Politecnico di Torino

Masters degree Course in Computer Engineering
Masters degree Thesis

Analysis and Mitigation of Single Event Transients (SET) effect on RISC-V based FPGA

Supervisors:

Prof. Luca Sterpone

Candidate:

Aditya Garg

Academic Year 2023-2024

Declaration

Index

Table Of Figures

Abstract

In the rapidly evolving field of electronics, Field-Programmable Gate Arrays (FPGAs) have emerged as a cornerstone technology for a wide range of applications. The versatility and programmability of FPGAs, combined with the increasing demand for low-power, high-performance computing, have led to the widespread adoption of Reduced Instruction Set Computing (RISC) architectures, notably the RISC-V, within FPGA designs. This open-source architecture offers significant advantages in terms of customization, scalability, and efficiency. However, the deployment of RISC-V based FPGAs in radiation-prone environments, such as space, raises substantial reliability concerns due to the potential effects of radiation-induced errors.

Radiation effects on semiconductor devices can lead to a myriad of operational challenges, including transient faults, permanent damage, and functional disruptions. For critical applications, these effects can compromise the integrity of the mission. Therefore, understanding the susceptibility of RISC-V based FPGA architectures to radiation is crucial for the development of effective mitigation strategies, ensuring reliability and functionality in adverse conditions.

This thesis aims to delve into the intricate dynamics of radiation effects on NEORV32 Processor is a customizable microcontroller-like system on chip (SoC) built around the open-source RISC-V compatible processor system that is written in platform-independent VHDL. Effects are studies by simulating single even transient, exploring the mechanisms behind radiation-induced failures and the implications for system reliability. By conducting a comprehensive analysis of these effects, this work seeks find ways to mitigate these effects. In summary, as the use of RISC-V based FPGA architectures continues to expand into new frontiers, the significance of ensuring their reliability under radiation exposure cannot be overstated.

Chapter 1

1 Introduction

1.1 RISC-V Overview

RISC-V (pronounced "risk-five") is a free and open ISA enabling a new era of processor innovation through open standard collaboration. Born at the University of California, Berkeley, in 2010, RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the next 50 years of computing design and innovation. Unlike proprietary ISAs, RISC-V is available under open licenses that do not require fees to use, making it appealing for various applications, from embedded systems to massive parallel computing.

1.1.1 Advantages of RISC-V

- Open and Extensible: One of the core tenets of RISC-V is its openness and extensibility. This means that any organization or individual can design, manufacture, and sell RISC-V chips and software without the need for royalties or licensing fees. This openness fosters innovation and allows for custom extensions to meet specific needs.

- Simplicity and Efficiency: The RISC (Reduced Instruction Set Computing) approach emphasizes simplicity and efficiency. RISC-V instructions are designed to be simple to understand and implement, which can lead to more efficient processor designs, especially in terms of power consumption and performance.

- Broad Support and Adoption: RISC-V has garnered widespread support from both academia and industry. This broad support ensures a rich ecosystem of tools, libraries, and resources, facilitating the development of RISC-V based solutions.

1.1.2 RISC-V Appeal for Space Applications

- Customizability for Specific Missions: The open-source nature of RISC-V allows for the creation of custom ISA extensions tailored to the unique computational needs of space missions. This could include optimized instructions for navigation, data processing, or communication, enhancing efficiency and performance.

- Reduced Power Consumption: Space missions critically need to manage power consumption due to limited energy resources aboard spacecraft. RISC-V's efficient instruction set can be optimized further for low-power operations, making it ideal for such constrained environments.

- Enhanced Reliability and Fault Tolerance: The simplicity of RISC-V facilitates the design of processors with inherent reliability and fault tolerance, crucial for handling the radiation-induced errors common in space. Coupled with FPGA's reconfigurability, systems can be designed to adapt and respond to failures, enhancing mission resilience. (1)

1.2 Single Event Transient

A Single Event Transient (SET) refers to a temporary change in the electrical state of a device due to the passage of a single ionizing particle. When this particle strikes a semiconductor material, such as those found in circuits and microchips, it can generate a localized charge along its path. This sudden generation of charge can momentarily change the state of transistors or other electronic components, leading to transient pulses in the output signals of the affected circuits. SETs are particularly significant in space applications due to the high-energy particles found in cosmic rays and solar flares, which are not shielded by Earth's atmosphere as they are at the surface. (2)

1.2.1 How SETs Affect Circuits in Space

- **Transient Faults:** Unlike permanent faults, which damage the physical structure of the device, SETs cause temporary malfunctions. These can result in erroneous data outputs or unexpected behaviour in digital circuits, potentially leading to critical errors in spacecraft operations, satellite communications, and other space-based systems.

- **System Disruptions:** In complex systems, such as onboard computers and sensors, a single transient can propagate through the system, leading to cascading failures or system resets. This is especially problematic in space missions, where reliability and autonomous operation are paramount.

- **Increased Error Rates:** The prevalence of high-energy particles in space means that circuits are continually exposed to conditions that can cause SETs. This increases the overall error rate, necessitating robust error detection and correction mechanisms to ensure data integrity and system reliability.

- **Design Challenges:** To mitigate the effects of SETs, space-bound electronic systems must be designed with radiation-hardened components and architectures. This includes the use of redundant systems, error detection and correction codes, and specific circuit design techniques to minimize the impact of transients. However, these measures often result in increased complexity, weight, and cost.

- **Operational Considerations:** SETs can affect not only the hardware but also the software running on space systems. Software algorithms may need to include checks for plausibility and redundancy to handle transient-induced errors gracefully, ensuring that critical operations can continue even in the presence of SETs.

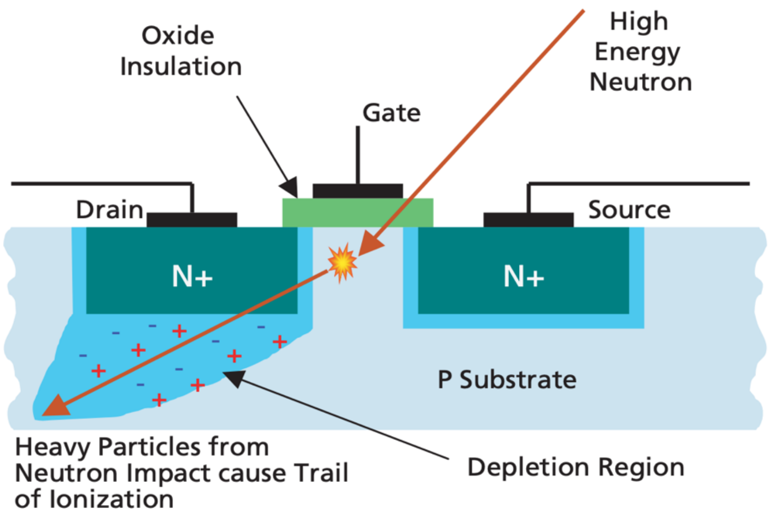
The understanding and mitigation of SETs are critical components of designing electronic systems for space. By considering the potential impact of these transients, engineers can create more reliable and resilient systems capable of withstanding the harsh conditions encountered beyond Earth's atmosphere. This is an area of ongoing research, as the continued miniaturization of electronics and the push for more powerful computing capabilities in space present new challenges in the management of SETs and other radiation-induced effects. (2)

1.3 Circuit-Level Affect Breakdown

A Single Event Transient (SET) affects digital circuits by causing a temporary, unintended change in the state of the circuit, which can disrupt its normal operation. This phenomenon occurs when a high-energy particle, such as a neutron or a proton from cosmic rays or solar flares, strikes a semiconductor material used in the circuit. Here's a breakdown of how an SET impacts a circuit: (3) (4)

1.3.1 Generation of Charge

- **Ionizing Particle Strike:** When the high-energy particle collides with the semiconductor material, it ionizes atoms along its path, generating electron-hole pairs.

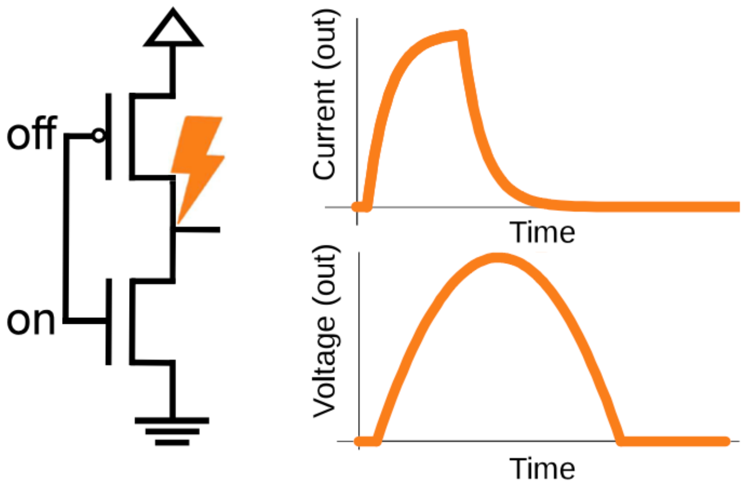


1 Particle strike path across a NMOS transistor, charge collection happens in three stages

- **Charge Collection:** The generated charge carriers (electrons and holes) are collected by nearby junctions, potentially causing a significant, localized change in voltage.

1.3.2 Transient Pulse Formation

- **Pulse Generation:** This localized voltage change can create a transient pulse in the electronic signal. The transient's amplitude and duration depend on several factors, including the particle's energy, the semiconductor material's properties, and the circuit's design.

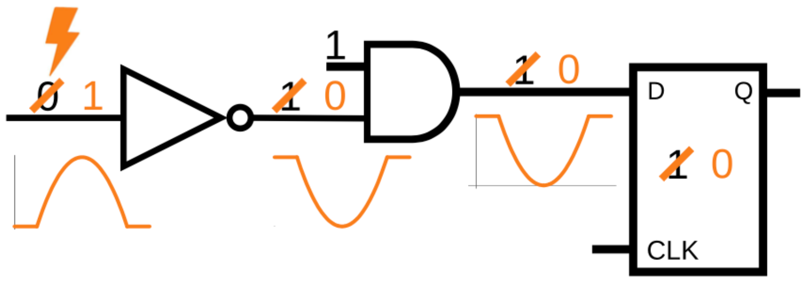


2 A particle strike at the drain of the inverter gate creates a burst of current, changing output voltage value (5)

- **Propagation:** The transient pulse can propagate through the circuit, affecting logic states, analog signal levels, or both. (5)

1.3.3 Effects on Circuit Operation

- **Logical Errors:** In digital circuits, an SET can lead to bit flips (changing a 0 to a 1 or vice versa) in memory elements or logic gates. This can cause errors in data processing or storage, potentially leading to incorrect outputs or system behaviour.



3 SET pulse reaches the memory element and its value is stored con- figuring a SEU (5)

- **Signal Disturbance:** In analog circuits, an SET can temporarily alter signal levels, potentially leading to misinterpretation of the signals or triggering false conditions.

- **System Disruption:** Critical systems, such as those used in aerospace, automotive, and medical applications, may experience malfunctions, leading to a failure in performing essential tasks. (5)

1.4 Design Level Mitigation Strategies

To counteract the effects of SETs, several mitigation strategies are employed in circuit design and system architecture:

- **Redundancy:** Implementing redundancy in critical components or circuits is a common strategy for mitigating radiation effects. Techniques like Triple Modular Redundancy (TMR) involve tripling critical logic and then voting on the outputs to ensure correctness, effectively filtering out transient errors caused by radiation.

