11-7-2006

A new architecture for single-event upset detection & reconfiguration of SRAM-based FPGAs

Ihesiaba Kamanu

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
A New Architecture for Single-Event Upset Detection
& Reconfiguration of SRAM-based FPGAs

by

Ihesiaba Eze Kamanu

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Supervised by

Dr. Pratapa Reddy
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
November 7th, 2006

Approved By:

_________________________________________
Dr. Pratapa Reddy
Primary Advisor – R.I.T. Dept. of Computer Engineering

_________________________________________
Dr. Kenneth Hsu
Secondary Advisor – R.I.T. Dept. of Computer Engineering

_________________________________________
Dr. Marcin Lukowaik
Secondary Advisor – R.I.T. Dept. of Computer Engineering
Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title: A New Architecture for Single-Event Upset Detection & Reconfiguration of SRAM-based FPGAs

I, IHESIABA EZE KAMANU, HEREBY GRANT PERMISSION TO THE WALLACE MEMORIAL LIBRARY TO REPRODUCE MY THESIS IN WHOLE OR PART.

Ihesiaba Eze Kamanu

11/07/06
Date
Dedication

I would like to dedicate this foremost to God for his unconditional love; and also to my uncle and aunt, Dr. & Mrs. O. S. Kamanu for their assistance in giving me the opportunity to study at Rochester Institute of Technology.
I would like to express my sincere gratitude to Dr. Pratapa Reddy for his insights, patience and dedication to seeing the successful completion of this thesis. I would also like to thank Dr. Kenneth Hsu and Dr. Marcin Lukowaik for their invaluable input and continual support. Special thanks to Dr. Ken Goodnow of IBM Corporations for his immense assistance in developing the proposal for this thesis.
Abstract

Field Programmable Gate Arrays (FPGA) are used in a variety of applications, ranging from consumer electronics to devices in spacecrafts because of their flexibility in achieving requirements such as low cost, high performance, and fast turnaround. SRAM-based FPGAs can experience single bit flips in the configuration memory due to high-energy neutrons or alpha particles hitting critical nodes in the SRAM cells, by transferring enough energy to effect the change. High energy particles can be emitted by cosmic radiation or traces of radioactive elements in device packaging. The result of this could range from unwanted functional or data modification, data loss in the system, to damage to the cell where the charged particle makes impact. This phenomenon is known as a Single Event Upset (SEU) and makes fault tolerance a critical requirement in FPGA design.

This research proposes a shift in architecture from current SRAM-based FPGAs such as Xilinx Virtex. The proposed architecture includes an inherent SEU detection through parity checking of the configuration memory. The inherent SEU detection sets a syndrome flag when an odd number of bit flips occur within a data frame of the configuration memory. To correct a fault, the FPGA the affected data frame is partially reconfigured. Existing and proposed solutions include: Triple Modular Redundancy (TMR) systems; readback and compare the configuration memory; and periodically reprogramming the entire configuration memory, also known as scrubbing. The advantages afforded by the proposed architecture over existing solutions include: faster error detection and correction latency over the readback method and better area and power overhead over TMR.
# Table of Contents

**THESIS RELEASE PERMISSION FORM** .............................................. II
**DEDICATION** ........................................................................ III
**ACKNOWLEDGEMENTS** .......................................................... IV
**ABSTRACT** ........................................................................... V
**TABLE OF CONTENTS** ........................................................... VI
**LIST OF FIGURES** ................................................................. IX
**LIST OF TABLES** ................................................................. XI
**LIST OF TABLES** ................................................................. XI
**GLOSSARY** ........................................................................... XII
**GLOSSARY** ........................................................................... XII

**CHAPTER 1 – INTRODUCTION** .................................................. 1
  **INTRODUCTION** .................................................................. 1
  **PROBLEM** .......................................................................... 3
  **CHAPTER OUTLINE** ............................................................ 4

**CHAPTER 2 - BACKGROUND** ................................................... 6
  **PROGRAMMABLE LOGIC** .................................................. 6
    *PLDs* ................................................................................. 6
    *CPLDs* .............................................................................. 6
    *FPGAs* ............................................................................. 7
      Logic Blocks ........................................................................ 8
      Interconnect Switches ...................................................... 9
      IO Block .......................................................................... 10
  **FAULTS** ............................................................................ 11
    *Single-Event Upsets (SEU)* ............................................ 11
    *Single-Event Transients (SET)* ....................................... 12
    *Single-Event Latch-up (SEL)* ......................................... 12
    *Single-Event Functional Interrupts (SEFI)* ...................... 12
  **RATE OF SEUS** .................................................................. 13

**CHAPTER 3 – XILINX VIRTEX** .................................................. 16
  **FPGA CONFIGURATION HIERARCHY** ................................. 16
List of Figures

FIGURE 2-2-1: INTERNAL STRUCTURE OF CPLD ................................................................. 7
FIGURE 2-2-2: GENERIC FPGA ARCHITECTURE ............................................................. 8
FIGURE 2-2-3: FPGA CONFIGURABLE LOGIC BLOCK .................................................. 8
FIGURE 2-2-4: INTERCONNECT SWITCH BOX TOPOLOGY .............................................. 10
FIGURE 2-2-5: GENERIC FPGA IO BLOCK ..................................................................... 11
FIGURE 2-2-6: DOUBLE CLOCKING, AN EFFECT OF SET .............................................. 12
FIGURE 2-7: MEASURED UPSET RATE AS FUNCTION OF TIME (ETS-V) [31] ...................... 14
FIGURE 2-8: RATE OF SEU OCCURRENCE PER DEVICE DAY [35] ................................. 15
FIGURE 3-1: FPGA CONFIGURATION HIERARCHY [17] .................................................... 17
FIGURE 3-2: CMOS SRAM STRUCTURE [19] .................................................................... 19
FIGURE 3-3: CMOS 6-T SRAM CELL .............................................................................. 20
FIGURE 3-4: CRITICAL NODE OF A 6-T SRAM CELL [38] ............................................... 21
FIGURE 3-5: DUAL-PORT SRAM CELL ............................................................................ 22
FIGURE 3-6: CONFIGURATION MEMORY DATA FRAME [8] ........................................... 23
FIGURE 3-7: CONFIGURATION COLUMN EXAMPLE ...................................................... 25
FIGURE 4-1: BLOCK DIAGRAM DEPICTION OF A CONFIGURATION MEMORY COLUMN ... 31
FIGURE 4-2: LAYOUT OF A 32-BIT WORD CONFIGURATION MEMORY ............................. 32
FIGURE 4-3: MODIFIED LAYOUT OF A 32-BIT WORD CONFIGURATION MEMORY USING DUAL-PORT SRAM CELLS. ................................................................. 34
FIGURE 4-4: SERIAL AND BINARY PARITY TREES .......................................................... 36
FIGURE 4-5: ERROR DETECTION CIRCUIT .................................................................... 37
FIGURE 4-6: ENABLE CONTROLLER .............................................................................. 40
FIGURE 4-7: MODIFIED FPGA CONFIGURATION MEMORY WITH FAULT DETECTION CAPABILITY .......................................................... 43
FIGURE 4-8: COMPLETE SEU DETECTION CONTROLLER .......................................... 44
FIGURE 4-9: MAJOR ADDRESSING OF COLUMNS AND ADDRESS SPACES ..................... 47
FIGURE 4-10: PARTIAL RECONFIGURATION INTERFACE .............................................. 53
FIGURE 4-11: PARTIAL RECONFIGURATION CONTROLLER .......................................... 53
List of Tables

TABLE 2-2-1: SNAPSHOT OF LUT FOR OUTPUT = (A AND B) OR (C AND D) ......................................................... 9

TABLE 3-1: CONFIGURATION MEMORY DATA FOR VIRTEX DEVICES [21][22] .............................................................. 24

TABLE 3-2: DIFFERENT TYPES OF CONFIGURATION MEMORY COLUMNS AND THEIR NUMBER OF DATA FRAMES ................................................................................................................. 25

TABLE 4-1: PARITY BIT EXAMPLE ...................................................................................................................................... 35

TABLE 4-2: VIRTEX FAMILY MAJOR ADDRESSING SCHEME .......................................................................................... 48

TABLE 4-3: I/O PORT ADDRESSES .................................................................................................................................. 53

TABLE 5-1: SUMMARY OF PERCENTAGE INCREASE IN AREA FROM SYNTHESIS REPORT ........................................ 66

TABLE 5-2: CRITICAL PATH GATE DELAYS FROM SYNOPSYS SYNTHESIS TOOL ...................................................... 68

TABLE 5-3: POWER DISSIPATION RATIO OF BAREBONE AND PROPOSED MODELS .................................................. 69

TABLE 5-4: SUMMARY OF SEU MITIGATION SCHEME’S PROPERTIES ........................................................................ 76
Glossary

ASIC  Application Specific Integrated Circuits

Bitstream  The file that configures the FPGA. The bitstream gets loaded into an

CMOS  Complementary Metal-Oxide Semiconductor

Configuration  The process of loading the “brain” of the FPGA or bitstream that defines the design implemented in the FPGA.

CPLD  Complex Programmable Logic Device

Data Frame  A 1-bit vertical slice of Xilinx Virtex FPGAs. It is the smallest unit of the configuration memory that can be written to or read from.

EPROM  A form of erasable programmable read-only memory. It is a typical non-volatile memory.

FPGA  Field Programmable Gate Arrays

Parity  A single-bit error detection code that counts the number of ones (odd parity) or zeros (even parity) in a given data to determine if a bit has flipped.

Partial Reconfiguration  The process of reconfiguring portions of the FPGA without interrupting normal operation within other portions.

Readback  Process of reading back Xilinx Virtex FPGA configuration memory

SEFI  Single Event Functional Interrupts

SEL  Single Event Latch-ups

SelectMAP  An 8-bid bi-directional data bus interface used for configuration in Xilinx Virtex FPGA
SET  Single Event Transients
SEU  Single Event Upsets.
SRAM Static Random Access Memory used to store FPGA configuration data
Synthesis Process of creating a netlist from a circuit description described using Hardware Description Languages (HDL) such as Verilog and VHDL.
TMR  Triple Modular Redundancy
VHDL Very High Speed Integrated Circuit Hardware Description Language
Virtex XCV150 A Xilinx FPGA device
XPower Power dissipation estimation tool for designs in Xilinx FPGAs.
Chapter 1 – Introduction

Introduction

The advantages afforded by Field Programmable Gate Arrays (FPGAs) over Application Specific Integrated Circuits (ASICs) make their use common across a wide range of applications. Those applications include various consumer electronics; commercial machinery and equipment; space applications; and other applications where high dependability and low cost are mandatory constraints. The principal advantages FPGAs offer over ASICs include: high flexibility in achieving multiple requirements such as fast turnaround time and small NRE (Non Refundable Engineering) costs[^1].

SRAM-based FPGAs such as those produced by Xilinx, Altera, and Actel store their configuration data stream in SRAM. The configuration data control FPGA components such as the logic control blocks, function look-up tables, interconnect matrix and IO Blocks. Hence, it controls the entire functionality of the FPGA. To maintain the dependability of the FPGA, it is paramount that the configuration bits are protected against faults such as Single Event Upsets (SEU)[^43].

