

Soft Error Resistant Design of the AES Cipher Using SRAM-based FPGA

by

Solmaz Ghaznavi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2011

©Solmaz Ghaznavi 2011

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis presents a new architecture for the reliable implementation of the symmetric-key algorithm Advanced Encryption Standard (AES) in Field Programmable Gate Arrays (FPGAs). Since FPGAs are prone to soft errors caused by radiation, and AES is highly sensitive to errors, reliable architectures are of significant concern. Energetic particles hitting a device can flip bits in FPGA SRAM cells controlling all aspects of the implementation. Unlike previous research, heterogeneous error detection techniques based on properties of the circuit and functionality are used to provide adequate reliability at the lowest possible cost. The use of dual ported block memory for *SubBytes*, duplication for the control circuitry, and a new enhanced parity technique for *MixColumns* is proposed. Previous parity techniques cover single errors in datapath registers, however, soft errors can occur in the control circuitry as well as in SRAM cells forming the combinational logic and routing. In this research, propagation of single errors is investigated in the routed netlist. Weaknesses of the previous parity techniques are identified. Architectural redesign at the register-transfer level is introduced to resolve undetected single errors in both the routing and the combinational logic.

Reliability of the AES implementation is not only a critical issue in large scale FPGA-based systems but also at both higher altitudes and in space applications where there are a larger number of energetic particles. Thus, this research is important for providing efficient soft error resistant design in many current and future secure applications.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Professor Catherine Gebotys, for all her support, guidance, and encouragement throughout this research.

I am grateful to Professor Mark Aagaard for his discussions and guidance in this research. I would like to thank Professor Andrew Kennings and Professor Doug Stinson for agreeing to read and comment on my thesis.

I would also like to thank Professor Howard Heys, my external examiner, for his time and energy so thoughtfully devoted to this endeavor.

Susan Xu and Hugh Pollitt-Smith with the CMC Microsystems were extremely helpful with their support for the tools and equipment.

I would also like to thank my colleagues Reouven Elbaz, Marcio Juliato, Patrick Longa, and Dave Kenney for useful discussions we have had in our research group.

My special thanks to Amir Khatib Zadeh, my parents and brother for their invaluable support.

Table of Contents

AUTHOR'S DECLARATION	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Figures	viii
List of Tables.....	xi
Chapter 1 Introduction.....	1
1.1 Thesis Organization.....	5
Chapter 2 Radiation Effects on Devices.....	7
2.1 Radiation Sources and Soft Error Mechanisms.....	7
2.2 Comparison of Soft Errors to Other Faults.....	14
2.2.1 Manufacturing Faults.....	14
2.2.2 Fault Attacks.....	16
2.3 Estimating Soft Error Rates of a Device	20
2.4 Technology Trends and Soft Errors	23
2.5 Summary	25
Chapter 3 Previous Research on Tackling Soft Errors	26
3.1 Fabrication Process Level Techniques to Tackle Soft Errors.....	27
3.2 Circuit and System Level Techniques to Tackle Soft Errors	29
3.3 Tackling Soft Errors by Architectural Methods	37
3.4 Soft Errors in FPGA vs. ASIC	40
3.5 Summary	42
Chapter 4 Security Needs of Data Systems	44
4.1 Security Needs and Cryptographic Algorithms.....	44
4.1.1 Advanced Encryption Standard.....	47
4.2 Block Cipher Modes.....	51
4.2.1 Confidentiality Modes	52
4.2.2 Authentication Mode.....	55
4.2.3 Authentication and Confidentiality Modes.....	56
4.3 Previous Research on AES Design.....	56
4.3.1 <i>SubBytes</i> implementations in AES	58

4.3.2 <i>MixColumns</i> implementations in AES	62
4.4 SEU-resistant AES	63
4.5 Summary	64
Chapter 5 Proposed AES with Error Detection.....	65
5.1 Error Detection in AES Logic Blocks.....	66
5.1.1 SubBytes Logic Blocks and Error Detection	68
5.1.2 <i>MixColumns</i> Logic Blocks and Error Detection	69
5.1.3 <i>AddRoundKey</i> Logic Blocks and Error Detection	74
5.2 Error Detection in Routing of AES.....	74
5.2.1 Error in Routing and Modeling	75
5.2.2 <i>MixColumns</i> Routing and Error Detection.....	76
5.2.3 <i>AddRoundKey</i> Routing and Error Detection.....	93
5.2.4 Control Circuit and Error Detection.....	94
5.3 Soft Error Resistant AES for Different Key Sizes and Decryption	94
5.4 Summary	95
Chapter 6 Comparison with Previous Research.....	97
6.1 AES Hardware Design.....	97
6.2 Experimental Results of <i>MixColumns</i>	101
6.3 Experimental Results of AES with Soft Error Mitigation	101
6.4 Summary	110
Chapter 7 Discussion and Conclusion	111
7.1 Future Work	115
Appendix A Glossary of Acronyms.....	117
Appendix B AES in System on Chip.....	119
B.1 Device Driver and Application Software	120
B.2 AES Module and IPs in System	124
B.3 Summary	129
Appendix C Control Circuitry in AES.....	131
Appendix D Device Driver for FPGA Reconfiguration	133
Appendix E Processor Local Bus IP Interface.....	136
Appendix F S-box of AES	138
Appendix G Pseudo Code for Key Expansion.....	139

Appendix H Routed Netlist Snapshots from FPGA Editor	140
Bibliography.....	142

List of Figures

Figure 1 DRAM error soft error rate caused by radiation [7]	9
Figure 2 Charge collection in a reverse-biased junction: (a) formation of electron-hole pairs, (b) funnel shape field extending deep into substrate, (c) diffusion process [7]	12
Figure 3 DRAM cell: (a) circuit, (b) layout [11]	13
Figure 4 Six transistor SRAM cell [13]	13
Figure 5 Faults: (a) soft error causing bit flip (b) manufacturing open wire	15
Figure 6 AES fault injection [20].....	19
Figure 7 SRAM system soft error rate increasing while bit soft error rate being about constant [32]	24
Figure 8 Structure of thin-film NMOS SOI [50]	28
Figure 9 Capacitors C_A and C_B added to SRAM cell mitigate SEUs [59]	30
Figure 10 Capacitance C_C added in SRAM cell [60]	30
Figure 11 Increased RC time constant by adding R_1 and R_2 in feedback loop of SRAM cell [62]..	31
Figure 12 Soft error robust SRAM cell using 10 transistors [63]	32
Figure 13 SET error detection using time redundancy [4].....	32
Figure 14 Error detection by DMR	33
Figure 15 Error correction by TMR.....	33
Figure 16 Register cell in LEON3-FT-RTAX using TMR [64]	34
Figure 17 Spatial and temporal redundancy [65].....	35
Figure 18 Parity error detection	36
Figure 19 Memory hierarchy in a typical microprocessor system [13]	37
Figure 20 Memory elements in Software-based system	38
Figure 21 Hardware components in LEON3 microprocessor [72]	39
Figure 22 Combinational logic on FPGA vs. ASIC (a) function implemented in LUT on FPGA, (b) function implemented using gates on ASIC	41
Figure 23 AES algorithm	48
Figure 24 <i>ShiftRows</i> transformation in AES [78]	51
Figure 25 Electronic codebook (ECB) mode [78]	52
Figure 26 Cipher block chaining (CBC) mode [78].....	53
Figure 27 Counter (CTR) mode [78]	54
Figure 28 Cipher-based message authentication code (CMAC) [88]	55

Figure 29 <i>SubBytes</i> with inversion in composite fields.....	59
Figure 30 Inversion of <i>SubBytes</i> in composite fields [100].....	59
Figure 31 Virtex-II Pro slice [2].....	66
Figure 32 AES including: (a) parity predictors and comparator for <i>SubBytes</i> result, (b) real and predicted parity comparator for <i>MixColumns</i> and <i>AddRoundKey</i>	67
Figure 33 Single error in LUT _A causing 2 bit flips in output	70
Figure 34 Proposed <i>MixColumns</i> LUTs mapped on FPGA: (a) bit position $i \in \{0, 2, 5, 6, 7\}$, (b) bit position $j \in \{1, 3, 4\}$	71
Figure 35 Switch box and PIP controlled by SRAM cell.....	75
Figure 36 XOR LUT propagating error whether (a) $X \oplus Y \oplus Z = 0$ or (b) $X \oplus Y \oplus Z = 1$	76
Figure 37 Routing example (a) logic blocks without considering routing, (b) actual routing detail showing pin_p	77
Figure 38 Routing of 2 fanout signal in <i>MixColumns</i> tested on FPGA.....	78
Figure 39 Detailed routing of 2 fanout signals in <i>MixColumns</i> tested on FPGA	78
Figure 40 SEU simulation in net	80
Figure 41 Routing with no pins leading up to undetectable errors.....	81
Figure 42 LUT labels a_7' (L) b_7' (L) c_7' (L) d_7' (L) and a_7' (R) related to input a_7 in Table 5.3.....	83
Figure 43 Input a_7 connection to output bits a_0' , a_1' , a_3' , a_4' , a_7' , b_7' , c_7' , d_0' , d_1' , d_3' , d_4' , and d_7' through LUTs.....	84
Figure 44 Proposed routing applied to net a_7	85
Figure 45 Part of datapath (shown in Figure 32(a)) that is routed in Figure 44	86
Figure 46 Input a_0 connection to output bits a_0' , a_1' , b_0' , c_0' , d_0' , and d_1' through LUTs	88
Figure 47 Proposed routing applied to net a_0	88
Figure 48 Proposed routing applied to net a_1 , a_4 , a_5 , and a_6	89
Figure 49 Proposed routing applied to net c_1 , c_4 , c_5 , and c_6	90
Figure 50 Proposed routing applied to net b_0 , b_2 , b_5 , b_6 , and b_7	92
Figure 51 Proposed routing applied to net d_0 , d_2 , d_5 , d_6 , and d_7	93
Figure 52 Higher level of parallelism provided by loop unrolling: (a) AES iterative looping structure, (b) N-time loop unrolling [119].....	98

Figure 53 Error coverage of single errors using parity in previous research	102
Figure 54 Error coverage of errors using Hamming code in previous research	102
Figure 55 Debugging software running on PowerPC 405 processor	120
Figure 56 Board level connections.....	120
Figure 57 Software layers for processor in FPGA.....	121
Figure 58 IPs in system on chip.....	125
Figure 59 IP Interface (IPIF).....	125
Figure 60 IP Interconnect (IPIC)	126
Figure 61 Master and slave attachments	127
Figure 62 Interrupt in AES module.....	128
Figure 63 State machine of transformations in AES.....	131
Figure 64 State machine of round in AES	132
Figure 65 Connections of Processor Local Bus (PLB) IP Interface (IPIF) [129]	136
Figure 66 FPGA Editor snapshot of 2 fanout signal in <i>MixColumns</i> tested on FPGA	140
Figure 67 FPGA Editor snapshot of routing with no pins leading up to undetectable errors	141

List of Tables

Table 2.1 FPGAs under test in Rosetta experiment [1]	22
Table 2.2 Mean time to error of Virtex-II (device XQR2V6000) in a geosynchronous orbit [9]	22
Table 3.1 Error detection and correction in register file [72]	40
Table 4.1 Lifetime and security strength of symmetric and public-key algorithm [81]	46
Table 4.2 AES parameters	48
Table 4.3 Gate counts and critical paths of <i>SubBytes</i> inversions in $GF(2^4)$ [100]	62
Table 5.1 Effects of single errors in net including pin_p on output bits	80
Table 5.2 Effects of single errors in net without any pins causing undetectable errors at the output ..	82
Table 5.3 Input nets to input pins of LUTs of 8-bit element a	83
Table 5.4 Input nets to input pins of LUTs of 8-bit element c	90
Table 5.5 Input nets to input pins of LUTs of 8-bit element b	91
Table 5.6 Input nets to input pins of LUTs of 8-bit element d	92
Table 6.1 AES previous implementations	99
Table 6.2 Experimental results of <i>MixColumns</i> implementations on FPGA	101
Table 6.3 Results of different designs of parity scheme in AES	104
Table 6.4 Error coverage of single SEUs on FPGA	107
Table 6.5 Error coverage of single SETs on FPGA	108
Table 6.6 Experimental results of DMR and proposed enhanced parity approach	108
Table 7.1 Memory address space of IPs	128
Table 7.2 IP Interconnect (IPIC) signals [129]	137
Table 7.3 S-box of AES	138

Chapter 1

Introduction

Modern electronic systems such as computers, network routers, cell phones, and smart cards communicate, store, access, or modify information. Almost any system that deals with information has security needs provided by cryptographic algorithms, in some way. In general, security requirements are categorized into authentication, data confidentiality, data integrity, non-repudiation, and access control. It has been observed that the cost of insecurity in electronic systems can be very high, and therefore security is an important issue. For instance, a security survey by the Computer Security Institute (CSI) and Federal Bureau of Investigation (FBI) revealed that just 223 organizations sampled from various industry sectors had lost hundreds of millions of dollars due to computer related security issues [13].

FPGAs have become popular platforms for implementing electronic solutions including security related applications that use cryptographic algorithms. Compared to Application Specific Integrated Circuits (ASICs), FPGAs provide a shorter time to market, less expensive design process, higher level of flexibility for debugging, and even support in field upgrading. Additionally, state of the art FPGAs comprise building blocks such as microprocessors, block memories, and logic resources. FPGAs attempt to fill the gap between hardware and software through achieving potentially higher performance than software, while providing a higher level of flexibility than hardware. For instance, high-density SRAM-based FPGAs can be an attractive platform for implementing cryptographic

algorithms, since standards change often and some applications, such as satellites, may have low volume or benefit from in field upgrade.

Despite all the benefits of SRAM-based FPGAs mentioned above, an important concern is their vulnerability to faults caused by radiation. This vulnerability of FPGAs is due to the high density of SRAM cells that control all aspects of the implementation. Energetic particles hitting an FPGA can cause faults in the device; these faults can affect multiplexers, interconnections, buffers, LUTs, control bits, and flip-flops [12]. For example, researchers in the Rosetta experiments [1] reported the measurements of approximately 295 and 290 failure in time per million bits of configuration cells in $0.15 \mu m$ and $0.13 \mu m$ technologies, respectively. They also reported 265 and 530 failure in time per million bits of block memory cells. Since these SRAM cells control the FPGA functionality this study of errors caused by radiation is very important. In the Rosetta experiments, the chips were placed in a radiation chamber in order to measure the failures from exposure to real particles. This method of measuring failures in time with a radiation chamber is normally extremely expensive. The likelihood of these errors increases at both higher altitudes and in space applications where there is larger number of energetic particles emanating from the sun and other galaxies. Hence resistant design against errors caused by radiation in satellite and space applications is an important area of study.

As technology advances, dimensions decrease, supply voltages and capacitances lower, and clock frequencies increase. Technology scaling increases the likelihood of errors caused by radiation in semiconductor devices. Density of components increases with minimizing dimensions as technology advances. In a system, denser circuitry results in higher number of sensitive nodes in the same area compared to older technologies. Since the clock frequencies will continue to increase the likelihood that a momentary glitch will be clocked as valid data increases and this error is then propagated through the logic path. Consequently, these technology trends conspire to increase semiconductor devices susceptibility to radiation. Thus, designing for error resistance is important not only for space but for other applications as well.

Reliability becomes a critical issue especially in large scale systems using multiple FPGAs, since the failure in time of a system increases linearly with each additional FPGA. For instance, in applications such as banking or ehealth, numerous FPGAs as high performance servers can be used to provide security services such as confidentiality or authentication. Additionally, reliability is an important issue due to error sensitivity of AES. For example a single bit flip in the early rounds of

AES encryption is expected to in 50% erroneous bits in the output. Therefore, designing a highly reliable cipher in an FPGA is very important.

The main focus of this research is on the reliability issues of the symmetric-key algorithm, AES, implemented on an FPGA. Traditionally, there have been expensive techniques in terms of hardware resources (e.g., triple modular redundancy uses more than 3 times as many hardware resources as the original design implementation) that provide error detection and correction to increase the reliability of a system. On the other hand, there have been other less expensive techniques (e.g., a parity technique providing error detection) but they have not provided reliability for all elements in a design such as one implemented on an FPGA. Hence there is a need to provide adequate reliability at the lowest possible cost. One of the main goals in this research is to enhance the reliability of a low cost parity scheme in order to improve the error coverage. Weaknesses of a general parity technique on the FPGA are researched. Given that current FPGAs are very dense (e.g., Virtex-II Pro FPGA has 34,292,768 SRAM cells [2]), and that the mapping details of the placed and routed FPGA design are proprietary, analyzing the effect of faults on an implementation is very difficult. In order to tackle this obstacle, this research exploits the high regularity in the FPGA (e.g., the same basic building block containing combinational and sequential elements is repeated throughout FPGA).

The pin fault model (faults are modeled as occurring on input and output pins of FPGA components e.g. look up tables, block memory, etc) is used for modeling and analysis of a wrong value in an SRAM cell due to radiation. Then a small portion of the AES is tested by simulating single errors (through flipping 1 bit at a time in the configuration file) to verify this model. Since the FPGA structure is regular, the result of this verification for this simple yet sufficiently accurate model is expanded for the whole device.

Parity techniques cover single errors in datapath registers, however, errors due to radiation can occur in the control circuitry as well as in SRAM cells forming the combinational logic and routing of a design implemented on FPGAs. Unlike previous research, propagation of single errors is investigated in the AES netlist after placement and routing. In combinational logic or routing, there are 2 situations when an error can be undetected. First, if a single error potentially affects an even number of data bits. Second, if both data and parity bit are affected by a single error. In these cases, LUTs are designed manually in the netlist and extra flip-flops are used at the register-transfer level to resolve errors being undetected in combinational logic and routing of FPGA, respectively.

There are some parts in the AES algorithm that can benefit from elements available in state of the art FPGAs to provide error detection. Therefore, the parity scheme is not used for all transformations in AES. For instance, unlike previous research, a dual ported block memory is used in *SubBytes* of AES to provide error detection. Additionally, duplication is used for error detection within the design of the control circuitry. By utilizing very few LUTs the overhead of the control circuitry is low.

An error in an FPGA can significantly modify the functionality of a design. Therefore, if these errors are not removed by reconfiguration, the dependability of a correction technique is questionable. In order to correct errors, self reconfiguration is demonstrated in this research. In case of an error occurrence, the AES module interrupts the integrated processor on the FPGA for a reconfiguration.

The contributions of this research include items as follows.

- Errors caused by radiation in SRAM-based FPGA are modeled by using the pin fault model. Then this simple and accurate enough modeling is verified by simulating errors in the configuration file.
- Radiation faults are simulated by flipping bits of interest in the configuration file that is downloaded on the FPGA. The desired net is removed in the placed and routed netlist to generate the modified configuration file. Then the modified and original configuration files are compared in software written in C++.
- The weaknesses of the parity scheme in error detection are found through a small design implemented on FPGA. The effects of a fault propagating to output, while simulating errors by flipping bits in the configuration file, are observed.
- The error coverage of the parity scheme is expanded from the datapath flip-flops to the control circuitry, logic blocks, and routing. This improvement is done by mitigating the weaknesses in the parity scheme at the register-transfer level.
- The insufficiency of known error correction techniques such as the triple modular redundancy and Hamming code in FPGAs is analyzed and self reconfiguration is suggested instead.
- AES with the enhanced parity scheme is designed and implemented on the FPGA as an IP core. Interfacing of the master IP core capable of interrupting the PowerPC 405 processor is implemented.

1.1 Thesis Organization

The outline of this thesis is as follows. In Chapter 2, background on radiation effects causing faults in semiconductor devices is presented. The sources of radiation effects and their mechanisms causing errors in DRAM and SRAM cells are discussed. Definition and comparisons of various faults including soft errors in a device are provided. Soft errors are dynamic radiation-induced faults which generally cause a storage bit to change its value. Methods of estimating errors caused by radiation in a device are described. The relationship between technology trends and errors caused by radiation is also presented.

In Chapter 3, soft error mitigation techniques at different levels from hardware to software are presented. Peculiar effects of soft errors in SRAM-based FPGAs are discussed and also compared to effects in ASICs.

In Chapter 4, general security requirements of a system such as authentication, data confidentiality, data integrity, and non-repudiation are briefly described. Then the primitives to provide these security needs (i.e. symmetric-key algorithms, public-key algorithms, and hash functions) are presented. Emphasis is placed on the standard symmetric-key algorithm, AES, which is the focus in this research. National Institute of Standards and Technology (NIST)-recommended block cipher modes are also covered briefly. Error propagation within the AES algorithm (due to confusion and diffusion properties) and error propagation in various cipher modes of operation are discussed. Different architectures previously used to implement AES are presented.

In Chapter 5, the proposed AES with soft error detection is introduced. The proposed error detection technique uses mathematical properties of AES and available hardware resources on FPGA to detect errors in *SubBytes* and the control circuitry implementations. Enhancements to the parity scheme (used for error detection in *MixColumns* and *AddRoundKey*) to increase its error coverage are also proposed in this research. In order to increase the error coverage of the parity technique, the weaknesses of it on FPGA are found and mitigated in the combinational logic and routing.

In Chapter 6, experimental results and comparisons are discussed. Chapter 7 presents the conclusion and future work. The thesis also includes 8 appendices. Appendix A provides a glossary of acronyms. In Appendix B, different aspects of design and implementation of the AES module as a system on chip on FPGA are covered. This system has the capability to self reconfigure in case of an error through the host PC. Communication and synchronization of the proposed AES module, the

Chapter 1: Introduction

PowerPC 405 (integrated) processor on the FPGA, and the host PC are described. Device drivers that provide interrupt handling of the PowerPC 405 processor and the host PC are presented. Appendix C illustrates the state machines of the control circuitry in the proposed AES. In Appendix D, the device driver code for FPGA reconfiguration is presented. Appendix E shows the processor local bus interface. S-box of AES and pseudo code for key expansion are provided in Appendices F and G, respectively. In Appendix H, routed netlist snapshots from FPGA Editor are provided.

Chapter 2

Radiation Effects on Devices

This chapter provides an introduction to radiation effects in semiconductor devices. Radiation sources and their effects are briefly presented. Specific mechanisms causing errors through radiation effects in storage elements including DRAM and SRAM are discussed. A comparison of radiation faults, faults injected by an attacker in cryptographic algorithms such as AES (the focus of the research), and faults caused in manufacturing is presented. Methods of estimating soft errors, which are caused by radiation, in a device are described. The relationship between technology trends and soft errors is also presented at the end of this chapter. Before proceeding to details, some terminologies used in this thesis are described for clarification.

2.1 Radiation Sources and Soft Error Mechanisms

Faults in a device do not necessarily cause errors in output results. For instance, in an FPGA a fault that happens in an unused resource does not cause an error at the output. Error mitigation techniques do not have to necessarily provide error correction. On the other hand, tolerance to error includes correction of error.

Energetic particles hitting an electronic device cause disturbances which typically are referred to as single event effects. Researchers have discovered various sources causing these effects either directly or indirectly through nuclear reactions between the particles and other materials in the struck device.

In addition to short-term radiation effects, other effects that are the result of cumulative long-term ionizing damage in a device are known as total ionizing dose effects. These effects encompass those that appear from long-term absorption of radiation over time. On the other hand, short-term effects are caused by the passage of a single ionizing particle through a device. Total ionizing dose effects can cause devices to suffer threshold shifts, increased device leakage and power consumption, timing changes, and etc [3]. Fabrication process-based techniques such as epitaxial CMOS process and silicon-on-insulator can reduce total ionizing dose effects [4]. Most solutions to total ionizing dose effects are fabrication process-based techniques or replacement of parts. The main focus of this research is on the short-term upsets caused by radiation. These short-term effects are provided later in this chapter.

Radiation effects have become an issue due to technology scaling. One of the first manifestations of radiation effects in literature were found in the process toward higher levels of integration in memory circuits at Intel [5, 6]. Specifically, researchers at Intel observed significant error rate increase in DRAM as integration density increased to 16kb and 64kb in the late 1970s. As shown in Figure 1 (SER refers to soft error rate), DRAM single bit error rate due to radiation has decreased by a factor of 4 to 5 per generation since then by mitigation techniques discussed in Chapter 3. However, due to increased demands for memory density (memory size per system being a microprocessor with embedded memories) the overall system error rate shown in Figure 1 caused by radiation has remained approximately the same. For instance, the size of the main memory of microprocessors has increased from 1kb in the 1970s to beyond 1Gb. Thus, there is significantly larger number of memory cells that can potentially cause a system failure in a modern microprocessor.

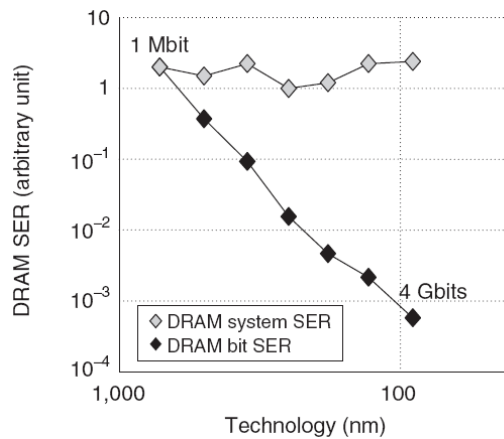


Figure 1 DRAM error soft error rate caused by radiation [7]

Compared to DRAM, early SRAM was more robust to single event effects. In recent technology (less than $0.25\mu m$), SRAM bit error rate has not increased. However, exponential growth in the amount of SRAM for instance in microprocessors and digital signal processors has resulted in the system error rate to increase with each generation. This trend is of great concern to manufacturers because SRAM constitutes a large part of all advanced Integrated Circuits (ICs) today.

A particle can pass through a semiconductor material, free electron-hole pairs along its path, and deposit energy directly. In addition to that, a particle can lose energy through indirect mechanisms by interacting with the struck material. Since radiation effects were discovered in semiconductor devices, 3 main sources described below have been found that are responsible for single event effects at terrestrial levels [7].

- One source is alpha particles that can be emitted by small traces of radioactive impurities, such as uranium and thorium, in packaging materials. The extent of radiation depends on the quality and purification grade of the materials. Another source of alpha particles is solder bumps especially those that are near areas sensitive to radiation, for instance, SRAM and DRAM cells on a chip.
- Another source of single event effects is particles generated when ever-present cosmic rays enter earth's atmosphere. For instance, high energy neutrons are one of these predominant particles. It should be noted that disturbances caused by these particles are more significant

in higher altitudes. High energy neutrons striking a device can cause displacement in the silicon lattice.

- Interactions of cosmic rays with materials on the struck device can also cause single event effects indirectly; an example is interaction of low energy neutrons in cosmic rays and boron. Boron is a dopant used in silicon devices for the formation of p-type regions. Boron is also in borophosphosilicate glass (BPSG) used in insulator layers of a device in older technologies (e.g., 0.25 and 0.18 μm SRAM cells fabricated with BPSG).

Single event effects can manifest themselves in various ways in an implementation. They might change the logic state, cause a transient disruption, or some types might even lead up to permanent destructive failures in a device. In general, if the damage is unrecoverable in a device it is considered as a hard error. Depending on their damage, single event effects are classified as follows [8][9].

- A single event latch-up occurs when current forces through the substrate; this might destroy the device.
- A single event gate rupture happens when there is a conducting path in the gate oxide; this destroys the gate control structure.
- A Single Event Functional Interrupt (SEFI) triggers an operation of the support circuitry and stops the normal operation of the device. The support circuitry in FPGAs provides the configuration capability, power on, reset, JTAG functionality and etc. For example, an SEFI that affects the power-on-reset circuitry can cause the current design on the FPGA to be lost in an attempt by the power-on-reset circuitry triggering reconfiguration [10].
- A Single Event Transient (SET) occurs when the charge collected generates voltage/current transitions which are commonly known as glitches.
- A Single Event Upset (SEU) flips the value of a single storage element or memory cell. For example, this could be a Look Up Table (LUT) element, D flip flop, block memory cell, or configuration memory cell affected by an SEU. Hence, SEUs are an important concern in SRAM-based FPGAs.
- A Multiple Bit Upset (MBU) causes more than one adjacent bit to flip depending on its strike angle.

Soft errors are recoverable errors. It should be noted that the likelihood of SEFIs and MBUs is extremely low. Therefore, the main concerns in soft errors are SEUs and SETs. It should be noted that MBUs are important in differential fault attacks discussed in Section 2.2.2.

The magnitude of the disturbance a particle causes depends on its Linear Energy Transfer (LET) that is defined as deposited energy per unit length. The unit of LET is typically $MeVcm^2 / mg$, since energy per unit distance length (MeV / cm) is normalized by the density of the material struck (mg / cm^3). For instance, in silicon, an LET of $97 MeVcm^2 / mg$ corresponds to a charge deposition of $1 pC / \mu m$ [11]. Typically, more massive and energetic particles in denser materials have higher LET [7].

In general, charge collection happens within a micron or 2 of the junction (boundary interface where the 2 regions of the semiconductor meet). The reverse-biased junction (p-type and n-type regions are connected to negative and positive voltages, respectively) is usually the most sensitive part of a circuit in charge collection, in particular if the junction is floating or weakly driven [7]. The high electric field in the reverse-biased junction depletion region (an insulating region with no free charge carriers) assists collection of charge. Strikes near a depletion region can also result in efficient charge collection.

At the beginning of an ionizing radiation event, a track of electron-hole pairs in the form of cylinder is shaped Figure 2(a). This cylindrical track with a submicron radius has high carrier concentration. Then the electric field rapidly collects the carriers causing a glitch Figure 2(b) and forms the field funnel [7]. This funneling effect, shown in Figure 2(b), increases charge collection at the struck node by extending the high electric field at the junction deep towards the substrate. This part of charge collection phase completes within tens of picoseconds. The following phase in charge collection is diffusion in which electrons diffuse into the depletion region. Diffusion takes longer in the range of hundreds of nanoseconds. It ends when all excess carries are collected, recombined, or diffused away from the junction.

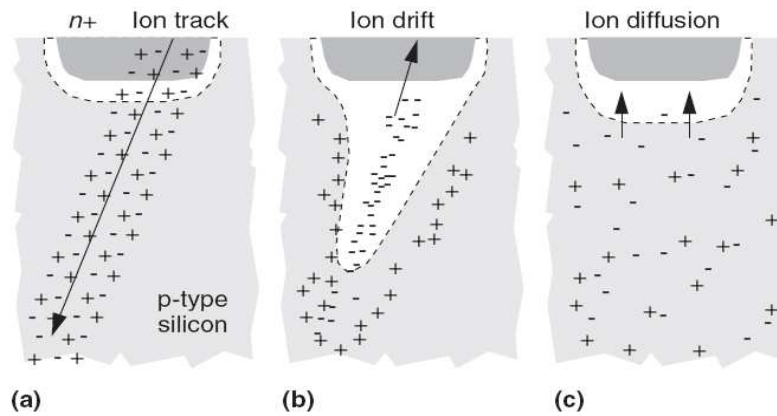


Figure 2 Charge collection in a reverse-biased junction: (a) formation of electron-hole pairs, (b) funnel shape field extending deep into substrate, (c) diffusion process [7]

In order to further clarify errors by soft errors, the mechanisms causing errors in DRAM and SRAM cells are briefly presented. The DRAM cell illustrated in Figure 3 has an access transistor and a storage capacitor. There are 2 main parameters related to DRAM errors when an ion strikes a cell. The first parameter is the critical charge (denoted Q_{crit} is defined as the minimum amount of charge collected at a sensitive node that can cause an error) that is closely connected to the concept of noise margin. The second parameter is the critical time window when the disturbance can get stored in the DRAM cell. Due to dynamics of the DRAM cell, timing of the strike is also an important factor; meaning the strike has to happen in the critical time window.

As shown in Figure 3, one of the most sensitive parts in the DRAM cell is the storage capacitor and the source of the access transistor. Ion strikes at these 2 nodes directly affect the stored charge and consequently the information stored in the DRAM cell.

Errors can be caused by ion strikes at bit lines as well. This happens when bit lines are in a floating voltage state, for instance, during a read operation. Therefore, an ion strike must happen in this critical time window. The disturbance caused at bit lines can reduce the sensing signal due to charge imbalance either prior to or during the sensing operation (when the sense amplifier amplifies the small differential voltage between the bit lines to the full swing).

Another mechanism (named the combined cell-bit line failure mode [12]) that can cause an error in the DRAM cell is the combination of the 2 mechanisms discussed above. In the combined cell-bit line

Chapter 2: Radiation Effects on Devices

failure mode, either of the mechanism above does not individually exceed Q_{crit} . However, when combined together, they can cause an error.

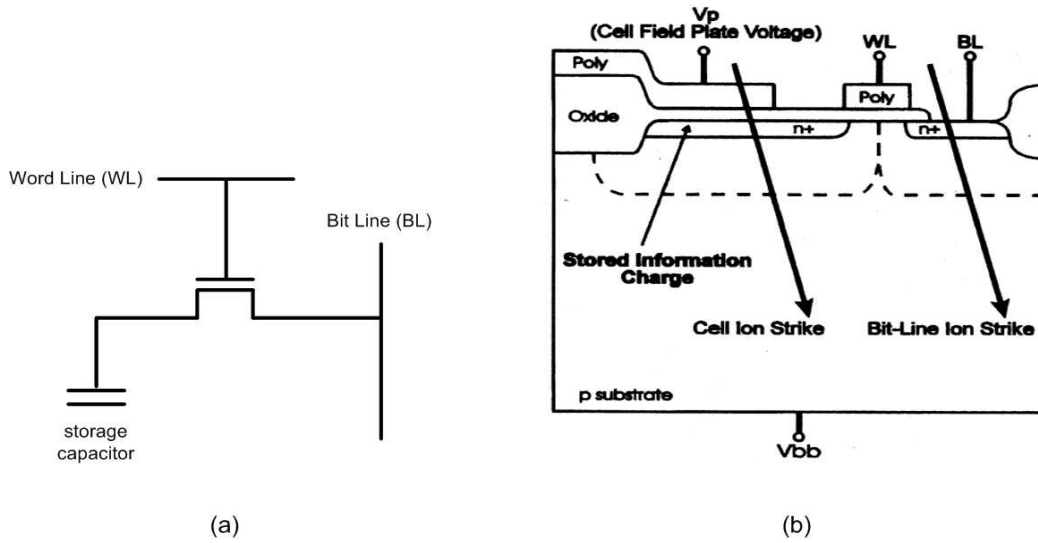


Figure 3 DRAM cell: (a) circuit, (b) layout [11]

The error mechanism in the SRAM cell is quite different from the DRAM cell due to the feedback loop formed by the cross-coupled inverter pairs in the SRAM cell circuit. A typical SRAM cell with 6 transistors is shown in Figure 4 where the positive feedback loop is formed by cross-coupled inverters $Q_1 - Q_3$ and $Q_2 - Q_4$ [13].

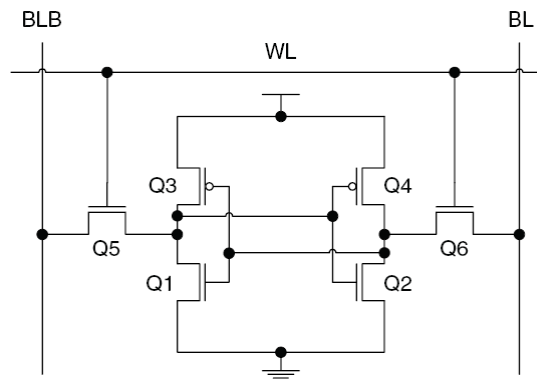


Figure 4 Six transistor SRAM cell [13]

Typically, the most sensitive part in the SRAM cell is the reverse-biased drain junction of a transistor when it is off [11]. Charge collected by the junction lead to a transient current in the struck transistor. This triggers a response in the SRAM cell that is similar to a write pulse and can cause a bit flip in the SRAM cell.

For instance, in the 6 transistor SRAM cell shown in Figure 4, the node storing a ‘1’ is the most sensitive node to errors. The reason behind this is the low Q_{crit} for a $1 \rightarrow 0$ transition. A $0 \rightarrow 1$ transition needs Q_{crit} that is about 22 times larger than that of a $1 \rightarrow 0$ transition [13]. The state of the storage node storing a ‘1’ is supported by a relatively weak PMOS pull-up transistor. Consequently, Q_{crit} of an SRAM cell is defined by Q_{crit} of a node storing a ‘1’.

2.2 Comparison of Soft Errors to Other Faults

In this section, we discuss the causes of other faults found in implementations, including faults during the manufacturing of a device and faults injected during a cryptographic attack. These are all different in nature; therefore, they need relevant mitigation techniques. The basic differences among these different faults are briefly described as follows.

2.2.1 Manufacturing Faults

Manufacturing faults are usually due to deformation of IC elements. There are global faults in manufacturing processes that affect large areas of fabricated silicon wafers in a uniform manner. On the other hand, there are spot faults affecting a very small area of fabricated silicon area which are much more difficult to detect. Spot faults are in general due to an extra or missing material in one of the layers (i.e. conductive, semiconductive, and insulating layers). Spot faults include: shorts due to extra conducting/semiconducting material or missing insulating material, breaks due to missing conducting/semiconducting material or extra insulating materials, new parasitic elements, and elements with degraded performance [14].

Manufacturing faults can be intermittent due to unstable or marginal hardware. As opposed to permanent faults that always exist, intermittent faults do not happen all the time. Their occurrences (activation and deactivation) depend on environmental conditions. Intermittent faults happen repeatedly at the same locations, while soft errors are random in space. Additionally, when the environmental conditions are encouraging, intermittent faults tend to happen in bursts, while soft

errors are random in time. Another key difference is that replacement or repair can resolve intermittent faults, but this is not the case for soft errors [15].

Fault models typically represent consequences of faults at the abstracted logic level. For instance, the stuck-at model indicates a wire is a '0' (connected to ground) or '1' (connected to supply voltage). Bridging faults between outputs and can be modeled by logic gates.

One major group of manufacturing faults is opens and shorts described by the stuck-at modeling. For instance, this can happen in a pass transistor as part of the programmable interconnect point in routing of FPGA. This pass transistor is controlled by an SRAM cell, shown in Figure 5. A short or open causes the pass transistor to be permanently closed or open, regardless of the value of the SRAM cell controlling it [16]. However, when an SEU happens in the same scenario, it flips the value of the SRAM cell controlling the pass transistor. The case of an SEU is similar to loading the FPGA with a different configuration file. The configuration file (also known as programming or bitstream file) of the SRAM-based FPGA defines an implementation (basically, the values stored in LUTs and block memory, the interconnection between resources, and the modes of the resources e.g., I/O standards, I/O drive strengths and LUT modes) [17].

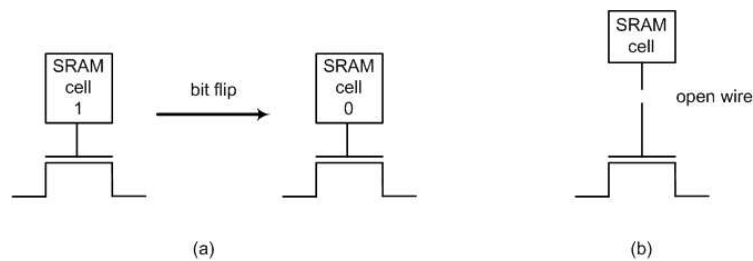


Figure 5 Faults: (a) soft error causing bit flip (b) manufacturing open wire

Manufacturing fault detection techniques usually include loading test configuration files and applying test vectors to detect faults. Test vectors can be generated externally or internally as in built-in self test. Since SEUs do not cause configuration-independent affects compared to shorts and opens and can be fixed by reconfiguration, the above offline method for detecting manufacturing fault does not apply to SEUs.

2.2.2 Fault Attacks

There are faults that are injected by an attacker to obtain some secret information. The goal in fault attacks is to replace the valid results with invalid results (causing errors in results) to perform the cryptanalysis. There are various ways to cause an error during a cryptographic operation, e.g., variations in supply voltage so the processor misinterprets or skips instructions [18], variations in the external clock, and lasers.

Fault injection can be with or without contact [19]. In a fault injection with contact, there is direct physical contact with the chip (e.g., at a pin) to cause voltage or current changes. For instance, in spike attacks the supply voltage is set to violate the operating voltage range tolerated by the chip. This voltage variation can be exploited to produce a wrong result that might be useful for fault analysis to get some secret information from the cryptosystem. An attacker needs to find specific parameters for a spike (in terms of timing and shape of a spike) to produce wrong results that could be used to perform the cryptanalysis successfully.

Another approach to inject errors within the cipher is to manipulate the clock signal so that it violates its operating characteristics [20]. For instance, the operating clock voltage or the rise or fall time can be set so that it will not fit within the proper range. However, generating a clock signal that is deviated in such a way that causes the desirable wrong ciphertext might be a challenge, since changing the clock signal can potentially cause the chip to lose its functionality.

In contrast to fault injection with contact, techniques such as heavy ion radiation and electromagnetic interference can be used in a fault injection without contact. For instance, light can be used to inject errors in non-volatile memory cells. It was shown that camera flash light can be used to target the memory of a microcontroller to set or reset an individual bit at a specified time [21]. Researchers in [22] showed that a non-volatile memory in a microcontroller can be erased by a UV-C light with the wavelength of $254nm$. They demonstrated an attack on a software AES implementation. It was shown that if a single byte of the S-box is changed and key expansion is not affected, 2500 pairs of correct and faulty encrypted inputs are sufficient to recover the key with a probability of 90% on an 8-bit microcontroller [22].

Electromagnetic emissions can also be used to induce current and target sensitive spots of the chip (e.g., memory) [23]. This approach is not invasive in the sense that one does not need to open the chip.