- **Hardened-by-Design (HBD):** This approach involves modifying the physical layout and electrical properties of integrated circuits to enhance their resistance to radiation effects. Techniques include increasing the critical charge required for a bit flip, using guard rings to prevent lateral charge collection, and designing latches and flip-flops that are inherently more resistant to SETs.

- **Time Redundancy:** Time redundancy involves performing critical operations multiple times and comparing the results to identify and correct errors. This can be effective for mitigating SETs but may not be suitable for time-sensitive applications due to the increased latency. (4)

1.4.1 Component-Level Techniques

- **Radiation-Hardened Components:** Using components specifically designed to withstand radiation is a straightforward approach to mitigate its effects. These components are manufactured using specialized materials and processes to enhance their tolerance to Total Ionizing Dose (TID), Single Event Upsets (SEUs), and SETs.

- **Error Detection and Correction (EDAC):** EDAC circuits, such as parity checkers and Hamming codes, are used to detect and correct errors in data storage and transmission. While more commonly associated with SEUs in memory, these techniques can also be adapted to address transient errors in logic circuits.

1.4.2 System-Level Techniques

- **System Architecture Adjustments:** Adjusting the overall system architecture can enhance resilience to radiation. This includes designing systems with fail-safe states, isolation of critical subsystems, and employing non-volatile memory for essential data storage to prevent corruption.

- **Software Mitigation:** Software techniques, including watchdog timers, periodic system resets, and software-based error detection and correction algorithms, can also mitigate radiation effects. These methods are particularly useful for correcting transient errors and ensuring system reliability without requiring hardware modifications.

- **Dynamic Reconfiguration:** For systems based on reconfigurable hardware like FPGAs, dynamic reconfiguration can be used to correct radiation-induced errors. Faulty logic blocks can be reconfigured or bypassed, allowing the system to recover from SETs and SEUs without manual intervention

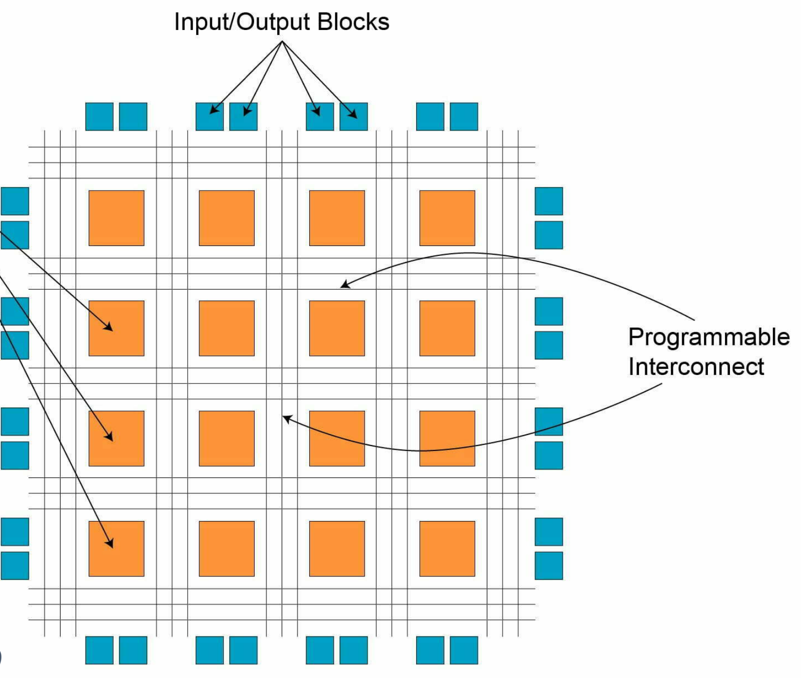
- **Environmental Shielding:** Although not a direct mitigation technique for the circuit itself, providing environmental shielding can significantly reduce the radiation exposure of electronic components. Materials like lead, aluminium, and hydrogen-rich compounds are commonly used to shield electronics from high-energy particles. (4)

Chapter 2

2 Technology Background

2.1 FPGAs

Field-Programmable Gate Arrays (FPGAs) are a class of semiconductor devices that offer a unique blend of versatility, performance, and adaptability, distinguishing them



4 FPGA Block (16)

from traditional fixed-function integrated circuits. An FPGA consists of an array of programmable logic blocks, interconnected by programmable routing channels. These logic blocks can be configured and reconfigured, even after manufacturing, to perform a wide variety of digital functions. This reconfigurability allows FPGAs to be customized for specific applications or to be updated post-deployment to enhance functionality or correct errors.

FPGAs find a prominent place in the pantheon of space electronics due to a confluence of their intrinsic features and the exigent requirements of space missions. The formidable environment of space, characterized by extreme temperature variations, high vacuum, and, most critically, intense radiation levels, presents unique challenges for electronic systems. These challenges demand solutions that not only withstand these conditions but also offer flexibility, high performance, and reliability—qualities inherent to FPGAs. (6)

2.2 FPGA Features

2.2.1 Performance and Efficiency

FPGAs excel in executing parallel processing tasks, a capability that is particularly beneficial for the data-intensive operations common in space applications, such as image processing, signal processing, and onboard data analysis. Their architecture enables them to handle multiple processes simultaneously, dramatically reducing the time required for data processing and analysis, a critical factor in time-sensitive missions and when communicating with Earth-based systems.

2.2.2 Radiation Tolerance

Space is a hostile environment filled with ionizing radiation from solar flares, cosmic rays, and the Van Allen belts. This radiation can cause severe damage to electronic circuits, leading to data corruption, system malfunctions, or even complete failure. Radiation-hardened FPGAs are specifically designed to resist such effects, incorporating design features and manufacturing processes that enhance their resilience to single-event upsets (SEUs), single-event transients (SETs), and total ionizing dose (TID) effects. These features make FPGAs an indispensable choice for reliable operation in space environments.

2.2.3 Reduced Size, Weight, and Power (SWaP)

The constraints of launching and operating systems in space necessitate stringent control over size, weight, and power consumption. FPGAs contribute significantly to SWaP optimization by integrating the functionalities of multiple discrete components into a single device. This integration not only reduces the physical footprint and weight of electronic systems but also enhances power efficiency, a paramount consideration for satellite and spacecraft designers.

2.2.4 Customization and Application-Specific Optimization

The programmable nature of FPGAs allows for the tailoring of their logic to meet the precise requirements of specific applications, enabling optimal performance for particular tasks. This level of customization is particularly advantageous in space applications, where specific processing algorithms, control logic, and data handling procedures can be implemented directly on the FPGA, reducing the need for external components and streamlining system architecture.

2.3 Implementing RISC-V on FPGAs for Space

The combination of RISC-V and FPGA technologies for space applications provides a powerful platform for developing highly adaptable and efficient computing systems. FPGA's inherent flexibility allows for on-the-fly reprogramming and adaptation to changing mission requirements or in response to hardware failures.

- Rapid Prototyping and Testing: Utilizing FPGAs for RISC-V implementation enables rapid prototyping of spaceborne processors, allowing for extensive testing and iteration in the development phase. This is critical in ensuring that the final design meets the stringent reliability and efficiency requirements of space missions.

- On-Mission Reconfigurability: The dynamic nature of space missions often necessitates adjustments to computational strategies. FPGAs allow for such reconfigurability in space, enabling adjustments to RISC-V based processors for optimized performance throughout the mission lifecycle.

- Radiation Hardening: While FPGAs are inherently susceptible to radiation, specialized radiation-hardened FPGAs combined with RISC-V's adaptable architecture can lead to the development of processors that are both resilient to space conditions and capable of recovery from radiation-induced faults. (7)

2.4 FPGA programming

FPGA programming, unlike traditional software development, involves configuring a Field-Programmable Gate Array (FPGA) to perform specific digital computations. An FPGA consists of an array of programmable logic blocks and a hierarchy of reconfigurable interconnects that allow these blocks to be wired together—somewhat like a blank canvas for digital circuits. Programming an FPGA involves defining how these blocks and interconnects work together to perform a desired function. This process transforms the FPGA into a hardware implementation of your specific requirements, whether it be a custom processor, a digital signal processing algorithm, or any other digital system.

- **Hardware Description Languages (HDLs):** FPGA programming is primarily done using Hardware Description Languages, such as VHDL (VHSIC Hardware Description Language) and Verilog. These languages allow developers to describe the hardware functionality and logic at a high level of abstraction.

- **Synthesis:** The HDL code is synthesized, meaning it is compiled and translated into a configuration that specifies how the FPGA's logic blocks and interconnects should be configured. This step essentially converts your design from a high-level description into a map of logic gates and connections.

- **Simulation:** Before loading the design onto an FPGA, it is crucial to simulate it to ensure it behaves as expected. Simulation tools allow developers to test their designs under various conditions without the need for physical hardware.

- **Implementation:** This phase involves placing and routing, where the synthesized design is fitted onto the FPGA's physical layout. The software tools allocate specific logic blocks for the design's components and connect them according to the design's needs, while optimizing for performance and resource utilization.

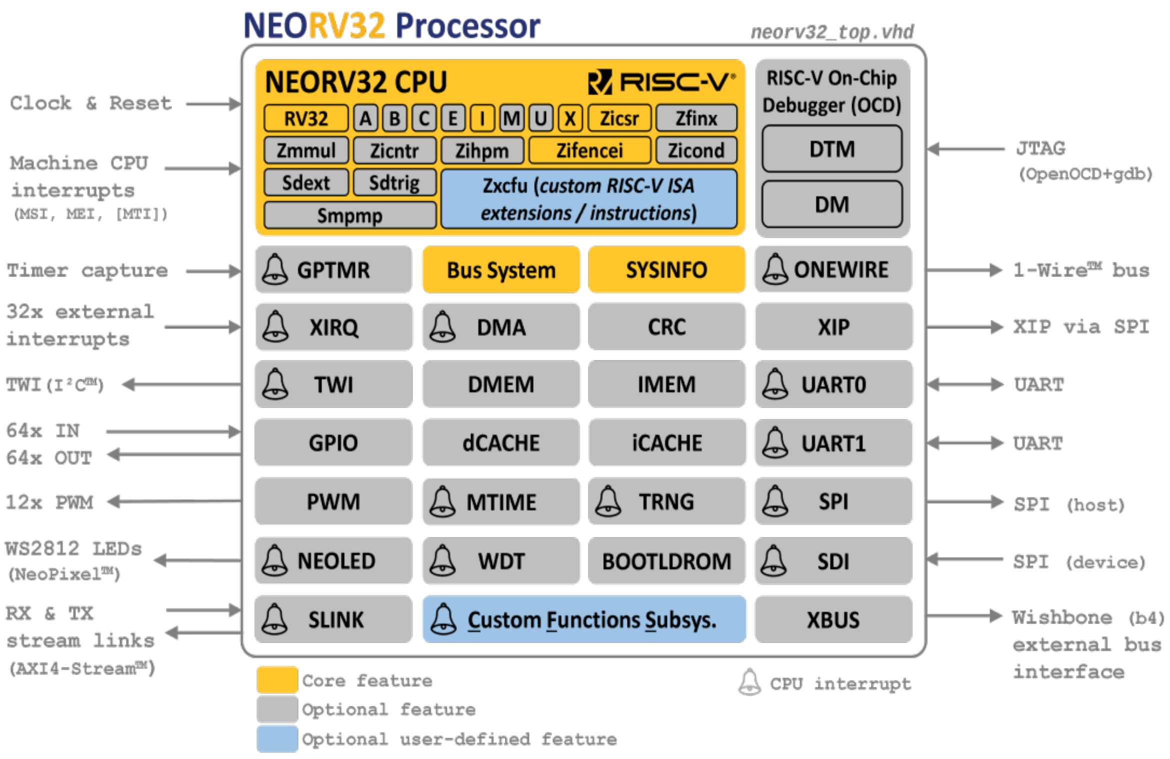
- **Bitstream Generation:** Once the design is implemented, the toolchain generates a bitstream file. This file contains the binary configuration data that will be loaded onto the FPGA, physically configuring its logic blocks and interconnections to realize the design.