The two main configurable structures in an SRAM-based FPGA are the look-up tables (LUT) and the interconnect switches; both are configured using SRAM cells. The configuration bit for a routing switch is the select bit(s), so a change in its value would change the routing of the function being implemented. If a change occurs in the configuration bits of the LUT table, the entire function being implemented could be affected. For instance, a LUT implementing an AND-gate could suddenly be implementing an OR-gate. This phenomenon is known as Single-Event Upset.
The effect of a fault in the sequential portion of the FPGA is transient because the fault can be corrected in the next load of the cell. For the combinational portion of an FPGA, upsets caused by flips in configuration bits first appear as transient faults, then become permanent if the transient fault is latched by a storage cell, unless some detection technique is used.

A significant amount of research has been done in making FPGAs more robust and fault tolerant. These researches have led to a number of solutions currently available in FPGAs, and many proposed ideas in publications. Xilinx Virtex FPGAs support Triple Modular Redundancy (TMR) for mitigation of SEUs in designs \(^4\). “XTMR” is a tool developed by Xilinx that automatically builds TMR into designs \(^5\).[37]. This tool was designed to aid developers in making their designs SEU tolerant.

Other methods that have been suggested in detecting and correcting transient faults in FPGAs include: using the readback function of the FPGA to detect when an upset to the configuration memory has occurred and reconfigure the affected frame \(^4\). This entails periodically reading back the entire configuration memory of the FPGA and then comparing with the original configuration data stored in flash memory. If a fault is detected, the affected frame is partially reconfigured. Another proposal is to periodically reload the entire configuration data of the FPGA in order to overwrite any SEUs that might have occurred. This method is commonly known as “scrubbing” \(^7\) and it bypasses the need for readback and partial reconfiguration. Other radical solutions include altering the structure of the SRAM cells currently used. Asymmetric SRAM cells are to be used across the ‘care bits’ of the configuration memory\(^3\). This solution is advantageous
because it reduces the chances of SEUs occurring in the most important configuration data of the FPGA.

Problem

The principal purpose of this research is to find a solution to designing single-event upset tolerant FPGAs that improves on key existing methods. Current and suggested solutions are effective in detecting and correcting upsets in the FPGA configuration memory with the drawback being the overhead incurred. As will be discussed in greater detail in a subsequent chapter, triple modular redundancy guarantees that any SEU will be “handled”, but it incurs a significant cost in area and power. The FPGA would effectively need to be at least three times larger and denser in order to achieve full SEU tolerance. The additional modules and the voting logic would also increase the power consumed. In addition, highly reliable systems require very fast indication when an error occurs. Therefore, the voting logic and other logic required to detect the error needs to be speed efficient.

In the same vein, readback function is an effective method of SEU detection. If a particle penetrates the susceptible portion of a configuration SRAM cell and thus alters its state, a readback and verification of the configuration data will detect the upset. Due to the details of how readback is performed, as will be discussed in subsequent chapters, it effectively requires three times the amount of system memory originally needed for configuration. This is not desirable for a lot of applications, especially in space applications where memory is expensive and board space is premium. It is also not desirable or efficient to periodically reprogram the entire FPGA in order to overwrite any
upsets in the configuration memory in highly reliable systems, as this will introduce a considerable ‘downtime’ in the system while it is being periodically reconfigured.

The solution proposed takes into account the limited efficiency in area and power provided by TMR and the additional memory required for readback of the configuration data. It is an inherent parity checking of the SRAM memory that stores the configuration bitstream. Dual-port SRAM cells are used, where one port is used for normal FPGA operation and the second port dedicated to parity checking and error detection. 1-bit wide slices of the configuration memory, known as data frames, would incorporate a binary XOR tree that sets a syndrome flag when an odd number of upsets occur within that frame. If a syndrome flag is set, then the FPGA rapidly reconfigures that affected frame while the rest of the system is fully operational.

Chapter Outline

An error detection circuitry inherent to the configuration memory of SRAM-based FPGAs is proposed to detect SEUs. The error detection is achieved by computing the parity of data frames within the configuration memory. Chapter 2 will present some background and definitions that would be helpful in understanding later chapters. The following chapter introduces the architecture of Xilinx Virtex FPGAs. Features of Virtex devices such as partial reconfiguration are also discussed. Chapter 4 introduces and explains the idea behind the inherent SEU detection circuitry and also the methodology of fault correction when an SEU is detected. Chapter 5 discusses the VHDL simulation of the FPGA models. It also shows the result of a fault insertion simulation and the general functionality of the new architecture. It presents results such as error detection and
correction latency, area overhead, and power consumption. The final chapter concludes the thesis and presents intended future work.
Chapter 2 - Background

Programmable Logic

Programmable logic devices are electronic devices used to design flexible digital logic circuits in hardware. They are manufactured to perform no predefined set of functions, unlike logic gates. They have to be programmed to perform desired functions by the user before being used in a circuit. There are number of different types of programmable logic which will be discussed next.

PLDs

The first programmable devices are known as Programmable Logic Devices (PLDs) or Programmable Array Logic (PAL) or Generic Array Logic (GAL). They are the low-end programmable devices and are mainly used in replacing 7400 TTL gates such as AND, OR and XOR from circuit boards. Inside each PLD is a set of connected macrocells. These macrocells are typically comprised of some amount of combinatorial logic (AND, OR gates, for example) and a flip-flop. In other words, a small boolean logic equation can be built within each macrocell. Hardware design for PLDs is generally written in languages such as ABEL and PALASM [9].

CPLDs

The evolution of PLDs led to larger and denser programmable logic known as Complex Programmable Logic Devices (CPLDs). They can be thought of as multiple PLDs in one semiconductor chip, including programmable interconnect. CPLDs as the
name suggests, are capable of implementing more complex digital logic circuits. Figure 2-2-1 shows a simplified depiction of the internal structure of a CPLD, with the logic blocks indicating an individual PLD. There can be more or fewer logic blocks, depending on the size and density of the CPLD being designed.

![Figure 2-2-1: Internal structure of CPLD][9]

**FPGAs**

Field Programmable Gate Arrays (FPGA) are semiconductor devices with programmable elements and programmable interconnects and provide a flexible means of “breadboarding” logic design [10]. They are the next device in line in the evolution of programmable devices. As can be predicted, they are far more complex than CPLDs and can be used to implement various digital logic circuits. Another notable difference between CPLDs and FPGAs is that modern FPGAs included embedded DSPs, processors and memories. They can be programmed and reprogrammed as desired by the user and can also be partially reconfigured while the system is operational. Due to their flexibility and complexity, they are beginning to replace traditional Application Specific Integrated
Circuits (ASICs). The main components of FPGAs include an array of logic blocks, an interconnect matrix and switches, and a set of IO blocks as is shown in Figure 2-2-2.

![Generic FPGA Architecture](image)

**Figure 2-2-2: Generic FPGA Architecture**

**Logic Blocks**

They are typically referred to as Configurable Logic Blocks (CLBs) in Xilinx FPGAs and represent the most basic unit of the FPGA. It is where the configured digital logic resides. The CLB of modern SRAM-based FPGAs include 4-input Look-Up Tables (LUT), multiplexers and flip-flops to enable it implementing any 4-input function as depicted in Figure 2-2-3.

![FPGA Configurable Logic Block](image)

**Figure 2-2-3: FPGA Configurable Logic Block**
The LUTs are arrays of SRAM cells in the configuration memory, while the four inputs represent addresses to SRAM.

Table 2-2-1 is a snapshot of the LUT implementing a simple function with inputs (A, B, C and D) representing a 4-bit address to an SRAM cell. Five states out of a possible sixteen is illustrated for simplicity.

Table 2-2-1: Snapshot of LUT for Output = (A and B) or (C and D)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Interconnect Switches

In order to implement large functions, multiple CLBs have to be connected in various patterns. For instance, each CLB in a Xilinx Virtex-4 can implement a 128x1 \[^{[11]}\] ROM (Read-only Memory); to design larger ROMs (e.g. 512x4) multiple CLBs have to be connected both down the rows and columns of the CLB array. The interconnect that achieves this is also programmable, controlled by configuration bits stored in SRAM.
Figure 2-2-4: Interconnect Switch Box Topology

Figure 2-2-4 above shows a typical switch box at the edge of every CLB. Depending on the configuration bits for any particular switch, vertical and horizontal interconnects could be connected.

IO Block

Each input of the 4-input CLBs is accessible from one side of the logic block, while the output pin can connect to routing wires in both the channel to the right and the channel below the logic block. Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it. Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. Figure 2-2-5 shows a typical IO Block.
Faults

There are a number of faults that could affect the proper functioning of an FPGA. They range from permanent faults such as ‘Stuck at 1 and 0,’ interconnect bridging to soft errors such as Single-Event Upsets, Single-Event Transients, Single-Event Latch-up and Single-Event Functional Interrupts. This thesis will not address permanent faults in FPGAs, their effects or any subsequent ramifications. It will focus instead on the effect of soft-errors caused by single-event upsets. The following is a description of the basic soft errors that could affect the configuration memory of an FPGA.

Single-Event Upsets (SEU)

A single-event upset is caused when a charged particle deposits its charge in a static semi-conductor device such as an SRAM cell. It can be caused by alpha particles hitting certain critical nodes of an SRAM cell, and generating a high density of holes and electrons in the substrate, causing an imbalance in the device’s electrical potential distribution and causing data to be corrupted or flipped\(^{12}\). Alpha particles are not only released through cosmic radiation, but can also be emitted by traces of radioactive

\[\text{Figure 2-2-5: Generic FPGA IO Block}\]
elements in device packaging\cite{12}. This thesis is concerned with introducing a mitigation technique for single event upsets.

**Single-Event Transients (SET)**

Single Event Transients occur when the charge from an energetic subatomic particle strikes a combinational logic element causing a transient voltage disturbance, which can propagate and be latched by a storing device and ultimately resulting in an SEU\cite{13}. SETs can manifest as erroneous double clocking as is shown in Figure 2-2-6.

![Heavy ion induced negative pulse](image)

*Figure 2-2-6: Double Clocking, an effect of SET*

**Single-Event Latch-up (SEL)**

SEL is a condition that causes loss of device functionality due to a single event induced high current state. It results in a high operating current, above device specification. They may or may not develop into permanent errors but are potentially destructive and are caused by heavy ions as well as protons in very sensitive devices \cite{14}\cite{15}\cite{16}. The common mitigation technique for SEL is radiation hardening of the FPGA.

**Single-Event Functional Interrupts (SEFI)**

Single event functional interrupts are not independent faults but occur in consequence to SEUs. They are caused when a severe SEU in a device’s control circuitry places the device in a test, mode, halt, or undefined state. The ramification is generally a
reset of the device. Preventing or mitigating SEUs would indirectly prevent any chances of an SEFI occurrence. Hence, this thesis is also concerned (indirectly) with SEFIs.