The accuracy of fault injections, in order to extract secret information, is an important factor in a successful cryptanalysis. For instance, the level of control over location and timing of fault injections is important for fault analysis. A technique such as a spike attack is random in terms of the location of faults while an optical attack provides accurate targeting of location [20]. Another important aspect is the number of faults injected. Depending on the fault attack, a single fault or multiple faults may be needed for cryptanalysis.

The first theoretical model for breaking cryptosystems by exploiting random hardware faults was introduced in [24]. In this research, it was shown that fault attacks were effective in the RSA system and Rabin signatures. The attacks were also applied to the Fiat-Shamir and Schnorr identification schemes. Fault attacks depend on how a cryptosystem is implemented. For instance, the fault attack in [24] on RSA is effective on the Chinese remainder based implementation. The attack in [24] is briefly summarized as follows. Based on the Chinese remainder the RSA signature E can be computed as follows.

$$E = aE_1 + bE_2 \pmod{N} \quad (1)$$

where N is the RSA modulus ($N=p*q$; p and q are prime), $E_1 = Message^{private\ key} \pmod{p}$ and $E_2 = Message^{private\ key} \pmod{q}$.

Assuming E' is a faulty signature and $E'_2 = E_2$ (meaning no faults during computation of E_2) it is observed that

$$\begin{aligned} E - E' &= aE_1 + bE_2 - (aE'_1 + bE'_2) \\ &= a(E_1 - E'_1) \end{aligned} \quad (2)$$

If $(E_1 - E'_1)$ is not divisible by p , then N is factored as shown in Equation 3 and the cryptosystem is compromised.

$$q = \gcd(E - E', N) \quad (3)$$

There are also reported fault attacks on symmetric-key algorithms. Differential Fault Analysis (DFA) was proposed by researchers as an attack against DES in [25]. The full DES key was extracted by analysis of 50 to 200 ciphertexts which were generated from unknown but related plaintexts [25]. Additionally, the same attack (with the same number of given ciphertexts) applied on TDEA resulted in a successful cryptanalysis.

Since AES does not have the Feistel structure as in DES, it is not possible to apply the fault attack introduced in [25] on AES. DFA against the AES was subsequently proposed in [20, 26-28]. Intermediate states are changed by faults injected in these attacks. There are also DFA against the key expansion of AES reported in [29, 30]

As shown in Figure 6, an attack based on injecting single faults in the intermediate result of the initial *AddRoundKey* transformation was introduced in [20]. The plaintext is set to 0 (every bit is a '0') and it is assumed that the attacker knows the correct ciphertext and his goal is finding the key. It is observed that

$$\textit{initial AddRoundKey result} = 0 + \textit{initial Round Key} \quad (4)$$

Or equivalently:

$$\textit{initial AddRoundKey result} = \textit{initial Round Key} \quad (5)$$

Then the attacker injects a '0' at every bit location of the *initial AddRoundKey result*. If the round key bit is a '0' then the ciphertext is correct. On the other hand, if the round key bit is a '1' then the ciphertext is wrong. Since it is assumed the attacker knows the correct ciphertext he is able to distinguish a wrong ciphertext from the right ciphertext. This process is repeated 128 times for all the key bits and the complete key is found in this cryptanalysis.

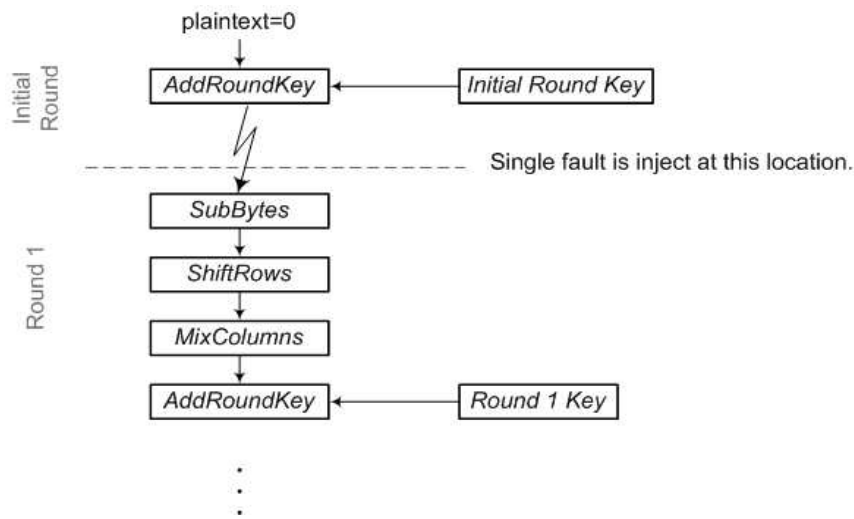


Figure 6 AES fault injection [20]

The assumption of the attacker being able to attack a specific single bit within the first round key is strong in the sense that it might not be practically feasible. This is where probabilistic fault analysis comes into play. In this case, the probability of an attack being successful or unsuccessful is also considered.

Another technique proposed in [20] is based on using the timing attack on AES (this was suggested by Koeune and Quisquater [31]) in a fault based cryptanalysis. Approximately 16 faulty ciphertexts were claimed to be sufficient to extract 1 byte of the key.

Researchers in [30] described 2 different attacks on AES by injecting faults. The first attack assumes a fault on only one bit of an intermediate result at the beginning of the final round. The location of a fault should be chosen. Fifty faulty ciphertexts were used to obtain the key completely. The second attack [30], which is more realistic, considers injecting faults in a whole byte. Researchers then performed differential fault analysis that resulted in obtaining the full 128-bit AES key with less than 250 faulty ciphertexts [30].

Researchers in [26] supposed that a single byte of the state after the *ShiftRows* of round 9 can be changed and the index of the faulty element of state is known. It is assumed the new value of the element of the state is unknown. The injected fault spreads over 4 bytes of the output state. A set of possible fault values for each faulty element of the output state is found. Possible fault values are

intersected for all the 4 faulty elements to reduce the number of required ciphertext for the cryptanalysis. Then possible values for 4 elements of the last round key are deduced. Four bytes of the 10th round key is obtained by more ciphertexts [26].

In fault attacks, faults are aimed at results (or registers). However, in the case of soft errors the perspective is broader in the sense that faults can affect registers, combinational logic and routing on a FPGA. Therefore, it is important that faults affecting parts other than registers be considered throughout the design as well. The occurrence of multiple faults is important in fault injection [32] since it is easier for an attacker to inject multiple faults (e.g. target a byte, than a single bit). On the other hand, in radiation faults the likelihood of single faults is exceedingly higher than multiple faults [33]. As opposed to fault injection mitigation techniques, the accumulation of faults is important in the case of SEUs. For instance, a strong method that detects and corrects errors is not sufficient, since the accumulation of faults over time will destroy the functionality of the method itself.

In this research, the main goal of error detection is in terms of reliability. However, the proposed technique can be applied against cryptanalysis of AES in the circumstances that follow. Multiple errors are detected in the *SubBytes* transformation, as will be described in Section 5.1.1. Therefore, the proposed error detection technique may be used against faults attacks where multiple attacks are injected during *SubBytes*. For *MixColumns* and *AddRoundKey* transformations, detection of a single or an odd number of errors is provided, as described in Section 5.1.2, 5.1.3, 5.2.2, and 5.2.3. Therefore, in scenarios where the number of bits flipped in these two AES transformations is not always even, the proposed method may be used against fault attacks. For example, attacks such as [20] (a fault is injected after the initial *AddRoundKey*) or the first attack in [30] (a fault injected at the beginning of the final round) may be thwarted by the proposed thesis work.

2.3 Estimating Soft Error Rates of a Device

Estimating soft errors of a device is complex and estimations of different studies also vary. There are 3 known approaches for the soft error rate estimation: accelerated testing using particle beams, software simulation of circuit, and estimation by real particles.

In accelerated testing [1][34], a device is subjected to particle beams generated by accelerators. For instance, researchers in [1] at the Los Alamos National Laboratory have used a linear accelerator that produces an 800MeV pulsed proton beam that strikes a water cooled tungsten target. This generates a

spectrum of neutrons whose energy distribution and intensity are precisely measured. The energy of this spectrum, which is similar to atmospheric spectrum, is in the range of $1MeV$ to $600MeV$. There is also a neutron flight path ($20m$ from the neutron production target) with an irradiation building where a device to be tested is placed in. The neutron beam is controlled by opening and closing a shutter. The angle of incidence is an important parameter in the estimation results of the soft error rate. The flux increase for higher than the sea level altitudes can be computed by assuming a 30% increase for every $1000ft$ rise in altitude (a commonly used rule to the measured flux versus altitude below $40000ft$). It should be noted that predicting atmospheric neutron flux in accelerated testing using particle beams is not exact. This is shown in [35] through investigation of different studies on terrestrial neutron flux. If the particle beam experiments are intended to predict actual soft error rates, the results can be different by a factor of 10 since there is a wide variation in energy versus flux as reported by the various studies [1].

Another approach to estimate soft error rate is software circuit simulation to determine the critical charge a particular node or latch can handle before it changes state, causing a bit flip. Different models of Q_{crit} have been proposed in [36][37-40]. Furthermore, the impact of process variation on Q_{crit} is investigated in [41]. Models that have been reported generally agree on the qualitative definition of Q_{crit} ; however, they are different in quantitative description [41]. Each modeling of Q_{crit} has its own limitations, for instance, the impact of a parameter might be underestimated or overestimated according to certain assumptions.

Another method to estimate soft error rate is to subject devices to real atmospheric radiation. In order to provide sufficient data, a large number of devices go under test. An extensive test on FPGAs is conducted in the Rosetta experiment [1] at different altitudes. The experimental setup in [1], shown in Table 2.1, is costly in terms of time and hardware to get large amount of data.

Table 2.1 FPGAs under test in Rosetta experiment [1]

Node technology in <i>nm</i>	Die	Locations	Altitude in feet	# of devices	Device hours
150	2V6000	San Jose	0	100	1060000
		New Mexico	5100	100	1670000
		White Mountain	12470	100	856800
		Mauna Kea	13200	100	353000
130	2VP50	San Jose	0	200	1191000
		New Mexico	5100	200	709000
		White Mountain	12470	200	655000
90	S31500	San Jose	0	500	256000
90	V4LX25	San Jose	0	100	20000

The Failure In Time (FIT, where 1 FIT is 1 bit flip in billion hours of a device) values of configuration cells at the sea level in the Rosetta experiments are obtained for different technologies as follows: 295 FIT/*Mb* for the 0.15 μm node, 290 FIT/*Mb* for the 0.13 μm node. Furthermore, there are 265 FIT/*Mb* and 530 FIT/*Mb* for block memory cells in the Rosetta experiments.

Soft errors increase as the altitude increases; therefore, applications suffer more severely in space than at terrestrial levels. Table 2.1 shows the mean time to error for a Virtex-II FPGA in a geosynchronous orbit in the case of SEUs in configuration memory, block memory, and power-on-reset circuitry SEFI [9].

Table 2.2 Mean time to error of Virtex-II (device XQR2V6000) in a geosynchronous orbit [9]

	Mean time to error
SEU in configuration cells	1.8 hours
SEU in block memory	11.8 hours
Power-on-reset circuitry SEFI	221 years

The FPGA architecture uses memory cells to control every programmable function and feature. Therefore, adding costly redundant circuitry to tackle soft errors can make the FPGA cost too high to be commercially viable.

2.4 Technology Trends and Soft Errors

As technology improves, dimensions decrease, supply voltages and capacitances lower, and frequencies increase. Density of components increases with minimizing dimensions as technology advances. Consequently, denser circuitry results in higher number of sensitive nodes in the same area compared to older technologies. Furthermore, smaller layout dimensions reduce capacitance of a node; this reduces Q_{crit} which is related to noise margin as well. Thus, a smaller charge deposited can upset a node. Q_{crit} is further decreased by lowering supply in advanced technologies. Since the clock frequencies continue to increase as technology advances the likelihood that a momentary glitch (SET) is propagated through the logic path and clocked as valid data increases. Consequently, all these technology trends unfortunately conspire to increase semiconductor devices susceptibility to radiation. Thus, designing for soft error resistance is important not only for space but for other applications as well.

DRAM cells were among the most vulnerable elements in earlier technologies in the late 1970s. Early DRAM cells stored a bit value in 2-dimensional $p-n$ junctions. Those DRAM cells were highly sensitive to radiation due to large planar reverse-biased junctions. The more compact 3-dimensional design of DRAM cell with a much smaller charge collection at $p-n$ junction significantly decreased the vulnerability of DRAM cells to radiation. This 3-dimensional design even compensated other adversely contributing factors (e.g., shrinking supply voltages) such that the soft error rate decreased for next generations, overall. However, the system soft error rate remains approximately the same due to denser DRAMs in a system in recent generations.

Compared to early DRAMs, early SRAMs were more robust against radiation mainly due to the feedback loop in their structure. However, in recent technologies, the SRAM cell area and therefore the junction area as well as the supply voltage has decreased. All these factors increase sensitivity to radiation. The SRAM bit soft error rate is saturated for technology nodes beyond $0.25 \mu m$ [33] due to the saturation of V_{DD} scaling, reductions in junction collection efficiency of highly doped $p-n$ junctions, and the increased charge sharing between the neighboring nodes. However, the exponential

growth of SRAM density in state of the art processors has led the SRAM system (referring to a microprocessor with embedded memories) soft error rate to increase with each technology generation.

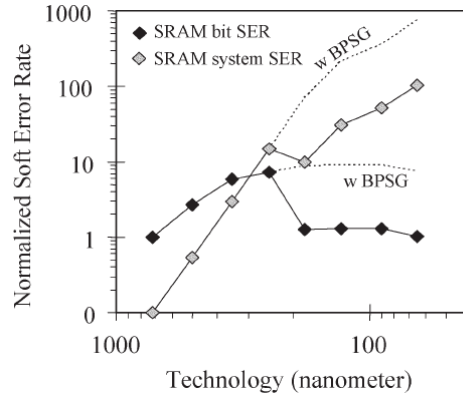


Figure 7 SRAM system soft error rate increasing while bit soft error rate being about constant [33]

If an SET (radiation-induced glitch) actually propagates to the input of a latch or flip-flop and meets the timing requirements (setup and hold times), the erroneous input will be latched and stored. In older technologies, an SET could not propagate because it usually could not produce a full output swing or was quickly attenuated because of large load capacitances and large propagation delays. In advanced technologies, where the propagation delay is reduced and the clock frequency is high, an SET can more easily traverse many logic gates, and the probability that it is latched increases. SET-induced soft errors are not expected to become an issue until the technology reaches or goes beyond the 65nm node [7]. Once an SET can propagate easily, synchronous and especially asynchronous (without clock signal) circuits would be extremely sensitive to such events. In technology beyond 90nm and at high operating clock frequencies, there is increased likelihood that a large fraction of observed soft errors will be related to SETs being stored [7].

An important bottleneck in recent technologies is the increase in the sub-threshold leakage power due to decrease in the threshold voltage (the threshold voltage is lowered to maintain enough gate overdrive and improve performance). Lowering the threshold voltage causes the transistor sub-threshold leakage current to increase exponentially. Therefore, the supply voltage is lowered to minimum level to reduce the sub-threshold leakage power. However, lowering the supply voltage

decreases Q_{crit} and thus increases vulnerability to radiation. It is important to consider the power reduction benefit vs. reliability concerns when choosing the minimal supply voltage [13].

2.5 Summary

The main goal of this chapter was to provide some background on radiation effects in semiconductor devices, known as single event effects. The sources of single event effects and their mechanisms causing errors in a DRAM and SRAM cells were discussed. The focus of this research is on soft errors including SEUs and SETs that are of great concern with respect to reliability.

A comparison of radiation faults, faults injected by an attacker in cryptographic algorithms, and faults caused in manufacturing was presented. Methods of estimating soft errors that are accelerated testing using particle beams, software simulation of circuit, and estimation by real particles, in a device were described.

The relationship between technology trends and soft errors was also presented. It was shown that the technology scaling (decreasing dimensions, supply voltages, and capacitances while increasing clock frequencies) increase the likelihood of soft errors in a system.

In the next chapter, previous research on tackling soft errors at different levels is presented. In SRAM-based FPGAs, peculiar effects of soft errors are discussed and also compared to effects in ASICs.

Chapter 3

Previous Research on Tackling Soft Errors

Soft error mitigation techniques range from low level hardware all the way up to software techniques. In this chapter, different techniques at the fabrication process level, circuit and system level including error detection and correction codes, and also software mitigation methods used in microprocessors are presented. Peculiar effects of soft errors in SRAM-based FPGAs are discussed. Furthermore, insufficiencies of ASIC methods which deal with soft errors on FPGAs are pointed out.

The first mitigation step would be to remove the sources causing soft errors that exist inside a device (refer to Section 2.1). These sources are related to the purity of materials used in the manufacturing process and packaging of a chip. For instance, in order to reduce alpha particle emissions, semiconductor manufacturers use extremely high purity materials to make sure they have acceptably low alpha emissions [7]. Another example is removing BPSG, which could cause soft errors indirectly, from virtually all advanced technologies. Figure 7 shows the reduction of soft error rate by elimination of BPSG in SRAM cells. Solders, mold, and underfill compounds with reduced emission rates also need to be chosen carefully.

When all the internal sources of soft errors are eliminated as much as feasible, there are still external sources that cause considerable number of soft errors. For instance, a large portion of high

energy cosmic neutrons will always reach devices and cause soft errors, and thus high energy cosmic neutrons ultimately become the main concern in causing soft errors [33].

Further mitigation techniques (other than removing radiation sources) can be roughly classified into 3 categories. At the lowest level, that is fabrication process (or technology) level, mitigation techniques require fundamental changes to the underlying fabrication technology used to manufacture ICs. Circuit level techniques rely on changes in the circuit design and layout to reduce sensitivity to soft errors. Eventually, any circuit or layout modification that increases Q_{crit} while maintaining or reducing Q_{coll} (collected charge) improves resistance against soft errors. At the highest level in hardware, system level techniques tackle soft errors by applying changes to the architecture of the system. Combining these techniques at different levels might provide the most efficient solution overall for a high-reliability application [7].

3.1 Fabrication Process Level Techniques to Tackle Soft Errors

A fundamental mitigation method for soft errors is to reduce charge collection at sensitive nodes in devices [11]. Substrate structures or doping profiles that decrease the depth from which carriers can be collected can reduce the charge collected. For instance, this can be accomplished in DRAMs and SRAMs by introducing extra doping layers to limit substrate charge collection [42]. In SRAMs, triple-well [43] and even quadruple-well structures [44] (these use multiple-well isolation) have been suggested to decrease sensitivity to soft errors. In multiple-well isolation, all strikes basically happen inside the well. Layers can also be used to provide an internal electric field that opposes collection of charge deposited in the substrate [45, 46]. Even using an epitaxial substrate instead of a bulk substrate reduces charge collection to some extent [11]. An epitaxial substrate consists of a heavily doped, low-resistance bulk substrate topped by a lightly doped, higher-resistance epitaxial layer [47]. The upper layer in is thin and extremely pure semiconductor that is chemically deposited in wafer using a process called epitaxial growth [48]. For instance, the radiation-tolerant Virtex-4QV FPGA technology incorporates a thin epitaxial layer in the wafer manufacturing process for single event latch-up immunity [49]. For each Virtex-4QV device type, the latch-up immunity at maximum V_{CC} and operating temperature, subjected to a heavy ion fluence exceeding $1.107 \text{ particles} / \text{cm}^2$, with LET exceeding $125 \text{ MeVcm}^2 / \text{mg}$, is verified [49].

In silicon devices, another effective technique used for reducing charge collection is the use of Silicon on Insulator (SOI) substrate shown in Figure 8 [50]. In this technology, the active device is fabricated in a thin silicon layer that is dielectrically isolated from the substrate. Therefore, the collection volume is reduced. The source and drain penetrate all the way to the buried isolation oxide in a typical thin-film SOI. Since the reverse-biased drain junction area is limited to the depletion region between the drain and the body of the transistor, this significantly reduces the area sensitive to SEUs. Due to the dielectric isolation in the SOI substrate, charge deposited in the silicon substrate underneath the buried isolation oxide cannot be collected at the drain. As opposed to the SOI substrate, the bulk silicon structure can collect charge from deep within the silicon substrate.

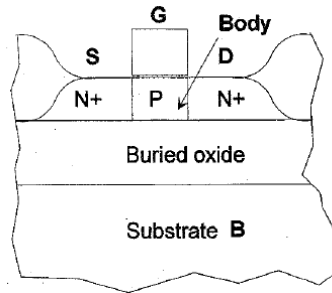


Figure 8 Structure of thin-film NMOS SOI [51]

It should be noted that bipolar capacitive coupling across the buried isolation oxide can lead to unexpected charge collection in SOI structures [52][53]. Charge deposited in the body region can trigger a bipolar mechanism (the parasitic lateral bipolar structure is inherent in all CMOS technologies [50]). This limits the SEU resistance of the SOI substrate [50, 51]. In order to reduce floating-body effects causing parasitic bipolar effects, careful body ties are used to maintain resistance against SEUs [50][54][55]. However, manufacturers have found even body-tied SOI substrates are not sufficiently resistant against SEUs for applications where very high upset limits are desired [52][56][57]. In some cases, fully depleted SOI transistors exhibit reduced floating-body effects.

Techniques at the fabrication process level provide a limited path to mitigate soft errors. Due to the invasive nature of these mitigation methods, which require fundamental changes in the manufacturing process, these low level methods usually come at the expense of additional process complexity and

steps, increased yield loss, or substrate cost [7]. Consequently, methods to increase soft error resistance at higher levels have been an alternative path to the fabrication process level techniques.

3.2 Circuit and System Level Techniques to Tackle Soft Errors

As opposed to mitigation methods at the fabrication process level, circuit and system level soft error mitigation techniques can provide portability across different fabrication processes. In addition to that, these higher level techniques could reduce the gap that exists between the state of the art fabrication technology and soft error sensitivity.

Before discussing mitigation techniques for various storage elements it is important to note that there should be proper choice of circuit types in a design to decrease its sensitivity to soft errors. Thus, elements that are known to be vulnerable to soft errors should be eliminated. For instance, compared with static CMOS circuitry, dynamic logic (clock signal is used to precharge the output in the precharge phase while the pull-down network can discharge the output in the evaluate phase [58]) due to its passive and highly charge-sensitive mode of operation is vulnerable to soft errors; therefore it should not be used [59].

DRAM cells used to be sensitive to soft errors when manufacturers used planar capacitor cells that stored the signal charge in 2D, large-area junctions, because these cells were very efficient at collecting radiation induced charge. This issue was later addressed by developing 3D capacitor designs that significantly increase Q_{crit} while greatly reducing junction collection efficiency by eliminating the large storage junction in silicon. Charge collection decreases by decreasing the junction's volume, whereas the cell capacitance remains relatively constant with scaling because it is dominated by the external 3D capacitor cell.

One mitigation technique for SEUs in an SRAM cell is to increase the gate capacitance or interconnect capacitance of its storage nodes, since Q_{crit} is proportional to C_{node} (node capacitance) and V_{node} (node voltage) as shown in Equation 6. As illustrated in Figure 9, parasitic capacitance between the interconnect metal layers (C_A and C_B) are used in [60] to add extra capacitance to the storage nodes to increase resistance against SEUs.

$$Q_{crit} \propto C_{node} V_{node} \tag{6}$$

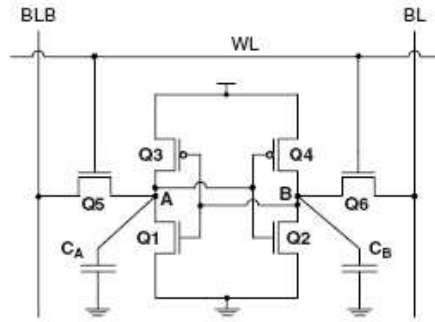


Figure 9 Capacitors C_A and C_B added to SRAM cell mitigate SEUs [60]

Researchers in [61] proposed an area efficient design shown in Figure 10 to add capacitance (C_C) in the SRAM cell. In this design, cross couple capacitance is fabricated by 2 local interconnects. This design provides 20% reduction in area compared to the conventional SRAM cell.

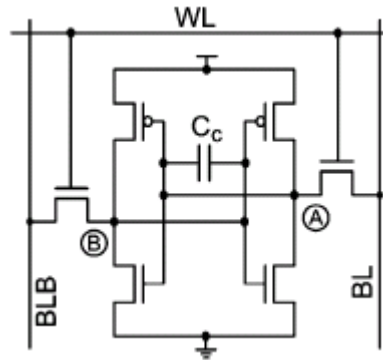


Figure 10 Capacitance C_C added in SRAM cell [61]

Vertical metal-insulator-metal capacitors are used by ST Microelectronics to add extra capacitance to storage nodes [62]. These capacitors are added in levels where there is no SRAM interconnect (the unused space above SRAM cells are used). Therefore, they do not change the SRAM cell and its area. However, there is an area penalty to route over SRAM.

Another method to mitigate SEUs is to insert resistors in the feedback loop of an SRAM cell. This increases the RC time constant of the cell. The increased RC time constant potentially allows the cell to recover from an SEU. Researchers in [63] add 2 extra resistors R_1 and R_2 , illustrated in Figure

11, to increase R_1C_1 and R_2C_2 time constants. As opposed to the mainstream six-transistor CMOS SRAM cell, shown in Figure 4, there are no direct connections between the input and output of the inverters shown in Figure 11. High-ohmic resistors R_1 and R_2 slow down the voltage transition on the input of inverters of the SRAM cell in Figure 11. For instance, if there is a current spike on node A , due to a high-energy particle hitting node A , it takes time for the disturbance to reach inverter $Q_1 - Q_3$. If this time is longer than the recovery time of node A the SRAM cell will not be flipped by the high-energy particle strike. In conclusion, if the recovery time of the output of the inverter is shorter than the RC time constant of the cell a bit flip does not happen.

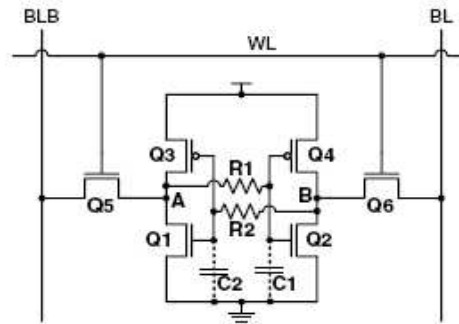


Figure 11 Increased RC time constant by adding R_1 and R_2 in feedback loop of SRAM cell [63]

Increasing the resistors in a feedback loop inevitably increases the write time of an SRAM cell. However, in some cases, the increased write time might not be significantly important and can be tolerated. For example, in FPGA SRAM cells are used mainly in the read mode and are usually written once during the FPGA configuration. In addition to the write time penalty, there is increased process complexity incurred by adding feedback resistors [11].

Another important method to mitigate SEUs is by using redundant transistors to build memory cells or latches. These designs are different from mainstream storage elements built without considering resistance against SEUs. Unlike the typical SRAM cell with 6 transistors, researchers in [64] proposed an SRAM cell, depicted in Figure 12, with 10 transistors. As opposed to the positive feedback of the mainstream SRAM cell with 6 transistors, the negative feedback of the design suggested in [64] prevents flipping of the SRAM cell when there is a glitch at a node due to a high-energy particle strike.

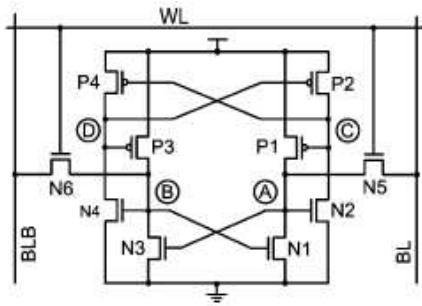


Figure 12 Soft error robust SRAM cell using 10 transistors [64]

One technique used at the circuit or system level to mitigate soft errors is based on time redundancy (also known as time multiplexing, temporal redundancy, and temporal parallelism) [4]. In this technique, data is sampled at different clock edges shifted relative to the global clock according to a clocking scheme to mitigate SETs. The effectiveness of the temporal parallelism scheme is based on the fact that the likelihood of 2 independent errors occurring in the same circuit path within a small period of time is extremely low. For instance, as is illustrated in Figure 13, time redundancy is used to detect errors due to glitches (SETs) in combinational logic or SEUs occurring in flip-flops [4].

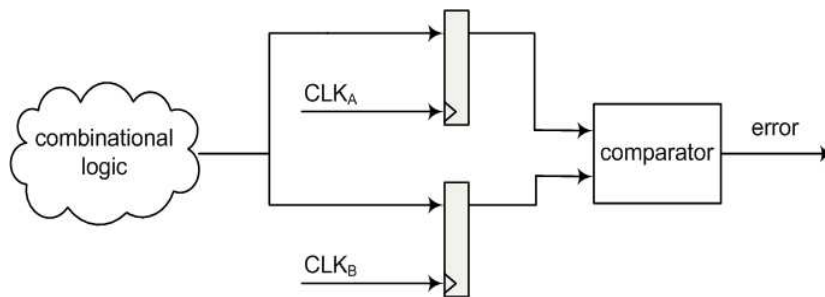


Figure 13 SET error detection using time redundancy [4]

Another method to tackle soft errors at the circuit or system level is based on the hardware modular redundancy (also known as spatial redundancy). In this method, 2 or more identical hardware modules are typically used to detect or further correct errors. Error detection by using Double Modular Redundancy (DMR) is shown in Figure 14. A mismatch in the output data detected in a DMR system will result in a restart of the system.

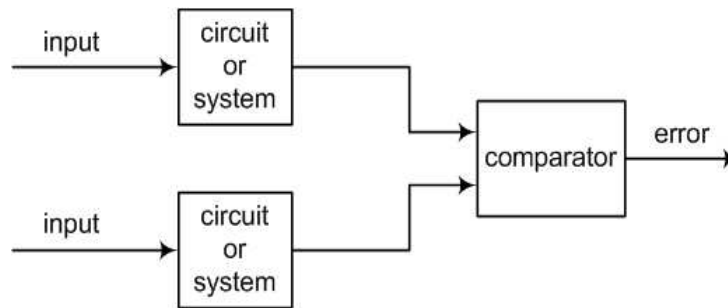


Figure 14 Error detection by DMR

As depicted in Figure 15, Triple Modular Redundancy (TMR) is used for single error correction at the circuit or system level. In TMR, 3 identical copies computing the same input are connected to a majority voter. A majority voter is used to identify which of the outputs provide the correct data. The error is ignored in favor of the majority that supplies the correct output. Therefore, the correct output appears as the final result of the computation.

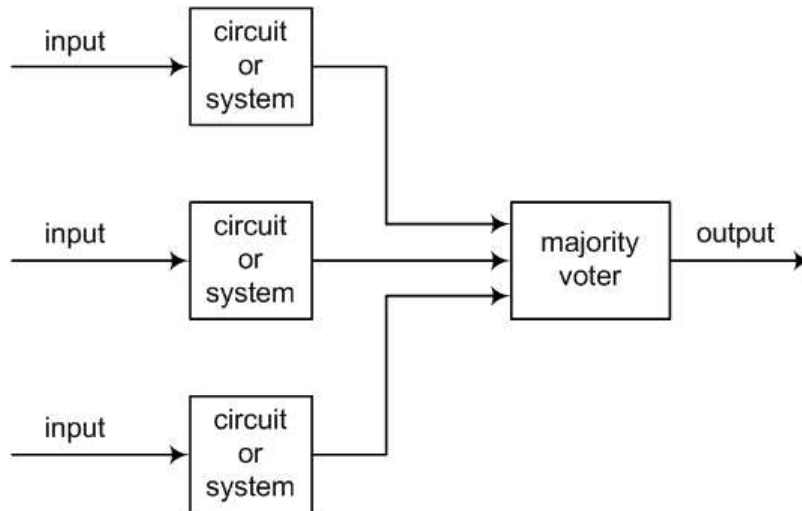


Figure 15 Error correction by TMR

In general, the downside to a circuit or system using hardware modular redundancy discussed above is the extra area, power, latency, and delay which is inherent to redundant schemes. For

instance, the TMR scheme shown in Figure 15 consumes approximately triple the chip area in addition to the majority voter logic.

As illustrated in Figure 16, the TMR method is used in register cells in LEON3-FT-RTAX (a fault tolerant FPGA-based microprocessor) to enhance resistance against soft errors [65]. If one of the latches is hit by a high-energy particle and starts to change state, the voter gate with the other 2 latches prevents the change from feeding back and permanently being latched. Layout of this circuit is done in such a way to ensure a single ion strike could not affect more than one latch, and thus causing multiple errors.

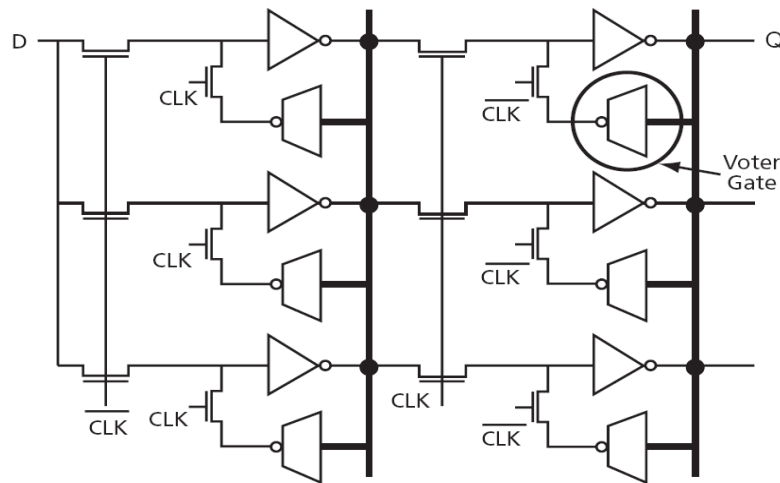


Figure 16 Register cell in LEON3-FT-RTAX using TMR [65]

Figure 17 shows a simplified generic view of the technique that uses both the spatial and temporal redundancy in flip-flops [66]. The spatial parallelism technique uses multiple memory cells to protect against SEUs. The spatial and temporal parallelism technique proposed in [66] mitigates SEUs and SETs in combinatorial logic, global clock, and global control lines. This method comes at the cost of both the clock frequency and area.

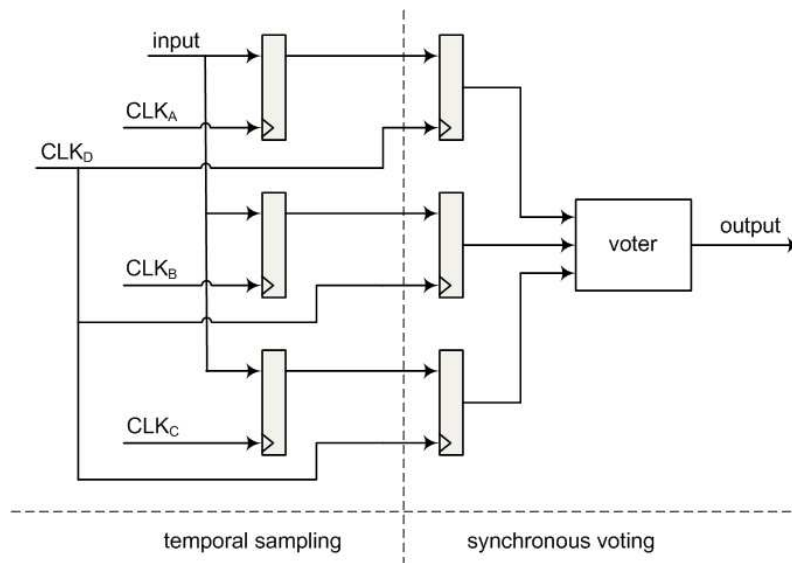


Figure 17 Spatial and temporal redundancy [66]

At the system level, one of the most effective methods in dealing with soft errors in memory components is to use error detection and correction codes. Error detection and correction codes add a certain degree of redundancy to the system, and therefore affect performance and occupy additional area. The choice of a detection or correction code is generally based on the nature of faults and required fault tolerance of the system. The choice of error detecting code in this project is discussed later in this section.

The parity code is the simplest error detection (but not an error correction) code. It adds an extra bit, the parity bit, to the data word (the actual information part in a word) so that the number of ‘1’s in the codeword (data word plus parity bit) becomes even in case of the even parity or odd in case of the odd parity. The obvious advantage of the parity code is its simplicity; and thus potentially minimal hardware overhead with just having 1 redundant parity bit in the whole codeword.

The parity code is effective in detecting an odd number of errors in a data word and the parity bit (codeword). However, the generated error vector does not locate which bit or bits have been corrupted. When the number of corrupted bits in a codeword is even, the parity bit is still valid and the parity bit is not able to flag the error. Therefore, cases where an even number of bits get corrupted in a codeword are not detected by the parity code.

As shown in Figure 18, the parity error detection is generally used to detect single errors in an output register. The parity bit for the output register is predicted in parallel with the computation of the output. Then the predicted parity is compared with the real output parity to detect any single error in the output register and set the error flag.

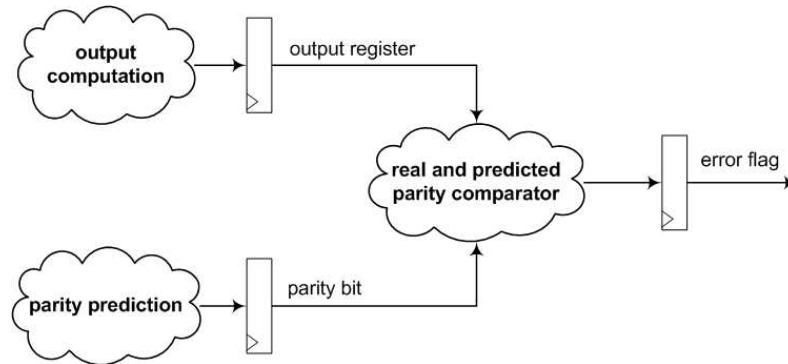


Figure 18 Parity error detection

As opposed to the parity code, the error correction codes (e.g., the Hamming code [67]) add additional redundant bits (check bits) that enable unique error syndromes to be generated. These unique error syndromes can locate the position of corrupted bits. The overhead of encoding to generate the check bits should be considered since this overhead can be high. The redundant hardware should be less than TMR to make the error detection and correction code a reasonable choice.

Error detection and correction codes provide an alternative to methods based on hardware modular redundancy since they (e.g., TMR and DMR) are usually expensive in terms of hardware cost and power consumption.

Depending on the nature of errors, the degree of robustness that these expensive hardware modular redundancy schemes provide can be higher than what is really needed. For instance, DMR provides error detection for multiple bits. In case of soft errors where an error is random in time and space, the likelihood of multiple errors in 1 clock cycle is exceedingly low. Therefore, in this scenario, a less expensive approach such as the parity error detection could suffice. It is important to investigate the capability of the parity scheme in detecting single errors on a specific platform to understand to what extent it is able to detect single errors.

3.3 Tackling Soft Errors by Architectural Methods

When there are storage elements in a system, soft errors can become an issue in reliability. In a microprocessor system, as shown in Figure 19, memory elements include registers known as register file, caches at different levels, main memory, and so on. Memories that are closer to a microprocessor are typically SRAM-based memories which are faster (have a shorter access time), smaller sized, and more expensive. The main memory is typically a DRAM-based memory that is less expensive with a longer access time.

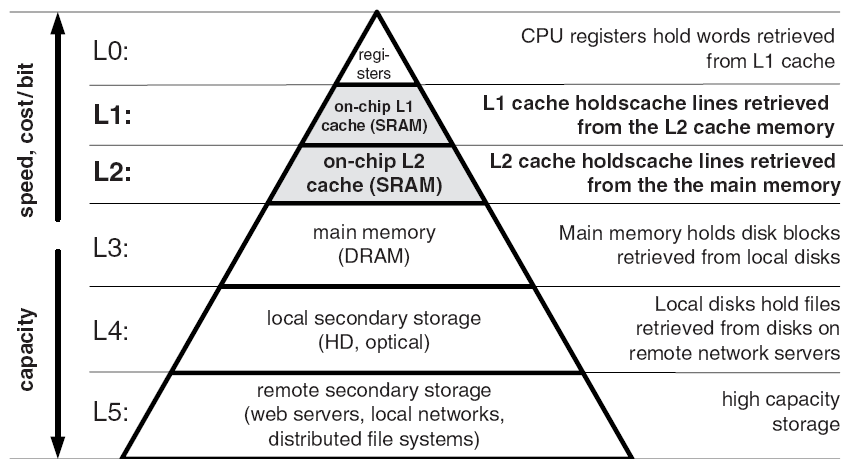


Figure 19 Memory hierarchy in a typical microprocessor system [13]

The connection amongst different levels of memory system in a microprocessor is shown in Figure 20. On a cache miss, data is typically provided by the next higher level (farther from the microprocessor) memory with longer access time in the hierarchy. An error in one memory level can propagate to other levels when a miss happens and corrupted data is copied to another level.

In order to mitigate SETs (glitches) propagating to registers, the time redundancy technique (refer to Section 3.2) to monitor combinational results by using a sequential design is proposed in [68]. This approach is then implemented in [69] to protect combinational logic of operations in the Arithmetic and Logic Unit (ALU). Researchers in [69] use the TMR technique to protect the register file against soft errors.