- **Configuration:** Finally, the bitstream is loaded onto the FPGA, configuring it as per the design. This step is where the FPGA becomes the digital circuit that you designed. The configuration can be volatile, meaning it needs to be reloaded if the FPGA loses power, or non-volatile, depending on the FPGA type and the configuration method used. (6)

2.5 NEORV32 RISC-V Processor

The NEORV32 RISC-V Processor presents a comprehensive, open-source system compatible with the RISC-V architecture, designed for seamless integration as an auxiliary processor within broader System-on-Chip (SoC) designs, or as a dedicated, tailor-made microcontroller. This processor system stands out for its extensive configurability, offering a suite of optional peripherals such as built-in memory modules, timers, serial communication interfaces, general-purpose input/output (GPIO) ports, and an external bus interface for the addition of custom Intellectual Property (IP) elements like memory blocks, Network-on-Chips (NoCs), and various peripherals. Additionally, it supports both online and in-system debugging through a debugger that is compatible with OpenOCD/gdb, accessible via a JTAG interface.

A key priority of the NEORV32 system is execution safety, aiming to ensure consistent and predictable performance under all circumstances. To this end, the CPU is designed to confirm all memory accesses and to reject any invalid or malformed instructions. In the event of an unforeseen issue, the system is engineered to notify the application code through hardware exceptions, maintaining operational integrity.

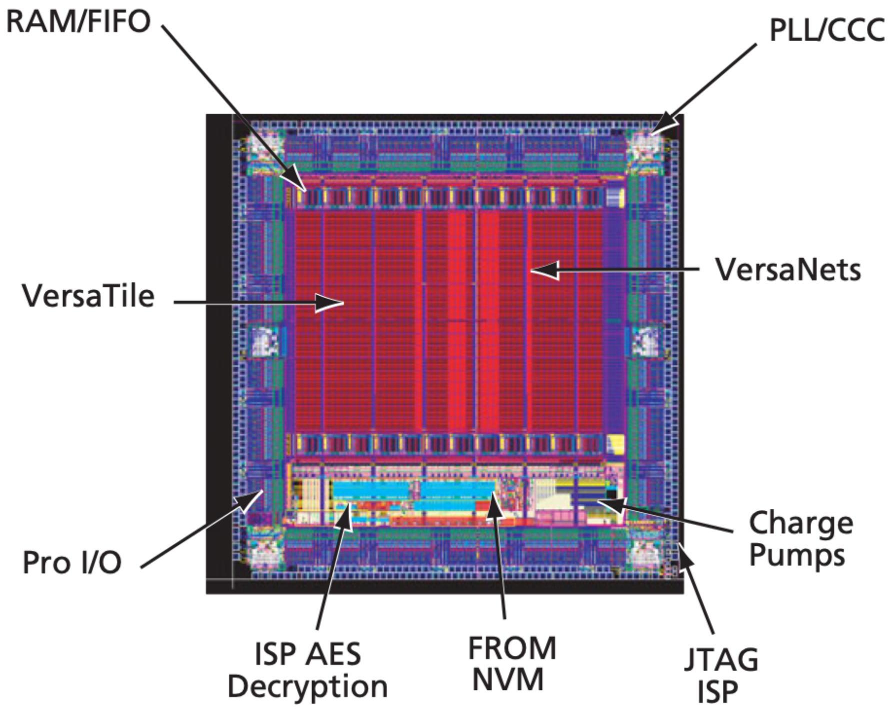


5 NEORV32 Architecture (8)

On the software front, the NEORV32 ecosystem is equipped with a robust framework that includes application-specific makefiles, libraries supporting all CPU and processor functionalities, a bootloader, a runtime environment, and a variety of example programs. This suite even features a version of the CoreMark microcontroller benchmark and the official test suite for RISC-V architecture, ensuring comprehensive testing capabilities. The default toolchain for software development is the RISC-V GCC, with prebuilt versions also available, facilitating a wide range of development and implementation scenarios. The NEORV32 is not based on another RISC-V core. It was build entirely from ground up (just following the official ISA specs. (8)

2.6 ProASIC3

The ProASIC3 FPGA, developed by Microsemi (now part of Microchip Technology), represents a significant advancement in FPGA technology, especially suited for applications demanding high reliability, low power consumption, and stringent security requirements. In the context of space applications, ProASIC3 FPGAs offer a compelling choice due to their non-volatile, flash-based technology, which inherently provides better resistance to radiation effects compared to SRAM-based FPGAs. This makes them particularly well-suited for the harsh environments of space, where radiation can cause bit flips and other errors. (9)



6 ProASIC3 Architecture (9)

2.6.1 Key Features of ProASIC3 FPGA

- Non-volatile and Instant-on: Unlike SRAM-based FPGAs, ProASIC3 devices are non-volatile, meaning they retain their configuration even after power is removed. They also feature instant-on capability, significantly reducing the initialization time after power-up, which is critical for time-sensitive space applications.

- Radiation Tolerance: ProASIC3 FPGAs are designed with inherent tolerance to radiation, making them less susceptible to Single Event Upsets (SEUs) and other radiation-induced failures. This is crucial for space missions, where exposure to high levels of cosmic rays and solar radiation is a significant concern.

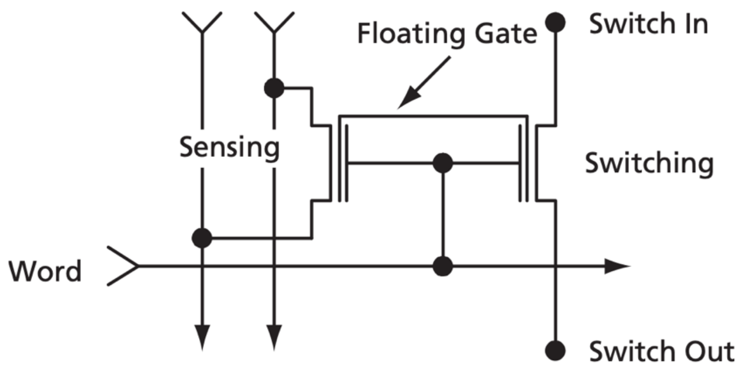
- Low Power Consumption: These FPGAs are optimized for low power operation, which is essential for space missions that often operate on limited power budgets. Their flash-based technology contributes to lower static power consumption compared to other FPGA technologies.

- Security Features: ProASIC3 FPGAs come with advanced security features, including built-in AES encryption and a unique FlashLock technology, which provides a method to secure the FPGA configuration against unauthorized access. This is particularly important for missions that handle sensitive data or require secure communication.

- High Performance and Density: Despite their focus on reliability and low power, ProASIC3 FPGAs do not compromise on performance. They offer a range of densities and support high-speed digital signal processing, making them suitable for complex computational tasks in space applications. (9)

2.7 Flash Technology

The superior density, performance, and security features distinguishing ProASIC3/E Flash-based FPGAs from traditional SRAM-based FPGAs stem from their innovative Flash-based LVCMOS process that incorporates seven metal layers. Utilizing conventional CMOS design approaches for logic and control functions, these devices achieve remarkable efficiency. The blend of fine-grained architecture, improved flexible routing capabilities, and plentiful Flash switches facilitates unmatched logic utilization rates, ensuring devices maintain excellent routability and performance. Central to ProASIC3/E devices is their Flash programming element, which is designed around a dual-transistor structure.



7 Flash Gate Design (9)

Flash switches, strategically integrated throughout the device, enable non-volatile and reconfigurable interconnect programming, setting Flash FPGAs apart from SRAM-based counterparts, which often struggle with place-and-route efficiency beyond 70% utilization. Flash FPGA architecture allows for near-total core utilization for a wide range of designs. Moreover, comprehensive on-device programming circuitry supports swift (3.3 V) programming through an IEEE1532 JTAG interface, enhancing the ProASIC3/E devices' convenience and accessibility. (9)

Contrastingly, SRAM-based FPGAs typically require four to six transistors for each programming element, leading to a larger die area. Consequently, to be cost-competitive, SRAM FPGAs need to be manufactured using smaller, more advanced, and thus more costly and power-intensive process technologies.

Flash technology underpins the ProASIC3/E devices, offering significant benefits over SRAM-based FPGAs. These include reduced system costs due to the elimination of external components, the inclusion of user-accessible non-volatile memory, enhanced security features, lower power requirements, immediate operation upon power-up, and robustness against firm errors, contributing to their appeal in cost-sensitive and reliability-critical applications.

2.7.1 Integration with RISC-V

The integration of RISC-V cores into ProASIC3 FPGAs for space applications leverages the FPGA's reliability and performance features while benefiting from RISC-V's flexibility and customizability. This combination allows for the development of highly specialized computing solutions that can be optimized for specific mission requirements, from enhanced data processing capabilities to improved fault tolerance.

Chapter 3

3 Analysing and Hardening of Implementation

In the initial phase of my thesis research, I embarked on a comprehensive process to prepare and evaluate a digital circuit designed to withstand the harsh conditions of space, particularly focusing on the mitigation of radiation effects such as Single Event Transients (SETs). The foundation of my work involved leveraging the NEORV32 RISC-V Processor, a versatile, open-source processor system compatible with the RISC-V architecture, renowned for its adaptability and efficiency in custom and stand-alone microcontroller applications.

3.1 Preparing the Circuit

The methodology commenced with deriving my circuit from the base Register Transfer Level (RTL) VHDL files provided by the NEORV32 framework. A pivotal element of this phase was the development of a succinct test program designed to perform matrix multiplication. This program was initially composed in C, subsequently compiled into instruction memory data utilizing the RISC-V GCC toolchains for Linux provided by the NEORV32 suite. This translation from a high-level programming language to machine language was a critical step, ensuring the program's compatibility with the intended hardware environment.

