Automated SEU Fault Injection Framework Using the Xilinx SEM IP on Zynq UltraScale+

by

Aryan ABRARI VAJARI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN ELECTRICAL ENGINEERING
M.A.Sc.

MONTREAL, AUGUST 25, 2026

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Dr. Claude Thibeault, Thesis Supervisor
Department of Electrical Engineering, École de technologie supérieure

Dr. Mostafa Darvishi, Thesis Co-supervisor
Department of Electrical Engineering, École de technologie supérieure

Dr. Ricardo Izquierdo, President of the Board of Examiners
Department of Electrical Engineering, École de technologie supérieure

Dr. Dominic Deslandes, Member of the jury
Department of Electrical Engineering, École de technologie supérieure

THIS THESIS WAS PRENSENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

ON AUGUST 4, 2026

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support and guidance of many people to whom I am deeply grateful.

First and foremost, I would like to express my sincere gratitude to my supervisors, Dr. Claude Thibeault and Dr. Mostafa Darvishi, for their invaluable guidance, patience, and expertise throughout this research. Their insightful feedback and encouragement shaped this work in ways I could not have achieved alone.

I am profoundly thankful to my parents, whose unwavering support and countless sacrifices gave me the opportunity to pursue my education and follow my ambitions. Everything I have accomplished is built on the foundation they provided, and I hope this work is a small reflection of their devotion.

To my beloved Niloufar, thank you for your love, your patience, and for being a steady source of warmth and encouragement through every difficult moment. No word can describe how much your presense changed and brightened my life.

I also owe a deep thank you to my family Parviz, Grace, Omid, and Mona, whose incredible support throughout my study far from home meant more to me than words can express. Knowing you were there gave me a sense of grounding that carried me through difficult times in immigration.

To my friends Amin, Milad, Mohammad Hasan, and Nima, thank you for the laughs, deep conversations, and for reminding me that there is life beyond the lab.

Finally, I am grateful to the faculty and staff of the École de technologie supérieure for providing an environment where I could study and this research could flourish.

To all of you.

Cadre Automatisé d'Injection de Fautes SEU par le Cœur IP SEM de Xilinx sur Zynq UltraScale+ : Évaluation de la Fiabilité de Conceptions RTL et de Processeurs Souples RISC-V

Aryan ABRARI VAJARI

RÉSUMÉ

Les circuits intégrés programmables par l'usager (Field Programmable Gate Array, FPGA) sont largement déployés dans des environnements critiques et exposés aux rayonnements, notamment dans le domaine aérospatial, ainsi que ceux de la physique des hautes énergies et des systèmes d'intelligence artificielle embarquée. Dans les FPGA à base de SRAM, la mémoire de configuration est vulnérable aux perturbations de bits uniques (Single Event Upsets, SEU), qui sont des événements de basculement de bit causés par des impacts de particules ionisantes à haute énergie. Un SEU dans la mémoire de configuration peut altérer silencieusement la logique d'un circuit implémenté et persister jusqu'à ce que le dispositif soit reconfiguré, faisant de l'évaluation de la fiabilité des conceptions FPGA une préoccupation fondamentale en ingénierie. Le cœur IP de mitigation des erreurs douces (Soft Error Mitigation,SEM) d'AMD fournit une solution intégrée pour émuler les SEU par injection contrôlée de bits dans la mémoire de configuration des FPGA, afin d'étudier la fiabilité des conceptions FPGA. Malgré la puissance de ce coeur IP, les campagnes complètes d'injection de pannes utilisant le SEM IP ont nécessité un effort manuel considérable, limitant la portée et la reproductibilité des études de fiabilité.