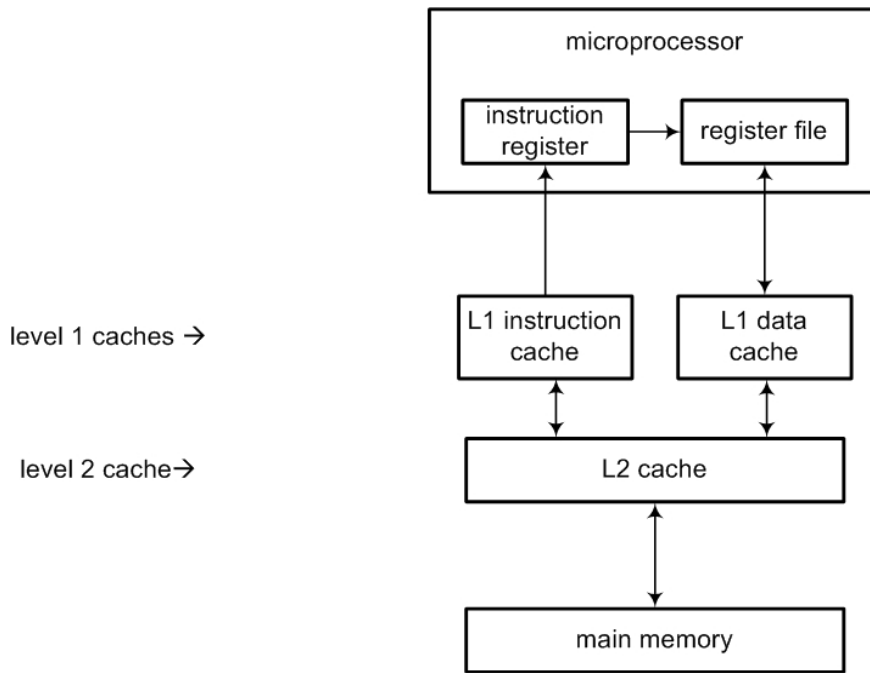


Figure 20 Memory elements in Software-based system

Cache lines that have not been used for a long time (dead cache lines) are used to hold replicas of lines that are used frequently (hot cache lines) in [70]. This approach can potentially increase the cache miss rate and decrease performance. It also depends on how many cache lines are used infrequently in a program so that they can be used for replicas.

Another approach that is proposed in [71] is software cache flushing. In this technique, the operating system flushes the entire cache periodically to remove errors. This affects performance due to the overhead of write-back and cache misses.

It is shown in [72] that vulnerability of a level 1 cache to SEUs is reduced by using a write-through policy rather than write-back policy. The reduction in vulnerability, in a write-through policy, is due to immediate updating of the next level cache. In a write-back, as opposed to a write-through policy, a cache line is not updated in the next level or main memory until the line needs to be replaced. Therefore, a cache line that resides longer in the cache is more likely to be affected by an SEU and propagate the error to the next level cache or main memory. Researchers in [73] proposed a technique that can be applied to a write-through cache to remove errors in a cache refetching. This technique

refetches cache lines from the next level cache or main memory to refresh cache lines. This basically reduces the residency of cache lines to increase the cache reliability.

LEON3-FT-RTAX is a fault tolerant FPGA-based microprocessor [74]. Figure 21 depicts the hardware components of this microprocessor. The combinational logic of LEON3-FT-RTAX is implemented in an antifuse FPGA which is RTAX-2000S. Unlike SRAM based FPGAs with SRAM cells controlling routing, this type of non-volatile FPGA has metal-to-metal antifuse programmable interconnect elements. Antifuses are normally open circuit and are programmed form a permanent, passive, and low impedance connection.

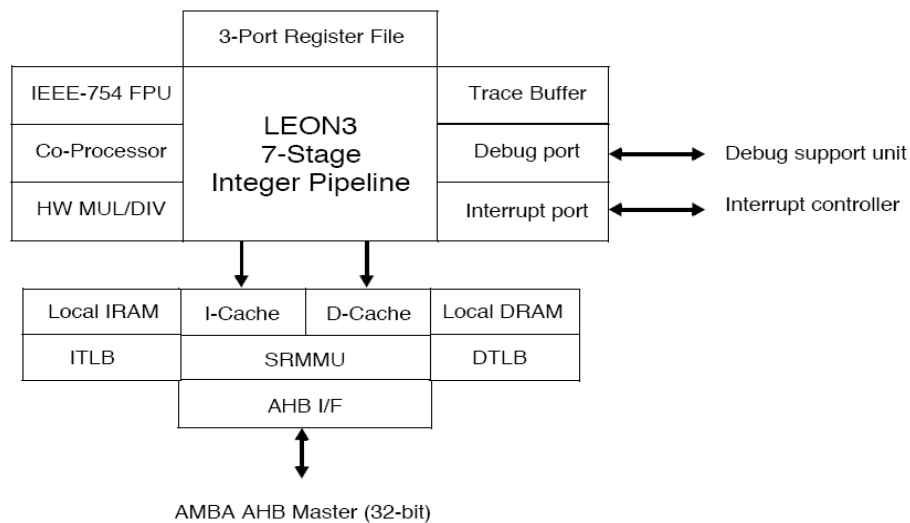


Figure 21 Hardware components in LEON3 microprocessor [74]

As opposed to SRAM cells, the downside of antifuse elements is that they are one-time programmable. On the other hand, their advantage is that they are not vulnerable to SEUs. Flip-Flops in this microprocessor are hardened (made more tolerant against errors) by using TMR on the FPGA (RTAX-2000S). Each TMR D flip-flop (shown in Figure 16) consists of 3 master-slave latch pairs, each with asynchronous self-correcting feedback paths.

There are different options, shown in Table 3.1, for error detection and correction of the register file in LEON3-FT-RTAX to mitigate soft errors. The desired option can be selected during synthesis. As discussed before in Section 3.2, these techniques add overhead to the system. For instance, the

correction in the 3rd and 5th rows of Table 3.1 takes 6 clock cycles. An uncorrectable error in the register file causes a trap.

Table 3.1 Error detection and correction in register file [74]

Error detection and correction technique	Description
Hardened flip-flops or TMR	Register file implemented with SEU hardened flip-flops. No error checking.
4-bit parity with restart	4-bit checksum per 32-bit word. Detects and corrects 1 bit per byte (4 bits per word) through restart.
8-bit parity without restart	8-bit checksum per 32-bit word. Detects and corrects 1 bit per byte (4 bits per word). Correction on-the-fly without pipeline restart.
7-bit BCH with restart	7-bit BCH checksum per 32-bit word. Detects 2 bits and corrects 1 bit per word. Pipeline restart on correction.

Each word in the cache tag or data memories has 4 check bits. An error during a cache access will cause a cache line flush, and a re-execution of the failing instruction. This will insure that the entire cache line (tag plus data) is refilled from external memory.

3.4 Soft Errors in FPGA vs. ASIC

This section presents the peculiar effects of soft errors in FPGAs and how these effects are different in FPGA vs. ASIC.

The effect of SETs are similar in an ASIC or FPGA. An SET (equivalent to a glitch) might propagate through the combinational logic up to a flip-flop. Depending on the timing of the glitch relative to the clock edge it might get stored and replace the valid data, or it might not have any effect at all. An error caused by an SET is not permanent in the sense that the implementation itself is not affected in terms of functionality and resetting the system will bring it back to its expected initial state.

An SEU occurring in sequential elements or flip-flops affects both FPGA and ASIC in a similar way. Due to an SEU the value of a flip-flop flips. Then at the next clock edge the new data is stored. Therefore, the affect is not permanent and the implementation is functionally correct after a reset.

Unlike ASICs, an SEU has a peculiar effect in FPGAs when it hits the combinational logic. Since the combinational logic in an ASIC does not have any storage elements SEUs can only affect flip-flops. On the other hand, combinational logic in an FPGA is thoroughly controlled by SRAM cells (e.g., SRAM cells control LUTs in the combinational logic). The memory cells of a LUT (combinational logic) can be affected by an SEU in the same way that flip-flops (sequential logic) are affected. For instance, in Figure 22(a), an SEU can change the function that is stored in SRAM memory cells of the LUT. This error in an SRAM cell is not transient, meaning that it will not resolve at the next clock edge. It will only resolve when the FPGA is reconfigured. Consequently, this changes the functionality of the function implemented in the LUT. As observed, the whole combinational logic of an implementation on FPGA is vulnerable to SEUs that can cause non-transient errors.

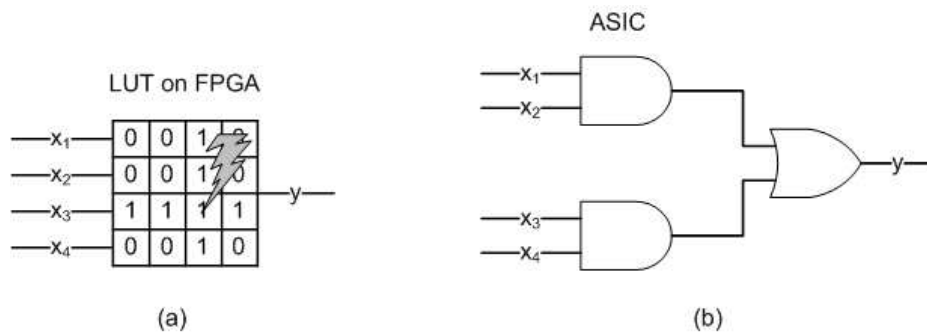


Figure 22 Combinational logic on FPGA vs. ASIC (a) function implemented in LUT on FPGA, (b) function implemented using gates on ASIC

It is important to note that the routing in an FPGA is also controlled by SRAM cells. These SRAM cells control pass transistors, multiplexers and tri-state buffers. Therefore, in addition to combinational logic, routing is vulnerable to SEUs. Thus, it is crucial to investigate the effectiveness of an error detecting scheme on FPGA with respect to this peculiar effect of SEUs in combinational logic and routing on FPGA.

In this research, the parity error detection on FPGA is thoroughly investigated and its weaknesses are found. Other than fabrication process level methods to tackle soft errors, some of the mitigation techniques such as a parity or Hamming code are used in FPGAs just as in ASICs. These techniques

do not cover errors in SRAM cells controlling logic blocks and routing of FPGAs. Techniques to eliminate these weaknesses are proposed (refer to Chapter 5). Therefore, the error coverage of the parity scheme is expanded to soft errors in combinational logic as well as routing on FPGA.

It is important to note that concurrent error correction techniques, e.g., the Hamming code in [75] and TMR can potentially lose their correction capability over time in an FPGA implementation. The reason behind this is the potential change in the functionality of the correction technique implemented. For example, SEUs can hit the SRAM bits defining the voter functionality of a TMR based design or the error correction circuitry of the Hamming code. In addition to that, there is accumulation of errors due to SEUs. The only way to remove these errors in SRAM cells of combinational logic and routing is to reconfigure the FPGA. If these errors are not removed by reconfiguration the dependability of a correction technique that a system has relied on is questionable. The necessity of reconfiguration to remove SEUs in SRAM-based FPGAs was mentioned in [4].

3.5 Summary

The main goal of this chapter was to present previous techniques to tackle soft errors at different levels. At the lowest level, fabrication process-based techniques have been used. These low level techniques can affect the fabrication process complexity and steps, increase yield loss, or substrate cost [7]. Additionally, they may not be portable across different fabrication processes. Soft error mitigation techniques at higher levels (circuit and system levels) provide portability and can eliminate the gap between the state of the art fabrication technology and soft error sensitivity. At the circuit and fabrication process-based levels, the main goal is to increase Q_{crit} while maintaining or reducing Q_{coll} to improve resistance against soft errors.

At the system level, spatial or temporal redundancies are used to tackle soft errors. Since the likelihood of 2 independent errors in the same circuit path within a small period of time is extremely low, data is sampled at different clock edges to detect SETs in temporal redundancy. Spatial redundancy is used in error detection and correction codes or in hardware modular redundancy techniques such as DMR and TMR to detect and correct errors, respectively.

Peculiar effects of SEUs in FPGA in the combinational logic and routing were discussed in this chapter. These effects do not happen in ASIC. Since the combinational logic and routing on ASIC do not have any storage elements, SEUs can only affect flip-flops. On the other hand, the combinational

logic and routing on FPGA is thoroughly controlled by SRAM cells (e.g., SRAM cells control LUTs in the combinational logic and pass transistors in routing).

In summary, previous research using the parity code [76][77][78] or Hamming code [75] for AES have not analyzed the underlying routing or logic of an SRAM-based FPGA implementation. Selecting a system level mitigation technique according to available resources and mathematical properties of a specific operation is also not considered in classic techniques such as TMR and DMR. Radiation hardened FPGAs typically use fabrication, circuit, or system level mitigation techniques (for instance, Virtex-4QV uses a thin epitaxial layer in wafer manufacturing [49] and flip-flops are designed using TMR in LEON3-FT-RTAX [74]) that are independent of the application implementation and possibly introduce redundancy in unused resources.

In the next chapter, security requirements of a system are briefly described. Then the primitives to provide these security needs are presented. Emphasis is put on symmetric-key algorithm AES that is the focus in this research. NIST-recommended block cipher modes are also covered briefly. Error propagation in the AES algorithm and modes are discussed at the end of the chapter.

Chapter 4

Security Needs of Data Systems

This chapter briefly describes the general security requirements of a system such as authentication, data confidentiality, data integrity, and non-repudiation. Then it describes the primitives for providing these security needs. Emphasis is placed on standard symmetric-key algorithms, such as AES described in section 3.1.1. Block cipher modes (suggested by NIST) which can be used with AES are also covered briefly. The AES error propagation in these modes is discussed. Previous approaches in implementing AES are provided at the end of this chapter.

4.1 Security Needs and Cryptographic Algorithms

In general, the security services to be provided for a system include authentication, access control, data confidentiality, data integrity, and non-repudiation [79]. The following presents a brief definition of each security service [80].

- Authentication is the assurance that the identities in a communication are the ones they claim to be.
- Data confidentiality is the protection of data from being disclosed by unauthorized parties. The protection could even include any information about the data traffic flow.
- Data integrity is the assurance that the received data has not been replayed or affected by modification, insertion, or deletion.

- Non-repudiation prevents against denial by an authorized party involved of having participated in a communication.

The basic types of cryptographic algorithms (primitives) that provide a means for the above security services are symmetric-key algorithms, public-key algorithms, and hash functions. These primitives are introduced briefly as follows [81][82].

- In the classical model of cryptography, a symmetric-key (also known as conventional or single-key) cryptographic algorithm encrypts plaintext into ciphertext using a secret key. The decryption algorithm uses the same secret key to transform the ciphertext to plaintext. Symmetric-key encryptions are typically used for confidentiality. They are also used in common keyed hash functions to provide data integrity and authentication.
- As opposed to a symmetric-key algorithm, a public-key algorithm uses 2 different keys (public and private keys) for encryption and decryption. For example, a public key is used to encrypt plaintext to produce ciphertext. The corresponding private key is then used to decrypt the ciphertext and recover the plaintext. It should be computationally infeasible to find the decryption rule from the encryption rule in a public-key cryptosystem. Public-key algorithms are typically used in security protocols for authentication. They are also used for generating and verifying digital signatures to provide non-repudiation and for exchanging keys in a symmetric-key cryptosystem.
- A hash function is a transformation that takes a variable-sized input data (message) and returns a fixed-size output (message digest also known as hash value). It should be relatively easy to compute the hash value for any message. On the other hand, it should be computationally infeasible to compute the message from the hash value. The problem of finding 2 messages having the same hash value should also be difficult to solve. Unkeyed hash functions provide data integrity, while keyed hash functions are used for data integrity and authentication. Another common use of hash functions is in signature schemes. In this case, the hash value of a message is computed first, and then the hash value is signed using a signature scheme.

In a symmetric-key cryptosystem, it is required that the 2 parties have already established a shared secret key between themselves in a secure manner before any ciphertext is transmitted. In a public-key cryptosystem the prior communication of a shared secret key is not needed. However, it should

be computationally infeasible to determine the private key given the public key. The idea behind public-key cryptography was introduced by Diffie and Hellman in 1976.

Most public-key algorithms have larger key sizes (denoted k) than symmetric algorithms; Table 4.1 shows a comparison of security strengths and lifetimes among RSA (a public-key algorithm), AES, 2TDEA (triple Data Encryption Standard (DES) symmetric-key algorithm with 2 independent keys), and 3TDEA (triple DES with 3 independent and different keys). Security strength, which is specified in bits, is a number associated with the amount of work (that is the number of operations) that is required to break a cryptographic algorithm or system [83]. It should be noted that the number of bits of security strength is not necessarily the same as the key sizes for the algorithms, due to attacks on algorithms that provide an attacker with computational advantages. In general, symmetric algorithms are faster and more efficient with respect to implementation and performance. However, they need secure establishment of a secret key; public-key algorithms can be used to provide a cryptosystem with the secure key establishment.

Table 4.1 Lifetime and security strength of symmetric and public-key algorithm [83]

Security lifetime	Security strength in bits	RSA (public-key algorithm)	Symmetric-key algorithm
Through 2010	80	$k=1024$	2TDEA
Through 2030	112	$k=2048$	3TDEA
Through 2030	128	$k=3072$	AES, $k=128$
Through 2030	192	$k=7680$	AES, $k=192$
Through 2030	256	$k=15360$	AES, $k=256$

A symmetric-key block cipher is a very important primitive in encryption/decryption, key transport for establishing session keys and keyed hash functions. The NIST standardized symmetric-key block cipher AES, which has been predicted to be secure well beyond 20-30 years, are discussed in the next section. Block cipher modes as well as how they provide confidentiality and integrity services and the error propagation issue related to the modes are also discussed in Section 4.2.

4.1.1 Advanced Encryption Standard

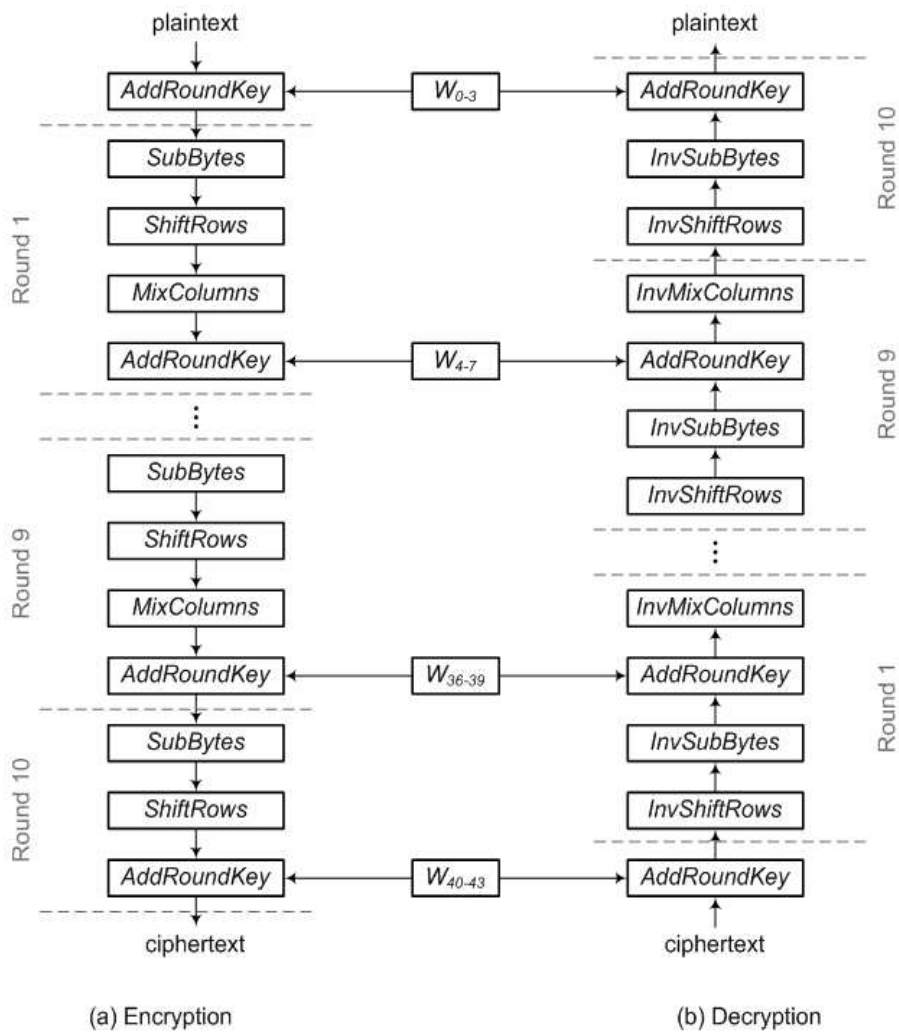
With technology rapidly advancing, DES, which uses a 56-bit key, was broken in 1999 in less than 22 hours [84] by brute force attack. Therefore, a robust new standard was needed to replace the aging DES which had been developed in the 1970s. In September 1997, NIST launched a worldwide call for submission of publicly disclosed encryption algorithms worldwide for the AES selection [85]; 15 candidate algorithms were submitted. After 3 publicly held conferences to discuss and analyze the candidates, the competition field narrowed down to five competitors. Finally, in 2000, Rijndael was named as the AES algorithm, the winner of the 3-year competition involving some of the world's leading cryptographers. In March 2008, due to the comprehensive investigation on AES and focus on its usage, Consultative Committee for Space Data Systems (CCSDS) proposed to adopt AES as the standard encryption algorithm for space application [86].

AES is based on the ideas of Shannon and the concepts of diffusion and confusion. Whilst most block ciphers follow these principles, few do so as clearly as AES that is said to be a substitution permutation (SP)-network [81]. Diffusion is intended to spread out the influence of all the bits of inputs, namely plaintext and key, to all the bits in ciphertext. Diffusion is provided in AES by the use of *ShiftRows* and *MixColumns* [87]. The goal of confusion is to make the relationship between ciphertext and a key and plaintext as complex as possible. In AES, confusion is provided by a very carefully chosen substitution transformation (referred to as *SubBytes*). *SubBytes* is the most complex and the only nonlinear operation of AES due to the multiplicative inversion it contains [87]; nonlinearity ensures a low correlation between input bits and output bits. The *SubBytes* transformation is designed in such a way that makes it resistant against linear and differential attacks, as well as interpolation attacks [88]. The *SubBytes* output is then spread by diffusion in each round.

AES supports 3 key sizes (i.e. 128, 192, and 256 bits) and has the fixed block size of 128 bits. The algorithm consists of a number of rounds depending on the key size as shown in Table 4.2. The encryption and decryption algorithm for the key size of 128 bits is depicted in Figure 23. Each round consists of four transformations, which are described below, except for the last round which includes only 3 operations. The AES key expansion algorithm takes the input key and generates 128-bit round keys for each round. A 128-bit data block is handled in a 4x4 matrix in which each element of the matrix is an 8-bit element; this matrix is referred to as the state.

Table 4.2 AES parameters

Key size in bits	Block size in bits	Number of rounds
128	128	10
192	128	12
256	128	14



$W_{k-(k+3)}$: Representing the expanded key of each round

Figure 23 AES algorithm

A Galois Field (GF) is a field that contains a finite number of elements and the order of the field is p^m where p is a prime number and $m \geq 1$ is an integer [82]. This field is denoted $GF(p^m)$. A commonly used representation for the elements in $GF(p^m)$ is the polynomial based representation which is shown as follows in Equation (7).

$$GF(p^m) = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_1x + a_0 \mid a_i \in \{0,1,\dots,p-1\}\} \quad (7)$$

Addition and multiplication of polynomials is performed modulo $m(x)$ where $m(x)$ is an irreducible polynomial of degree m . An 8-bit element, b , of a state in AES is an elements in $GF(2^8)$; b can be represented as follows in the polynomial form.

$$b = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0; \quad \text{where } b_i \in \{0,1\} \quad (8)$$

- The *SubBytes* transformation, as shown in Equation (9), computes the affine transformation on multiplicative inverse b^{-1} of an 8-bit input b in $GF(2^8)$ with the corresponding irreducible polynomial being $m(x) = x^8 + x^4 + x^3 + x + 1$. Equation (10) shows *SubBytes* where the affine transformation is done on the 8-bit multiplicative inverse element, $b^{-1} = [b_7^{-1} \ b_6^{-1} \ b_5^{-1} \ b_4^{-1} \ b_3^{-1} \ b_2^{-1} \ b_1^{-1} \ b_0^{-1}]^T$ in the matrix form. *SubBytes* is the only nonlinear transformation in AES due to the multiplicative inversion it contains. The 8-bit elements b and b^{-1} have an inverse relationship in which $bb^{-1} = 1 \pmod{m(x)}$. The affine transformation is multiplication by a constant matrix, M , and an addition with a constant $C = (63)_{16}$ as shown in Equation (10). Note that a single-bit multiplication is an AND, and a single-bit addition is an XOR in the Galois field.

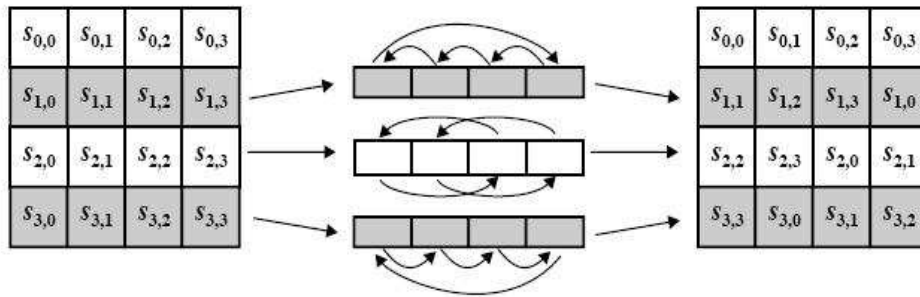
$$SubBytes(b) = affine(b^{-1}); \quad \text{where } b^{-1} = inverse(b) \quad (9)$$

$$\begin{aligned}
 \text{affine}(b^{-1}) &= Mb^{-1} + C \\
 &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_7^{-1} \\ b_6^{-1} \\ b_5^{-1} \\ b_4^{-1} \\ b_3^{-1} \\ b_2^{-1} \\ b_1^{-1} \\ b_0^{-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{10}
 \end{aligned}$$

- The mix columns transformation (referred to as *MixColumns*) is a function on each column of the state that outputs a corresponding column (four 8-bit elements). Given a column of the state $[a \ b \ c \ d]^T$ in which elements are in $GF(2^8)$ where $m(x) = x^8 + x^4 + x^3 + x + 1$, the *MixColumns* transformation is multiplication by polynomial $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ modulo $x^4 + 1$, as it is shown in Equation (11). Equivalently, this can be done by a matrix multiplication as shown in Equation 12.

$$d'x^3 + c'x^2 + b'x + a' = (dx^3 + cx^2 + bx + a)(\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}) \pmod{(x^4 + 1)} \tag{11}$$

$$\begin{aligned}
 \begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} &= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \\
 &= \begin{bmatrix} 2a + 3b + c + d \\ a + 2b + 3c + d \\ a + b + 2c + 3d \\ 3a + b + c + 2d \end{bmatrix} \tag{12}
 \end{aligned}$$

Figure 24 *ShiftRows* transformation in AES [80]

- The shift rows transformation (referred to as *ShiftRows*) is an 8-bit circular left shift operation, as shown in Figure 24. The shift is done on each row of the state except for the first row.
- In the add round key transformation (referred to as *AddRoundKey*), a round key is added to the state by a simple bitwise XOR operation. Each round key produced by the key expansion has the same structure as the state so each 8-bit element of the round key is XORed with the corresponding element of the state.

4.2 Block Cipher Modes

A block cipher mode of operation, or mode for short, is a technique for adapting a symmetric-key block cipher algorithm for an application and a message length to provide security services required. For instance, a mode can feature the use of a symmetric-key block cipher algorithm to provide a security service, such as confidentiality or authentication.

NIST has approved 8 modes for block ciphers in a series of special publications [89]. Currently, there are 5 confidentiality modes (electronic codebook, cipher block chaining, output feedback, cipher feedback, and counter modes), 1 authentication mode (cipher-based message authentication code mode), and 2 combined modes for confidentiality and authentication (counter with cipher block chaining-message authentication code and Galois/counter modes).

This section describes some of the modes briefly in order to point out how the structure of different modes affects their sensitivity to errors. In the figures that follow, plaintext blocks, ciphertext blocks

and a symmetric key are denoted as $\{P_1, P_2, \dots, P_N\}$, $\{C_1, C_2, \dots, C_N\}$ and K , respectively. Plaintext must be a sequence of one or more complete data blocks. In other words, the total number of bits in the plaintext must be a multiple of the block size. If the data string to be encrypted does not initially satisfy this property, then the formatting of the plaintext must entail an increase in the number of bits. A common way to achieve the necessary increase in length is to append some extra bits, called padding. One example of a padding method is to append a single bit '1' to the data string and then append as few '0' bits necessary, possibly none, to complete the final block.

4.2.1 Confidentiality Modes

Three of the confidentiality modes are outlined in this section. The error propagation issue with respect to the structure of the modes is also discussed.

The electronic codebook (referred to as ECB) mode is a confidentiality mode that has the simplest structure, as depicted in Figure 25. In the ECB encryption, the cipher encryption is applied directly and independently to each block of the plaintext. The resulting sequence of output is the ciphertext blocks. In the decryption, the decryption function is applied directly and independently to each block of the ciphertext. The resulting sequence of output decrypted is the plaintext blocks.

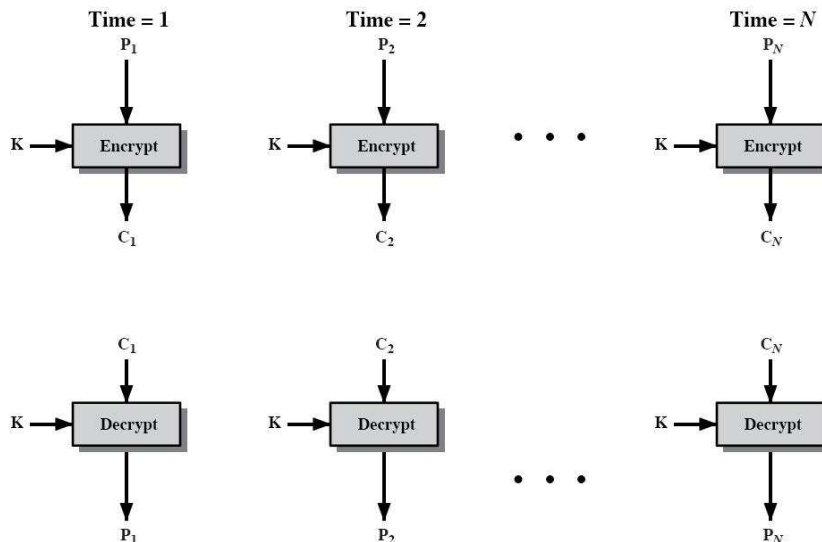


Figure 25 Electronic codebook (ECB) mode [80]

In both ECB encryption and decryption, multiple cipher encryptions and decryptions can be computed in parallel. In this mode, under a given key, the same plaintext block always gets encrypted to the same ciphertext block. This property called pattern identification might be undesirable in some particular applications. For instance, when a message is highly structured it may be possible for an attacker to exploit these regularities. To overcome this issue, other modes propose the chaining of encryption/decryption blocks in which there is a connection from the output of one encryption/decryption to the input of the subsequent encryption/decryption.

The encryption in cipher block chaining (referred to as CBC) mode features the chaining of the plaintext blocks with the previous ciphertext block as illustrated in Figure 26. It requires an Initialization Vector (IV) to XOR with the first input plaintext block. The IV, which is an additional input block, does not have to be secret but it must be unpredictable. As opposed to ECB, the same plaintext/ciphertext block if repeated generates a different ciphertext/plaintext block in CBC. This is due to fact that in CBC the ciphertext/plaintext block depends on not only the plaintext/ciphertext block but also the results of the previous encryptions/decryptions. However, the drawback is that the encryption/decryption cannot be performed in parallel.

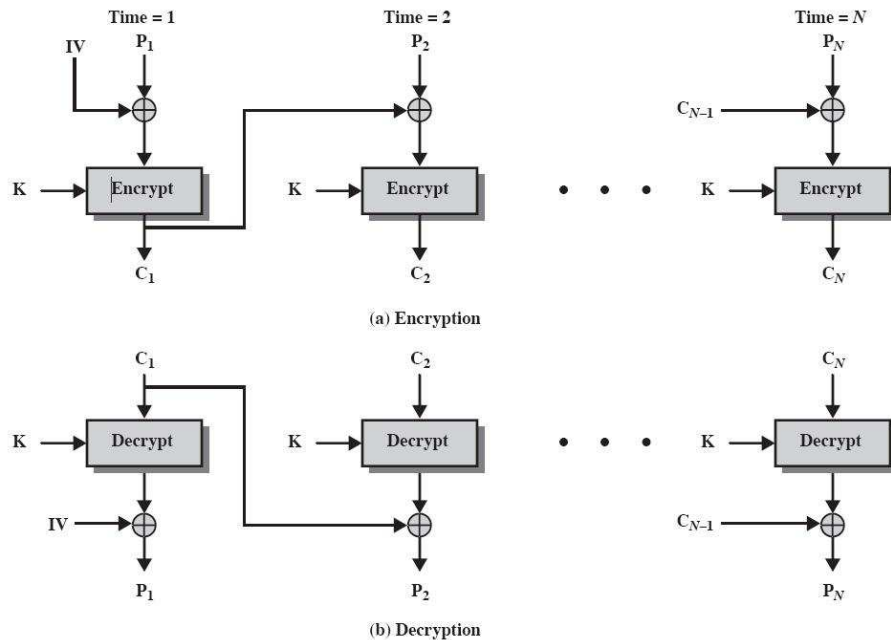


Figure 26 Cipher block chaining (CBC) mode [80]

The Counter (referred to as CTR) mode provides the capability of parallel performance. Figure 27 depicts the CTR mode. The counter value, which is the same size as the plaintext block, should be different for each plaintext that is encrypted. Therefore, repetitive ciphertext blocks corresponding to the same plaintext block cannot be recognized. The CTR mode requires only the implementation of the encryption and not the decryption.

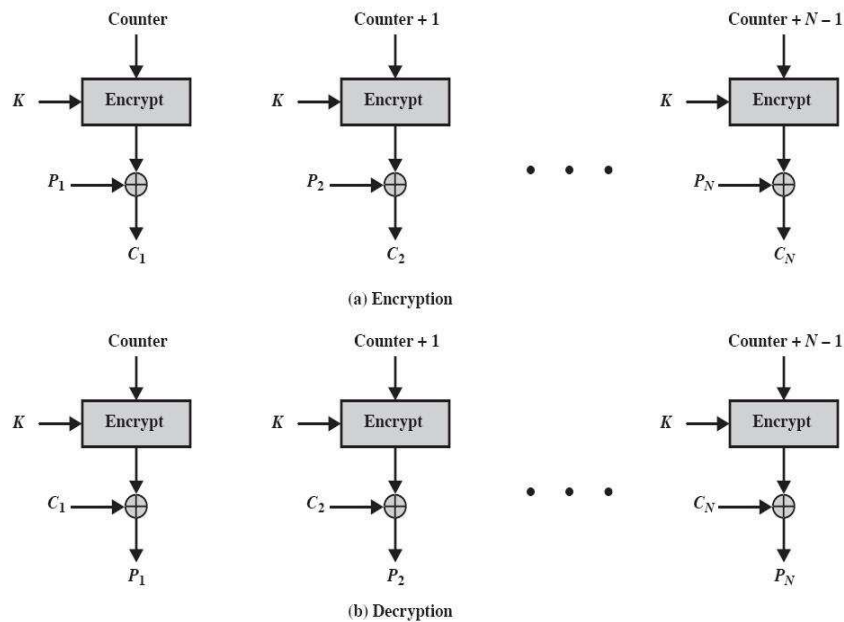


Figure 27 Counter (CTR) mode [80]

In general, modes that have the chaining of an encryption/decryption to the preceding encryption/decryption have the issue of propagating errors through the ciphertext/plaintext blocks (this is known as infinite error propagation). The modes that have this chaining structure are CBC, Output Feedback (OFB [89]) and Cipher Feedback (CFB [89]). On the other hand, ECB and CTR, which do not have this chaining structure, isolate errors within the corresponding block.

A single bit flip in the early rounds of AES encryption is expected to result in 50% erroneous bits in the output [76]. This shows a good diffusion in the AES algorithm. Diffusion is a desirable property from a cryptographic point of view and makes a strong symmetric-key algorithm; however it becomes an issue in error propagation. This problem is even worse in CBC, OFB and CFB modes

since a single bit error leads up to erroneous subsequent ciphertext/plaintext blocks which is dramatically different from the expected result.

4.2.2 Authentication Mode

NIST has recommended a cipher-based message authentication code (referred to as CMAC) algorithm that is based on a symmetric-key algorithm such as AES. CMAC is designed to detect intentional, unauthorized modifications of the data as well as accidental modifications. Figure 28 depicts the MAC generation in CMAC. The message is divided into a sequence of bit strings $\{M_1, M_2, \dots, M_n^*\}$, in which they all have the block size of 128 for AES except for the very last string M_n^* that might have a smaller size. The 2 keys $\{K_1, K_2\}$ are generated by the subkey generation (provided in [90]) by using the symmetric-key algorithm. CMAC has 2 parts illustrated in Figure 28. The left side that uses K_1 is applied when M_n^* has the size of a block. Otherwise, the right side with K_2 is used, and a single '1' bit followed by the appropriate number of '0' bits are appended to M_n^* to form a complete block. The same procedure is done at the destination to compute the MAC (T) which then compares it with the received MAC. Since an encryption is connected to the following encryption an error in any block propagates through the CMAC computation, finally it will affect the MAC (T). This shows the importance of detecting errors in all the encryption modules.

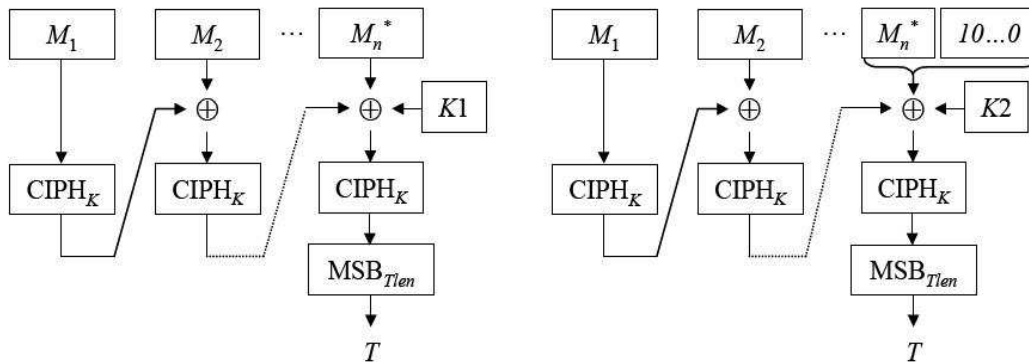


Figure 28 Cipher-based message authentication code (CMAC) [90]

4.2.3 Authentication and Confidentiality Modes

The 2 NIST-specified modes which provide both authentication and confidentiality are compared below in terms of their error propagation properties. The Galois/counter mode (referred to as GCM) provides authenticated encryption and authenticated decryption. It does not use the decryption module of the block cipher (e.g., AES). It encrypts the confidential data and computes an authentication tag on both confidential and non-confidential data. The confidentiality mechanism of GCM is a variation of the CTR mode. A particular incrementing function is specified [91] for generating the counter blocks in GCM. It also uses the block cipher (e.g., AES) for generating the authentication tag. Similar to the CTR mode, there is no connection between the ciphers; therefore the error is contained in its block in GCM. The authentication mechanism of GCM is provided by a hash function named GHASH which is multiplication by a fixed hash subkey [91].

The counter with cipher block chaining-message authentication code mode (referred to as CCM) provides authenticity of the confidential and non-confidential data and generates ciphertext for the confidential data [92]. It uses only the encryption of the symmetric-key algorithm (e.g., AES). The authentication mechanism of CCM uses CBC which has the chaining structure in the block ciphers. As a result, an error spreads throughout the blocks. On the other hand, the confidentiality mechanism in CCM uses the CTR mode that isolates an error within its block.

4.3 Previous Research on AES Design

AES is a complex and computationally intensive algorithm. Therefore, it needs significant amount of hardware resources in implementation. Block memories and combinational logic have been used for the AES implementation. The amount of block memories vs. combinational logic varies significantly in different approaches in previous research. One approach that uses the largest memory space amongst previous research combines the *SubBytes* and *MixColumns* transformation in a block memory named *T*-table [93][80].

A *T*-table is constructed as follows. In Equation 13, a' , b' , c' , and d' are 8-bit elements of the state column in the *MixColumns* result, while a , b , c , and d are 8-bit input elements to *SubBytes*. For simplicity, *ShiftRows* is not shown in Equation 13 since it does not need any logic resources. *T*-tables $T_0(a)$, $T_1(b)$, $T_2(c)$, and $T_3(d)$ are defined in Equation 14 for each 8-bit element.

$$\begin{aligned}
 \begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} &= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \text{SubBytes}(a) \\ \text{SubBytes}(b) \\ \text{SubBytes}(c) \\ \text{SubBytes}(d) \end{bmatrix} \\
 &= \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \text{SubBytes}(a) + \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \text{SubBytes}(b) + \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \text{SubBytes}(c) + \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \text{SubBytes}(d)
 \end{aligned} \tag{13}$$

Each T -table in Equation 14 is a block memory of size 256x32-bit. In order to provide parallel memory accesses, 16 T -tables are needed. It should be noted that the last round of AES does not include the *MixColumns* transformation. Therefore, *SubBytes* must be obtained from different tables in the last round.

$$\begin{aligned}
 T_0(a) &= \begin{bmatrix} 02 \times \text{SubBytes}(a) \\ 01 \times \text{SubBytes}(a) \\ 01 \times \text{SubBytes}(a) \\ 03 \times \text{SubBytes}(a) \end{bmatrix}, & T_1(b) &= \begin{bmatrix} 03 \times \text{SubBytes}(b) \\ 02 \times \text{SubBytes}(b) \\ 01 \times \text{SubBytes}(b) \\ 01 \times \text{SubBytes}(b) \end{bmatrix} \\
 T_2(c) &= \begin{bmatrix} 01 \times \text{SubBytes}(c) \\ 03 \times \text{SubBytes}(c) \\ 02 \times \text{SubBytes}(c) \\ 01 \times \text{SubBytes}(c) \end{bmatrix}, & T_3(d) &= \begin{bmatrix} 01 \times \text{SubBytes}(d) \\ 01 \times \text{SubBytes}(d) \\ 03 \times \text{SubBytes}(d) \\ 02 \times \text{SubBytes}(d) \end{bmatrix}
 \end{aligned} \tag{14}$$

In approaches other than T -table, *SubBytes* and *MixColumns* are designed separately. The following 2 sections present previous implementations of *SubBytes* and *MixColumns* of AES.

