---0'* into the state transition table input column (see Table 8.1(a) column *Input*). The information '---' in front of the *'0'* of this input row entry, represents the 'don't care' condition for these particular input stimuli. Regardless of these inputs this state transition is going to be executed as long as the input stimulus coin is zero. In the case of an insertion of any type of coin this input flag will alter to one in conjunction with the input stimulus representing the type of coin. For simplification of the input stimulus information the data can be coded with the coding information shown in Table 8.2(b) and similar coding has been done for the output information with the help of Table 8.2(c). Coding of the input information offers another benefit and this is due to the coding a linear structure within the input stimulus can achieve. The secondary benefit through this effect is that the addressing structure can be altered into a linear format if applicable.

After both alterations, by applying the coding information onto the table data of Table 8.1(a), the reduced table is outlined in Table 8.1(b). The next step towards memory LUT usable data is to replace the states with the coding information shown in Table 8.2(a) and the result of this state coding is outlined in Table 8.1(c). Adapting Table 8.1(c) towards usable data for memory LUT FSM requires a unique addressing structure for each row of the state table. Because of creating

unique addressing, the data of Table 8.1(c) is required to be reformatted for a specific format. The memory address format needs to be of the following format. The coded state information needs to be within the high part of the address word and coded input stimulus in the low part of the address word. Applying this format onto the address word creates a unique address-pointer and the result is shown within the address column of Table 8.1(d). The selection of the address format has been done because the coded state information is unique by itself and the coded input stimulus is a linear-bit sequence done with a fixed number of bits. The combining of this information for each row of the state table will generate a unique address as demonstrated in Table 8.1(d), but it is possible that unused addresses are amongst these sets of addresses. These unused addresses within a linear addressable memory could potentially affect the transition of the FSM if these addresses are selected by random chance.

The format of the word data of Table 8.1(d) has been created out of the combination of the coded state in the upper part of the word, which is followed by the coded input stimulus *'no coin'* in the middle part of the word and the output coding in the low part of the word. The input stimulus '*no coin'* is a fixed value throughout all the words in Table 8.1(d) because these bits are going to be replaced with actual coded input stimulus data for triggering a state transition.

Input	State	Output
---0	a1 a1	0000
0011	a1 a2	0000
0101	a1 a3	0000
1001	a1 a4	0000
---0	a2 a2	0000
0011	a2 a3	0000
0101	a2 a5	0000
1001	a2 a1	0001
---0	a3 a3	0000
0011	a3 a5	0000
0101	a3 a6	0000
1001	a3 a1	0011
---0	a4 a4	0000
0011	a4 a1	0001
0101	a4 a1	0011
1001	a4 a1	1101
---0	a5 a5	0000
0011	a5 a6	0000
0101	a5 a4	0000
1001	a5 a1	0101
---0	a6 a6	0000
0011	a6 a4	0000
0101	a6 a1	0000
1001	a6 a1	0111

(a)

Input	State	Output
00	a1 a1	000
01	a1 a2	000
10	a1 a3	000
11	a1 a4	000
00	a2 a2	000
01	a2 a3	000
10	a2 a5	000
11	a2 a1	001
00	a3 a3	000
01	a3 a5	000
10	a3 a6	000
11	a3 a1	010
00	a4 a4	000
01	a4 a1	001
10	a4 a1	010
11	a4 a1	011
00	a5 a5	000
01	a5 a6	000
10	a5 a4	000
11	a5 a1	100
00	a6 a6	000
01	a6 a4	000
10	a6 a1	000
11	a6 a1	101

(b)

Input	State	Output
00	000 000	000
01	000 001	000
10	000 010	000
11	000 011	000
00	001 001	000
01	001 010	000
10	001 100	000
11	001 000	001
00	010 010	000
01	010 100	000
10	010 101	000
11	010 000	010
00	011 011	000
01	011 000	001
10	011 000	010
11	011 000	011
00	100 100	000
01	100 101	000
10	100 011	000
11	100 000	100
00	101 101	000
01	101 011	000
10	101 000	000
11	101 000	101

(c)

Address	Word
00000	00000000
00001	00100000
00010	01000000
00011	01100000
00100	00100000
00101	01000000
00110	10000000
00111	00000001
01000	01000000
01001	10000000
01010	10100000
01011	00000010
01100	01100000
01101	00000001
01110	00000010
01111	00000011
10000	10000000
10001	10100000
10010	01100000
10011	00000100
10100	10100000
10101	01100000
10110	00000000
10111	00000101

(d)

Table 8.1: These tables are showing the transfer of state transition table information into memory LUT data : (a) state transition table; (b) state transition table with coded input and output stimulus in accordance to Table 8.2(b & c); (c) state transition table like (b) with coded states according Table 8.2(a); (d) memory LUT data

State	Coding
a1	000
a2	001
a3	010
a4	011
a5	100
a6	101

(a)

	Input	Coding
no coin	---0	00
N	0011	01
D	0101	10
Q	1001	11

(b)

Output	Coding
soda	--1
N	-1-
D	1--

(c)

Table 8.2: Coded information: (a) different states; (b) input coin information; (c) output information

The total number of addresses for this soda memory LUT-adapted FSM example can be calculated with the equation 8.1. For the variable N_SP_bit the value of 5 bits is used for addressing a row of the memory LUT and for this case N_{row} is 32. In this example the numbers of rows defined within Table

8.1(d) is 24. In this example eight rows are unused within the memory LUT structure of the FSM. Due to the address structure these eight unused addresses are beyond the used memory rows and a simple address-limit detector can be added to the circuit to protect the FSM from upsets. In the case of elimination of these rows of addresses a CAM can also offer the necessary expectation.

8.4.3. Comparison between memory LUT and PLD

The implementation of the soda vending machine into two different application platforms forms the base for the comparison. For the first implementation the memory-mapped logic solution of the FSM has been chosen and the second one is a PLD-based platform, which in this case is an 8051 microcontroller development board Silab C8051f120. The implementation of the FSM has been coded in assembler programming language. (see appendix 9 & 10) It was used for the implementation due to the fact that the assembler code is hardwired logic functionality of the chosen microcontroller and can be directly executed by defined clock cycles. The coding style for both implementations has been done for both adaptations of the FSM into the platform in accordance with Figure 8.4. For the PLD platform it is implemented by using a limited number of assembler commands, which are MOV, ADD, AND, XOR, relative jump at zero and jump. The assembler programme has been done within a total byte count of 94 bytes for this application. With this assembler code a comparable state table handling programme has been created, which reads out a memory address and decodes the information for the next state transition. In this regard it is acting as a state transition pointer working through the state transition table and creating the necessary output information in accordance with the state table. Because of this fact the state table also needed to be implemented within the 8051 microcontroller memory, which fills up 24 bytes. So the total byte usage within the PLD-based platform is 118 bytes. The implementation of the memory-mapped logic solution of the FSM only requires the state transition table, which needs to be implemented into a memory block. The memory usage of the state transition table is 24 bytes and it is of the same format as the one for the PLD application. The memory usage ratio between these two application platforms is 4.92. The memory-mapped logic solution requires 4.92 times less memory that the PLD-based solution.

The comparison of the runtime within the state transition table is evaluated for different cases and the results are shown in Table 8.3. For the creation of the data of the table the coin wait-related times have been excluded from the data for both applications, because of the unpredictable time between injecting coins.

Case	Coins	PLD	MMLS
1	5*5 cent& 25 cent	339 cycles	9 cycles
2	6*5 cent	305 cycles	7 cycles
3	3*10 cent	161 cycles	4 cycles
4	5 cent & 25 cent	113 cycles	3 cycles

Table 8.3: Comparison of cycle time for both FSM implementations within the two application platforms; (PLD programme logic device; MMLS memory-mapped logic solution)

Table 8.3 is showing the maximum and minimum timing cases of the two applications and some in-between cases. The setting for the longest run through the state diagram is evaluated with case one because this case uses the most extreme coin input sequence with the most outputs. The fastest input-triggered sequence is case four reaching the 30 cents for the price of the soda which can be done with two coins. As indicated within the table the reaction time of the memory-mapped logic solution uses one cycle for each state transition. Where the PLD-based platform runtime varies, no direct timing ratio between these two applications can be found.

8.5. Comparison of different memory LUT concepts

The central component of the memory LUT-based FSM is illustrated in Figure 8.4 and this is the memory block. Memory is cheap and available in all kinds of diverse types and it can be distinguished in general between read/write and read-only memory. For both types in general the linear addressing for accessing the data is the common factor. Through the JK-FF and soda machine FSM examples it has been found that due to the linear addressing the transferring of the state transition table into memory LUT can create gaps in the memory-addressing structure. Due to the coding of the input stimulus for the vending machine example the memory-addressing structure has been designed from the transferred state table without address gaps. This addressing of the memory has been of linear form. This was possible through the use of coding the input information, which is not always possible for all the cases. The example of the soda machine only had some unused addresses of the memory addressing unallocated. These addresses would require a filter function at the state pointer for avoiding the generation of these addresses by mistake inside the state pointer. Another possible memory type which is useful for the memory LUT-FSM implementation is the content-related search system, which is part of the CAM technique [33].
The content-related search system is built out of two blocks with different tasks and is displayed in Figure 8.10 as one of different possibilities for this technology. The functional block on the left-side of the figure is the CAM block and on the right-side of the figure is the information memory. The CAM block in Figure 8.10 on the left is the memory used for finding the corresponding entry

of the search request within the CAM. The CAM block is in a way a big matrix of two input AND gates. One input of all these AND gates within one column is enabled with the one of the search bit information at the same time. The other input of each AND gate within this column is connected to a memory cell in which the content of each row is programmed into individual memory cells. After a match has been established within one row of the search memory of the CAM, the resultant address-pointer is generated for the accessing of the data within the information memory. This search match happens within one cycle due to the parallel information comparison within this AND gate matrix structure. The parallel search and match approach this the advantage of the CAM technology. The address-pointer from the CAM is used for reading out the relevant data out of the information memory. The information block can be a RAM or ROM circuit but in most cases it is RAM circuit for flexibility. Both memory types are of a linear addressing nature and not capable of searching inside of the whole data stored with a single cycle to find a match. CAM search pattern works in a way that in the case of not finding a 100% match within the data stored inside its matching matrix it looks for the close match. The close match is identified by only a single bit deviation and is going to be taken as a match. The corresponding information address-pointer will be used. For the correct state transition defined with the state diagram a deviation in terms of using the close data match cannot be done for the correct state transition of an FSM and has to be eliminated.

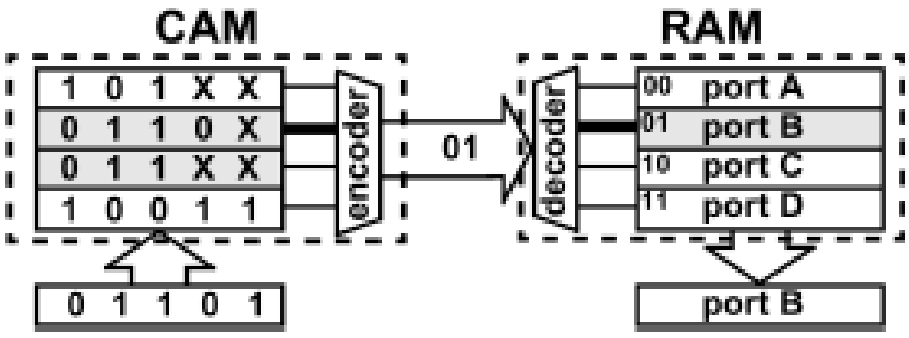


Figure 8.10: CAM based implementation of a content-related search system out of [33]

The internal structure of a CAM is an array of cells and the dimensions are a certain number of bits, which are used for the data word length and a certain number of rows of different data words. Each cell is capable of storing a single bit of a word and performs the comparison against the parallel-supplied search data word. Within one data word row of this cell matrix a match-line runs across all row cells connecting all these cells together for generating the signal data match at the match-line sensing amplifiers. This match-line-sensing amplifier converts an analogue signal into a digital

signal. The match-line signal is generated in the way that before a new search word is applied onto the cell array the match-line-sensing amplifier gets charged up into the state of data match found. During the phase of finding the match after the search data has been applied onto the cell array at each row which does not match, the match-line sensing gets discharged. This will only leave the one match-line with a match-indicating charge remaining [33]. In this way the CAM internal circuit structure is a combination of digital and analogue design and the creating of a fault-tolerant circuit design against SEUs can be done for the data-storing part. But not for the analogue part where SEUs could potentially discharge the match-line-sensing amplifier by direct hit of a particle. The protection of the stored data within the cells can be done with parity checking and this concept was proposed in [130].

8.5.1. Creating a fault-tolerant CAM circuit concept

The current structure of the CAM internal circuit is due to the combination of the digital and analogue circuit not being fault-tolerant by itself. Protection of the stored data within the CAM cell array can be done with parity checker concepts as proposed in paper [130], which is part of the CAM internal circuity. For creating a fault-tolerant CAM structure, the protection of the data is one concept. The protection of the CAM internal circuit that is performing the data matching and comparison finding, requires a different hardware solution to make it fault-tolerant. The CAM hardware alteration, which will achieve fault-tolerance, is shown in Figure 8.11 and this hardware alteration only uses digital circuity for data match finding and indication through address-pointer to the RAM. The change to the original CAM circuity is that the search cell matrix is replaced by a combination of programmable inverters per single data bit feeding into an AND gate. This AND gate is performing the data matching of supplied and stored information. Due to the parallel structure of this AND gate arrangement the match-and-find time is still one cycle similar to the original CAM design which also used parallel match-and-find hardware structures. This example selected as showing the fault-tolerant approach is displayed in Figure 8.11 and contains a four word content searchable memory block with the following search structure vertically by four bits per word horizontal. Each of these four bits forming the vertical word is feeding the output of the programmable inverters into a four-input AND gate and if all four-inputs are high signals, this will generate a high signal at its output. This high signal at the AND gate indicates that a match has been found in this vertical word row and the corresponding address-pointer will be generated for pointing onto the correct data inside the RAM block. The RAM block of this design example is of the same structure as the one of the original CAM and has not been altered.

The programmable inverter is a combination of a single memory cell, which controls, if the input signal feeding into the AND gate inputs, is inverted or not. The block diagram of the programmable

inverter is described in Figure 8.12 and will be used within the block diagram illustrated in Figure 8.11.

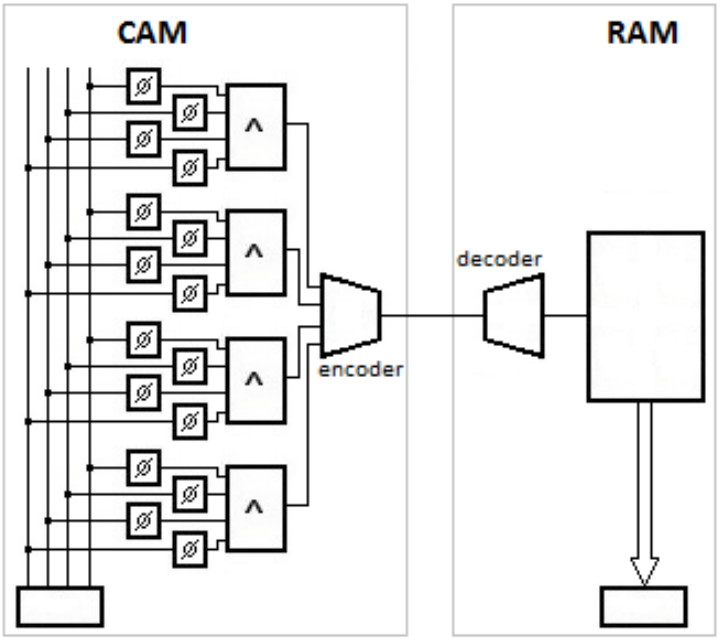


Figure 8.11: Fault-tolerant CAM block diagram

The function of this altered CAM circuit is by only using digital logic gates and for these the SAFR logic gates of Chapter 7 can be used. Internal faults related to stuck-at faults can be masked or clearly indicated if a non-maskable fault occurs with an increase in I_{ddq} current. Using the SAFR logic gates makes the altered CAM circuit fault-tolerant against hardware related faults of stuck-at fault nature. With regards to alteration of individual data information stored inside the memory elements due to SEUs, a different technique can be applied.

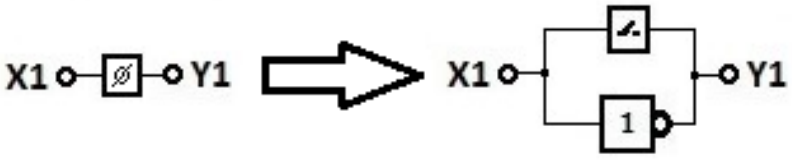


Figure 8.12: Block diagram of the programmable inverter

8.5.2. Protecting data memory inside CAMs against SEUs

The concept of the CAM method finding a data word structure within the entire memory data in one cycle is based around an array of memory SRAMs cells where comparison functionality has been added [130]. The central data storing part of an SRAM cell for storing one bit is done through a bi-stable latching circuit, which is sensitive against alpha particle or neutron hits. These are most commonly known as SEUs and in which case the stored bit data of this cell can be altered and a data error occurs. In the case of SRAMs, which are used as memory for a computer system, a parity check bit per data word reveals the alteration of a single data bit within a defined data word. Because of SEUs happening within a computer system some systems are performing for each read data word from the SRAM, a check of the parity bit and if necessary correct the fault before sending it to the requesting circuit unit. This check of the data can only happen when the specific data word in memory is being accessed, otherwise any altered data bits within the memory are dormant faults and accumulative adding faults in the event of constant SEUs is taking place within the memory. In this case it is possible that more than one data alteration can happen within one data word and for this case the parity check would be ok or cannot correct the fault. In general a process like constant data scrubbing would help in this case. But for this a copy of the data has to be stored somewhere else for write-back or a constant parity check of the whole memory regardless of the currently accessed data has to take place.

Within paper [130] it was proposed to add parity bits to the row data word as part of the data word match-and-find process. In this paper a Hamming code with four parity bits is being proposed with results in nine added parity CAM cells per word. This type of Hamming code is an adaptation of the extended Hamming (256, 247, 4). The 256 represents the total number of bits, whilst the 247 defines the total number of data bits, which are being protected by a certain number of parity bits. The four indicates the total number of parity bits and in this case is done with nine data bits of the overall number of bits. Due to the fact that the parity bits are part of the search data word the match-and-find sequence requires that these parity bits are identical to the one stored inside the CAM. A one-bit miss is still considered a match in accordance to the paper [130] if this missing bit is part of the error-correcting code. This check can only be performed after the row data of the CAM would have been read and the parity check can be generated. But this is not part of the concept proposed in [130] and a one-bit mismatch is the general feature of the CAM method identified in [33].

The error-correcting feature proposed as research work for this thesis is targeted at eliminating single data-bit alteration within the search memory array and is displayed in Figure 8.13. This concept is based on the same concept of utilisation of parity bits as the one proposed in [130] but with the difference of only using a single parity bit per horizontal and vertical data arrangement of the memory. By altering the parity-bit orientation towards a cross matrix this concept is capable of

identifying the bit position within the matrix through the overlay of horizontal and vertical parity checks. Through a simple reverse of the data of this specific cell the fault can be fixed. The parity check of the CAM cell matrix can be done through a constant evaluation of the data stored inside the CAM matrix, which will require some hardware overhead to it. This hardware overhead is due to the permanent generation of the parity bit for this arrangement. In the case of the requirement of reduced hardware overhead this can be achieved for the price of delayed altered bit identification with the help of a MUX switching for each of the vertical and horizontal rows, which can switch through them for performing parity bit generation. This parity-bit can then be compared with the required parity bit for this matrix arrangement. In case of a mismatch a correction of this fault can be achieved by inverting the data bit at the parity checked cross section for fault correction. All these fault tolerant features applied to the hardware used by a FSM implementation will create a fault tolerant FSM platform with inherent fault detection features. The constant evaluation of the data stored within the altered CAM structure detects and corrects data which is affected by SEU within the memory and the use of SAFR logic gates covers the stuck-at type faults.

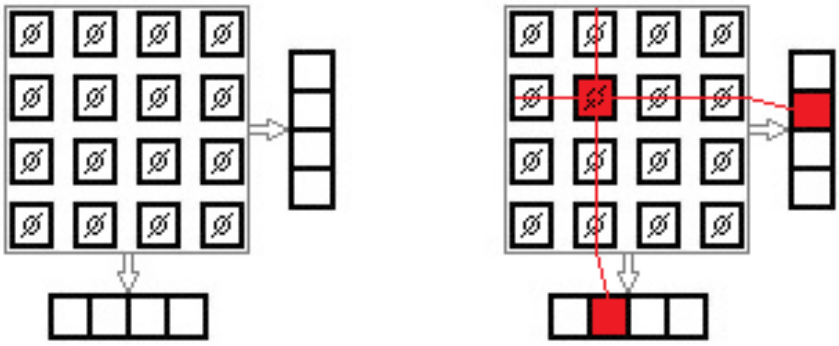


Figure 8.13: Concept of identification of a single data-bit alteration within a stored data matrix of a CAM circuit

8.6. Summary of the chapter

This chapter was focused on resolving the problem of creating a minimal hardware requiring FSM platform with fault-tolerant features. Fault-tolerance should be achieved with minimal hardware overhead and must not impact the performance of the FSM.

The solution for this problem has been achieved by mapping FSM functionality into a memory-only-based system platform. Application specific behavioural transitions can be broken down into a sequence of individual steps and transferred into a step or state flow in tabular form.

The functionality of an FSM works on the concept of having only one state active at a time together with state transitions triggered by input stimuli. If required, the outputting of data may be performed during state transition. Current FSM applications are based on fixed hardware circuits or programmed into PLD systems. The structure of the FSM state table makes it possible to transfer the behaviour into a memory-based platform wherein the functionality of the approach operates by utilising the memory address as a combination of state and input data that represents the FSM state and state transition triggers. Utilising the memory-only platform for the FSM design also requires an address-pointer, which generates the next memory location to be read out of the combination of input and current state transition data.

Comparison with a PLD-based system revealed that the memory-mapped approach displays a constant response time for each state transition and requires less memory. It may be argued that a potential drawback of the memory-only platform is that stored data in memory is not necessarily linear in nature and contains data gaps that may compromise the integrity of the FSM. Research performed on the memory structure revealed that, by using a data coding approach, the data structure within the memory may be manipulated into gap-free data structures. This FSM state transition data coding approach has the combined benefits of memory structure compaction and requiring less memory. The next step towards further memory compaction was arrived at through the utilisation of CAM hardware. Within this modified memory-only structure the required data can be stored within this memory and any possible remaining data gaps are eliminated.

In summary, a novel design of content access memory has been designed by a combination of SAFR-type logic gates and CAM-type memory implementation. Utilisation of SAFR-type logic gates adds the property of underlying, intrinsic fault-tolerance and discrimination against stuck-at fault-types (as discussed in Chapter 7) while the CAM implementation increases the integrity of the next state transition.

Chapter 9: Design of self-healing logic structure

9.1. Introduction

The concept of self-healing in electronic systems is the attempt of the system designers to mimic the capability nature has given animals and even humans to be gifted by fixing, for example, injuries that are happening to their bodies. An example of self-healing performed in animals or humans is the capability in the event of an injury to heal or repair a cut through the skin without external intervention. Self-healing in nature is performed through intrinsic capabilities of the body and it is always performed without external intervention of any kind. The underlying and important feature of self-healing in nature is built on self-diagnosis of an incident on any part of the body tissue that has made an impact requiring an inherent healing feature. Also the mechanism of healing is performed by means of moving material to the place of need or working with the material at the location where the healing needs to take place. There are cases where electronic systems are becoming part of autonomous systems where the key feature is put into maintaining system functionality as long as possible even in the event of a fault within a part of the circuit [131]. This can be achieved through giving the electronic circuit design if done within reconfigurable FPGAs that can alter their circuit layout and structure to exclude a faulty region or activate redundant structures. Both of these features require an external or built-in checker system for detecting, locating and repairing. This reliance upon external features defeats the idea of intrinsically reacting self-healing systems. The key feature of self-healing within any electronic system with this feature relies on inherent self-diagnosis of the occurrences of faults within the system hardware similar to biological systems.

System designers seek self-healing capabilities to be incorporated as part of the system design hence creating fault-tolerant or self-maintaining systems that prolong their operational life-time. Compared with the capability of nature, where these capabilities are intrinsically part of the natural system, man-made systems require external logic circuitry to detect abnormal output values or irregular system responses. System-checkers currently take the form of a common design, which is created either on the same chip or else using a second chip that is subjected to the same failure mechanism as the logic circuit that is being protected. However protection of the system-checker itself will lead to even more complex circuit designs that will become even harder to protect. As a result, the ideal self-healing circuit would incorporate all necessary redundancy logic structures and self-checking mechanisms as part of their intrinsic system design.

This chapter is focused on answering the three fundamental questions of how self-healing properties can be achieved within electronic circuits. The first research question was left unanswered from Chapter 7 regarding the utilisation of intrinsic self-diagnosis of the manifestation of stuck-at faults within a SAFR-logic gate and the initiation of circuit alteration without the

influence of an external logic circuitry. These aspects are addressed below with the help of an example circuit capable of reconfiguration at runtime in case of a stuck-at fault without an impact on the operational performance. The second research question is centred on the self-healing feature of the QLC element, which is capable of determining the location of faults at the logic unit level. The third research question relates to incorporating FSM capable of fault location and repair within logic designs, thus becoming part of the overall self-healing strategy. BIST and BISR functionality is combined to achieve fault-localisation with FSM logic structures.

9.2. A self-healing fine grained logic structure

The concepts of self-healing within an electronic logic circuit in comparison to the features nature offers for self-healing are not directly comparable. Out of this perspective the concept of self-healing within any man-made system would require an intrinsically built-in capability for fault detection and a means of fixing faulty components through circuit manipulation at fine-grained level to be a self-healing system. Due to this a concept of self-healing within any electronic system can be seen as an umbrella statement for other self-* capabilities such as self-diagnostic, self-detection and self-reconfiguration. For example, in nature the axolotl [34] (see Figure 9.1) (known as Mexican salamander or ambystoma mexicanum) is able to perform epimorphosis on the regeneration of limbs, organs, parts of the non-vital regions of the brain and heart. The axolotl can do all of this without any external help or having redundant body parts waiting to be used in the case of a fault. The key enabling feature present in the case is that of self-diagnosis that triggers self-healing when required.

Self-healing inherently present in nature is necessary in electronic systems and it is needed to be adapted into any complex electronic system requiring the capability of prolonging operational life-time. Present-day electronic systems do not offer sufficient built-in self-diagnostics as part of the inherent circuit fault detection capabilities. Whenever these systems are affected by the occurrence of a fault any detection is done through limited fixed resources of this logic circuitry. Self-healing is focused on prolonging the uptime of the system through reconfiguration and selective exclusion of hardware parts of the system that require enhanced self-diagnostics. But in contrast to any natural system, every known instance of self-healing electronic systems requires strategically placed spare hardware structures within the overall design that can be used for replacement. Another disadvantage of man-made systems versus natural approaches is that the approach of self-healing requires self-limiting or inhibiting of self-healing action within an electronic system utilising self-healing. Additional requirements for the self-healing strategy require capabilities for assessing the success of healing if the functionality has been restored to the level prior to the occurrence of the fault. Evaluation of the remaining resources is also required for the establishment

if further self-healing attempts are to be made. All these requirements are needed within a man-made adaption of a self-healing concept within an electronic system.

A central tenet of this thesis is focused on empowering logic gates with the capability of an inherently built-in indication of the occurrence of a fault within its logic gate structure. For an electronic-based system done, for example, within an FPGA, the task of self-healing is mostly done by means of reconfiguration performed with the help of pre-compiled configuration logic structure data. Furthermore, reconfiguration of the FPGA is done by an external device and it is not part of the internal logic configured within the target FPGA. The reconfiguration of an electronic system is therefore limited to coarse-grained logic resources such as interconnect fabric and CLBs and cannot influence the necessary fine-grained logic resources. The function of the CLB and QLC is to perform the required logic operation for the application and in this regards a direct replacement within a FPGA circuit structure would be possible. This replacement can be done in a way that a single QLC is put in place of a CLB and this replacement defines the level of scaling achievable with the QLC design. The use of single QLCs maintains the features of fault detection within the QLC and the self-initiation of fine-grained self-healing features. For the self-healing of a QLC, local interconnection reconfiguration is required. This is provided through the FPGA interconnection design, which can be maintained or needs expansion for providing the required interconnection structure between individual QLCs.