**Rate of SEUs**

Occurrence of single event upsets is much greater in space applications than applications used on earth. Therefore, to determine a good rate for their occurrence, it is best to measure the number of faults occurrence in orbit. Limited direct measurements have been done in space, even though research in the field is growing. In 1991, the on-orbit data of single event phenomena were obtained for the CMOS SRAMs equipped in Engineering Test Satellite-V (ETS-V) in a geostationary orbit[^31]. It was observed that the rate of SEUs increased when solar flares occurred. The monitor used for measurement of SEU and SEL by ETS-V developed by NASDA in collaboration with NTT, Japan has the following functions as reported in the referenced publication:

1) Measurement of the frequency of SEU occurring at the RAM devices.

2) Measurement of the frequency of SEL by monitoring the current supplied to the RAM devices. When SEL occurs, the RSM turns off the current, turns it on again and resets the state of RAMs.

3) Measurements of the number of bits which lose the memory function by a hard error.

4) Measurement of the total current supplied to the RAM devices aiming at the detection of any deterioration of the devices owing to the total-dose effect.

5) Telemetering of the data acquired by the above measurements, the RAM identification and the state of RAM devices.
The rate of upsets found was highly influenced on days where solar flares were observed. Figure 2-7 shows the graph of the number of upsets obtained per week. The peaks in the graph indicate days with solar flares. Deducing the data given in the graph, it can be seen that an average of about 5 upsets occur per week on days/weeks with little or no solar flares. This translates to a meager rate of less than one SEU occurrence per day. Other research that has been done in this subject includes predicting rates of SEU in space.

Figure 2-7: Measured Upset rate as Function of time (ETS-V)\textsuperscript{[31]}

Another recent study\textsuperscript{[35]} on the rate of SEU occurrence performed to test SEU mitigation techniques for the Xilinx Virtex FPGA shows similar results. The rate expectedly increases during solar flares. Figure 2-8 shows the rate of SEU occurrence per device day at various altitudes at 60 degrees inclination in orbit. The SEU occurrence estimate was derived using the CREME96 model\textsuperscript{[36]}. On “quiet sun” days, meaning no solar flares, the average SEU occurrence rate in the XQVR300 devices is just under 1 per day. There are tools\textsuperscript{[38]} developed for estimating the probability of SEU impact on an FPGA design.
XQVR300 SEU Rates
Static hits upset only

Orbital Altitude (km) (at 60 deg)

Figure 2-8: Rate of SEU occurrence per device day.\textsuperscript{[35]}
Chapter 3 – Xilinx Virtex

For the purposes of this thesis, Xilinx Virtex has been chosen as the base FPGA architecture to implement an SEU mitigation scheme. This thesis would propose a modified architecture to Virtex that could also easily be adapted to other SRAM-based FPGA architectures. Virtex was chosen because it is the state-of-the-art technology and has been the base architecture for a number of previous researches in SEU mitigation. This chapter would describe its architecture and various components related to this work.

FPGA Configuration Hierarchy

At power off, SRAM-based FPGAs are unconfigured or blank and therefore implement no functions. When it is powered on, “as the power supply voltage rises and crosses a certain threshold, the FPGA begins to load its "brains" (configuration) and all I/O pins are set in a tri-state condition. The internal configuration clock becomes active and begins to clock data from the configuration data storage into the configuration latches” [17]. After the configuration data is loaded, its resources such as CLBs, IO Blocks and interconnects “come alive”. Each configuration bit controls a specific portion of the FPGA, and the configuration data stream is stored in SRAM.

Xilinx Virtex-4 is organized in a hierarchal manner with Configuration, User Logic and Routing on different layers. Using a house analogy, the configuration bit streams are stored in the basement. The resources that make up the user logic, such as look-up tables, I/O pin definition, clock distribution, flip-flops reside on the next level,
the first floor. The routing and interconnect layer is on the top floor. This Xilinx Virtex hierarchal architecture is depicted in Figure 3-1: FPGA Configuration Hierarchy

![Figure 3-1: FPGA Configuration Hierarchy][17]

The “Configuration” layer is much larger than the “User Logic” because it requires numerous configuration bits to control a single CLB. In Virtex-4 it takes approximately 30 configuration latches to configure a CLB, and each latch controls a property of the CLB, routing or I/O block [17].

**Configuration Layer**

Single Event Upsets generally occur in this layer and it therefore represents an ideal layer to mitigate (detect and correct) any such occurrence. A flipped bit in this layer, as has been described, directly affects a property of the CLB, I/O Block or Interconnect it is controlling. For an SRAM-based FPGA, such as Virtex-4, the configuration bit stream is stored in SRAM.
SRAM

Static Random Access Memory (SRAM) is static semiconductor memory that can be accessed randomly. It is volatile memory, meaning it only retains its value if the device is powered. Another form of volatile semiconductor memory is the dynamic random access memory (DRAM). It is dynamic because a periodic refresh of its content is required. The access time of SRAM is smaller than that for DRAM. SRAM is however more expensive and consumes more power. The most common use for SRAM is design of caches in microprocessors. Figure 3-2 is the block diagram of a complete SRAM circuit with all necessary components.

The row and column decoders decodes an input address to a particular word or cell. The precharge circuit is used to precharge the bit lines before a read or write operation is performed while the sense amplifiers serve two functions: improves the discharge time of the bit lines and amplifies the output voltage rail-to-rail. The SRAM cells making up the ‘RAM ARRAY’ in Figure 3-2 would be discussed in greater detail as this is the only portion of the SRAM circuit pertinent to this thesis.
SRAM Cell

Individual bits are stored in two cross-coupled CMOS inverters with two access transistors, forming a six transistor (6-T) cell. The valid states for a 6-T SRAM cell are logic ‘0’ and ‘1’. Figure 3-3 below is the transistor level diagram of a 6-T CMOS SRAM Cell. The cell is accessed by enabling the word line (WL) which controls the two access transistors, labeled M₅ and M₆. If WL is asserted, then the bit lines (BL and \(\overline{BL}\)) are directly connected to the storage cell, and can hence read or write to it. The two bit lines are inverses of each other and provides an improvement in noise margin compared to if only one bit line is used.
The three distinct operations of an SRAM are: standby, read and write.

**Standby**

As the name suggests, this is the state of the SRAM when it is not being actively used; i.e. it is not being read or written to. The word line is deasserted and therefore the access transistors M₅ and M₆ in Figure 3-3 are disconnected from the cell. At this state, the cross-couple inverters continue to reinforce each other and maintain any data previously written.

**Reading**

Assume that the content of the memory is a 1, stored at Q. The read cycle is started by precharging both the bit lines to a logical 1, then asserting the word line WL, enabling both access transistors. The second step occurs when the values stored in Q and \( \overline{Q} \) are transferred to the bit lines by leaving BL at its precharged value and discharging
\( \overline{BL} \) through \( M_1 \) and \( M_5 \) to a logical 0. On the BL side, the transistors \( M_4 \) and \( M_6 \) pull the bit line towards \( V_{dd} \), a logical 1. If the content of the memory was a 0, the opposite would happen and \( \overline{BL} \) would be pulled towards 1 and BL towards 0.

**Writing**

This process is much easier than the reading operation. To write a logical 1 to the cell, the word line WL is asserted and logic 0 is applied to BL while logic 1 is applied to \( \overline{BL} \). It is important to note that proper sizing of the transistors in the SRAM cell is critical to its correct functionality.

**SEU Critical Node**

SRAM cells are susceptible to single event upsets caused by alpha particles and other high-energy atmospheric neutrons. It occurs when a high-energy particle strikes a sensitive node of the SRAM cell, leaving behind an ionized track. This can effect a bit flip if the voltage deposited is high enough. The shaded portions of Figure 3-4 are the critical nodes of a typical 6-T SRAM cell.

![Critical Node of a 6-T SRAM Cell](image)
Dual-port SRAM Cell

Some applications require that the SRAM cells be dual-accessed. This is accomplished by adding two extra access transistors on either side of the cross coupled inverters. As well as the two additional access transistors, two extra bit lines and word lines are introduced as is shown in Figure 3-5. This allows for simultaneous reading of the cell through the two ports but can only be written to through one point at any particular time. Dual-port SRAM as would be discussed in a subsequent chapter forms the bedrock of the SEU mitigation solution proposed.

![Figure 3-5: Dual-port SRAM Cell](image)

Configuration Memory

Xilinx Virtex-4 FPGA has its configuration memory laid out in a regular pattern below the user logic layer and individual cells are close to the specific functions they control. The atomic component or the smallest part of the memory that can be read or written to is known as a data frame[^8]. It is a 1-bit slice of the memory across its vertical
axis. Configuration data stream is written to memory one data-frame at a time. A data frame in the configuration memory of a Virtex FPGA is illustrated below in Figure 3-6. Each data frame stores multiple 32-bit words depending on the size of the FPGA device.

Table 3-1 shows different Xilinx Virtex devices. The table has the number of rows and columns of configuration memory for each device, the number of bits per frame, and the number of words per frame [21]. The number of configuration bits in each device can then be calculated by multiplying the product of the rows and columns by the number of frames and bits per frame. The number of words per frame is determined by dividing the number of bits per frame by 32-bits.

Figure 3-6: Configuration Memory Data Frame [8]
<table>
<thead>
<tr>
<th>DEVICE</th>
<th>ROW X COL.</th>
<th>BITS/FRAME</th>
<th>WORDS/FRAME</th>
<th>FRAMES</th>
<th>CONFIGURATION BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCV50</td>
<td>16 x 24</td>
<td>384</td>
<td>12</td>
<td>1453</td>
<td>559,200</td>
</tr>
<tr>
<td>XCV100</td>
<td>20 x 30</td>
<td>448</td>
<td>14</td>
<td>1741</td>
<td>781,216</td>
</tr>
<tr>
<td>XCV150</td>
<td>24 x 36</td>
<td>512</td>
<td>16</td>
<td>2029</td>
<td>1,040,096</td>
</tr>
<tr>
<td>XCV200</td>
<td>28 x 42</td>
<td>576</td>
<td>18</td>
<td>2317</td>
<td>1,335,840</td>
</tr>
<tr>
<td>XCV300</td>
<td>32 x 48</td>
<td>672</td>
<td>21</td>
<td>2605</td>
<td>1,751,808</td>
</tr>
<tr>
<td>XCV400</td>
<td>40 x 60</td>
<td>800</td>
<td>25</td>
<td>3181</td>
<td>2,546,048</td>
</tr>
<tr>
<td>XCV600</td>
<td>48 x 72</td>
<td>960</td>
<td>30</td>
<td>3757</td>
<td>3,601,968</td>
</tr>
<tr>
<td>XCV800</td>
<td>56 x 84</td>
<td>1088</td>
<td>34</td>
<td>4333</td>
<td>4,715,552</td>
</tr>
<tr>
<td>XCV1000</td>
<td>64 x 96</td>
<td>1248</td>
<td>39</td>
<td>5277</td>
<td>6,587,520</td>
</tr>
<tr>
<td>XCV2000E</td>
<td>80 x 120</td>
<td>1376</td>
<td>48</td>
<td>6613</td>
<td>10,159,648</td>
</tr>
<tr>
<td>XCV3200E</td>
<td>104 x 156</td>
<td>1952</td>
<td>61</td>
<td>8341</td>
<td>16,283,712</td>
</tr>
</tbody>
</table>

Table 3-1: Configuration memory data for Virtex Devices\textsuperscript{[21][22]}

As can be deduced from Figure 3-6, the configuration memory is laid out in a fashion that matches the "user logic" layer of the Virtex FPGAs. The configurable logic block (CLB) configuration data stream occupy the middle of the memory, while the configuration data stream of the I/O blocks and Block SelectRAM (BRAM) occupy the outer columns of the configuration memory.