4.3.1 *SubBytes* implementations in AES

The memory based *SubBytes* implementations [94-97] basically use a block memory to store the results of this transformation. The *SubBytes* in the form of a block memory, known as an S-box, is given in Appendix F. Each 8-bit element of the state needs a 256x8-bit block memory for S-box. The size of an S-box is 25% of the *T*-table size (a *T*-table needs a block memory of size 256x32-bit). In addition to providing simplicity, the fact that state of the art FPGAs provide built-in block memories makes the memory based implementation of *SubBytes* an attractive option for this transformation. However, it might not be suitable for a heavily pipelined AES aiming to achieve the highest clock frequency and throughput.

On the other hand, the *SubBytes* implemented thoroughly in combinational logic [98-104] can be heavily pipelined. However, implementation of *SubBytes* in the original Galois field, $GF(2^8)$ with polynomial $P(x) = x^8 + x^4 + x^3 + x + 1$, is complex and uses significant number of hardware resources. Composite fields, briefly described as follows, have been suggested in previous research to reduce the complexity of operations in $GF(2^8)$. The complexity indicates the cost of hardware resources used for implementation of operations in a Galois field.

The complexity of various operations, such as multiplication and inversion depend on the chosen Galois field. Composite fields, first introduced in [105], were extensively studied in [106] to reduce the complexity of operations such as inversion in a Galois field. A composite field $GF((2^n)^m)$ is isomorphic to the field $GF(2^k)$ where $k = m \times n$. These 2 fields are of order $2^{m \times n}$. However, their complexity may be different depending on the choice of m , n and the irreducible polynomials [106].

Different composite fields for $GF(2^8)$ have been suggested in [98-102][107] for implementation of AES. The composite field is applied to the whole *SubBytes* in [99, 100], whereas researchers in [98, 102, 107] use composite fields in only the inversion of *SubBytes* as illustrated in Figure 29. The constant multiplications in *MixColumns* are shown to be more expensive in composite fields in [102]. Therefore, researchers in [102] concluded that the only operation that benefits from composite fields is the inversion of *SubBytes*, while the rest of the transformations are more efficient in the original $GF(2^8)$.

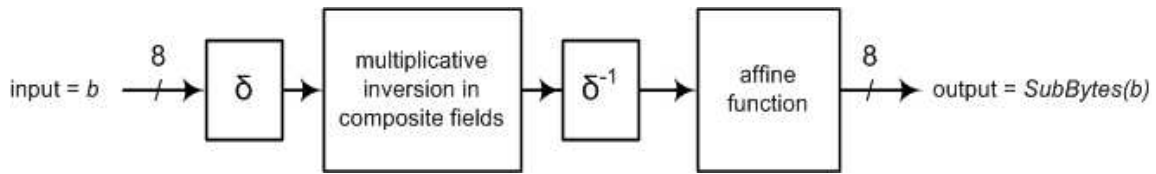


Figure 29 *SubBytes* with inversion in composite fields

In Figure 29, symbols δ and δ^{-1} are the linear functions for isomorphic mapping of elements from one field to another. These mapping functions are matrix multiplication. First, an input b is mapped from $GF(2^8)$ to the corresponding element in the composite field by δ . Then the inversion is done in the composite field. The inversion result is then mapped from the composite field to the corresponding element in $GF(2^8)$ by δ^{-1} . Finally, the *SubBytes* output is generated after applying the affine function.

The overall picture of the inversion of *SubBytes* in composite field $GF((2^4)^2)$ is shown in Figure 30. Aside from δ and δ^{-1} , which are matrix multiplications, the other operations are in $GF(2^4)$. These operations include square, XOR, multiplication, multiplication by a constant, and inversion. It should be noted that $GF(2^4)$ can be further decomposed to compute any of these operations.

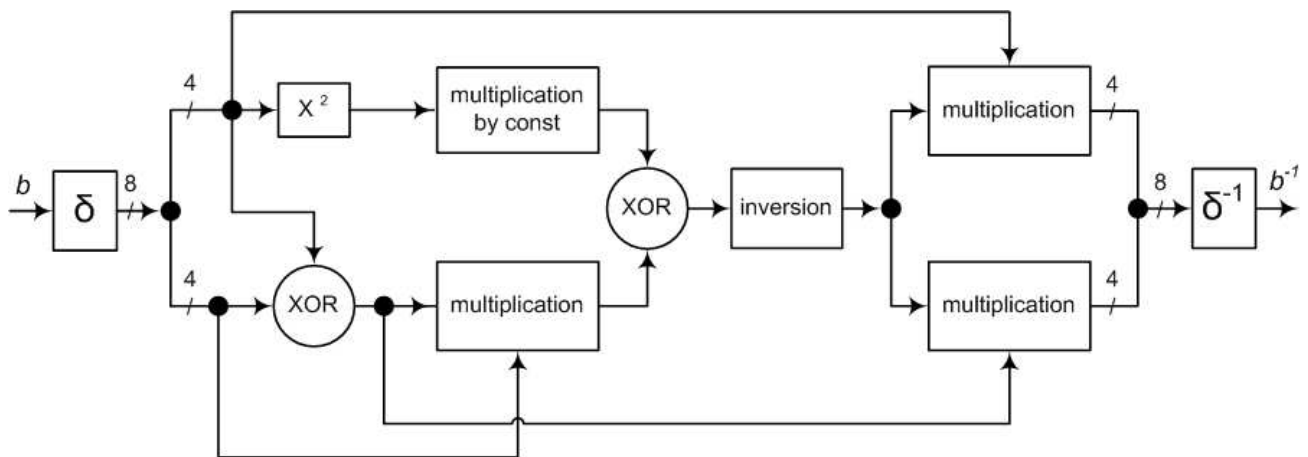


Figure 30 Inversion of *SubBytes* in composite fields [102]

In $GF((2^4)^2)$, it is assumed the irreducible polynomial of degree 2 is of the form $P(x) = x^2 + x + A$ where A in $GF(2^4)$. The inverse of an element $bx + c$ denoted $px + q$ is computed in Equation 15. Therefore, computing the inversion in $GF(2^8)$ is translated to computing the inversion of $(Ab^2 + bc + c^2)$ in $GF(2^4)$ instead. Equation 15 is illustrated in Figure 30.

$$\begin{cases} (bx + c)(px + q) = 1 \text{ mod } P(x) \\ p = b(Ab^2 + bc + c^2)^{-1} \\ q = (c + b)(Ab^2 + bc + c^2)^{-1} \end{cases} \quad (15)$$

Different composite fields result in different costs in terms of usage of hardware resources. $GF((2^4)^2)$, rather than $GF(2^8)$, is used in [100] with the polynomials shown in Equation 16 for $GF(2^4)$ and $GF((2^4)^2)$. Symbol ω is a generator of $GF(2^4)$ with polynomial $Q(y) = y^4 + y + 1$. In $P(x) = x^2 + x + \omega^{14}$, ω^{14} can be presented as binary vector (1001).

$$\begin{cases} GF(2^4) & : Q(y) = y^4 + y + 1 \\ GF((2^4)^2) & : P(x) = x^2 + x + \omega^{14} \end{cases} \quad (16)$$

As shown in Equation 17, $GF(2^8)$ is decomposed into $GF((2^4)^2)$ in [99]. However, a different constant value ($\beta = (1000)$ in binary vector form) from [100] is used for the polynomial of the Galois field. Then the inversion in $GF(2^4)$ is stored in block memory [100].

$$\begin{cases} GF(2^4) & : Q(y) = y^4 + y + 1 \\ GF((2^4)^2) & : P(x) = x^2 + x + \beta \end{cases} \quad (17)$$

Unlike [99, 100], researchers in [107] use further decomposition of $GF(2^8)$ to $GF(((2^2)^2)^2)$. The composite fields shown in Equation 18 are used in [107] for the inversion of *SubBytes*. Symbols ϕ and λ are (10) and (1100) in binary vector notation, respectively. This inversion is pipelined in [98] to achieve a high throughput for the AES on FPGA.

$$\begin{cases} GF(2^2) & : Q(z) = z^2 + z + 1 \\ GF((2^2)^2) & : P(y) = y^2 + x + \phi \\ GF(((2^2)^2)^2) & : R(x) = x^2 + x + \lambda \end{cases} \quad (18)$$

$GF(2^8)$ is decomposed into $GF((2^4)^2)$ with polynomial $P(x) = x^2 + x + \beta$ and constant $\beta = (1100)$ in [102]. The following equations were proposed to compute the 4-bit inversion in $GF(2^4)$. The 4-bit input and output of the inversion are $(x_3 \ x_2 \ x_1 \ x_0)$ and $(x'_3 \ x'_2 \ x'_1 \ x'_0)$, respectively.

$$\begin{cases} x'_3 = x_3 + x_3 x_2 x_1 + x_3 x_0 + x_2 \\ x'_2 = x_3 x_2 x_1 + x_3 x_2 x_0 + x_3 x_0 + x_2 + x_2 x_1 \\ x'_1 = x_3 + x_3 x_2 x_1 + x_3 x_1 x_0 + x_2 + x_2 x_0 + x_1 \\ x'_0 = x_3 x_2 x_1 + x_3 x_2 x_0 + x_3 x_1 + x_3 x_1 x_0 + x_3 x_0 + x_2 + x_2 x_1 + x_2 x_1 x_0 + x_1 + x_0 \end{cases} \quad (19)$$

As shown in Table 4.3, researchers in [102] provide a comparison of different inversion implementations in terms of number of gates on ASIC. The square-multiply approach is based on the Fermat's theorem[†]. According to this comparison, the composite fields used in [102] have the least gate count and shortest critical path compared to others.

[†] Suppose p is a prime. If $\gcd(a, p) = 1$, then $a^{p-1} \equiv 1 \pmod{p}$.

Table 4.3 Gate counts and critical paths of *SubBytes* inversions in $GF(2^4)$ [102]

<i>SubBytes</i> inversions in $GF(2^4)$	Total # of gates		Gates in critical path	
	XOR	AND	XOR	AND
Square-multiply approach	54	18	12	2
[98, 107]	17	9	7	2
[102]	14	9	3	2

4.3.2 *MixColumns* implementations in AES

Previous research on implementation of *MixColumns* is described in this section while the proposed *MixColumns* is described in Chapter 5 . Rearrangement of the *MixColumns* equation with respect to the structure of FPGA potentially results in a better optimized design in terms of utilizing hardware resources. Significant research [107-111] has been done on resource sharing between *MixColumns* and *InvMixColumns*; however, there is limited work on optimizing the *MixColumns* transformation on its own on FPGA. This can be applied directly in several modes (e.g., the NIST approved modes mentioned in Section 4.2) that do not need the decryption function.

Researchers in [98, 102] suggested the *MixColumns* shown in Equation 20 where a bit position $i \in \{0, 1, \dots, 7\}$. In the left column of this equation, 2 bytes are XORed and the multiplication by 2, $xtime()$ (refer to Equation 22), is then applied to the XORed result [98, 102].

$$\begin{bmatrix} a_i' \\ b_i' \\ c_i' \\ d_i' \end{bmatrix} = \begin{bmatrix} 2(a_i + b_i) \\ 2(c_i + b_i) \\ 2(c_i + d_i) \\ 2(a_i + d_i) \end{bmatrix} + \begin{bmatrix} b_i + c_i + d_i \\ a_i + c_i + d_i \\ a_i + b_i + d_i \\ a_i + b_i + c_i \end{bmatrix} \quad (20)$$

Researchers in [112] suggested the original *MixColumns* shown in Equation 12. In this approach, $xtime(z)$ and $xtime(z) + z$ are computed for each 8-bit element of the state column. There is also

Equation 21 used for *MixColumns* architecture in [108, 109]. These designs were all implemented on FPGA to fairly compare the experimental results (refer to Section 6.2) of *MixColumns*.

$$\begin{bmatrix} a_i' \\ b_i' \\ c_i' \\ d_i' \end{bmatrix} = \begin{bmatrix} a_i + b_i + c_i + d_i \\ a_i + b_i + c_i + d_i \\ a_i + b_i + c_i + d_i \\ a_i + b_i + c_i + d_i \end{bmatrix} + \begin{bmatrix} 2(a_i + b_i) \\ 2(c_i + b_i) \\ 2(c_i + d_i) \\ 2(a_i + d_i) \end{bmatrix} + \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix} \quad (21)$$

4.4 SEU-resistant AES

The main system level techniques that have been used to tackle SEUs in an AES implementation are the parity [76][77][78] and Hamming code [75]. The parity bits for the S-box values are stored in the block memory to cover errors in the memory cells and an extra block memory is used to cover errors in the memory decoder [76][78] (refer to Section 6.3). In the AES using composite fields, the parity prediction is presented in [77]. The parity prediction for the *MixColumns* transformation is provided in [78]. The *SubBytes* in composite fields suggested in [102] and parity predictions in [77] are further discussed in Chapter 6 (see Table 6.3). The Hamming code for error correction of AES was suggested in [75] for space applications.

The parity for S-box implemented using distributed RAMs was proposed in [113] for error detection. In [114], *SubBytes* in composite fields and its inverse were divided into blocks and the parities of these blocks were predicted.

A 32-bit datapath for a compact ASIC implementation of AES was proposed in [115]. In order to provide error detection, S-boxes were duplicated and parity bits were used for other AES transformations.

Researchers in [116] proposed a two-dimensional parity-based concurrent error detection method to detect errors in both horizontal and vertical direction in the data matrix for AES against differential fault attack (refer to Section 2.2.2).

None of the above techniques consider the underlying SRAM cells in routing or logic in an FPGA implementation of AES. Reconfiguration of the FPGA to ensure correct functionality of the implementation after an SEU is also not considered.

4.5 Summary

General security requirements of a system such as authentication, data confidentiality, data integrity, and non-repudiation were briefly described in this chapter. Then the primitives to provide these security needs (i.e. symmetric-key algorithms, public-key algorithms, and hash functions) were presented. Emphasis is placed on standard symmetric-key algorithm AES in this research.

Block cipher modes (recommended by NIST) which can be used with AES were also covered briefly. Error propagation of the AES algorithm due to confusion and diffusion properties was discussed. It has been shown that an error in the early rounds of AES encryption is expected to in 50% erroneous bits in the output [76]. It was discussed how the chaining structure in modes can further propagate errors throughout blocks.

Different approaches in implementing AES were presented. This provides clarification for the experimental results and comparisons presented in Chapter 6.

In summary, unlike previous AES implementations using the parity code [76][77][78] and Hamming code [75], the proposed design considers errors in the FPGA routing and logic. Considering the available resources on the FPGA, the dual ported block memory is suggested in this research for error detection in *SubBytes* as opposed to the parity coding in [76][77][78].

In the next chapter, the proposed AES implementation providing error detection is introduced. In the proposed error detection technique, some of the mathematical properties of AES and available hardware resources on FPGA are used to detect errors in *SubBytes* and the control circuitry implementations. Enhancements to the parity scheme (used for error detection in the *MixColumns* and *AddRoundKey* transformations) to increase its error coverage are also proposed in this research. In order to increase the error coverage of the parity technique, the weaknesses of it on FPGA are found. The enhancements are then discussed in 2 categories: combinational logic and routing.

Chapter 5

Proposed AES with Error Detection

The proposed error detection technique in AES detects errors in both logic blocks and routing on FPGA. An important category of errors is soft errors caused by radiation affecting SRAM cells that build and control every aspect of FPGA implementation (refer to Section 2.1). Soft errors cause a single error in 1 clock cycle (the likelihood of multiple errors is extremely low). The error detection in this research is considered early in design as opposed to being an after part to AES. The proposed error detection technique based on the parity scheme exploits some of the AES algorithm algebraic characteristics. The weaknesses of the parity scheme used in previous research and mitigation techniques to provide the lowest cost adequate method in this research are described in this chapter.

Error detection is investigated in 2 different categories: logic blocks and routing. First, the category of logic blocks is discussed. The proposed mapping of logic blocks on FPGA is examined thoroughly to ensure any single error is detected. Second, the category of routing is presented. A simple and yet accurate model for soft errors is verified and applied through experiments. This model is used to examine the routing of AES implemented on FPGA in terms of propagating single errors, to find its weaknesses and mitigate them in this research.

5.1 Error Detection in AES Logic Blocks

In this thesis, the term logic block is referred to as a slice of the FPGA shown in Figure 31. The XOR operation and multiplexers, the only logic operations used in *MixColumns* and *AddRoundKey*, are implemented using logic blocks, Figure 31 shows 2 LUTs within a slice on a Virtex-II Pro slice. A result bit exits a LUT through multiplexers. An error in a logic block configuration bit controlling a LUT cell or a multiplexer can eventually manifest itself as an erroneous result bit of a logic operation. Basically, an error in logic can be considered as an erroneous combinational bit (FX, Y, F5, X in Figure 31) or sequential bits (YQ, XQ in Figure 31).

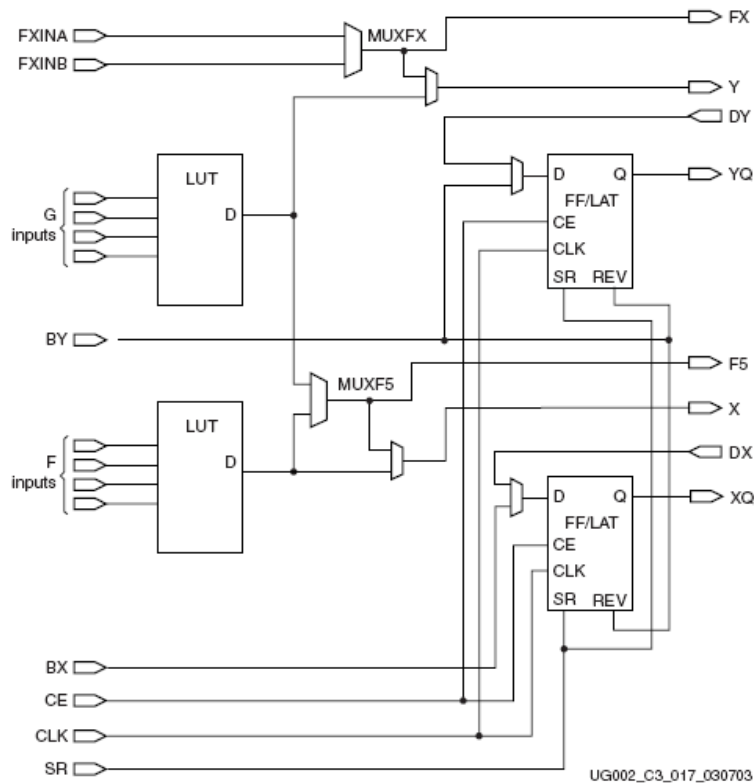


Figure 31 Virtex-II Pro slice [2]

The top level view of the proposed architecture providing error detection in the AES encryption datapath is illustrated in Figure 32.

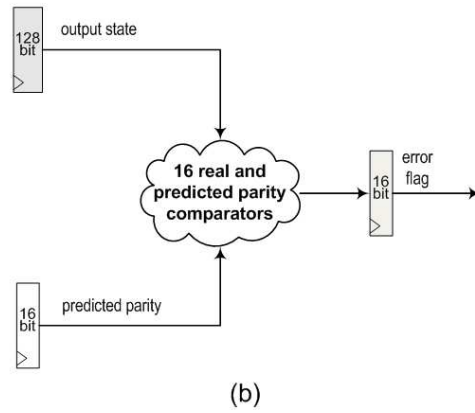
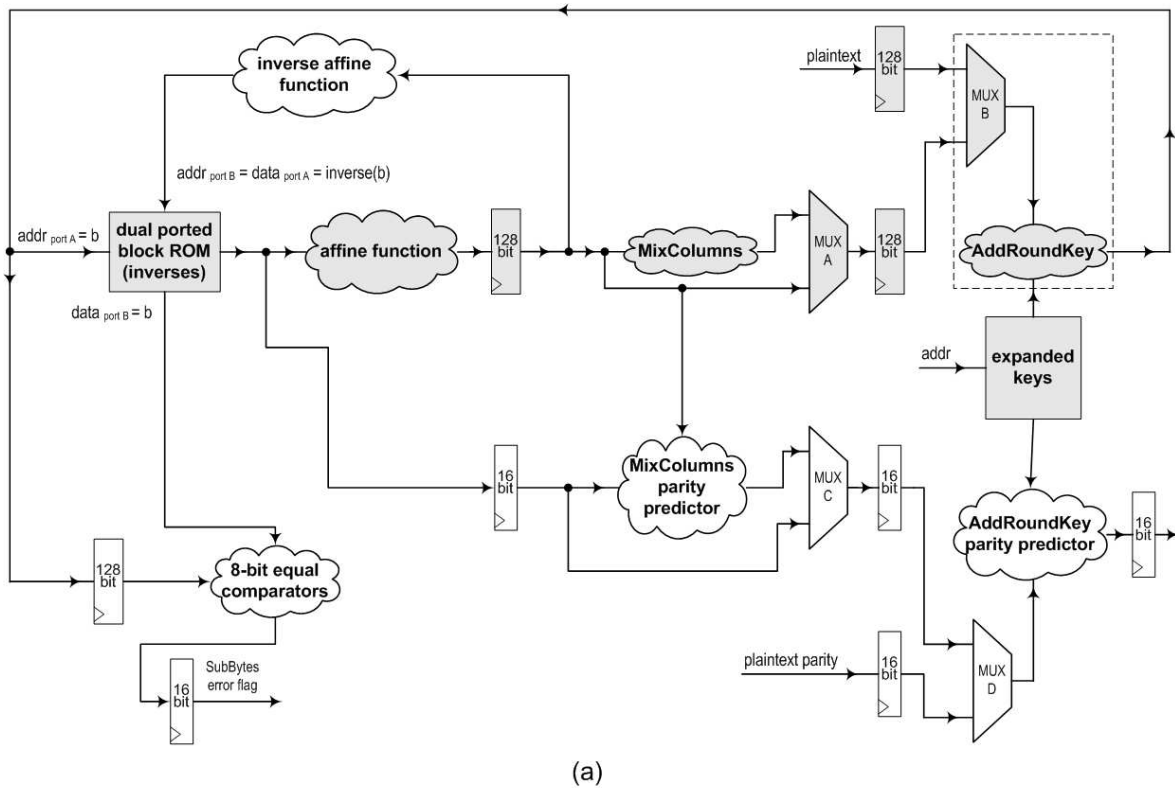


Figure 32 AES including: (a) parity predictors and comparator for *SubBytes* result, (b) real and predicted parity comparator for *MixColumns* and *AddRoundKey*

This section focuses on the logic blocks mapped on FPGA and error detection corresponding to logic blocks. In Figure 32(a), the grey elements represent AES (without showing *ShiftRows* that is

just rewiring, for the sake of simplicity) while the rest belong to error detection. Multiplexers (MUX A and B) select depending on the round number. In the initial round, the input plaintext is selected by MUX B for the initial *AddRoundKey* transformation while in the other rounds the output of *MixColumns* is sent to *AddRoundKey*. In the last round, the *MixColumns* transformation is skipped by MUX A. MUX C and MUX D select the corresponding party bits for each round. The select lines are derived by the control circuitry (shown in Appendix C) that mainly keeps track of the current round number and transformation. A block memory is used to produce the inverse in SubBytes. A dual ported block memory and the involution property in the inverse function motivate an attractive technique for error detection in *SubBytes*. Parity bits are used for *MixColumns* and *AddRoundKey*. Error detection for *ShiftRows* is not considered since it only cyclically shifts the rows of the state by different offsets (refer to Section 4.1.1).

5.1.1 SubBytes Logic Blocks and Error Detection

The *SubBytes* transformation shown in Equation (9) is the inverse function followed by the affine function (refer to Section 4.1.1 for more details). The inverse function is an involution. An involution is a type of function having the property that it is its own inverses. This is formally described in the following definition [82].

Definition: Let S be a finite set and let f be a bijection from S to S (i.e. $f : S \rightarrow S$). The function f is called an *involution* if $f = f^{-1}$. An equivalent way of stating this is $f(f(x)) = x$ for all $x \in S$.

The concept of error detection for involution ciphers was first introduced in [117]. The involution property proposed in [117] is used to detect errors in the implementation of *SubBytes* using the dual ported block memory in this research. In Figure 32(a), the 8-bit input on the address line of $port_A$ is denoted b ($addr_{portA} = b$). The 8-bit output on data line of $port_A$ is the inverse of b ($data_{portA} = inverse(b)$). Then $inverse(b)$ is passed through the affine function, fed back through the inverse of the affine function, and it finally reaches the address line of $port_B$ ($addr_{portB} = inverse(b)$). If there have not been any errors in reading the block memory or in

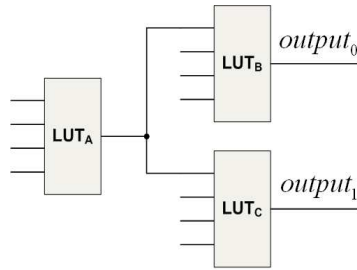
computation of affine function then data line of $port_B$ should show b , in other words ($data_{portB} = inverse(inverse(b)) = b$). In other words, if $data_{portB} \neq addr_{portA}$ (or equivalently $inverse(inverse(b)) \neq b$) then some error, possibly multiple errors, have occurred in the block memory or affine function. The occurrence of errors triggers the *SubBytes* error flag depicted in Figure 32(a).

5.1.2 *MixColumns* Logic Blocks and Error Detection

An enhanced parity scheme is used for error detection of *MixColumns*. The goal is to detect any single errors in logic blocks, as opposed to detecting only the single errors in the output registers in previous research. In the error detection scheme, a parity bit is predicted for each 8-bit element of the output state of *MixColumns*. The *MixColumns* output state and 16 corresponding parity bits are computed in parallel. At the output registers, as is shown in Figure 32(b), real parity bits computed directly from the output state are compared to the corresponding predicted parity bits to detect an error in the 8-bit registers of the *MixColumns* state. There are 2 important requirements in implementing the enhanced parity scheme.

- First, a single error in the logic blocks should not affect an even number of bits in an 8-bit element of the state.
- Second, a single error in the logic blocks should not affect the parity prediction and output producing circuit simultaneously such that the error is not detected.

If these 2 important factors are not considered in the design mapped on FPGAs there could be cases that a single error can be missed without being detected. Figure 33 illustrates a simple example to clarify the points discussed above. Assume that output bits $output_0$ and $output_1$ shown in Figure 33 belong to an 8-bit element $output$ that has a parity bit for error detection. The term $output_i$ represents the i^{th} bit of $output$, where $output$ is 8-bit wide. Each LUT in Figure 33 implements a 4-input XOR gate. In general, if there is an error in LUT_A both output bits, $output_0$ and $output_1$, are affected. Therefore a parity bit is not able to detect that single error in LUT_A . As can be seen from this example, the reason for this undesired outcome is the sharing of logic blocks (in this example, LUT_A in Figure 33) in the circuit that generates an 8-bit element ($output$). Therefore, after mapping a design onto a FPGA, shared logic blocks need to be thoroughly examined to find if any of the above 2 requirements has been violated.



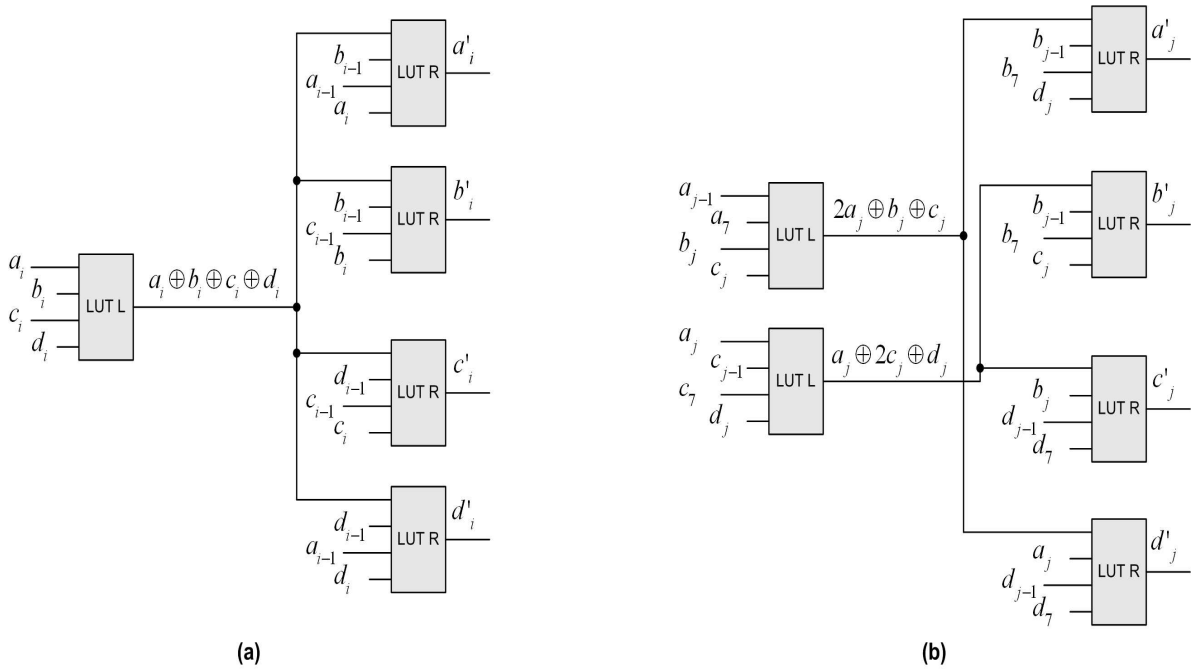
LUT: representing 4-input XOR

Figure 33 Single error in LUT_A causing 2 bit flips in output

The FPGA mapping of *MixColumns* on LUTs is depicted in Figure 34. Four 8-bit elements of a column of input state are shown as a , b , c , and d . The *MixColumns* equation is expanded and rearranged so that it uses the smallest number of LUTs on an FPGA that contains 4-input LUTs. The proposed *MixColumns* mapping uses the smallest number of LUTs compared to previous research discussed in Section 6.2. In order to achieve this area optimization, the proposed architecture distinguishes 2 different groups of output bits in 8-bit elements as follows.

- The output bits at positions $\{0, 2, 5, 6, 7\}$ for which the $xtime()$ operation shown in Equation 22 requires only a bit shift.
- The output bits at positions $\{1, 3, 4\}$ for which the $xtime()$ operation shown in Equation 22 requires a bit shift and an XOR.

This grouping of bits in 8-bit elements of the *MixColumns* input column, described above, is based on the multiplication by 2 equation denoted function $xtime()$ in Equation 22. In this equation, z and z' are 8-bit input and output elements, respectively. $xtime()$ can be implemented by 3 XOR operations and 4 single-bit shifts.




 : Representing a 4-input LUT generating the 4-input XOR, L and R correspond to left and right LUT

Figure 34 Proposed *MixColumns* LUTs mapped on FPGA: (a) bit position $i \in \{0, 2, 5, 6, 7\}$, (b) bit position $j \in \{1, 3, 4\}$

$$\text{input} : z = (z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0)$$

$$\text{output} : z' = 2z = \text{xtime}(z) = (z_7' z_6' z_5' z_4' z_3' z_2' z_1' z_0')$$

$$z'_0 = z_7$$

$$z'_1 = z_0 + z_7$$

$$z'_2 = z_1$$

$$z'_3 = z_2 + z_7$$

$$z'_4 = z_3 + z_7$$

$$z'_5 = z_4$$

$$z'_6 = z_5$$

$$z'_7 = z_6$$

(22)

The expansion and rearrangement of *MixColumns* in Equation 23 is proposed for the first group where a bit position $i \in \{0, 2, 5, 6, 7\}$, $i-1$ is performed modulo 8, and the operator $+$ is addition over GF^8 . The term $a_i + b_i + c_i + d_i$ is shared amongst all 4 bits (a'_i, b'_i, c'_i, d'_i) of the output state column.

$$\begin{aligned}
 \begin{bmatrix} a'_i \\ b'_i \\ c'_i \\ d'_i \end{bmatrix} &= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix} \\
 &= \begin{bmatrix} a_i + b_i + c_i + d_i \\ a_i + b_i + c_i + d_i \\ a_i + b_i + c_i + d_i \\ a_i + b_i + c_i + d_i \end{bmatrix} + \begin{bmatrix} 2b_i + 2a_i + a_i \\ 2b_i + 2c_i + b_i \\ 2d_i + 2c_i + c_i \\ 2d_i + 2a_i + d_i \end{bmatrix} \\
 &= \begin{bmatrix} a_i + b_i + c_i + d_i \\ a_i + b_i + c_i + d_i \\ a_i + b_i + c_i + d_i \\ a_i + b_i + c_i + d_i \end{bmatrix} + \begin{bmatrix} b_{i-1} + a_{i-1} + a_i \\ b_{i-1} + c_{i-1} + b_i \\ d_{i-1} + c_{i-1} + c_i \\ d_{i-1} + a_{i-1} + d_i \end{bmatrix}
 \end{aligned} \tag{23}$$

Equation 23 is then mapped to 2 levels of 4-input LUTs of FPGA shown in Figure 34(a). The resource sharing amongst 4 rows of the state column where $i \in \{0, 2, 5, 6, 7\}$ is illustrated in this figure where each LUT implements the 4-input XOR function.

Next, the *MixColumns* transformation is expanded and rearranged as shown in Equation 24 for the second group of bits where a bit position $j \in \{1, 3, 4\}$. Figure 34(b) shows Equation 24 mapped to 2 levels of 4-input LUTs for the second group of bits. This mapping allows $2a_j + b_j + c_j$ and $a_j + 2c_j + d_j$ to be shared between the (a'_j, d'_j) and (b'_j, c'_j) output bits of the state column, respectively. There are totally forty three 4-input XORs in the proposed *MixColumns* implementation. The 4-input LUTs generating the 4-input XORs had to be manually instantiated to produce the desired LUT schematic in Figure 34 after synthesis.

$$\begin{aligned}
 \begin{bmatrix} a'_j \\ b'_j \\ c'_j \\ d'_j \end{bmatrix} &= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_j \\ b_j \\ c_j \\ d_j \end{bmatrix} \\
 &= \begin{bmatrix} 2a_j + b_j + c_j \\ a_j + 2c_j + d_j \\ a_j + 2c_j + d_j \\ 2a_j + b_j + c_j \end{bmatrix} + \begin{bmatrix} 2b_j + d_j \\ 2b_j + c_j \\ b_j + 2d_j \\ a_j + 2d_j \end{bmatrix} \\
 &= \begin{bmatrix} a_{j-1} + a_7 + b_j + c_j \\ a_j + c_{j-1} + c_7 + d_j \\ a_j + c_{j-1} + c_7 + d_j \\ a_{j-1} + a_7 + b_j + c_j \end{bmatrix} + \begin{bmatrix} b_{j-1} + b_7 + d_j \\ b_{j-1} + b_7 + c_j \\ b_j + d_{j-1} + d_7 \\ a_j + d_{j-1} + d_7 \end{bmatrix}
 \end{aligned} \tag{24}$$

The first consideration, in the enhanced parity scheme, is that a single error in LUTs mapped on the FPGA should not affect an even number of bits in an 8-bit element of the output state. To investigate this, logic blocks that are shared between output bits need to be carefully examined. In Figure 34, where each LUT is an XOR, there is no logic block sharing among bits of an 8-bit result (for instance, there is no LUT sharing among bits $\{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$ in the 8-bit element a). However, there is LUT sharing between different 8-bit elements of a state column. For instance, although there is no LUT sharing between a_i and a_j , there is an LUT shared between a_i and b_i in Figure 34. Therefore, a single error in a shared LUT is detected by different bits of the 16-bit error flag register. Consequently, there are not any single errors in logic blocks that affect an even number of bits in an 8-bit element of the state. Thus single errors are not missed and thus the first consideration of the enhanced parity scheme is met.

The parity prediction that was given and proved in [76] is expanded and rearranged as shown in Equation 25 with respect to the 4-input LUT structure of FPGA to reduce the number of LUTs. $P_{out,z}$ and $P_{in,z}$ correspond to the output and input parity bits of an 8-bit element z in Equation 25.

The error detection hardware overhead is five 4-input LUTs per column of the state in the *MixColumns* transformation implemented on FPGA.

$$\begin{bmatrix} P_{out,a'} \\ P_{out,b'} \\ P_{out,c'} \\ P_{out,d'} \end{bmatrix} = \begin{bmatrix} P_{in,a} + P_{in,b} + P_{in,c} + P_{in,d} \\ P_{in,a} + P_{in,b} + P_{in,c} + P_{in,d} \\ P_{in,a} + P_{in,b} + P_{in,c} + P_{in,d} \\ P_{in,a} + P_{in,b} + P_{in,c} + P_{in,d} \end{bmatrix} + \begin{bmatrix} P_{in,b} + a_7 + b_7 \\ P_{in,c} + b_7 + c_7 \\ P_{in,d} + c_7 + d_7 \\ P_{in,a} + a_7 + d_7 \end{bmatrix} \quad (25)$$

The second consideration is that a single error should not affect the parity prediction and output producing circuits simultaneously such that the error is not detected. Any single error that affects both logic blocks of parity prediction and output producing circuits is detected in the *SubBytes* error detection where it goes to the inverse affine functions in Figure 32(a). Consequently, there are not any single errors in logic blocks that affect both the *MixColumns* outputs and predicted parity bits without being detected. It should be noted that errors in routing are not considered yet and are discussed later in Section 5.2.

5.1.3 AddRoundKey Logic Blocks and Error Detection

In the logic blocks related to *AddRoundKey*, 1 LUT combining multiplexer MUX B and *AddRoundKey* (dashed square in Figure 32(a)) is used for each output bit results. There is no sharing between logic blocks of bits in an 8-bit element (for example, there is no LUT sharing among bits $\{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$ in the 8-bit element a), and thus the first consideration is met.

The parity prediction and output producing circuits do not share any logic blocks, therefore the second consideration is met as well. The block memory of round keys stores the parity bits for the *AddRoundKey* parity predictor in Figure 32. Consequently, any single error in a logic block related to *AddRoundKey* does not affect more than 1 output bit and is detected by the parity scheme.

5.2 Error Detection in Routing of AES

Since the proposed error detection of the *SubBytes* (refer to Section 5.1.1) is capable of detecting multiple errors, it does not have issues with a case where there is more than a single error whether in

the logic blocks or routing. Thus, any faults that cause multiple errors at the output of *SubBytes* are detected.

However, since the parity scheme (capable of detecting singles errors only) is used in the *MixColumns* and *AddRoundKey* transformations, the routing needs to be investigated. Since a single error in the routing can affect multiple output bits, the error can be missed in error detection of output bits using the parity scheme. First, the error in routing is modeled in Section 5.2.1. Then this modeling is verified by inserting errors into a small part of the *MixColumns* implementation and observing the effects on the output bits.

5.2.1 Error in Routing and Modeling

This section focuses on the routing within the FPGA. Routing provides interconnections between logic blocks through nets [118][16]. A net contains static (non-configurable) wires, which are embedded in the FPGA fabric, as well as a configurable part. In a net, the configurable part which is called Programmable Interconnect Points (PIPs) provides connections between these static wires; a PIP is shown in Figure 35.

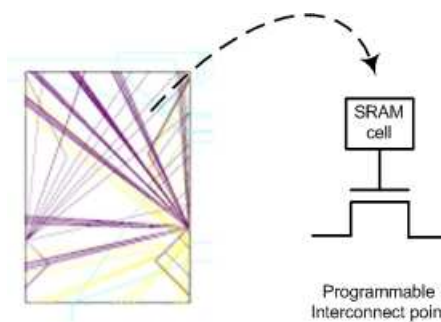


Figure 35 Switch box and PIP controlled by SRAM cell

A PIP is basically a CMOS transistor switch that can be programmed to be turned on or off by an SRAM cell. Switch boxes are a collection of switches located between logic blocks. This allows some of the wire segments incident to the switch box to be connected to others. The term pin refers to a physical point in the FPGA. For instance, a pin can be an input or output point attached to a LUT, flip-flop, or a multiplexer. A snap shot of a switch box and a PIP is shown in Figure 35 (note that pin connections through the switch box are provided by PIPs).

Since routing of a design is complex a simple yet accurate model needs to be defined so that effects of errors particularly soft errors can be understood. Input pins connect to logic blocks that compute each transformation and produce the values on 8-bit output pins of output flip-flops. Errors manifest themselves at pins. These pins either connect to combinational logic blocks (that can propagate an error) or sequential logic blocks (flip-flops that can store invalid data). Therefore, a pin fault model is used. Since the likelihood of multiple errors (in the context of soft errors) on input pins is extremely low the routing of each input pin is considered separately.

In *MixColumns* and *AddRoundKey*, where the parity scheme is used, the only logic operation that is needed is XOR. An XOR operation does not mask an error. For instance, as is shown in Figure 36, a 4-input XOR does not stop the propagation of a bit flip on its input pin (a bit flip is an error in the context of error detection). Therefore, combinational logic blocks propagating bit flips (or equivalently errors) can be ignored when the routing is being examined.