Within any electronic system structure the system designer needs to predict possible fault conditions within parts of the application circuit structure or else use a large scale modular redundant logic design of the main logic structure. Unfortunately, the resource cost for this case is very high in order to mitigate for single errors. Figure 5.5 of Chapter 5 represents one possible example where a predefined spare column within a set of columns will be used in the case of a fault within another column as a replacement. This concept of using a single column within a set of other columns as a replacement is proposed within [23] as a concept of self-repair. In a sense, self-repair is part of self-healing because a predefined replacement is used in case of a fault. For performing self-repair the logic system is required to have resources for fault detection which are either built-in or by external means. Otherwise the logic circuit would need resources of intrinsically built-in capability of fault detection on which it can react.

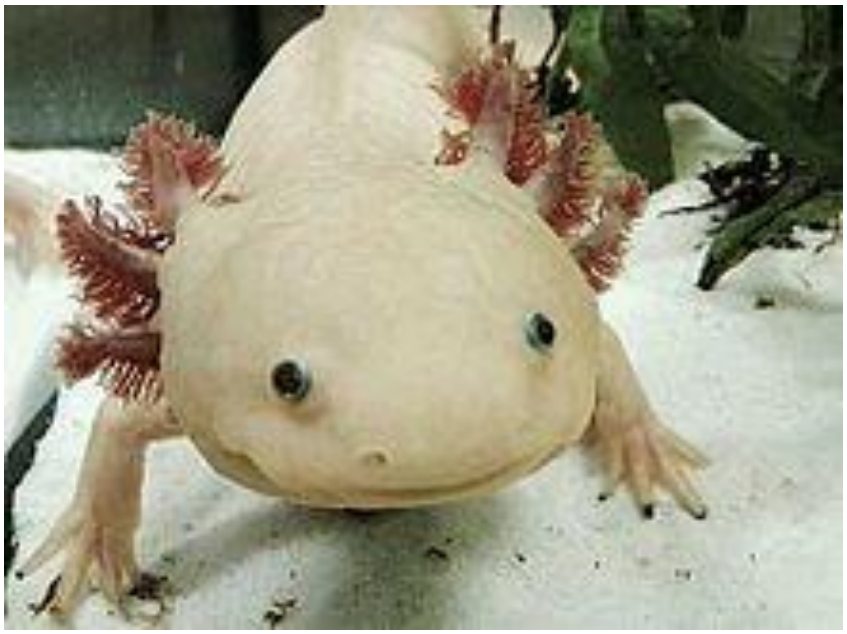


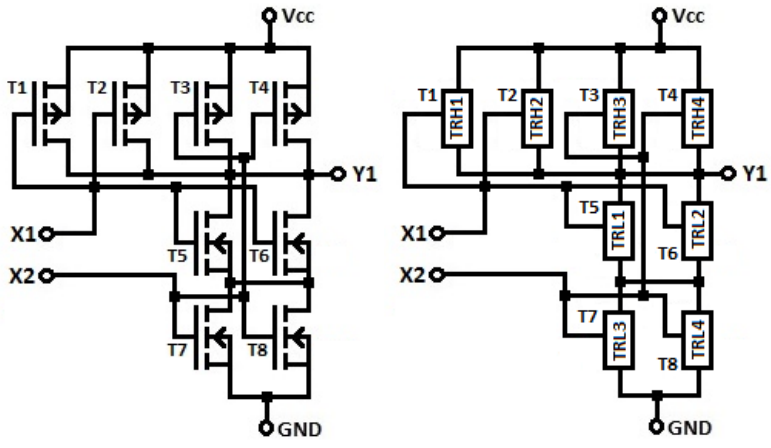
Figure 9.1: Axolotl (ambystoma mexicanum) [34]

9.2.1. Concepts for fault self-detection with the SAFR-NAND gate

The most commonly used fault-tolerant concept in logic systems, which requires fault-masking and/or fault-tolerance is based on the NMR system with majority voter. A generic block diagram of this type of logic system is shown in Figure 4.6 and it is based on the work proposed by von Neumann [99]. The most commonly used adaptation of an NMR-based system structure is the TMR system with majority voter. A simple example of this type of fault-tolerant system is described in Figure 4.7. For creating a fault-tolerant and self-repairing logic circuit the majority voter of the TMR system needs to be expanded with the capability of information comparator and its design would require it to be fault-tolerant by itself. The concept of information assessment at the majority-voted output signal requires having a voted output signal feed into a comparison circuit in which this signal is individual when compared with the output signals of each redundant module of the TMR system. A simple example of this concept is illustrated in Figure 4.10 where it is used for the indication of the incorrect TMR module path. This indication signal can also be used for triggering a reconfiguration of the TMR system if it is running on a runtime reconfigurable platform like an FPGA, for example, and predefined reconfiguration data is available within an external device. Different reconfiguration concepts are available to perform this task and each of them has been designed for a certain concept and purpose. Reconfiguration is an action responding on identifying fault hardware down to a pre-defined logic block. The key for doing so is fault-localisation within a given logic circuit down to a gate level, in best case down to the single faulty component of this logic system. In this case the fault-tolerant design of the system is based upon

fine-grained redundancy structures. Fault-localisation performed down into this kind of detail can only be done with significant hardware overhead and test time.

In chapter 7.2.3 the design of a SAFR-NAND gate has been illustrated and the outcome is displayed in Figure 7.6. The evaluated SAFR-NAND logic gate design has the inherent feature of being able to mask single SAL faults and indicating non-maskable SAH faults through an increase of the I_{ddq} current. This increase of the I_{ddq} current is a unique indicator of a non-maskable single SAH fault within the gate. The current increase can also be used as an intrinsically built-in feature of the SAFR-NAND gate to indicate the need for repair or healing action. To evaluate the increase of the I_{ddq} current value during a non-maskable SAH fault within the SAFR-NAND gate, the transistor design of the gate is transferred into the resistor-based model for evaluation. This transfer from transistor to resistor-based model is shown in Figure 9.2. The resistor-based model of the SAFR-NAND gate is defined in Figure 9.2(b) and the resistor replacement of a transistor is a voltage-controlled switch between two resistor values, which are the two states of R_{DSOFF} and R_{DSON} of the associated transistor. These two state approaches of the transistor are only working for the digital stimulation when the transistor is either switched off and replaceable by R_{DSOFF} or switched on meaning R_{DSON} is active. For the simulation and calculation of the I_{ddq} current verification the following transistors have been used: the transistors for the pull-up side or network (labelled with TRH1..4) are BSP230 and for the pull-down side or network (labelled with TRL1..4) are BSP126. For both transistors of the pull-up and pull-down network the R_{DSOFF} is defined as 10 M Ohm within the device specification. For the R_{DSON} for the pull-up network transistor, which is the BSP230 transistor, the resistor value is defined as 17 Ohm taken out of [132]. The R_{DSON} of the transistor transfer of the pull-down network transistor-,which is the BSP126 transistor, is defined as 5 Ohm in accordance with [133].



(a) (b)

Figure 9.2: (a) Internal transistor structure of SAFR-NAND gate; (b) SAFR-NAND gate converted from transistor into variable resistors structure

For the evaluation of the I_{ddq} current value during an increase caused by a single SAH fault at a single transistor of the SAFR-NAND gate a single circumstance out of Table 7.4 has been selected which is the transistor T6 under the influence of an SAH fault (see Figure 9.2(a)). For this case the I_{ddq} current increase is going to happen during the input sequence X1=0 and X2=1, which has been taken out of Table 7.4. The T6 or TRL2 represents the transistor with the SAH fault within the pull-down network of the SAFR-NAND gate. Due to this specific input sequence the following transistors are being turned on and because of its identical resistor value the following simplification can be defined: the values for TRH1 and TRH2 are identical and R_{TRHON} is used instead and the same applies for R_{TRLON}. The overall resistor value of the resistor replacement estimation of the SAFR-NAND gate can be calculated as follows:

$$R_{Com} = \frac{R_{TRHON}^2}{2*R_{TRHON}} + R_{TRLSAH} + \frac{R_{TRLON}^2}{2*R_{TRLON}} \qquad \text{(Equation 9.1.)}$$

$$I_{ddq} = {}^{V_{cc}}/_{R_{Com}} \qquad \text{(Equation 9.2.)}$$

The I_{ddq} current for this example with V_{cc}=5V is a theoretical I_{ddq} current of 312.5 mA but the maximum current capability of the BSP230 transistor is in accordance with the datasheet 210mA [132]. The transistor maximum current capability will limit the maximum current flow within the SAFR-NAND gate within this arrangement in case of a short circuit path between V_{cc} and GND. Another point has to be made about the level which the I_{ddq} current can reach and this is that it is the highest possible current flow within the logic gate. This current path inside the logic gate is a result of this fault condition and it is a constant flow over time. This current flow can cause damage to local chip structure. Because of this danger precautions for the current presences within a given chip have to be taken to avoid permanent chip faults.

A spice simulation where the SAFR-NAND gate is using these specified transistors was performed to verify the I_{ddq} current value for the fault present indication which exists within the SAFR-logic gate circuit structure. The spice simulation result is outlined in Figure 9.3.

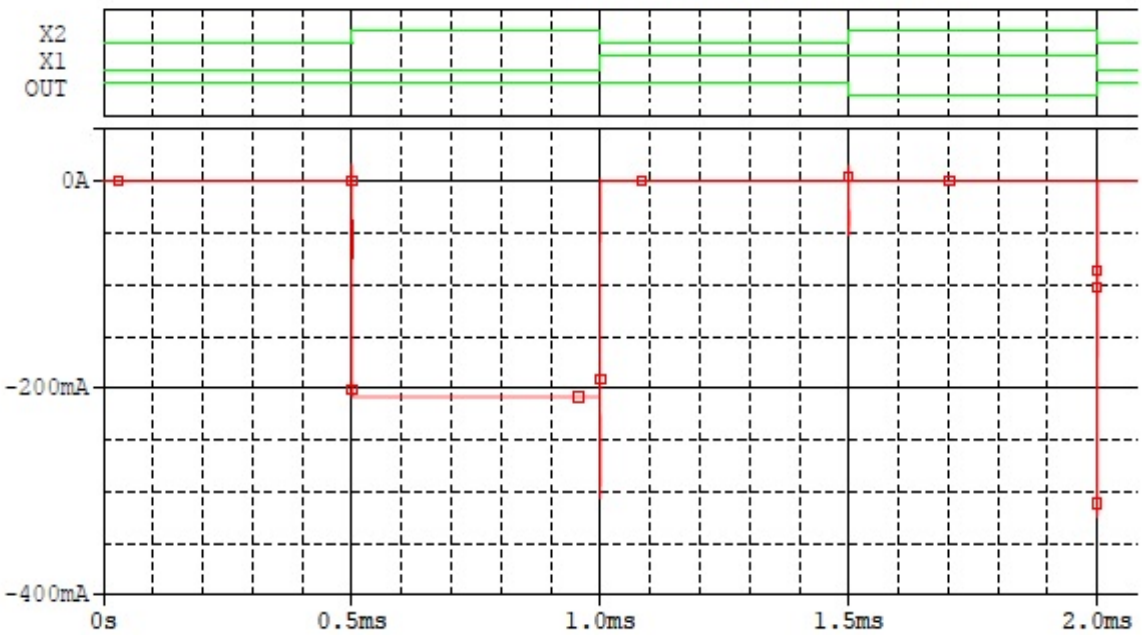


Figure 9.3: Current response of the SAFR-NAND gate with the presence of a single SAH fault at T6 transistor (see Figure 9.2(a)) and required input stimulus

The spice simulation results, which are displayed in Figure 9.3, are clearly showing the presence of a single SAH fault within the circuit structure of the SAFR-NAND gate indicated through the increase of the I_{ddq} current. This fault is triggered by the input stimulus X1=0 and X2=1. These are the correct input stimuli for creating the short circuit path between V_{cc} and GND as defined within Table 7.4. The value of the I_{ddq} current is -210mA during the time the correct input stimulus is applied to the SAFR-NAND gate. The value of the I_{ddq} current is the same value as the saturation current of the BSP230 transistors. In this way the calculation of the current level matches the spice simulation hardware constellation specified within the transistor specifications. Due to the presence of the single SAH fault at T6 of the SAFR-NAND gate structure this type of fault condition is out of the range where the output value is correct and only the increased I_{ddq} current indicates the existence of the fault. The arrangement of the transistor structure within the SAFR-NAND gate is able to mask a single SAH fault at the transistors T5 to T8 of the SAFR-NAND gate (see Figure 9.2(a)) for a set of input stimulations and for the input stimulation defined within Table 7.4 it is indicating this condition through the increased I_{ddq} current.

9.2.2. Initiation of self-healing of a circuit designed out of SAFR-NAND gates

The initiation of self-healing or self-repairing after a permanent fault presented within the SAFR-NAND gate requires a clear and distinguished signal to the outside of the gate structure. With the I_{ddq} current increase in the case of a single SAH fault affecting one of the SAFR-NAND gate transistors this is a suitable indicator, which has been designed into the gate structure. A possible method of introducing self-healing could be the use of a burning open fine-fuse within the different transistor paths of the SAFR-NAND gate. This fuse would disconnect the affected transistor and the remaining redundancy transistor maintains a working logic gate. The level of the I_{ddq} current increase should be in the region of using a burning open fine-fuse for permanently interrupting the connection of an individual transistor out of the logic gate structure. The problem of using this type of disconnecting fuse within the internal logic gate structure is the component space required on a possible chip. By creating a disconnection fuse on a chip through fine metal traces the reliability of the consistency of the required reacting current could be a production challenge. Also the required component space needed on the chip area would be significant.

The disconnection of a transistor by fine-fuse requires the correct current over time. The time it takes for the fine-fuse to react on the current at which level it should disconnect is according to [134] a problem, which makes the introduction of conventional fine-fuses within the SAFR-NAND gate obsolete. The I_{ddq} current increase during the occurrence of a single SAH fault-injection should be enough to disconnect an installed 50mA fine-fuse. This particular value has been chosen because with the selected transistor types the I_{ddq} is going to be around -210mA. The ratio between these two values is four times. The datasheet specifies for a 400% ampere rating compared to the face value of a certain response time and in this case the reaction time can be between 3ms and 300ms. This makes this technology inappropriate to use within this type of application.

Another disconnecting fusing technology could be used instead of conventional burning open fine-fuses, which can be the memristor proposed in [135], with its application-specific definition of its breakdown behaviour in accordance with [136]. The basic principle of a memristor is that it can switch between two resistor values triggered through the influence of a certain current level passing through the memristor. Certain current levels are associated with each one of the two possible inherent resistor values of the memristor. So it can be used as a low and high impedance current triggered switch. The functionality of the memristor is similar to the function of a fuse. The important difference between these two fuses is that the disconnecting fine-fuse can only react on a current with a certain level for one time only and the memristor can be used as a switch. The alteration between both resistor values happens within a specified time frame and this timing parameter is the key parameter to make this technology usable for the requirements of the SAFR-NAND gate. Research work on finding the right parameters for the memristor is beyond the scope of this thesis and cannot be worked on at this time. But the memristor could be easily integrated

within a chip design as part of the silicon versus having conventional burn-through fuses. The dimensions of the memristor would fit into an interconnection via within the silicon structure of the chip and if a non-permanent disconnection of a single transistor of a SAFR type gate is needed it could be selected as a design solution or chip component.

A third possible fusing technology usable for the decommissioning of a faulty individual transistor within an SAFR-NAND gate due to a single SAH fault could be the eFUSE technique proposed in [137]. An application specific solution regarding autonomic hardware self-healing was proposed in [138] where a general concept of using eFUSE technology for switching in redundant autonomist chips has been proposed. This fuse technology uses the electro migration capability within silicide polysilicon, which created an electronically writable chip element [137]. The initial resistor value for the eFUSE has been indicated in the paper of 120 ohms, which only allows the placement directly in the supply or ground connection of the SAFR-NAND gate. The noticeable disadvantage of adding this resistor value of around 240 ohms to the I_{ddq} current-creating path will reduce the current flow significantly. In the example beforehand the resistor value was theoretically 16 ohms creating 312.5 mA and with 240 ohms added only 19.5 mA. By placing the eFUSE into supply and ground connection and in the case of a single transistor fault within the gate the entire SAFR-NAND gate is going to be disconnected. Because of the disconnection of the entire SAFR-NAND gate within a given logic circuit detection logic has to be developed for this case. This detection logic would lead to hardware overhead and an entire SAFR-NAND gate is needed to replace all the faulty ones. The usage of the eFUSE inline within the supply lines of a logic structure has not been proposed and researched within these papers. It could be possible that the eFUSE technology cannot be used in this regards. The normal use of the eFUSE element is for storing non changeable information within a chip. The eFUSE capability of supplying a logic structure would require research to see the limitations and capabilities for this specific application. It is the writer's perspective that this is beyond the capabilities and scope of this thesis, but the technology of eFUSE could lead to usable logic structures, which also could be included within a chip during the normal design phase.

All proposed solutions to disconnect a single faulty transistor or the entire SAFR logic gate require detection logic for identifying the occurrence of this event. Without the detection the remaining fault-absorbing capacity of the logic circuit cannot be accounted for.

The solution, which is going to be applied within this thesis in terms of identifying the I_{ddq} current increase after a non-maskable single SAH fault affecting a transistor within the SAFR-NAND gate, is a current sensing by means of a current shunt. The voltage drop across the current shunt is in conjunction with the current level. This voltage drop is changed by a signal convertor into a single digital signal, which can trigger possible selective deactivation or reconfiguration within a logic system. This concept is more a current sensing followed by converting than a current measurement

with built-in level detection. It will be referred to as current sensing and conversion fuse (ccfuse) within this thesis. A block diagram of the ccfuse concept is illustrated in Figure 9.4.

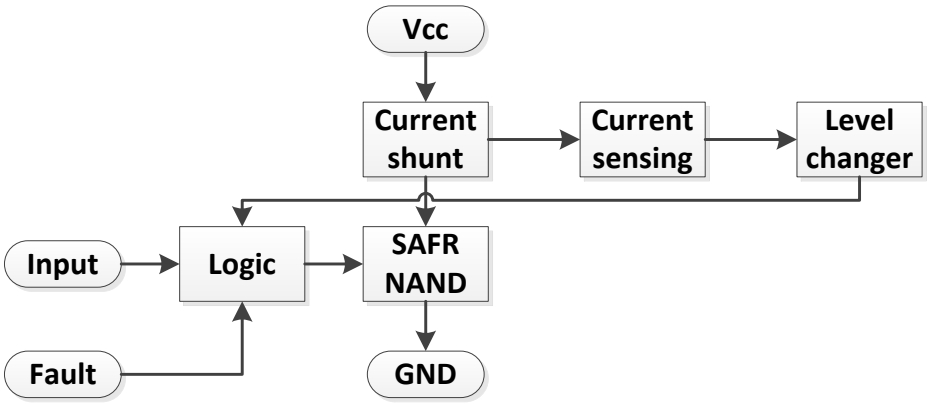


Figure 9.4: Block diagram of the SAFR-NAND gate simulating a SAH fault-injection and ccfuse fault-clearing capability

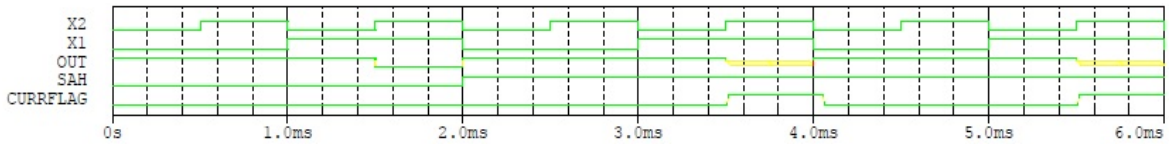
The self-healing approach proposed with the block diagram illustrated in Figure 9.4 needs to be evaluated within a spice simulation to measure the timing of the self-healing feature. For this evaluation the faulty transistor of the SAFR-NAND gate needs to alter the output value of the gate. The alteration of the output value into a faulty output is required to evaluate the capability of the self-healing process in terms of how quick the output value is reflecting the correct value again and with regard to the timing of the self-healing. In accordance with Table 7.4 the transistors T1 to T4 (see Figure 9.2(a)) are showing the required impact of fault-behaviour. For this evaluation the transistor T3 of the SAFR-NAND gate has been selected. The output value of the SAFR-NAND gate will indicate an undefined output value during the presence of a single SAH fault at T3 of the logic gate in conjunction with the input stimulus of X1=1 and X2=1. At the same time the I_{ddq} current will increase for the indication of the presence of a non-maskable fault within the logic gate.

The block of Figure 9.4 with the label *logic* is representing the logic structure where the input sequence gets altered for creating a permanent SAH condition of a specific transistor during simulation. In this case it is the T3 transistor. This logic block is capable on request to clear or add the presence of any type of simulated stuck-at faults at a certain logic gate transistor within the logic equations. By doing it this way the hardware overhead and complexity around each individual transistor of the SAFR-NAND gate is reduced.

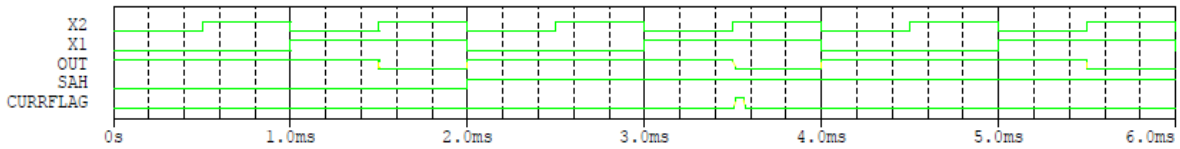
Figure 9.5(a) shows the spice simulation of the block diagram of Figure 9.4 with the simulation of an injected single SAH fault at T3 at the SAFR-NAND gate. For this spice simulation the self-healing capability specified within the block diagram has not been activated; only the presence of the current increase is active to show the I_{ddq} current increase. The increased I_{ddq} current is shown

through the currflag within the spice simulation data. The SAH fault happens at the 2ms marker of this spice simulation. The SAFR-NAND gate response caused by this fault is in accordance with the condition specified in Table 7.4 and simulation response can be seen in the time frame 3.5ms to 4ms for the input stimulus X1=1 and X2=1. The output value for this time frame is indicated through the spice simulation as not being able to indicate an appropriate digital logic level for this time frame. This means that the output value for this time frame is undefined. This undefined output value caused by the required input stimulus and the presence of a single SAH fault within this SAFR-NAND logic gate remains active until the SAH fault or IC is cleared or altered.

In Figure 9.5(b) the same spice simulation as for Figure 9.5(a) is re-simulated but at this time the self-healing features of the circuit are active. As previously, the single SAH fault is injected into T3 of the SAFR-NAND logic gate and this is happening at the same time marker of 2ms. The required input stimulus is happening at 3.5ms and at this time the fault shows up within the spice simulation data with this time stamp. This is similar so far to the simulation data shown in Figure 9.5(a) without self-healing being active. At the time marker of 3.5ms in Figure 9.5(b) for the circuit with self-healing capability the functionality can also be seen in Figure 9.6. In this figure a detailed simulation plot is shown for the time frame 3.2ms to 3.8ms of the simulation data taken from Figure 9.5(b).



(a)



(b)

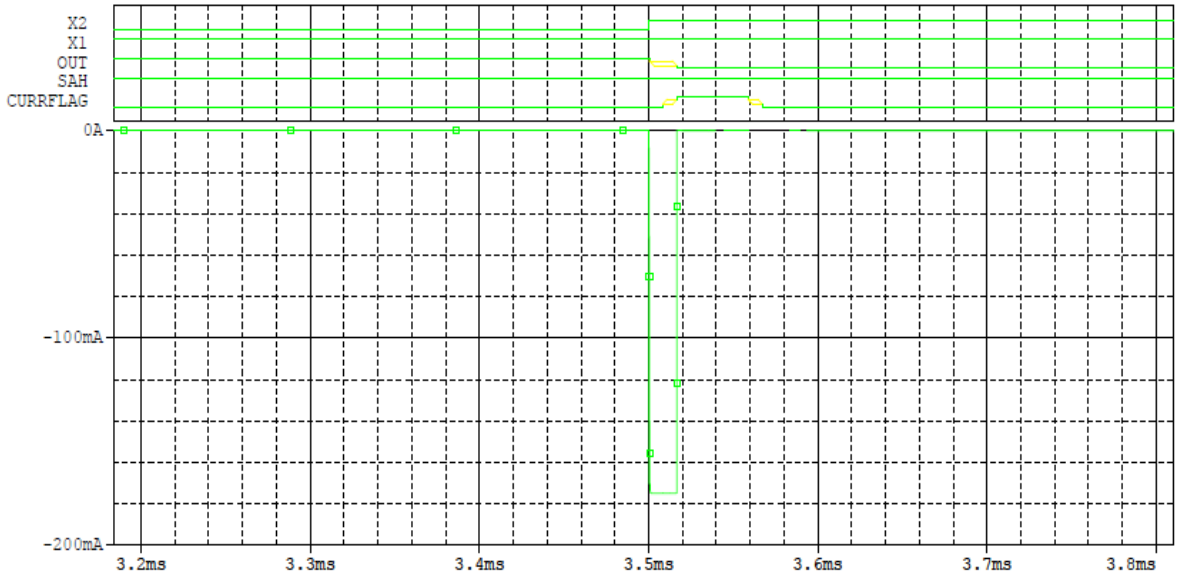
Figure 9.5: (a) SAFR-NAND gate with SAH fault at T3 without self-healing capabilities; (b) the same condition as in (a) including this time self-healing capabilities for fault correction

Within Figure 9.6 a higher resolution plot of the simulation is shown of a particular time frame of the data taken from Figure 9.5(b). This time plot illustrates the behaviour of the self-healing capability that is described by the block diagram illustrated in Figure 9.4. At the time of 3.5ms both X1 and X2 are high, which are creating the critical input sequence triggering the fault identification and this will cause the SAH fault to affect T3 of the logic gate to alter the output value into undefined. At the same time the I_{ddq} current increases as the indicating signal of the non-maskable

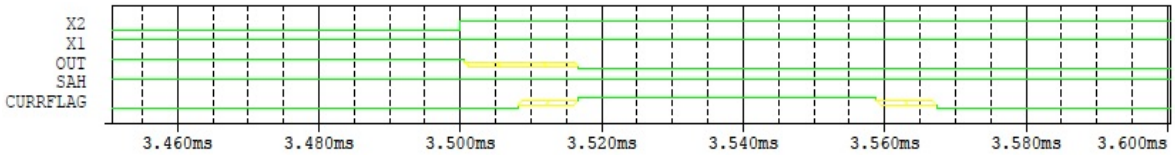
fault affecting the logic gate. From this time stamp onwards the fault-tolerance of the logic gate in conjunction with the ccfuse is required and the timing plot shows the flow of the events. After the critical input stimulus is applied at the inputs of the logic gate, its output response under the influence of the SAH fault is an undefined output condition. This is indicated through the yellow line within the *out* signal of the plot illustrated in Figure 9.6. This output condition lasts until 3.58ms until the ccfuse circuit, which is combining analogue to digital current-based conversions, is applied onto the I_{ddq} current signal of the logic gate. The ccfuse circuit is transforming the I_{ddq} current signal into a digital signal. The I_{ddq} current increase for this fault case is defined by the equation 9.2 up to the maximum current-carrying capability of the transistor from the time 3.5ms onwards. The ccfuse circuit identifies this current from the time 3.5ms onwards and until the time of 3.55ms the current to digital signal transforming circuit holds the *currflag* at zero within the simulation data. This is due to the internal signal runtime within the ccfuse circuit. After this time of 3.55ms the *currflag* output is in the undefined logic state identifiable through the yellow line within the plot until the time stamp of 3.58ms and at this time point a clear high signal is present for the *currflag* within the simulation data. This high signal at the *currflag* output of the ccfuse circuit triggers the self-healing of the SAFR-NAND gate and takes place immediately by deactivation of the presence of the SAH fault influence on T3 of this logic gate. This deactivation is simulating a decommissioning of the faulty transistor T3 of the SAFR-NAND gate. This decommissioning could be a burn open fine-fuse component, which is part of the logic gate circuit, as an example without the addressed problems described beforehand. As described beforehand there are different components usable for decommissioning a faulty transistor. This definition and integration into the SAFR-logic gate is undefined at the moment of this research work and this is because different possibilities for a fuse device have been investigated and all require prolonged research work, which is beyond the timeline of this thesis. The fault presence deactivation shows the effect within the logic gate in changing the output signal *out* into a defined logic state of zero. This is the correct output value for the current input stimulus of X1=1 and X2=1. The key concept of this simulation is that the decommissioning of the faulty transistor within the SAFR logic gate takes place without the use of an external checker system.