Detailed Steps:

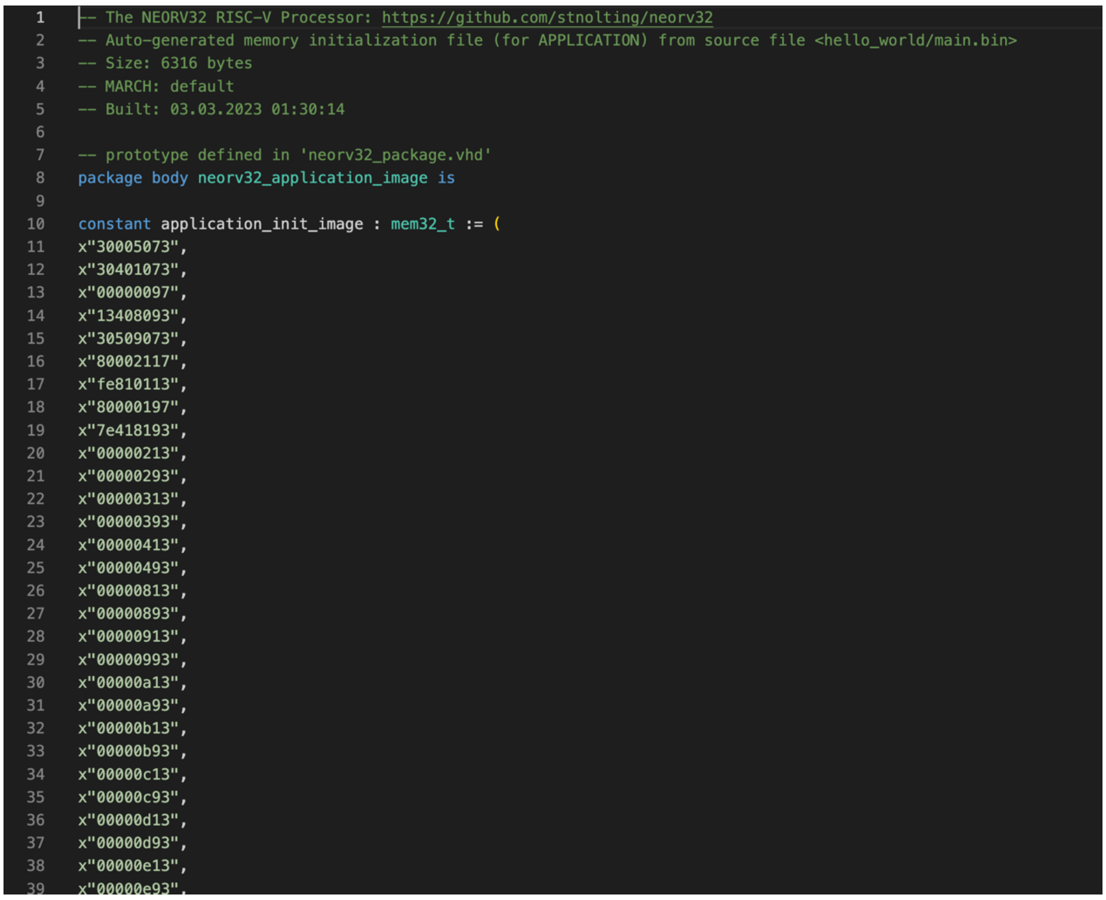
- **Derivation of Circuit:** The base RTL VHDL files from the NEORV32 framework were used as the starting point. These files define the hardware architecture and behaviour of the NEORV32 processor.

- **Development of Test Program:** A test program performing matrix multiplication was developed in C. This program served as a benchmark to evaluate the performance and functionality of the NEORV32 processor within the FPGA environment.

- **Compilation to Machine Language:** Using the RISC-V GCC toolchains for Linux, the C program was compiled into machine language, specifically into instruction memory data compatible with the NEORV32 processor. The RISC-V GCC toolchains translate the high-level C code into RISC-V assembly language and then into binary instructions that can be executed by the NEORV32 processor.

- **Loading into Instruction Memory:** The compiled machine language instructions were loaded into the instruction memory of the NEORV32 processor within the FPGA. This step ensured that the test program could be executed by the processor during the subsequent phases of the methodology.

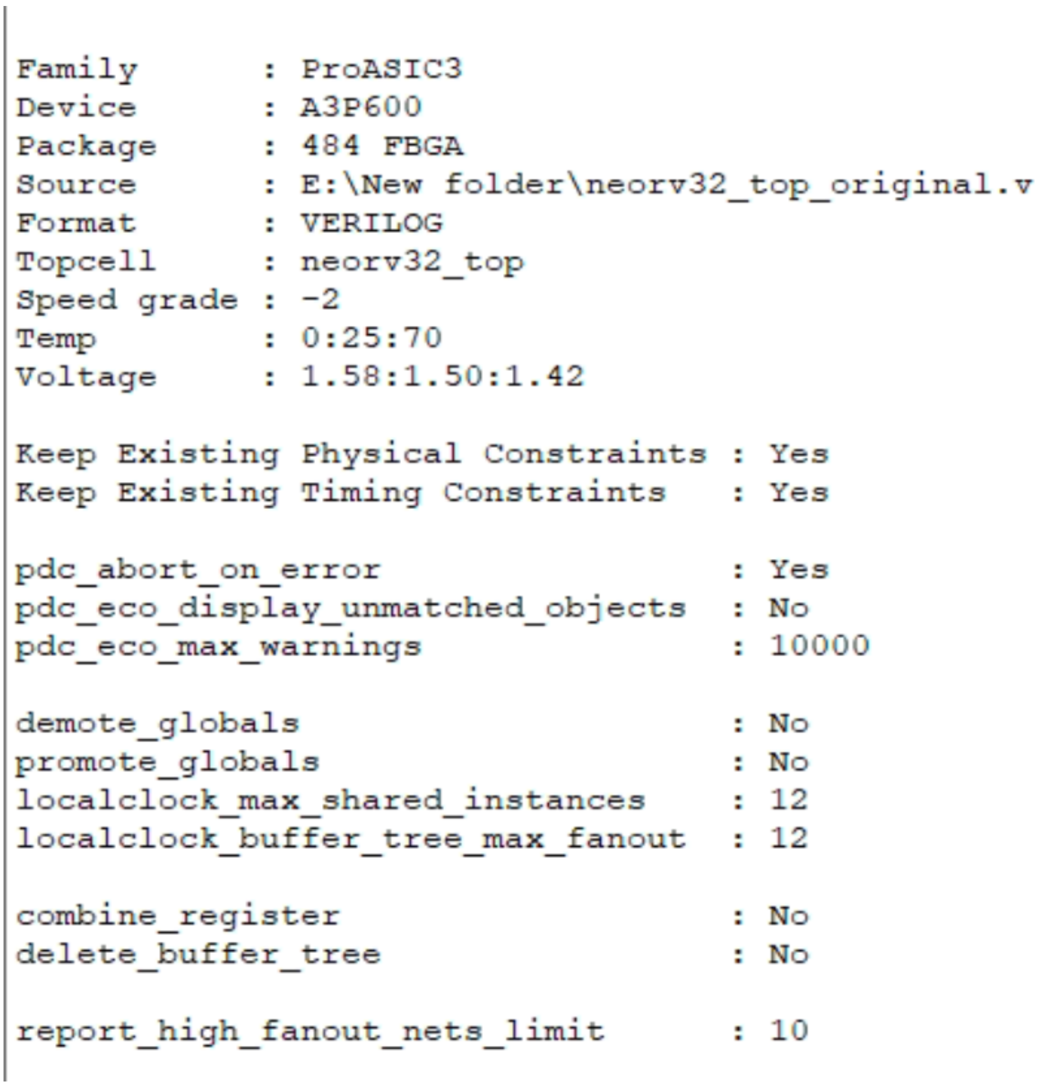
By meticulously following these steps, the foundation was laid for the subsequent phases of circuit design, testing, and modification aimed at mitigating the effects of radiation induced Single Event Transients (SETs). This initial setup was crucial for validating the processor's functionality and performance under normal operating conditions. (8)



```
-- The NEORV32 RISC-V Processor: https://github.com/stnolting/neorv32
-- Auto-generated memory initialization file (for APPLICATION) from source file <hello_world/main.bin>
-- Size: 6316 bytes
-- MARCH: default
-- Built: 03.03.2023 01:30:14

-- prototype defined in 'neorv32_package.vhd'
package body neorv32_application_image is

constant application_init_image : mem32_t := (
x"30005073",
x"30401073",
x"00000097",
x"13408093",
x"30509073",
x"80002117",
x"fe810113",
x"80000197",
x"7e418193",
x"00000213",
x"00000293",
x"00000313",
x"00000393",
x"00000413",
x"00000493",
x"00000813",
x"00000893",
x"00000913",
x"00000993",
x"00000a13",
x"00000a93",
x"00000b13",
x"00000b93",
x"00000c13",
x"00000c93",
x"00000d13",
x"00000d93",
x"00000e13",
x"00000e93",
```

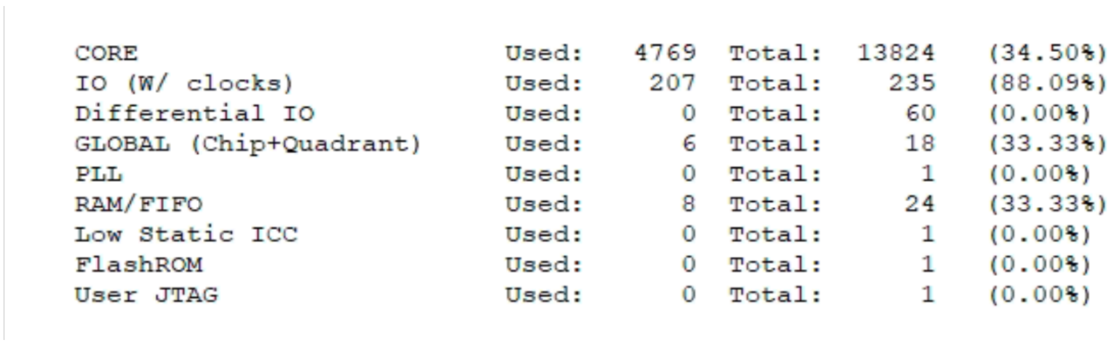
8 Sample application code in machine Language

Following the program compilation, the RTL description was synthesized using the Libero SoC software, targeting a ProASIC3 FPGA platform. This choice of hardware was deliberate, selected for its capability to accommodate the demands of the compiled program, aligning with the specific requirements dictated by the foundational program structure. A thorough examination of timing and space utilization reports ensued, providing an analytical basis for subsequent modifications aimed at enhancing radiation resilience.



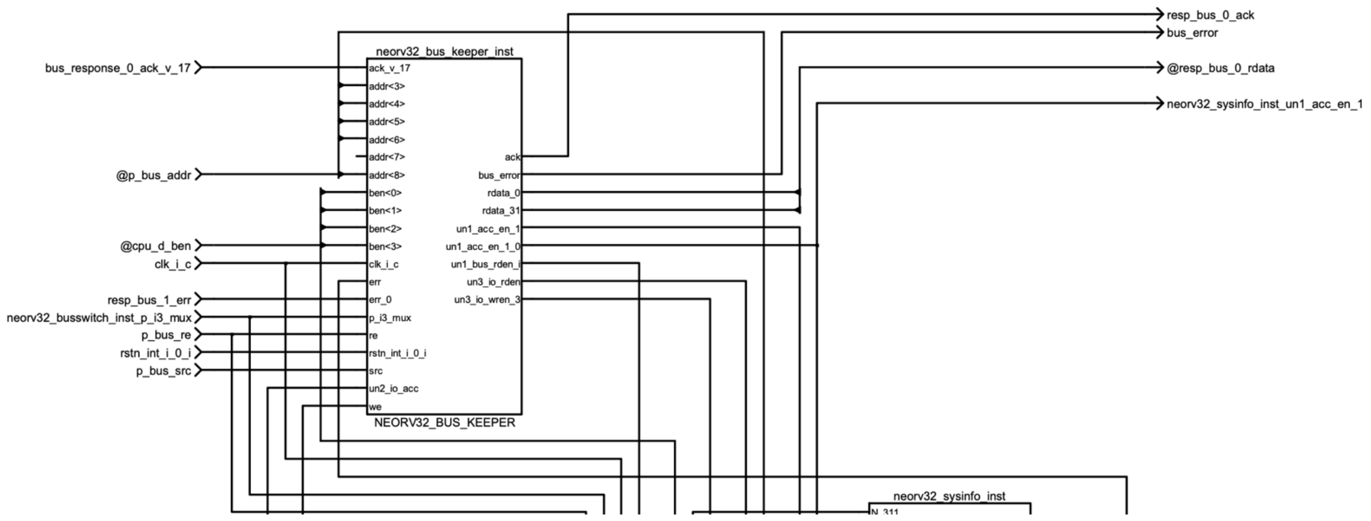
```
Family       : ProASIC3
Device       : A3P600
Package      : 484 FBGA
Source       : E:\New folder\neorv32_top_original.v
Format       : VERILOG
Topcell      : neorv32_top
Speed grade  : -2
Temp         : 0:25:70
Voltage      : 1.58:1.50:1.42

Keep Existing Physical Constraints : Yes
Keep Existing Timing Constraints   : Yes

pdc_abort_on_error                 : Yes
pdc_eco_display_unmatched_objects  : No
pdc_eco_max_warnings               : 10000

demote_globals                     : No
promote_globals                    : No
localclock_max_shared_instances    : 12
localclock_buffer_tree_max_fanout  : 12

combine_register                   : No
delete_buffer_tree                 : No

report_high_fanout_nets_limit      : 10
```