The number of data frames per column varies depending on the type of configuration data stored in that column\textsuperscript{[21][22]}. The center column which includes configuration for the global clock pins has 8 data frames. Each column that stores configuration data for CLBs has 48 data frames. The number of data frames is also
different for the I/O blocks, block SelectRAM Interconnect and block SelectRAM Content as is summarized in Table 3-2. An illustration of the different columns and the number of data frames for Xilinx Virtex device, XCV50 is shown in Figure 3-7.

<table>
<thead>
<tr>
<th>COLUMN TYPE</th>
<th># OF FRAMES</th>
<th># PER DEVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Center</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>CLB</td>
<td>48</td>
<td># of CLB Columns</td>
</tr>
<tr>
<td>IOB</td>
<td>54</td>
<td>2</td>
</tr>
<tr>
<td>Block SelectRAM Interconnect</td>
<td>27</td>
<td># of block SelectRAM columns</td>
</tr>
<tr>
<td>Block SelectRAM Content</td>
<td>64</td>
<td># of block Select RAM columns</td>
</tr>
</tbody>
</table>

Table 3-2: Different types of configuration memory columns and their number of data frames.

Figure 3-7: Configuration Column Example
Partial Reconfiguration

An important feature in Xilinx Virtex FPGAs is the ability to reconfigure parts of the system while other portions of the design implemented continue to be executed. This is known as “partial reconfiguration”. This is especially useful for applications that require the flexibility of loading different parts of a design without resetting the system or completely reconfiguring it \[23\]. A perfect example for this application is a proposed parallel computing using programmable chips, specifically FPGAs \[24\]. The idea of the proposal is to tailor parallel systems to particular parallel applications. This is achieved by partially reconfiguring the FPGA in order to change the topology of the parallel network and the number/nature of resources in the processing elements as different stages of the application is encountered.

Besides being important and critical to several proposed SEU mitigation techniques, partial reconfiguration offers many advantages such as: in-the-field hardware upgrades and updates to remote sites, runtime reconfiguration, adaptive hardware algorithms, continuous service applications, reduced device count, and more efficient use of available board space \[23\]. The smallest unit of the configuration memory that can be partially reconfigured is a data frame.

There are two methods of partially reconfiguring a Xilinx Virtex:

- **Module-Based Partial Reconfiguration**: used when communication is required between two or more modules. A special bus macro allows signals to cross over a partial reconfiguration boundary.
Difference-Based Partial Reconfiguration: this is accomplished by making changes to the design, usually done using software tool known as ‘FPGA Editor’, and then generating a bitstream based on the differences in the two designs.
Chapter 4 – SEU Detection and Correction

A number of existing and proposed approaches in detecting single-event upsets (SEU) in SRAM-based FPGAs require software support. For instance, to protect designs in FPGAs from SEU using triple modular redundancy (TMR), Xilinx provides a software tool ‘Xilinx TMRTool’ or generally known as XTMR\[^{37}\], that converts HDL (hardware description language) designs into TMR designs before they are loaded or configured into an FPGA. This means that each time the design source code is modified, XTMR would be needed to regenerate an SEU tolerant design before the FPGA is configured.

Other SEU mitigation techniques such as ‘readback’ with partial reconfiguration \[^{4}\][\(^{6}\], scrubbing \[^{8}\], and attachment of error correction codes (ECC) to the configuration data \[^{25}\] all require full software support. In other words, SEU detection is not inherent to the FPGA architecture. It is general knowledge that if an operation is to be performed frequently and repeatedly, the fastest and most efficient method of performing such an operation would incorporate hardware solution rather than software solution. The inefficiency of a software solution is highlighted by the SEU detection method that attaches an ECC to each data frame of the configuration memory. It then repeatedly recomputes the parity of each frame (including the original ECC) while it is performing a readback. A syndrome flag is set if an error is detected which would trigger a partial reconfiguration of the affected frame(s) \[^{25}\]. The process would be less cumbersome if the parity checker is inherent in the configuration memory. This would eliminate the need to repeatedly readback and would also perform the parity checking quicker. This motivation led to the SEU detection method proposed.
Concept

The concept is to add parity trees to individual data frames of the configuration memory. If an odd-number of SEUs occur within that frame, then a syndrome flag for that particular frame is set to indicate a fault. When a frame is affected, determined by the syndrome flag, it is partially reconfigured using the original bit stream for that particular frame. Hence, a copy of the original configuration bitstream has to be available on-board in flash memory such as EPROM. The system can also be partially reconfigured from a remote location if the device supports it. The critical requirements considered in developing a solution are:

➢ It should not affect the normal operation of the FPGA.
➢ It should improve on fault detection latency in 'readback' method available in Xilinx Virtex FPGAs.
➢ There must be an area improvement on triple modular redundancy solution.

In order to meet the first requirement, the structure of the SRAM cells has to be modified. Dual-port 8-T SRAM cells provide the capability of error checking on one dedicated port while the second port is used for normal FPGA operations.

Readback method requires that all frames in the FPGA be read serially. Therefore the complexity of the error detection latency is $O(N)$, where $N$ is the number of frames. This latency can be reduced by parallelizing the detection of SEUs in different frames. It will be shown later in this chapter how this parallelization can be achieved and hence, improved fault detection latency over the readback method.
A simple mathematical approach in proving that using parity trees improves on the area overhead incurred when TMR is used is a transistor count. The number of XOR gates required to scan one data frame is the number of bits in that frame minus one (N-1) and a typical CMOS XOR gate requires six transistors\textsuperscript{26}. When the six transistors is added to the two extra access transistors for a dual-port SRAM, eight additional transistors is required to detect any fault in any particular frame bit. Hence, a theoretical 133\% increase in transistor count will be required. Considering TMR generally introduces a minimum of 200\% area overhead, this would result in a theoretical 34\% improvement in area overhead.

### Original Architecture

The configuration memory of an FPGA is laid out in a rectangular fashion similar to what is represented in Figure 3-7. Each column contains configuration bits that control one of CLBs, Block SelectRAM Interconnect/Content and I/O Blocks. Each column contains a certain number of data frames as was described in Chapter 3. For instance, the block SelectRAM Content column in Figure 3-7 has 64 data frames, while the CLB columns have 48 frames. A block diagram depiction of a column with 64 frames is shown in Figure 4-1 below. Within each frame, the number of words varies depending on the device, as summarized in Table 3-1.

Figure 4-2 is the SRAM layout of a 1-word-frame column. Each word in a frame has a common word line (WL) and a common active-low write signal (RW). The bit line (BL) and its compliment (BL\_BAR) runs across the entire column. Hence:
> Only one frame in a column can be written to or read from at any particular time.

> Read and Write operations can occur in parallel between columns.

Figure 4-1: Block diagram depiction of a configuration memory column
Figure 4-2: Layout of a 32-bit word Configuration Memory
Proposed Architecture

Dual-Port SRAM Cells

In order to enable the FPGA to operate normally while the system scans itself for any single event upsets, the SRAM cell structure has to be modified. Dual-port SRAM cell affords this capability. The first port, which would henceforth be referred to as Port 1, is responsible for normal FPGA functionality. The typical functions performed in FPGA configuration memory are configuration of its resources and partial reconfiguration operations. The second port is exclusively responsible for error checking and detection and would be referred to as Port 2 in this document. It should be noted that Port 2 can only perform READ operations. Any writing to the configuration memory, also known as configuration of resources (or partial reconfiguration), is strictly performed through Port 1. Other memory operations in Xilinx Virtex FPGAs are also strictly executed through Port 1.

Figure 4-3 shows the modified architecture with dual-port SRAM cells replacing the original single port cells depicted in Figure 4-2. As in the diagram for the original architecture, Figure 4-2, it shows 64 1-word frames column of a configuration memory. It can be seen that each cell has two word lines (WL0 & WL1) and two bit lines and their complements (B0, B1, BBAR0 & BBAR1). The dual-port cells allow for simultaneous reading but not writing. When the cells are being written from Port 1, writing and reading from Port 2 is not allowed. When Port 1 is being read, Port 2 can concurrently perform its single-event faults scanning. The next section describes the parity tree used for error detection that would be directly connected to Port 2.
Figure 4-3: Modified layout of a 32-bit word Configuration Memory using Dual-Port SRAM Cells.
Parity Tree

Parity checking bit is a common technique for error checking in a number of applications. A parity bit is a binary digit that indicates whether the number of bits with value of one in a given set of bits is even or odd. Parity bits are used as the simplest error detecting code. There are two types of parity bits: even parity bit and odd parity bit. Even parity bit is set to 1 if number of ones in given set of bits is odd (making the number of ones even). Odd parity bit is set to 1 if number of ones in given set of bits is even (making the number of ones odd)\[^{27}\]. In many applications as will be discussed later, an additional parity bit is attached to the original bitstream. This is illustrated for a 7-bit bitstream that becomes 8 bits with the parity bit included in Table 4-1.

<table>
<thead>
<tr>
<th>7 bits of data</th>
<th>Byte with parity bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>even</td>
</tr>
<tr>
<td>0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>1010001</td>
<td>10100011</td>
</tr>
<tr>
<td>1101001</td>
<td>11010010</td>
</tr>
<tr>
<td>1111111</td>
<td>11111110</td>
</tr>
</tbody>
</table>

Table 4-1: Parity bit example

If odd number of bits (including the parity bit) is changed in transmission of a set of bits then parity bit will be incorrect and will thus indicate that an error in transmission has occurred. Therefore, parity bit is an error detecting code, but is not an error correcting code as there is no way to determine which bit is corrupted. It can only detect an odd number of bit flips. The major advantage of parity checking is that it is the best error detection code that uses only a single bit of storage and it is generated easily using only a
few XOR gates. Some applications of parity checking include detection of transmission errors in SCSI buses \(^{28}\), parity protection for instruction cache in microprocessors \(^{29}\), and error detection in high-speed serial communication \(^{30}\).

To compute the parity of a set of bits, XOR gates can be connected in a number of different ways, including serially or forming a binary tree. The type of binary tree used varies from application to application, depending on the critical constraint of the application. If the critical constraint is layout area, then it makes sense in most cases to use a serial parity tree. However, if the top constraint is timing, then the most efficient parity tree is a binary tree, which has a logarithmic timing complexity. The tree on the left of Figure 4-4 is a serial parity tree for an 8-bit code while the tree on the right is the binary tree for the same 8-bit code. They both require seven XOR gates. This means for an N-bit \(^1\) code, N-1 XOR gates are required to compute its parity.

\[\text{Figure 4-4: Serial and Binary Parity Trees}\]

\(^1\) N = any integer.
A binary parity tree is used for this thesis because of its speed efficiency. Each 32-bit SRAM word includes a binary tree of XOR gates directly connected to their cells' Port 2. Each 32-bit word in a frame then outputs a single bit parity signal. Another parity tree is then constructed for all the parity signals for each word, and hence the parity of an entire frame can be known. This is illustrated in Figure 4-7 below.