The desired circuit effect was the decommissioning of the fault-causing transistor of the logic gate and in this case the decommissioning of transistor T3 of the SAFR-NAND gate. The deactivation of the influence of the single SAH fault on T3 of the logic gate alters the output into the correct state, which also causes the I_{ddq} current to decrease immediately to zero current flow. This change of the current flow can be seen in Figure 9.6(a) bottom graph, which is showing the current flow through the SAFR-NAND gate. At the time of 3.9ms the current is zero, which is normal for this type of circuit. This triggering of the self-healing is latched inside a flip-flop and maintains the deactivation of fault influences on the T3 transistor of the logic gate for the duration of the

simulation onwards. It can be seen in Figure 9.5(b) that at time 5.5ms no incorrect output value at *out* is present and no I_{ddq} current increase triggers the *currflag* signal.



(a)



(b)

Figure 9.6: (a) Detailed time slot taken out of Figure 9.5b of the simulation of a single SAH fault at T3 of an SAFR-NAND gate with self-healing capabilities for fault correction; (b) Higher time frame resolution of the digital signals of the self-healing phase

Within Figure 9.6(b) the response timing of the self-healing feature is outlined and the present limits can be observed. The response time for the self-healing circuit used at this time within this research is around 16.5µs. This time frame is the time difference between the presence of the fault-causing IC applied onto the logic gate and in this simulation it is 3.5ms. The 3.5ms is the zero point or t0 for estimation of the time difference of fault-condition triggering through IC and logic gate responses. The time of the logic gate of 16.5µs is mainly due to the current-converting circuit design, which is the standard positive supply rail current-sensing circuit design taken out of [35]. This circuit is shown in Figure 9.7 and converts the current through a current shunt into a signal between GND and Vcc.

The converting capability can be calculated with the following equation:

$$V_O = I_{Load} \cdot R_S \left({}^{R_2}/_{R_1} \right) \qquad \text{(Equation 9.3) [35]}$$

The response time of the detection of the I_{ddq} current increase by the current sensing circuit is important for the performance of the self-healing feature for this logic gate. For the measurement the corresponding voltage graph of the output voltage of the supply rail current-sensing circuit in relation to the digital signals is outlined in Figure 9.8. The time marked with t1 represents the time point where the voltage level has been reached for the logic gate to alter its output value into a high value. This voltage level is within this spice simulation defined at 2V and causes the current detection circuit response time to be the 16.5µs, which can be identified within the graph of this simulation. Compared with the standard specification of CMOS 5V technology logic this voltage level for identifying a high level at its input is ≥3.5V. Applying this voltage level onto the output voltage graph displayed in Figure 9.8 of the current-sensing graph would push the switch to a definite high signal to 28µs identified with t3. The delta between both times t1 and t3 is 11.5µs or 41.07% difference. This time difference between the spice simulation and an actual circuit is of significance and cannot be neglected. The time where the current flag indication switches back to zero is labelled with t2 at the same voltage level of 2V for this spice simulation. The standard specification of CMOS 5V technology defines the voltage level for the input at a voltage level of ≤1.5V. Applying this voltage level at the current-sensing graph of Figure 9.8 the time marked with t4 is 63µs. The delta between t2 and t4 is 4µs or 6.78%. Overall, the performance of a real circuit would be different to the spice simulation in terms of self-healing response time and further work should be focused on the supply rail current-sensing solution regarding the use of only using a digital circuit instead of an analogue one with altered voltage levels for identifying high and low signals.

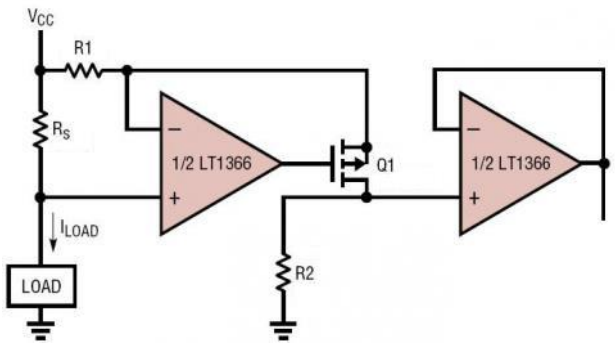


Figure 9.7: Standard positive supply rail current-sensing circuit taken out of [35]

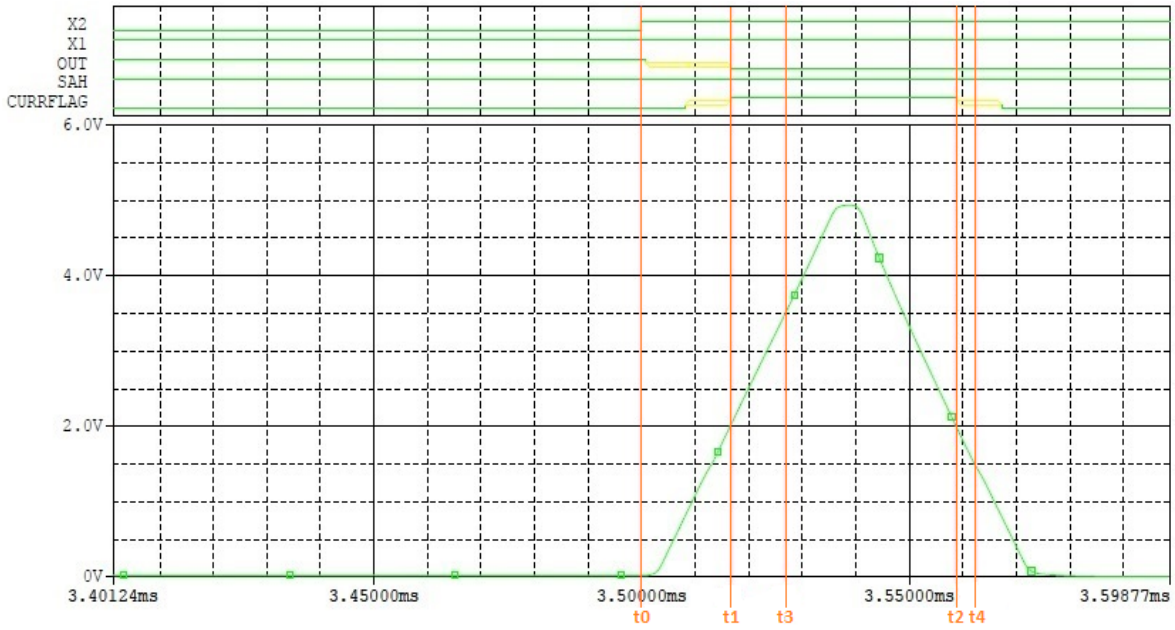


Figure 9.8: Output voltage graph of the supply rail current sensing in relationship to the digital signals of the self-healing phase

The timing of the impact of the self-healing feature triggered by the I_{ddq} current increased in relationship to the output behaviour indicates the response time of the intrinsic feature of this circuit. The timing simulation of these signals has been performed. The effect on the I_{ddq} current flowing in the SAFR-NAND gate in relationship to the output voltage of the supply rail current-sensing circuit is represented in Figure 9.9. At the time t0 the I_{ddq} current increases within a short time of around 1µs to its maximum current level limited through the transistor-specific drain-source current, also known as the maximum current capability which is specified within the datasheet [133]. As analysis with Figure 9.8 shows at the time t1 the ccfuse circuit switches from zero to high indicating a single SAH fault, which is affecting a single transistor of the SAFR-NAND gate. The alteration of the currflag (see Figure 9.5) activates the self-healing capability of this gate and eliminates the effect of the SAH fault happening to this transistor. Due to the elimination of the SAH fault the I_{ddq} current decreases at the time t1 back to zero. This is the normal condition of the I_{ddq} current for this situation. Also at this time t1 the output value of the logic gate changes into the correct output value from being undefined beforehand and for this case to be zero. Also within this figure the duration of the increased I_{ddq} current can be evaluated and the current is flowing 16.5µs within the SAFR-logic gate.

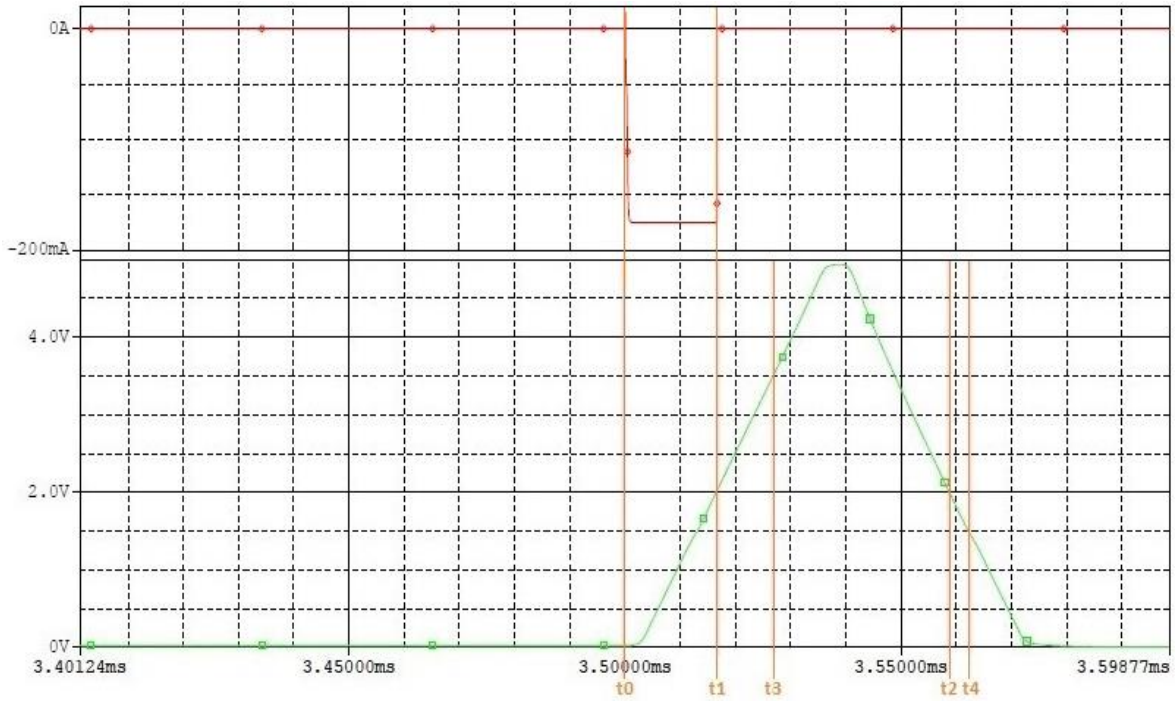


Figure 9.9: Output voltage graph of the supply rail current-sensing circuit in relationship to the I_{ddq} current of the SAFR-NAND logic gate

9.2.3. Initiation of self-healing at SAFR-NAND gate with reconfiguration

In Chapter 5.3.2.2 the concept of using reconfiguration with predefined configuration data for altering the logic structure within an FPGA in the case of a fault present inside a certain logic block of the whole logic structure is presented. For performing this approach, the system designer needs to evaluate all possible fault conditions and counteract with an altered logic design. For all these different logic designs he has to generate the appropriate configuration data files. All these data files of the different logic design configuration data sets are going to be stored outside of the FPGA within a memory circuit and this memory circuit is part of an external checker system. After evaluating the fault location within the logic structure of the FPGA, the system-checker will select the appropriate data set and alters the configuration of the FPGA by reprogramming. This specific concept requires a system-checker and also a memory element storing all predefined data sets. All of this represents hardware overhead and increases the likelihood of a system-fault due to SEUs or hardware related issues.

The SAFR-NAND gate has an intrinsically built-in indicator for non-maskable single SAH faults affecting a single transistor of the logic gate. An SAH fault and the required IC will increase the I_{ddq} current and this current increase can be used to trigger self-healing concepts without an external checker circuity. All other fault conditions within the SAFR-logic gate are masked and not noticeable for the user of this logic gate. This self-healing circuity is able to fix a transistor with a

fault by means of selective decommissioning of the SAFR-NAND gate. This would be possible if an appropriate fusing technology had been added to each of the individual transistors of the SAFR-logic gate.

Due to the fact that the current design is not equipped with this fusing technology another approach of fine-grained self-healing has been chosen. The *currflag* signal defined within Figure 9.5 will be used for triggering a fine-grained reconfiguration of two SAFR-NAND logic gates. This reconfiguration has the goal of maintaining the required logic functionality of the logic structure during runtime of the systems in the case of a non-maskable fault affecting the logic gate. All this is going to be done by only using the I_{ddq} current indicator for the initiation of self-healing by means of reconfiguration without external use of a system-checker.

The basic principle of the logic structure is shown in Figure 9.10 where, in this logic structure, the intrinsically built-in capability of fault fed-back through I_{ddq} current triggers the switch between two SAFR-NAND gates. Both SAFR-NAND gates are altered in a way that a transistor has been added inline within the GND and Vcc line each. The function of these transistors is to act as an isolation switch, isolating the SAFR-NAND gate from the power rail. By isolation of the central logic part of the SAFR-NAND gate from the supply voltages the logic gate is put into a floating output condition and the output will reflect this condition as being undefined. This condition is referred to as a tristate logic condition of a logic gate in accordance with [36]. The *select logic* block represented in Figure 9.10 located between both SAFR-NAND gates is going to generate the required digital signals for switching between both gates in terms of working or tristate condition. The select logic block is triggered by the *currflag* signal. The *output logic* block of this block diagram is situated where both logic gates are feeding their output signals in and it is capable of selecting the valid output signal of the active SAFR-NAND gate. The correct selection is performed due to the fact that the faulty one or the standby gate has a floating output, which represents an undefined output condition. Everything else of this block diagram shown in Figure 9.10 is similar to the one shown in Figure 9.4 in terms of functionality.

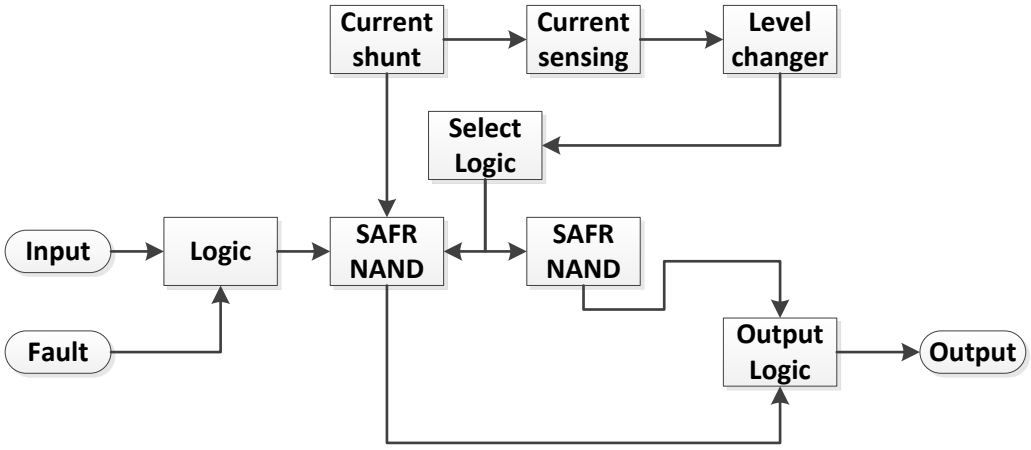


Figure 9.10: Block diagram of I_{ddq} current triggered self-healing of the system performance in the case of the presences of a SAH fault by means of reconfiguration

Figure 9.11 shows the simulation results of the circuit constructed within spice following the block diagram as it is displayed in Figure 9.10. At this time the single SAH fault is injected into the T1 transistor (see Figure 9.2(a)) of the first SAFR-NAND gate. In accordance with Table 7.4 a single SAH fault injected into T1 of the SAFR-NAND gate requires the IC condition X1=1 and X2=1. Through this IC combination and the transistor T1 under the influence of SAH the I_{ddq} current increases and the logic gate output value is undefined. The different outputs are labelled in the block diagram as the following *out1*, *out2* and *out3* which are going to represent the different sub-functionalities. The labels *out1* and *out2* are the representation of the output values of the two SAFR-NAND gates and *out3* represents the overall output value of this logic construction coming out of the output selector. The yellow lines within the timing plot for the labels *out1* and *out2* represent within the simulation data the time frames in which one of the two outputs is switched into tristate or floating state [36]. During this time the output value is floating, which means that it is undefined and this is due to the fact that the logic gate is isolated from both sides of the supply rail. The output *out3* represents during the whole simulation a constant output value, which means that at any time during the simulation *out3* does not have an undefined state. This output is the only visible output for the user of this logic structure and during the simulation it experiences no incorrect output value that has been generated. At the time of 2ms a single SAH fault is injected into transistor T1 of the first SAFR-NAND gate, which is the one on the left of the block diagram illustrated in Figure 9.10. The required IC condition of X1=1 and 2=1 in accordance to Table 7.4 is happening at 3.5ms during this simulation run and at this time point the first SAFR-NAND gate on the left becomes faulty with the known features of increasing the I_{ddq} current and undefined output value. The self-initiated switchover between both these SAFR-NAND gates is triggered through the I_{ddq} current increase and this can be seen within the simulation timing data regarding output

depending switch of the tristate value after the time 3.5ms. A more detailed timing diagram of this switchover of the time around 3.5ms is illustrated in Figure 9.12.

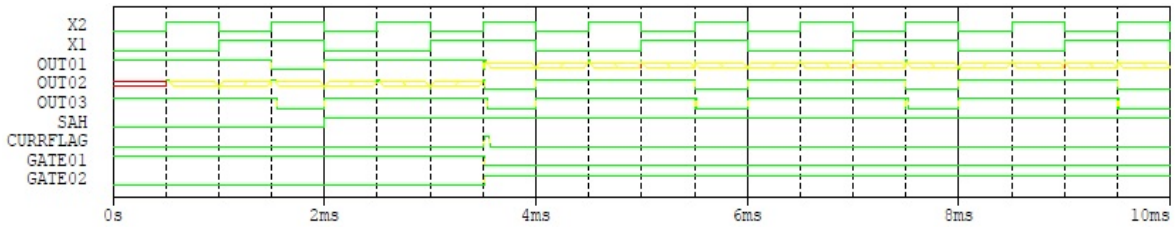


Figure 9.11: Self-initiated switchover between two SAFR-NAND gates triggered through the I_{ddq} current for maintaining functionality after SAH fault occurred

With Figure 9.12 a more detailed timing diagram of the spice simulation illustrated in Figure 9.11 is outlined to show the fine detailed switchover between both SAFR-NAND logic gates around the time of 3.5ms. The trigger for the self-initiated switchover or self-healing is the IC combination X1=X2=1. Within this timing diagram shown in Figure 9.12 the features of how the output value *out3* is going to be generated is evaluated and indicated as trustworthy. As long as the *currflag* is set high the logic structure using the output value of *out3* should wait until this flag is switched back to zero. At this time the overall output value *out3* has come to a stable and correct output value. A detailed evaluation and comparison against the timing parameters can be found within chapter 9.2.2. Figure 9.13 shows the same timing breakdown as shown in Figure 9.8 for the evaluation of the response time evaluation of the current sensing circuit in relation to the digital signals. The reference timing point is t0 with 3.5ms. The *currflag* switches at 16.5µs, indicated through t1 from undefined into high triggering the self-initiated self-healing of this logic structure by means of logic gate reconfiguration. In Figure 9.8 the SAH fault was cleared and in Figure 9.13 the switchover or reconfiguration between both logic gates is performed. Both these events happen at the same time point t1. All the timing parameters found in Figure 9.13 match those found in Figure 9.8. In this regard the performance of the self-healing in terms of clearing a fault or switching between two SAFR-NAND gates, matches and is showing the same performance. A difference between both timing parameters was not expected due to the fact that both ccfuse circuits are identical copies of each other. The I_{ddq} current behaviour found in Figure 9.9 matches the one illustrated in Figure 9.14 in both performance and timing.

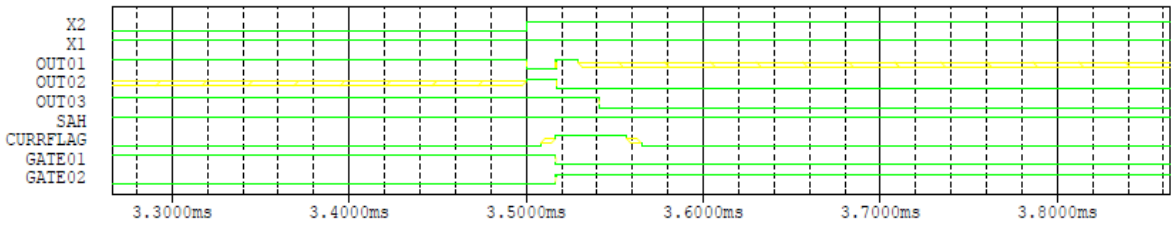


Figure 9.12: Ttiming diagram of the self-initiated switchover between two SAFR-NAND gates triggered through the I_{ddq} current for maintaining functionality after SAH fault occurred

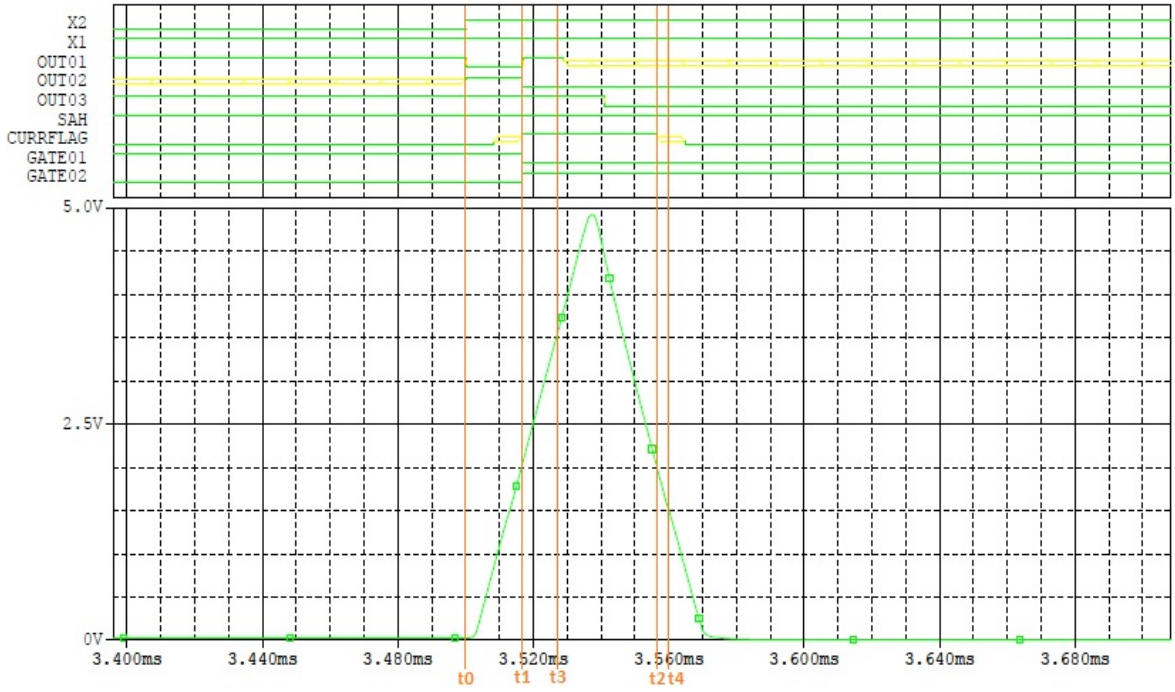


Figure 9.13: Output voltage graph of the supply rail current sensing in relationship to the digital signals of the self- initiated switchover between two SAFR-NAND gates triggered through the I_{ddq} current for maintaining functionality after SAH fault occurred

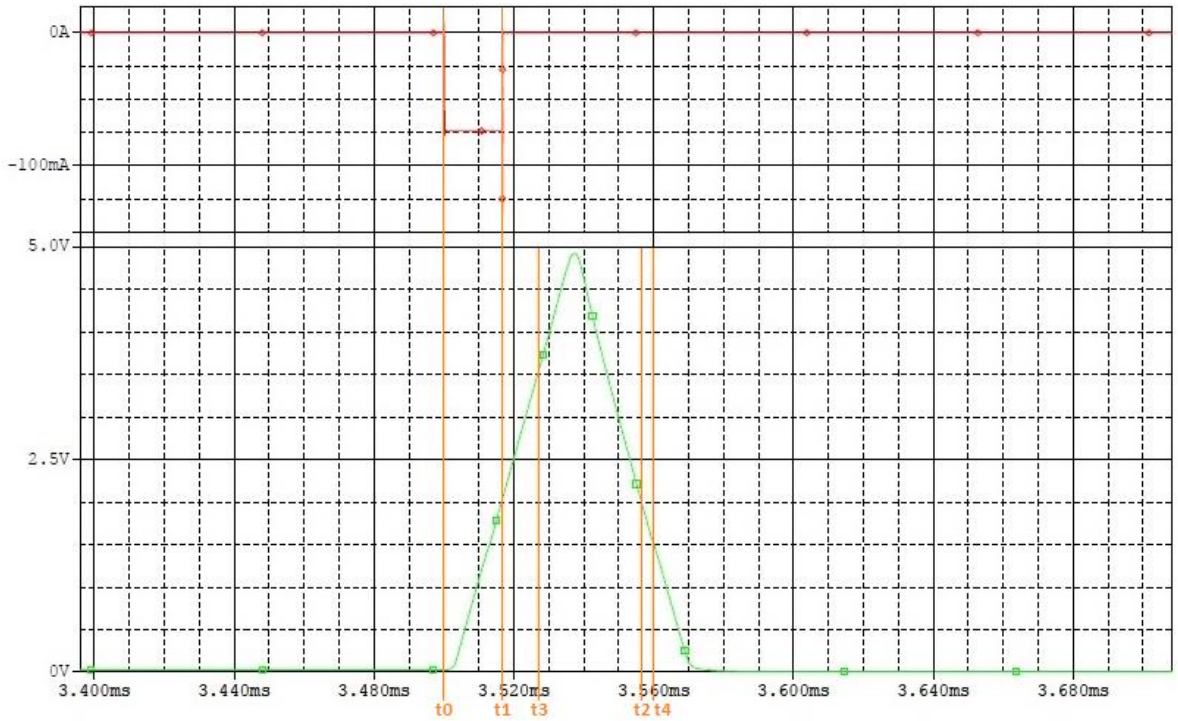


Figure 9.14: Output voltage graph of the supply rail current sensing circuit in relationship to the I_{ddq} current of the SAFR-NAND logic gate similar to the Figure 9.13

9.3. Fault identification capabilities within the QLC logic structure

The self-diagnostic feature is required for triggering the self-healing in the case of a fault within its logic structure. A comparison between the QLC and quadded logic structure is performed for showing the fault identification capability of the QLC logic structure. The QLC logic structure has the fault-tolerant capability due to the time-triggered round-robin reconfiguration of a fixed logic circuit. A faulty logic unit will rotate through this logic set-up and will generate altered output results for each cycle. The quadded logic structure is designed with the approach of having built-in fault correction and fault-masking. For both logic structures the majority voter performs the bulk of the fault-masking, similar to every other system where a majority voter is used. Fault identification within both logic structures requires external checker systems because neither of these logic structures can perform this task through its internal logic structure. Both logic structures are fault-tolerant but not for all possible logic combinations applicable within the fixed logic structure which the data shows of the simulated carried out within Chapter 6. The data of the different numbers of faults per logic set-up within the fixed logic structure can be found within Tables 6.5 and 6.6 and an overview of the main conditions can be found within Table 6.7. In this table the total number of faults (F) has been accounted for which can pass through the functional boundary, in this case the majority voter, as valued output results. Both logic structures require an external checker to detect these faulty output results for the required logic structure performed. This external checker will

work on the concept of identifying the faulty part or functional block of the system and will also identify the fault-causing logic part through the evaluation of the data set. After the identification of the fault-causing logic part the system-checker can trigger a reconfiguration or self-healing by the use of spare logic hardware designed into the logic structure for this purpose. For a more detailed fault identification down into the logic unit of the QLC reference can be found within Chapter 7, which focused on altering the internal logic structure into a fault-tolerant design.

One possible common approach for making both logic structures to be 100% fault-tolerant is a parallel system approach. Both logic structures in parallel have to be of equal logic structures, which are using the same input signals for generating an output result at the same time as a single logic structure. After the generation of both these output results a comparison takes place and both output results have to match in order to generate a valid overall output value. This set-up would be considered as a lock-step parallel system (see chapter 4.6.2). By doing so the idea of creating a minimal hardware requiring a fault-tolerant system would be obsolete because of the 100% hardware overhead due to the parallel logic circuit and the use of a comparator. Because of the hardware overhead a simpler external system-checker is the other option, which could be applied to both logic structures and two possible applications are described within this chapter out of a wide-ranging set of solutions. The first application is working on the principle of identifying output result inconsistencies of a single functional block of these two logic structures. Another application is working on the concept of a majority-voted output result fed-back solution, which can be used for a TMR-based system or any other system in which a clear separation of the output-generating logic circuit can be made (see chapter 4.6.1).