Cette thèse présente un cadre automatisé basé sur Python qui pilote le SEM IP via une interface UART afin de mener des campagnes systématiques d'injection de pannes sur la plateforme FPGA avec un minimum d'interventions manuelles. Le cadre automatise le cycle complet d'injection : initiation de l'injection, surveillance de la réponse du circuit sous test (DUT) via UART, journalisation des résultats et réinitialisation du SEM IP pour l'injection suivante. Trois études de cas démontrent l'applicabilité du cadre à différents types de conception. La première est une preuve de concept basée sur une machine à états finis (FSM) implémentée sous forme de trois instances parallèles pilotées par un registre à décalage à rétroaction linéaire configurable (cLFSR). Les deuxième et troisième études sont des campagnes d'injection de fautes sur deux cœurs de processeur souple RISC-V32I exécutant respectivement un générateur de suite de Fibonacci et un classificateur d'images binaires en virgule fixe.

Les résultats quantifient la fraction de bits critiques de chaque conception et caractérisent les comportements de défaillance produits par les SEU, notamment la corruption silencieuse des données, les erreurs de calcul et les blocages système. Le cadre est conçu pour être indépendant de la conception, ne nécessitant qu'un canal de communication UART, et est extensible à toute cible compatible avec le SEM IP. Ce travail comble une lacune dans l'évaluation automatisée de la fiabilité à l'aide du SEM IP d'AMD et fournit une méthodologie reproductible pour la caractérisation des SEU avant déploiement des conceptions à base de FPGA.

Mots-clés : FPGA, perturbation de bit unique (SEU), injection de fautes, SEM IP, RISC-V, évaluation de la fiabilité, apprentissage profond, Zynq UltraScale+, automatisation

Automated SEU Fault Injection Framework Using the Xilinx SEM IP on Zynq UltraScale+: Reliability Assessment of RTL Designs and RISC-V Soft-Core Processors

Aryan ABRARI VAJARI

ABSTRACT

Field Programmable Gate Arrays (FPGAs) are widely deployed in safety-critical and radiation-exposed environments, including aerospace, high-energy physics, and edge AI systems. In SRAM-based FPGAs, the configuration memory is vulnerable to Single Event Upsets (SEUs), which are bit flip events caused by ionising high-energy particle strikes. An SEU in configuration memory can silently alter the logic of an implemented circuit and remain until the device is reconfigured, making reliability assessment of FPGA designs a core engineering concern. AMD[1]'s Soft Error Mitigation (SEM) IP core provides a built-in solution for emulating SEUs by controlled bit injection into the configuration memory of FPGAs to investigate the reliability of FPGA designs. Despite the capabilities of this IP core, comprehensive fault injection campaigns using the SEM IP have required significant manual effort, limiting the scope and reproducibility of reliability studies.

This thesis presents an automated Python-based framework that drives the SEM IP over a UART interface to conduct systematic fault injection campaigns on the FPGA platform with minimum manual intervention. The framework automates the full injection cycle: initiating an injection, monitoring the design-under-test (DUT) response via UART, logging outcomes, and resetting the SEM IP for the next injection. Three case studies demonstrate the framework's applicability across different design types. The first is a finite state machine (FSM) proof-of-concept implemented as three parallel instances driven by a configurable Linear Feedback Shift Register (cLFSR). The second and third are fault injection campaigns on two RISC-V32I soft-processor cores running a Fibonacci sequence generator and a fixed-point binary image classifier, respectively.

The results quantify the critical bit fraction of each design and characterize the failure behaviours produced by SEUs, including silent data corruption, computation errors, and system stalls. The framework is designed to be design-agnostic, requiring only a UART communication channel, and is extensible to any SEM IP-compatible target. This work addresses a gap in automated reliability assessment using AMD SEM IP and provides a reproducible methodology for pre-deployment SEU characterization of FPGA-based designs.

Keywords: FPGA, Single Event Upset(SEU), fault injection, SEM IP, RISC-V, reliability assessment, deep learning, Zynq UltraScale+, automation

---

1. Formerly known as Xilinx Inc.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS AND ACRONYMS

ABI	Application Binary Interface
APU	Application Processing Unit
ARM	Advanced RISC Machine
AXI	Advanced eXtensible Interface
BRAM	Block Random-Access Memory
CB	Critical Bit
CIFAR	Canadian Institute for Advanced Research
CMOS	Complementary Metal-Oxide Semiconductor
CRAM	Configuration Random-Access Memory
DNN	Deep Neural Network
DPR	Dynamic Partial Reconfiguration
DUT	Design Under Test
EBD	Essential Bits Database
ÉTS	École de technologie supérieure
EMIO	Extended Multiplexed I/O
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GCR	Galactic Cosmic Ray
HDL	Hardware Description Language
ICAP	Internal Configuration Access Port
ILA	Integrated Logic Analyzer
ISA	Instruction Set Architecture
ITC	International Test Conference
JTAG	Joint Test Action Group
LET	Linear Energy Transfer
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look-Up Table