9 Compile Options in Libero SOC



```
    CORE                       Used:   4769  Total:  13824   (34.50%)
    IO (W/ clocks)             Used:    207  Total:    235   (88.09%)
    Differential IO            Used:      0  Total:     60   (0.00%)
    GLOBAL (Chip+Quadrant)     Used:      6  Total:     18   (33.33%)
    PLL                        Used:      0  Total:      1   (0.00%)
    RAM/FIFO                   Used:      8  Total:     24   (33.33%)
    Low Static ICC             Used:      0  Total:      1   (0.00%)
    FlashROM                   Used:      0  Total:      1   (0.00%)
    User JTAG                  Used:      0  Total:      1   (0.00%)
```

10 Compile Report Example of final resource usage



9 Circuit Placement by Libero SOC

The next phase involved the exportation of the project's netlist, facilitating the transition to the physical layout stage. At this juncture, the circuit's layout was meticulously organized using a variety of parameters to pre-emptively address potential radiation-induced challenges. A novel approach was employed using a script to calculate the optimal placement of components based on the average Manhattan distance, constrained by a predetermined maximum spatial allowance for the circuit. This script ingeniously allocated all components within a specified average Manhattan distance, effectively optimizing the layout for radiation resistance. (10)

3.2 Manhattan Circuit Placement Technique

The Manhattan placement technique, often referred to in the context of integrated circuit design and particularly FPGA layouts, involves arranging circuit components in a grid-like pattern that resembles the street layout of Manhattan, New York. This method optimizes the physical placement of components to minimize interconnect lengths and improve overall circuit performance. (11)

3.2.1 Implementation Steps

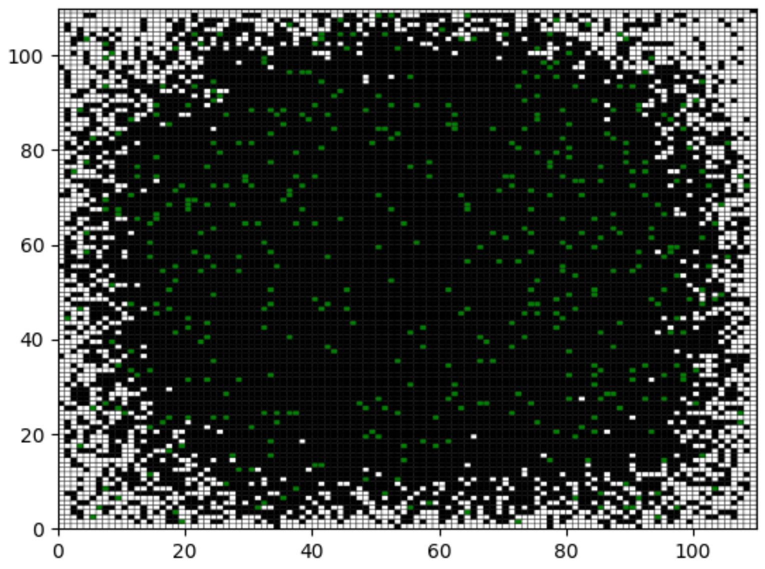
- Initial Component Placement: Start by placing the most critical components at strategic locations within the grid. These are typically components with the highest connectivity or those that are sensitive to delays. Subsequent components are placed relative to these critical nodes to ensure minimal interconnect distance.

- Heuristic Algorithms: Heuristic algorithms, such as simulated annealing or genetic algorithms, can be employed to find an optimal placement that minimizes the Manhattan distance. These algorithms iteratively adjust component positions to improve the overall placement efficiency.

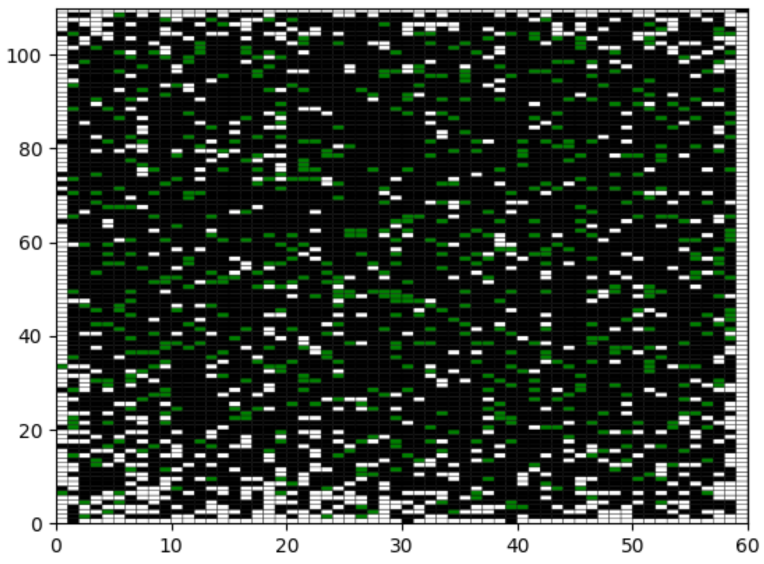
- Spacing and Alignment: Ensure that components are aligned with grid points, maintaining uniform spacing to facilitate orthogonal routing. Adjust spacing to account for component sizes and connectivity requirements, ensuring that there is sufficient space for routing interconnects without congestion.

- Routing Optimization: Once components are placed, routing algorithms are used to connect them, following the grid paths to maintain the orthogonal routing structure. The routing process must ensure that all connections are made without violating design rules, such as maximum allowed wire length or crosstalk limits.

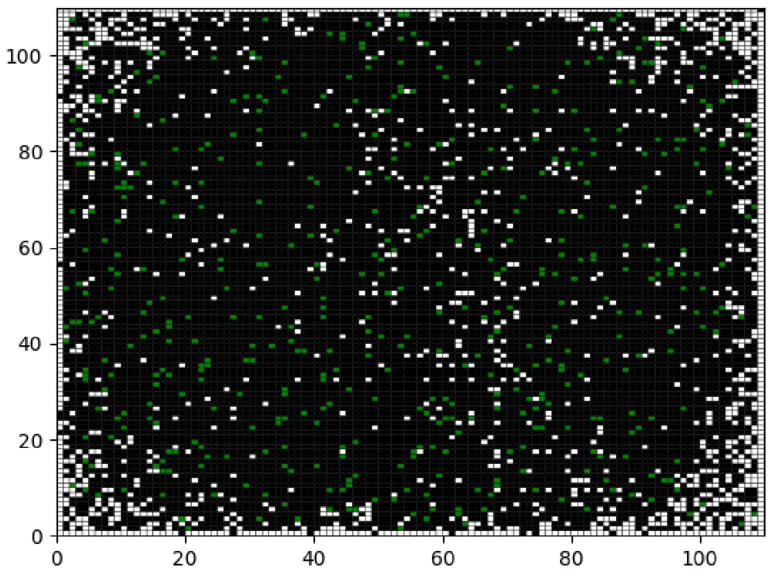
- Performance Evaluation: Evaluate the placement and routing for key performance metrics, including signal delay, power consumption, and overall circuit area. Use simulation tools to validate the functionality and performance of the placed and routed design. (12)



10 Bit-Count Circuit Manhattan Placement



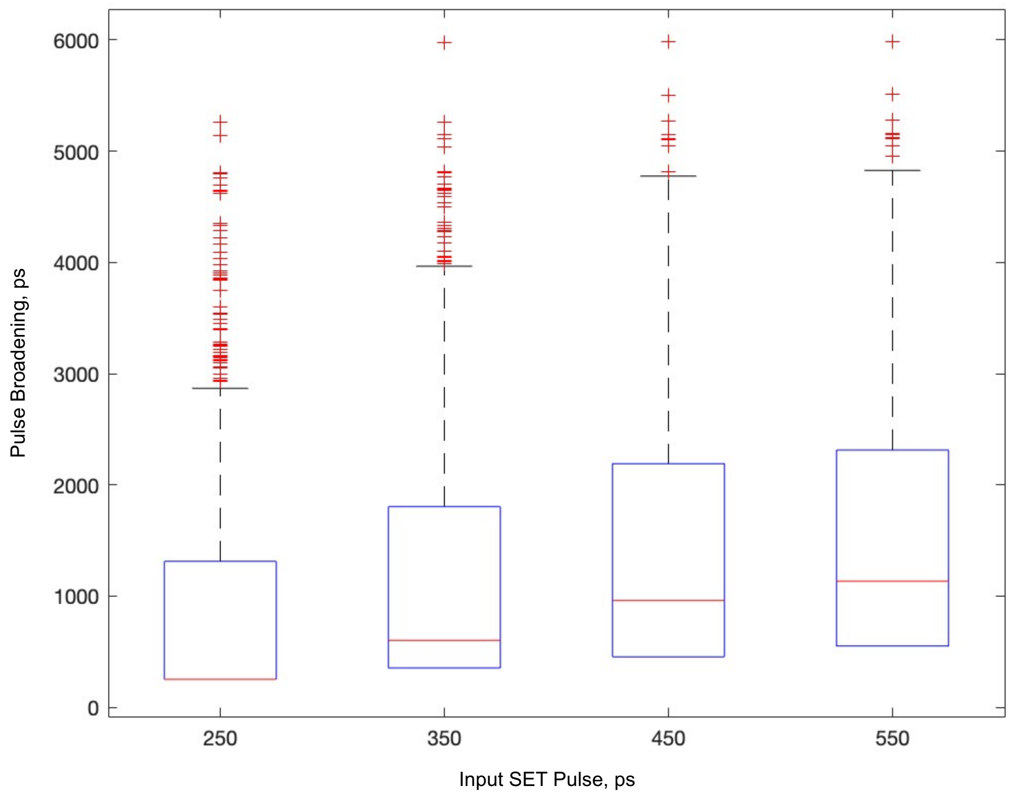
11 Matrix Multiplication Circuit Manhattan Placement