The first application solution for a simpler checker is to add an additional logic structure to the logic system, which can monitor the output values of each output-generating functional block to spot abnormal output sequences and indicate it to a higher-level control system. This solution would check if all the output values of the functional blocks match and no single deviation of one output result exists (principal logic structure can be seen in Figure 4.9 and Figure 4.12). This is a simple way of performing fault identification of a single fault-causing output-generating logic path. This would be a system, which is based on diagnosing a fault but has no features of correcting the fault-causing hardware.

The second application is designed around the concept of comparing the majority-voted output result to each of the individual output results of each functional block. By the means of individual result comparison any deviation between output results can be used for identifying a fault-generating functional block. The deviation between output results indicates that a fault within one of the functional blocks is actively affecting the logic circuit behaviour. An example block diagram of a TMR-based system with majority-voted output signal fed-back for comparing the individual output result of each signal path is described in Figure 9.15. This figure shows a TMR-based system with majority voter fed-back system for comparison against the individual functional blocks

of the TMR system. Because of this an identification of the fault-causing functional block is possible and actions of repair can be taken.

Both external checker applications can be used on either one of the logic structures and are capable of detecting faults. These system-checker solutions cannot fix a fault within a functional block, they are only capable of indicating its existence to a higher control system.

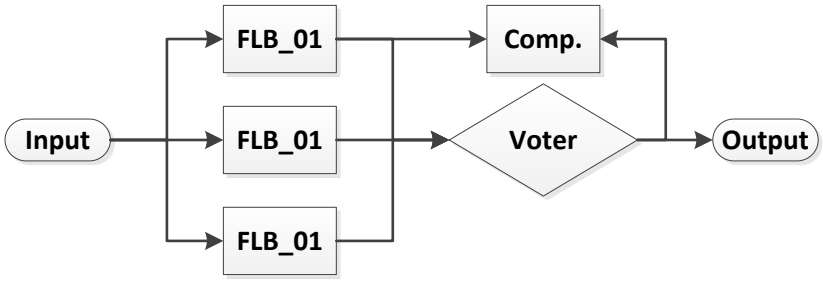


Figure 9.15: TMR-based block diagram with majority-voted output signal fed-back into individual output signal comparison

A different type of the second external checker application for the QLC structure is built around feeding back the majority-voted output signal to a comparator identifying abnormalities within the individual output signals of the functional blocks, which are feeding into the majority voter. With the help of this set-up the incorrect logic structure path can be identified in case of a clear separation between the individual primary logic paths. Through this a faulty primary logic path can be identified and corrected through reconfiguration of this part of an FPGA to match a new fault-free circuit layout. This task is possible for a TMR-based system (represented in Figure 9.15) and a QLC-based system but not for a quadded logic-based system. The quadded logic structure is done with the focus of fault-tolerance with the help of an interwoven interconnected structure between the different logic gate levels. Isolation of an individual faulty logic gate or sub-system is not possible for quadded logic-style structures or not even with a sophisticated external checker system. The design of the temporal-dependent reconfigurable round-robin matrix element or QLC was focused on the use of a set of logic gates within a fixed logic structure by constant reconfiguration. The arrangement of the reconfiguration is represented in Figure 6.6(b) with the individual clock-related logic unit arrangements. Due to the round-robin reconfiguration and the defined $^{2}/_{3}$ logic gate overlay an identification of a fault down to a single logic unit is possible. For achieving the diagnosing of a faulty logic unit within the QLC structure, the external checker system for the majority-voted output fed-back signal, shown in Figure 9.15, had to be redesigned for the QLC logic structure. The timing of the majority voter after the QLC structure has to be expanded by one clock cycle after all the necessary four output results had been generated. Also the newly designed checker system requires separated and associated with one of the clock cycles as

fault-flag memory element for each individual output result validation. Each of these fault-flag memories represents a specific clock cycle of the time-triggered QLC round-robin cycle. These fault-flag memory elements' task is to set flags in case a fault in this particular clock cycle has been identified and latched until a certain number of faults are detected. The block diagram of this altered and expanded QLC system is displayed in Figure 9.16. The fault-flag memory element stores the occurrence of a fault within its associated clock cycle flag until it is cleared. With the help of these fault-flags and the knowledge of the clock cycle defined logic units utilisation for creating the pre-defined logic structure, an identification of the fault-causing logic unit can be performed. After the identification of the fault-causing logic unit self-healing features can be triggered for exclusion of this unit and replacing it with a working fault-free spare unit.

An example of the faulty logic unit identification is shown in Table 9.1 where the faulty logic unit is logic unit B with an SAH output fault of a given QLC structure. This permanent stuck-at fault occurs before the second full execution of the full cycle of the QLC. The logic unit B is defined within Figure 6.6(b). Each row of the table represents a full cycle of four clock cycles required for a full round-robin approach of the QLC structure for generating a set of four output results. The correct output values for the different clock cycle results are represented through OC1 to OC4. These values are compared to the values OF1 to OF4 which were created under the possible influence of a fault within one of the individual logic units. Within Table 9.1 the following labels are used in this way. The labels I1 to I4 represent the individual input stimulus applied at this point onto the QLC element. Through the labels L1 to L3 the required logic gate functionality selected at this moment is represented in accordance with Figure 6.5(b). Within the column labelled with M the occurrence of a maskable fault at this moment within the QLC structure has been identified and the opposite has happened within the column labelled NM. In this column the occurrence of a non-maskable fault is reported. The column labelled with F represents a fault generated through the majority voter in comparison to the known correct value. The majority voter will generate an overall zero value in the case of a contradictory input data sequence. This case is given in the case of an equal split of zeros and ones feeding into the majority voter.

The first identifiable fault is detected at order number 2 in faulty output 4 (OF4) and the deviation against the correct value OC4 sets the fault-flag 4 (FF4). The internal structure of the comparator has the built-in feature of switching from a four to a three input comparator after the first fault-flag has been set. This is needed to maintain a trustworthy comparator. By not deactivating the fault-causing output a situation of equal value distribution can occur from now on and no correct comparison of the fed-back majority-voted signal is possible. The same feature of input downscaling happens for the majority voter to maintain a trustworthy system. This approach of input downscaling can be seen as the first layer of self-healing capability designed into the logic structure shown in Figure 9.16. A functional logic circuit of the input decreasing majority voter is illustrated in Figure 9.17. The central part of this logic circuit is a standard four-input majority

voter in conjunction with an input and output switching unit (SW). Through these switching units a selective deselection on specific input signals is possible. The input switching unit isolates the fault-carrying input signal and switches the remaining input signals in a way towards the majority voter that a working voter missing one input signal is possible. This means that the top AND logic gate of the majority voter shown in Figure 9.17 is isolated and is not used for the voting. The hardware overhead for this type of majority voter in compared against a standard four-input majority voter is in the switching units, which is controlled through the fault-flag memories. The minimum input signals used by the decreasing majority voter are two input signals. At this point the majority voter is converting the redundant logic system into a lock-step system. The self-diagnosing required for triggering this self-healing feature of the decreasing majority voter is the identification of the first faulty output comparison. The loss of one output-generating logic path decreases the quadded system to a TMR system. A further deselection of another output-generating logic path for the example shown in Figure 9.16 will put this system into a lock-step approach. Within a lock-step system both outputs have to be of equal value and due to this in the case of a disagreement, the system cannot determine which the incorrect value is. This set-up of the deselecting majority voter cannot be used within this set-up because a fault-localisation capable of identifying the fault-causing logic unit of the QLC element is not possible. The second identifiable fault happens at order number 6 in faulty output 1 (OF1) setting the fault-flag 1 (FF1). The final fault-flag is set at order number 8 though the faulty output 2 (OF2) sets fault-flag 2 (FF2) and triggers the identification of the individual logic gate. The trigger for the identification of the faulty logic unit happens, after three fault-flags are set. In this case it is exclusively specified through the missed fault-flag 3 (FF3) which identifies the logic unit B. This also matches the logic unit utilisation per clock cycle. Within this clock cycle the fault-flag for logic unit 3 is not used (see Figure 6.6(b)) and has not been showing up as a fault during the simulation run which is shown in Table 9.1. Through the use of the logic unit arrangement during each round-robin clock cycle which was shown in Figure 6.6(b) and with the help of the fault-flag the faulty logic unit can be identified. The clock cycle is matching the fault-flags in this regard, that the fault-flag 1 is associated with the clock cycle 0. By comparing the fault-flags with the definition of the used logic units in conjunction with the fault-flag which is not set, the clock cycle 2 shows that in this logic unit use the logic unit B is not used and no fault has been detected for these clock cycles. By not using logic unit B within this clock cycle a fault-causing unit could not alter the output result and because of this it is the fault-causing unit. The correct identification of this faulty logic unit can now be used for replacing this unit with a spare fault-free unit.

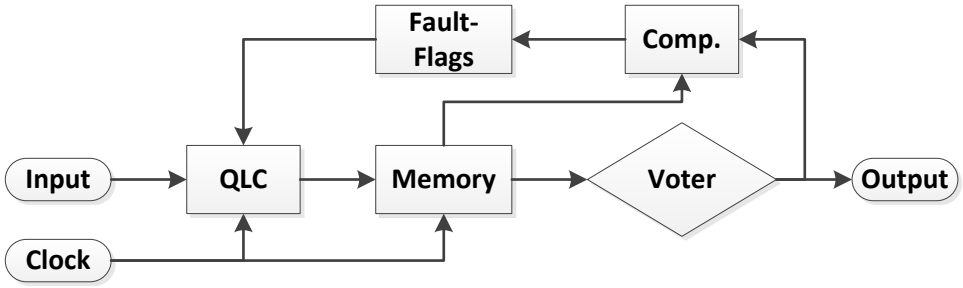


Figure 9.16: QLC with majority voter output fed-back into comparator for identification of faulty individual output signal stored in fault-flag associated with clock cycle

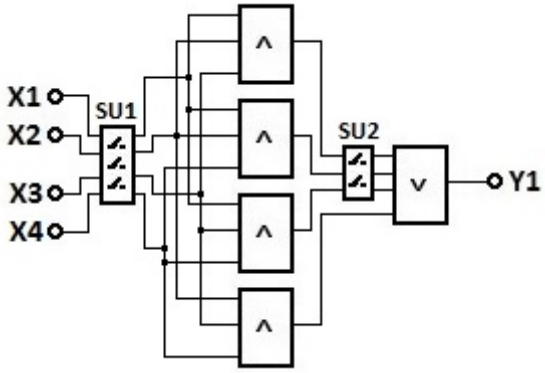


Figure 9.17: Functional diagram of a decreasing input using majority voter through the use of the two switching units (SUx)

Order	I1	I2	I3	I4	L1	L2	L3	M	F	NM	OC1	OC2	OC3	OC4	OF1	OF2	OF3	OF4	FF1	FF2	FF3	FF4
1	0	0	0	0	1	1	1				0	0	0	0	0	0	0	0				
2	0	0	0	0	1	1	1	x			0	0	0	0	0	0	0	1				x
3	0	0	0	0	1	1	1				0	0	0	0	0	0	0	1				x
4	0	0	0	0	1	1	1				0	0	0	0	0	0	0	1				x
5	0	0	0	0	1	1	1				0	0	0	0	0	0	0	1				x
6	1	1	0	0	1	1	1			x	0	0	0	0	1	0	0	1	x			x
7	0	0	0	0	1	1	1	x			0	0	0	0	0	0	0	1	x			x
8	0	0	1	1	1	1	1				0	0	0	0	0	1	0	1	x	x		x
9	0	0	0	0	1	1	1				0	0	0	0	0	0	0	0				
10	1	1	1	1	1	1	1				1	1	1	1	1	1	1	1				

Table 9.1: The simulation data flow of a fault within a logic unit and the approach of using round-robin logic structure reconfiguration for the identification of the single faulty logic unit. In this case the logic unit B with an SAH fault (see Figure 6.6(b))

At this point the logic structure within each logic unit represents single logic gate functionality. The analysis of this fault identification capability up to this point shows that this concept of the round-

robin reconfigurable logic structure inside the QLC logic element can handle altered logic complexity within each logic unit for identifying a faulty logic unit. The altered logic complexity represents logic circuits with n-number logic gates which are beyond the single logic gate currently designed into each logic unit today. One of the disadvantages of upscaling the number of logic gates used within a single logic unit is that the current approach of time-triggered reconfiguration will lead to hardware overhead controlling logic gate functionality and interconnection forming this logic structure. Due to this problem a fixed logic structure would be a better approach in implementing this. Also identification of a faulty individual logic gate inside this logic structure would require specific self-testing capabilities added to each QLC element or as an external checker testing as faulty identified logic units through a QLC element array.

9.4. Circuit interconnection fault-localisation through memory-based BIST functionality

Every digital system is a combination of logic functionality and interconnection between the logic function and an interface. Faults like stuck-at can affect both parts of a given digital system. Digital systems only have two valued signal types operating within the structure. Stuck-at faults are also altering these two types into only one of these signal types. Through these fault identification can work on pattern matching for possible fault-localisation. This task is performed by an external checker system monitoring the signals of a digital system. The complexity of this type of checker is defined by the task of this checker and is in most cases controlled by a type of microcontroller. Within Chapter 8 the mapping of a given FSM into a memory-based platform replacing combinational and sequence-based platforms has been analysed. The system-checker performing the localisation of a fault within the interconnection structure of a digital system will be based on this concept due to the digital nature of the signals. An analogue system has the same structure like the digital system of interconnection between components. The difference between these two systems means that the analogue-based system is using a wide range of internal system signals and fault-localisation, which cannot be based on the same concept as in digital systems. The research work presented throughout in this thesis is focused on digital systems only with the exception of the I_{ddq} current used in the SAFR-type logic gates. Stuck-at faults in a digital system can affect each part of the system in a different way. For creating fault-tolerant digital systems the distinction between the affected parts of the systems and fixing it is the challenge. The detection of this type of fault requires hardware for performing BIST and built-in self-repair (BISR) functionality. A self-healing system requires at least both of these functionalities as part of the system and in the best case without external intervention during the task of fault-handling. This chapter is focusing on combining BIST and BISR into a single function for interconnection fault-localisation and adding it to a given logic system. This newly created checker will be designed in a minimal hardware requiring design based on the findings presented in Chapter 8. This logic system needs to be

capable of handling a wide spectrum of single stuck-at faults affecting different interconnection parts of a given logic system by itself.

The effects of single stuck-at faults at the interconnection points of a TMR majority voter system have been illustrated in Figure 4.8 and defined in Table 4.2. This table illustrates the effects of output alteration due to stuck-at faults measured in FR on the TMR majority voter system after the impact of injecting stuck-at faults into a majority voter system at the injection points defined in Figure 4.8. The FR range of this system is 13% as a minimum and 50% as a maximum for different injection points. Its average FR is 22.77% taken over all injection points for the TMR majority voter. Analysing the effect of stuck-at fault onto the internal transistors of a logic gate is described in Figure 7.2 for the example of a NAND gate and shows that the average FR is 18.75%. By comparing both it can be evaluated that the FR for the internal transistor associated set-up is less than the one for the interconnection one. This is because stuck-at faults injected into the interconnection structure of a logic system alter the input information to any number of logic gates altogether. This alteration of the input information at any number of logic gates input is because all are connected to this fault-occurring part of this interconnection. A stuck-at fault affecting a single transistor of a logic gate affects only a sub-input information within an individual logic gate. The distinction between the faults happening at these different elements of a digital system is the key for the development of unique fault identification test systems as part of the self-healing requirement. This test system is adapted into a memory-mapped FSM for the protection of the test system against faults.

The fault concept of identifying interconnection faults caused by stuck-at conditions within a given digital circuit is based on the utilisation of only using SAFR type logic gates within the digital circuit structure. By only using SAFR type logic gates the internal non-maskable stuck-at faults at the individual transistors of each gate can be identified through the clear current increase of I_{ddq}. This current increase of I_{ddq} is only happening during the time of affecting the circuit with the stuck-at fault and if the required IC is applied at the faulty SAFR type logic gate. In all other cases the fault can be masked. The test circuit C17 of [27] is illustrated in Figure 9.18 and has been constructed only by using SAFR-NAND gates. The required corresponding stuck-at fault-injection points at the interconnection within the C17 circuit have been added and are shown in the same figure. Table 9.2 shows all possible IC combinations, which can be applied to a fault-free C17 circuit and the resulting output values for each of these different IC stimuli. The C17 circuit was simulated within MATLAB including the stuck-at fault location points indicated in Figure 9.18. In Table 9.4 the results of the SAL injected faults at the fault-injection point S1 to S17 are displayed. This table results have been cross checked by creation of the C17 circuit on a breadboard by using standard NAND logic gates and the same SAL injection simulation has been performed. The result of this simulation on the hardware matches the results found by software simulation. (see appendix 6 of the test set-up)

The data within Table 8.6 represents the correlation between the fault-injection point and the external IC stimulus at the C17 circuit for injecting SAL faults at the different possible injection points. In this table in the case of a deviation from the correct output value (defined within Table 9.3) happening it is indicated through the coded deviated output value at this fault-injection versus IC cross point. The deviated output value is in accordance with Table 9.3 and by using the same colours for coding, it can be better identified within Table 9.4 and Table 9.5. In Table 9.5 the results of the SAH-injected faults similar in structure as the ones for SAL outlined in Table 9.4 are exposed. Also the results shown in this table have been verified through running the same set-up as before on the breadboard for SAH-injected faults and both results match.

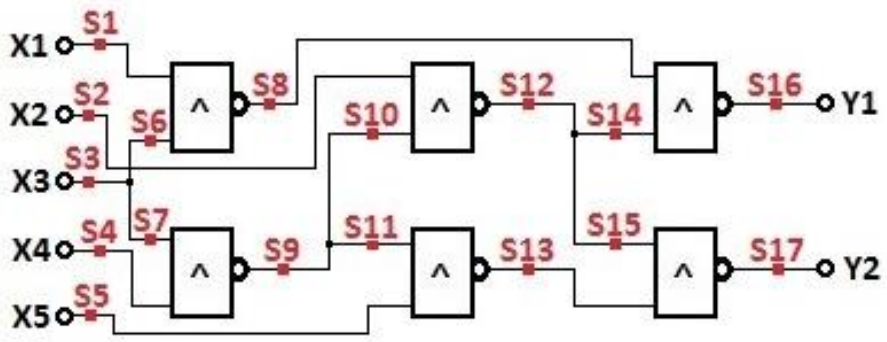


Figure 9.18: Test circuit C17 of [27] with added stuck-at fault-injection points for interconnection fault simulation

	IC1	IC2	IC3	IC4	IC5	IC6	IC7	IC8	IC9	IC10	IC11	IC12	IC13	IC14	IC15	IC16	IC17	IC18	IC19	IC20	IC21	IC22	IC23	IC24	IC25	IC26	IC27	IC28	IC29	IC30	IC31	IC32
X1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
X2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
X3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
X4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
X5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Y1	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0	1	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0	1
Y2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0

Table 9.2: IC related output results of the C17 circuit without the presences of a fault within its circuit

Y1	Y2	
0	0	A
1	0	B
0	1	C
1	1	D

Table 9.3: Output result coding of the C17 circuit

	IS1	IS2	IS3	IS4	IS5	IS6	IS7	IS8	IS9	IS10	IS11	IS12	IS13	IS14	IS15	IS16	IS17	IS18	IS19	IS20	IS21	IS22	IS23	IS24	IS25	IS26	IS27	IS28	IS29	IS30	IS31	IS32
S1					B								B		B						D								B		B	
S2	D	D			D	D			D	D							D	D			D				D	D						
S3		B								B	A	B						D							A	B	A	B				
S4				A	B																A	B	A	B								
S5	C	C			C	D			C	C																						
S6		B								B								D								D						
S7											A	A													A	A	A	A				
S8						A								A		A						A								A		A
S9															D	D													C	D	D	D
S10																D	D														D	D
S11																													C	D	C	D
S12			A	A			A	B			A	A							C	C			C				C	C				
S13																	A	A			A	B			A	A						
S14			C	C			C				C	C							C	C			C				C	C				
S15			B	B			B	B			B	B																				
S16	B	B			B				B	B			B		B		D	D			D				D	D			B		B	
S17	C	C			C	D			C	C			C	D	C	D													C	D	C	D

Table 9.4: Corresponding fault location with IC and resulting output values in accordance with Table 9.2 for SAL fault-injection at C17 fault-injection points S1 to S17

	IS1	IS2	IS3	IS4	IS5	IS6	IS7	IS8	IS9	IS10	IS11	IS12	IS13	IS14	IS15	IS16	IS17	IS18	IS19	IS20	IS21	IS22	IS23	IS24	IS25	IS26	IS27	IS28	IS29	IS30	IS31	IS32
S1						A								A		A						C								A		A
S2			A	A			A	B			A	A							C	C			C				C	C				
S3						A								A	D	D						C							C	C	D	D
S4															D	D													C	D	D	D
S5																	A	A			A	B			A	A						
S6								A						A		A						C								A		A
S7															D	D													C	D	D	D
S8	B	B			B				B	B			B		B		D	D			D				D	D			B		B	
S9			A	A			A	B			A	A					A	A	A	A	A	B	A	B	A	A	A	A				
S10			A	A			A	B			A	A							C	C			C				C	C				
S11																	A	A			A	B			A	A						
S12	D	D			D	D			D	D			D	D	D	D	D	D			D				D	D			D	D	D	D
S13	C	C			C	D			C	C			C	D	C	D													C	D	C	D
S14	B	B			B				B	B			B		B		D	D			D				D	D			B		B	
S15	C	C			C	A			C	C			C	D	C	D													C	D	C	D
S16			C	C		A	C	C			C	C		A		A			C	C		C	C	C			C	C		A		A
S17			B	B			B	B			B	B					A	A	B	B	A	B	B	B	A	A	B	B				

Table 9.5: Corresponding fault location with IC and resulting output values in accordance with Table 9.2.4 for SAH fault-injection at C17 fault-injection points S1 to S17

The FR for the SAL fault-injection data is displayed in Table 9.4, which gets to an overall value of 5.93% and the FR for the SAH fault-injection data is displayed in Table 9.5 reaching an overall value of 9.01%. More interesting is the overlay of SAL- and SAH-related faults tables 9.4 and 9.5 into one combined table. This overlay of both these tables has been done in Table 9.6 and reveals that the fault patterns for SAL and SAH do not overlay. Each fault caused by an injected stuck-at fault is a unique fault and this can be useful for a system-checker whose task is the identification of fault locations with the help of these unique configurations. These unique configurations are the

combination of IC and one of the incorrect output sequences. For example, at IC19 its pattern is in accordance with Table 9.2 *'01001'* and the correct output value is *'11'*. Each deviation from this output value is caused by a stuck-at fault within the interconnection part of the C17 circuit. In this way with an output value of *'00'*, which is coded as *"A"* in combination with the IC19 stimulus, the fault-causing interconnection injection point can be identified. For this example the fault-causing interconnection injection point or the faulty interconnection can be directly identified and it is S9 due to the data of Table 9.6. Identifying the interconnection injection point S9 within Figure 9.18 reveals that this interconnection is connected at an output feeding two inputs from two different logic gates. In this fault case the interconnection between output and the split into two interconnections needs to be repaired.

Within this thesis two concepts have been researched for fixing the fault-creating interconnection. The first solution is shown in Figure 9.19 in which the two data sets that are IC and circuit output are fed into an address of the memory-mapped FSM presented in Chapter 8. The difference is that no feedback data path will be used in this application because alteration of the address through the memory data is not needed. The generated address in the pointer is fed into the memory and the evaluation signal is read out of the memory controlling the output filter. In the case of an incorrect output value of the C17 circuit the filter suppresses the generation of the output values at the output register and indicates the presence of a fault within the C17 circuit per a status flag. In this case this concept is more of a fault-monitoring system without the capability of fault correction. Fault correction is possible by storing the correct output values within the memory data of the evaluation signal. By using the stored data for replacing the incorrect output data, it could be argued that the memory-based system-checker can replace the C17 circuit altogether. This is an open point of discussion because eventually every digital system where all the output combinations created through input stimulus are known would be subject to being replaced. This discussion is beyond the research timeline of this thesis.

The second concept is built on the capability of selective reconfiguration of the interconnection between the logic gates. This concept is showing the capability of the technique of identifying a faulty interconnection within a logic circuit and responding to this with self-healing features. The system-checker for this concept is based on the memory-mapped FSM strategy presented in Chapter 8 and utilised in the same way as the first concept. The memory-mapped logic functionality will not be used as one FSM operating within one memory block. Instead several small FSMs are defined within the same memory block. Each of these individual FSMs is associated with an IC applied to the C17 test circuit. The IC and the output data resulting from this stimulus create the required data set for the address-pointer which points to a certain memory location. If the data at this memory location has a specific type of structure, a fault free circuit response is indicated and no further action of the checker system is required. Every other data entry is linked to an interconnection fault, which is present within the test circuit. The combination of IC

and output data contains the information about the faulty interconnection and these combinations are presented within Table 9.6. The column of this table represents the IC condition and each row entry, which contains a value, is linked to an interconnection of the test circuit. Empty row elements are indicating that faults at these interconnections do not impact the output data or the circuit is capable of masking these faults. The same output data value can be linked to different interconnection faults. For finding the fault causing interconnection all these links have to be coded and, for stepping through this list, address-pointer alteration is required.

The feature of the address-pointer alteration through the memory data is needed for this concept because in some cases a fault search within a set of possible interconnections is necessary. By altering the data within the address-pointer different locations can be checked until the fault-causing interconnections has been found. The strategy of identifying faulty interconnection can be seen as self-diagnosing within a logic system with the help of minimal hardware overhead. As described within the example of IC19 the fault location has been identified as interconnection in which injection point S9 is feeding in (see Figure 9.18). After this localisation of the faulty interconnection, now a selective reconfiguration of this interconnection can be triggered for fixing the fault. Table 9.6 also shows that for almost every IC stimulus a set of identical incorrect output values exists. In this case the memory-based fault identification checker needs to work through the list of fault locations and alter the interconnection indicating the injection point until the output shows the correct output value. In case the reconfiguration does not resolve the fault the performed reconfiguration needs to be reset to the original interconnection. The system should only alter the faulty interconnection but not all possible interconnections to save resources.

	IC0	IC1	IC2	IC3	IC4	IC5	IC6	IC7	IC8	IC9	IC10	IC11	IC12	IC13	IC14	IC15	IC16	IC17	IC18	IC19	IC20	IC21	IC22	IC23	IC24	IC25	IC26	IC27	IC28	IC29	IC30	IC31
S1					B	A							B	A	B	A					D	C							B	A	B	A
S2	D	D	A	A	D	D	A	B	D	D	A	A					D	D	C	C	D		C		D	D	C	C				
S3		B				A				B	A	B		A	D	D		D				C			A	B	A	B	C	C	D	D
S4				A	B										D	D					A	B	A	B					C	D	D	D
S5	C	C			C	D			C	C							A	A			A	B			A	A						
S6		B						A		B				A		A		D				C				D				A		A
S7											A	A			D	D									A	A	A	A	C	D	D	D
S8	B	B			B	A			B	B			B	A	B	A	D	D			D	A			D	D			B	A	B	A
S9			A	A			A	B			A	A			D	D	A	A	A	A	A	B	A	B	A	A	A	A	C	D	D	D
S10			A	A			A	B			A	A				D	D		C	C			C				C	C			D	D
S11																	A	A			A	B			A	A			C	D	C	D
S12	D	D	A	A	D	D	A	B	D	D	A	A	D	D	D	D	D	D	C	C	D		C		D	D	C	C	D	D	D	D
S13	C	C			C	D			C	C			C	D	C	D	A	A			A	B			A	A			C	D	C	D
S14	B	B	C	C	B		C		B	B	C	C	B		B		D	D	C	C	D		C		D	D	C	C	B		B	
S15	C	C	B	B	C	A	B	B	C	C	B	B	C	D	C	D													C	D	C	D
S16	B	B	C	C	B	A	C	C	B	B	C	C	B	A	B	A	D	D	C	C	D	C	C	C	D	D	C	C	B	A	B	A
S17	C	C	B	B	C	D	B	B	C	C	B	B	C	D	C	D	A	A	B	B	A	B	B	B	A	A	B	B	C	D	C	D

Table 9.6: Overlaid corresponding fault location with IC and resulting output values in accordance with Table 9.2 for SAH and SAL faults injection at C17 fault-injection points S1 to S17

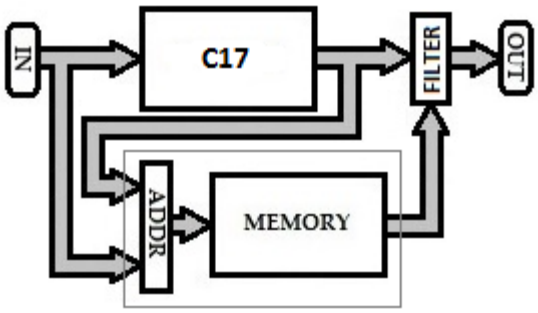


Figure 9.19: Expansion of C17 circuit by memory-based fault-existing checker

9.5. Summary of the chapter

Within this chapter the remaining question raised in Chapter 7 of utilising the inherent built-in feature of the SAFR-logic gate for the self-initiation of circuit alteration without the influence of external logic circuitry needed to be researched.