MSB	Most Significant Bit
PCAP	Processor Configuration Access Port
PL	Programmable Logic
PS	Processing System
Q16	16-bit Q16 Fixed-Point (1 sign, 7 integer, 8 fractional bits)
ReLU	Rectified Linear Unit
RISC-V	Reduced Instruction Set Computer — Version Five
RTL	Register Transfer Level
SDC	Silent Data Corruption
SEM	Soft Error Mitigation
SER	Soft Error Rate
SEU	Single Event Upset
SoC	System on Chip
SPE	Solar Particle Event
SRAM	Static Random-Access Memory
TMR	Triple Modular Redundancy
UART	Universal Asynchronous Receiver-Transmitter
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

CHAPTER 1

INTRODUCTION

1.1 Motivation

Field Programmable Gate Arrays (FPGAs) have become essential components in modern electronic systems, valued for their reconfigurability, parallel processing capabilities, and versatility across applications ranging from telecommunications and data centers to machine learning acceleration and aerospace [46]. The dominant FPGA technology today is based on Static Random-Access Memory (SRAM), where every logic function, routing connection, and I/O setting is defined by bits stored in a large configuration memory array [3]. This architecture enables rapid design iteration and in-field updates, but it also introduces a fundamental reliability concern: the SRAM cells that define the device's behavior are susceptible to radiation- or configuration-induced bit flips known as Single Event Upsets (SEUs) [11].

An SEU occurs when a high-energy particle, typically a neutron from cosmic ray showers at terrestrial altitudes or a heavy ion in space, deposits sufficient charge at a sensitive node to flip the stored bit. The sensitivity of a device to SEUs varies significantly with process technology, as documented in AMD (formerly Xilinx) application note UG116 [3]. Modern nanometer-scale devices exhibit different upset cross-sections compared to older technology nodes, and the relationship is not strictly monotonic. When an SEU occurs in user logic or data registers, the effect is transient and self-correcting on the next clock write. However, when an SEU occurs in the configuration memory, the behavior is fundamentally different: the upset alters the circuit's structure, changing routing connections or logic functions, and persists silently until the device is explicitly reconfigured [11]. The resulting failure can range from a minor output deviation to a complete malfunction, depending on which configuration bit was affected.

This distinction makes configuration memory SEUs a particularly serious concern for FPGA-based systems deployed in radiation-prone environments, including satellites, aircraft, high-energy physics experiments, and increasingly, edge AI platforms that operate at high

altitudes or in industrial environments with elevated neutron flux. As FPGA designs grow in complexity and are trusted with safety-critical functions, the need for rigorous, systematic, and reproducible reliability assessment methods has become a core engineering challenge [3].

1.2 Limitations of the AMD SEM IP for Fault Injection

AMD provides the Soft Error Mitigation (SEM) IP core, documented in PG187 [30], as an integrated solution for configuration memory error detection and correction in AMD FPGAs. The SEM IP interfaces directly with the internal configuration infrastructure through the FRAME_ECC primitive, which provides hardware-accelerated Hamming code-based error detection. Beyond its runtime protection role, the SEM IP exposes a fault injection interface that allows a user to deliberately flip a specific bit in the configuration memory at a specified frame address and bit offset. This makes it a natural engine for pre-deployment reliability assessment: rather than waiting for a radiation event, the designer can inject faults systematically and observe their effects on the running design [48].