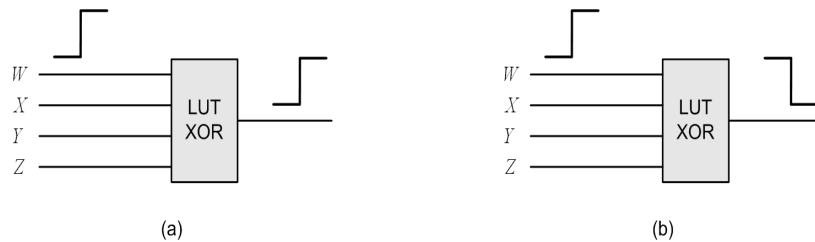


Figure 36 XOR LUT propagating error whether (a) $X \oplus Y \oplus Z = 0$ or (b) $X \oplus Y \oplus Z = 1$

In order to verify the pin fault model for soft errors in SRAM cells of routing, a small part of *MixColumns* is tested on the FPGA. This is discussed in more detail in the next section.

5.2.2 *MixColumns* Routing and Error Detection

The enhanced parity scheme is used in *MixColumns* and *AddRoundKey* to detect errors in the routing other than logic blocks. The 2 factors that are considered in the logic blocks (refer to Section 5.1.2) should be investigated in the routing as well.

- First, a single error in routing should not affect an even number of bits in an 8-bit element of the state.

- Second, a single error in routing should not affect the parity prediction and output producing circuit simultaneously such that the error is not detected.

A design that provides any single error detection in logic blocks can have a routing circuit that does not achieve an overall 100% single error detection. To clarify this problem in routing, an example is given in Figure 37. This figure depicts a small part of a circuit that implements a parity scheme for two 8-bit elements. These two 8-bit elements in Figure 37 are $output_a$ and $output_b$ (i^{th} bit of $output_a$ is $output_a_i$). Each 8-bit element has its own parity bit. Ignoring the routing circuitry in Figure 37(a), any single error in the logic blocks is detected, specifically by parity of $output_b$ (thus the parity of $output_a$ does not need to, nor will, detect the error).

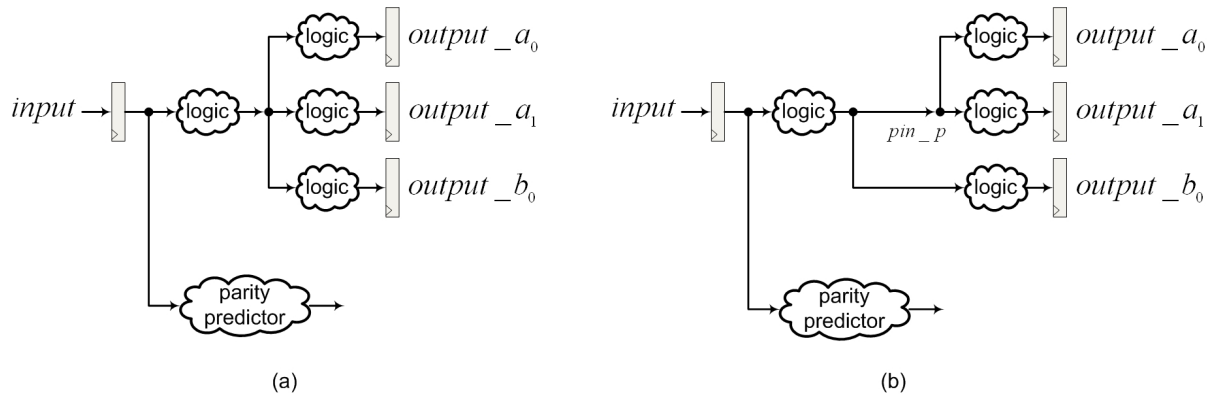


Figure 37 Routing example (a) logic blocks without considering routing, (b) actual routing detail showing pin_p

Consider the pin labeled pin_p in Figure 37(b). If a single error affects the value at pin_p then both $output_a_0$ and $output_a_1$ can be erroneous. In this case, since 2 bits of the 8-bit element $output_a$ are flipped the parity bit is not able to detect this error. As can be seen, this undesirable effect is due to a multiple fanout signal (at pin_p in Figure 37) that is connected to multiple bits of an 8-bit element ($output_a_0$ and $output_a_1$).

This simple example demonstrates the importance of examining the effectiveness of the parity scheme implementation in detecting single errors in routing other than logic blocks. In this research, the goal is to find out all the pins leading up to potential undetectable single errors and provide a mitigation technique.

As discussed in Section 5.1.2, weaknesses of the parity scheme in logic blocks are due to resource sharing whereas weaknesses in routing are due to multiple fanout signals according to the pin fault model. In order to verify this on the FPGA, a small part of *MixColumns*, shown in Figure 38, is implemented with a multiple fanout signal at pin_p . The output bits connected to LEDs are observed to find if the multiple fanout signal causes this problem in routing. Figure 39 illustrates more detailed information i.e. slices and SRAM cells. The snapshot of the FPGA Editor is shown in Figure 66 and Figure 67 of Appendix H.

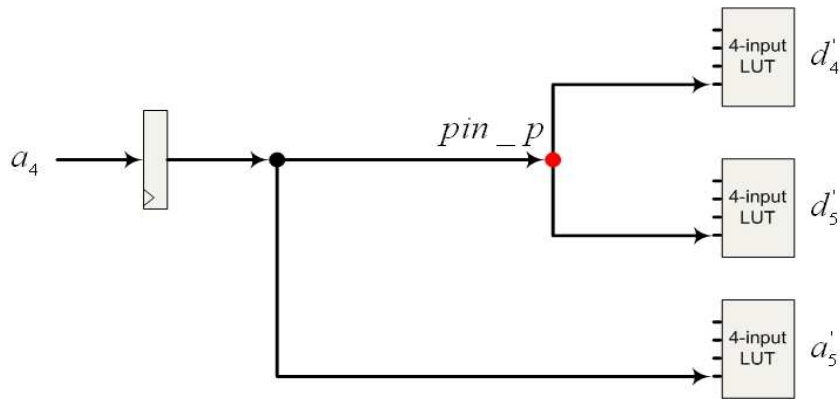


Figure 38 Routing of 2 fanout signal in *MixColumns* tested on FPGA

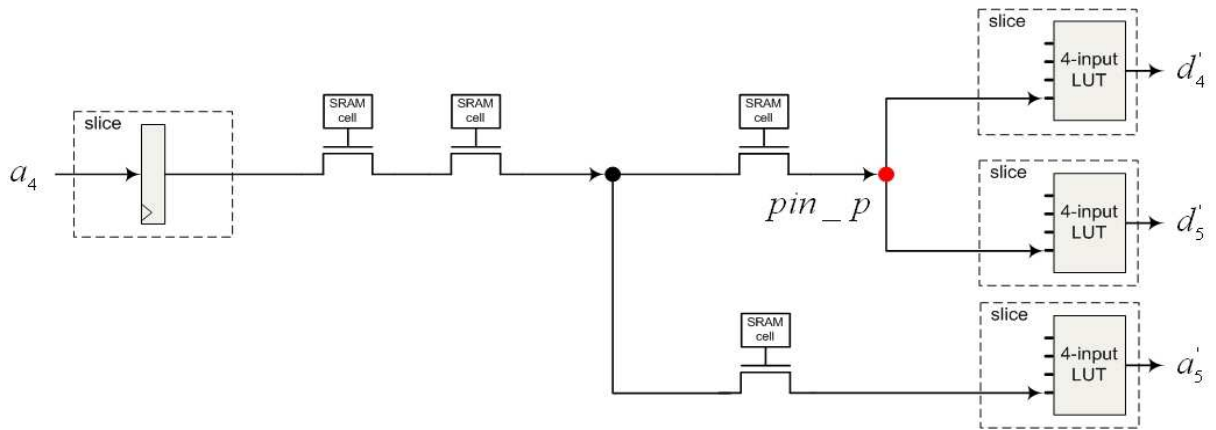


Figure 39 Detailed routing of 2 fanout signals in *MixColumns* tested on FPGA

In Figure 38, the input flip-flop is one bit result (a_4 of 8-bit element a) from *SubBytes* and the LUTs implement 4-input XOR operations in *MixColumns*. The output bits (d'_4 , d'_5 , and a'_5) are 3 bits of the *MixColumns* result. Output bits d'_4 and d'_5 are 2 bits of the 8-bit element d' . According to the pin fault model, if there is an error in the routing that affects both these bits (d'_4 and d'_5) it is not detected by the parity bit of 8-bit element d .

5.2.2.1 Experimental Validation of Proposed Routing Mitigation Technique

In order to investigate the effects of single errors on FPGA, the configuration bits that control the routing should be manually flipped one at a time to simulate a soft error occurrence. Then the output bits connected to LEDs need to be observed to examine the effect of an error on the result. Other techniques used previously to estimate the soft error rate for a device (not a specific design implementation) are expensive accelerated testing using particle beams, software simulation of a circuit (different quantitative models based on Q_{crit}), and estimation by real particles. In this research, soft errors are simulated on a specific net by flipping the relevant configuration bits and the effects are observed. Then the mitigation technique proposed is verified by the second soft error simulation on the specific net.

The FPGA used in this research is Virtex-II Pro whose configuration file size is 34,292,768 bits [2]. It is important to note that the mapping of a netlist after place and route onto the configuration bits (or FPGA SRAM cells) is proprietary information. Therefore, there is no direct way of finding configuration bits that are related to the net between the flip-flops and LUTs in Figure 38 so they can be flipped for the experiment to simulate SEUs.

To overcome this problem the net is manually removed from the design netlist by the FPGA Editor tool and the modified configuration file is generated. Then the original configuration file is compared with the modified configuration file by a program written in C++. After running this comparison in software there are totally 14 bits that are different in these 2 configuration files. These different bits indicate the bits that are related to the net (between the flip-flops and LUTs in Figure 38) that is removed in the modified netlist. Next, these 14 bits related to the routing are flipped one at a time. Figure 40 illustrates the flow to simulate SEUs in a net.

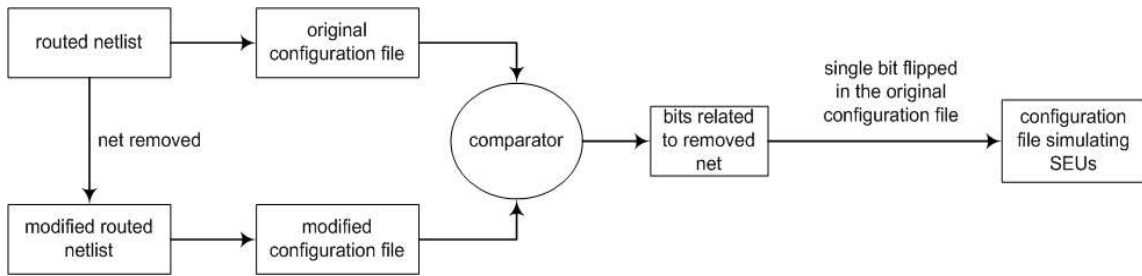


Figure 40 SEU simulation in net

After each bit flip, the configuration file is downloaded on the FPGA and the output bits (d'_4 , d'_5 , and a'_5) connected to LEDs are observed. Cyclic Redundancy Checking (CRC) is turned off during the configuration. The effects of single errors simulating soft errors in routing on the output bits are categorized in Table 5.1.

Table 5.1 Effects of single errors in net including pin_p on output bits

output bits	d'_4	d'_5	a'_5
effects of single errors in net on output bits	wrong	wrong	correct
	wrong	correct	correct
	correct	wrong	correct
	correct	correct	wrong

As seen in the first row of Table 5.1, a single error in the routing can affect both d'_4 and d'_5 . In this case, since 2 bits of the 8-bit element d' are affected the parity scheme does not detect the error. This example shows that the pin fault model while ignoring the XORs can accurately demonstrate the effects of single errors on the FPGA. In Figure 38, as predicted in the pin fault model a single error on pin_p affects both d'_4 and d'_5 .

In general, if there are multiple output bits of an 8-bit element that are connected to a pin of a net that pin can lead up to undetectable single errors. Mitigating this problem requires modification in the

routing. However, there is not much modification that can be done on a netlist at such a fine-grained level by the tool. Additionally, even if this modification at the placed and routed netlist were possible it would make the design process tedious.

A mitigation technique is introduced in this research that is done at the register-transfer level. The goal is to avoid pins causing undetectable single errors (pins to which multiple bits of an 8-bit output are connected) to avoid an even number of errors in the output (an even number of errors are not detected by the parity scheme). The proposed technique uses extra flip-flops in order to force the FPGA tool at the register-transfer level to eliminate these pins. For instance, Figure 41 illustrates the proposed technique to eliminate pin_p shown in Figure 38. The snapshot of the FPGA Editor is shown in Figure 66 and Figure 67 of Appendix H.

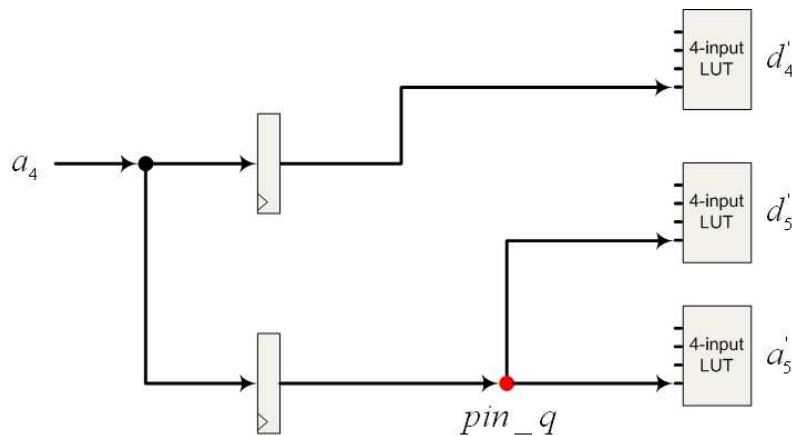


Figure 41 Routing with no pins leading up to undetectable errors

Unlike the routing in Figure 38, there are no single errors in the routing shown in Figure 41 that affects d'_4 and d'_5 only. This is verified by flipping the routing bits (the workaround to find the related configuration bits in routing was discussed for the experiment in Figure 38) and the effects of single errors in the net including pin_q is categorized Table 5.2. As seen in this table there is no cases in which d'_4 and d'_5 are wrong only. In the case where d'_5 and a'_5 are affected, parties of 8-bit elements d and a both detect the error in the net.

Table 5.2 Effects of single errors in net without any pins causing undetectable errors at the output

output bits	d'_4	d'_5	a'_5
effects of single errors in net on output bits	correct	wrong	wrong
	correct	wrong	correct
	correct	correct	wrong

5.2.2.2 Detailed *MixColumns* SEU-resistant Routing

Next, the proposed technique using extra flip-flops in routing is going to be applied throughout the *MixColumns* implementation. The first step is to find the pins leading to undetectable errors in the nets of *MixColumns* shown in Figure 34. The extra flip-flops are used to avoid these pins.

There are 2 levels of LUTs, referred to as L for LUTs on the left and R for LUTs on the right in each circuit of Figure 34. The net between LUTs L and LUTs R in Figure 34(a) is connected to 4 input ports of LUTs which define bits (a'_i, b'_i, c'_i, d'_i) . Each of these bits belongs to a different 8-bit element (a'_i, b'_i, c'_i, d'_i belong to a', b', c', d' , respectively). Therefore, there is no pin which has multiple fanout signals among bits of an 8-bit element. This holds true for the nets between LUTs L and LUTs R in Figure 34(b) as well. As seen in Figure 34(b), there are 2 multiple fanout signals connected to (a'_j, d'_j) and (b'_j, c'_j) . Therefore, there are no pins causing undetectable errors in routing between LUTs L and LUTs R in Figure 34(b) either.

Investigating signals that are not necessarily between LUT L and LUT R is more complicated. The available routing between an input pin and output pin of LUTs is shown in Table 5.3 to Table 5.6. Letters L and R are associated to the left and right LUT connected in a signal path to and connected directly to an output pin, respectively. For example, in row of input a_7 in Table 5.3, a_7 is connected to the input pin of several LUT Ls which are connected to LUT Rs to output signals $a'_1, a'_3, a'_4, a'_7, b'_7, c'_7, d'_1, d'_3, d'_4, d'_7$ and a_7 is also connected to the input pin of three LUT R's whose output pins are a'_0, a'_7, d'_0 . Table 5.3 presents a notation for labeling of the LUTs (or equivalently

input pins to LUTs). In the last row of Table 5.3 for input a_7 , $a'_7(L)b'_7(L)c'_7(L)d'_7(L)$ refers to LUT L connected to output bits a'_7, b'_7, c'_7 , and d'_7 in Figure 42. Additionally in Figure 42 $a'_7(R)$ (also listed in last row of Table 5.3) refers to LUT R connected to a'_7 .

Table 5.3 Input nets to input pins of LUTs of 8-bit element a

Input	a'	b'	c'	d'
a_0	$a'_0(L,R), a'_1(L)$	$b'_0(L)$	$c'_0(L)$	$d'_0(L), d'_1(L)$
a_1	$a'_2(R)$	$b'_1(L)$	$c'_1(L)$	$d'_1(R), d'_2(R)$
a_2	$a'_2(L,R), a'_3(L)$	$b'_2(L)$	$c'_2(L)$	$d'_2(L), d'_3(L)$
a_3	$a'_4(L)$	$b'_3(L)$	$c'_3(L)$	$d'_3(R), d'_4(L)$
a_4	$a'_5(R)$	$b'_4(L)$	$c'_4(L)$	$d'_4(R), d'_5(R)$
a_5	$a'_5(L,R), a'_6(R)$	$b'_5(L)$	$c'_5(L)$	$d'_5(L), d'_6(R)$
a_6	$a'_6(L,R), a'_7(R)$	$b'_6(L)$	$c'_6(L)$	$d'_6(L), d'_7(R)$
a_7	$a'_0(R), a'_1(L), a'_3(L), a'_4(L), a'_7(L,R)$	$b'_7(L)$	$c'_7(L)$	$d'_0(R), d'_1(L), d'_3(L), d'_4(L), d'_7(L)$

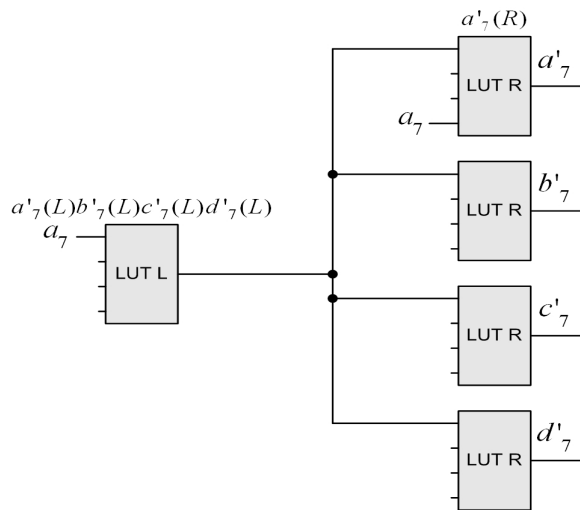


Figure 42 LUT labels $a'_7(L)b'_7(L)c'_7(L)d'_7(L)$ and $a'_7(R)$ related to input a_7 in Table 5.3

The complete row of input a_7 is illustrated in Figure 43. For example, a'_0 (R) refers to the LUT R in the middle circuit in Figure 43 (4th column of LUTs from the left), specifically LUT R whose input is a_7 and whose output is a'_0 . Similarly, a'_1 (L) refers to the LUT L in the circuit on the right in Figure 43 whose output is routed to 2 LUT Rs one of which outputs a'_1 .

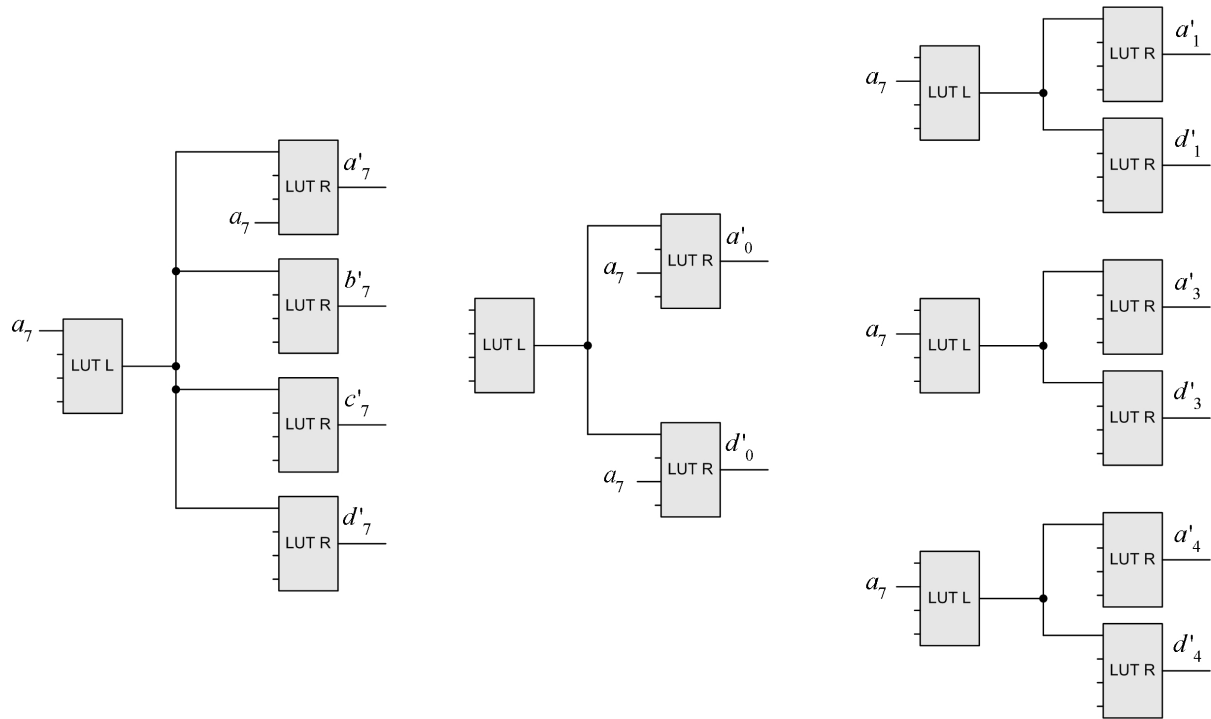


Figure 43 Input a_7 connection to output bits a'_0 , a'_1 , a'_3 , a'_4 , a'_7 , b'_7 , c'_7 , d'_0 , d'_1 , d'_3 , d'_4 , and d'_7 through LUTs

The proposed mitigation is discussed for input a_7 then the same concept is expanded for the whole *MixColumns* design. There is a potential of errors not being detected when there is a multiple fanout signal connecting input a_7 to the following LUTs:

- a'_0 (R), a'_1 (L), a'_3 (L), a'_4 (L), a'_7 (R)
- d'_0 (R), d'_1 (L), d'_3 (L), d'_4 (L)

The input pin of the far left LUT L in Figure 43 (this LUT is referred to as $a'_7(L)b'_7(L)c'_7(L)d'_7(L)$) is not included since any single error in this a_7 net that affects the results at $a'_7(L)$ and $d'_7(L)$ is detected by the parity bits of b' and c' (LUT L is shared between bits a'_7 , b'_7 , d'_7 , and c'_7), even though this error might not be detected by parity the bits of a' and d' . For instance, an error in the a_7 net that affects $a'_7(L)$ and $a'_0(R)$ is not detected by the parity bit of a' (2 errors are not detected by a parity bit); however the parity bits of b' and c' detect this error. The proposed technique, shown in Figure 44, uses extra flip-flops at the register-transfer level to prevent pins which cause undetected single errors in a net. For instance, in the a_7 net discussed above, the mapped design illustrated in Figure 44 does not have any of these pins.

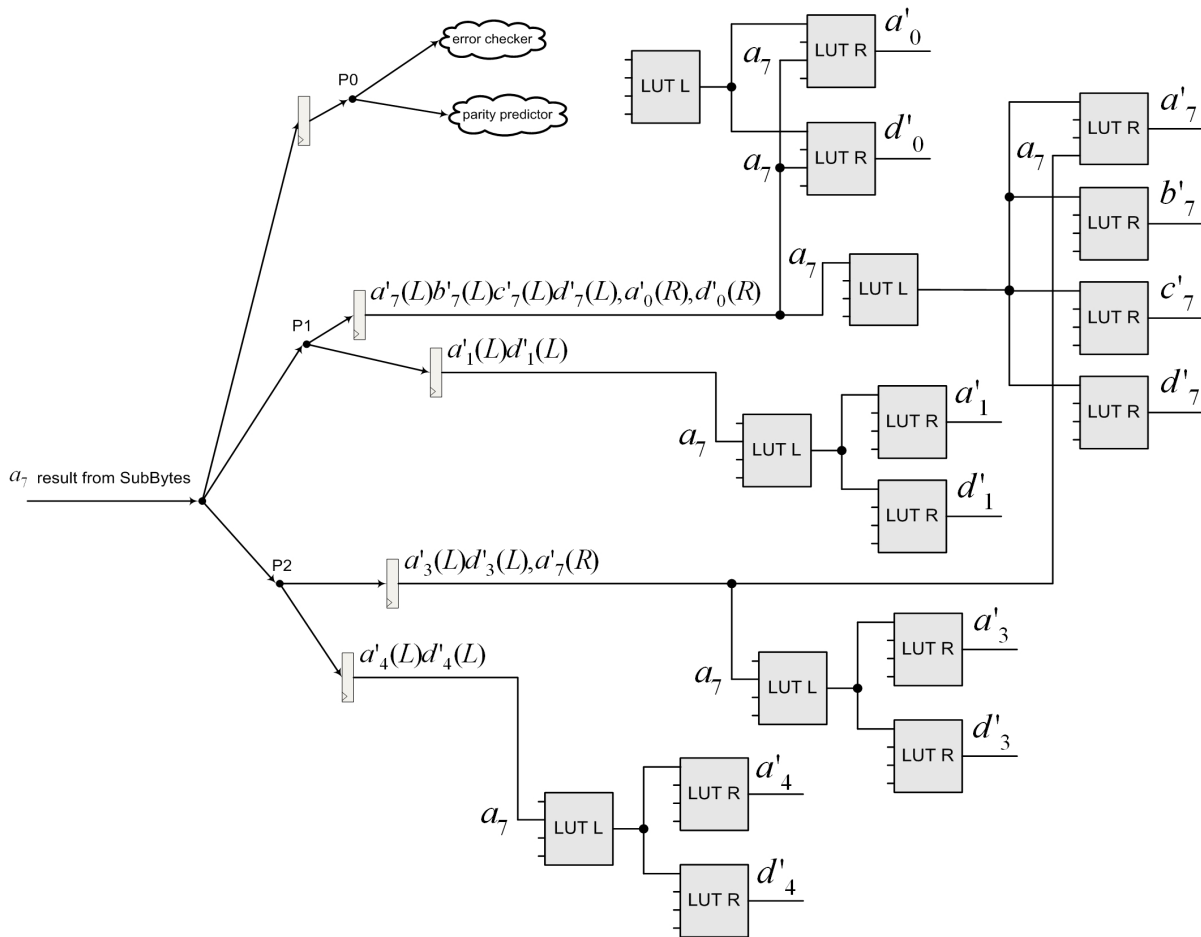


Figure 44 Proposed routing applied to net a_7

For further clarification, Figure 45 shows part of the original *MixColumns* datapath (see Figure 32(a) for the complete datapath) which was modified in Figure 44. The a_7 flip-flop (resulting from *SubBytes*) in Figure 45 is copied 5 times in Figure 44 to prevent undetected single errors in routing. Input a_7 generates output bits $a'_0, a'_1, a'_3, a'_4, a'_7, b'_7, c'_7, d'_0, d'_1, d'_3, d'_4,$ and d'_7 . The error checker (of Figure 44) is basically the part of datapath (in Figure 32(a)) that includes inverse affine function (refer to Section 5.1.1).

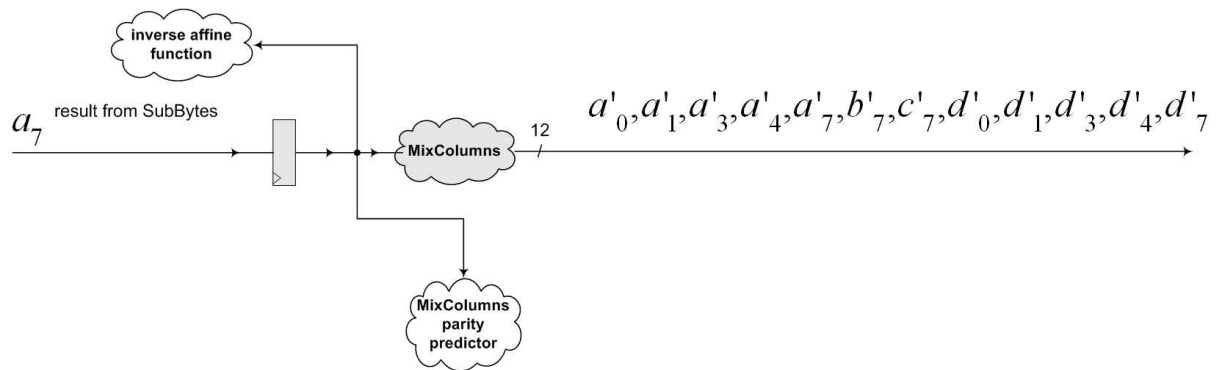


Figure 45 Part of datapath (shown in Figure 32(a)) that is routed in Figure 44

The multiple fanout signal a_7 in the proposed routing shown in Figure 44 is a solution which separates bits that cause potentially undetected single errors in the multiple fanout signal by using extra flip-flops. Each flip-flop in Figure 44 is connected to the input pins of the corresponding LUTs (from Figure 43). The multiple fanout pins (a_7 result from *SubBytes*, P1, and P2) before the flip-flops in Figure 44 are not problematic pins. An error in pin a_7 is detected through the error checker. An error in pin P1 is detected by the parity bits of $a', b', c',$ and d' , since there is an odd number of them (specifically in nets $a'_7(L), b'_7(L), c'_7(L), d'_7(L), a'_0(R), d'_0(R)$ and $a'_1(L), d'_1(L)$ located after pin P1 in Figure 44 there are 3 a' nets, $a'_7(L), a'_0(R), a'_1(L)$, and 3 d' nets, etc). An error in pin P2 is detected by the parity bit of a' (due to an odd number of a' elements).

The parity predictor does not have any multiple fanout signals in the net producing output pins (e.g., P0 is a multiple fanout signal connected to the error checker and none of the output pins in

Figure 44). Therefore, a single error in routing does not affect the parity prediction and output producing circuit simultaneously such that the error is not detected.

The multiple fanout signals after the flip-flops in Figure 44 do not have any pins leading up to undetected single errors either. In the nets after the flip-flops, any single error in routing that affects multiple output bits is detected by at least one parity bit in Figure 44. This is discussed in more detail as follows.

- In net $a_7'(L)b_7'(L)c_7'(L)d_7'(L), a_0'(R), d_0'(R)$, any single error that affects input a_7' of LUT $a_7'(L)b_7'(L)c_7'(L)d_7'(L)$ (this is the LUT L shown in Figure 43 for output bits a_7' , b_7' , c_7' , and d_7') is definitely detected by parity bits of 8-bit elements b' and c' . Other single errors in this net that affect $a_0'(R)$ or $d_0'(R)$ are detected by parity bits of 8-bit elements a' and d' .
- In net $a_1'(L)d_1'(L)$ shown in Figure 44, any single error is detected by parity bits of 8-bit elements a' and d' independently.
- In net $a_3'(L)d_3'(L), a_7'(R)$ shown in Figure 44, any single error is detected by parity bits of 8-bit elements a' or d' .
- In the $a_4'(L)d_4'(L)$ shown in Figure 44, any single error is detected by parity bits of 8-bit elements a' and d' independently.

All the multiple fanout pins investigated above confirm that all single errors are detected in the proposed routing for input net a_7' . The proposed method is then used for the whole *MixColumns* routing for each row of Table 5.3 to Table 5.6. It should be noted that not every row in the tables has single errors not being detected. For instance, in the row of net a_0' of Table 5.3 (shown in Figure 47) there is no multiple fanout pin that can be problematic. If there is a single error in this net that affects the input pin at LUT $a_0'(L)b_0'(L)c_0'(L)d_0'(L)$ (this is the LUT L shown in Figure 46 for output bits a_0' , b_0' , c_0' , and d_0') it is definitely detected by parity bits of 8-bit elements b' and c' . Other single errors in the net are detected by parity bits of a' and d' independently.

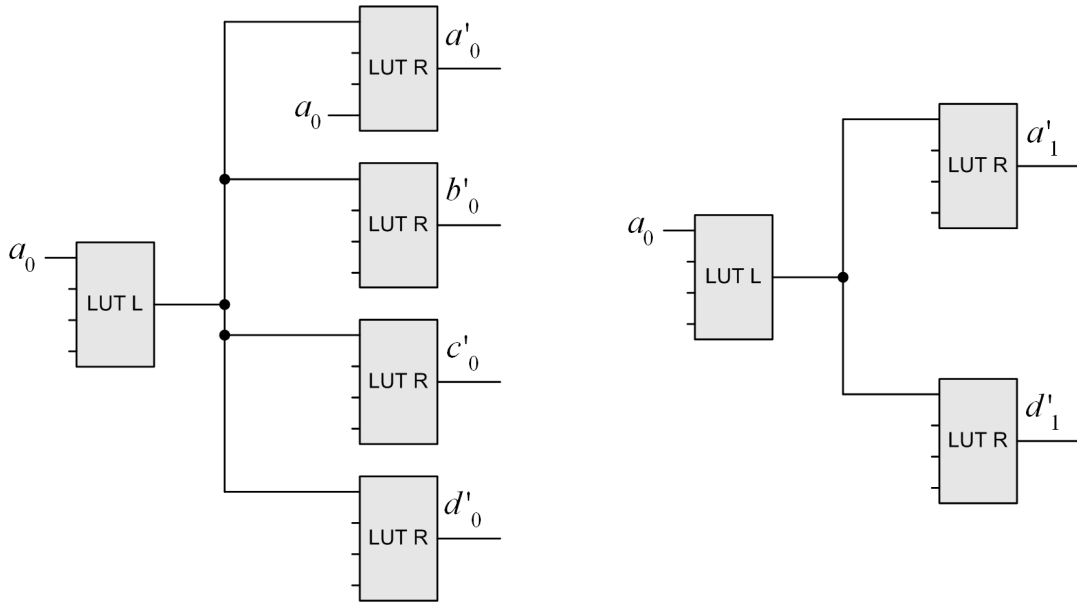


Figure 46 Input a_0 connection to output bits a'_0 , a'_1 , b'_0 , c'_0 , d'_0 , and d'_1 through LUTs

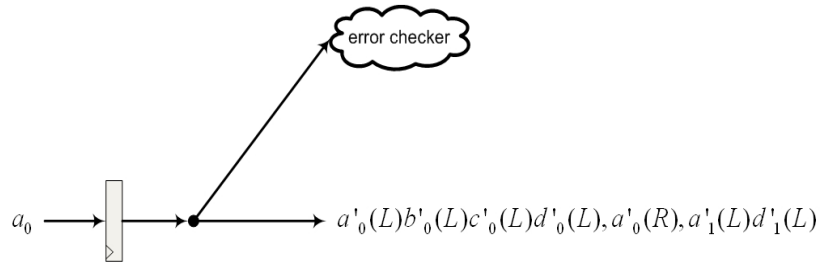


Figure 47 Proposed routing applied to net a_0

The proposed method in routing is shown for the other input bits (a_1 , a_4 , a_5 , a_6) with potential problematic pins of a mitigated in Figure 48. Similar to the routing of a_0 , routing of a_2 and a_3 do not have any multiple fanout signals that lead to single undetected errors.

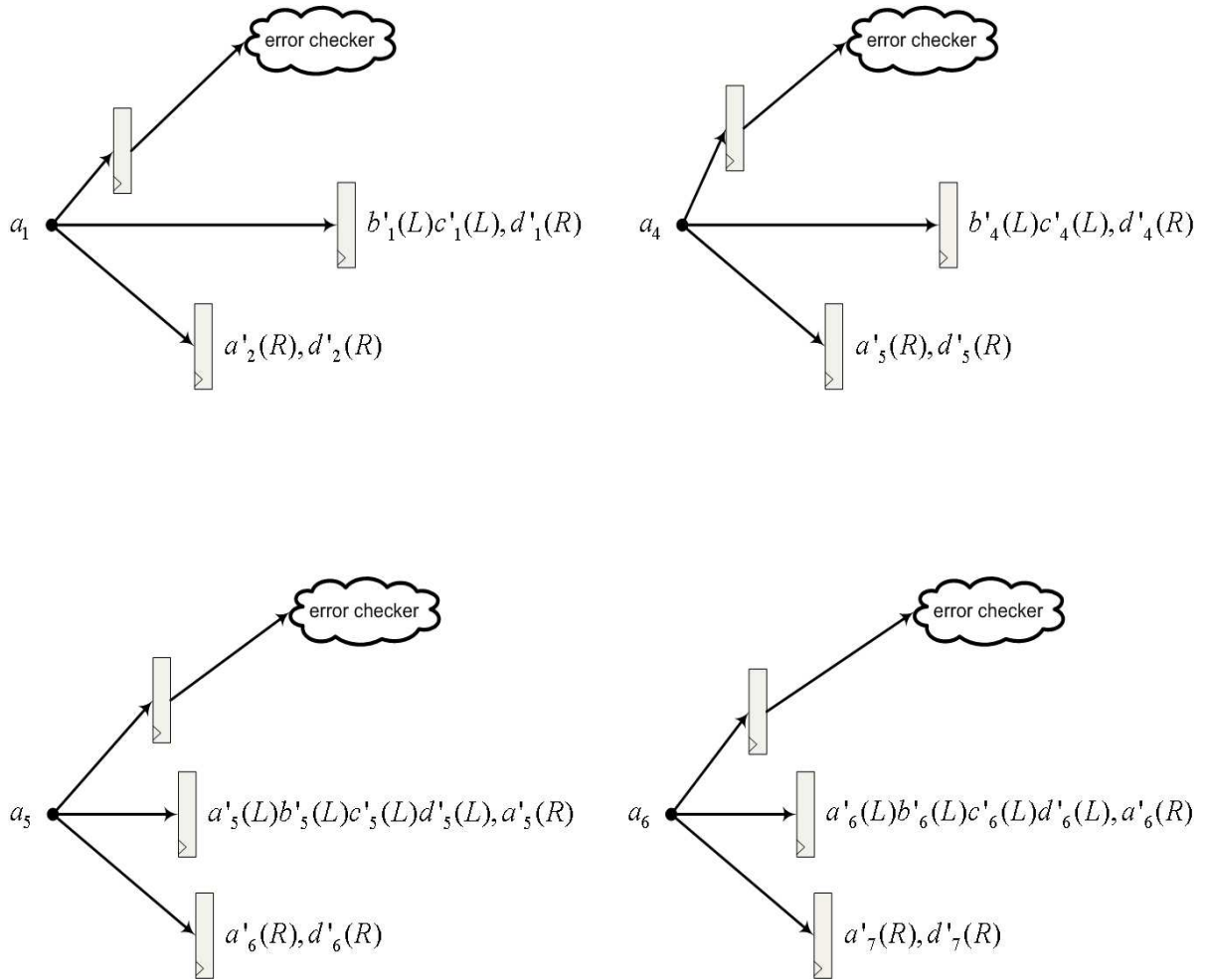


Figure 48 Proposed routing applied to net a_1 , a_4 , a_5 , and a_6

Input nets of 8-bit element c connected to output pins of LUTs in Table 5.4 have similar structure to that of a ; therefore the routings are similar as well. Columns that are similar in both a in Table 5.3 and c in Table 5.4 are as follows: columns (1 and 3), (2 and 4), (3 and 1), and (4 and 2), respectively. The routings of c are shown in Figure 49. Nets c_0 , c_2 , and c_3 do not have any multiple fanout signals that cause undetected single errors.