As in Chapter 7, the SAFR type logic gate is equipped with an inherent feature of indicating non-maskable SAH faults by means of I_{ddq} current increase. The indication of a fault presented within the circuit designed out of SAFR-logic gates, renders dedicated system-checkers that ordinarily monitor such events obsolete. Explicit fault detection and correction through traditional modular redundancy and fault-masking through majority-voting is not required. The hardware overhead of this system design has been generated through fine-grained redundancy at transistor level, which offers inherently built-in fault identification and masking.

Using the I_{ddq} current increase within an SAFR-logic gate for indication of a non-maskable fault produces a distinctive current signal, which can be converted into a digital signal. This signal can then be used for triggering self-reconfiguration action, thus maintaining functionality of the circuit. The digital signal is used either to decommission single transistors or reconfigure the logic gate structure. Simulations demonstrated a simple current converter design that is capable of eliminating stuck-at faults within a given redundant logic gate structure without external system checking. It was also proposed that this self-elimination of stuck-at faults through inherently built-in logic gate strategies could be regarded as self-healing.

Decommissioning of single faulty transistors within a logic gate through different types of fine-fuse technologies has been looked into. The standard burned open fine-fuse has been evaluated and has been neglected because of an unreliable current level and response time.

The reconfiguration between two SAFR-logic gates triggered by the I_{ddq} current signal has been analysed and successfully demonstrated by simulation. This concept can be applied to an entire functional logic circuit performing an application specific task.

A further extension of SAFR design is to protect the majority voter within conventional modular systems such as TMR, thus increasing their fault-tolerance. This exploits the immunity of SAFR gates against the influence of stuck-at faults so that the fault-tolerance of the majority voter will increase even further as was proven within Chapter 7. This fault-tolerant concept will impact the reliability of the majority voter.

The question about the self-healing capability of the QLC in terms of fault-localisation down to the faulty logic unit has also been analysed. The combination of time-triggered reconfiguration response in conjunction with alternate logic unit utilisation is done within the QLC element, thus forming a basis for fault-localisation. The altered and fixed overlay between each cycle rotates a faulty logic unit through a fixed quadded cell logic structure. Each output result is generated at different timing cycles and hence are affected through the faulty logic unit in a different way.

Therefore one, and only one, result is fault-free due to its exclusion. The analysis of the resulting set of output values indicates any persistent inconsistency. In the case of two incorrect values appearing, the generation of the majority-voted output value has the tendency of creating a faulty-voted output value, which has been revealed within Chapter 5 and Chapter 6. Because of this failure the fault-localisation concept was proposed to perform a decreasing input utilisation of the majority voter. Through this, a single detected faulty output value will trigger the deselection of this output-generating path and alters the four-input voter into a three input voter. From this time onwards the system's behaviour is like a TMR system and faults within this set-up can be truly related to the fault-causing logic path. By keeping a track on fault-causing logic paths through the help of fault latches, the required data for the localisation of the fault-causing logic unit can be collected. Through a certain data configuration in conjunction with the know-how of the utilisation of the logic units used for each cycle, the faulty logic unit can be identified. After the identification a selective process of self-healing can be triggered for re-establishing a fault-free QLC element.

The third research question raised in this thesis was about creating an FSM structure, which is capable of performing the task of fault-localisation and repairing it within a given logic circuit. For this task, BIST and BISR are required. This may be achieved through the memory-mapped FSM approach, as researched in Chapter 8, and further combined with fault-tolerance. The identification and repairing of a fault-occurring within a given logic structure was achieved by incorporating the logic functionality by SAFR-type logic gates. This allows a distinction between stuck-at faults within logic gates and their interconnections. The analysis of the fault responses generated by stuck-at faults affecting the interconnections revealed a correlation between fault location and input/output data sets. It was found that this correlation is unique for the different fault locations within interconnection structures and is described with the help of input and output data analysis. These data sets are transferred into a memory-mapped FSM taking the form of state transitions and that are in turn used for the BISR function. The functionality of BIST is part of the BISR due to the fact that if the input/output data does not generate a request for repair then this output value is assumed correct. This overall concept was proven by applying it to the standard C17.

Chapter 10: Conclusions and further work

10.1. Conclusions

The primary objective of this research work was to investigate the usefulness of novel self-healing and fault-tolerant concepts based on fine-grained redundancy for electronics structures. As observed in nature, comprehensive self-diagnosis is required for inherent self-healing capability and specific structures for detecting the occurrence of a fault are needed within the functional circuit. Equipping a logic gate with inherently built-in self-diagnosis was the main focus of this research work. The realisation of this feature for self-diagnostic triggered self-healing has been achieved within a range of logic structure designs wherein the equivalent of healing a cut to the skin of a human is triggered by the self-diagnostic of this part of the body and will set off the complex process of repairing the injury. The proposed self-healing concepts have been validated by means of various numerical simulations and hardware set-ups ranging from low-level logic gate structures to combined redundancy concepts implemented within a single logic structure all with the objective of improving the fault-tolerance and intrinsic, triggered self-healing capability. Any reliable system that incorporates fault-tolerance within an electronic system is based on one of three redundancy strategies: spatial (hardware), temporal (time) or pertaining to information, each of which offers advantages for the fault-tolerant response of the electronic system to which it is applied. These three strategies have formed the central set of redundancy concepts of this research work.

The ever increasing logic performance in electronic systems is due in part to the downscaling of the individual components manufactured on a given chip which are in turn increasingly sensitive to fault effects that may be permanent or transient by nature, and which can be counteracted by fault-correcting or masking techniques by means of applied redundancy in conjunction with a robust majority voter. Reliable systems are centred on one or both techniques through certain logic structures, increasing the fault-tolerance of the system by means of a certain provision of redundancy. These types of systems generate N equal output signals through which fault-tolerance is assured. Due to the N-number of output results, a single overall output is formed by majority-voting and through this fault-masking occurs. The majority voter investigated by itself is, however, not fault-free and represents a single point of failure within an otherwise reliable system. As a result, analysis has been carried out in Chapter 4 and enhancement of the fine-grained transistor structure performed within Chapter 7 resulted in improved fault-behaviour of the majority voter. A detailed analysis of the fault-behaviour of each redundancy concept has been carried out within Chapter 5, where the analysis conclusion was that the spatial (hardware) redundancy had the overall best fault-tolerance performance for permanent and transient faults out of the analysed set of redundancy concepts. For example, evaluated on system performance by using spatial as the

reference, temporal requires N-executions times and information time for valuation and correcting. The fault-tolerance improvement of each redundancy concepts comes with a price to pay with regard to hardware overhead, performance or a system-checker that is not part of the original functional logic structure.

This thesis considered three core contributions outlined within the abstract of this work for self-healing in electronics. The central core contribution investigated was the combining of all three redundancy strategies into one logic design with the objective of creating a structure possessing advanced, inherent fault-tolerant features that each strategy cannot offer individually. For comparison of each fault-handling capability, the quadded logic structure performing a set of logic functionalities was used. Quadded logic structures with majority-voting perform fault correction and masking simultaneously within their logic structure and their derived architecture is briefly summarised here. The combination of the three redundancy concepts within one logic structure was investigated according to a matrix element structure, or QLC, composed of four-tiled logic elements. Within each logic element, a set of logic functions can be selected through configuration switches that control the connections between logic functions and their appropriate input/output interfacing. These switches are connected to a loop-back shift-register and internal connections are established through runtime configurable control switches to create a predefined logic functional arrangement out of three of the four logic elements. These switches are in turn controlled by the loop-back shift-register located within the matrix element. The shift-registers are themselves central to the control of all QLC functions, and which are therefore responsible for selection of logic functionality within each logic element and interconnections. Finally, they also nominate the active selection of the used three out of four logic elements within a specific round-robin cycle. By using three out of four logic elements, a reuse of logic elements is possible and allows for fault-localisation through 50% overlap of elements from the previous cycle. Through this arrangement, the inherent capability of identifying a fault-causing logic unit within the QLC structure was further investigated. It had been established that, due to the use of a round-robin cycle, unique identification of the fault-causing logic unit is achievable and further that this can permits self-initiated repairing or elimination of the offending logic unit without an external system-checker. This may be seen as a system-level structure containing self-healing capabilities by design.

The QLC architecture was then used to create a circuit which is tolerant to permanent and/or transient faults within one of its configurable logic elements, thus preventing errors from manifesting at QLC outputs. From the perspective of fault-tolerant behaviour, the QLC was shown to be equivalent to the classic quadded logic structure for a variety of different applied logic functions evaluated by injection of stuck-at faults and calculating FR numbers. The complete FR comparison showed that the QLC with majority voter has an average FR of 2.25% whereas the quadded logic structure with majority voter has an average FR of 2.02% (see Table 6.8, Chapter 6) and therefore both logic structures are very similar in terms of fault-handling. Hence, in contrast to

the singular redundancy strategy of quadded logic, comparable fault-tolerance capability is possible by combining different redundancy strategies. As a result of this new-found flexibility the QLC is also capable of locating the individual logic element out of the four elements that contains a fault, thereby going beyond basic fault-masking. This was achieved by analysing the fault signature contained within the set of outputs generated by the temporal round-robin scheme. Finally, a comparison between the number of logic gate transistors between both QLC and equivalent quadded designs showed that the QLC structure uses one-third fewer transistors excluding interconnection overheads.

The second core concept associated with the core contribution explored in this thesis focused on improvements made to the fault-tolerance of the majority voter logic. The analysis within Chapter 4 identified that the majority voter is considered a single point of failure within a reliable system. Because of this, any fault-tolerant system using a majority voter is susceptible to non-maskable faults coinciding within the voting logic. The most common type of internal fault is the stuck-at fault, on which this fault-handling analysis was based. Instead of designing a majority voter with fault-tolerant logic an alternative voter design capable of both masking stuck-at faults within its gate logic and indicating non-maskable stuck-at faults was introduced. Traditional redundancy methods have used industry-standard logic gates designed out of transistor structures without redundancy. Fine-grain redundancy is applied in a way wherein each transistor within the network is replaced by four transistors, thus forming a design boundary in which minimal logic gate designs have been explored by others. Further, it was shown that fault-masking and detection may be achieved with certain redundancy structures i.e., masking one type of stuck-at fault and indicating the occurrence of another type of stuck-at fault by means of a clear signal. This analysis was done within Chapter 7 and specific redundant transistor structures analysed for their combined masking and indicating properties. Through this, a universal structure was proposed based on utilising twice as many transistors as the standard logic gate that is suitable for building a variety of standard logic gates such as NOT, NAND and NOR. Each of these was analysed in turn. Each gate implementation achieves SAL fault-masking combined with a selective indication of all SAH fault conditions. For the latter case, indication is guaranteed for one input state that triggers a distinct I_{ddq} current increase. Thus, in addition to SAL masking, all SAH faults can be indicated by this SAFR strategy. Returning to the problem of building resilient voting logic, the SAFR strategy was then applied to a generic majority voter design using NAND SAFR gates and it was shown that the resulting FR was reduced by a factor of 3.5 times. Furthermore the benefit of SAH fault-indication is retained within the voter via individual I_{ddq} current indication. Through this fine-grained alteration of the transistor redundancy structure within the logic gates a stuck-at fault-tolerant majority voter has been created.

The next concept reported in this thesis focused on the approach of utilising the SAFR fault-indication capability as a trigger for self-healing logic, which is related to the central core

contribution. Contemporary faults tolerant systems operate in conjunction with an external checker agent that monitors the behaviour of the system and seeks to protect against faults that would otherwise propagate across multiple logic blocks. By contrast, intrinsic fault-indication internal to the logic gates itself is a fundamentally different approach that does not rely upon external circuity. Within Chapter 9 a novel logic circuit built out of SAFR-NAND gates including reconfigurable means forms the basis for self-healing by measuring the I_{ddq} current and selectively triggering upon a certain threshold level a self-initiated reconfiguration action. Reconfiguration was demonstrated principally via switchover. Following injection of an SAH fault into the nominal logic gate the offending fault is indicated through the increased current that is converted into a current to digital voltage signal. This digital fault-indicating signal is used for triggering rapid reconfiguration whereby the nominal gate is isolated and a fault-free logic gate activated in its place within a short time period. After this reconfiguration, which happens within the valid clock cycle of the logic, a fault-free result is maintained without interruption. Thus inherent fault-masking, selective indication and reconfiguration properties that amount to self-healing capability have been demonstrated.

Another concept, which covers the third contribution, was investigated in this thesis and focused on designing a memory-only-based FSM platform with fault-tolerant features as a foundation concept for the concept of localising gate interconnection faults. Most common FSM platforms are based on logic circuits or PLDs, the research approach taken here was to confine the FSM utilising platform into a memory-only implementation thus reducing the necessary execution logic to a bare minimum. Fault-tolerant features specific to protecting data within memory have been explored based on a continuous approach that operates on the entire data set. Further cross-correlated parity checking was designed that combines the detection of dormant data faults and direct localisation and identification of the incorrect data bit.

A strategy was developed for transforming a given FSM functionality into memory-only-based platform through the observation that its state transition table contains the starting information for this. By applying a unique coding to this data a uniform and independently addressable data structure was developed. An example of an FSM soda machine application showed the following advantages over a non-memory-only-based FSM implementation: i) the implementation was done in memory-only and compared to a conventional PLD implementation; ii) the comparison showed that the state transitions of the memory-based version could be executed within a fixed time for every transition while the PLD implementation exhibited inconsistent state transition timings and, iii) the memory usage for the memory-based implementation was 4.92 times less than the PLD-based implementation. Point ii demonstrates the potential for EDC-protected memory-only-based FSM platforms for real-time applications. A further adaptation of this concept into a novel CAM design allowed the further reduction of memory utilisation and advanced fault-tolerance, which were built upon the use of SAFR type logic gates.

The final concept considered in this thesis is based on localising gate interconnection faults occurring between logic gates within a given logic structure by a process of cross-referencing input stimuli and output data within a memory-only FSM. Logic gate faults and gate interconnection faults may be distinguished by observing the SAFR gate current together with the overall output. This was demonstrated in Chapter 9 using the C17 standard benchmark circuit wherein the SAH and SAL fault-injection simulation revealed 59.74% of all faults produced an incorrect output result. The remaining faults were masked within the circuit, thus indicating that the circuit masked 40.26% interconnection faults. The data further indicated that each of these faults can be associated with a unique input stimulus and output value, which revealed that each individual fault may be specified through input stimuli and output value sets. Thus FSMs can be designed that are capable of localising single faulty gate interconnections and triggering the necessary self-reconfiguration that acts to exclude the offending interconnection. Importantly, this concept successfully achieves self-initiated fault-localisation and repair without reliance upon external checking systems, a core requirement considered at the outset of the thesis with respect to all self-healing systems whether biological or artificial.

10.2. Further work

The research work introduced within this thesis builds upon related research in the area of redundant design and identifies building blocks for creating self-healing logic structures through the steps of theoretical analysis, implementation and fault response evaluation. However, as in any scientific research, ongoing improvements and advancements are still possible. Potential directions to be followed include:

i. Design of building blocks of SAFR-type logic gates

The generic structure of the three fundamental logic gates NOT, NAND and NOR are designed, analysed and simulated. The next step is to create a set of these gates on a defined structure with fixed input/output/fault-injection interfaces. These building blocks can then be used for the creation of fundamental logic circuits for the verification of their fault-tolerant behaviour, the altered design being done through SAFR type logic gates.

ii. Design of self-healing features for the QLC matrix structure:

The present design of the QLC element in conjunction with analysing circuity is capable of identifying the faulty logic unit within its own structure. The follow-on steps of self-healing in this particular logic unit require a scaling to larger design involving multiple QLC design cells based on

the individual QLC element considered here. Spare logic resources could be formed within this matrix structure and would need to be shared between neighbouring QLC elements to minimise hardware overhead. This idea is one possible way for creating a fault-tolerant self-healing QLC fixed matrix structure. A further idea is to create a 'sea' of individual logic and control units which during runtime is used to create the requisite logic structure for different QLC elements through the allocation of nearby fault-free logic units. This allocation follows predefined rules and, in the case of a fault within one logic unit, the reallocation of the remaining fault-free logic units for the logic structure is triggered.

iii. Design of custom System-on-Chip (SoC) Platform for SAFR-type logic gate array:

The present designs of SAFR type logic gates exist as individual circuits and not as a working integrated chip. The future work proposal in (i) will help to explore the limitations and functionalities of larger-scale SAFR type logic gate circuits and should be used within chip circuit simulations before the final array structure is designed. The resulting array or matrix-type structure would contain configurable interconnections and the fundamental-type logic gates. These fundamental logic gates could in turn be replaced by the logic unit structure designed in Chapter 7, which offers configurable logic functionality, with further analysis opportunities using EDA tools.

iv. Design of custom SoC Platform for QLC:

The SoC concept could be further extended to include QLC arrays wherein SAFR gates are incorporated according to the methods described in Chapter 6. With this SoC chip different standard logic circuits could be configured to evaluate the performance of the QLC structure. Through fault-injection capabilities added to the chip, comprehensive fault-behaviour simulation could be carried out and analysed.

References:

[1] K. Ruhl. (2010). *An introduction to Space Weather*. Available: http://www.land-of-kain.de/docs/spaceweather/

[2] J. Cong and X. Bingjun, "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration," in *Nanoscale Architectures (NANOARCH), 2011 IEEE/ACM International Symposium on*, 2011, pp. 1-8.

[3] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," *Found. Trends Electron. Des. Autom.,* vol. 2, pp. 135-253, 2008.

[4] ivc. (2011). *Logic Devices*. Available: http://beta.ivc.no/blog/2011/03/30/logic-devices/

[5] M. Niknahad, *Using Fine Grain Approaches for Highly Reliable Design of FPGA-based Systems in Space* vol. 9: KIT Scientific Publishing, 2013.

[6] A. Doumar and H. Ito, "Detecting, diagnosing, and tolerating faults in SRAM-based field programmable gate arrays: a survey," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 11, pp. 386-405, 2003.

[7] G. Cellere, P. Pellati, A. Chimenton, J. Wyss, A. Modelli, L. Larcher*, et al.*, "Radiation effects on floating-gate memory cells," *Nuclear Science, IEEE Transactions on,* vol. 48, pp. 2222-2228, 2001.

[8] A. Bavavik, "Solar wind and earth's magnetic field," ed.

[9] Booyabazooka, "Van Allen radiation belt," ed. wikipedia.

[10] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE,* vol. 25, pp. 10-16, 2005.

[11] S. Ningfang, Q. Jiaomei, P. Xiong, and D. Yan, "Fault injection methodology and tools," in *Electronics and Optoelectronics (ICEOE), 2011 International Conference on*, 2011, pp. V1-47-V1-50.

[12] R. Kothe, H. T. Vierhaus, T. Coym, W. Vermeiren, and B. Straube, "Embedded Self Repair by Transistor and Gate Level Reconfiguration," in *Design and Diagnostics of Electronic Circuits and systems, 2006 IEEE*, 2006, pp. 208-213.

[13] E. A. Amerasekera and F. N. Najm, *Failure mechanisms in semiconductor devices*: J. Wiley, 1997.

[14] R. Blish and N. Durrant, "Semiconductor device reliability failure models," *Int. Sematech. Technol. Transfer,* 2000.

[15] K. Way and K. Taeho, "An overview of manufacturing yield and reliability modeling for semiconductor products," *Proceedings of the IEEE,* vol. 87, pp. 1329-1344, 1999.

[16] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *Device and Materials Reliability, IEEE Transactions on,* vol. 5, pp. 305-316, 2005.

[17] S. Mukherjee, *Architecture design for soft errors*, 2008.

[18] T. Ban and L. A. de Barros Naviner, "A simple fault-tolerant digital voter circuit in TMR nanoarchitectures," in *NEWCAS Conference (NEWCAS), 2010 8th IEEE International*, 2010, pp. 269-272.

[19] K. S. Morgan, D. L. McMurtrey, B. H. Pratt, and M. J. Wirthlin, "A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs," *Nuclear Science, IEEE Transactions on,* vol. 54, pp. 2065-2072, 2007.

[20] A. Djupdal and P. C. Haddow, "Evolving efficient redundancy by exploiting the analogue nature of CMOS transistors," in *Fourth International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS)*, 2007, pp. 81-86.

[21] S. Naran, B. C. Paul, and K. Roy, "Enhancing yield at the end of the technology roadmap," *Design & Test of Computers, IEEE,* vol. 21, pp. 563-571, 2004.

[22] A. H. El-Maleh, B. M. Al-Hashimi, A. Melouki, and F. Khan, "Defect-tolerant n^2-transistor structure for reliable nanoelectronic designs," *Computers & Digital Techniques, IET,* vol. 3, pp. 570-580, 2009.

[23] S. Mitra, H. Wei-Je, N. R. Saxena, S. Y. Yu, and E. J. McCluskey, "Reconfigurable architecture for autonomous self-repair," *Design & Test of Computers, IEEE,* vol. 21, pp. 228-240, 2004.

[24] H. Wei-Je, S. Mitra, and E. J. McCluskey, "Fast run-time fault location in dependable FPGA-based applications," in *Defect and Fault Tolerance in VLSI Systems, 2001. Proceedings. 2001 IEEE International Symposium on*, 2001, pp. 206-214.

[25] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 6, pp. 212-221, 1998.

[26] T. Koal, D. Scheit, Scho, x, M. lzel, and H. T. Vierhaus, "On the Feasibility of Built-In Self Repair for Logic Circuits," in *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2011 IEEE International Symposium on*, 2011, pp. 316-324.

[27] *ISCAS85 Benchmark circuits*. Available: http://www.cbl.ncsu.edu/benchmarks/ISCAS85/

[28] R. Leveugle, R. Rochet, G. Saucier, L. Martinez, and C. Pitot, "A synthesis tool for fault-tolerant finite state machines," in *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, 1993, pp. 502-511.

[29] N. I. Rafla and B. L. Davis, "A Study of Finite State Machine Coding Styles for Implementation in FPGAs," in *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*, 2006, pp. 337-341.

[30] P. Schiefer, R. McWilliam, and A. Purvis, "Creating a Self-configuring Finite State Machine out of Memory Look-up Tables," *Procedia CIRP,* vol. 11, pp. 363-366, // 2013.

[31] P. Schiefer, R. McWilliam, and A. Purvis, "Self-healing Fuel Pump Controller Mapped into Memory Based Finite State Machine," *Procedia CIRP,* vol. 22, pp. 132-137, // 2014.

[32] D. R. Wright. (Summer 2005). *CSC216 Finite State Machines*. Available: http://www4.ncsu.edu/~drwrigh3/docs/courses/csc216/fsm-notes.pdf

[33] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *Solid-State Circuits, IEEE Journal of,* vol. 41, pp. 712-727, 2006.

[34] W. contributors. *Axolotl*. Available: http://en.wikipedia.org/w/index.php?title=Axolotl&oldid=655300593

[35] L. L. Technology. *LT1366 - Dual and Quad Precision Rail-to-Rail Input and Output Op Amps*. Available: http://cds.linear.com/docs/en/datasheet/1366fb.pdf

[36] R. L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits," *Bell System Technical Journal,* vol. 57, pp. 1449-1474, 1978.

[37] W. H. Pierce, "Interwoven redundant logic," *Journal of the Franklin Institute,* vol. 277, pp. 55-85, 1964.

[38] W. contributors. *Logikpegel*. Available: http://de.wikipedia.org/wiki/Logikpegel#Literatur

[39] R. R. Schaller, "Moore's law: past, present and future," *Spectrum, IEEE,* vol. 34, pp. 52-59, 1997.

[40] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin, "Dynamic reconfiguration for management of radiation-induced faults in FPGAs," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, p. 145.

[41] M. A. Breuer, S. K. Gupta, and T. M. Mak, "Defect and error tolerance in the presence of massive numbers of defects," *Design & Test of Computers, IEEE,* vol. 21, pp. 216-227, 2004.

[42] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: a large-scale field study," *SIGMETRICS Perform. Eval. Rev.,* vol. 37, pp. 193-204, 2009.

[43] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," presented at the Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, 2003.

[44] E. Normand, "Single event upset at ground level," *Nuclear Science, IEEE Transactions on,* vol. 43, pp. 2742-2750, 1996.

[45] T. Heijmen, "Radiation-induced soft errors in digital circuits--A literature survey," 2002.

[46] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, 2002, pp. 389-398.

[47] J. A. Walker, R. Sinnott, G. Stewart, J. A. Hilder, and A. M. Tyrrell, "Optimizing electronic standard cell libraries for variability tolerance through the nano-CMOS grid," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,* vol. 368, pp. 3967-3981, August 28, 2010 2010.

[48] K. Parnell and R. Bryner, "Comparing and contrasting FPGA and microprocessor system design and development," *White Paper, XILINX Corporation,* 2004.

[49] T. J. Todman, G. A. Constantinides, S. J. Wilton, O. Mencer, W. Luk, and P. Y. Cheung, "Reconfigurable computing: architectures and design methods," *IEE Proceedings-Computers and Digital Techniques,* vol. 152, pp. 193-207, 2005.

[50] T. L. Turflinger, M. V. Davey, and J. P. Bings, "Radiation effects in analog CMOS analog-to-digital converters," in *Radiation Effects Data Workshop, 1996., IEEE*, 1996, pp. 6-12.

[51] M. V. O'Bryan, K. A. LaBel, R. L. Ladbury, C. Poivey, J. W. Howard, R. A. Reed*, et al.*, "Current single event effects and radiation damage results for candidate spacecraft electronics," in *Radiation Effects Data Workshop, 2002 IEEE*, 2002, pp. 82-105.

[52] AMD, "State Machine Design " June 1993 1993.

[53] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal,* vol. 34, pp. 1045-1079, 1955.

[54] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata studies,* vol. 34, pp. 129-153, 1956.

[55] R. Senhadji-Navarro, I. Garcia-Vargas, and J. L. Guisado, "Performance evaluation of RAM-based implementation of Finite State Machines in FPGAs," in *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*, 2012, pp. 225-228.

[56] L. Frigerio and F. Salice, "RAM-based fault tolerant state machines for FPGAs," in *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on*, 2007, pp. 312-320.

[57] P. Horowitz and W. Hill, *The Art of Electronics*: Cambridge University Press, 1989.

[58] H. Ando, Y. Yoshida, A. Inoue, I. Sugiyama, T. Asakawa, K. Morita*, et al.*, "A 1.3 GHz fifth generation SPARC64 microprocessor," in *Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC. 2003 IEEE International*, 2003, pp. 246-491 vol.1.

[59] S. Brown and J. Rose, "FPGA and CPLD architectures: A tutorial," *IEEE design & test of computers,* vol. 13, pp. 42-57, 1996.

[60] E. J. McDonald, "Runtime FPGA Partial Reconfiguration," in *Aerospace Conference, 2008 IEEE*, 2008, pp. 1-7.

[61] F. Berthelot, F. Nouvel, and D. Houzet, "Partial and dynamic reconfiguration of FPGAs: a top down design methodology for an automatic implementation," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, p. 4 pp.

[62] B. Osterloh, H. Michalik, S. A. Habinc, and B. Fiethe, "Dynamic Partial Reconfiguration in Space Applications," in *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, 2009, pp. 336-343.

[63] B. Schulz, C. Paiz, J. Hagemeyer, S. Mathapati, M. Porrmann, and J. Bocker, "Run-time reconfiguration of FPGA-based drive controllers," in *Power Electronics and Applications, 2007 European Conference on*, 2007, pp. 1-10.

[64] R. Senhadji-Navarro, I. Garcia-Vargas, G. Jimenez-Moreno, and A. Civit-Ballcels, "ROM-based FSM implementation using input multiplexing in FPGA devices," *Electronics Letters,* vol. 40, pp. 1249-1251, 2004.

[65] T. Łuba, K. Gorski, and L. Wroński, "ROM-based Finite State Machines with PLA address modifiers," in *Proceedings of the conference on European design automation*, 1992, pp. 272-277.

[66] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC v8 architecture," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, 2002, pp. 409-415.

[67] K. Reick, P. N. Sanda, S. Swaney, J. W. Kellington, M. J. Mack, M. S. Floyd*, et al.*, "Fault-Tolerant Design of the IBM Power6 Microprocessor," *Micro, IEEE,* vol. 28, pp. 30-38, 2008.

[68] Xilinx, "Xilinx UG360 Virtex-6 FPGA Configuration User Guide," *User Guide,* 2009.