Despite this capability, using the SEM IP for comprehensive fault injection campaigns presents several practical limitations. First, the injection interface is command-driven over UART, meaning that without an automation layer, each injection requires manual interaction: sending the injection command, observing the design-under-test (DUT) output, recording the outcome, resetting the SEM IP, and preparing the next injection. For a design with tens of thousands of configuration bits, this process is entirely impractical to perform manually. Second, no general-purpose automation framework for SEM IP-based injection exists for the Zynq UltraScale+ platform, leaving researchers to build custom tooling from scratch for each new study [17]. These limitations together have constrained the scope and reproducibility of using SEM IP for reliability studies, and represent the gap this thesis addresses.

1.3 Contributions

The primary contribution of this work is an automation methodology that leverages the SEM IP's error injection capability to systematically evaluate FPGA designs under an emulated radiation condition. Our framework integrates the AMD SEM IP targeting the Zynq UltraScale+ platform, the automation process is controlled and monitored by a Python-based

setup running on a PC through UART interface for communication between FPGA and PC. By automating the entire fault injection workflow, we enable rapid assessment of design vulnerability against SEUs by identification of critical configuration regions, and monitoring the misbehaviour of the design under the circumstance of a SEU fault happening, a task that would otherwise require weeks or months of manual experimentation depending on the extent of the RTL design. The automation framework is designed to be scalable and independent from RTL design, allowing researchers and engineers to quickly evaluate the reliability of diverse FPGA implementations without deep expertise in SEM IP operation or manual fault injection procedures.

The framework is validated through three case studies of increasing complexity. The first is a finite state machine (FSM) proof-of-concept, implemented as three parallel instances driven by a configurable Linear Feedback Shift Register (cLFSR) generating pseudo-random input sequences. This study serves as a functional validation of the injection and monitoring methodology. The second and third case studies target two RISC-V32I soft-processor cores running in parallel: one executing a Fibonacci sequence generator and one executing a fixed-point Q16 binary image classifier trained on a subset of the CIFAR-10 dataset. These studies extend the framework to processor-based designs and to deep learning inference workloads, respectively. Across all three studies, the framework identifies the critical bit fraction of each design and characterizes the failure modes produced by configuration memory SEUs, including silent data corruption, computation errors, and system stalls.

While this framework was developed and validated specifically for the Zynq UltraScale+ platform, the SEM IP is supported across many other AMD device families such as 7 Series and Ultrascale, and the framework is in principle adaptable to other AMD FPGAs. The Python-based automation layer, EBD parsing,[1] UART communication, injection sequencing, and result logging, is platform-independent and would require minor modification. However, the RTL integration of the SEM IP is heavily architecture-dependent. On the Zynq UltraScale+, a PS-side (Processing System) bare-metal application is required to disable the

1. EBD parsing: Extracting the FPGA CRAM addresses that are considered essential by Vivado

PCAP[1] interface and transfer configuration memory access to ICAP[2] before the SEM IP can operate; this initialization sequence is specific to Zynq devices and has no direct equivalent on pure PL (Programmable Logic) devices such as the Kintex or Virtex UltraScale+ families, where an alternative soft-core initialization must be designed instead. Beyond initialization, frame address layouts, configuration memory organization, and SEM IP instantiation constraints differ across device families, all of which require careful reading and consideration of the applicable device configuration user guide and PG187 before any porting effort. In summary, adapting the framework to a non-Zynq AMD FPGA would leave the Python automation largely intact, but would require redesign of the hardware-side SEM IP RTL integration.

1.4 Organization

The remainder of this thesis is organized as follows. Chapter 1 provides the background on soft errors and SEUs in SRAM-based FPGAs, the SEM IP architecture, and a literature review of existing fault injection methodologies and related reliability studies, closing with a research gap analysis. Chapter 2 describes the automated Python-based fault injection framework and presents the proof-of-concept validation results on a FSM design implemented as three parallel cLFSR-driven instances. Chapter 3 extends the framework to RISC-V32I soft-processor designs, presenting fault injection results for two workloads: a Fibonacci sequence generator and a fixed-point Q16 binary image classifier. Chapter 4 concludes the thesis with a summary of contributions and directions for future work.