Table 5.4 Input nets to input pins of LUTs of 8-bit element c

Input	a'	b'	c'	d'
c_0	$a'_0(L)$	$b'_0(L), b'_1(L)$	$c'_0(L,R), c'_1(L)$	$d'_0(L)$
c_1	$a'_1(L)$	$b'_1(R), b'_2(R)$	$c'_2(R)$	$d'_1(L)$
c_2	$a'_2(L)$	$b'_2(L), b'_3(L)$	$c'_2(L,R), c'_3(L)$	$d'_2(L)$
c_3	$a'_3(L)$	$b'_3(R), b'_4(L)$	$c'_4(L)$	$d'_3(L)$
c_4	$a'_4(L)$	$b'_4(R), b'_5(R)$	$c'_5(R)$	$d'_4(L)$
c_5	$a'_5(L)$	$b'_5(L), b'_6(R)$	$c'_5(L,R), c'_6(R)$	$d'_5(L)$
c_6	$a'_6(L)$	$b'_6(L), b'_7(R)$	$c'_6(L,R), c'_7(R)$	$d'_6(L)$
c_7	$a'_7(L)$	$b'_0(R), b'_1(L), b'_3(L), b'_4(L), b'_7(L)$	$c'_0(R), c'_1(L), c'_3(L), c'_4(L), c'_7(L,R)$	$d'_7(L)$

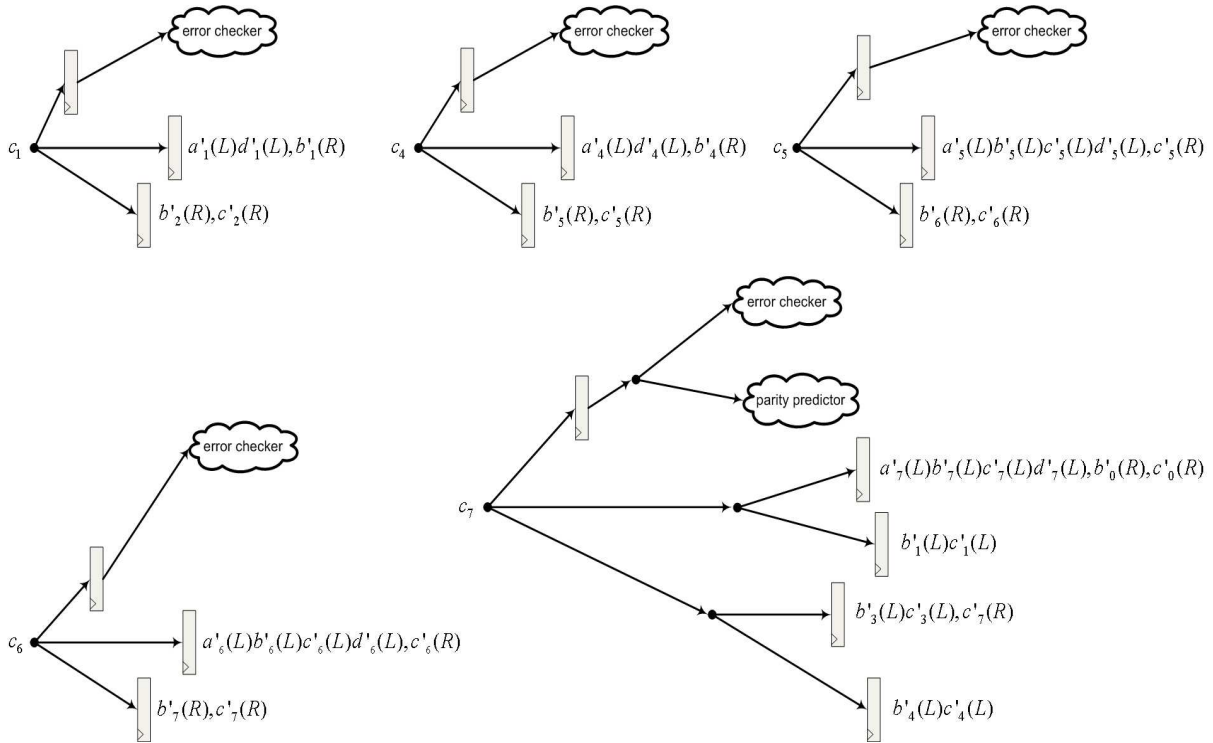


Figure 49 Proposed routing applied to net c_1 , c_4 , c_5 , and c_6

The proposed routing based on Table 5.5 is used for rows of b and is shown in Figure 50. Nets b_1 , b_3 , and b_4 are not problematic. For instance, in net b_1 if there is any error that affects LUT $a_1'(L)d_1'(L)$ it is always detected by the parity bit of 8-bit element d' . Any other error that affects LUTs $a_2'(R)$, $b_2'(R)$, or $c_1'(R)$ is detected by parity bits of a' (if LUT $a_1'(L)d_1'(L)$ is not affected), b' , or c' , respectively.

Table 5.5 Input nets to input pins of LUTs of 8-bit element b

Input	a'	b'	c'	d'
b_0	$a_0'(L), a_1'(R)$	$b_0'(R,L), b_1'(L)$	$c_0'(L)$	$d_0'(L)$
b_1	$a_1'(L), a_2'(R)$	$b_2'(R)$	$c_1'(R)$	$d_1'(L)$
b_2	$a_2'(L), a_3'(R)$	$b_2'(L,R), b_3'(R)$	$c_2'(L)$	$d_2'(L)$
b_3	$a_3'(L), a_4'(R)$	$b_4'(R)$	$c_3'(R)$	$d_3'(L)$
b_4	$a_4'(L), a_5'(R)$	$b_5'(R)$	$c_4'(R)$	$d_4'(L)$
b_5	$a_5'(L), a_6'(R)$	$b_5'(L,R), b_6'(R)$	$c_5'(L)$	$d_5'(L)$
b_6	$a_5'(L), a_6'(R)$	$b_6'(L,R), b_7'(R)$	$c_6'(L)$	$d_6'(L)$
b_7	$a_0'(R), a_1'(R), a_3'(R), a_4'(R), a_7'(L)$	$b_0'(R), b_1'(L), b_3'(L), b_4'(L), b_7'(L,R)$	$c_7'(L)$	$d_7'(L)$

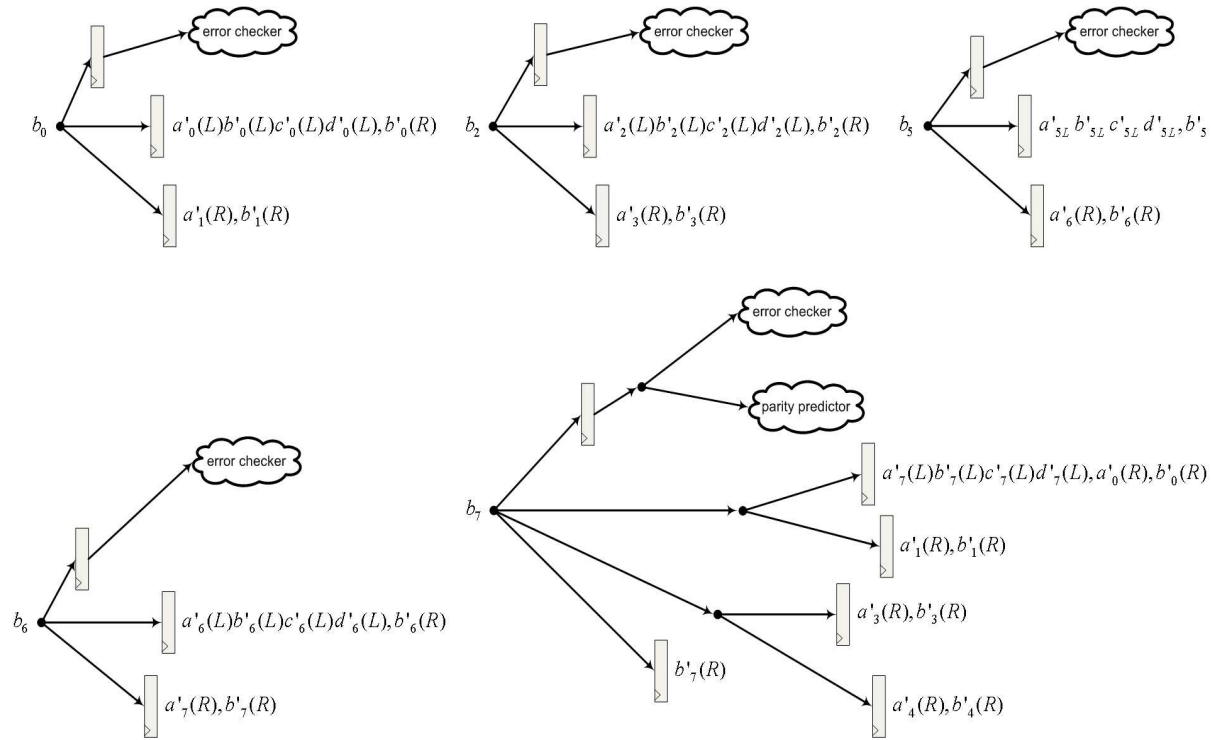


Figure 50 Proposed routing applied to net b_0 , b_2 , b_5 , b_6 , and b_7

Input nets of d to output pins of LUTs in Table 5.6 are similar to that of b therefore the routings are similar. In Table 5.6 and Table 5.5, similar columns of b and d are (1 and 3), (2 and 4), (3 and 1), and (4 and 2), respectively. The routings of d are shown in Figure 51. Nets d_1 , d_3 , and d_4 do not have any multiple fanout signals that lead up to undetected single errors.

Table 5.6 Input nets to input pins of LUTs of 8-bit element d

Input	a'	b'	c'	d'
d_0	$a'_0(L)$	$b'_0(L)$	$c'_0(L), c'_1(R)$	$d'_0(L,R), d'_1(L)$
d_1	$a'_1(R)$	$b'_1(L)$	$c'_1(L), c'_2(R)$	$d'_2(R)$
d_2	$a'_2(L)$	$b'_2(L)$	$c'_2(L), c'_3(R)$	$d'_2(L,R), d'_3(R)$
d_3	$a'_3(R)$	$b'_3(L)$	$c'_3(L), c'_4(R)$	$d'_4(R)$

d_4	$a'_4(R)$	$b'_4(L)$	$c'_4(L), c'_5(R)$	$d'_5(R)$
d_5	$a'_5(L)$	$b'_5(L)$	$c'_5(L), c'_6(R)$	$d'_5(L,R), d'_6(R)$
d_6	$a'_6(L)$	$b'_6(L)$	$c'_5(L), c'_6(R)$	$d'_6(L,R), d'_7(R)$
d_7	$a'_7(L)$	$b'_7(L)$	$c'_0(R), c'_1(R), c'_3(R), c'_4(R), c'_7(L)$	$d'_0(R), d'_1(L), d'_3(L), d'_4(L), d'_7(L,R)$

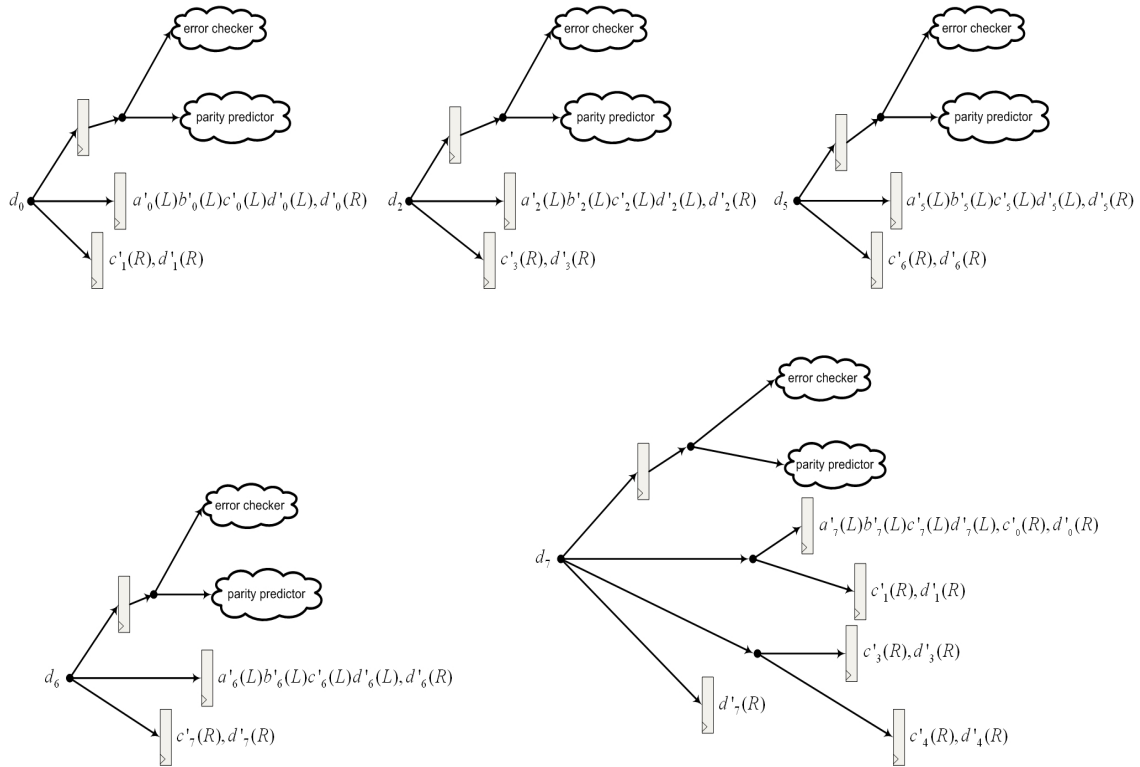


Figure 51 Proposed routing applied to net $d_0, d_2, d_5, d_6,$ and d_7

5.2.3 AddRoundKey Routing and Error Detection

As discussed in Section 5.1.3 logic blocks of *AddRoundKey* are separate. There is 1 LUT associated with each output bit of *AddRoundKey*. Therefore, there are not multiple fanout signals related to bits of an 8-bit output. Consequently, when an error propagates it is not able to affect more than 1 output bit and the parity scheme detects the error.

Since the parity prediction and output producing circuits do not share any logic blocks, there are not any multiple fanout signals related to parity prediction and output bits either. Thus, an error is not

able to affect a parity bit and its 8-bit output at the same time. Consequently, the parity scheme detects all the single errors in the *AddRoundKey* implementation.

5.2.4 Control Circuit and Error Detection

The control circuitry, consisting of 2 state machines shown in Appendix C, generates the select signals to the multiplexers (MUX A, MUX B, MUX C, and MUX D) in the datapath shown in Figure 32(a). In the proposed state machines, shift registers instead of counters are used to keep track of rounds and transformations. An interesting feature of the Virtex-II Pro FPGA used is that a LUT can be set to implement a shift register [119]. This shift register LUT can be of length 1-16. Using this feature keeping track of rounds uses just 1 LUT to implement. Without this feature, N number of flip-flops (where N is the number of rounds in AES) are needed to implement an N -bit shift register.

The control circuit affects the datapath through select signals of the multiplexers (see Figure 32(a)). This could make the parity scheme used in the datapath ineffective in detecting single errors since it can violate the 2 requirements described in Section 5.1.2. Therefore, the control circuit should also be considered in error detection.

In order to ensure a single error in logic blocks does not affect multiple bits in an 8-bit element of the state, the control circuitry is duplicated for each bit of an 8-bit output and shared between 16 elements of the state. Additionally, to make certain that a single error in logic blocks does not affect the parity prediction and output producing circuit simultaneously, the control circuitry is duplicated for the parity prediction. The duplications use relatively small number of resources (1 LUT is minimal resource used for a shift register as discussed above) compared to that of AES. Usage of hardware resources is discussed in Section 6.3. Since duplication is used for the control circuitry all single errors are detected.

5.3 Soft Error Resistant AES for Different Key Sizes and Decryption

In this research, the proposed error detection of AES with key size of 128 bits is expanded to the other versions of AES with key sizes of 192 and 256 bits. The part that is different in the AES algorithm with key sizes 128, 192, and 256 bits is the control circuitry. The shift register LUT supports length of 1-16. The maximum number of rounds is 14 for key size of 256 bits (refer to

Table 4.2). Since the maximum number of rounds does not exceed the maximum length of the shift register LUT, all key sizes use 1 LUT to keep track of rounds.

All versions of AES with different key sizes are implemented in this research. Compared to the AES implementation including datapath and control circuit, the overhead of the control circuitry is about 2.6% and 3% in terms of flip-flops and LUTs, respectively. Overall, the overhead of hardware resources of the control circuitry is not significant although duplication and comparison are used for error detection.

The proposed method can also benefit the AES decryption. The inverse *SubBytes* can directly benefit from the approach proposed for *SubBytes*, since it has a very similar structure that includes the inverse function implemented in a dual ported block memory (refer to 5.1.1). The inverse *MixColumns* transformation includes XOR operations (XOR propagate errors as opposed to other logical operations) similar to *MixColumns*. Therefore, the same 2 requirements in the introduced enhanced parity scheme (refer to 5.1.2 and 5.2.2) can be applied to the inverse *MixColumns* implementation. Inverse *AddRoundKey* and the control circuit also benefit from the error detection proposed for the AES encryption.

5.4 Summary

The proposed error detection technique uses some of the mathematical properties of AES and available hardware resources on FPGA to detect errors in *SubBytes* and the control circuitry implemented. Enhancements to the parity scheme (used for error detection in *MixColumns* and *AddRoundKey*) to increase its error coverage were also proposed in this research.

The inverse function of *SubBytes* is an involution meaning that it is its own inverse. A dual ported block memory exploits this property for error detection. The inverse result is fed back to the second address port of the dual ported block memory. If the data on the second port is different from the value on the first address line, then there have been some errors. This method using the dual ported block memory plus the inverse of affine is used for error detection in *SubBytes*.

The control circuitry uses shift registers to control the select lines of multiplexers. Shift registers were implemented with an interesting feature on the FPGA that implements a shift register of length 1-16 by a single LUT. Since 1 LUT is very small in terms of hardware, duplication is used for the control in this research.

The enhanced parity scheme was used for *MixColumns* and *AddRoundKey* transformations. In order to increase the error coverage of the parity technique, the weaknesses of it on FPGA were first found in this research. The high density of SRAM cells and lack of available information (mapping of configuration bits onto netlist after place and route is proprietary information) make analyzing the effect of faults on an implementation challenging. In order to tackle this problem, the high regularity in the FPGA structure is exploited. The pin fault model was suggested for modeling and analysis when there is a wrong value in an SRAM cell due to soft errors. A small part of the *MixColumns* was tested by simulating single errors to verify this model. Simulating soft errors was achieved by basically flipping 1 bit at a time in the configuration file, then downloading it on FPGA and observing the output bits. This simple and yet accurate enough model was verified. Since FPGA has a very regular structure the result of the verification was expanded for the whole device.

The parity technique covers the errors in datapath registers only. However, errors due to radiation can occur in any SRAM cells forming the combinational logic and routing of a design implemented on FPGA. Propagation of single errors was thoroughly examined in the AES netlist after place and route by using the pin fault model. There are 2 situations when an error can go undetected in the parity scheme. First, if a single error can potentially affect an even number of output bits. Second, errors can go undetected if both output bits and parity bit are affected by a single error. LUTs were designed manually in the netlist to resolve single errors being undetected in combinational logic. Extra flip-flops were used at the register-transfer level to tackle errors being undetected in routing on FPGA in this research.

In the next chapter, experimental results of different implementations on FPGA and comparisons with the proposed design are provided. The coverage of soft errors in all the techniques implemented is also discussed.

Chapter 6

Comparison with Previous Research

In this chapter, the proposed error mitigation approach to AES is compared with previous research. To support a fair quantitative comparison, previously researched architectures were implemented in the same technology as the proposed AES architecture. The error coverage of the proposed technique is compared with previous parity schemes and DMR. Additionally, as the new proposed part of AES, the new implementation of *MixColumns*, is compared with the state of the art. Comparisons of the number of LUTs and flip-flops, block memory size, clock frequency, throughput, and power consumption are provided.

6.1 AES Hardware Design

There have been numerous hardware (FPGA and ASIC) implementations proposed for AES since it was accepted in 2000. Each design typically focuses on one or more constraints i.e. throughput, area, or power and also must target a specific technology. The relevant architectural designs in this research have been implemented in order to provide a fair and exact comparison utilizing the same technology and are discussed in detail in the sections that follow. Moreover, this section reviews some of the previous research from a general hardware design perspective.

There have been mainly 3 different datapath widths (128 bits [98][120][121], 32 bits [115][122], and 8 bits [123][124][125][126, 127]) for the AES architecture. Obviously, wider datapaths aim for higher throughput while narrower datapaths typically target reducing area and power. Hardware

resources are reused in narrow datapaths to reduce area. This lowers the level of parallelism and thus reduces performance (throughput) [121].

Since the AES algorithm has an iterative looping structure (Figure 52(a)), loop unrolling illustrated in Figure 52(b) can be used to increase the level of parallelism for high performance applications. Therefore, throughput can improve by this technique. However, this comes at a price as hardware resources are increased by approximately the number of times the loop is being unrolled [98] [120][121].

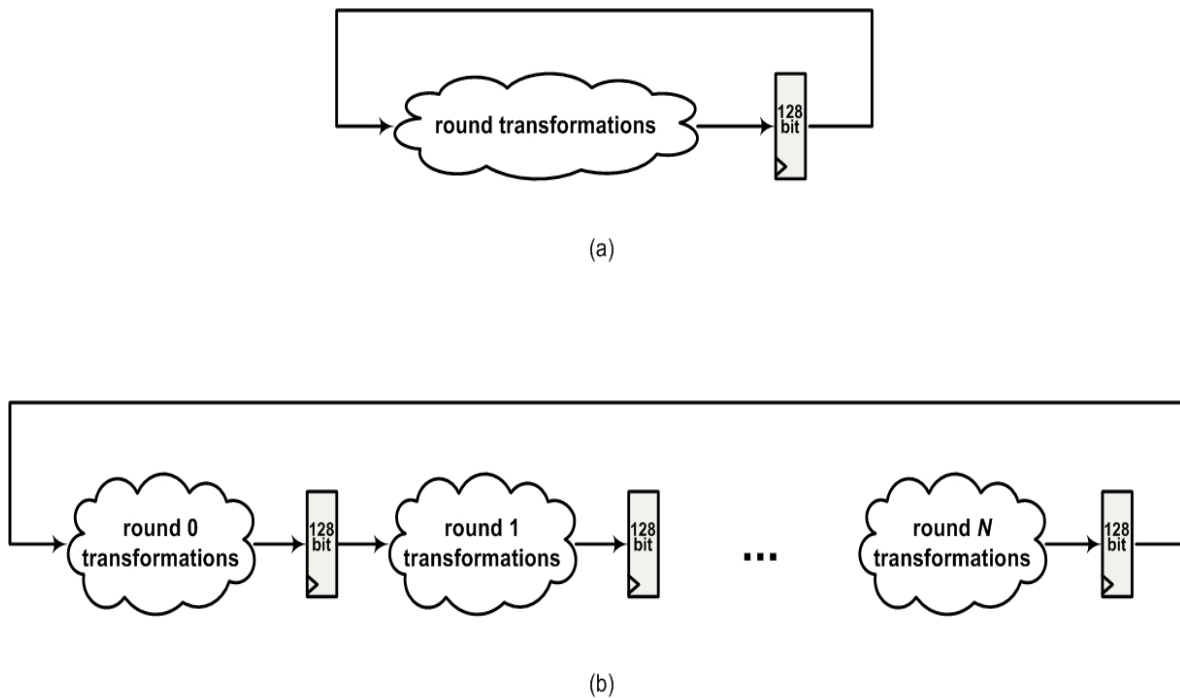


Figure 52 Higher level of parallelism provided by loop unrolling: (a) AES iterative looping structure, (b) N-time loop unrolling [121]

Throughput is computed as shown in Equation 26 where N is the number of times that the loop is unrolled. As seen in Equation 26, loop unrolling can significantly improve throughput. If a design is fully unrolled, then N is equal to the number of rounds and the maximum throughput is achieved at the cost of approximately N times the area. Furthermore, round transformations can be pipelined to increase the clock frequency [98] [120][121].

$$throughput = \frac{128 \cdot f_{clk} \cdot N}{number\ of\ rounds} \quad (26)$$

Experimental results of some of the previous research are presented in Table 6.1 and briefly discussed without going into specific details of each architecture.

Table 6.1 AES previous implementations

	Technology	Area	Block memories	Clock frequency in MHz	Throughput in Gbit/s	Power consumption
Good et al. [98] encryption/decryption	Spartan-III XC3S4000-5	20720 slices	-	240.9	30.83	-
Good et al. [98] encryption/decryption	Virtex-II XC2V8000-5	31674 slices	-	222.8	28.52	-
Hodjat et al. [120] encryption	Virtex-II Pro	8285 LUTs	84	168.3	21.54	-
Hodjat et al. [120], <i>SubBytes</i> in composite fields encryption	Virtex-II Pro	22358 LUTs	-	168.3	21.54	-
Zambreno et al. [121] encryption	Xilinx XC2V4000	16938 slices	-	184.1	23.57	-
Mozaffari et al.[113] encryption	Virtex-5	9806 slices	-	482.998	61.8	-
Mozaffari et al. [114], parity for <i>SubBytes</i> encryption	Virtex-II Pro	9405 slices	-	60.8	-	-
Yu et al. [115] encryption/decryption	0.18 μ m CMOS	10.9k gates	-	-	0.112	-

Chapter 6: Comparison with Previous Research

Good et al. [123] encryption	0.13 μ m CMOS	5.5k NAND gates	-	100	-	692 μ W at 0.75V
Chang et al. [124] encryption/decryption	Spartan-II	200 slices	2	38.5	0.031	-
Haghighizadeh [125] encryption/decryption	0.18 μ m CMOS	5.5k NAND gates	-	128	0.102	49 μ W
Hamalainen et al. [127] encryption	0.13 μ m CMOS	3.1k NAND gates	-	152	0.121	37 μ W
Dalmisli et al. [126] encryption	Spartan-III	294 slices	-	142.8	0.0147	43mW
Dalmisli et al. [126] encryption	Spartan-III	299 slices	-	68.9	0.013	26mW

Previous researchers in [98] [120][121] aimed at high performance. Design [120] (row 5 in Table 6.1) uses composite field $GF((2^4)^2)$ for the *SubBytes* transformation while *SubBytes* in row 4 is memory based.

The parity for S-box was proposed in [113] for error detection. Distributed RAMs were used for implementing *SubBytes*. Their results seem to be after XST synthesis and not place and route. In [114], the composite field *SubBytes* and its inverse were divided into blocks and the predicted parities of these blocks were computed. Optimum solution for the composite field in terms of overhead was found through exhaustive search.

Researchers in [115] used a 32-bit datapath for a compact ASIC implementation of AES. To provide error detection, S-boxes were duplicated and parity bits were used for other AES transformations.

Research [123][124][125][126, 127] proposed different architectures for the 8-bit datapath aiming at reducing area and power. Results have been reported on various technologies including CMOS and FPGAs.

6.2 Experimental Results of *MixColumns*

Previous equations of *MixColumns* presented in Section 4.3.2 are implemented in FPGA technologies for a fair comparison with the proposed design (refer to 5.1.2). As discussed in 4.3.2, the *MixColumns* in [98, 102], [112], and [108, 109] used Equation 20, Equation 12, and Equation 21, respectively.

The experimental results use the Xilinx Virtex-4 FPGA (target device: xc4vlx15-10ff668) as well as Altera Cyclone (target device: EP1C3T144C6). These FPGAs use 4-input LUTs as function generators. Manual instantiation was used to prevent further modification by the tool (due to optimization) and the netlist was verified after place and route.

The number of LUTs and the corresponding LUT savings for the *MixColumns* implementations are shown in Table 6.2. The input and output signals are 32-bit columns of the state. As observed in Table 6.2, improvements are more significant in case of the Xilinx synthesizer than Altera synthesizer, since they use different mapping algorithms. The LUT savings on Virtex-4 and Cyclone FPGAs are at least 20% and 10%, respectively.

Table 6.2 Experimental results of *MixColumns* implementations on FPGA

<i>MixColumns</i> implementations	# of LUTs on Virtex-4	% of LUTs of LUT savings Virtex-4	# of LUTs on Cyclone	% of LUT savings on Cyclone
Proposed <i>MixColumns</i>	43	-	43	-
<i>MixColumns</i> in [98, 102]	56	23.21	51	15.68
<i>MixColumns</i> in [112]	54	20.37	48	10.41
<i>MixColumns</i> in [108, 109]	55	21.28	51	15.68

6.3 Experimental Results of AES with Soft Error Mitigation

As illustrated in Figure 53, previous research in error detection using the parity scheme in AES [76-78] focuses mainly on covering errors occurring in datapath registers. The Hamming code with 8 data bits and 4 check bits (12-bit codeword) for single error correction in AES was suggested in [75].

Chapter 6: Comparison with Previous Research

As shown in Figure 54, the Hamming code in [75] also considers errors happening only in the datapath registers.

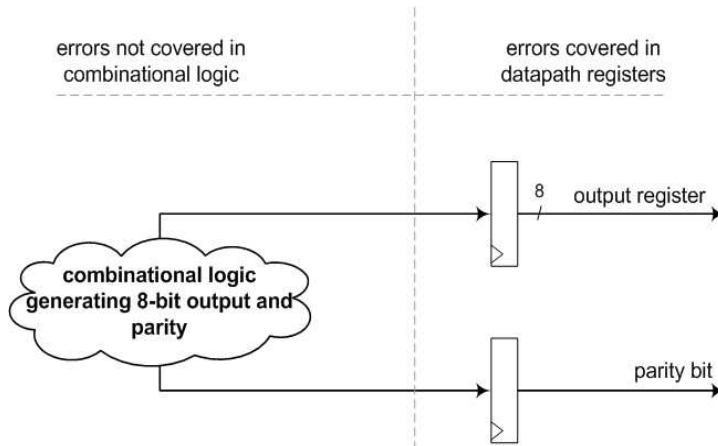


Figure 53 Error coverage of single errors using parity in previous research

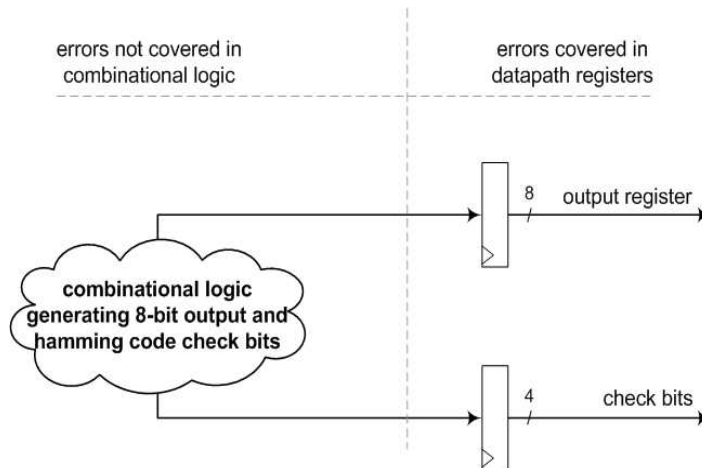


Figure 54 Error coverage of errors using Hamming code in previous research

These techniques do not provide error coverage for other logic and routing elements. They do not cover any errors happening in the control circuitry either. SEUs affect all the hardware resources (logic elements such as LUTs and SRAM switches controlling routing) on an FPGA. SETs can also generate glitches in the combinational logic and routing of a design implemented. Therefore, these methods covering errors in datapath registers are not sufficient for an implementation on FPGA.

However, they provide sufficient error coverage caused by SEUs for the datapath of an ASIC implementation, since SEUs cause errors only in flip-flops on ASIC (refer to Section 3.4). Other hardware modular redundancy techniques such as DMR and TMR (refer to Section 3.2) provide coverage for multiple errors but they are expensive and the likelihood of having multiple soft errors is extremely low.

It should also be noted that the concurrent error correction capability of techniques such as the Hamming code and TMR can be affected on FPGA when SEUs occur. These error correcting techniques can potentially lose their effectiveness (correct functionality), since an SEU can affect the functionality of the correction circuitry. Additionally, SEUs keep accumulating on FPGA until it is reconfigured. Due to unknown outcome of concurrent correcting techniques in previous research, re-computation of the last input after reconfiguration of FPGA is suggested in this research. A reconfiguration is done when an error is detected (refer to Appendix B).

SEU and SET occurrences are random in time and space. The likelihood of multiple SEUs or SETs happening in 1 clock cycle is extremely low. Among detection techniques, the parity scheme is the lowest cost (in terms of hardware resources) and potentially adequate choice that matches the nature of SEU and SET occurrences.

In this section, the experimental results show usage of hardware resources, timing information, power consumption, and detection and correction capabilities in some of the relevant techniques in previous research and the proposed design. All these techniques have been implemented on the same FPGA (Virtex-II Pro device: xc2vp100-6ff1704) to provide a fair basis for comparing the results. There have been different approaches for the *SubBytes* in composite fields [98-102][103, 104]. Since the implementation of *SubBytes* [102] in composite fields in this research is not heavily pipelined to achieve the highest frequency, it is not expected to have significantly different results from approaches other than [102].

Table 6.3 Results of different designs of parity scheme in AES

Designs using parity scheme in AES	# of LUTs	# of flip-flops	Utilization of block memories	Clock frequency in MHz	Throughput in Gbit/s	Total power consumption in mW
Proposed AES with enhanced parity scheme	1188	1126	4%	186.7	2.39	865.60
Composite fields AES in [102] with parity (<i>SubBytes</i> parity in [77])	2363	662	1%	145.9	1.86	4136.22
Memory based AES [76]	665	516	8%	215.1	2.75	641.48

Researchers in [77] provide the parity prediction formula, shown in Equation 27, for an 8-bit element b input to *SubBytes*. In this equation, symbol δ^{-1}_{input} is the input to function δ^{-1} for isomorphic mapping (refer to Figure 29). The indexes indicate bit position in an 8-bit δ^{-1}_{input} in Equation 27 (for instance, $\delta^{-1}_{input_0}$ is bit 0). This parity prediction formula can be used in the *SubBytes* that is implemented using composite fields. The *SubBytes* in composite fields suggested in [102] and parity predictions in [77] and Equation 27 are implemented on the FPGA in VHDL. The results are shown in Table 6.3.

$$parity_SuBytes(b) = \delta^{-1}_{input_0} + \delta^{-1}_{input_1} + \delta^{-1}_{input_2} + \delta^{-1}_{input_4} + \delta^{-1}_{input_6} \quad (27)$$

It should be noted these results, except for the throughput and clock frequency, are overly optimistic for the composite field implementation of AES encryption since this design can be further pipelined thus adding a large number of flip-flops to the 128-bit datapath. As expected, the implementation in composite fields uses the largest number of LUTs in order to achieve a high throughput if it is heavily pipelined.

The parity coding for error detection is used in [76] with a memory based implementation of *SubBytes*. In a memory based *SubBytes*, an S-box alone is implemented by a 256x8-bit block memory for each 8-bit element of the state. In order to generate the predicted parity bit, a parity bit is added for each 8-bit element in [76]. The parity bit is also added to the address line of the memory to detect single errors at the output of *AddRoundKey* (input to *SubBytes*). If the output of the *AddRoundKey* is wrong a deliberate wrong content (e.g., 000000001) is stored in the memory. Overall, a 519x9-bit block memory is used for *SubBytes* with parity coding in [76].

Another part of the block memory that is partially covered against errors in [76] is the address decoder. A single error in the decoder causes the wrong memory location to be accessed. However, the content of this address has a valid parity bit. In order to mitigate this, adding a separate 256x1-bit block memory was suggested in [76]. This stores an extra set of parity bits for 8-bit *SubBytes* values. Each parity bit from the 519x9-bit memory is compared to the parity bit from the 256x1-bit block memory to find an error in the decoder. The detection in this method provides error coverage of 50% for single errors in the block memory decoder, since there is still a probability of 0.5 that both parity bits match while the wrong address is accessed.

The *MixColumns* parity prediction used for all the designs is shown in Equation 25 [76]. The parity prediction of *AddRoundKey* is the XOR of the input parity and key parity.

The proposed design of AES with the enhanced parity scheme has about the same number of LUTs and flip-flops. The balance between the number of LUTs and flip-flops is still reasonable in terms resource utilization since there is 1 flip-flop for each LUT on the FPGA. The number of flip-flops is larger than other implementations since data (output of *AddRoundKey*) need to be delayed for the equality comparator shown in Figure 32(a) and the routing is controlled by flip-flops as well. The largest number of LUTs is in AES in composite fields [102] with *SubBytes* parity [77] and *MixColumns* parity in Equation 25 (row 2 in Table 6.3); it has about double the number of LUTs than the proposed AES with the enhanced parity scheme. Implementation of [76] has the highest block memory utilization on the FPGA due to extra block memories needed to store the parity of *SubBytes*.

In the proposed AES with the enhanced parity scheme, there is about 13% reduction in the clock frequency and throughput compared to implementation of [76]. The post place and route static timing analysis shows that the second port of the dual ported block memory to the output of the 8-bit equal comparator is the critical path.

In order to estimate power consumption, the same random input data generated by function UNIFORM() in VHDL at the clock frequency of 100 MHz is used in all the implementations. Static power consumption is constant 204.38mW for the FPGA. Implementation of [76] has the lowest dynamic and total power consumption since it uses the largest number of block memories instead of LUTs for computations.

It should be noted that the overhead of the control circuitry compared to the overall AES implementation is about 2.6% and 3% in terms of flip-flops and LUTs, respectively. Although duplication and comparison are used for error detection, the overhead of hardware resources of the control circuitry is not significant.

The error coverage of SEUs for different parity designs are shown in Table 6.4. In the proposed design using the enhanced parity scheme, the propagation of an error from logic and routing resources to the output bits has been thoroughly investigated (refer to Chapter 5). In this research, error detection is incorporated in the design process rather than being added as an after part. It is ensured that a single error in logic or routing resources does not affect an even number of bits in an 8-bit register of the state. It is also made certain that a single error in logic or routing resources does not affect the parity prediction and output producing circuit simultaneously such that the error is not detected. The control circuitry using minimal hardware resource (1 LUT for one-hot encoding) is duplicated for error detection. Thus, single errors due to SEUs in combinational and sequential elements of datapath, control circuitry, and routing are detected.

Through analyzing and classifying the errors caused by SEUs in research [17], it is concluded that about 78% to 84.8% of the failures are due to SEUs in routing while the remaining approximately 20% are due to upsets in logic excluding registers (researchers in [17] focused on configuration bits of the FPGA). This indicates the importance of mitigating SEUs in routing.

Table 6.4 Error coverage of single SEUs on FPGA

	Errors in combinational logic of datapath due to SEUs	Errors in routing due to SEUs	Errors in flip-flops of datapath due to SEUs	Errors in combinational/sequential logic and routing of control circuitry due to SEUs
Proposed AES with enhanced parity scheme	Detected	Detected	Detected	Detected
Composite fields AES in [102] with parity (<i>SubBytes</i> parity in [77])	Not detected	Not detected	Detected	Not detected
Memory based AES [76] with parity	Not detected	Not detected	Detected	Not detected

An SET, being equivalent to a glitch, can propagate through the combinational logic and get stored in result flip-flops. As opposed to SEUs, SETs do not change the functionality of the combinational logic on FPGA. They can potentially change the output flip-flops for 1 clock cycle. Since glitches in the combinational logic or routing are not considered in previous research, using the previous parity schemes, an SET can potentially propagate through to flip-flops of the output and parity bits. These single errors due to SETs are not detected in previous research.

Propagation of single errors in combinational logic as well as routing is thoroughly investigated in the proposed design (refer to Chapter 5). It is ensured that an SET (glitch) does not affect an even number of output flip-flops or the parity and output flip-flops simultaneously such that an error caused by an SET can escape undetected. The routing is controlled by using extra flip-flops at the register-transfer level, since it is not possible to change the routing at a fine-grained level after place and route. Therefore, single errors caused by SETs are detected in sequential logic of datapath and control circuitry in the proposed design as is shown in Table 6.5.

Table 6.5 Error coverage of single SETs on FPGA

	Errors in sequential logic of datapath due to SETs	Errors in sequential logic of control circuitry due to SETs
Proposed AES with enhanced parity scheme	Detected	Detected
Composite fields AES in [102] with parity (<i>SubBytes</i> parity in [77])	Not detected	Not detected
Memory based AES [76] with parity	50% only in the block memory (S-box) decoder	Not detected

Overall, the proposed technique drastically expands the soft error coverage of parity coding to both logic and routing resources of datapath and control circuitry on an FPGA.

The DMR method is also implemented for further comparisons with the proposed approach. Table 6.6 shows the experimental results. In order to provide the redundancy in DMR, all the resources (i.e. I/O pins, logic resources, and block memories) have to be duplicated. It should be noted that the S-box is used in DMR (*SubBytes* is memory based in DMR) since the inverse and affine functions do not need to be separated, as opposed to the proposed AES with error detection.

Table 6.6 Experimental results of DMR and proposed enhanced parity approach

	I/O pins	# of LUTs	# of flip-flops	Utilization of block memory	Clock frequency in MHz	Throughput in Gbit/s	Total power consumption in mW
Proposed AES with enhanced parity scheme	339	1188	1126	4%	186.7	2.39	865.60
DMR	451	1233	865	9%	177.3	2.26	846.78

Compared to the proposed technique, there is about 33% and 3.8% increase in I/O pins and LUTs, respectively, in DMR. Block memory utilization in DMR is more than twice compared to that of the proposed AES with enhanced parity scheme. The clock frequency and throughput is about 5% higher

in the proposed technique than DMR. There is also slight decrease of about 2% in the total power consumption of DMR than that of the proposed technique, since manual instantiation of LUTs and flip-flops slightly increase the power consumed by logic and routing.

The proposed design has a relatively large number of flip-flops since the output of *AddRoundKey* needs to be delayed and the routing is also controlled by flip-flops to expend the error coverage of the parity scheme. One solution to enhancing routing for the parity scheme without using flip-flops could be defining new implementation constraints to the tool in the future work. The enhancement in routing refers to removing the pins that lead up to undetected single errors (refer to Chapter 5). Implementation constraints are instructions that are given to software tools to direct different steps such as placement and routing in the design flow. For instance, locations constraints define the absolute or relative location of a design element on FPGA for the placement tool in Xilinx. In general, implementation constraints are placed in a constraint file or in the HDL code. Therefore, specifying constraints at the high level does not make the design process tedious. The subject of adding new constraints in the tool to control the pins along multiple fanout signals can be further investigated in future work.

DMR provides multiple error coverage in all parts of design implemented but this is not necessary for soft errors, since the likelihood of multiple errors in 1 clock cycle in an implementation is extremely low. Therefore, the expensive feature of multiple error coverage is not needed to provide reliability. However, it is important to provide complete error coverage for single errors since the likelihood of single errors is high. This is provided in the proposed design using the enhanced parity error detection.