Error Detection Block

To determine if a fault has occurred in any particular frame, the current scan sequence parity bit has to be compared with the latest previous scan parity value. This is accomplished by serially linking two latches (forming a shift register) and enabled by the word line (WL) AND the ‘read or write’ signal (RW). The previous scan parity value in the first latch is shifted to the second latch while the current parity value is latched onto the first. If the two values do not agree, then an odd number of bit-flips must have occurred in that frame. The comparison is achieved by XORing the output of the two latches. If the value in the first latch does not match that in the second latch, then the ‘Fault’ signal is set, indicating a SEU occurred in that frame. The circuit of the detection block is illustrated in Figure 4-5 and each frame would have its own dedicated error detection block.

![Error Detection Circuit](image)
Error Detection Control

To successfully scan the entire FPGA and detect faults using parity checking, the complete circuit for a new FPGA configuration memory architecture is illustrated below in Figure 4-7. It should be pointed out that Figure 4-7 represents a single column (with multiple frames) FPGA with its control logic. The error detection circuit including its control logic is then developed as follows:

1. The Port 2 word lines (WL1) of every word in each frame are connected.

2. Two important facts/requirements necessary to develop the control logic are:
   
   a. The maximum number of frames within a column of configuration memory in Xilinx Virtex FPGAs is known to be 64 as is implied in Table 3-2 and Figure 3-7 in Chapter 3.
   
   b. Error detection can occur in parallel between configuration memory columns.

3. The main control circuitry for error detection includes: a 6-bit synchronous counter, a 6-to-64 decoder, an OR-gate and a circuit to determine if the error detection scan should be enabled or disabled.

   a. The 6-bit output of the 6-bit counter is connected to the 6-bit input of the 6-to-64 decoder. The counter addresses different frames within a column. The counter has an ‘enable’ signal which stops the count when it is deasserted. The counter retains its last count when it is disenbled. It starts from its last count when the enable signal is re-asserted.
b. Each bit of the 64 outputs of the decoder is connected to its corresponding word line in each column, across the entire memory. For instance, output bit 0 of the decoder is connected to WL1_0 of every column in the memory. If there are 10 columns, then output bit 0 of the decoder would be driving 10 inputs. Hence, a critical constraint for the design of the decoder is that it has at least a fanout of 10. It should also be noted that not all columns have 64 frames as can be deduced from Table 3-2. Therefore, some output bits of the decoder might not be connected to any word lines in some columns of the configuration memory.

c. The circuitry labeled “Enable Controller” in Figure 4-6 outputs the enable signal that controls the counter. At startup or after a reconfiguration, it would require two clock cycles for the error detection block to stabilize; therefore the fault signal should be ignored until after the second clock cycle. This is achieved by using the two-latch shift register in Figure 4-6 with the input tied to VDD. A genuine fault occurs when the output of the shift register is logic 1 and the fault signal is asserted. The enable signal is then deasserted when a genuine fault occurs. The enable signal becomes asserted when the ‘Resolved’ signal is pulsed. The overall function of the enable controller is to stop the scan when a fault is detected and restart it after partial reconfiguration, determined by a pulse sent to the ‘Resolved’ signal.
d. The OR-gate in Figure 4-7 determines if at least one frame in a column detected a SEU. The inputs are the fault syndrome signals from individual frames within that column. The output is fed back to the 'Enable Controller' circuitry.

4. When the clock for the synchronous counter is enabled, the 6-bit counter starts counting from 0 to 63. The decoder then enables the word lines depending on the count or address at the counter's output. For instance, if the output of the counter is 0, then bit 0 output of the decoder would be asserted, enabling the word line for Port 2 of all 'frame zeros' in the memory. (Each column has a frame zero).

5. When a word line of Port 2 is enabled, and the RW signal controlled by the FPGA is asserted (meaning the FPGA is in 'read' mode), then the values stored in the cells are output to the bit lines of Port 2. The parity for that frame is then computed by the parity tree. The detection block then latches the overall parity of each frame.
6. After the second cycle, the detection block compares the parity computed for each frame in the previous cycle with the parity computed in the current cycle. If the parity values do not agree, then a SEU has occurred in that frame.

7. When a fault occurs in any frame, the scan is immediately halted by the enable controller block.

8. The software controller responsible for partial reconfiguration that will be described in a later section handles the fault. When it completes reconfiguration of any affected frame(s), it sends a pulse to the ‘Resolved’ input of the ‘Enable Controller’.

9. If the ‘Resolved’ signal receives a pulse, the scan circuitry is quickly re-enabled and normal scan operation resumes.

10. For a multiple column FPGA, the fault signals of every column are ORed and the output fed back to the ‘Enable Controller’ circuitry as depicted in Figure 4-8.

11. Partial reconfiguration requires the address of the exact frame that needs to be reconfigured. The two important addresses are the frame address and the column address.

   a. To determine the column address, the fault signals of each column is encoded into an 8-bit address. The maximum number of columns in a Xilinx Virtex-4 is just over 158 as can be seen in the ‘Row x Col’ column of Table 3-1 (it does not include the additional I/O Block columns, Block SelectRAM Interconnect and Content columns). It is therefore assumed that the possible maximum number of columns is
256, and hence an 8-bit address is needed. If a fault occurred in column 10, then the column address will read x0A.

b. As described in step 7 above, when a fault occurs, the control circuitry is halted. This means the 6-bit counter stops counting, but retains its previous count. This count represents the address of the faulty frame. This is fully illustrated in Figure 4-8 below.
Figure 4-7: Modified FPGA Configuration Memory with Fault Detection capability

(A Single Column FPGA)
Figure 4-8: Complete SEU Detection Controller
Design Constraints

A few design constraints of note for the controller include:

- The clock enabling the 6-bit synchronous counter, decoder and ultimately the word lines has to meet setup and hold time requirements of the latches in the fault detection unit.

- Each output bit of the decoder has to drive one word line (WL) in every column of the configuration memory. In other words, if there are 256 columns, then the fanout of each output bit of the decoder has to be 256. Appropriate drivers would have to be designed to meet fanout requirements across the FPGA.

- It is expected that the entire design, including the parity tree, controllers, and error detection unit would be Complementary MOS design. However, pseudo-NMOS design can be used in some digital circuits (e.g. OR gates) to reduce transistor count.

SEU Correction

The next step after a fault has been detected in any frame is to correct the fault by partially reconfiguring the affected frame. Due to the highly unlikely occurrence of multiple SEUs in one frame within one scan cycle, it is safe to assume that only one frame would need to be partially reconfigured at any given cycle. This is backed up with the discussion in Chapter 2 on the rate of SEU occurrence; which shows a very small chance of multiple SEUs occurring within a 64 KB CMOS SRAM in one day. 

45
The existing process of partial reconfiguration after an error has been detected with the readback method is as follows:

**Partial Read/Write Operations**

According to [4], to write a series of data frames, the ‘Frame Address Register (FAR)’ must first be set to the address of the first frame in the series. The next step is to specify the number of words in that frame and then load the original bitstream for that frame into the ‘Frame Data Register Input (FDRI)’. FDRI is a pipeline input stage for configuration data frames to be stored in the configuration memory [7]. As a reminder, Table 3-1 shows that the number of 32-bit words within a frame varies from device to device. However, there are only a set number of words within frames in any particular device. Additional information on configuration and partial reconfiguration can be obtained from the following references: [5] [8]. The pertinent information needed to successfully have the FPGA partially reconfigure for the proposed architecture in the same vein as when readback is used for SEU detection is the frame address.

**Virtex Configuration Addressing**

In Xilinx Virtex devices, the configuration memory is divided into columns, but the total address space is divided into two block types: RAM and CLB. The RAM block type include only the SelectRAM content (not interconnect), while the CLB block type include all other columns [21], as shown in Figure 3-7. Both address spaces are subdivided into major and minor addressing. Major addressing represents the addresses of columns while minor addressing represents the addresses of frames. Each column has a unique
major address within its address space (RAM or CLB) while each frame has a unique address within its column.

**Major Address**

The CLB address space begins with ‘0’ for the center frame and then alternates in a ping-pong fashion around the center frame. Even number addresses are to the left of the center column while odd number addresses are to its right. The addresses increase from the center to the leftmost and rightmost columns. The RAM address space has ‘0’ for the left block SelectRAM content and ‘1’ for the right block SelectRAM content column. The bottom row of Figure 4-9 shows the major addressing for a Xilinx device (XCV50). The shaded portion is the RAM address space. This addressing scheme varies slightly among Virtex families as described in Table 4-2.

![Figure 4-9: Major Addressing of Columns and Address spaces](image-url)
### Column Type, Block Type, Virtex, Virtex-E, Virtex-E Extended Memory

<table>
<thead>
<tr>
<th>Column Type</th>
<th>Block Type</th>
<th>Virtex</th>
<th>Virtex-E</th>
<th>Virtex-E Extended Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>First MJA</td>
<td>CLB</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>RAM</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>RAM</td>
<td>BRAM Content</td>
<td>BRAM Content</td>
<td>BRAM Content</td>
</tr>
</tbody>
</table>

Table 4-2: Virtex Family Major Addressing scheme

### Minor Addressing

The number of frames in each column varies. The center CLB column has 8 frames, other CLB columns have 48 frames and the Interconnect columns have 27 frames. Therefore a minor address of 30 is not valid in the center CLB column and the Interconnect column but valid in other CLB columns.

### SelectMap Interface

Virtex devices can be configured through the SelectMAP interface, master/slave serial interfaces, or the Boundary-Scan interface. For the purposes of this thesis, the SelectMAP interface would be the choice of configuration. The SelectMap interface is an 8-bit bi-directional interface in Virtex devices with data pins labeled D[0:7] (pin D(7) is LSB). It also has other control bits such as BUSY/DOUT, INIT, WRITE, and CS. Virtex devices can be configured to retain all its pins allowing further reconfiguration via those pins or configured as user I/O pins if no reconfiguration is required [21]. These pins will be retained for this thesis to allow for partial reconfiguration.
Virtex Partial Reconfiguration Steps

According to Xilinx Application Note: “Correcting Single-Event Upsets Through Virtex Partial Configuration,” when a SEU is detected through readback in a Virtex device, the following steps are taken to reconfigure the device:

1) **Abort**: An Abort command is issued by holding the CS Low and the WR High for at least three clock cycles. This will reset the SelectMAP and configuration logic so that the interface may be re-synchronized.

2) **Synchronize**: Before a new process can commence the SelectMAP interface must be resynchronized by reloading the Synchronization Word.

3) **Issue Write Access to CMD Register**: Enable write access to the configuration memory array by loading a write command into the CMD (command) register.

4) **Load FAR**: Specify the frame address in the FAR (Frame Address Register) with a major and minor address location.

5) **Access FDRI Register**: Issue a write command to the FDRI (Frame Data Register Input) register specifying the frame data length in 32-bit words plus one 32-bit dummy word.