[69] Kilopass. *2T Bitcell*. Available: http://www.kilopass.com/technology/2t-bitcell/

[70] D. N. Nguyen, S. M. Guertin, G. M. Swift, and A. H. Johnston, "Radiation effects on advanced flash memories," *Nuclear Science, IEEE Transactions on,* vol. 46, pp. 1744-1750, 1999.

[71] Z. Quming and K. Mohanram, "Gate sizing to radiation harden combinational logic," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 25, pp. 155-166, 2006.

[72] NASA. *NASA/GSFC Radiation Effects & Analysis Home Page*. Available: http://radhome.gsfc.nasa.gov/

[73] W. contributors. *Sun*. Available: http://en.wikipedia.org/wiki/Sun

[74] W. contributors. *Van Allen radiation belt*. Available: http://en.wikipedia.org/wiki/Van_Allen_radiation_belt

[75] R. Baumann, "Soft errors in advanced computer systems," *Design & Test of Computers, IEEE,* vol. 22, pp. 258-266, 2005.

[76] M. Nicolaidis, *Soft Errors in Modern Electronic Systems* vol. 1: Springer, 2010.

[77] P. Hazucha, T. Karnik, J. Maiz, S. Walstra, B. Bloechel, J. Tschanz*, et al.*, "Neutron soft error rate measurements in a 90-nm CMOS process and scaling trends in SRAM from 0.25-µm to 90-nm generation," in *Electron Devices Meeting, 2003. IEDM '03 Technical Digest. IEEE International*, 2003, pp. 21.5.1-21.5.4.

[78] Altera. *Single Event Upsets*. Available: http://www.altera.com/support/reliability/seu/seu-index.html

[79] R. Mariani, P. Fuhrmann, and B. Vittorelli, "Fault-robust microcontrollers for automotive applications," in *On-Line Testing Symposium, 2006. IOLTS 2006. 12th IEEE International*, 2006, p. 6 pp.

[80] B. H. Pratt, *Analysis and Mitigation of SEU-induced Noise in FPGA-based DSP Systems*: BRIGHAM YOUNG UNIVERSITY, 2011.

[81] J. Gaisler, "LEON3-FT-RTAX SEU Test results," ed: Issue, 2005.

[82] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. H. Leber, "Comparison of physical and software-implemented fault injection techniques," *Computers, IEEE Transactions on,* vol. 52, pp. 1115-1133, 2003.

[83] A. E. Waskiewicz, J. W. Groninger, V. H. Strahan, and D. M. Long, "Burnout of Power MOS Transistors with Heavy Ions of Californium-252," *Nuclear Science, IEEE Transactions on,* vol. 33, pp. 1710-1713, 1986.

[84] U. Gunneflo, J. Karlsson, and J. Torin, "Evaluation of error detection schemes using fault injection by heavy-ion radiation," in *Fault-Tolerant Computing, 1989. FTCS-19. Digest of Papers., Nineteenth International Symposium on*, 1989, pp. 340-347.

[85] D. Alnajjar, H. Kounoura, Y. Mitsuyama, M. Hashimoto, and T. Onoye, "MTTF Measurement under Alpha Particle Radiation in a Coarse-Grained Reconfigurable Architecture with Flexible Reliability," in *Proc. 2011 IEEE Workshop on Silicon Errors in Logic-System Effects*, 2011.

[86] C. Chia-Hsiang, P. Knag, and Z. Zhengya, "Characterization of Heavy-Ion-Induced Single-Event Effects in 65nm Bulk CMOS ASIC Test Chips," *Nuclear Science, IEEE Transactions on,* vol. 61, pp. 2694-2701, 2014.

[87] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey, "Static Proton and Heavy Ion Testing of the Xilinx Virtex-5 Device," in *Radiation Effects Data Workshop, 2007 IEEE*, 2007, pp. 177-184.

[88] A. Namazi and M. Nourani, "Gate-Level Redundancy: A New Design-for-Reliability Paradigm for Nanotechnologies," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 18, pp. 775-786, 2010.

[89] C. Hu, "Future CMOS scaling and reliability," *Proceedings of the IEEE,* vol. 81, pp. 682-689, 1993.

[90] J. R. Carter, S. Ozev, and D. J. Sorin, "Circuit-Level Modeling for Concurrent Testing of Operational Defects due to Gate Oxide Breakdown," presented at the Proceedings of the conference on Design, Automation and Test in Europe - Volume 1, 2005.

[91] E. Stott, P. Sedcole, and P. Cheung, "Fault tolerant methods for reliability in FPGAs," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, 2008, pp. 415-420.

[92] B. Joshi, D. Pradhan, and S. Mohanty, "Fault Tolerant Nanocomputing," in *Robust Computing with Nano-scale Devices*. vol. 58, C. Huang, Ed., ed: Springer Netherlands, 2010, pp. 7-27.

[93] P. O'Connor and A. Kleyner, *Practical reliability engineering*: John Wiley & Sons, 2011.

[94] C. Scherrer and A. Steininger, "Dealing with dormant faults in an embedded fault-tolerant computer system," *Reliability, IEEE Transactions on,* vol. 52, pp. 512-522, 2003.

[95] X. Iturbe, M. Azkarate, I. Martinez, J. Perez, and A. Astarloa, "A novel SEU, MBU and SHE handling strategy for Xilinx Virtex-4 FPGAs," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009, pp. 569-573.

[96] C. M. K. Israel Koren, *Fault-Tolerant Systems*, 2007.

[97] C. Constantinescu, "Intermittent faults in VLSI circuits," in *Proceedings of the IEEE Workshop on Silicon Errors in Logic-System Effects*, 2007.

[98] S. S. Mukherjee, J. Emer, T. Fossum, and S. K. Reinhardt, "Cache scrubbing in microprocessors: myth or necessity?," in *Dependable Computing, 2004. Proceedings. 10th IEEE Pacific Rim International Symposium on*, 2004, pp. 37-42.

[99] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies,* vol. 34, pp. 43-98, 1956.

[100] K. Kyriakoulakos and D. Pnevmatikatos, "A novel SRAM-based FPGA architecture for efficient TMR fault tolerance support," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009, pp. 193-198.

[101] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini, "Fault-tolerant platforms for automotive safety-critical applications," presented at the Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems, San Jose, California, USA, 2003.

[102] P. A. Jensen, "Quadded NOR Logic," *Reliability, IEEE Transactions on,* vol. R-12, pp. 22-31, 1963.

[103] A. E. Barbour and A. S. Wojcik, "A general constructive approach to fault-tolerant design using redundancy," *Computers, IEEE Transactions on,* vol. 38, pp. 15-29, 1989.

[104] A. E. Barbour, "Solutions to the minimization problem of fault-tolerant logic circuits," *Computers, IEEE Transactions on,* vol. 41, pp. 429-443, 1992.

[105] M. Niknahad, O. Sander, and J. Becker, "Fine grain fault tolerance- A key to high reliability for FPGAs in space," in *Aerospace Conference, 2012 IEEE*, 2012, pp. 1-10.

[106] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *Nuclear Science, IEEE Transactions on,* vol. 51, pp. 2957-2969, 2004.

[107] M. Niknahad, O. Sander, and J. Becker, "A study on fine granular fault tolerance methodologies for FPGAs," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, 2011, pp. 1-5.

[108] S. Habinc, "Functional Triple Modular Redundancy (FTMR)," *Design and Assessment Report, Gaisler Research, FPGA-003-01, ver. 0.2,* pp. 1-55, 2002.

[109] K. Nikolic, A. Sadek, and M. Forshaw, "Fault-tolerant techniques for nanocomputers," *Nanotechnology,* vol. 13, p. 357, 2002.

[110] H. Yuang-Ming and E. E. Swartzlander, "Time redundant error correcting adders and multipliers," in *Defect and Fault Tolerance in VLSI Systems, 1992. Proceedings., 1992 IEEE International Workshop on*, 1992, pp. 247-256.

[111] K. G. Shin and K. Hagbae, "A time redundancy approach to TMR failures using fault-state likelihoods," *Computers, IEEE Transactions on,* vol. 43, pp. 1151-1162, 1994.

[112] W. J. Townsend, J. A. Abraham, and E. E. Swartzlander, "Quadruple time redundancy adders [error correcting adder]," in *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*, 2003, pp. 250-256.

[113] R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Technical Journal,* vol. 29, pp. 147-160, 1950.

[114] J. Emmert, C. Stroud, B. Skaggs, and M. Abramovici, "Dynamic fault tolerance in FPGAs via partial reconfiguration," in *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, 2000, pp. 165-174.

[115] C. Bolchini, A. Miele, and M. D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs," in *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on*, 2007, pp. 87-95.

[116] E. Johnson, M. J. Wirthlin, and M. Caffrey, "Single-event upset simulation on an FPGA," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2002, pp. 68-73.

[117] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," *Nuclear Science, IEEE Transactions on,* vol. 52, pp. 2438-2445, 2005.

[118] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," *Xilinx Corporation,* 2000.

[119] J. G. Tryon, "Redundant logic circuitry," ed: Google Patents, 1960.

[120] J. G. Tryon, *"Quadded Logic" in Redundancy Techniques for Computing Systems, Ed. R.H. Wilcox and W.C. Mann*: Spartan Books, 1962.

[121] I. Rectifier, "Datasheet IRFD9024PbF," ed.

[122] I. Rectifier, "Datasheet IRFD020," ed.

[123] P. Schiefer, R. McWilliam, and A. Purvis, "Fault Tolerant Quadded Logic Cell Structure with Built-in Adaptive Time Redundancy," *Procedia CIRP,* vol. 22, pp. 127-131, // 2014.

[124] R. McWilliam, P. Schiefer, and A. Purvis, "Experimental Validation of a Resilient Electronic Logic Design with Autonomous Fault Discrimination/Masking," *Procedia CIRP,* vol. 38, pp. 265-270, // 2015.

[125] I. Garcia-Vargas, R. Senhadji-Navarro, G. Jimenez-Moreno, A. Civit-Balcells, and P. Guerra-Gutierrez, "ROM-Based Finite State Machine Implementation in Low Cost FPGAs," in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, 2007, pp. 2342-2347.

[126] W. contributors. *Turing machine*. Available: https://en.wikipedia.org/wiki/Turing_machine

[127] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," *Proceedings of the IEEE,* vol. 84, pp. 1090-1123, 1996.

[128] W. contributors. *Flip-flop (electronics)*. Available: http://en.wikipedia.org/wiki/Flip-flop_%28electronics%29

[129] X. Wendling, R. Rochet, and R. Leveugle, "ROM-based synthesis of fault-tolerant controllers," in *Defect and Fault Tolerance in VLSI Systems, 1996. Proceedings., 1996 IEEE International Symposium on*, 1996, pp. 304-308.

[130] K. Pagiamtzis, N. Azizi, and F. N. Najm, "A Soft-Error Tolerant Content-Addressable Memory (CAM) Using An Error-Correcting-Match Scheme," in *Custom Integrated Circuits Conference, 2006. CICC '06. IEEE*, 2006, pp. 301-304.

[131] F. d. N. Kucinskis and M. G. V. Ferreira, "Taking the ECSS Autonomy Concepts One Step Further."

[132] Philips. (1997). *Datasheet BSP230 P-channel enhancement mode vertical D-MOS transistor*. Available: http://www.nxp.com/documents/data_sheet/BSP230.pdf

[133] Philips. (2002). *Datasheet BSP126 N-channel enhancement mode vertical D-MOS transistor*. Available: http://www.nxp.com/documents/data_sheet/BSP126.pdf

[134] Littelfuse. *Axial Lead & Cartridge Fuses 5x20mm > Fast-Acting > 216 Series*. Available: http://www.farnell.com/datasheets/1886806.pdf

[135] L. O. Chua, "Memristor-The missing circuit element," *Circuit Theory, IEEE Transactions on,* vol. 18, pp. 507-519, 1971.

[136] P. Mazumder, S. M. Kang, and R. Waser, "Memristors: devices, models, and applications," *Proceedings of the IEEE,* vol. 100, pp. 1911-1919, 2012.

[137] C. Kothandaraman, S. K. Iyer, and S. S. Iyer, "Electrically programmable fuse (eFUSE) using electromigration in silicides," *Electron Device Letters, IEEE,* vol. 23, pp. 523-525, 2002.

[138] N. Robson, J. Safran, C. Kothandaraman, A. Cestero, C. Xiang, R. Rajeevakumar*, et al.*, "Electrically Programmable Fuse (eFUSE): From Memory Redundancy to Autonomic Chips," in *Custom Integrated Circuits Conference, 2007. CICC '07. IEEE*, 2007, pp. 799-804.

Appendix:

Appendix 1: Publications

Edited works: journals

R. McWilliam, P. Schiefer, and A. Purvis, Experimental Validation of a Resilient Electronic Logic Design with Autonomous Fault Discrimination/Masking, *Procedia CIRP*, vol. 38, pp. 265-270, // 2015.

Schiefer, Philipp, McWilliam, Richard & Purvis, Alan (2014) Fault Tolerant Quadded Logic Cell Structure with Built-in Adaptive Time Redundancy *Procedia CIRP* 22: 127-131

Schiefer, Philipp, McWilliam, Richard & Purvis, Alan (2014) Self-healing Fuel Pump Controller Mapped into Memory-based Finite State Machine *Procedia CIRP* 22: 132-137

Schiefer, Philipp, McWilliam, Richard & Purvis, Alan (2013) Creating a Self-configuring Finite State Machine out of Memory Look-up Tables *Procedia CIRP* 11: 363-366

McWilliam, Richard, Schiefer, Philipp & Purvis, Alan (2013) Demonstration of Self-recovering ALU Using a Convergent Cellular Automata *Procedia CIRP* 11: 373-378

Conference papers

McWilliams, R, Frei, R, Schiefer, P, Purvis, A, Tiwari, A & Zhu, M (2012), Self-repair technologies for maintenance-free mechanical and electronic components and subsystems, 1st EPSRC Conference on Manufacturing the Future Loughborough, England, EPSRC

McWilliams, R, Purvis, A, Jones, D, Schiefer, P, Tiwari, A & Zhu, M (2012), Self-repairing Electronic Logic Units Based on Convergent Cellular Automata 1st International Conference on Through-life Engineering Services Shrivenham, England, Cranfield University Press, 353-360

Books: sections

McWilliam, Richard, Schiefer, Philipp & Purvis, Alan (2015) Building Dependable Electronic Systems for Autonomous Maintenance In Through-life Engineering Services Redding, Louis & Roy, Rajkumar Springer International Publishing. 375-394

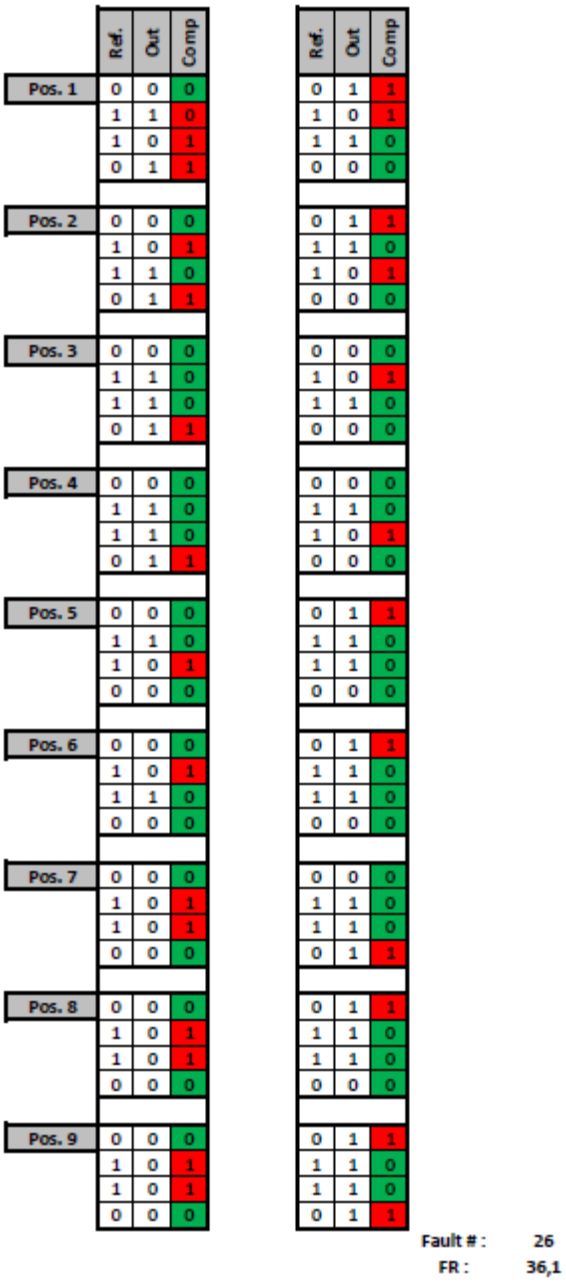
In-progress / accepted articles:

McWilliam R, Schiefer P and Purvis A, Creating self-configuring logic with built-in resilience to multiple-upset events Proc of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture (accepted, awaiting DOI)

McWilliam, R., Schiefer P., Purvis. A., Tiwari, A., Designing self-maintenance into electronics: a survey of hardware techniques, (Submitted to ACM Surveys)

Appendix 2: Example of FR calculation for SAH and SAL fault injection into a XOR logic gate structure in accordance with Figure 5.9(a)

With this example the evaluation of the FR calculation, which is the basis for the analysis, is performed on the XOR logic gate of Figure 5.9(b). At each injection point SAH and SAL faults are going to be injected and all possible input combinations are getting applied. Every deviation is marked with red and the total fault number matches the number shown in Table 5.2.



	Ref.	Out	Comp	Ref.	Out	Comp
Pos. 1	0	0	0	0	1	1
	1	1	0	1	0	1
	1	0	1	1	1	0
	0	1	1	0	0	0
Pos. 2	0	0	0	0	1	1
	1	0	1	1	1	0
	1	1	0	1	0	1
	0	1	1	0	0	0
Pos. 3	0	0	0	0	0	0
	1	1	0	1	0	1
	1	1	0	1	1	0
	0	1	1	0	0	0
Pos. 4	0	0	0	0	0	0
	1	1	0	1	1	0
	1	1	0	1	0	1
	0	1	1	0	0	0
Pos. 5	0	0	0	0	1	1
	1	1	0	1	1	0
	1	0	1	1	1	0
	0	0	0	0	0	0
Pos. 6	0	0	0	0	1	1
	1	0	1	1	1	0
	1	1	0	1	1	0
	0	0	0	0	0	0
Pos. 7	0	0	0	0	0	0
	1	0	1	1	1	0
	1	0	1	1	1	0
	0	0	0	0	1	1
Pos. 8	0	0	0	0	1	1
	1	0	1	1	1	0
	1	0	1	1	1	0
	0	0	0	0	0	0
Pos. 9	0	0	0	0	1	1
	1	0	1	1	1	0
	1	0	1	1	1	0
	0	0	0	0	1	1

Fault # : 26
FR : 36,1

Appendix 3.1: MATLAB program for Chapter 4 for the FR generation data of the majority voter under the influence of stuck-at fault injected at specified injection points

Within this MATLAB program the majority voter under the influence of stuck-at high or low faults is getting simulated. The entire possible combinational inputs sequence is getting applied onto the majority voter for one injected fault. Each set of results is getting compared against a known good set of results for the identification of the incorrect output sequences. The fault-injection location and the type of fault can be specified at the time of the function call.

```
function [out] = voter01 (st,lo)
in = [0 0 0;0 0 1;0 1 0;0 1 1;1 0 0;1 0 1;1 1 0;1 1 1];
ou = [0;0;0;1;0;1;1;1];

z = [];
z1 = size(in); le = z1(1);

for i = 1:le
   x1 = in(i,1); x2 = in(i,2); x3 = in(i,3);
   if (st==1) x1 = lo; end
   if (st==2) x2 = lo; end
   if (st==3) x3 = lo; end
   x11 = x1; x21 = x2;
   if (st==4) x11 = lo; end
   if (st==5) x21 = lo; end
   y1 = x11&x21;
   if (st==10) y1 = lo; end
   x12 = x1; x32 = x3;
   if (st==6) x12 = lo; end
   if (st==7) x32 = lo; end
   y2 = x12&x32;
   if (st==11) y2 = lo; end
   x23 = x2; x33 = x3;
   if (st==8) x23 = lo; end
   if (st==9) x33 = lo; end
   y3 = x23&x33;
   if (st==12) y3 = lo; end
   y = y1|y2|y3;
   if (st==13) y = lo; end
   z = [z;y];
end
z = [ou z];
out = z;
end
```

Appendix 3.2: MATLAB program for Chapter 5 for the FR generation data of the logic circuits XOR-gate and quadded logic version of the XOR function under the influence of stuck-at fault injected at specified injection points

```
function [out] = xorlogic02 (lo,st)
in = [0 0;0 1;1 0;1 1];
ou = [0;1;1;0];

z = [];
z1 = size(in); le = z1(1);

for i = 1:le
   x1 = in(i,1); x2 = in(i,2); x3 = in(i,1); x4 = in(i,2);
   if (lo==1) x1 = st; x3 = st; end
   if (lo==2) x2 = st; x4 = st; end
   if (lo==3) x1 = st; end
   if (lo==4) x2 = st; end
   if (lo==5) x3 = st; end
   if (lo==6) x4 = st; end
   y1 = ~(x1&x2); y2 = x3|x4;
   if (lo==7) y1 = st; end
   if (lo==8) y2 = st; end
   y = y1&y2;
   if (lo==9) y = st; end
   z = [z;y];
end
z = [ou z];
out = z;
end

function [out] = quaddedlogic02 (lo,st)
in = [0 0;0 1;1 0;1 1];
ou = [0;0;0;0;1;1;1;1;1;1;1;1;0;0;0;0];

z = [];
z1 = size(in); le = z1(1);

for i = 1:le
   x1 = [in(i,1) in(i,1) in(i,1) in(i,1)];
   x2 = [in(i,2) in(i,2) in(i,2) in(i,2)];
   p = zeros(1,4); q = zeros(1,4); o = zeros(1,4);
   if (lo>=1)&&(lo<=4) x1(1) = st; end
   if (lo>=5)&&(lo<=8) x2(1) = st; end
   xn = [x1(1) x1(2) x2(1) x2(2) x1(2) x1(1) x2(2) x2(1) x1(3) x1(4) x2(3) x2(4) x1(4) x1(3) x2(4) x2(3)];
   if (lo>=9)&&(lo<=24) xn(1,lo-8) = st; end
   p(1) = ~(xn(1)&xn(2)&xn(3)&xn(4));
   p(2) = ~(xn(5)&xn(6)&xn(7)&xn(8));
   p(3) = ~(xn(9)&xn(10)&xn(11)&xn(12));
   p(4) = ~(xn(13)&xn(14)&xn(15)&xn(16));
   xn = [x1(1) x1(2) x2(1) x2(2) x1(2) x1(1) x2(2) x2(1) x1(3) x1(4) x2(3) x2(4) x1(4) x1(3) x2(4) x2(3)];
   if (lo>=25)&&(lo<=40) xn(1,lo-24) = st; end
```

```
   q(1) = (xn(1)|xn(2)|xn(3)|xn(4));
   q(2) = (xn(5)|xn(6)|xn(7)|xn(8));
   q(3) = (xn(9)|xn(10)|xn(11)|xn(12));
   q(4) = (xn(13)|xn(14)|xn(15)|xn(16));
   if (lo>=41)&&(lo<=44) p(1,lo-40) = st; end
   if (lo>=45)&&(lo<=48) q(1,lo-44) = st; end
   yn = [p(1) p(4) q(1) q(4) p(2) p(3) q(2) q(3) p(3) p(2) q(3) q(2) p(4) p(1) q(4) q(1)];
   if (lo>=49)&&(lo<=64) yn(1,lo-48) = st; end
   o(1) = yn(1)&yn(2)&yn(3)&yn(4);
   o(2) = yn(5)&yn(6)&yn(7)&yn(8);
   o(3) = yn(9)&yn(10)&yn(11)&yn(12);
   o(4) = yn(13)&yn(14)&yn(15)&yn(16);
   if (lo>=65)&&(lo<=68) o(1,lo-64) = st; end
   y = [o(1);o(2);o(3);o(4)];
   z = [z;y];
end
z = [ou z];
out = z;
end

function [out] = nandfailvoter (st)

ls = size(st);
x1 = ls(1);
x2 = ls(2);
z = [];

for i = 1:x1
   if (xor(st(i,1),st(i,3)))
      y = st(i,1:x2);
      z = [z;y];
   end
end
out = z;
end
```

Appendix 3.3: MATLAB program for Chapter 6 for the FR generation data of the comparison of the fault-behaviour of the generic logic gate structure and the QLC structure under the influence of stuck-at fault injected at specified injection points