12 Hello World Circuit Manhattan Placement

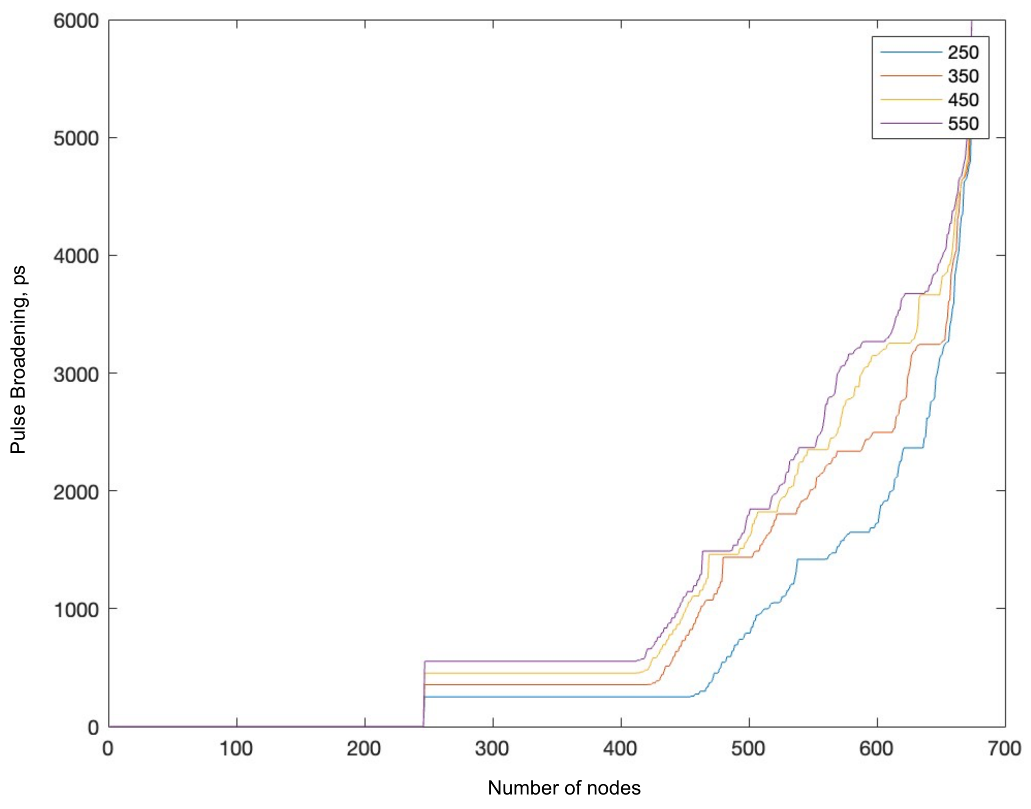
3.3 SET Simulation

With the physical design layout (PDD file) established, the focus shifted towards simulating the impact of radiation on the circuit's integrity. A Python script was developed for this purpose, ingeniously designed to simulate the effect of radiation by specifying parameters such as the maximum radiation pulse width (in picoseconds) and its amplitude. This simulation targeted the sensitive nodes within the circuit, identifying every connection as a potential vulnerability to SETs. The execution of this script culminated in a detailed report, elucidating the simulated radiation's impact on the circuit's operational efficacy. The report highlighted the extent of data transmission width expansion across sensitive nodes, a metric crucial for evaluating the circuit's resilience to SETs. (13)



13 Set Propagation Box Plot showing pulse width elongation at different input pulse

This foundational work sets the stage for the exploration of strategies to mitigate the adverse effects of radiation on the circuit, ensuring its reliability and performance in environments susceptible to high levels of ionizing radiation. The results derived from these initial experiments are instrumental in guiding the development of robust, radiation-hardened digital circuits capable of operating within the demanding conditions of space and other radiation-rich environments.



14 Set Propagation of Varying Width to analyse trend across different width

3.4 Mitigation Process

Upon obtaining the diagnostic reports from the previously executed Single Event Transient (SET) simulation script, our methodology progresses to a meticulous analysis of each sensitive node within the circuit. This entails evaluating the extent of pulse broadening induced by the simulated SETs and cataloguing the unique identifiers of affected nodes. To facilitate this analysis, a specialized script was developed to parse through the data, extracting the pertinent information regarding the breadth of impact on each node.

Subsequent to this analytical phase, our approach leverages the Libero SoC software suite to generate a Verilog netlist of the circuit, specifically the iteration subjected to SET analysis across varying pulse widths. In response to the insights garnered from the SET impact reports, another bespoke script was conceived. The core objective of this script is to amend the circuit design in a manner that effectively mitigates the observed radiation effects. The proposed modification entails the integration of a delay mechanism, ingeniously constructed from a series of inverters coupled with an AND gate. This configuration exploits the inherent gate delays to introduce a corrective temporal offset when an SET event is detected, thereby averting potential bit flips and preserving the integrity of the circuit's operation in accordance with the specified parameters.

3.5 Delay Circuit: Inverters and AND Gates

Delay circuits introduce a controlled delay in the signal path, effectively filtering out transient pulses that are short-lived. The combination of inverters and AND gates can create such delays (14). Here's a breakdown of how these components work together to mitigate SET effects:

Inverters

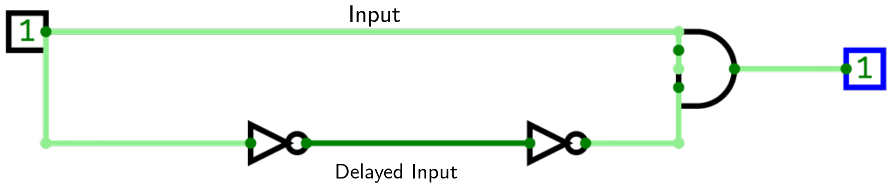
- **Inverting Logic State:** An inverter flips the logic state of a signal (e.g., from 0 to 1 or 1 to 0). (14)

- **Introducing Delay:** Each inverter introduces a small propagation delay due to the time it takes for the signal to travel through the transistor gate. By chaining multiple inverters, you can achieve a significant cumulative delay.

AND Gates

- **Logic Condition:** An AND gate outputs a 1 only if all its inputs are 1; otherwise, it outputs 0. (14)

- **SET Filtering:** By using an AND gate in conjunction with delayed signals, you can effectively filter out transient pulses. The idea is that a transient pulse on one input will not align with the delayed signals on other inputs, thus preventing an incorrect output.



15 Example of type of Delay Circuit Inserted

3.5.1 Implementation Strategy

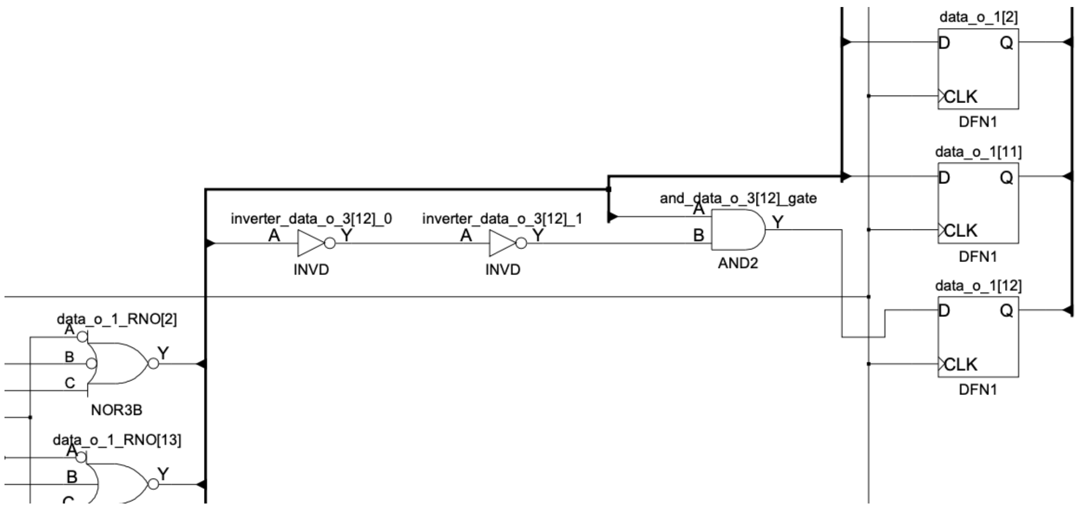
Signal Delaying with Inverters:

- **Delay Line:** Create a delay line using a series of inverters. The number of inverters determines the total delay introduced to the signal.
- **Delayed Signal:** The output of the delay line provides a version of the original signal delayed by a specific amount of time.

Combining Signals with an AND Gate:

- **Original and Delayed Signals:** Feed both the original and delayed signals into an AND gate.
- **Pulse Filtering:** Since an SET-induced pulse is typically very short, it will not appear on both the original and delayed signals simultaneously. The AND gate will output a logic high only when both signals are high, effectively filtering out the transient pulse.

The implementation strategy involves a precise identification of each component within the Verilog netlist, followed by the extraction of its input signal. The mitigation circuitry characterized by a calculated sequence of inverters designed to match the maximum observed pulse broadening—is then inserted at strategic points within the circuit. This modification process takes the original signal from each identified component and subjects it to a controlled delay, effectively replicating the initial signal with a calculated temporal displacement. The modified signal is then rerouted to the input of the affected component, thus providing a robust countermeasure against the disruptive effects of SETs.



16 Modified Circuit snippet showing delay circuit

However, it is pertinent to note that this mitigation technique is contingent upon the magnitude of the pulse broadening observed at each node. Given the practical limitations of introducing delay elements, this strategy is selectively applied to components exhibiting a maximum delay exceeding a predefined threshold. This ensures that the corrective measures are both feasible and effective, targeting the most susceptible components within the circuit while maintaining overall system performance and reliability. (15)

Chapter 4

4 Experimental Analysis

Once the circuit placement and the selective implementation of the mitigation strategies were completed, as determined from the previously gathered reports, a thorough analysis of the modified circuit was conducted across various critical parameters.

4.1 Resource Utilization Analysis

The first step involved assessing the additional resource requirements introduced by the mitigation strategies. This step was crucial to ensure that the FPGA had sufficient capacity to accommodate the extra hardware components, such as the inverters and AND gates, integrated for delay-based SET mitigation. The analysis focused on quantifying the increase in logic elements, interconnect usage, and any other relevant resources. The goal was to verify that the enhancements did not exceed the FPGA's resource limits and could be seamlessly integrated into the existing design framework.

4.2 Timing and Power Constraints Verification

Following the resource utilization analysis, the next phase involved verifying that the modified circuit met the stringent timing and power constraints necessary for its intended operational environment.

4.2.1 Timing Analysis

This included checking setup and hold times, propagation delays, and overall timing closure to ensure that the circuit performed its functions within the required temporal parameters. Given the low power and high precision demands of space applications, it was imperative that the timing analysis confirmed the circuit's ability to operate without timing violations despite the introduced delays for SET mitigation.

	Original Circuit	Modified Circuit
Core Components	4634	4917
COMB Circuit	3992	4275
Frequency	44.226	50.566
Max Delay (ns) (Register to Register)	22.428	19.348
Max Clock-To-Out (ns)	7.387	3.912
Max delay (Input to Output)	5.132	3.907
Min Clock-To-Out (ns)	2.178	2.218

Upon conducting a comprehensive resource analysis, it was determined that the incorporation of the delay circuits into the original design resulted in a minimal increase of approximately 6% in the core components. This modest increment was deemed acceptable given the substantial benefit of enhancing the circuit's resilience against Single Event Transients (SETs).

4.2.2 Negligible Impact on Timing Delay

The strategic insertion of delay circuits at specific, critical points within the circuit architecture was executed with precision, ensuring that these modifications did not introduce significant timing delays. As a result, there were no violations of the minimum timing requirements essential for the circuit's operation. This meticulous placement ensured that the overall timing integrity of the circuit was maintained.

During the simulation phase, it was observed that the automated optimization processes inherently adjusted the circuit frequency upwards. This adjustment was seamlessly integrated into the simulation workflow, confirming that the circuit could handle the increased frequency without adverse effects on power consumption. This indicates a well-balanced design where the benefits of added delay elements for SET mitigation are realized without detracting from the circuit's overall power efficiency.