---

1 .PCAP: Processor Configuration Access Port
2. ICAP: Internal Configuration Access Port

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Background

2.1.1 Soft Errors and Their Effect on Electronic Devices

Soft errors represent a significant reliability concern in modern electronic systems, particularly for memory elements and sequential logic circuits. Unlike hard errors (such as gate oxide breakdown and electromigration-induced open circuits) that result from permanent physical damage or manufacturing defects, soft errors are transient faults that temporarily alter the logical state of a device without causing lasting physical damage. The affected system can recover from a soft error through rewriting the corrupted data or resetting the device, hence the term "soft" to distinguish these failures from permanent "hard" failures [3].

2.1.2 Physics of Soft Errors

The physical mechanism underlying soft errors involves the interaction between energetic particles and semiconductor materials. When a high-energy particle strikes a semiconductor device, it deposits energy through ionization as it traverses the material, creating a dense track of electron-hole pairs along its path. The particles responsible for soft errors include alpha particles from radioactive contaminants in packaging materials, neutrons from cosmic ray interactions in the atmosphere, and heavy ions present in space radiation environments [11]. Figure 2.1 illustrates a bit flip induced by a soft error.

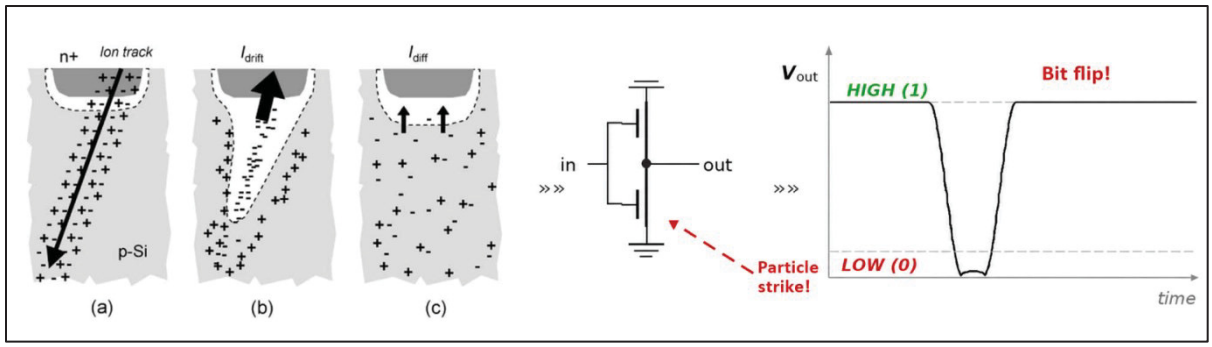


Figure 2. 1 A soft error happening in an n-type CMOS transistor causing bit flip. Adapted from Baumann [11]

May and Woods (1979) first identified alpha particles from trace uranium and thorium impurities in ceramic packaging as a significant source of soft errors in dynamic RAM (DRAM) devices [30]. Their pioneering work demonstrated that alpha particle emissions could generate sufficient charge to flip bits in memory cells. Subsequently, further research demonstrated that cosmic rays, particularly the secondary neutron flux produced when primary cosmic rays interact with atmospheric nuclei, constitute the dominant soft error mechanism at sea level and avionic altitudes [48]. It is also demonstrated that the atmospheric neutron flux varies with altitude, solar activity, and geographic location, and elevation. Neutron-induced soft error rates increase with elevation, reaching a peak at approximately 60,000 feet, beyond which the flux declines as the atmosphere becomes too thin to sustain the secondary particle cascades produced by primary cosmic ray interactions [31].

The charge collection mechanism that leads to soft errors can be understood through the behavior of ionized charge in semiconductor p-n junctions. When an energetic particle strikes near a reverse-biased junction, such as those in SRAM cells or flip-flops, the generated electron-hole pairs are separated by the electric field. This charge collection creates a transient current pulse that can temporarily alter the voltage at the sensitive node. If the collected charge ($Q_{collected}$) exceeds a critical threshold (Q_{crit}), the circuit node may experience a transition to an incorrect logic state, resulting in a bit flip [17]. Q_{crit} depends on the capacitance at the sensitive node, the restoring current capability of the circuit, and the node voltage swing [17].