6) **Load Frame Data**: Load the data frame into the FPGA followed by one dummy frame. Each frame must be followed by a dummy word; however, the bitstream includes these dummy words at the end of each data frame.

7) **Reset CRC**: Issue a RCRC command to the CMD register to clear the CRC register.
8) **Abort:** Although a second Abort command may be superfluous, a resetting of the SelectMAP interface and subsequent resynchronization for any new process increases the likelihood that the process will be successful.

Most of these steps would be needed to partially reconfigure the proposed FPGA architecture proposed. However, calculating CRC is a function of the ‘readback’ detection system and would not be necessary for the proposed architecture. These steps are performed in a controller executed by a PowerPC processor. The controller’s finite state machine would need to be modified in order to function with the proposed architecture. The steps (states in FSM) necessary to perform a partial reconfiguration when a fault is detected is as follows:

**State 0:** This is the power-on phase. When the FPGA is powered on, it is automatically configured. At this stage, the reconfiguration controller is not yet “alive” predictably, but becomes fully functional after the configuration process is completed. At this stage, the ‘Resolved’ input pin would be reset to logic 0. This is done by assigning the appropriate port address (given in Table 4-3) and setting pin D7 of the SelectMAP data bus to logic ‘0’.

**State 1:** The controller polls the global fault I/O pin. The global fault signal would act as an interrupt to the system. This step would represent the idle state in a state machine. The state machine would be triggered into action when the global fault pin is asserted. The controller interfaces with the signal through SelectMAP and the global fault signal would be read from bit D7 (LSB) of the data bus.
State 2: The next step is to read the ‘Column Address’ output port. This is done by assigning the port address binary address “01” (see Table 4-3). When the address port is assigned, the SelectMAP interface reads the value at the output port and stores the value in an internal variable. The column address is 8-bit wide and therefore would utilize all 8-bits of the SelectMAP data bus.

State 3: The next step is to read the frame address. This is achieved by setting the port address to “10”. When the appropriate address is set, the data bus should read the 6-bit ‘Frame Address’ output port with pin D2-D7.

State 4: The controller converts the address read in Step 2 and Step 3 to normal FPGA frame addressing using major and minor notation. Depending on the device and the manufacturer of an FPGA with the modified architecture proposed, two steps can be followed in determining the correct address of ever frame:

i. A look-up table can be used to match addresses from the FPGA address output pins to the actual address. For instance, a column address of 8 and frame address of 20 could represent a major address of 42 and minor address of 20.

ii. An algorithm can be developed to translate the addresses obtained from the address output pins to the minor and major Virtex addressing scheme.

State 5: Normal reconfiguration operations detailed in the previous section would constitute the next state in a state machine for the controller. The normal operation would entail amongst other operations, interfacing between an on-board flash memory such as
EPROM used to store original configuration data and the FPGA device. This step in itself can make up another finite state machine.

**State 6**: Send a pulse to the ‘Resolved’ input pin of the FPGA device. To accomplish this, the appropriate I/O port address has to be set (address “11”), with Pin D0 (MSB) of the data bus set to logic ‘1’.

The state machine that executes these steps is graphically illustrated in Figure 4-12. The configuration controller can be executed from a CPLD, but for Xilinx Virtex devices, a PowerPC processor. The controller interfaces between EPROM memory storing configuration data with the FPGA device through SelectMAP interface as shown in Figure 4-10. The interface is similar to the existing \[34\] interface in Xilinx Virtex FPGAs, with an additional port, ‘Port Address’. The port address determines which I/O port the SelectMAP data bus should be reading or writing to. A closer look at the configuration controller would reveal the interface between memory and SelectRAM interface controlled by a finite state machine as illustrated in Figure 4-11. The state machine needed for partial reconfiguration when the global fault pin indicates a fault in the modified architecture has the steps described above as individual states. It should be noted that Step 5 can be a state machine of its own executing the steps detailed in the “Virtex Partial Reconfiguration Steps” section above.
<table>
<thead>
<tr>
<th>PORT NAME</th>
<th>ADDRESS (2-BIT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL FAULT</td>
<td>00</td>
</tr>
<tr>
<td>COLUMN ADDRESS</td>
<td>01</td>
</tr>
<tr>
<td>FRAME ADDRESS</td>
<td>10</td>
</tr>
<tr>
<td>RECOVERED</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 4-3: I/O Port Addresses

Figure 4-10: Partial Reconfiguration Interface

Figure 4-11: Partial Reconfiguration Controller
Figure 4-12: Partial Reconfiguration Finite State Machine
Chapter 5 – Simulation and Results

A model of a Virtex XCV150 device configuration was developed using Very High Speed Integrated Circuit Hardware Description Language, popularly known as VHDL. This model was modified to the architecture proposed. The two models were simulated using Mentor Graphics HDL simulator: ModelSim and then synthesized using Synopsys synthesis tool: ‘Design Analyzer’. Testbenches were developed for all entities within the models.

Original Model

SRAM Cell: The basic atomic unit in the model is an SRAM cell. A behavioral model of an SRAM cell was developed with reading and writing capability.

SRAM Word: The next level of design is a 32-bit word. This involves connecting the word lines of 32 SRAM cells.

FPGA Frame: An FPGA frame as described in earlier chapters is a 1-bit vertical slice of the configuration memory. The slice could have multiple words. For Virtex XCV150 device, there are 16 words. To model a frame, 16 32-bit words were connected to form a 1x512 SRAM memory.

FPGA Column: The configuration memory is divided into columns. Each column contains a certain number of frames. The number of frames vary depending on the type of column e.g. CLB column, Center column, Interconnect column etc. To model a column, a number of frames were connected together through the bit lines of individual corresponding SRAM cells.
**XCV150**: To model the entire Virtex XCV150 device, different columns were modeled according to guideline in documentation of the device [4]. The device has 36 CLB columns (see Table 3-1 on page 24); a center column; two I/O block columns; two interconnect columns; and two SelectRAM Content columns. Each of these columns was modeled and all connected to the detection controller in one top layer design entity.

**Proposed Model**

**Dual-Port SRAM Cell**: The original single-port SRAM cell was modified into a dual-port cell. The new cell has two word lines and bit lines. The behavioral model of the reading and writing operations strictly adhere to the rules for dual-port SRAM access.

**Dual-Port SRAM Word**: Besides the dual ports, the other changes made to the original SRAM word include adding a parity binary tree to each word (see Figure 4-4 on page 36). Figure 5-1 is the snapshot of the synthesis circuit for the new SRAM word. The cells are arranged as the leftmost vertical units. A visible binary tree of XOR gates (used for parity calculation) then develops to the right. The output port seen on the right is the parity of the 32-bit word.

![Figure 5-1: Synthesis result of new SRAM Word](image)
**New Frame:** Another binary tree of XOR gates computes the parity of the frame. Each word has a parity output; therefore another tree is needed to obtain the parity of the frame. Another feature of the new frame is an ‘SEU Detection Block’ shown in Figure 4-5 on page 37. Figure 5-2 is a snapshot of the synthesis result. The frame has 16 words lined up at the top of the figure. An XOR parity tree then converges the parity for that frame. The bottommost unit is the SEU Detection Circuit and is fed by the frame parity; the word line of Port 2 and the ‘RW’ (read or write) signal for control.

![Figure 5-2: Synthesis result of new FPGA Frame](image)

**New Column:** Each frame in a column has its own syndrome output. The syndrome is asserted if an odd number of SEUs occur within that frame. To determine the fault syndrome of an entire column, all frame syndromes are fed into an N-input OR-gate (where N is the number of frames in that column). This was modeled for various types of columns (CLB, Center, Interconnect, Content and IOBs).

**Fault Detection Controller:** The control described in Chapter 4 for detecting SEU in the configuration memory includes a 6-bit counter; a 6-to-64 decoder; and circuitry that determines when to disable the controller labeled as ‘Enable Controller’ (see Figure 4-6 on page 40 for more details). The output of the decoder is then connected to the word line input of each frame within a column (WL1). This controller was also modeled in VHDL and the synthesis result is shown below in Figure 5-3. For a better illustration of how this
controller fits into the entire design, see Figure 4-7 and Figure 4-8 on page 43 and 44 respectively.

![Figure 5-3: Synthesis result of SEU Detection Controller](image)

**Proposed XCV150:** The modified SRAM Cell, SRAM Word, Frame, Columns were all connected together along with the controller exactly as shown in Figure 4-8. The column fault signals were ORed to obtain a global FPGA fault signal. This signal is then fed back to the Fault input of the controller illustrated in Figure 5-3.

**Simulation Waveforms**

This section would present the ModelSim simulation waveforms obtained from the models of key components. This demonstrates the functional correctness of the idea proposed.

**Fault Detection**

For simplicity, the model used to illustrate functional correctness of the error detection scheme is an 8-frame (one 8-bit word per frame) column. The idea behind the simulation is to inject fault into an already configured configuration memory. The syndrome flag for each frame should be asserted one clock cycle after the fault has been injected. The fault signal should reset back to logic ‘0’ after the fault has been handled. Figure 5-4 is the waveform obtained for the 8-frame column. Description of events with
timeline follows the figure. Each scan cycle for this model is 8 clock cycles of clock with 10 ns period.

Figure 5-4: Fault Detection simulation waveform for a single word, eight frame column

0 to 80 ns: This is the configuration (writing to memory) phase. Data is written to the configuration memory through Port 1.

80 to 240 ns: The configuration memory goes through two scan sequences (or in SRAM lingo, read sequences). It takes at least two clock cycles for a valid fault to be detected,
because the parity value from a previous scan is compared to the current parity value. The detection circuit stabilizes after two scan sequences and would then produce a valid fault signal after every clock cycle.

240 to 320 ns: The fault signal for each frame shows no faults occurring as expected. If any faults occurred, then the fault signal would show a value other than x00.

320 to 400 ns: In this phase, single bit flip errors are written to each frame of the configuration memory. In the next clock cycle, the fault signal is expected to indicate the errors.

400 to 560 ns: The fault signal indicates a fault in every frame. The two cycles of this phase is required for the system to stabilize again. An important point here is that if the controller were included in this model, it would immediately stop the scanning of errors until the fault has been handled. Without stopping the scan, like is done in the waveform above, after two clock cycles, the fault signal would stabilize again and show x00 value. The fault would then not have been handled, and it would become a permanent fault. This is to be avoided, and that is why the controller that would be shown next would stop the scan sequence as soon as a fault is detected.

560 ns and later: Predictably, the fault signal resets to x00 indicating no fault occurred.

The important point presented with the waveform in Figure 5-4 is that single event upsets can be detected with the FPGA architecture proposed. The entire design is however not represented in that waveform for the sake of simplicity. The next two waveforms would show fully how faults detected would be handled.
SEU Detection Controller

The function of the controller is to start a counter that addresses different frames within a column. When each frame is addressed, its parity is taken and compared to its previous parity. If a fault is detected, the controller is to be disabled by stopping the counter. The counter however retains its last count. That count indicates which frame the fault occurred in. For the counter to be re-enabled, a pulse to the ‘Resolved’ bit is required. This pulse is sent from the reconfiguration controller after a partial reconfiguration of the affected frame has been completed. When the counter is re-enabled, it starts counting from its last count before the interruption.