This MATLAB programs can be controlled through input values to perform different logic functionality within a fixed logic structure. The same logic selections can be used on the QLC structure for performing the same logic functionality. Both logic structures are subjected to stuck-at high and low fault-injection at specified injection points within the interconnection. The generic logic set-up FR data is the basis for the comparison against the QLC structure.

```
function [out] = genlogic01 (lo,st,l1,l2,l3)
in = [0 0 0 0;0 0 0 1;0 0 1 0;0 0 1 1;0 1 0 0;0 1 0 1;0 1 1 0;0 1 1 1;
    1 0 0 0;1 0 0 1;1 0 1 0;1 0 1 1;1 1 0 0;1 1 0 1;1 1 1 0;1 1 1 1];

z = [];
z1 = size(in); le = z1(1);

for i = 1:le
   x1 = in(i,1); x2 = in(i,2); x3 = in(i,3); x4 = in(i,4);
   if (l1==1) p1 = (x1&x2); end
   if (l1==2) p1 = ~(x1&x2); end
   if (l1==3) p1 = (x1|x2); end
   if (l1==4) p1 = ~(x1|x2); end
   if (l2==1) p2 = (x3&x4); end
   if (l2==2) p2 = ~(x3&x4); end
   if (l2==3) p2 = (x3|x4); end
   if (l2==4) p2 = ~(x3|x4); end
   if (l3==1) ou = (p1&p2); end
   if (l3==2) ou = ~(p1&p2); end
   if (l3==3) ou = (p1|p2); end
   if (l3==4) ou = ~(p1|p2); end
   if (lo==1) x1 = st; end
   if (lo==2) x2 = st; end
   if (lo==3) x1 = st; end
   if (lo==4) x2 = st; end
   if (l1==1) p1 = (x1&x2); end
   if (l1==2) p1 = ~(x1&x2); end
   if (l1==3) p1 = (x1|x2); end
   if (l1==4) p1 = ~(x1|x2); end
   if (l2==1) p2 = (x3&x4); end
   if (l2==2) p2 = ~(x3&x4); end
   if (l2==3) p2 = (x3|x4); end
   if (l2==4) p2 = ~(x3|x4); end
   if (lo==5) p1 = st; end
   if (lo==6) p2 = st; end
   if (l3==1) y1 = (p1&p2); end
   if (l3==2) y1 = ~(p1&p2); end
```

```
   if (l3==3) y1 = (p1|p2); end
   if (l3==4) y1 = ~(p1|p2); end
   if (lo==7) y1 = st; end
   l = l1*100+l2*10+l3;
   z = [z;l ou y1];
end
out = z;
end


function [out] = runqlc01 ()
z = []; ftag = 3;
for i1 = 1:4
   for i2 = 1:4
      for i3 = 1:4
         c2 = 0; c4 = 0;
         for ii = 1:15
            s = [i1 i2 i3];
            c1 = qlcmat01(s,ftag,ii,0); c2 = c2 + nandfail06(c1);
            c3 = qlcmat01(s,ftag,ii,1); c4 = c4 + nandfail06(c3);
         end
         cc = c2 + c4; cd = cc(2:3); ce = (sum(cd)/480)*100;
         z = [z;i1 i2 i3 cc ce];
      end
   end
end
out = z;
end
function [out] = qlcmat01 (s,fa,ft,sh)
in = [0 0 0 0;0 0 0 1;0 0 1 0;0 0 1 1;0 1 0 0;0 1 0 1;0 1 1 0;0 1 1 1;
    1 0 0 0;1 0 0 1;1 0 1 0;1 0 1 1;1 1 0 0;1 1 0 1;1 1 1 0;1 1 1 1];

z = [];
z1 = size(in); le = z1(1);

for i = 1:le
   x1 = in(i,1:4);
   yc1 = []; yc2 = []; y1 = Q_cell_matrix01(1,s,x1,0,0,0);
   for c = 1:4
      y2 = Q_cell_matrix01(c,s,x1,fa,ft,sh);
      yc1 = [yc1 y1]; yc2 = [yc2 y2];
   end
   z = [z;s yc1 yc2];
end
out = z;
end


function [out] = Q_cell_matrix01 (con,sel,inp,ftag,ift,ishl)
aft=0;bft=0;cft=0;dft=0;
ashl=0;bshl=0;cshl=0;dshl=0;

if(ftag == 1) aft=ift;ashl=ishl; end
```

```
if(ftag == 2) bft=ift;bshl=ishl; end
if(ftag == 3) cft=ift;cshl=ishl; end
if(ftag == 4) dft=ift;dshl=ishl; end

 h_i = [0,0];
 a1 = sel(1); a2 = sel(2); a3 = sel(3);
 b1 = [inp(1),inp(2)]; b2 = [inp(3),inp(4)];
 if (con == 1)
  h_i(1) = Gate_a1 (a1,b1,aft,ashl);
  h_i(2) = Gate_b1 (a2,b2,bft,bshl);
  out = Gate_c1 (a3,h_i,cft,cshl);
 end
 if (con == 2)
  h_i(1) = Gate_b1 (a1,b1,bft,bshl);
  h_i(2) = Gate_c1 (a2,b2,cft,cshl);
  out = Gate_d1 (a3,h_i,dft,dshl);
 end
 if (con == 3)
  h_i(1) = Gate_c1 (a1,b1,cft,cshl);
  h_i(2) = Gate_d1 (a2,b2,dft,dshl);
  out = Gate_a1 (a3,h_i,aft,ashl);
 end
 if (con == 4)
  h_i(1) = Gate_d1 (a1,b1,dft,dshl);
  h_i(2) = Gate_a1 (a2,b2,aft,ashl);
  out = Gate_b1 (a3,h_i,bft,bshl);
 end
end


function [ out ] = Gate_a1( ii_s,ii_i,ft,ftshl)

 if (ft == 0)
  out = Q_cell_lu4 (ii_s, ii_i);
 else
  out = Q_cell_luft4 (ii_s, ii_i, ft, ftshl);
 end
end

function [ out ] = Gate_b1( ii_s,ii_i,ft,ftshl)

 if (ft == 0)
  out = Q_cell_lu4 (ii_s, ii_i);
 else
  out = Q_cell_luft4 (ii_s, ii_i, ft, ftshl);
 end
end


function [ out ] = Gate_c1( ii_s,ii_i,ft,ftshl)

 if (ft == 0)
  out = Q_cell_lu4 (ii_s, ii_i);
```

```
 else
  out = Q_cell_luft4 (ii_s, ii_i, ft, ftshl);
 end
end


function [ out ] = Gate_d1( ii_s,ii_i,ft,ftshl)

 if (ft == 0)
  out = Q_cell_lu4 (ii_s, ii_i);
 else
  out = Q_cell_luft4 (ii_s, ii_i, ft, ftshl);
 end
end


function [out] = rungelo09 ()
z = [];
for i1 = 1:4
   for i2 = 1:4
      for i3 = 1:4
         c2 = 0; c4 = 0;
         for i = 1:68
            c0 = quaddedlogic05(0,0,i1,i2,i3);
            c1 = [c0 quaddedlogic05(i,0,i1,i2,i3)]; c2 = c2 + nandfail03(c1);
            c3 = [c0 quaddedlogic05(i,1,i1,i2,i3)]; c4 = c4 + nandfail03(c3);
         end
         cc = c2 + c4; cd = cc(2:3); ce = (sum(cd)/544)*100;
         z = [z;i1 i2 i3 cc ce];
      end
   end
end
out = z;
end
function [out] = quaddedlogic05 (lo,st,l1,l2,l3)
in = [0 0;0 1;1 0;1 1];

z = [];
z1 = size(in); le = z1(1);

for i = 1:le
   x1 = [in(i,1) in(i,1) in(i,1) in(i,1)];
   x2 = [in(i,2) in(i,2) in(i,2) in(i,2)];
   p = zeros(1,4); q = zeros(1,4); o = zeros(1,4);

   if (lo>=1)&&(lo<=4) x1(1) = st; end
   if (lo>=5)&&(lo<=8) x2(1) = st; end

   xn = [x1(1) x1(2) x2(1) x2(2) x1(2) x1(1) x2(2) x2(1) x1(3) x1(4) x2(3) x2(4) x1(4) x1(3) x2(4) x2(3)];
   if (lo>=9)&&(lo<=24) xn(1,lo-8) = st; end
   if(l1==1)
p(1)=(xn(1)&xn(2)&xn(3)&xn(4));p(2)=(xn(5)&xn(6)&xn(7)&xn(8));p(3)=(xn(9)&xn(10)&xn(11)&xn(
12));p(4)=(xn(13)&xn(14)&xn(15)&xn(16)); end
```

```
   if(l1==2)
p(1)=~(xn(1)&xn(2)&xn(3)&xn(4));p(2)=~(xn(5)&xn(6)&xn(7)&xn(8));p(3)=~(xn(9)&xn(10)&xn(11)&
xn(12));p(4)=~(xn(13)&xn(14)&xn(15)&xn(16)); end
   if(l1==3)
p(1)=(xn(1)|xn(2)|xn(3)|xn(4));p(2)=(xn(5)|xn(6)|xn(7)|xn(8));p(3)=(xn(9)|xn(10)|xn(11)|xn(12));
p(4)=(xn(13)|xn(14)|xn(15)|xn(16)); end
   if(l1==4)
p(1)=~(xn(1)|xn(2)|xn(3)|xn(4));p(2)=~(xn(5)|xn(6)|xn(7)|xn(8));p(3)=~(xn(9)|xn(10)|xn(11)|xn(1
2));p(4)=~(xn(13)|xn(14)|xn(15)|xn(16)); end

   xn = [x1(1) x1(2) x2(1) x2(2) x1(2) x1(1) x2(2) x2(1) x1(3) x1(4) x2(3) x2(4) x1(4) x1(3) x2(4) x2(3)];
   if (lo>=25)&&(lo<=40) xn(1,lo-24) = st; end
   if(l2==1)
q(1)=(xn(1)&xn(2)&xn(3)&xn(4));q(2)=(xn(5)&xn(6)&xn(7)&xn(8));q(3)=(xn(9)&xn(10)&xn(11)&xn(
12));q(4)=(xn(13)&xn(14)&xn(15)&xn(16)); end
   if(l2==2)
q(1)=~(xn(1)&xn(2)&xn(3)&xn(4));q(2)=~(xn(5)&xn(6)&xn(7)&xn(8));q(3)=~(xn(9)&xn(10)&xn(11)&
xn(12));q(4)=~(xn(13)&xn(14)&xn(15)&xn(16)); end
   if(l2==3)
q(1)=(xn(1)|xn(2)|xn(3)|xn(4));q(2)=(xn(5)|xn(6)|xn(7)|xn(8));q(3)=(xn(9)|xn(10)|xn(11)|xn(12));
q(4)=(xn(13)|xn(14)|xn(15)|xn(16)); end
   if(l2==4)
q(1)=~(xn(1)|xn(2)|xn(3)|xn(4));q(2)=~(xn(5)|xn(6)|xn(7)|xn(8));q(3)=~(xn(9)|xn(10)|xn(11)|xn(1
2));q(4)=~(xn(13)|xn(14)|xn(15)|xn(16)); end

   if (lo>=41)&&(lo<=44) p(1,lo-40) = st; end
   if (lo>=45)&&(lo<=48) q(1,lo-44) = st; end

   yn = [p(1) p(4) q(1) q(4) p(2) p(3) q(2) q(3) p(3) p(2) q(3) q(2) p(4) p(1) q(4) q(1)];
   if (lo>=49)&&(lo<=64) yn(1,lo-48) = st; end
   if(l3==1)
o(1)=(yn(1)&yn(2)&yn(3)&yn(4));o(2)=(yn(5)&yn(6)&yn(7)&yn(8));o(3)=(yn(9)&yn(10)&yn(11)&yn(
12));o(4)=(yn(13)&yn(14)&yn(15)&yn(16)); end
   if(l3==2)
o(1)=~(yn(1)&yn(2)&yn(3)&yn(4));o(2)=~(yn(5)&yn(6)&yn(7)&yn(8));o(3)=~(yn(9)&yn(10)&yn(11)
&yn(12));o(4)=~(yn(13)&yn(14)&yn(15)&yn(16)); end
   if(l3==3)
o(1)=(yn(1)|yn(2)|yn(3)|yn(4));o(2)=(yn(5)|yn(6)|yn(7)|yn(8));o(3)=(yn(9)|yn(10)|yn(11)|yn(12));
o(4)=(yn(13)|yn(14)|yn(15)|yn(16)); end
   if(l3==4)
o(1)=~(yn(1)|yn(2)|yn(3)|yn(4));o(2)=~(yn(5)|yn(6)|yn(7)|yn(8));o(3)=~(yn(9)|yn(10)|yn(11)|yn(1
2));o(4)=~(yn(13)|yn(14)|yn(15)|yn(16)); end

   if (lo>=65)&&(lo<=68) o(1,lo-64) = st; end
   y = [o(1);o(2);o(3);o(4)];
   z = [z;y];
end

out = z;
end


function [out] = nandfail03 (st)
```

```
ls = size(st);
x1 = ls(1); x2 = ls(2);
z0 = 0; z1 = 0; z2 = 0;

for i = 1:x1/4
   p1 = ((i-1)*4)+1; p2 = i*4;
   y1 = st(p1:p2,1); y2 = st(p1:p2,2);
   a1 = sum(y1); a2 = sum(y2); a3 = 0;
   if (a1 < a2) a3 = a2-a1; end
   if (a2 < a1) a3 = a1-a2; end
   if (a3==1) z0 = z0+1; end
   if (a3==2)
      z2 = z2+1;
   else
      v1 = voter4(y1); v2 = voter4(y2);
      if (xor(v1,v2)) z1 = z1+1; end
   end
end
out = [z0 z1 z2];
end
```

Appendix 3.4: MATLAB program for Chapter 7 for the purpose of analysing the fault behaviour of the QLC structure

With this programmes the fault behaviour of the QLC structure is evaluated in regards of fault localisation down into the individual logic unit.

```
function QcellTestAll (sel,inp)
 tx=Qcelltestseq(sel,inp,1,0);
 xlswrite('d:\test1.xls',tx,1,'A1');
 tx=QcellResultCheck(tx,4,4);
 xlswrite('d:\test1.xls',tx,1,'A17');
tx=Qcelltestseq(sel,inp,1,1);
 xlswrite('d:\test1.xls',tx,2,'A1');
 tx=QcellResultCheck(tx,4,4);
 xlswrite('d:\test1.xls',tx,2,'A17');
tx=Qcelltestseq(sel,inp,2,0);
 xlswrite('d:\test2.xls',tx,1,'A1');
 tx=QcellResultCheck(tx,4,4);
 xlswrite('d:\test2.xls',tx,1,'A17');
tx=Qcelltestseq(sel,inp,2,1);
 xlswrite('d:\test2.xls',tx,2,'A1');
 tx=QcellResultCheck(tx,4,4);
 xlswrite('d:\test2.xls',tx,2,'A17');
tx=Qcelltestseq(sel,inp,3,0);
 xlswrite('d:\test3.xls',tx,1,'A1');
 tx=QcellResultCheck(tx,4,4);
 xlswrite('d:\test3.xls',tx,1,'A17');
tx=Qcelltestseq(sel,inp,3,1);
 xlswrite('d:\test3.xls',tx,2,'A1');
 tx=QcellResultCheck(tx,4,4);
 xlswrite('d:\test3.xls',tx,2,'A17');
tx=Qcelltestseq(sel,inp,4,0);
 xlswrite('d:\test4.xls',tx,1,'A1');
 tx=QcellResultCheck(tx,4,4);
 xlswrite('d:\test4.xls',tx,1,'A17');
tx=Qcelltestseq(sel,inp,4,1);
 xlswrite('d:\test4.xls',tx,2,'A1');
 tx=QcellResultCheck(tx,4,4);
 xlswrite('d:\test4.xls',tx,2,'A17');
end
```

```
function [out] = Qcelltestseq (sel,inp,ftag,ishl)
 xi=inp;
 for j = 0:12
   tdata = Q_cell_Test(sel,inp,ftag,j,ishl);
   xi = [xi,tdata];
```

```
 end
 out = xi;
end


function [out] = Q_cell_Test (sel,inp,ftag,ift,ishl)
 tdata = zeros(size(inp)); x=size(inp);
 for j = 1:x(1)
  for i = 1:x(2)
   tdata(j,i) = Q_cell_matrix (i,sel,inp(j,1:x(2)),ftag,ift,ishl);
  end
 end
 out = tdata;
end

function [out] = Q_cell_matrix (con,sel,inp,ftag,ift,ishl)
aft=0;bft=0;cft=0;dft=0;
ashl=0;bshl=0;cshl=0;dshl=0;
if(ftag == 1) aft=ift;ashl=ishl; end
if(ftag == 2) bft=ift;bshl=ishl; end
if(ftag == 3) cft=ift;cshl=ishl; end
if(ftag == 4) dft=ift;dshl=ishl; end
h_i = [0,0];
 a1 = [sel(1),sel(2)];a2 =[sel(3),sel(4)];a3 = [sel(5),sel(6)];
 b1 = [inp(1),inp(2)];b2 = [inp(3),inp(4)];
 if (con == 1)
  h_i(1) = Gate_a (a1,b1,aft,ashl);
  h_i(2) = Gate_b (a2,b2,bft,bshl);
  out = Gate_c (a3,h_i,cft,cshl);
 end
 if (con == 2)
  h_i(1) = Gate_b (a1,b1,bft,bshl);
  h_i(2) = Gate_c (a2,b2,cft,cshl);
  out = Gate_d (a3,h_i,dft,dshl);
 end
 if (con == 3)
  h_i(1) = Gate_c (a1,b1,cft,cshl);
  h_i(2) = Gate_d (a2,b2,dft,dshl);
  out = Gate_a (a3,h_i,aft,ashl);
 end
 if (con == 4)
  h_i(1) = Gate_d (a1,b1,dft,dshl);
  h_i(2) = Gate_a (a2,b2,aft,ashl);
  out = Gate_b (a3,h_i,bft,bshl);
 end
end

function [ out ] = Gate_a1( ii_s,ii_i,ft,ftshl)

 if (ft == 0)
  out = Q_cell_lu4 (ii_s, ii_i);
 else
```

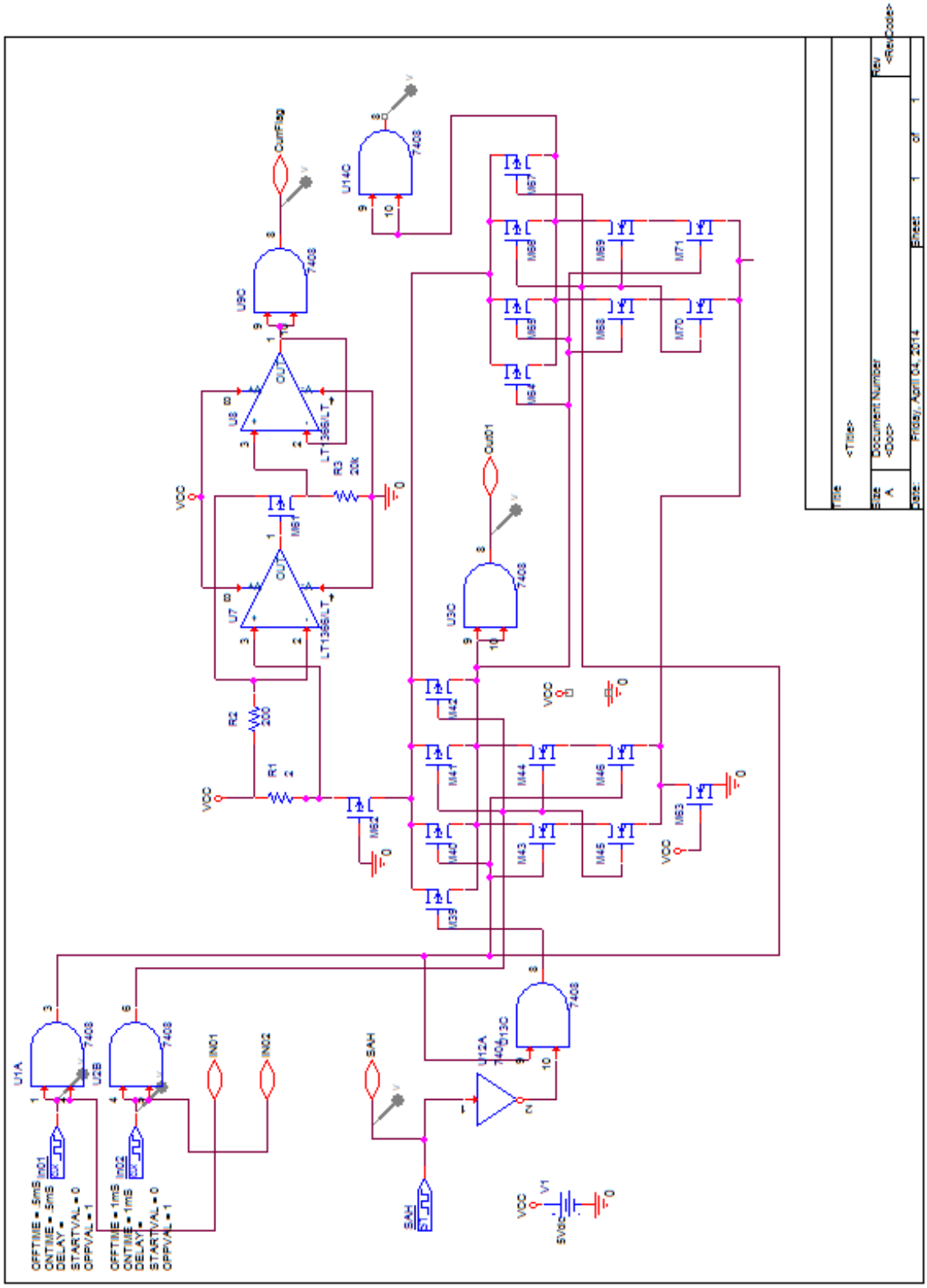
```
 out = Q_cell_luft4 (ii_s, ii_i, ft, ftshl);
 end
end

function [ out ] = Gate_b1( ii_s,ii_i,ft,ftshl)

 if (ft == 0)
 out = Q_cell_lu4 (ii_s, ii_i);
 else
 out = Q_cell_luft4 (ii_s, ii_i, ft, ftshl);
 end
end

function [ out ] = Gate_c1( ii_s,ii_i,ft,ftshl)

 if (ft == 0)
 out = Q_cell_lu4 (ii_s, ii_i);
 else
 out = Q_cell_luft4 (ii_s, ii_i, ft, ftshl);
 end
end

function [ out ] = Gate_d1( ii_s,ii_i,ft,ftshl)

 if (ft == 0)
 out = Q_cell_lu4 (ii_s, ii_i);
 else
 out = Q_cell_luft4 (ii_s, ii_i, ft, ftshl);
 end
end

function [out] = QcellResultCheck(xi,pi,pr)
 a1 = size(xi);
 x1 = xi(1:a1(1),(pi+1):(pi+pr));
 x2 = xi(1:a1(1),(pi+pr+1):a1(2));
 l1 = ((a1(2)-(pi+pr)))/pr;
 erg3 = zeros((pr+a1(1)),l1);
 for i1 = 1:l1
    erg1 = zeros(1,pr);
    erg2 = zeros(1,l1);
    p1 = 1+((i1-1)*pr);
    p2 = i1*pr;
    xi1 = x2(1:a1(1),p1:p2);
    xi2 = abs(xi1-x1);
    for i2 = 1:pr
       erg1(1,i2) = sum(xi2(1:a1(1),i2));
    end
    for i2 = 1:a1(1)
       erg2(1,i2) = sum(xi2(i2,1:pr));
    end
    erg5 = [erg1,erg2];
    erg5 = rot90(fliplr(erg5));
```

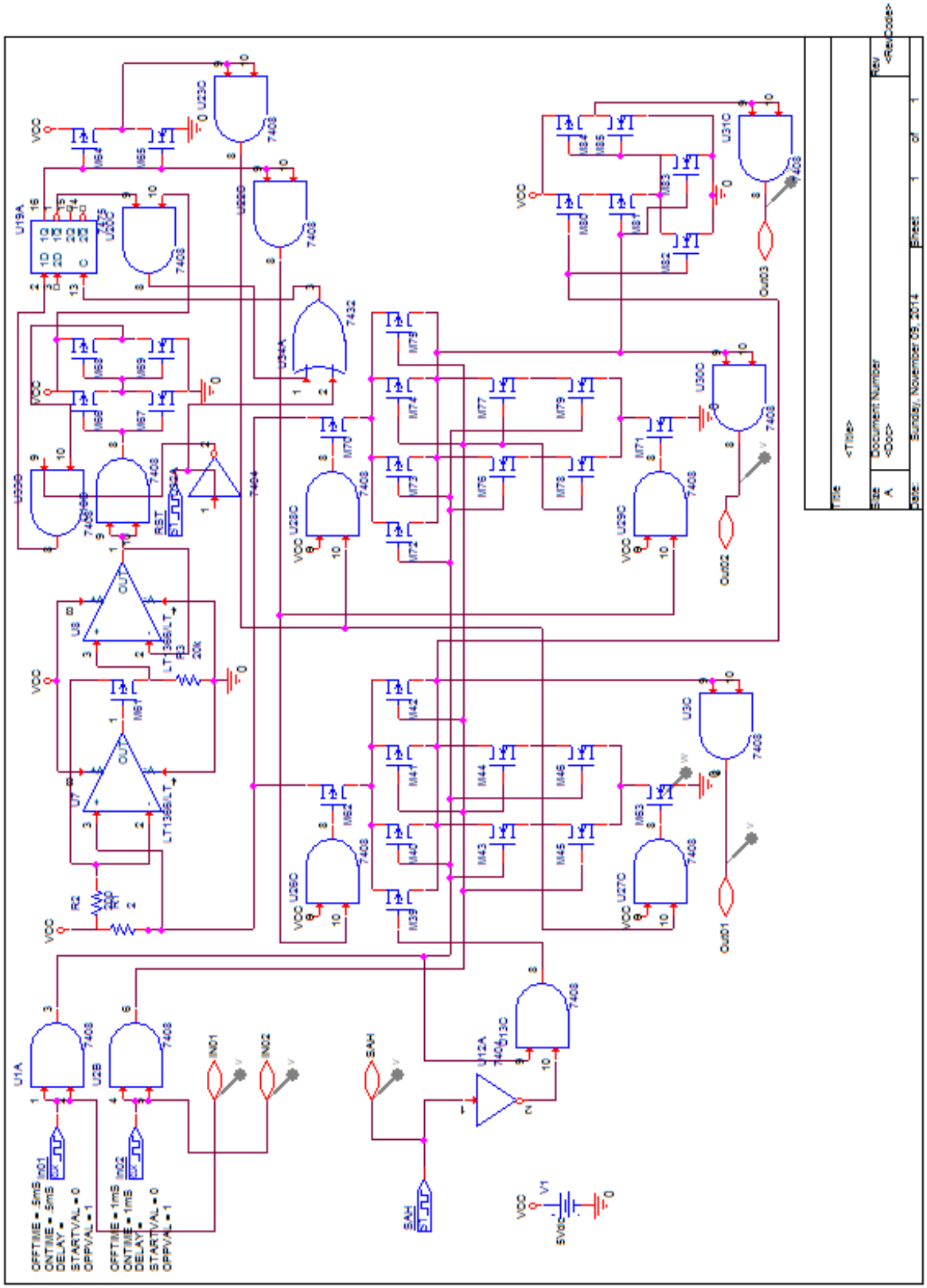
```
    erg3(1:(a1(1)+pr),i1) = erg5;
 end
 out = erg3;
end

function [out] = Q_cell_Test_full_adder ()
 sel0 = [1,0,0,0,1,1]; sel1 = [1,1,1,1,0,0];
 ftag = 0; ift = 0; ishl = 0;
 a=[0,0,0;0 0 1;0 1 0;0 1 1;1 0 0;1 0 1;1 1 0;1 1 1];
 b=[a(1:8,1:2)]; b(1:8,3)=b(1:8,1); b(1:8,4)=b(1:8,2); inp = b;
 c=[a(1:8,3)];
 td1 = zeros(size(inp)); x=size(inp); td2 = td1;
 for j = 1:x(1)
  for i = 1:x(2)
   aftag = 1; aift = 4; aishl = 1;
   ih0 = Q_cell_matrix (i,sel0,inp(j,1:x(2)),aftag,aift,aishl);
   ih1 = [ih0 c(j) ih0 c(j)];
   td1(j,i) = Q_cell_matrix (i,sel0,ih1,ftag,ift,ishl);
   ih2 = [b(j,1) b(j,2) c(j) ih0];
   td2(j,i) = Q_cell_matrix (i,sel1,ih2,ftag,ift,ishl);
  end
 end
 out = [td2 td1];
end

function [out] = sa_fc17 (sal,sac)
y = []; t = zeros(1,6);
a = [0 0 0 0 0 0 0;1 0 0 0 0 0 0;0 1 0 0 0 1 1;1 1 0 0 0 1 1;0 0 1 0 0 0 0;
     1 0 1 0 0 1 0;0 1 1 0 0 1 1;1 1 1 0 0 1 1;0 0 0 1 0 0 0;1 0 0 1 0 0 0;
     0 1 0 1 0 1 1;1 1 0 1 0 1 1;0 0 1 1 0 0 0;1 0 1 1 0 1 0;0 1 1 1 0 0 0;
     1 1 1 1 0 1 0;0 0 0 0 1 0 1;1 0 0 0 1 0 1;0 1 0 0 1 1 1;1 1 0 0 1 1 1;
     0 0 1 0 1 0 1;1 0 1 0 1 1 1;0 1 1 0 1 1 1;1 1 1 0 1 1 1;0 0 0 1 1 0 1;
     1 0 0 1 1 0 1;0 1 0 1 1 1 1;1 1 0 1 1 1 1;0 0 1 1 1 0 0;1 0 1 1 1 1 0;
     0 1 1 1 1 0 0;1 1 1 1 1 1 0];
b = size(a);
ii = b(1);
for i = 1:ii
   x = a(i,1:b(2));
   i1 = sanand1_1(x(1),x(3),t(1),sal,sac); t(1) = i1;
   i2 = sanand1_2(x(3),x(4),t(2),sal,sac); t(2) = i2;
   i3 = sanand1_3(x(2),i2,t(3),sal,sac); t(3) = i3;
   i4 = sanand1_4(x(5),i2,t(4),sal,sac); t(4) = i4;
   y1 = sanand1_5(i1,i3,t(5),sal,sac); t(5) = y1;
   y2 = sanand1_6(i3,i4,t(6),sal,sac); t(6) = y2;
   y = [y;x(6) x(7) i y1 y2];
end
out = y;
end

function [out] = sa4_fc17 (sal,sac)
y = [];
a = [0 0 0 0 0 0 0;1 0 0 0 0 0 0;0 1 0 0 0 1 1;1 1 0 0 0 1 1;0 0 1 0 0 0 0;
```

```
    1 0 1 0 0 1 0;0 1 1 0 0 1 1;1 1 1 0 0 1 1;0 0 0 1 0 0 0;1 0 0 1 0 0 0;
    0 1 0 1 0 1 1;1 1 0 1 0 1 1;0 0 1 1 0 0 0;1 0 1 1 0 1 0;0 1 1 1 0 0 0;
    1 1 1 1 0 1 0;0 0 0 0 1 0 1;1 0 0 0 1 0 1;0 1 0 0 1 1 1;1 1 0 0 1 1 1;
    0 0 1 0 1 0 1;1 0 1 0 1 1 1;0 1 1 0 1 1 1;1 1 1 0 1 1 1;0 0 0 1 1 0 1;
    1 0 0 1 1 0 1;0 1 0 1 1 1 1;1 1 0 1 1 1 1;0 0 1 1 1 0 0;1 0 1 1 1 1 0;
    0 1 1 1 1 0 0;1 1 1 1 1 1 0];
b = size(a);
ii = b(1);
for i = 1:ii
   x = a(i,1:b(2));
   i1 = sanand4_1(x(1),x(3),sal,sac);
   i2 = sanand4_2(x(3),x(4),sal,sac);
   i3 = sanand4_3(x(2),i2,sal,sac);
   i4 = sanand4_4(x(5),i2,sal,sac);
   y1 = sanand4_5(i1,i3,sal,sac);
   y2 = sanand4_6(i3,i4,sal,sac);
   y = [y;x(6) x(7) i y1 y2];
end
out = y;
end

function [out] = safc17 (fl,fs)
y = [];
a = [0 0 0 0 0 0 0;1 0 0 0 0 0 0;0 1 0 0 0 1 1;1 1 0 0 0 1 1;0 0 1 0 0 0 0;
    1 0 1 0 0 1 0;0 1 1 0 0 1 1;1 1 1 0 0 1 1;0 0 0 1 0 0 0;1 0 0 1 0 0 0;
    0 1 0 1 0 1 1;1 1 0 1 0 1 1;0 0 1 1 0 0 0;1 0 1 1 0 1 0;0 1 1 1 0 0 0;
    1 1 1 1 0 1 0;0 0 0 0 1 0 1;1 0 0 0 1 0 1;0 1 0 0 1 1 1;1 1 0 0 1 1 1;
    0 0 1 0 1 0 1;1 0 1 0 1 1 1;0 1 1 0 1 1 1;1 1 1 0 1 1 1;0 0 0 1 1 0 1;
    1 0 0 1 1 0 1;0 1 0 1 1 1 1;1 1 0 1 1 1 1;0 0 1 1 1 0 0;1 0 1 1 1 1 0;
    0 1 1 1 1 0 0;1 1 1 1 1 1 0];
b = size(a);
ii = b(1);
for i = 1:ii
   x = a(i,1:b(2));
   if (fl==1) x(1) = fs; end
   if (fl==2) x(2) = fs; end
   if (fl==3) x(3) = fs; end
   if (fl==4) x(4) = fs; end
   if (fl==5) x(5) = fs; end
   xh1 = x(3); xh2 = x(3);
   if (fl==6) xh1 = fs; end
   if (fl==7) xh2 = fs; end
   i1 = ~(x(1)&xh1);
   if (fl==8) i1 = fs; end
   i2 = ~(xh2&x(4)); hi21 = i2; hi22 = i2;
   if (fl==9) hi21 = fs; hi22 = fs; end
   if (fl==10) hi21 = fs; end
   if (fl==11) hi22 = fs; end
   i3 = ~(x(2)&hi21); hi31 = i3; hi32 = i3;
   i4 = ~(x(5)&hi22);
   if (fl==12) hi31 = fs; hi32 = fs; end
   if (fl==13) i4 = fs; end
```

```
    if (fl==14) hi31 = fs; end
    if (fl==15) hi32 = fs; end
    y1 = ~(i1&hi31);
    y2 = ~(hi32&i4);
    if (fl==16) y1 = fs; end
    if (fl==17) y2 = fs; end
    tr = 100*fl+i;
    y = [y;x(6) x(7) tr y1 y2];
end
out = y;
end
```