4.2.3 Power Analysis:

In evaluating the power consumption of FPGA technologies, it is important to consider it from a system point of view. Generally, the overall power consumption should be based on static, dynamic, inrush, and configuration power. Few FPGAs implement ways to reduce static power consumption utilizing sleep modes.

ProASIC3/E Total Power Consumption = $P^{static} + P^{dynamic}$

Mode	Power Supplies / Clock Status	Needed to Start Up
Active	On – All, clock Off – None	N/A (already active)
Static (Idle)	On–All Off – No active clock in FPGA Optional: Enter User Low Static (Idle) Mode by enabling ULSICC macro to further reduce power consumption by powering down FlashROM.	Initiate clock source. No need to initialize volatile contents.
Sleep	On – VCCI Off – VCC (core voltage), VJTAG (JTAG DC voltage), and VPUMP (programming voltage) LAPU enables immediate operation when power returns. Optional: Save state of volatile contents in external memory.	Need to turn on core. Load states from external memory. As needed, restore volatile contents from external memory.

Shutdown	On – None Off – All power supplies Applicable to all ProASIC3E, all ProASIC3 nano, and the A3P030 and A3P015 devices, cold-sparing and hot-insertion allow the device to be powered down without bringing down the system. LAPU enables immediate operation when power returns.	Need to turn on VCC, VCCI.

Power consumption was scrutinized to ensure the modifications did not lead to excessive power usage, which could compromise the low power requirements crucial for spaceborne and other energy-sensitive applications. This involved both static and dynamic power analysis to capture the complete power profile of the modified circuit.

At 30 Deg Celsius

	Active	Sleep	Static
Total Power	8.705	0.104	8.220
Static power	8.220 (94.4%)	0.104	8.220
Dynamic Power	0.485 (5.6%)	0	0

At 70 Deg. Celsius (Worst)

	Active	Sleep	Static
Total Power	44.947	0.235	44.370
Static power	44.370 (98.7%)	0.235	44.370
Dynamic Power	0.577 (1.3%)	0	0

At 0 Deg (Typical)

	Active	Sleep	Static
Total Power	8.156	0.104	7.755
Static power	7.755 (95.1%)	0.104	7.755
Dynamic Power	0.401 (4.9%)	0	0

Breakdown by Type

	Power (mW	Percentage
Type Net	0.000	0.0%
Type I/O	0.401	4.9%
Type Core Static	6.555	80.4%
Type Banks Static	1.200	14.7%

To offset any minor timing adjustments introduced by the delay circuits, the circuit's operating frequency was slightly increased. Remarkably, this frequency increment did not result in an appreciable rise in power consumption. Simulations demonstrated that the circuit could operate at the enhanced frequency without exceeding the predefined power budget. This outcome is particularly advantageous, as it signifies that the circuit can achieve improved performance metrics while simultaneously mitigating SET effects without compromising power efficiency.

4.3 Principal Objective Verification

The final and most critical parameter was to determine the efficacy of the implemented mitigation strategies in achieving the principal objective of the thesis: reducing the impact of SETs without overburdening the system.

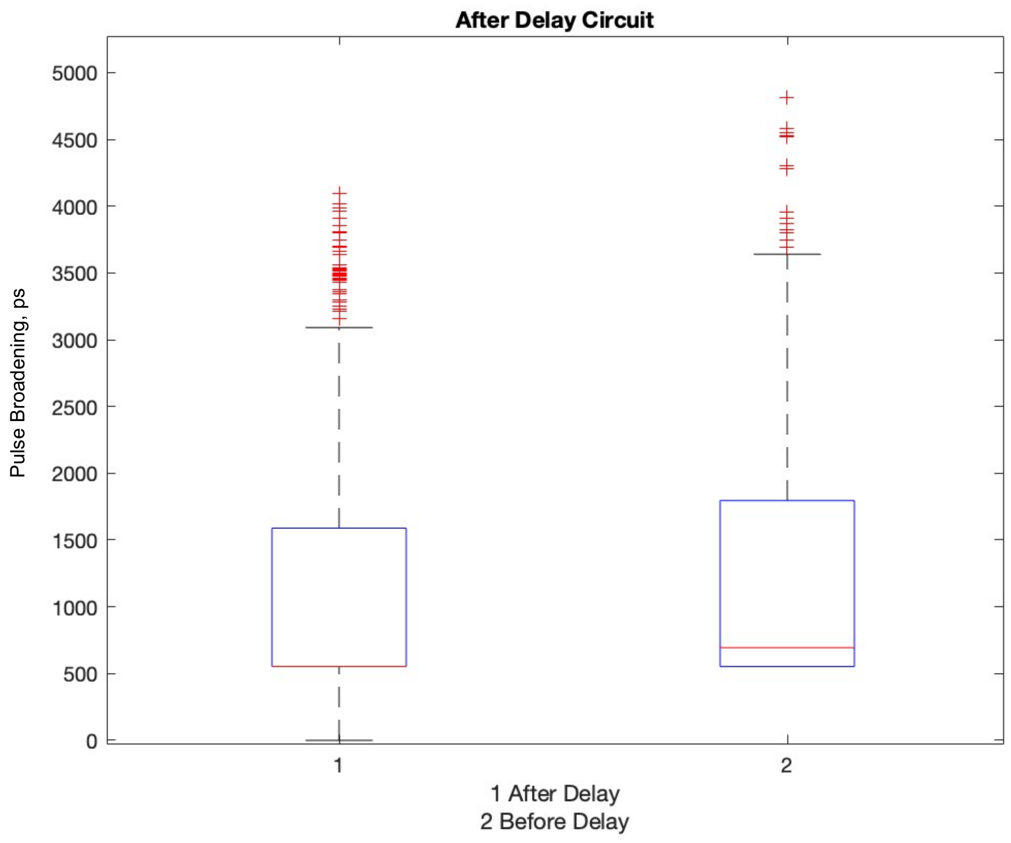
4.3.1 SET Mitigation Effectiveness

This was evaluated by simulating SETs on the modified circuit and comparing the results to the baseline circuit without mitigation. The key metrics included the frequency and severity of bit flips and other transient faults. The analysis aimed to confirm that the mitigation measures successfully reduced the occurrence and impact of SETs on sensitive nodes.

The highest value of 550 picoseconds for the Single Event Transient (SET) impact width was selected for this study, as it necessitated the insertion of the greatest number of delay circuits. Delay circuits were strategically implemented at points where the SET radiation impact exceeded the minimum inverter delay threshold, resulting in the integration of delay circuits at 170 critical locations within the circuit.

4.3.2 Analysis of Delay Circuit Effectiveness: Box Plot Comparison

The attached graph illustrates the comparative impact of SET radiation on the circuit before and after the insertion of delay circuits. Notably, the nodes experiencing delays above the minimum inverter delay threshold of 4200 picoseconds were effectively mitigated in the modified circuit.



17 Box Plot Comparison after and before inserting the delay component

Before Delay Circuit Implementation:

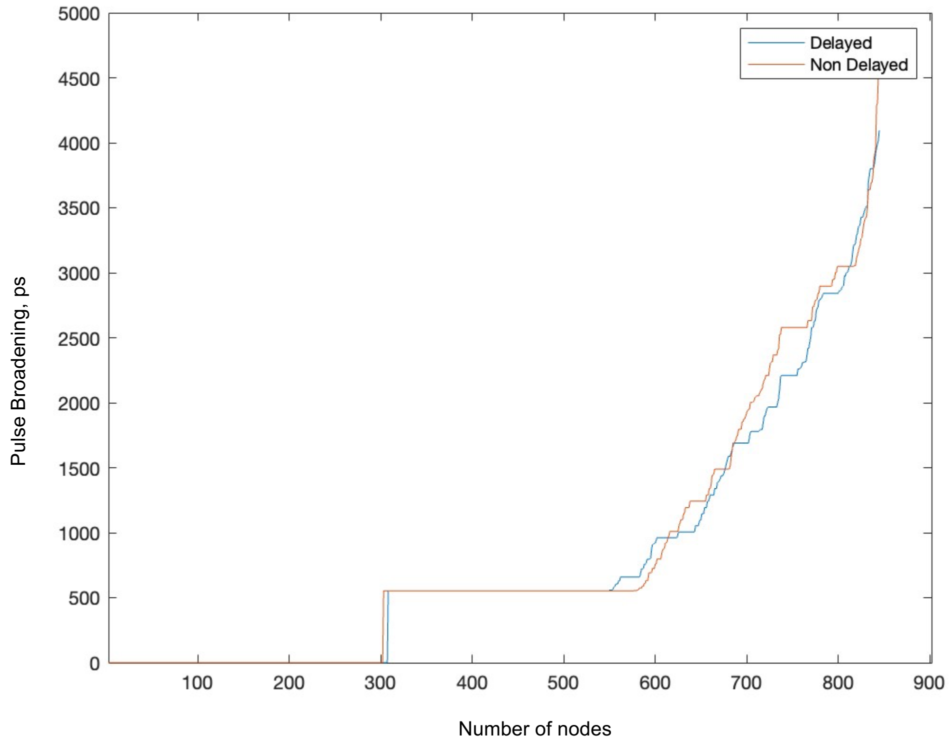
Significant impact of SET radiation was observed at several nodes, with delay values often exceeding the threshold, indicating susceptibility to transient radiation effects.

The distribution before delay circuit insertion shows a higher median delay and a wider interquartile range, demonstrating greater variability and susceptibility to SETs.

After Delay Circuit Implementation:

The impact of transient radiation at several critical nodes was substantially reduced. The delay circuits successfully filtered out transient pulses, thereby enhancing the circuit's robustness against SETs. The distribution of delays after implementing delay circuits shows a lower median value and a tighter interquartile range compared to the non-delayed circuit. The median delay is significantly reduced, indicating a successful mitigation of SET effects.

The whiskers for the delayed circuit are shorter than those for the non-delayed circuit, suggesting that the range of delays experienced by the delayed circuit is more constrained and controlled.



18 Cumulative Distribution Graph

The provided graph depicts the cumulative distribution of SET-induced delays in the circuit, comparing scenarios before and after the insertion of delay circuits. The x-axis represents the different nodes in the circuit, while the y-axis shows the delay in picoseconds. The blue line indicates the delayed circuit (after delay circuit insertion), and the orange line represents the non-delayed circuit (before delay circuit insertion).

4.3.3 Key Observations

- **Overall Trend**

  Both lines exhibit a similar trend initially, with minimal differences in delays up to approximately 300 nodes. This indicates that, for a significant portion of the circuit, the impact of SETs is either low or well-managed in both scenarios.

- **Divergence Point**

  The lines start to diverge noticeably after the 300-node mark. This divergence indicates the point where the impact of SETs begins to differ significantly between the delayed and non-delayed circuits.

- **Delayed Circuit Performance**

  - The blue line (delayed circuit) shows a generally smoother and more gradual increase in delays, suggesting a more consistent handling of SETs across the nodes.

  - The presence of the delay circuits effectively mitigates the impact of SETs, leading to fewer instances of extreme delay spikes.

- **Non-Delayed Circuit Performance**

  - The orange line (non-delayed circuit) shows a sharper increase in delays beyond the 300-node mark, indicating higher susceptibility to SET-induced delays.
  - This sharper increase is indicative of more frequent and severe SET impacts in the absence of delay circuits.

- Peak Delays
  - At the upper end of the graph, the delayed circuit (blue line) demonstrates a plateau, suggesting that the maximum delay introduced by SETs is capped at a lower level compared to the non-delayed circuit.
  - The non-delayed circuit (orange line) shows more variability and higher peak delays, reflecting greater vulnerability to SETs.