Figure 5-5 is the simulation waveform of the controller. The timeline of events is as follow:

0 to 40 ns: The ‘enable’ signal is high and the counter addresses different frames after every clock cycle (‘frame_add’). The ‘fault’ signal is low indicating no fault.

@ 40 ns: The ‘fault’ signal goes high, indicating a fault has been detected in frame 4. Immediately, the ‘enable’ signal goes low, stopping the counter at count ‘04’.

@ 60 ns: Two clock cycles after the fault is detected, frame 4 is partially reconfigured and the ‘Resolved’ signal is sent high indicating the fault has been handled. Note: in reality, partial reconfiguration would require more than two clock cycles. This simulation is just illustrating functionality.

@ 70 ns: The counter is re-enabled (‘enable’ signal goes high) and frame 5, 6 and so on are addressed.
Partial Reconfiguration Controller

A finite state machine that would be executed on an on-board processor controls the partial reconfiguration process. The finite state machine is detailed in Chapter 4 and illustrated in Figure 4-12. The key point is that the SelectMAP interface data bus reads the I/O ports for frame and column addresses and also sends out a pulse to the ‘Resolved’ input port when the reconfiguration is completed. The detail of the waveform in Figure 5-6 is next:

**STEP 0:** The MSB of the data bus ‘d’ is set to 0. The address bus is also ‘11’ indicating that the ‘Resolved’ input pin is reading from the bus (see Table 4-3 for I/O port address). This step is the power on state, so the ‘global fault’ signal is not read being read yet.

**STEP 1:** At the following clock cycle, the port address becomes “00” indicating that the data bus is reading the ‘global fault’ output port. The data bus reads this signal through its LSB as can be seen. At the next clock cycle, the ‘global fault’ signal becomes one, which indicates a fault has been detected. This triggers the reconfiguration process.
STEP 2: This step reads the column address port. The port address is addressed “01” which causes the data bus to read the ‘Column Address’ output port. The internal variable labeled ‘col_add’ in the waveform stores the address read from the bus.

STEP 3: The next state reads the frame address port. The port address is correctly addressed to “10” and so the data bus reads the ‘Frame Address’ output port. The frame address is then stored in internal variable labeled ‘frm_add’ on the waveform.

STEP 4: The column and frame addresses previously read is converted to major (‘major_add’) and minor (‘minor_add’) FPGA addresses respectively.

STEP 5: It is this state that the actual reconfiguration is performed. The waveform shows nothing happening in this state because it entails the same procedure currently used in Xilinx Virtex devices.

STEP 6: If the partial reconfiguration is completed successfully in step 5, then the ‘Resolved’ signal needs to see a pulse. The port address is again set to “11” and the MSB of the data bus is set to logic ‘1’. This is read by the addressed ‘Resolved’ input port.

STEP 7: The address port is still addressed “11” so data can be written to the ‘Resolved’ input port. Logic ‘0’ is then written to the MSB of the data bus and this then pulls the ‘Resolved’ signal low. Also, the ‘global fault’ signal shows no fault (because it has just been handled) and the state machine transitions to Step 1 (the idle state). At Step 1, the port address is “00” addressing ‘Global Fault’ output port. The LSB of the data bus continuously reads the ‘Global Fault’ signal. The state machine only transitions to Step 2 if another fault is detected.
Properties of Proposed Architecture

The properties that would be examined and used in comparison to existing methods of SEU detection and correction are area, timing and power dissipation. Each property would be analyzed based on mathematical (theoretical) expectations and results from synthesis reports.

Area – Mathematical

Original Architecture: The transistor count would be used to calculate the mathematical area. A typical SRAM cell has six transistors\textsuperscript{[20]} Therefore, for Virtex XCV150:

- 1 word is 32 bits: $6 \times 32 = 192$ transistors per word.
- 16 words per frame: $192 \times 16 = 3072$ transistors per frame
- 2029 frames: $3072 \times 2029 = 6233088$ transistors

Figure 5-6: Partial Reconfiguration Controller’s Finite State Machine simulation waveform
Proposed Architecture: The transistor count would include the two additional access transistors in the dual-port SRAM cell. It would also include the XOR parity tree and the fault detection circuitry. A typical CMOS XOR gate has a transistor count of six [24] and a typical latch has a transistor count of six. The transistor count is derived as follows:

- 1 dual-port SRAM cell has 8 transistors
- 1 SRAM word is 32-bit: $8 \times 32 = 256$ transistors per word
- 16 words per frame: $256 \times 16 = 4096$ transistors per frame
- 512 SRAM cells per frame require 511 XOR gates for a parity tree: $6 \times 511 = 3066$ XOR transistors per frame.
- Fault Detection Circuit has two latches and an XOR gate: $(2 \times 6) + (1 \times 6) = 18$
- Total number of transistors per frame: $4096 + 3066 + 18 = 7180$ transistors per frame.
- 2029 Frames: $7180 \times 2029 = 14568220$ transistors

The percentage increase in transistor count between the original XCV150 and the proposed XCV150 is:

$$\% \text{Increase} = \left( \frac{\text{Proposed} - \text{Original}}{\text{Original}} \right) \times 100 = \left( \frac{14568220 - 6233088}{6233088} \right) \times 100 \approx 134\%$$

Equation 1

Note: This is an ideal calculation and does not take into account the additional control logic for SEU detection and correction. It is not expected that the control logic would significantly change this estimate of 134% increase in transistor count.

Area – Synthesis Report

The synthesis result of the VHDL models was obtained from Synopsys synthesis tool: “Design Analyzer”. The process used for synthesis is TSMC 0.35 µm technology
and simulation was performed at worst case conditions. The area data collected does not include interconnects area but just the cell area. Table 5-1 is the summary of area information collected for various key units in the original and new models. The result shows an approximately 127% increase in area between the original FPGA architecture and the proposed architecture proposed for Virtex XCV150 device.

<table>
<thead>
<tr>
<th>UNIT</th>
<th>BAREBONE</th>
<th>PROPOSED</th>
<th>% INCREASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM Cell</td>
<td>225</td>
<td>474</td>
<td>110.67</td>
</tr>
<tr>
<td>SRAM Word</td>
<td>7225</td>
<td>16035</td>
<td>121.94</td>
</tr>
<tr>
<td>Frame</td>
<td>113044</td>
<td>256989</td>
<td>127.34</td>
</tr>
<tr>
<td>Controller + Miscellaneous</td>
<td>0</td>
<td>4137</td>
<td>N/A</td>
</tr>
<tr>
<td>XCV150</td>
<td>229366276</td>
<td>521434818</td>
<td>127.34</td>
</tr>
</tbody>
</table>

Table 5-1: Summary of percentage increase in area from synthesis report

**Timing – Mathematical**

The timing analysis will be based on the minimum clock period that will be required for an error to be detected in one frame of the configuration memory. This model is for the ideal situation and does not take into account all non-trivial timing components. The formula to calculate the clock period is developed as follows:

Let:

- \( N \) = Number of bits per word
- \( T_d \) = XOR gate delay
- \( A_T \) = SRAM Read Access Time
- \( W \) = Number of words per frame
To access the data stored in SRAM cells within a frame, the access time \(A_T\) needs to be considered.

All SRAM cells within a frame can be accessed in parallel; therefore the overall access time remains \(A_T\).

The time complexity of a binary tree is \(O(\log_2 N)\). Therefore, the arrival time of the parity signal of a 32-bit word is: \(T_d \times \log_2 (32) = 5T_d\).

The arrival time of the parity tree for \(W\) words is: \(T_d \times \log_2 (W)\).

Therefore the arrival time of the parity signal for an entire frame is: \(T_d \times (5 + \log_2 N)\).

The overall minimum clock period required to scan individual frames becomes:

\[
\text{Scan Time} = \left[ A_T + T_d(5 + \log_2 W) \right] \times 2 \quad \text{Equation 2}
\]

For an SRAM with a typical access time of 15ns \((A_T = 15\text{ ns})\) and a typical CMOS XOR gate delay of 170 ps \((T_d = 170\text{ ps})\) and for XCV150 with 16 words per frame \((W = 16)\), then the minimum required clock period for the scan sequence is: 33 ns or frequency of 30.3 MHz. To scan the entire FPGA, the bottleneck of the scan time will be the column with the largest number of frames because all columns will be scanned in parallel. It would then require 64 clock cycles between scans for SEU in Virtex XCV150.

**Timing – Synthesis Report**

The timing report summarized in Table 5-2 is derived from the gate delay of the critical path of each unit after synthesis. The overall minimum clock period is 12.5 ns or a minimum frequency of 80 MHz. It would require 64 clock cycles for a complete SEU detection scan sequence or 0.8 \(\mu\)s.
Partial Reconfiguration Time

After fault detection, the next sequence of event is correction through partial reconfiguration. Following guidelines in [21][22], for XCV150 it would require 0.8 µs to partially reconfigure one frame, where a byte of data is written every clock cycle at 80 MHz. It should be noted that this calculation does not take into account the number of clock cycles required for partial reconfiguration setup. However, if Virtex XCV150 was developed with the architecture proposed in TSMC 0.35 µm technology, it would require approximately 1.6 µs to detect and correct any faults due to single event upsets.

Power Dissipation Analysis

The static power dissipation due to leakage current, subthreshold current and substrate current in CMOS is negligible in an ideal case. Dynamic dissipation due to transient switching behavior and capacitive load are however the principal source of power dissipation. The general mathematical characteristic of dynamic power dissipation is Equation 3. C represents the load capacitance and f represents the frequency of the
dynamic circuit. Both the load capacitance and the frequency are directly proportional to
the power dissipation, therefore an increase or decrease in either value would have a
corresponding effect on the power. Heavy loads and high fanout would increase the load
 capacitance, and should therefore be avoided.

\[ P = (CV_{dd}^2) \times f \]  \hspace{1cm} \text{Equation 3}

The simulation results from the Synopsys synthesis tool ‘Design Compiler’ show
an ideal zero static power dissipation. With TSMC 0.35 \( \mu \)m process, the ratio of dynamic
power between the modeled original FPGA configuration memory and the proposed
memory is approximately 2. This represents a 100\% increase in power dissipation. Table
5-3 is the summary of the dynamic power for various units of the modeled original (bare
bone) and proposed FPGA configuration memory. Besides, power loss due to the error
detection technique, there will also be power loss due to partial reconfiguration. Research
has been done to estimate the power loss due to reconfiguration \(^{[44]}\).