Appendix 4: Spice simulation circuit of SAFR-logic gates




Out03
Out02
Out01
IN01
IN02
SAH
VCC
5Vdc
Title
<Title>
Size
A
Document Number
<Doc>
Rev
<RevCode>
Date:
Sunday, November 09, 2014
Sheet
1
of
1

Appendix 5: Fault results of the fault simulation in accordance of logic gate alteration for a certain selection of eight transistor-style variation

SAH & Stage 4
Design A two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	1	1	1	2	2	2	2
T2	1	x	1	1	2	2	2	2
T3	1	1	x	1	2	2	2	2
T4	1	1	1	x	2	2	2	2
T5	2	2	2	2	x	3	2	1
T6	2	2	2	2	3	x	1	2
T7	2	2	2	2	2	1	x	3
T8	2	2	2	2	1	2	3	x

SAH & Stage 4
Design B two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	1	1	1	2	2	2	2
T2	1	x	1	1	2	2	2	2
T3	1	1	x	1	2	2	2	2
T4	1	1	1	x	2	2	2	2
T5	2	2	2	2	x	1	3	3
T6	2	2	2	2	1	x	3	3
T7	2	2	2	2	3	3	x	1
T8	2	2	2	2	3	3	1	x

SAH & Stage 4
Design C two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	1	0	0	1	1	1	1
T2	1	x	0	0	1	1	1	1
T3	0	0	x	1	1	1	1	1
T4	0	0	1	x	1	1	1	1
T5	1	1	1	1	x	3	1	2
T6	1	1	1	1	3	x	2	1
T7	1	1	1	1	1	2	x	3
T8	1	1	1	1	2	1	3	x

SAH & Stage 4
Design D two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	0	1	1	1	1	1	1
T2	0	x	1	1	1	1	1	1
T3	1	1	x	0	1	1	1	1
T4	1	1	0	x	1	1	1	1
T5	1	1	1	1	x	3	2	1
T6	1	1	1	1	3	x	1	2
T7	1	1	1	1	2	1	x	3
T8	1	1	1	1	1	2	3	x

SAH & Stage 4
Design E two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	1	0	0	1	1	1	1
1	1	x	0	0	1	1	1	1
T3	0	0	x	1	1	1	1	1
T4	0	0	1	x	1	1	1	1
T5	1	1	1	1	x	1	3	3
T6	1	1	1	1	1	x	3	3
T7	1	1	1	1	3	3	x	1
T8	1	1	1	1	3	3	1	x

SAH & Stage 4
Design F two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	0	1	1	1	1	1	1
T2	0	x	1	1	1	1	1	1
T3	1	1	x	0	1	1	1	1
T4	1	1	0	x	1	1	1	1
T5	1	1	1	1	x	1	3	3
T6	1	1	1	1	1	x	3	3
T7	1	1	1	1	3	3	x	1
T8	1	1	1	1	3	3	1	x

Appendix

SAL & Stage 4
Design A two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	1	0	0	0	0	0	0
T2	1	x	0	0	0	0	0	0
T3	0	0	x	1	0	0	0	0
T4	0	0	1	x	0	0	0	0
T5	0	0	0	0	x	0	1	1
T6	0	0	0	0	0	x	1	1
T7	0	0	0	0	1	1	x	0
T8	0	0	0	0	1	1	0	x

SAL & Stage 4
Design B two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	1	0	0	0	0	0	0
T2	1	x	0	0	0	0	0	0
T3	0	0	x	1	0	0	0	0
T4	0	0	1	x	0	0	0	0
T5	0	0	0	0	x	1	0	0
T6	0	0	0	0	1	x	0	0
T7	0	0	0	0	0	0	x	1
T8	0	0	0	0	0	0	1	x

SAL & Stage 4
Design C two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	1	3	3	1	1	1	1
T2	1	x	3	3	1	1	1	1
T3	3	3	x	1	1	1	1	1
T4	3	3	1	x	1	1	1	1
T5	1	1	1	1	x	0	1	1
T6	1	1	1	1	0	x	1	1
T7	1	1	1	1	1	1	x	0
T8	1	1	1	1	1	1	0	x

SAL & Stage 4
Design D two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	3	2	1	1	1	1	1
T2	3	x	1	2	1	1	1	1
T3	2	1	x	3	1	1	1	1
T4	1	2	3	x	1	1	1	1
T5	1	1	1	1	x	0	1	1
T6	1	1	1	1	0	x	1	1
T7	1	1	1	1	1	1	x	0
T8	1	1	1	1	1	1	0	x

SAL & Stage 4
Design E two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	1	3	3	1	1	1	1
T2	1	x	3	3	1	1	1	1
T3	3	3	x	1	1	1	1	1
T4	3	3	1	x	1	1	1	1
T5	1	1	1	1	x	1	0	0
T6	1	1	1	1	1	x	0	0
T7	1	1	1	1	0	0	x	1
T8	1	1	1	1	0	0	1	x

SAL & Stage 4
Design F two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	3	1	2	1	1	1	1
T2	3	x	2	1	1	1	1	1
T3	1	2	x	3	1	1	1	1
T4	2	1	3	x	1	1	1	1
T5	1	1	1	1	x	1	0	0
T6	1	1	1	1	1	x	0	0
T7	1	1	1	1	0	0	x	1
T8	1	1	1	1	0	0	1	x

Total numbers of faukts
Design A two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	2	1	1	2	2	2	2
T2	2	x	1	1	2	2	2	2
T3	1	1	x	2	2	2	2	2
T4	1	1	2	x	2	2	2	2
T5	2	2	2	2	x	3	3	2
T6	2	2	2	2	3	x	2	3
T7	2	2	2	2	3	2	x	3
T8	2	2	2	2	2	3	3	x

Total numbers of faukts
Design A two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	2	1	1	2	2	2	2
T2	2	x	1	1	2	2	2	2
T3	1	1	x	2	2	2	2	2
T4	1	1	2	x	2	2	2	2
T5	2	2	2	2	x	2	3	3
T6	2	2	2	2	2	x	3	3
T7	2	2	2	2	3	3	x	2
T8	2	2	2	2	3	3	2	x

Total numbers of faukts
Design A two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	2	3	3	2	2	2	2
T2	2	x	3	3	2	2	2	2
T3	3	3	x	2	2	2	2	2
T4	3	3	2	x	2	2	2	2
T5	2	2	2	2	x	3	2	3
T6	2	2	2	2	3	x	3	2
T7	2	2	2	2	2	3	x	3
T8	2	2	2	2	3	2	3	x

Total numbers of faukts
Design A two faults

	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	3	3	2	2	2	2	2
T2	3	x	2	3	2	2	2	2
T3	3	2	x	3	2	2	2	2
T4	2	3	3	x	2	2	2	2
T5	2	2	2	2	x	3	3	2
T6	2	2	2	2	3	x	2	3
T7	2	2	2	2	3	2	x	3
T8	2	2	2	2	2	3	3	x

Total numbers of faukts
Design A two faults

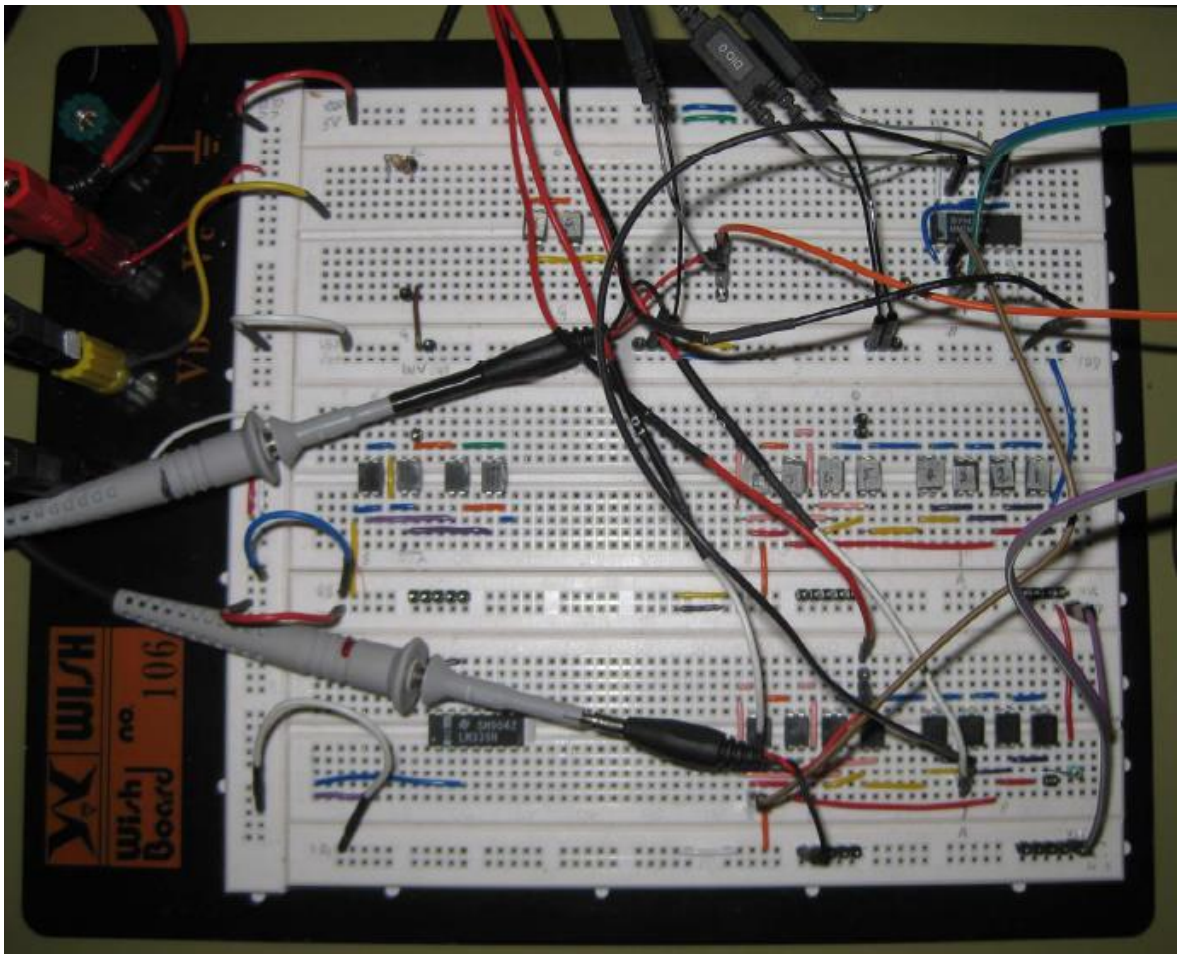
	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	2	3	3	2	2	2	2
T2	2	x	3	3	2	2	2	2
T3	3	3	x	2	2	2	2	2
T4	3	3	2	x	2	2	2	2
T5	2	2	2	2	x	2	3	3
T6	2	2	2	2	2	x	3	3
T7	2	2	2	2	3	3	x	2
T8	2	2	2	2	3	3	2	x

Total numbers of faukts
Design A two faults

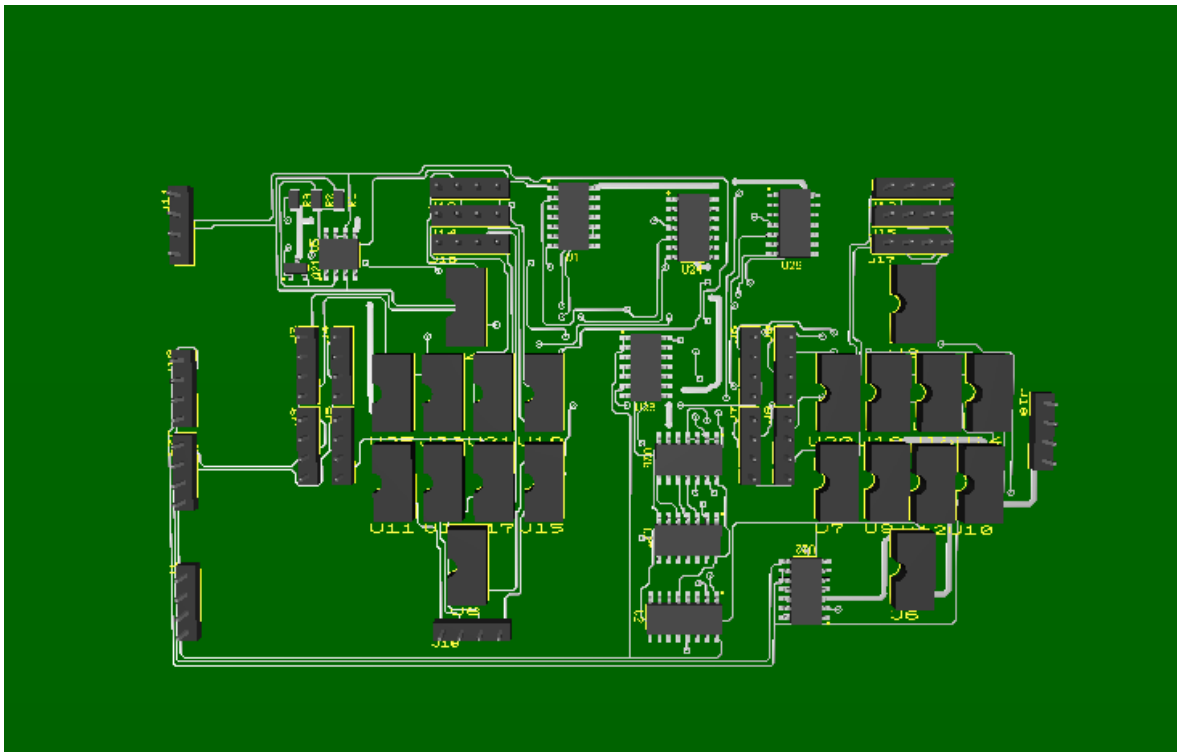
	Transistor							
	T1	T2	T3	T4	T5	T6	T7	T8
T1	x	3	2	3	2	2	2	2
T2	3	x	3	2	2	2	2	2
T3	2	3	x	3	2	2	2	2
T4	3	2	3	x	2	2	2	2
T5	2	2	2	2	x	2	3	3
T6	2	2	2	2	2	x	3	3
T7	2	2	2	2	3	3	x	2
T8	2	2	2	2	3	3	2	x

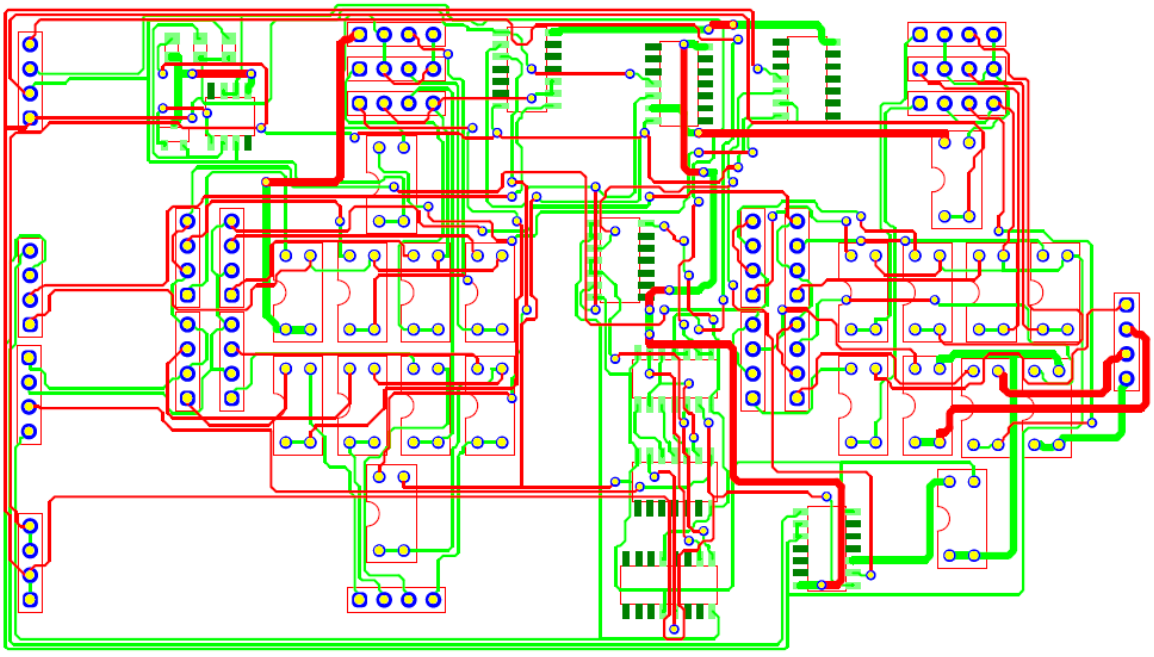
With these fault maps the correlation between fault causing transistor in relation the SAH and SAL fault injected are demonstrated. The highest number of faults per transistor is identified.

Appendix 6: Breadboard of the SAFR-NAND gate design

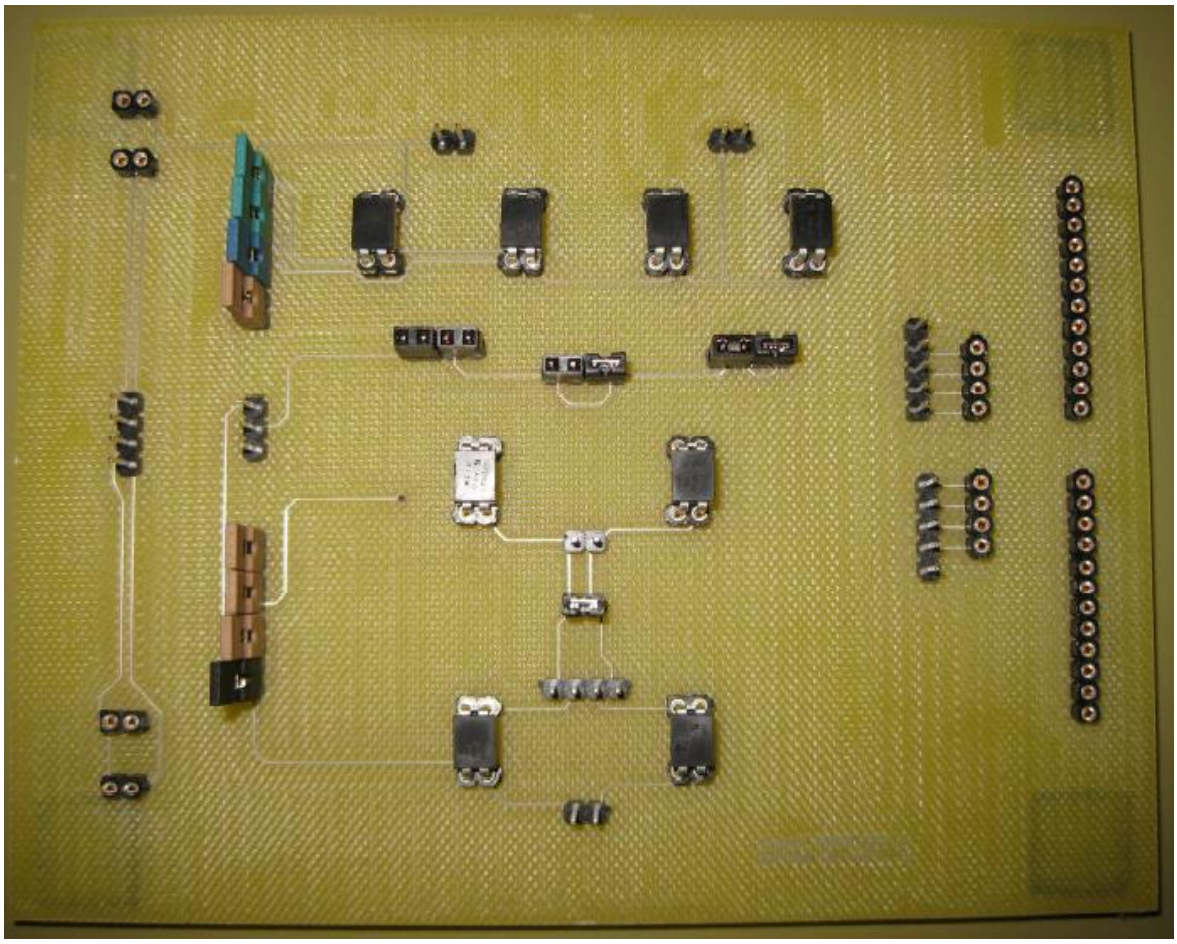


Appendix 7: PCB design of self-healing SAFR-NAND gate

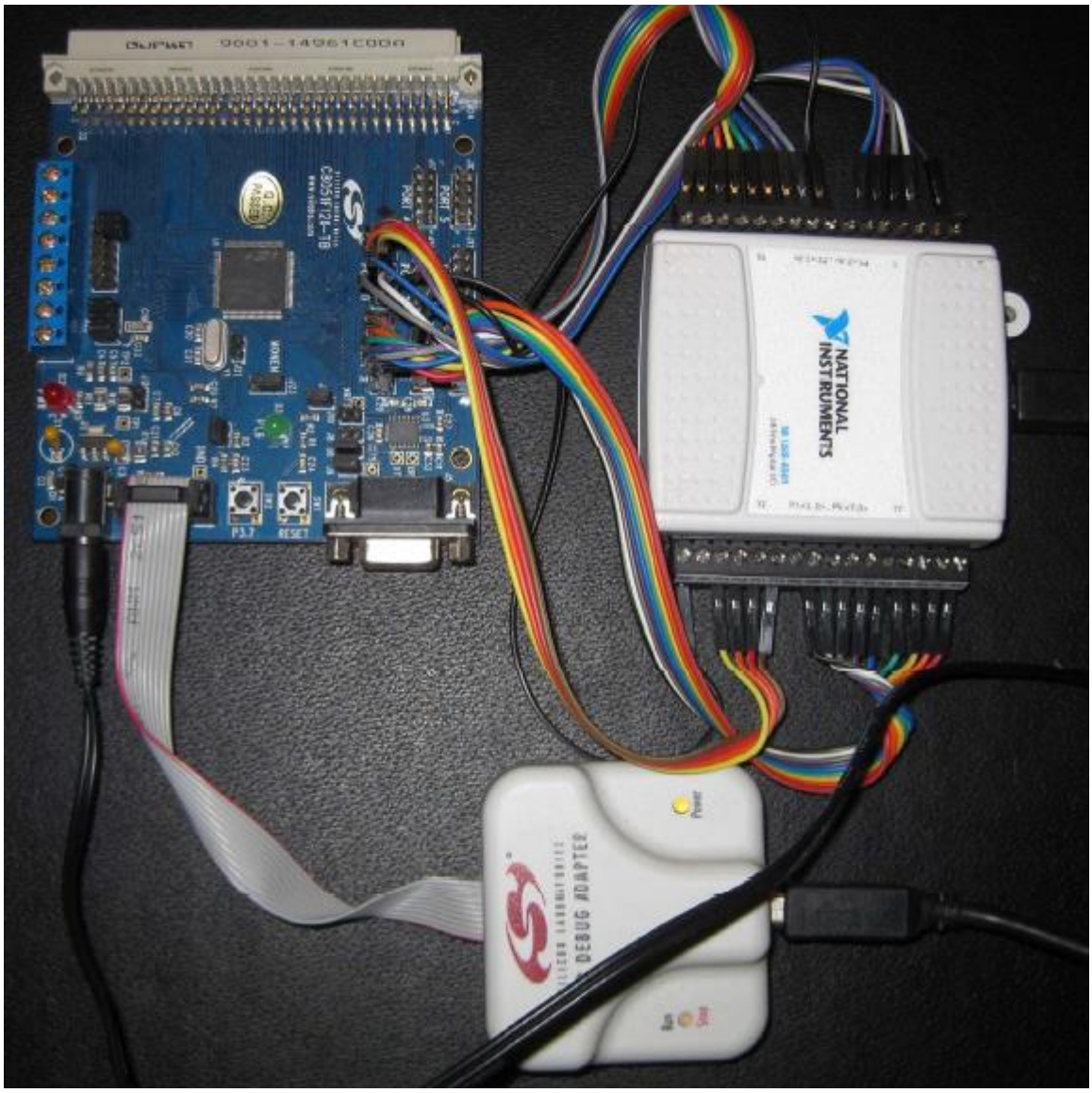




Appendix 8: Circuit board design of the SAFR-NAND gate



Appendix 9: 8051 set-up for the simulation of the soda machine FSM



Appendix 10: Assembler code for the FSM soda machine

```
$NOMOD51

$include (c8051f120.inc)

Array       equ 040h
OutReq      equ P1.7
OutCan      equ P1.2
OutDim      equ P1.1
OutCen      equ P1.0

cseg    AT 0
ljmp    Main

        rseg    Blink
        using   0

Main:
        mov     WDTCN, #0DEh
        mov     WDTCN, #0ADh
        org     0
        mov     Array   ,#000000000b
        mov     Array+1,#000000100b
        mov     Array+2,#000001000b
        mov     Array+3,#000001100b
        mov     Array+4,#000000100b
        mov     Array+5,#000001000b
        mov     Array+6,#000010000b
        mov     Array+7,#011111010b
        mov     Array+8,#000001000b
        mov     Array+9,#000010000b
        mov     Array+10,#000010100b
        mov     Array+11,#001111011b
        mov     Array+12,#000001100b
        mov     Array+13,#011111010b
        mov     Array+14,#001111011b
        mov     Array+15,#010111100b
        mov     Array+16,#000010000b
        mov     Array+17,#000010100b
        mov     Array+18,#000001100b
        mov     Array+19,#010111001b
        mov     Array+20,#000010100b
        mov     Array+21,#000001100b
        mov     Array+22,#000111000b
        mov     Array+23,#001111000b
        mov     Array+24,#010111001b
        mov     Array+25,#011111010b
        mov     Array+26,#000100000b
```

```
        mov     Array+27,#011100000b
        mov     Array+28,#010111001b

        mov     SFRPAGE, #CONFIG_PAGE

        mov     XBR2,#040h
        mov     P2MDout,#000h
        mov     P3MDout,#0ffh
        mov     P1MDOUT,#0ffh
        mov     P0MDOUT,#000h
        mov     P3,#00h
        mov     P2,#0ffh
        mov     P1,#00h
        mov     P0,#0ffh

Main1:  jb      P0.7,Main2
        jmp     Main1
Main2:  mov     A,P2
        anl     A,#01fh
        orl     A,#040h
        mov     R1,A
        mov     A,@R1
        mov     P3,A
Main3:  jnb     P0.7,Main4
        jmp     Main3
Main4:  jmp     Main1

END
```