2.1.3 Single Event Upsets

Single Event Upsets (SEUs) represent the most prevalent manifestation of soft errors, characterized by the corruption of a single bit in a memory element or latch. The term "single event" emphasizes that each particle strike constitutes an independent event capable of causing one single upset. The sensitivity of a device to SEUs is quantified by the soft error rate (SER), typically measured in Failures in Time (FIT), where 1 FIT equals one failure per 10^9 device-hours of operation [32].

The probability of an SEU occurring in a given device depends on the incident particle flux, the sensitive volume of the memory cell, and the critical charge Q_{crit} required to flip the stored bit. The relationship between technology scaling and SEU susceptibility is not straightforward. While shrinking feature sizes reduce the charge storage capacity of individual nodes, the introduction of three-dimensional transistor architectures such as FinFETs has partially reversed the trend by significantly increasing Q_{crit} relative to planar transistors of comparable geometry, owing to their improved electrostatic control and reduced charge collection volume [17]. As a result, modern nanometer-scale devices do not exhibit a simple monotonic increase in SEU sensitivity with technology generation. The actual soft error rate of a given device depends on a complex interplay of process technology, cell geometry, supply voltage, and operational environment, and must be characterized empirically rather than inferred from feature size alone [11].

2.1.4 SEU Effects on SRAM-Based FPGAs

FPGAs face unique and particularly severe soft error challenges due to their fundamental reliance on SRAM-based configuration memory to define circuit functionality. Unlike application-specific integrated circuits (ASICs) or microprocessors where soft errors primarily affect data storage elements and sequential logic states, SRAM-based FPGAs are vulnerable to SEUs in multiple memory domains, each with fundamentally different consequences for system reliability. Understanding this distinction is fundamental to designing a targeted and efficient fault injection campaign.

2.1.4.1 Configuration Memory SEUs: Permanent Functional Modifications

The most critical vulnerability in SRAM-based FPGAs arises from SEUs occurring in configuration memory. Configuration memory, implemented using SRAM cells distributed throughout the FPGA fabric, stores the complete definition of the implemented circuit, including lookup table (LUT) contents that define combinational logic functions, routing multiplexer settings that establish interconnections between logic elements, block RAM initialization values and operating modes, I/O buffer configurations and electrical characteristics, clock distribution network settings, and DSP slice operational parameters [48]. Unlike user data or logic state, configuration memory defines the structure and behavior of the RTL design itself. Consequently, an SEU in configuration memory does not simply corrupt a data value, it fundamentally alters the implemented circuit.

Carmichael et al. (2000) conducted seminal research on SEU effects in Xilinx Virtex FPGAs using heavy-ion radiation testing, establishing the critical distinction between configuration memory upsets and user logic upsets [31]. When a bit flip occurs in configuration memory, the effect persists indefinitely until the corrupted bit is detected and corrected through reconfiguration or scrubbing[5]. This persistence distinguishes configuration memory SEUs from all other soft error types in digital systems. For example, if an SEU corrupts a LUT configuration bit, the affected LUT will produce incorrect outputs for specific input combinations until the configuration is restored. The corrupted LUT continues to implement the wrong logic function across millions or billions of clock cycles, potentially causing systematic errors in computation results, control decisions, or data processing [31].

The permanent nature of configuration memory SEUs creates cascading failure scenarios not possible with transient faults. A single bit flip in routing configuration can break critical signal paths, causing functional blocks to become disconnected or incorrectly connected to wrong nets. An upset in clock configuration bits can alter clock frequency, phase, or distribution, inducing timing violations throughout the design. Configuration bits controlling

[5] Scrubbing: a periodic background process that reads back the configuration memory, detects any bit flips using error-correcting codes, and rewrites the corrected values.

I/O standards can change voltage levels or termination settings, causing communication failures or even physical damage to external components. These effects persist across the entire subsequent operation of the device, making configuration memory SEUs fundamentally more severe than transient data corruption [48].