<table>
<thead>
<tr>
<th>UNIT</th>
<th>BAREBONE</th>
<th>PROPOSED</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM Cell</td>
<td>20.85 ( \mu )W</td>
<td>41.68 ( \mu )W</td>
<td>1.999</td>
</tr>
<tr>
<td>SRAM Word</td>
<td>0.667 mW</td>
<td>1.36 mW</td>
<td>2.03</td>
</tr>
<tr>
<td>Frame</td>
<td>10.27 mW</td>
<td>21.76 mW</td>
<td>2.11</td>
</tr>
</tbody>
</table>

Table 5-3: Power Dissipation Ratio of Barebone and Proposed Models

**Comparison with Existing Methods**

There are many proposals \([1, 2, 3, 4, 5, 39]\) for designing SEU tolerant SRAM-
based FPGAs. The most common methods include: Readback \(^{[4]}\)\(^{[7]}\) and Triple Modular
Redundancy \(^{[5]}\)\(^{[37]}\). These two methods will form the basis of comparison with the
proposed architecture. There are several variations of implementation method for TMR that have been proposed \[2][17][39]. XTMR with scrubbing \[37] will serve as the standard for comparisons. The comparison will be an analysis of the advantages and disadvantages of the two error correction and detection methods over the architecture proposed.

**Readback with Partial Reconfiguration**

The concept is to detect when there is a SEU in the configuration memory by continuously reading back its content frame-by-frame and performing bit-by-bit compare with the original configuration bitstream stored in flash memory. Readback is not a valid operation for block RAM which stores user data while the system is fully functional. This is because its content can be corrupted if it is being written to by a user program at the same time as when readback is being performed. In other words, simultaneous read and write is not permitted. Thus, error detection in block RAM has to be performed using error correction codes (ECC) like checksum or cyclic redundancy check (CRC) \[43].

In some designs, the look-up tables (LUT) can also be used as RAM \[45]. For this case, care must be taken when reading back the frames involved in order to prevent data corruption. For instance, the readback clock can be disabled while running a design that uses LUTs as RAM \[44]. This access conflict is solved by using dual-ported SRAM cells, as is the case in the proposed architecture. When LUTs are used as RAM, user design changes the content of the involved frames. This means a change in value of any cell involved does not necessarily represent a fault. To combat reporting “false” fault, readback uses a mask file to mask configuration data from certain frames. As with block RAM frames, fault detection would have to be performed through ECC.
The readback process begins by the entire configuration memory being read and stored as a readback file in memory such as SRAM or DRAM. For each frame, if the mask file indicates no mask is required, then a bit-by-bit comparison is performed. If there is disagreement between any two corresponding bits, then partial reconfiguration of the affected frame is triggered. This process requires three files of similar sizes: the original bitmap file, the mask file, and the readback file \([4][7]\).

**Advantages**

- Its main advantage is the minimal FPGA core area overhead incurred. Even though hardware implementation of algorithms for reading and evaluating each data frame would be needed, the additional area to actual FPGA core is minimal compared with the proposed architecture.

**Disadvantages**

- It effectively triples the amount of system memory required because of the three files needed for error detection \([4]\). A high end Virtex FPGA has over 16 million configuration bits. This translates to 16 MB of memory to store each one of the processing files. This is not a desirable overhead for space program devices because memory is expensive and board space is premium. Only the original configuration bitstream is required for the proposed architecture.

- The error detection latency will be higher because of two key factors:
  - All frames are read sequentially for readback method. The proposed architecture offers parallel read for corresponding frames in different columns.
Additional latency is introduced by the memory access time of the original bitstream file and the mask file.

The higher error detection latency is supported by the following mathematical analysis.

- If the number of frames in a device is \(F\) and the number of 32-bit words per frame is \(W\), then it would require \([21][22]\) \(F \times W \times 4\) clock cycles to readback. Where the ‘4’ multiplicand represents 4 clock cycles for 4 bytes per word. Therefore XCV150 would require \(1898 \times 16 \times 4 = 113880\) clock cycles or 1.4 ms at 80 MHz to complete a readback sequence.

- To reconfigure one frame if an error is detected, it would require \(W \times 4\) clock cycles \((16 \times 4 = 64)\) or 0.8 \(\mu s\) at 80 MHz, following the guideline in \([21][22]\).

- Therefore, a complete detection and correction sequence would take approximately 1.42 ms. This is significantly (over 1000%) worse than the time it would take to detect and correct SEUs using the proposed architecture under similar conditions.

- The “Timing – Synthesis Report” section earlier in this chapter shows it would take 1.2 \(\mu s\) at 80 MHz to detect and correct a fault in any given frame for Virtex XCV150.

**Triple Modular Redundancy**

The basic concept of triple modular redundancy is to have three redundant copies of a design run in parallel. The final output is determined by a majority vote between the three modules. Xilinx has developed a tool known as “XTMRTool”\([37]\) that automatically
transforms a non-TMR design to a TMR design for their FPGAs. XTMR also triplicates the majority voter logic in order to achieve better coverage as shown in the before and after design in Figure 5-7. There are proposals to combine TMR with scrubbing (periodically reconfiguring the FPGA). The scrubbing rate would be dependent on the device and its operating environment.

According to [4], "a good rule of thumb is to place the scrub rate at one order of magnitude from the upset rate. In other words, the system should scrub, on average, ten times between upsets." Therefore, if an upset rate of one per hour is assumed, then there should be 10 scrub times in the space of an hour or every six minutes. At 80 MHz, the scrub latency for Xilinx Virtex XCV150 will be approximately 1.6 ms. Scrubbing would also contribute to the power dissipation of the FPGA. It has been shown that the rate of SEU occurrence in space is about one per day, increasing to over ten per day with the occurrence of solar flares.

![Design Before XTMR](image1)

![Design After XTMR](image2)

*Redundant domains converge on PCB trace*

Figure 5-7: Xilinx TMRTool before and after design
**Advantages**

- There is no error detection latency because faults are not detected but masked. It is therefore faster to “handle” an SEU with TMR than the parity checker + partial reconfiguration proposed. It should be noted that TMR does affect system performance due to the majority voting circuitry.

**Disadvantages**

- TMR has static fault coverage while SEUs are dynamic faults. In other words, if SEUs at different times affects two modules of a TMR design, then it becomes a permanent fault until the FPGA is reconfigured. This cannot happen with the proposed architecture because the configuration memory is refreshed as soon as an SEU occurs, Scrubbing\[4\] the configuration memory periodically would eliminate such permanent faults syndrome of TMR. Thus, the scrubbing rate would then determine the reliability of the system.

- The major cost associated with TMR solutions is the significant increase in hardware resources. A quick glance at Figure 5-7 would reveal the considerable increase in area required to make that design SEU tolerant. A typical TMR design increases the area by approximately 200% due to the two redundant modules (plus the voting logic). In contrast, the “Area – Synthesis Report” section of this chapter shows a 127% increase in area with the parity detection scheme proposed. This equates to a 37% improvement in hardware overhead over TMR.

- Power consumption is another negative consideration when implementing TMR. Using a power estimation tool, “XPower”\[42\] developed by Xilinx for its FPGAs,
it has been shown [41] that TMR designs consumes three times as much power as their Non-TMR counterparts. The ratio of power increase for different benchmarks is shown to range from 3.1 to 4.22. Table 5-3 shows a maximum 2.11 increase in power consumption for the proposed architecture. This equates to 32% - 50% improvement in power overhead.
Summary of SEU Mitigation Schemes' Properties

Table 5-4 below summarizes the various comparisons observed from simulation of the proposed method to previous research done on the existing methods. The dynamic power dissipation for the readback method is similar to that of the proposed architecture, because they perform very similar functions.

<table>
<thead>
<tr>
<th>PROPERTIES</th>
<th>READBACK</th>
<th>TMR</th>
<th>PROPOSED METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARDWARE</td>
<td>200% Increase in System Memory</td>
<td>200% Increase in FPGA Configuration Memory</td>
<td>127% Increase in FPGA Configuration Memory</td>
</tr>
<tr>
<td>DETECTION LATENCY</td>
<td>1.42 ms (^2)</td>
<td>N/A</td>
<td>1.2 (\mu) s (^3)</td>
</tr>
<tr>
<td>DYNAMIC POWER</td>
<td>Similar to “proposed method”</td>
<td>200% to 300% Increase in Power [41]</td>
<td>100% Increase in Power</td>
</tr>
<tr>
<td>DISSIPATION</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5-4: Summary of SEU Mitigation Scheme’s Properties

\(^2\) Calculation for Virtex XCV150 device. See Readback with Partial Reconfiguration Disadvantages section of Chapter 5 (page 71) for how calculation was derived.

\(^3\) Calculation for Virtex XCV150 with new architecture. See Readback with Partial Reconfiguration Disadvantages section of Chapter 5 (page 71) and “Timing – Synthesis Report” (page 67) for how calculation was derived.
Chapter 6 – Conclusion and Future Work

The proposed solution to detecting and correcting single event upsets (SEUs) in SRAM-based FPGAs entails a shift in architecture. The proposed architecture is modeled against Xilinx Virtex FPGAs. The configuration memory will be modified to become self error checking. This is accomplished by using dual-ported SRAM configuration memory. One port is dedicated to detecting SEUs through parity checking data frames. Each data frame outputs a parity value which is compared with the parity from a previous scan. If there is disagreement between the parity values from two scan sequences for any particular frame, then a syndrome flag is set to indicate the occurrence of an SEU in that frame. The erring frame would then be immediately corrected by partial reconfiguration. Partial reconfiguration is the process of reconfiguring parts of an FPGA without interrupting normal operation of the remaining portions of the system.

The obvious consideration in choosing to implement the proposed architecture is its level of improvement on existing methods such as triple modular redundancy (TMR) and readback. As with most designs, it would require a decision made based on trade-offs between latency, area overhead, power considerations, and system reliability.

Against the readback and partial reconfiguration method, it improves on the fault detection latency by over 1000% and reduces the amount of system memory required by 200%. However, readback is superior in terms of hardware overhead to the FPGA core. Against TMR, the proposed detection scheme shows an approximate improvement on area overhead of 37%. TMR designs also contribute 200% to 300% increase in power
dissipation \cite{41} as opposed to 100% increase shown in simulations of the proposed method, a 32% - 50% improvement.

The proposed detection scheme is relatively high-speed (in terms of error detection latency) and offers high reliability. However, the disadvantage is the area overhead and increase in power dissipation incurred. Its high frequency of scan sequences translates into higher dynamic power dissipation. This can be countered by reducing the frequency of scan, however it becomes a trade-off issue between power and system reliability.

Simulation and synthesis of the VHDL model for the proposed architecture confirms its functionality. The next phase of design would be the actual implementation of the detection circuitry and a fully functional controller for partial reconfiguration. An adaptive algorithm that tailors the number of cells being scanned to the design implemented would improve on power dissipation. Some designs in FPGAs do not require the entire configuration memory. It would be wasteful to scan cells and frames that are not in use.

Finally, this thesis does not address the issue of SEUs occurring in various flip-flops and latches in configuration logic blocks (CLBs). Unlike SRAM cells, latches in CLBs are not addressable. This means that their state cannot be accessed and analyzed to determine if an SEU occurred. These flip-flops often form the building blocks for shift registers and other important parts of designs in FPGAs. Therefore, providing SEU immunity to those latches will improve system reliability. One method of providing the needed immunity would be to make the latches and flip-flops TMR. It should however be noted that SEUs in latches are phased out the next time data is latched.
Bibliography


[34] Carl Carmichael, Xilinx Application Note, “Configuring Virtex FPGAs from Parallel EPROMs with a CPLD,” XAPP137, March 1, 1999, Version 1.0.