In this research, round keys are stored in a block memory. The operations in key expansion generating round keys are *SubBytes* and shifts and XORs (key expansion pseudo code is given Appendix G). If round keys are to be computed they can use the similar error detection technique used for the *SubBytes*. In the control circuitry of key expansion, the same technique implementing shifts by LUTs and using duplication for the minimal hardware can be applied (refer to Section 5.2.4).

NIST approved modes such as CFB, OFB, CTR (confidentiality modes), CMAC (authentication mode), and CCM (authenticated encryption mode) directly benefit from the proposed AES with error detection. In the other authenticated encryption mode, GCM, multiplication in GHASH needs to be investigated against soft errors. This multiplication is in the Galois field of 2^{128} elements. The parity

technique and duplication should be compared in terms of costs for the right selection. There are various techniques for Galois field multiplication investigated in [106].

6.4 Summary

The proposed AES with enhanced parity scheme was compared with other implementations of AES using the parity scheme. Overall, in terms of LUT and block memory utilization the proposed technique does not exceed the composite fields and memory based implementations, respectively. Compared to composite fields [102][77] and memory based [76] implementations of AES, the proposed design uses about half the number of LUTs and block memories, respectively. Compared to the memory based AES, there is about 13% reduction in the clock frequency and throughput. The memory based AES had about 25% less power consumption than the proposed design.

DMR was also implemented for further comparisons. Compared to the proposed design, the DMR approach has about 33%, 3.8%, and 100% increase in I/O pins, LUTs, and block memories, respectively. The clock frequency is about 5% higher in the proposed design than DMR. There is a slight decrease of about 2% in power consumption of DMR than the proposed implementation.

The most noticeable drawback in the proposed design experimental results was the relatively large number of flip-flops overall. This is due to delaying of the output of *AddRoundKey* to be synchronized with the dual ported block memory. Additionally, the routing has been controlled by flip-flops at the register-transfer level. However, since the number of flip-flops is not greater than that of LUTs this is still not an unbalanced design practice.

Compared to previous AES designs using the parity scheme for error detection, the proposed technique significantly expands the soft error coverage from datapath registers to both logic and routing resources of datapath and control circuitry on FPGA. DMR provides multiple error coverage in all parts of design implemented. However, this is not necessary in the case of soft errors since the likelihood of multiple errors in 1 clock cycle is exceedingly low. Therefore, the proposed AES with enhanced parity scheme provides a low cost adequate method.

Chapter 7

Discussion and Conclusion

This research presented a new design for reliability of the symmetric-key algorithm AES in FPGAs. The AES algorithm is highly sensitive to errors by nature. For instance, a single-bit flip in the early rounds of AES encryption is expected to in 50% erroneous bits in the output [76]. This indicates a good diffusion in the AES algorithm. Diffusion is a desirable property for a strong cryptographic algorithm; however it becomes a critical issue in error propagation and especially reliability of AES in the FPGA. Reliable implementations of the error sensitive AES in the FPGA is important since FPGAs are prone to soft errors caused by radiation. Energetic particles hitting a device can flip SRAM cells controlling all aspects of the implementation in a dense SRAM-based FPGA. For instance, a Virtex-II Pro FPGA contains 34,292,768 SRAM cells [2]. Different error detection techniques based on properties of the circuit and AES transformations were used to provide adequate reliability at the lowest possible cost. Dual-ported block memory for *SubBytes*, duplication for the control circuitry, and the proposed enhanced parity technique for *MixColumns* were used. In this research, propagation of single errors was investigated in the placed and routed netlist. Weaknesses of the previous parity techniques were researched. Flip-flops at the register-transfer level were introduced to resolve undetected single errors in the routing. LUTs were designed for *MixColumns* with minimum number of LUTs to prevent undetected single errors in the combinational logic.

The peculiar effects of soft errors in a design implemented on SRAM-based FPGA were investigated. Furthermore, the errors caused by these faults in ASIC and FPGA were compared. Logic blocks in the combinational logic as well as routing of the FPGA are controlled thoroughly by SRAM cells. Therefore, SEUs can affect the combinational logic and routing on the FPGA and are not eliminated until a reconfiguration is done. This effect does not happen in an ASIC. Since the combinational logic and routing on ASIC do not have any storage elements, SEUs can only affect flip-flops. Previous research [75][76][77] on mitigating soft errors in the AES implementations (primarily focusing on datapath registers) did not consider all aspects of soft errors in the FPGA.

A dual ported block memory and the inverse function of *SubBytes* were used for error detection in this transformation of AES. Unlike previous research [76][77] using a parity scheme for every AES transformation, mathematical properties of *SubBytes* and dual ported block memory were used to expand soft error coverage from datapath registers to combinational logic and routing in the FPGA.

The control circuitry uses shift registers (one-hot encoding) to control the select lines of multiplexers. Shift registers were implemented by a single LUT. Since 1 LUT is the minimal hardware resource, duplication was used for the control circuitry to provide error detection in this research. Error detection for the control circuitry was not considered in previous research [75][76][77].

Since soft errors threatening the reliability of FPGA implementations occur randomly and the likelihood of multiple errors in 1 clock cycle is exceedingly low, the low cost parity scheme is a suitable error detection technique. Previously, a parity scheme was used for error detection in registers of datapath and block memories in AES implementation. However, the parity schemes in previous research [76][77] did not cover errors occurring in the logic blocks, routing, and control circuitry.

In this research, novel enhancements to a parity scheme were introduced and applied to the *MixColumns* transformation. Unlike previous research, the enhancement proposed at the register-transfer level increases the error coverage of a parity scheme from the datapath flip-flops to logic blocks, and routing. This is important since not only flip-flops can be target of soft errors but also SRAM cells building logic blocks and routing can be affected.

Soft errors on FPGA were modeled using the pin fault model, verified by simulating errors in the implementation. In order to simulate SEUs, bits were flipped in the configuration file one at a time.

Then the configuration file simulating SEUs was downloaded on the FPGA. Since the mapping of the netlist after place and route in the configuration file is proprietary information, the challenge was to find the configuration bits that were related to a specific net. To overcome this challenge, the net was removed from the netlist manually after place and route and the modified configuration file was generated. The original and modified configuration files were compared to find the bits related to the specific net that was removed. Then each bit of the net was flipped one at a time to produce the configuration file simulating SEUs. Finally, the output bits of this implementation containing a soft error were observed. The pin fault model was confirmed by simulating SEUs.

The pin fault model was used to find the weaknesses of a parity scheme. Weaknesses include 2 cases: 1) a single error affects an even number of bits in an 8-bit element of the state and 2) a single error affects the parity prediction and output producing circuit simultaneously such that the error is not detected. These weaknesses were investigated in 2 separate phases: logic blocks and routing. In logic blocks, to avoid the 2 cases, LUTs were examined thoroughly. LUTs were designed manually if found to contain any weaknesses that were the result of LUT sharing between output bits having the same parity bit. Therefore, sharing of LUTs was modified to eliminate the weaknesses (refer to Section 5.1.2).

In the routing phase, the weaknesses caused by multiple fanout signals connected to output bits having the same parity bit were investigated. Manually changing of multiple fanout pins in a net at the fine-grained level is not possible with the available FPGA tools after place and route. Therefore, modification of the routing became a challenge. To overcome this obstacle, extra flip-flops were inserted at the register-transfer level to manually change the way pins were formed along a multiple fanout net. This modification was verified by simulating SEUs (flipping 1 bit at a time in the configuration file) for a small part of *MixColumns*. Then this was expanded for all the output bits protected by the parity bit. The proposed enhanced parity scheme in this research expands the error coverage to combinational logic and routing.

The insufficiency of known error correction techniques such as TMR and Hamming code [75] on FPGAs was pointed out and self reconfiguration was suggested instead. This was designed and implemented as a system on chip that communicates with the host PC to trigger reconfiguration of FPGA. The hardware part of this system was implemented using the PowerPC 405 processor and the UART IP core and implementing master, slave, and interrupt attachments in the proposed AES module with error detection. In case of an error, the AES module interrupts the PowerPC 405

processor that sends the error message to the host PC through the serial communication. Then the host PC interrupt handler performs JTAG boundary scan configuration on the FPGA. The software part of this system on chip was programming device drivers for the PowerPC 405 processor and host PC.

To provide fair comparisons, different approaches providing error detection for AES were implemented on the same platform. Unlike previous AES designs using a parity scheme for error detection, the proposed technique significantly expands the soft error coverage from datapath registers to both logic and routing resources of datapath and control circuitry on FPGA. When compared with the AES in composite fields [102] with the parity scheme [77], the proposed design uses about half the number of LUTs. The other implementation of AES [76] that is memory based utilizes twice the block memories used in the proposed implementation. The routing of the dual ported block memory is the critical path in design of this research. There is about 13% reduction in the clock frequency and throughput compared to [76]. Since the memory based AES relies heavily on block memories to provide various computations it has about 25% less power consumption than the proposed design.

In addition to the parity schemes, DMR was implemented for further comparisons. The DMR implementation compared to the proposed design has about 33%, 3.8%, and 100% increase in I/O pins, LUTs, and block memories, respectively. The clock frequency and throughput is approximately 5% higher in the proposed design than DMR. Since the DMR approach relies more on block memories (*SubBytes* as opposed to inverse is computed completely using block memories), there is a slight decrease of about 2% in power consumption of DMR compared to the proposed implementation. DMR detects multiple errors having extremely low likelihood in case of soft errors. Unlike DMR, the proposed AES with enhanced parity scheme provides low cost adequate method that covers single errors.

The number of flip-flops in the proposed implementation is the largest amongst all. The flip-flops are used to delay the input of the dual ported block memory and also control routing at the register-transfer level. Nevertheless, the number of flip-flops is not greater than that of LUTs, and thus this is still a balanced design practice.

The main goal of error detection was reliability in this research. However, the proposed technique may also be applied against cryptanalysis of AES with certain assumptions. Multiple errors are detected in the *SubBytes* transformation. Therefore, the proposed error detection technique may

thwart faults attacks where multiple attacks are injected during *SubBytes*. Detection of a single or an odd number of errors is provided in *MixColumns* and *AddRoundKey* transformations. Hence in cases where the number of errors in these two AES transformations is not always even, the proposed method may be applied. Thus attacks such as [20] (a fault is injected after the initial *AddRoundKey*) or the first attack in [30] (a fault injected at the beginning of the final round) may be thwarted by the proposed technique.

Reliability of the AES implementation is a critical issue especially in large scale systems using multiple FPGAs and space applications. In these types of applications, soft error resistant design is an important concern. For the first time this research has proposed a heterogeneous error detection approach utilizing properties of the circuit and functionality in order to provide adequate reliability at the lowest possible cost. Unlike previous research, architectural redesign at the register-transfer level was introduced to resolve undetected single errors in both the routing and the combinational logic. This research is important for providing soft error resistant design for FPGAs in applications which are crucial for many secure space and terrestrial applications.

7.1 Future Work

Future work in this research will mainly proceed in 3 different directions: (a) adding enhanced features to software tools, (b) exploiting the proposed technique on ASIC for SETs (glitches that potentially cause errors), and (c) examining other reconfiguration techniques. Propagation of errors in routing from an input net to output bits after place and route was investigated manually in this research. This can be incorporated and automated as a feature in the software tools. Therefore, the process of finding the pins that lead up to undetected single errors can be speeded up. An error at these pins can potentially affect an even number of output bits or both the parity bit and output bits, and thus go undetected in the parity scheme. The process of simulating SEUs in the combinational logic or routing of an implementation can also be provided as another feature. Therefore, the designer would be able to see the effects of soft errors on outputs.

The way pins are modified in multiple fanout nets is by manually inserting extra flip-flops. The modifications basically include grouping of outputs bits connected to a pin in a multiple fanout net by using the extra flip-flops. This results in an increase in the number of flip-flops. In order to improve this, software tools should allow fine-grained control and modification over pins in multiple fanout nets in the design flow. Therefore, the numbers of flip-flops used could be reduced.

Implementations in ASICs do not suffer from SEUs in the combinational logic or routing. However, there is still the issue of SETs propagating through logic paths and getting stored in registers. Since clock frequencies continue to increase, this problem remains important as technology advances. The proposed technique can be further used in AES implemented in ASICs to mitigate SETs.

In this research, JTAG boundary scan configuration mode was used in the FPGA. Partial configuration that involves determining which frames (a frame is the smallest segment of configuration memory space) to reconfigure can also be investigated to examine possible delay reductions. Remote reconfiguration that includes the Ethernet port, SDRAM, and external Flash is also an interesting topic to be further studied.

Appendix A

Glossary of Acronyms

<i>AddRoundKey</i>	add round key transformation
AES	Advanced Encryption Standard
ALU	Arithmetic and Logic Unit
ASIC	Application Specific Integrated Circuit
BPSG	borophosphosilicate glass
CBC mode	cipher block chaining mode
CCM mode	counter with cipher block chaining-message authentication code mode
CCSDS	Consultative Committee for Space Data Systems
CFB	cipher feedback
CMAC mode	cipher-based message authentication code mode
CRC	Cyclic Redundancy Checking
CTR mode	counter mode
DES	Data Encryption Standard
DFA	Differential Fault Analysis
DMR	Double Modular Redundancy
DRAM	Dynamic RAM
ECB mode	electronic codebook (ECB) mode

FIT	Failure In Time
FPGA	Field Programmable Gate Arrays
GCM mode	Galois/counter mode
<i>GF</i>	Galois Field
LET	Linear Energy Transfer
LUT	Look Up Table
OFB	output feedback
MBU	Multiple Bit Upset
<i>MixColumns</i>	mix columns transformation
NIST	National Institute of Standards and Technology
PIP	Programmable Interconnect Point
Q_{crit}	critical charge
SEFI	Single Event Functional Interrupt
SET	Single Event Transient
SEU	Single Event Upset
<i>ShiftRows</i>	shift rows Transformation
SOI	Silicon on Insulator
SRAM	Static RAM
<i>SubBytes</i>	substitution transformation
TMR	Triple Modular Redundancy
VHDL	Very high speed integrated circuit (VHSIC) Hardware Description Language

Appendix B

AES in System on Chip

In this appendix, the system including the AES module is presented. Software and hardware elements that build the whole system on chip are discussed. Board level communication with the host PC is described briefly. The hardware elements include the proposed design of AES with error detection along with other IPs on the FPGA. Different hardware IPs and attachments (i.e. master, slave, and interrupt attachments) to the AES module are also discussed.

The software elements are device drivers and application software. There is a device driver for the AES module and another device driver for the host PC to perform self reconfiguration of the FPGA in case of a soft error.

The baseline for the AMIRIX board provided by CMC is used in this project. The Xilinx Platform Studio (XPS) version 10.1.03 is the development environment for designing the hardware and software of the embedded processor system. XPS which is part of the Embedded Development Kit (EDK) provides an environment to build hardware IPs and device drivers and libraries for embedded software development.

Debug and verification of the software program running on the PowerPC 405 processor is mainly done by Xilinx Microprocessor Debugger (XMD). As shown Figure 55, XMD connects to the PowerPC 405 processor through a JTAG connection. The JTAG chain inside the FPGA is through the 2 processors. The JTAG chain includes an interface bus that contains all the JTAG signals.

Communication to and control of the application software as illustrated in Figure 55 is done by the GNU Debugger remote TCP protocol.

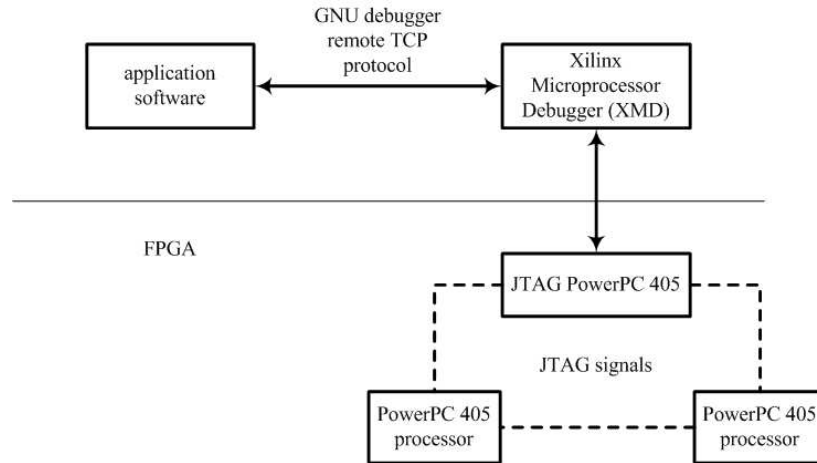


Figure 55 Debugging software running on PowerPC 405 processor

B.1 Device Driver and Application Software

The high level view of the system interconnection including the board (AMIRIX AP1100) and the host PC is illustrated in Figure 56. The Virtex-II Pro FPGA, which is the central component on the board, connects to the host PC processor by sending messages through the PowerPC 405 processor (PowerPC 405 is the processor on the Virtex-II Pro FPGA).

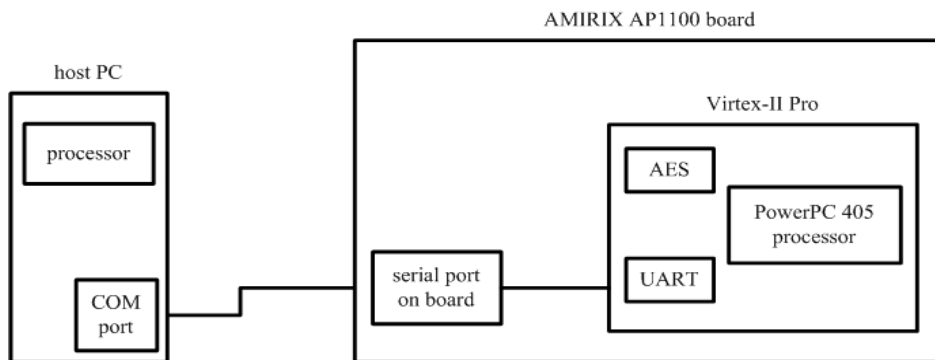


Figure 56 Board level connections

In case of an error occurrence, the PowerPC 405 processor sends an error message through the Universal Asynchronous Receiver Transmitter (UART) IP on the FPGA to the host PC processor. Then this error message triggers an interrupt in the host PC to perform reconfiguration on the FPGA. This mechanism includes interrupting both the PowerPC 405 processor on the FPGA and the host PC processor. This is further described in more details as follows.

The PowerPC 405 processor should be interrupted when the AES Module sends an interrupt to it in case of an error. The operating system of the PowerPC 405 processor is standalone which provides the board support package. The board support package is a set of software modules that provide access to processor specific functions. These functions are used when an application accesses board or processor features directly. This package is a single threaded library in which there is no operating system between the application and the hardware platform. Application software operates on the hardware platform through direct driver calls. The software layers are illustrated in Figure 57.

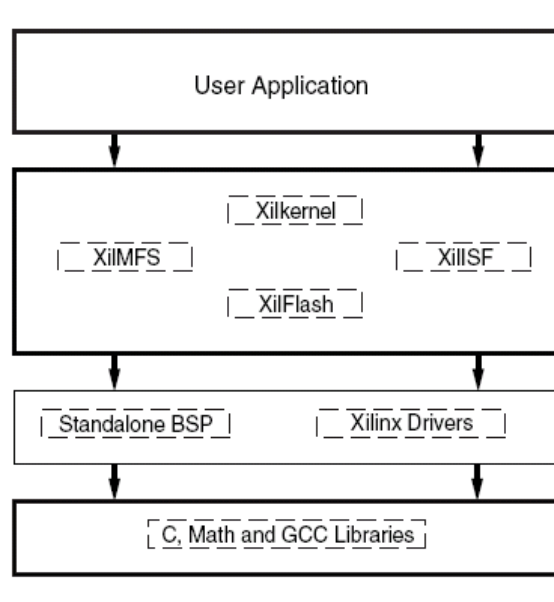


Figure 57 Software layers for processor in FPGA

The device driver for the AES module sends the error message “Error!” onto the serial port through UART. In order to set up the interrupt in the software on the FPGA side, the following steps are taken. The interrupt vector is setup through the initialization function XExc_Init(). This function should be called before registering any interrupts.

- The interrupt handler is registered through function `XExc_RegisterHandler()`.
- After registering the interrupt handler, the interrupt should be enabled. This is done by function `XExc_mEnableExceptions()`.

On the host PC side, there should be an interrupt handler that reacts upon receiving the error message sent to the serial port through UART. The interrupt handler basically reconfigures the FPGA. The following steps show how this is achieved. All the functions used are in header “windows.h”, which is a header file for the Windows API.

- The serial port (COM1) is opened by function `CreateFile()`.
- The UART parameters (baud rate, data bits, parity, stop bits, and flow control) for the receiver serial port on the host PC should be set properly.
- Time-out parameters should be set correctly, otherwise reads and writes from the serial port return unexpected values. In order to read all the values until the UART buffer is empty, the time-out values are applied as follows.

```
timeouts.ReadIntervalTimeout = MAXDWORD;
timeouts.ReadTotalTimeoutMultiplier = 0;
timeouts.ReadTotalTimeoutConstant = 0;
```

- The following is a snippet from the software that is written to reconfigure the FPGA in case of an error occurrence (the complete code is provided in Appendix D). Function `WaitCommEvent()` is used to wait on an event of receiving a byte on the UART of the receiver serial port. After receiving an event, the UART buffer is read. The advantage of this approach is that the host computer does not need to poll for synchronization, which is inefficient for incoming data. As seen in this piece of code with clarifying comments, the message read from the buffer is compared to error message “Error!”. The configuration tool `iMPACT` is run in the batch mode and the commands are passed to it through the `.cmd` file. To run `iMPACT` in the batch mode function `CreateProcess()` is used. All the arguments of the `CreateProcess()` function are described by comments in the code snippet below.

```

for ( ; ; ) {
    index=0;
    if (WaitCommEvent(comport, &dwCommEvent, NULL)) {
        do {
            if (ReadFile(comport, &INBUFFER,1, &bytes_read,NULL)) {
                if (bytes_read!=0) { // A byte has been read.
                    message[index]=INBUFFER[0];
                    index ++;
                }
                if (bytes_read==0) { // Buffer is empty.
                    message[index]=NULL; // Terminating array
                    if (strcmp(message, "Error!\0")==0) {
                        printf("FPGA should be reconfigured!\n");
                        // FPGA reconfiguration below
                        CreateProcess(NULL, // module name
                            "impact -batch _impact.cmd", // Command line
                            NULL, // Process handle not inheritable
                            NULL, // Thread handle not inheritable
                            FALSE, // Set handle inheritance to FALSE
                            0, // No creation flags
                            NULL, // Use parent's environment block
                            NULL, // Use parent's starting directory
                            &si, // Pointer to STARTUPINFO structure
                            &pi) // Pointer to PROCESS_INFORMATION structure
                    }
                }
            } else {
                // An error occurred in the ReadFile call.
                break;
            }
        } while(bytes_read); // Buffer is not empty.
    } else { // Error in WaitCommEvent.
    }
}

```

The main steps in the .cmd file to perform configuration are briefly noted below (configuration details can be found in [2, 128]).

- Switching to JTAG boundary scan configuration mode by command setMode
- Specifying the cable parameters (such as speed and port) by command setCable
- Identifying the devices in the JTAG boundary scan chain by command Identify
- Adding the device to the list of devices to be configured by command addDevice
- Deletes unnecessary devices from a device chain by command deleteDevice
- Programming the FPGA by command Program

The steps above are for configuring the FPGA when JTAG boundary scan is used. JTAG boundary scan, formally known as IEEE Standard 1149.1, is primarily a testing standard created to alleviate the growing cost of testing digital systems. The primary benefit of the standard is the ability to transform extremely difficult printed circuit board testing problems (that could only be attacked with ad-hoc testing methods) into well-structured problems that software can handle easily and efficiently. Furthermore, vendor-specific extensions to boundary scan JTAG have been developed to allow execution of maintenance and diagnostic applications as well as programming algorithms for reconfigurable parts.

B.2 AES Module and IPs in System

In addition to the main functionality of AES, the proposed module implemented should provide other interfaces to communicate with other IPs on the FPGA to build the complete system on chip. Figure 58 depicts the main IPs that are used to build the system. The Processor Local Bus (PLB) is the 64-bit local bus for the embedded PowerPC 405 processor. This PLB is compatible with IBM CoreConnect PLB.

Along with the PowerPC 405 processor, the AES module is connected to the PLB. The ciphertext outputs are stored in a block memory that is also connected to the PLB through a controller as a slave device. The main features required in the AES module to interact with other IPs on chip include the PLB master and slave supports and interrupt (these features are discussed in more detail later on in this section).

The UART IP on the 32-bit On-chip Peripheral Bus (OPB) is the standard input/output device. Therefore, input/output functions such as `xil_printf()` (this function is similar to `printf()` but much smaller in size (only 1 kB)) are sent to UART to be handled. The PLB to OPB bridge allows access from the PLB to the OPB. This bridge translates PLB transactions into OPB transactions. It functions as a slave on PLB side and a master on OPB side.

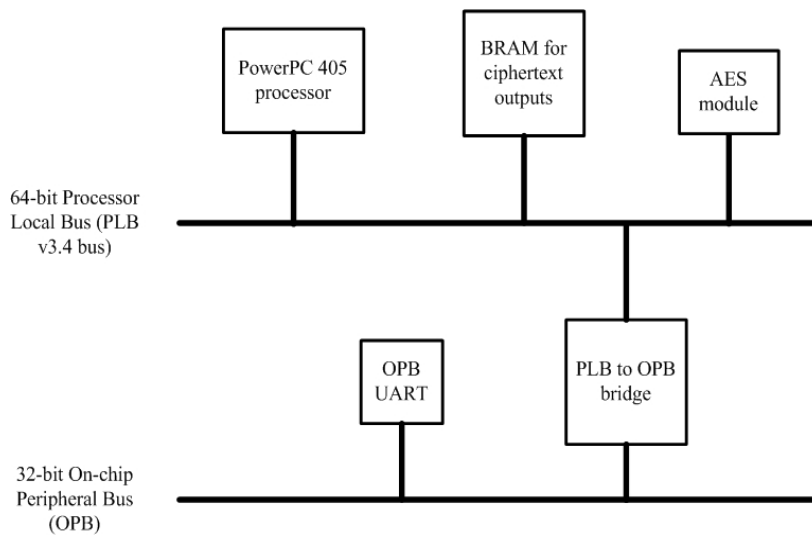


Figure 58 IPs in system on chip

The structure of the AES module is depicted in more detail in Figure 59. As shown in this figure, the master and slave features and interrupt are provided as attachments in the PLB IP interface (IPIF).

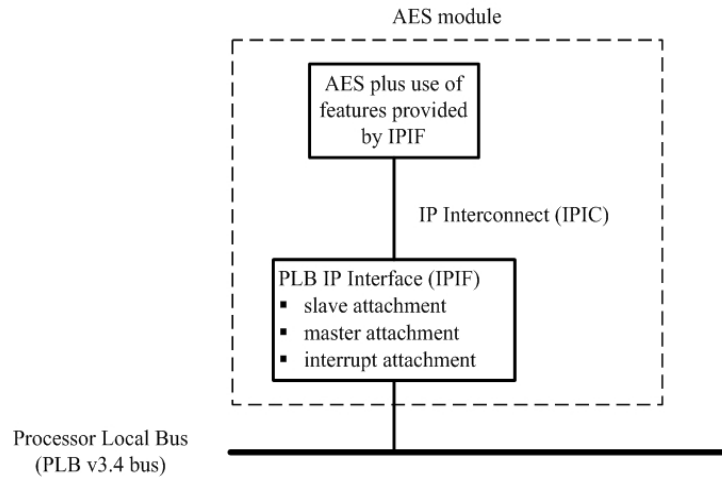


Figure 59 IP Interface (IPIF)

The interconnection between the IPIF and AES using features provided by the IPIF is denoted IP Interconnect (IPIC) in Figure 59. The signals used in the IPIC are shown in Figure 60. Detailed information on IPIF and all the IPIC signals are provided in Appendix E.

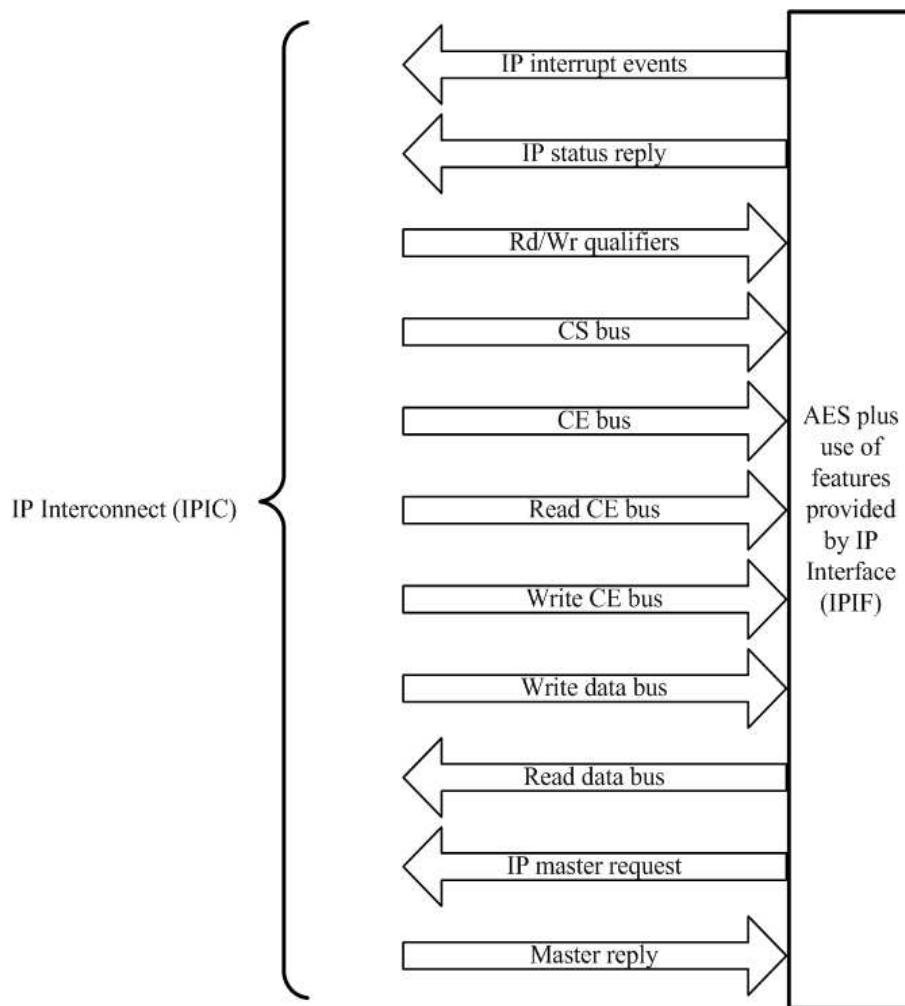


Figure 60 IP Interconnect (IPIC)

The master attachment enables the AES module to initiate transfers on the PLB. The master device requires access to the slave device to perform operations. This is illustrated as local transfer request and reply between the master and slave attachments in Figure 61. For instance, in a master read or write transaction, the slave registers provided by the slave attachment are either read from or written into. The address of the slave registers is the source for a master write and the sink for the master read. There are 128 control signals needed to be set for the IP master request in IPIC that is shown in Figure 60 (refer to Appendix E listing the IPIC signals).

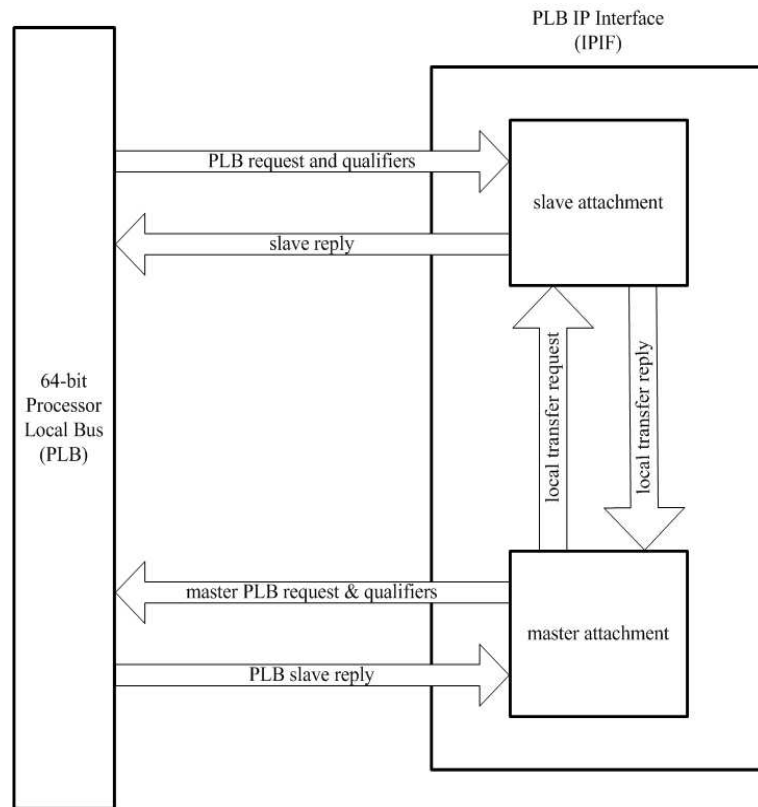


Figure 61 Master and slave attachments

To receive inputs from the software application running on the PowerPC 405 processor, the slave attachment registers (slave address space) are used. The slave registers get the inputs from the 64-bit PLB from the application software.

On the other hand, when the ciphertext output is ready the master attachment initiates a PLB transfer to write into the block memory residing on the same bus. The ciphertext output is first stored in the slave address space (or equivalently, in the slave registers). When writing the ciphertext output, the source address is a range from the slave address space that corresponds to the result registers. These slave registers are the same size as the PLB data width that is 64 bits. The destination address in the master write transfer on the PLB is a block memory address.

As shown in Figure 62, the interrupt port of the AES module is the output port in the interrupt attachment of the IPIF. The interrupt port of the AES module is connected to an interrupt port of the PowerPC 405 processor (Figure 62). The interrupt is triggered by the AES module when there is an error in computing the output. In the *MixColumns* and *AddRoundKey* transformations of the AES

module, the error detection is done through parity detection technique that is enhanced to expand its error coverage to combinational logic of datapath and routing in the FPGA. Additionally, dual ported block memory and duplication is used in *SubBytes* and the control circuitry, respectively, to detect errors (refer to Chapter 5).

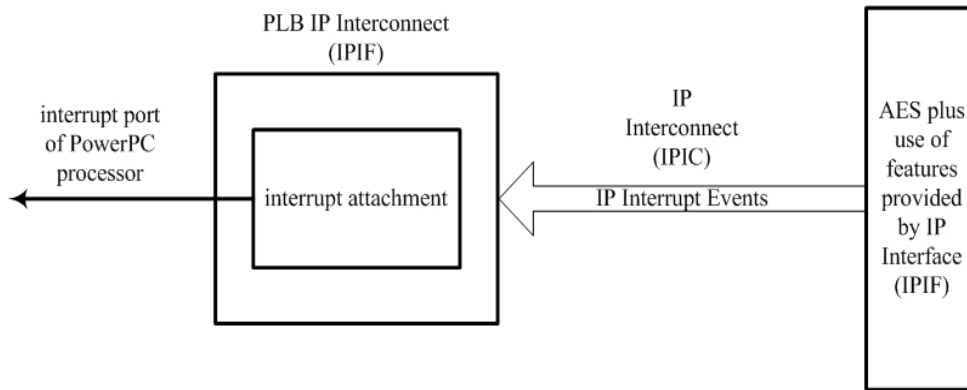


Figure 62 Interrupt in AES module

In case of an error occurrence, the PowerPC 405 processor sends the error message “Error!” to the host PC through the UART IP and the serial port (the sending of error message is done in the interrupt handler of the Aes module). Setting up the device driver (interrupt initialization, registering interrupt handler, and enabling the interrupt) in the AES module is described briefly in Section 0. The error message received on the host PC serial port initiates FPGA reconfiguration (interrupt handling due to error message on the host PC side is also presented briefly in Section 0). The memory address space of the PowerPC 405 processor is 4GB. The unused memory address space of the baseline provided by CMC is used to add new IPs to buses. Table 7.1 shows the address space assignment to the IPs shown in Figure 58.

Table 7.1 Memory address space of IPs

IP	Base address	High address	Size	Buss connection
AES module	0x40000000	0x400003ff	1kB	PLB bus
block memory	0xffff0000	0xffffffff	64kB	PLB bus
UART	0x4c000000	0x4c000fff	64kB	OPB bus

The AES module can further implement block cipher modes using the proposed AES with error detection discussed in Chapter 5. There are different options for implementing these modes. They can be implemented entirely in hardware or as a hardware/software codesign (designing a block cipher mode simultaneously using the microprocessor and FPGA logic blocks). As discussed in Section 4.2, these modes use the AES algorithm multiple times either in parallel (e.g., in ECB and CTR) or in serial (e.g., in modes with the chaining structure such as CBC) in the algorithm. Depending on the hardware, throughput, and latency requirements in an application specification, design choices should be made properly. In this research, the main goal is to provide high error coverage for single bit errors in the FPGA while having the minimum hardware resources.

Reusing hardware in a partially pipeline design (that cannot accept data every clock cycle, and thus has a throughput less than 1 per clock cycle) allows for hardware savings at the cost of throughput. On the other hand, a fully pipelined design with a typically higher hardware usage can achieve the highest throughput and accept data every clock cycle.

In an implementation that is a hardware/software codesign interfacing should also be considered carefully. Since the hardware and software parts of the design need to communicate with each other through some sort of synchronization technique (for instance, polling and interrupts). Synchronization costs in terms of performance. For instance, when the hardware part of the design finishes its computation it needs to communicate with the software part and vice versa. This communication through the bus and synchronization has a delay that can be significant compared to the overall latency.

The latency (number of clock cycles from inputs to outputs) of block cipher modes that have the chaining of plaintext blocks with the previous ciphertext blocks is constrained by this chaining structure (e.g., in CBC shown in Figure 26). Therefore, the latency cannot be improved by adding more AES ciphers (increasing hardware usage) to run in parallel. On the other hand, the latency in block cipher modes that do not have this chaining structure, for instance in the CTR mode shown in Figure 27, is reduced by running more AES ciphers in parallel.

B.3 Summary

The main goal of this appendix was to cover different aspects of design and implementation of the AES module as a system on chip in the FPGA. This system has the capability to self reconfigure in case of an error in the FPGA through serial communication with the host PC. Communication and

synchronization of the proposed AES module having error detection capability, the PowerPC 405 processor in the FPGA, and the host PC were described.

In the AES module, different features that were implemented as slave, master, and interrupt attachments to communicate with other IPs and to interrupt the PowerPC 405 processor in the FPGA were described. Device drivers that provide interrupt handling on the PowerPC 405 processor and the host PC were also covered in this appendix. Considerations in design and implementing different modes were discussed at the end.

Appendix C

Control Circuitry in AES

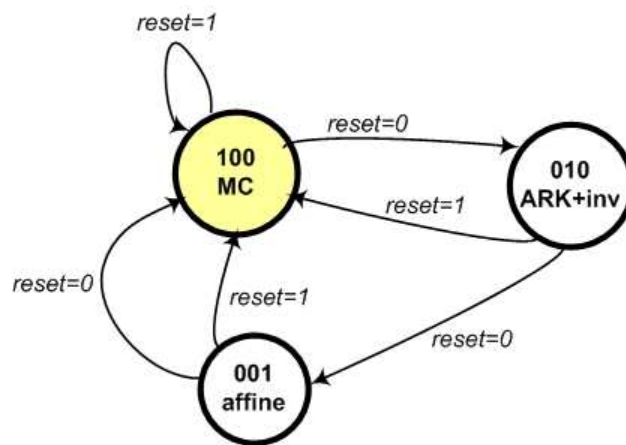
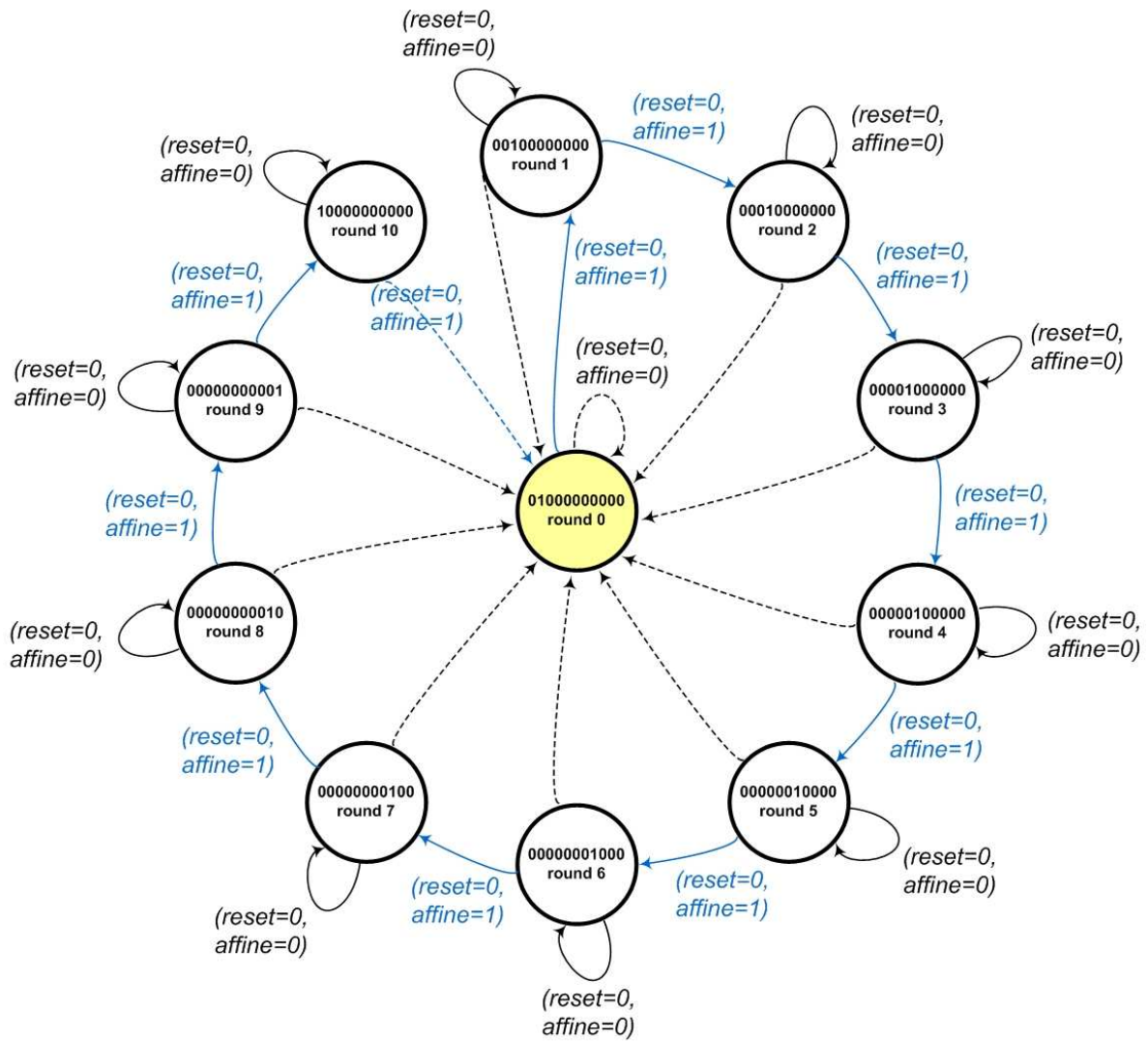


Figure 63 State machine of transformations in AES



Dotted arrows have inputs (reset=1, affine=-) on them in case of a reset. This has been removed for the figure simplicity.