Chapter 5

5 Conclusion

This thesis aimed to address the critical challenge of mitigating the impact of Single Event Transients (SETs) on digital circuits, particularly within the context of space applications where radiation-induced errors can significantly compromise system reliability. The approach involved integrating delay circuits strategically within the circuit design to counteract the effects of transient pulses caused by high-energy particles.

5.1 Key Findings and Contributions

5.1.1 Implementation of Delay Circuits

The study began by deriving the circuit from the base RTL VHDL files of the NEORV32 framework and developing a matrix multiplication test program in C, compiled into instruction memory data using the RISC-V GCC Toolchains for Linux.

Delay circuits, comprising inverters and AND gates, were inserted at specific points where the SET impact was significant, as determined from detailed analysis reports.

5.1.2 Resource Utilization and Performance

The resource analysis revealed that the inclusion of delay circuits resulted in a minimal 6% increase in core components, which was within acceptable limits given the substantial benefits in SET mitigation.

Timing analysis confirmed that the delay circuits did not introduce significant timing delays, nor did they violate minimum timing requirements. The slight increase in operating frequency, automatically adjusted during simulation, did not lead to additional power consumption.

5.1.3 Effectiveness of SET Mitigation

The implementation of delay circuits effectively reduced the impact of transient radiation on critical nodes. The comparative analysis of the cumulative distribution graph and the box plot demonstrated a significant reduction in both the median delay and the number of severe outliers after the delay circuits were introduced.

These findings validated that the delay circuits provided a robust defence against SETs, resulting in a more stable and predictable circuit performance.

6 Future Work

The successful mitigation of SETs using delay circuits has significant implications for the design of radiation-hardened electronic systems, particularly in space applications. This methodology provides a practical and efficient approach to enhancing circuit reliability in radiation-prone environments.

In this research, we focused on testing the reliability and hardening of the entire circuit. However, in real-world applications, each process and application is unique, utilizing different components and configurations. Depending on the specific use case, future work could target different parts of the system selectively. Some components might be critical to the mission's success, while others may be less significant. By identifying and prioritizing critical components for targeted hardening, we can optimize resource usage and enhance overall system reliability more efficiently. This approach will allow for tailored mitigation strategies that address the specific needs of different applications and operational environments.

Future work could explore the following areas:

- Optimization Techniques: Further optimization of the delay circuit design to minimize resource usage and enhance performance.

- Advanced Mitigation Strategies: Combining delay circuits with other mitigation strategies, such as redundancy and error correction codes, to further improve reliability.

- Field Testing: Conducting field tests in actual radiation environments to validate the effectiveness of the delay circuits in real-world scenarios.

7 Bibliography

1. ***The Case for RISC-V in Space.*** **Mascio, Stefano Di.** 2019, Applications in Electronics Pervading Industry, Environment and Society.

2. **Petersen, .** Single-Event Effects in Aerospace. [book auth.] Edward Petersen. s.l. : John Wiley & Sons, 2011, pp. 13-102.

3. **Fleetwood, and Schrimpf, .** ***Radiation Effects and Soft Errors in Integrated Circuits and Electronic Devices.*** s.l. : World Scientific Publishing, 2004.

4. **Baumann, R. C.** Radiation-induced soft errors in advanced semiconductor technologies. ***Transactions on Device and Materials Reliability.*** s.l. : IEEE, 2005. Vol. 5, pp. 305-316. https://ieeexplore.ieee.org/abstract/document/1545891.

5. ***Single Event Transient on Combinational Logic: An Introduction and their Mitigation.*** **Henrique, , et al.** 3, s.l. : Journal of Integrated Circuits and Systems, 2022, Journal of Integrated Circuits and Systems, Vol. 17.

6. **Advanced FPGA Design: Architecture, Implementation, and Optimization. [book auth.] S. Kilts. *Advanced FPGA Design: Architecture, Implementation, and Optimization.* s.l. : John Wiley & Sons, 2007.**

7. **Wirthlin, . High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond. *Proceedings of the IEEE.* s.l. : IEEE, 2015. Vol. 103.**

8. **NEORV32. NEORV32 Project Documentation. [Online] https://stnolting.github.io/neorv32/.**

9. Actel. ProASIC3/E Production FPGAs. [Online] https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/SupportingCollateral/pa3_e_tech_wp.pdf.

10. Microchip. Libero SOC User Guide. [Online] https://ww1.microchip.com/downloads/aemdocuments/documents/fpga/ProductDocuments/UserGuides/libero_v116_ug.pdf.

11. Brown, Stephen Dean. ***Routing Algorithms and Architectures for Field-Programmable Gate Arrays.*** 1992.

12. Sarrafzadeh, and & Wong, C. K. ***An Introduction to VLSI Physical Design.*** s.l. : McGraw-Hill, 1996.

13. ***Analysis of SET Propagation in Flash-Based FPGAs by Means of Electrical Pulse Injection.*** L., , N., and V., . 2010 : IEEE Transactions on Nuclear Science, Vol. 54, pp. 1820-1830.

14. Microsemi. Using the BUFD and INVD Delay Macros. [Online] https://www.microsemi.com/document-portal/doc_view/129887-ac147-using-the-bufd-and-invd-delay-macros.

15. Weste, N. H. E. and Harris, . ***CMOS VLSI Design: A Circuits and Systems Perspective.*** 2010.

16. Punia, . [Online] 2023. https://www.logic-fruit.com/blog/fpga/fpga-design-architecture-and-applications/.

17. Waterman, . The RISC-V Instruction Set Manual, Volume 1. [Online] May 2014. http://www2.eecs.berkeley.edu/Pubs/ TechRpts/2014/EECS-2014-54.html.

18. ***A Technique to Reduce Glitch Power during Physical Design Stage for Low Power and Less IR Drop.*** Murti sarma, Nimushakavi Satyanarayana.P, Vasantha. s.l. : International Journal of Computer Applications, 2012.

8 Appendix

8.1 Extraction and Inserting Delay Circuit

First Step- It extracts where to insert the delay circuit by checking if propagation value is greater than minimum delay of circuit to be inserted.

Second Step- Finding corresponding component in Verilog circuit netlist and modify it to insert the desired circuit to mitigate the effect.

```
import csv
import re
# Replace 'path_to_your_file.txt' with the path to your SET report file
file_path = '/Users/adi/untitled folder 2/SET_report_552ps.txt'
# Specify the path to your folder containing Verilog Netlist file
folder_path = '/Users/adi/ts/neorv32_top_original.v'  # Update with your file
path

def extract_pdd_and_pulse(file_path):
    pdd_pulse_data = []

    with open(file_path, mode='r') as file:
        reader = csv.reader(file, delimiter=';')
        for row in reader:
            if len(row) > 1:  # Ensure the row has multiple elements
                pdd_name = row[0].strip()
                if pdd_name.startswith("neo"):
                    max_pulse = row[-3].strip()  # Assuming the last value is
the max pulse width
                    pdd_pulse_data.append((pdd_name, max_pulse))

    return pdd_pulse_data

def extract_signal_pdd(pdd_name):
    # Find the last occurrence of '/'
    last_inst_index = pdd_name.rfind('_inst')
    return pdd_name[last_inst_index+6:]

def generate_inverter_chain_with_and(input_signal, num_inverters,
final_output_signal):
    verilog_code = ""
    previous_signal = input_signal

    # Generate the chain of inverters
    for i in range(num_inverters):
        current_signal = f"\intermediate_{i}_{final_output_signal}"
        verilog_code += f"INVD \inverter_{final_output_signal}_{i}
(.A({previous_signal} ), .Y({current_signal} ));\n"
        previous_signal = current_signal

    # Add an AND gate that takes the output of the last inverter and the
original input signal
    verilog_code += f"AND2 \\and_{final_output_signal}_gate (.A({input_signal}
), .B({previous_signal} ), .Y(\{'Delayed_'+ final_output_signal} ));\n"

    return verilog_code

def generate_inverter_chain_with_and2(input_signal, num_inverters): # Signal
declarations list
    signal_declarations = []

    # Generate the declarations of  the inverters
    for i in range(num_inverters):
        current_signal2 = f"intermediate_{i}_{input_signal}"
        signal_declarations.append(current_signal2)
    signal_declarations.append(input_signal)
    wire_declarations = " ,\\".join(signal_declarations)
    wire_declaration_code = f" \{wire_declarations} ,"
    return wire_declaration_code


def parse_vhdl_entity_names(folder_path,signal,max_pulse):
    entities = set()
    with open(folder_path, 'r') as file:
            content = file.read()  # Read the whole file into a single string
            if signal != None:
             escaped_signal = re.escape(signal)
             flexible_pattern = r'\\' +re.sub(r'_', r'[._]', escaped_signal) +
r' ' # Putting both _ as . for search pattern
             pattern = re.compile(flexible_pattern)

            match = pattern.search(content)
            # Find first occurance
            if match:

              find_output_insert = content.rfind(';\n',0, match.start())+1
              find_output_signal_index = content.find('), .',
match.start(),match.start()+40)
              if find_output_signal_index != -1:
                find_output_signal_index_start=
content.rfind('(',find_output_signal_index-25,find_output_signal_index)
                insertion_index1 = content.rfind('wire',0,match.start())+5 #
Index to find where to declare


                if find_output_signal_index_start !=-1:

                 signal_name_chain =
content[find_output_signal_index_start+1:find_output_signal_index]
                 signal_name_chain2= signal_name_chain.replace("\\", "") #
Removing spaces and tabs
                 signal_name_chain2= signal_name_chain2.replace("\t", "")
                 signal_name_chain2= signal_name_chain2.replace("\n", "")
                 signal_name_chain2= signal_name_chain2.replace(" ", "")
                 signal_name_chain2= signal_name_chain2.replace("\n", "")
                 if signal_name_chain2:

                  delay_ps = float(max_pulse)
                  num_inverters = round(delay_ps/4200)*2 # Adding filter for
minimum gate date
                  if num_inverters != 0:
                    #print(delay_ps)
                    #print(num_inverters)
                    verilog_code =
generate_inverter_chain_with_and(signal_name_chain ,
num_inverters,signal_name_chain2) # Generate Code of chain
                    verilog_codeS =
generate_inverter_chain_with_and2('Delayed_'+ signal_name_chain2 ,
num_inverters, signal_name_chain) # Generate Signal Declarations

                    updated_content = content[:insertion_index1] +
verilog_codeS +"\n" +content[insertion_index1:find_output_insert+1] + "\n"+
verilog_code +
"\n"+content[find_output_insert+1:find_output_signal_index_start+1]
+'\Delayed_' + signal_name_chain2 + ' '  + content[find_output_signal_index:]

                    with open(folder_path, 'w') as file:
                      file.write(updated_content)
    return entities

def perform_analysis(signal,max_pulse):
    entity_names = parse_vhdl_entity_names(folder_path,signal,max_pulse)
    return -1

signals=[]
data = extract_pdd_and_pulse(file_path)
```

```
# Assuming 'data' is the list of tuples with PDD names and max pulses extracted
from the file
for pdd_name, max_pulse in data:
    #last_part = extract_last_part_of_pdd(pdd_name) To find which component
signal belong to
    signal= extract_signal_pdd(pdd_name)
    perform_analysis(signal,max_pulse)

    #signals.append((signal, max_pulse))
# For parallelezing the code
#Parallel(n_jobs=1,prefer="threads")(delayed(perform_analysis)(signal,max_pulse
)for signal,max_pulse in unique_signals_list)
```