Figure 64 State machine of round in AES

Appendix D

Device Driver for FPGA Reconfiguration

```
#include "windows.h"
#include "stdio.h"
#include "conio.h"

int main(int argc, char* argv[]) {

    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    unsigned int index=0;
    COMMTIMEOUTS timeouts, orig_timeouts;
    DWORD dwCommEvent;
    char INBUFFER[500];
    char message[500];
    char OUTBUFFER[20];
    DWORD bytes_read=0; // Number of bytes read from port
    DWORD bytes_written=0; // Number of bytes written to the port
    HANDLE comport=NULL; // Handle COM port
    DCB comSettings, orig_comSettings; // Contains various port settings

    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    // Open COM port
    comport=CreateFile("COM1", // open io port:
```

```

        GENERIC_READ | GENERIC_WRITE, // for reading and writing
        0, // exclusive access
        NULL, // no security attributes
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

if ((comport==INVALID_HANDLE_VALUE)) { // error processing code goes here
    printf("error opening the port!\n");
    exit(1);
}

GetCommTimeouts(comport, &orig_timeouts);
GetCommState(comport, &orig_comSettings);

comSettings=orig_comSettings;
comSettings.BaudRate=9600;
comSettings.StopBits=1;
comSettings.fBinary=false;
if (!SetCommState(comport, &comSettings)) {
    printf("SetCommState!\n");
}

printf("BaudRate %i\n", comSettings.BaudRate);
printf("StopBits %i\n", comSettings.StopBits);
printf("fParity %i\n", comSettings.fParity);
printf("Parity %i\n", comSettings.Parity);

//In order for ReadFile to return 0 bytes read, the ReadIntervalTimeout member of the
COMMTIMEOUTS structure is set to MAXDWORD, and the ReadTimeoutMultiplier and
ReadTimeoutConstant are both set to zero.
timeouts.ReadIntervalTimeout=MAXDWORD;
timeouts.ReadTotalTimeoutMultiplier=0;
timeouts.ReadTotalTimeoutConstant=0;
timeouts.WriteTotalTimeoutMultiplier=0;
timeouts.WriteTotalTimeoutConstant=0;
if (!SetCommTimeouts(comport, &timeouts)) {
    printf("SetCommTimeouts!\n");
}

if (!SetCommMask(comport, EV_RXCHAR)) {
    printf("error in SetCommMask!\n");
}

for ( ; ; ) {
    index=0;
    if (WaitCommEvent(comport, &dwCommEvent, NULL)) {
        do {
            if (ReadFile(comport, &INBUFFER,1, &bytes_read,NULL)) {

```

```

    if (bytes_read!=0) { // A byte has been read.
        message[index]=INBUFFER[0];
        index ++;
    }
    if (bytes_read==0) { // Buffer is empty.
        message[index]=NULL; // Terminating array
        if (strcmp(message, "Error!\0")==0) {
            printf("FPGA should be reconfigured!\n");
            // FPGA reconfiguration below
            CreateProcess(NULL, // module name
                "impact -batch _impact.cmd", // Command line
                NULL, // Process handle not inheritable
                NULL, // Thread handle not inheritable
                FALSE, // Set handle inheritance to FALSE
                0, // No creation flags
                NULL, // Use parent's environment block
                NULL, // Use parent's starting directory
                &si, // Pointer to STARTUPINFO structure
                &pi) // Pointer to PROCESS_INFORMATION structure
        }
    }
} else {
    // An error occurred in the ReadFile call.
    break;
}
} while(bytes_read); // Buffer is not empty.
} else { // Error in WaitCommEvent.
}
}

SetCommTimeouts(comport, &orig_timeouts);
SetCommState(comport, &orig_comSettings);
CloseHandle(comport);

char quit;
while ((quit=getchar())!='q');
printf("%c", quit);

return 0;
}

```

Appendix E

Processor Local Bus IP Interface

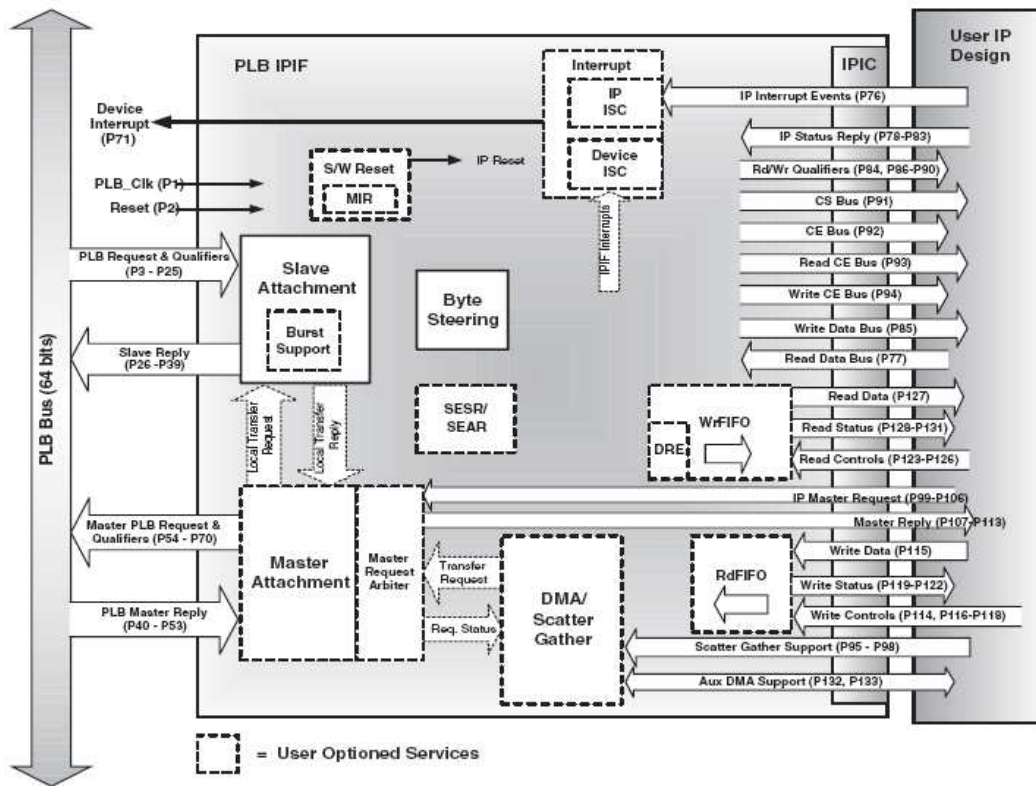


Figure 65 Connections of Processor Local Bus (PLB) IP Interface (IPIF) [129]

Table 7.2 IP Interconnect (IPIC) signals [129]

-- IP Interconnect (IPIC) signal declarations	
-- Prefix 'i' stands for IPIF while prefix 'u' stands for user logic.	
-- Typically user logic will be hooked up to IPIF directly via i<sig> unless signal slicing and muxing	
-- are needed via u<sig>	
signal iBus2IP_Clk	: std_logic;
signal iBus2IP_Reset	: std_logic;
signal iIP2Bus_IntrEvent	: std_logic_vector(0 to IP_INTR_MODE_ARRAY'length - 1);
signal iIP2Bus_Data	: std_logic_vector(0 to C_PLB_DWIDTH-1);
signal iIP2Bus_WrAck	: std_logic := '0';
signal iIP2Bus_RdAck	: std_logic := '0';
signal iIP2Bus_Retry	: std_logic := '0';
signal iIP2Bus_Error	: std_logic := '0';
signal iIP2Bus_ToutSup	: std_logic := '0';
signal iBus2IP_Addr	: std_logic_vector(0 to C_PLB_AWIDTH - 1);
signal iBus2IP_Data	: std_logic_vector(0 to C_PLB_DWIDTH - 1);
signal iBus2IP_RNW	: std_logic;
signal iBus2IP_BE	: std_logic_vector(0 to (C_PLB_DWIDTH/8) - 1);
signal iBus2IP_Burst	: std_logic;
signal iBus2IP_WrReq	: std_logic;
signal iBus2IP_RdReq	: std_logic;
signal iBus2IP_RdCE	: std_logic_vector(0 to calc_num_ce(ARD_NUM_CE_ARRAY)-1);
signal iBus2IP_WrCE	: std_logic_vector(0 to calc_num_ce(ARD_NUM_CE_ARRAY)-1);
signal iIP2Bus_Addr	: std_logic_vector(0 to IPIF_AWIDTH - 1);
signal iIP2Bus_MstBE	: std_logic_vector(0 to (IPIF_DWIDTH/8) - 1);
signal iIP2IP_Addr	: std_logic_vector(0 to IPIF_AWIDTH - 1);
signal iIP2Bus_MstWrReq	: std_logic := '0';
signal iIP2Bus_MstRdReq	: std_logic := '0';
signal iIP2Bus_MstBurst	: std_logic := '0';
signal iIP2Bus_MstBusLock	: std_logic := '0';
signal iIP2Bus_MstNum	: std_logic_vector(0 to log2(DEV_MAX_BURST_SIZE/(C_PLB_DWIDTH/8)));
signal iBus2IP_MstWrAck	: std_logic;
signal iBus2IP_MstRdAck	: std_logic;
signal iBus2IP_MstRetry	: std_logic;
signal iBus2IP_MstError	: std_logic;
signal iBus2IP_MstTimeOut	: std_logic;
signal iBus2IP_MstLastAck	: std_logic;
signal ZERO_IP2RFIFO_Data	: std_logic_vector(0 to find_id_dwidth(ARD_ID_ARRAY, ARD_DWIDTH_ARRAY, IPIF_RDFIFO_DATA, 32)-1);
signal iBus2IP_CS	: std_logic_vector(0 to ((ARD_ADDR_RANGE_ARRAY'LENGTH)/2)-1);
signal uBus2IP_Data	: std_logic_vector(0 to USER_DWIDTH-1);
signal uBus2IP_BE	: std_logic_vector(0 to USER_DWIDTH/8-1);
signal uBus2IP_RdCE	: std_logic_vector(0 to USER_NUM_CE-1);
signal uBus2IP_WrCE	: std_logic_vector(0 to USER_NUM_CE-1);
signal uIP2Bus_Data	: std_logic_vector(0 to USER_DWIDTH-1);
signal uIP2Bus_Data	: std_logic_vector(0 to USER_DWIDTH-1);
signal uBus2IP_ArData	: std_logic_vector(0 to USER_MAX_AR_DWIDTH-1);
signal uBus2IP_ArBE	: std_logic_vector(0 to USER_MAX_AR_DWIDTH/8-1);
signal uBus2IP_ArCS	: std_logic_vector(0 to USER_NUM_ADDR_RNG-1);
signal uIP2Bus_ArData	: std_logic_vector(0 to USER_MAX_AR_DWIDTH-1);

Appendix F

S-box of AES

Table 7.3 S-box of AES

63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Appendix G

Pseudo Code for Key Expansion

In the following pseudo code for the key expansion, parameters N_k and N_r represent number of words forming a key and number of rounds, respectively. Subword takes four 8-bit elements of a word and applies the substitution transformation on them.

```
KeyExpansion(byte key[4*Nk], word w[4*(Nr+1)], Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while
    i = Nk
    while (i < 4 * (Nr+1))
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
```

Appendix H

Routed Netlist Snapshots from FPGA Editor

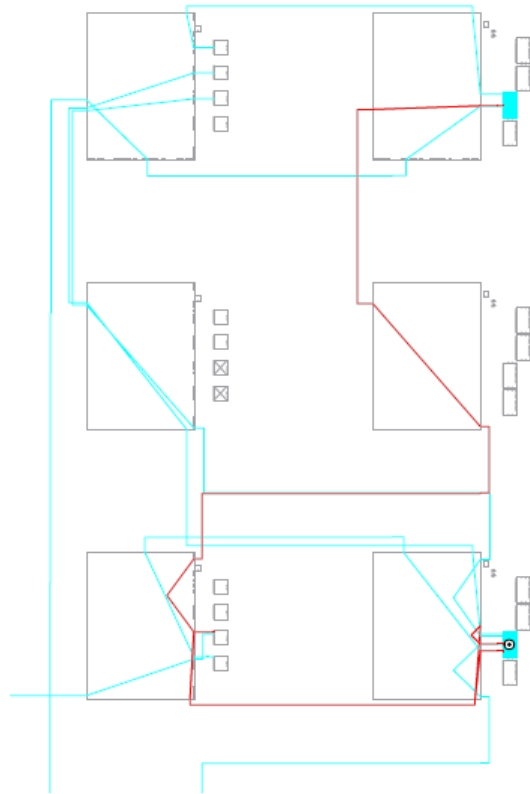


Figure 66 FPGA Editor snapshot of 2 fanout signal in *MixColumns* tested on FPGA

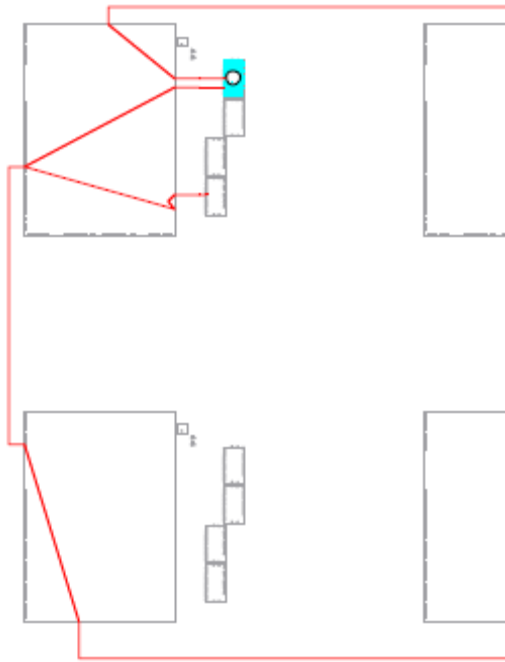


Figure 67 FPGA Editor snapshot of routing with no pins leading up to undetectable errors

Bibliography

- [1] A. Lesea, S. Drimer, J. J. Fabula, C. Carmichael and P. Alfke, "The rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, pp. 317-328, 2005.
- [2] "Virtex-II Pro and Virtex-II Pro X FPGA User Guide," November, 2007,.
- [3] "Total Ionizing Dose (TID) Effects," 2004.
- [4] F. L. Kastensmidt, L. Carro and R. Reis, *Fault-Tolerance Techniques for SRAM-Based FPGAs*. Springer US, 2006.
- [5] T. C. May and M. H. Woods, "A new physical mechanism for soft errors in dynamic memories," in *Reliability Physics Symposium, 1978. 16th Annual*, 1978, pp. 33-40.
- [6] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *Electron Devices, IEEE Transactions on*, vol. 26, pp. 2-9, 1979.
- [7] R. Baumann, "Soft Errors in Advanced Computer Systems," *IEEE Des. Test*, vol. 22, pp. 258-266, 2005.
- [8] P. Adell and G. Allen, "Assessing and Mitigating Radiation Effects in Xilinx FPGAs," 2008. Jet Propulsion Laboratory.
- [9] B. Bridgford, C. Carmichael and C. W. Tseng, "Single-Event Upset Mitigation Selection Guide," 2008. Xilinx.
- [10] J. George, R. Koga, G. Swift, G. Allen, C. Carmichael and C. W. Tseng, "Single event upsets in xilinx virtex-4 FPGA devices," in *Radiation Effects Data Workshop, 2006 IEEE*, 2006, pp. 109-114.
- [11] P. E. Dodd and L. W. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *Nuclear Science, IEEE Transactions on*, vol. 50, pp. 583-602, 2003.
- [12] T. V. Rajeevakumar, N. C. C. Lu, W. H. Henkels, H. Wei and R. Franch, "A new failure mode of radiation-induced soft errors in dynamic memories," *Electron Device Letters, IEEE*, vol. 9, pp. 644-646, 1988.
- [13] A. Pavlov and M. Sachdev, *CMOS SRAM Circuit Design and Parametric Test in Nano-Scaled Technologies: Process-Aware SRAM Design and Test*. Springer Publishing Company, Incorporated, 2008.
- [14] W. Maly, "Realistic fault modeling for VLSI testing," in *Design Automation, 1987. 24th Conference on*, 1987, pp. 173-180.

- [15] C. Constantinescu, "Intermittent faults and effects on reliability of integrated circuits," in *Reliability and Maintainability Symposium, 2008. RAMS 2008. Annual, 2008*, pp. 370-374.
- [16] M. B. Tahoori, S. Mitra, S. Toutouchi and E. J. McCluskey, "Fault grading FPGA interconnect test configurations," in *Test Conference, 2002. Proceedings. International, 2002*, pp. 608-617.
- [17] P. Graham, M. Caffrey, J. Zimmerman and E. Johnson, "Consequences and categories of SRAM FPGA configuration SEUs," in 2003, .
- [18] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE*, vol. 94, pp. 370-382, 2006.
- [19] M. Hsueh, T. K. Tsai and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, pp. 75-82, 1997.
- [20] J. Blomer and J. P. Seifert, "Fault based cryptanalysis of the advanced encryption standard (AES)," in *Wright, R.N.. (2003). Financial Cryptography. 7th International Conference, FC 2003. Revised Papers (Lecture Notes in Comput. Sci. Vol.2742)(Pp.162-181). Berlin: Springer-Verlag. viii+320pp.; Financial Cryptography. 7th International Conference, FC 2003. Revised Papers, 27-30 Jan. 2003, Guadeloupe. pp. 162-181.*
- [21] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002. 4th International Workshop Revised Papers, 13-15 Aug. 2002, Redwood Shores, CA, USA. pp. 2-12.*
- [22] J. M. Schmidt, M. Hutter and T. Plos, "Optical fault attacks on AES: A threat in violet," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009 Workshop on, 2009*, pp. 13-22.
- [23] C. H. Kim and J. J. Quisquater, "Faults, Injection Methods, and Fault Attacks," *Design & Test of Computers, IEEE*, vol. 24, pp. 544-545, 2007.
- [24] D. Boneh, R. A. DeMillo and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *EUROCRYPT '97. International Conference on the Theory and Application of Cryptographic Techniques. Proceedings, 11-15 may 1997, Konstanz, Germany. Int. Assoc. Cryptologic Res, pp. 37-51.*
- [25] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Kaliski, B.S. Jr.. (1997). Advances in Cryptology - CRYPTO '97. 17th Annual International Cryptology Conference. Proceedings(Pp.513-525). Berlin: Springer-Verlag. xii+537pp.; Advances in Cryptology - CRYPTO'97. 17th Annual International Cryptology Conference. Proceedings, 17-21 Aug. 1997, Santa Barbara, CA, USA. pp. 513-525.*
- [26] P. Dusart, G. Letourneux and O. Vivolo, "Differential fault analysis on A.E.S," in *ACNS, 2003*, pp. 293-306.

- [27] G. Piret and J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD," vol. 2779, pp. 77-88, 2003.
- [28] A. Moradi, M. T. M. Shalmani and M. Salmasizadeh, "A generalized method of differential fault attack against AES cryptosystem," in *8th International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006, Startdate 20061010-Enddate 20061013*, 2006, pp. 91-100.
- [29] C. Chen and S. Yen, "Differential fault analysis on AES key schedule and some countermeasures," in *8th Australasian Conference on Information Security and Privacy, ACISP 2003, Startdate 20030709-Enddate 20030711*, 2003, pp. 118-129.
- [30] C. Giraud, "DFA on AES," in *4th International Conference on Advanced Encryption Standard - AES 2004, Startdate 20040510-Enddate 20040512*, 2005, pp. 27-41.
- [31] J. - Q. F. Koeune, "A timing attack against Rijndael," 1999.
- [32] C. Moratelli, F. Ghellar, E. Cota and M. Lubaszewski, "A fault-tolerant, DFA-resistant AES core," in *ISCAS 2008. 2008 IEEE International Symposium on Circuits and Systems; ISCAS 2008. 2008 IEEE International Symposium on Circuits and Systems, 18-21 may 2008, Seattle, WA, USA*. pp. 244-247.
- [33] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, pp. 305-316, 2005.
- [34] E. Johnson, M. Caffrey, P. Graham, N. Rollins and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," *Nuclear Science, IEEE Transactions on*, vol. 50, pp. 2147-2157, 2003.
- [35] J. F. Ziegler, "Trends in Electronic Reliability - Effects of Terrestrial Cosmic Rays," .
- [36] S. M. Jahinuzzaman, M. Sharifkhani and M. Sachdev, "An Analytical Model for Soft Error Critical Charge of Nanometric SRAMs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, pp. 1187-1195, 2009.
- [37] P. Roche, J. M. Palau, G. Bruguier, C. Tavernier, R. Ecoffet and J. Gasiot, "Determination of key parameters for SEU occurrence using 3-D full cell SRAM simulations," *Nuclear Science, IEEE Transactions on*, vol. 46, pp. 1354-1362, 1999.
- [38] J. M. Palau, G. Hubert, K. Coulie, B. Sagnes, M. C. Calvet and S. Fourtine, "Device simulation study of the SEU sensitivity of SRAMs to internal ion tracks generated by nuclear reactions," *Nuclear Science, IEEE Transactions on*, vol. 48, pp. 225-231, 2001.
- [39] Y. Z. Xu, H. Puchner, A. Chatila, O. Pohland, B. Bruggeman, B. Jin, D. Radaelli and S. Daniel, "Process impact on SRAM alpha-particle SEU performance," in *Reliability Physics Symposium Proceedings, 2004. 42nd Annual. 2004 IEEE International*, 2004, pp. 294-299.

- [40] Bin Zhang, A. Arapostathis, S. Nassif and M. Orshansky, "Analytical modeling of SRAM dynamic stability," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, 2006, pp. 315-322.
- [41] S. M. Jahinuzzaman, M. Sharifkhani and M. Sachdev, "Investigation of process impact on soft error susceptibility of nanometric SRAMs using a compact critical charge model," in *Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on*, 2008, pp. 207-212.
- [42] Sai-Wai Fu, A. M. Mohsen and T. C. May, "Alpha-particle-induced charge collection measurements and the effectiveness of a novel p-well protection barrier on VLSI memories," *Electron Devices, IEEE Transactions on*, vol. 32, pp. 49-54, 1985.
- [43] D. Burnett, C. Lage and A. Bormann, "Soft-error-rate improvement in advanced BiCMOS SRAMs," in *Reliability Physics Symposium, 1993. 31st Annual Proceedings., International*, 1993, pp. 156-160.
- [44] J. D. Hayden, R. C. Taft, P. Kenkare, C. Mazure, C. Gunderson, B. Y. Nguyen, M. Woo, C. Lage, B. J. Roman, S. Radhakrishna, R. Subrahmanyam, A. R. Sitaram, P. Pelley, J. H. Lin, K. Kemp and H. Kirsch, "A quadruple well, quadruple polysilicon BiCMOS process for fast 16," *Electron Devices, IEEE Transactions on*, vol. 41, pp. 2318-2325, 1994.
- [45] T. Kishimoto, M. Takai, Y. Ohno, T. Nishimura and M. Inuishi, "Control of Carrier Collection Efficiency in n^+p Diode with Retrograde Well and Epitaxial Layers," *Japanese Journal of Applied Physics*, vol. 36, pp. 3460-3462, 1997.
- [46] M. Takai, T. Kishimoto, Y. Ohno, H. Sayama, K. Sonoda, S. Satoh, T. Nishimura, H. Miyoshi, A. Kinomura, Y. Horino and K. Fujii, "Soft error susceptibility and immune structures in dynamic random access memories (DRAMs) investigated by nuclear microprobes," *Nuclear Science, IEEE Transactions on*, vol. 43, pp. 696-704, 1996.
- [47] D. Belot and T. Blalack, "Modeling Substrate Effects in RF Integration," 1999. EETimes.
- [48] M. N. Horenstein, *Microelectronic Circuits and Devices*. Prentice Hall, 1996.
- [49] "Virtex-4QV Multi-Platform FPGAs," 2008. Xilinx.
- [50] S. E. Kerns, L. W. Massengill, D. V. Kerns Jr., M. L. Alles, T. W. Houston, H. Lu and L. R. Hite, "Model for CMOS/SOI single-event vulnerability," *Nuclear Science, IEEE Transactions on*, vol. 36, pp. 2305-2310, 1989.
- [51] O. Musseau, "Single-event effects in SOI technologies and devices," *Nuclear Science, IEEE Transactions on*, vol. 43, pp. 603-613, 1996.
- [52] P. E. Dodd, A. R. Shaneyfelt, K. M. Horn, D. S. Walsh, G. L. Hash, T. A. Hill, B. L. Draper, J. R. Schwank, F. W. Sexton and P. S. Winokur, "SEU-sensitive volumes in bulk and SOI SRAMs from

- first-principles calculations and experiments," *Nuclear Science, IEEE Transactions on*, vol. 48, pp. 1893-1903, 2001.
- [53] J. R. Schwank, P. E. Dodd, M. R. Shaneyfelt, G. Vizkelethy, B. L. Draper, T. A. Hill, D. S. Walsh, G. L. Hash, B. L. Doyle and F. D. McDaniel, "Charge collection in SOI capacitors and circuits and its effect on SEU hardness," *Nuclear Science, IEEE Transactions on*, vol. 49, pp. 2937-2947, 2002.
- [54] G. E. Davis, L. R. Hite, T. G. W. Blake, C. E. Chen, H. W. Lam and R. DeMoyer, "Transient Radiation Effects in SOI Memories," *Nuclear Science, IEEE Transactions on*, vol. 32, pp. 4431-4437, 1985.
- [55] M. L. Alles, S. E. Kerns, L. W. Massengill, J. E. Clark, K. L. Jones Jr. and R. E. Lowther, "Body tie placement in CMOS/SOI digital circuits for transient radiation environments," *Nuclear Science, IEEE Transactions on*, vol. 38, pp. 1259-1264, 1991.
- [56] L. R. Hite, H. Lu, T. W. Houston, D. S. Hurta and W. E. Bailey, "An SEU resistant 256 K SOI SRAM," *Nuclear Science, IEEE Transactions on*, vol. 39, pp. 2121-2125, 1992.
- [57] N. van Vonno and B. R. Doyle, "A 256 K static random-access memory implemented in silicon-on-insulator technology," in *Radiation and its Effects on Components and Systems, 1993., RADECS 93., Second European Conference on*, 1993, pp. 392-395.
- [58] J. M. Rabaey, A. Chandrakasan and B. Nikolic, *Digital Integrated Circuits*. Prentice Hall, 2003.
- [59] R. W. Berger, D. Bayles, R. Brown, S. Doyle, A. Kazemzadeh, K. Knowles, D. Moser, J. Rodgers, B. Saari, D. Stanley and B. Grant, "The RAD750TM-a radiation hardened PowerPCTM processor for high performance spaceborne applications," in *Aerospace Conference, 2001, IEEE Proceedings*. 2001, pp. 2263-2272 vol.5.
- [60] J. Liaw, "SRAM cell for soft-error rate reduction and cell stability improvement," 7257017, 2007.
- [61] F. Ootsuka, M. Nakamura, T. Miyake, S. Iwahashi, Y. Ohira, T. Tamaru, K. Kikushima and K. Yamaguchi, "A novel 0.20 μm full CMOS SRAM cell using stacked cross couple with enhanced soft error immunity," in *Electron Devices Meeting, 1998. IEDM '98 Technical Digest., International*, 1998, pp. 205-208.
- [62] R. Wilson, "ST tames soft errors in SRAM by adding capacitors," *EE Times*, 2004.
- [63] J. McCollum, "RADIATION TOLERANT SRAM BIT," 2008.
- [64] S. M. Jahinuzzaman, D. J. Rennie and M. Sachdev, "A Soft Error Tolerant 10T SRAM Bit-Cell With Differential Read Capability," *Nuclear Science, IEEE Transactions on*, vol. 56, pp. 3768-3773, 2009.

- [65] "RTAX-S/SL RadTolerant FPGAs," 2009. Actel Corporation.
- [66] D. G. Mavis and P. H. Eaton, "Soft error rate mitigation techniques for modern microcircuits," in *Reliability Physics Symposium Proceedings, 2002. 40th Annual*, 2002, pp. 216-225.
- [67] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005.
- [68] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *VLSI Test Symposium, 1999. Proceedings. 17th IEEE*, 1999, pp. 86-94.
- [69] R. P. Bastos, F. L. Kastensmidt and R. Reis, "Design of a soft-error robust microprocessor," *Microelectron. J.*, vol. 40, pp. 1062-1068, 2009.
- [70] W. Zhang, S. Gurusurthi, M. Kandemir and A. Sivasubramaniam, "ICR: In-cache replication for enhancing data cache reliability," in *Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on*, 2003, pp. 291-300.
- [71] P. A. Bernstein, "Sequoia: a fault-tolerant tightly coupled multiprocessor for transaction processing," *Computer*, vol. 21, pp. 37-45, 1988.
- [72] G. H. Asadi, V. Sridharan, M. B. Tahoori and D. Kaeli, "Balancing performance and reliability in the memory hierarchy," in *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, 2005, pp. 269-279.
- [73] V. Sridharan, H. Asadi, M. B. Tahoori and D. Kaeli, "Reducing Data Cache Susceptibility to Soft Errors," *Dependable and Secure Computing, IEEE Transactions on*, vol. 3, pp. 353-364, 2006.
- [74] "SPARC V8 32-bit Processor LEON3 / LEON3-FT," 2008. Aeroflex Gaisler AB.
- [75] R. Banu and T. Vladimirova, "On-board encryption in earth observation small satellites," in *Carnahan Conferences Security Technology, Proceedings 2006 40th Annual IEEE International*, 2006, pp. 203-208.
- [76] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," *Computers, IEEE Transactions on*, vol. 52, pp. 492-505, 2003.
- [77] M. M. Kermani and A. Reyhani-Masoleh, "Parity prediction of S-box for AES," in *Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on*, 2006, pp. 2357-2360.
- [78] L. Breveglieri, I. Koren and P. Maistri, "An Operation-Centered Approach to Fault Detection in Symmetric Cryptography Ciphers," *Computers, IEEE Transactions on*, vol. 56, pp. 635-649, 2007.
- [79] "Recommendation X.800," 1991. International Telecommunication Union.

- [80] W. Stallings, Prentice Hall, .
- [81] D. R. Stinson, *Cryptography: Theory and Practice*. Chapman & Hall, 2005.
- [82] A. J. Menezes, P. C. V. Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.
- [83] E. Barker, W. Barker, W. Burr, W. Polk and M. Smid, "Recommendation for Key Management – Part 1," 2007. NIST Special Publication 800-57.
- [84] "EFF's DES Cracker Puts Final Nail in Coffin of Insecure Government Data Encryption Standard," 1999. Electronic Frontier Foundation (EFF).
- [85] "Announcing Approval of Federal Information Processing Standard (FIPS) 197, Advanced Encryption Standard (AES)," 2001. National Institute of Standards and Technology (NIST).
- [86] "Encryption Algorithm Trade Survey," 2008. Consultative Committee for Space Data Systems (CCSDS).
- [87] C. Cid, S. Murphy and M. Robshaw, *Algebraic Aspects of the Advanced Encryption Standard (Advances in Information Security)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc, 2006.
- [88] J. Daemen and V. Rijmen, "Rijndael: The Advanced Encryption Standard," 2001. Dr. Dobb's Journal.
- [89] "Current Modes," 2008. National Institute of Standards and Technology (NIST).
- [90] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication," 2005. NIST Special Publication 800-38B.
- [91] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," 2007. NIST Special Publication 800-38D.
- [92] M. Dworkin, "Recommendation for block cipher modes of operation: The ccm mode for authentication and confidentiality," 2004.
- [93] J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc, 2002.
- [94] A. J. Elbirt, W. Yip, B. Chetwynd and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, pp. 545-557, 2001.
- [95] W. McLoone and J. V. McCanny, "Rijndael FPGA implementation utilizing look-up tables," *Signal Processing Systems, 2001 IEEE Workshop on*, pp. 349-360, 2001.

- [96] H. Kuo and I. Verbauwhede, "Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the AES Rijndael Algorithm," *Cryptographic Hardware and Embedded Systems — CHES 2001*, pp. 51-64, 2001.
- [97] V. Fischer and M. Drutarovský, "Two Methods of Rijndael Implementation in Reconfigurable Hardware," *Cryptographic Hardware and Embedded Systems — CHES 2001*, pp. 77-92, 2001.
- [98] T. Good and M. Benaissa, "Pipelined AES on FPGA with support for feedback modes (in a multi-channel environment)," *Information Security, IET*, vol. 1, pp. 1-10, 2007.
- [99] K. U. Jarvinen, M. T. Tommiska and J. O. Skytta, "A fully pipelined memoryless 17.8 gbps AES-128 encryptor," in *FPGA '03: Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, Monterey, California, USA, 2003, pp. 207-215.
- [100] A. Rudra, P. Dubey, C. Jutla, V. Kumar, J. Rao and P. Rohatgi, "Efficient Rijndael Encryption Implementation with Composite Field Arithmetic," *Cryptographic Hardware and Embedded Systems — CHES 2001*, pp. 171-184, 2001.
- [101] G. Saggese, A. Mazzeo, N. Mazzocca and A. Strollo, "An FPGA-Based Performance Analysis of the Unrolling, Tiling, and Pipelining of the AES Algorithm," *Field-Programmable Logic and Applications*, pp. 292-302, 2003.
- [102] Xinmiao Zhang and K. K. Parhi, "High-speed VLSI architectures for the AES algorithm," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, pp. 957-967, 2004.
- [103] J. Wolkerstorfer, E. Oswald and M. Lamberger, "An ASIC implementation of the AES SBoxes," in *Preneel, B.. (2002). Topics in Cryptology - CT-RSA 2002. Cryptographers' Track at the RSA Conference 2002. Proceedings (Lecture Notes in Computer Science Vol.2271)(Pp.67-78). Berlin: Springer-Verlag. x+309pp.; Topics in Cryptology - CT-RSA 2002. . the Cryptographers'Track at the RSA Conference 2002. Proceedings, 18-22 Feb. 2002, San Jose, CA, USA. pp. 67-78.*
- [104] D. Canright, "A very compact S-box for AES," in *Cryptographic Hardware and Embedded Systems - CHES 2005. 7th International Workshop. Proceedings, 29 Aug.-1 Sept. 2005, Edinburgh, UK. 3659*, pp. 441-455.
- [105] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases," *Inf.Comput.*, vol. 78, pp. 171-177, 1988.
- [106] C. Paar, "Efficient {VLSI} Architectures for Bit-Parallel Computation in {Galois} Fields," 1994. Institute for Experimental Mathematics, University of Essen.
- [107] A. Satoh, S. Morioka, K. Takano and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Advances in Cryptology — ASIACRYPT 2001*, pp. 239-254, 2001.
- [108] P. Chodowicz and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," *Cryptographic Hardware and Embedded Systems - CHES 2003*, pp. 319-333, 2003.

- [109] V. Fischer, M. Drutarovsky, P. Chodowiec and F. Gramain, "InvMixColumn decomposition and multilevel resource sharing in AES implementations," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, pp. 989-992, 2005.
- [110] Ning Chen and Zhiyuan Yan, "Compact designs of mixcolumns and subbytes using a novel common subexpression elimination algorithm," *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pp. 1584-1587, 2008.
- [111] A. Satoh and S. Morioka, "Unified Hardware Architecture for 128-Bit Block Ciphers AES and Camellia," *Cryptographic Hardware and Embedded Systems - CHES 2003*, pp. 304-318, 2003.
- [112] Hua Li and Z. Friggstad, "An efficient architecture for the AES mix columns operation," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, 2005, pp. 4637-4640 Vol. 5.
- [113] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A structure-independent approach for fault detection hardware implementations of the advanced encryption standard," in (2007). *2007 Workshop on Fault Diagnosis and Tolerance in Cryptography (Pp.47-53). Piscataway, NJ: IEEE.; 2007 Workshop on Fault Diagnosis and Tolerance in Cryptography, 10 Sept. 2007, Vienna, Austria.* pp. 47-53.
- [114] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A Lightweight High-Performance Fault Detection Scheme for the Advanced Encryption Standard Using Composite Fields," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, pp. 85-91, 2011.
- [115] N. Yu and H. M. Heys, "A compact ASIC implementation of the advanced encryption standard with concurrent error detection," in *Proceedings of the Fifth IASTED International Conference on Circuits, Signals and Systems*, Banff, Alberta, Canada, 2007, pp. 152-159.
- [116] J. Zhao, J. Han, X. Zeng and L. Han, "A Two-Dimensional Parity-Based Concurrent Error Detection Method for AES Against Differential Fault Attack and Its VLSI Implementation," *Jisuanji Yanjiu Yu Fazhan (Computer Research and Development)*, vol. 46, pp. 593-601, Apr., 2009.
- [117] N. Joshi, K. Wu and R. Karri, "Concurrent error detection schemes for involution ciphers," in Joye, M., *Quisquater, J.-J.. (2004). Cryptographic Hardware and Embedded Systems - CHES 2004. 6th International Workshop. Proceedings (Lecture Notes in Comput. Sci. Vol.3156)(Pp.400-412). Berlin: Springer-Verlag. xiii+454pp.; Cryptographic Hardware and Embedded Systems - CHES 2004. 6th International Workshop. Proceedings, 11-13 Aug. 2004, Cambridge, MA, USA.* pp. 400-412.
- [118] J. Note and \. Rannaud, "From the bitstream to the netlist," in *FPGA '08: Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, Monterey, California, USA, 2008, pp. 264-264.
- [119] "Virtex-II Pro Libraries Guide for HDL Designs," 2008. Xilinx.

[120] A. Hodjat and I. Verbauwhede, "A 21.54 Gbits/s fully pipelined AES processor on FPGA," in *Proceedings - 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2004, Startdate 20040420-Enddate 20040423*, 2004, pp. 308-309.

[121] J. Zambreno, D. Nguyen and A. Choudhary, "Exploring area/delay tradeoffs in an AES FPGA implementation," in *Becker, J., Platzner, M., Vernalde, S.. (2004). Field-Programmable Logic and Applications. 14th International Conference, FPL 2004. Proceedings (Lecture Notes in Comput. Sci. Vol.3203)(Pp.575-585). Berlin: Springer-Verlag. xxix+1198pp.; Field-Programmable Logic and Applications. 14th International Conference, FPL 2004. Proceedings, 30 Aug.-1 Sept. 2004, Antwerp, Belgium. pp. 575-585.*

[122] Bin Zhou, Yingning Peng, K. Gaj and Zhonghai Zhou, "Implementation and comparative analysis of AES as a stream cipher," in *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, 2009, pp. 396-400.

[123] T. Good and M. Benaissa, "692-nW Advanced Encryption Standard (AES) on a 0.13- μ m CMOS," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, pp. 1753-1757, 2010.

[124] Chi-Jeng Chang, Chi-Wu Huang, Hung-Yun Tai and Mao-Yuan Lin, "8-bit AES implementation in FPGA by multiplexing 32-bit AES operation," in *Data, Privacy, and E-Commerce, 2007. ISDPE 2007. the First International Symposium on*, 2007, pp. 505-507.

[125] F. Haghizadeh, H. Attarzadeh and M. Sharifkhani, "A compact 8-bit AES crypto-processor," in *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, 2010, pp. 71-75.

[126] K. V. Dalmisli and B. Ors, "Design of new tiny circuits for aes encryption algorithm," in *3rd International Conference on Signals, Circuits and Systems, SCS 2009, 2009; 3rd International Conference on Signals, Circuits and Systems, SCS 2009, Startdate 20091106-Enddate 20091108*, 2009, .

[127] P. Hamalainen, T. Alho, M. Hannikainen and T. D. Hamalainen, "Design and implementation of low-area and low-power AES encryption hardware core," in *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, 2006, pp. 577-583.

[128] "Categorical Listing of Batch Mode Commands," 2007,.

[129] "PLB IPIF," 2005. Xilinx.