The configuration memory in contemporary FPGAs contains tens to hundreds of millions of SRAM cells. For example, the Xilinx Virtex UltraScale+ VU9P device contains approximately 148 million configuration bits organized in configuration frames [17]. Each bit represents a potential single-point-of-failure for the implemented design if that bit controls an essential circuit element. The vast configuration memory space, combined with the permanent nature of upsets, makes SRAM-based FPGAs particularly vulnerable in radiation environments.

2.1.4.2 User Logic SEUs: Transient Effects

In contrast to configuration memory upsets, SEUs occurring in user logic elements, flip-flops, latches, FSM registers implemented in the RTL design, produce transient effects that persist only until the affected storage element is overwritten with new data. When a particle strike flips a bit in a user flip-flop, the corrupted value may propagate through combinational logic and affect subsequent computations or decisions. However, on the next clock cycle when that flip-flop is updated with a new value based on its input logic, the error is naturally corrected without requiring external intervention [32].

The transient nature of user logic SEUs provides inherent temporal masking opportunities. If the corrupted value in a flip-flop is overwritten before it can influence observable outputs or critical system state, the error has no lasting effect. For example, in a processor register file, an upset in a temporary register may be masked if that register is written with a new value before its corrupted content is read. Similarly, pipeline registers in signal processing datapaths naturally flush corrupted values as new data flows through the pipeline [32].

Wirthlin et al. (2005) analyzed the probability of user logic SEU propagation to observable failures, demonstrating that many flip-flop upsets are masked by logical, temporal, or electrical mechanisms before they can cause functional errors [32]. Logical masking occurs when the corrupted value does not affect the logic function being computed (e.g., a corrupted

bit in an unused register field). Temporal masking arises when the corrupted value is overwritten before being sampled or used. These natural masking phenomena substantially reduce the effective failure rate from user logic SEUs compared to the raw flip-flop upset rate.

2.1.4.3 Block RAM and Distributed RAM SEUs: Data Corruption

SRAM-based FPGAs also contain user-accessible memory elements including block RAMs (BRAMs) and distributed RAMs implemented using LUT resources. SEUs in these memory elements corrupt stored data values but, like user flip-flops, produce transient rather than permanent effects. A bit flip in BRAM corrupts the data stored at a specific address until that location is overwritten with new data or the memory is re-initialized [13].

The criticality of BRAM SEUs depends entirely on how the corrupted data is used by the application. If the upset occurs in unused memory locations or in data that is overwritten before being read, the error may have no observable effect. Conversely, if the corrupted data represents critical program instructions, configuration parameters, or algorithmic constants, the resulting functional failure can be severe. However, the key distinction from configuration memory remains: BRAM upsets affect data content, not circuit structure, and can be corrected by writing correct data to the affected address.

Asadi and Tahoori (2005) analyzed soft error rates in various FPGA resources, showing that BRAMs exhibit higher upset cross-sections than configuration memory due to their larger cell sizes optimized for density rather than radiation hardness [13]. However, they noted that effective BRAM error rates are often lower than configuration memory error rates because many applications use only a fraction of available BRAM capacity, leaving large portions of BRAM unpopulated and thus not vulnerable to functional failures when upset.

2.1.4.4 Configuration Memory vs. User Logic: The Permanence Distinction

The fundamental difference between configuration memory SEUs and all other FPGA soft errors lies in permanence versus transience. Table 2.1 summarizes this critical distinction:

Table 2.1 Comparison of SEU Effects in Different FPGA Memory Domains

Memory Domain	**Effect of SEU**	**Duration**	**Recovery Mechanism**	**Impact on RTL Design**
Configuration Memory	Circuit modification	Permanent until corrected	Requires reconfiguration/scrubbing	Changes implemented logic and routing
User Flip-Flops	State corruption	Transient (1 clock cycle typical)	Natural overwrite on next update	Affects data values, not circuit structure
Block RAM	Data corruption	Transient (until overwritten)	Write correct data to affected address	Affects stored data, not circuit function
Distributed RAM	Data corruption	Transient (until overwritten)	Write correct data to affected address	Affects stored data, not circuit function

The permanence of configuration memory upsets has direct implications for reliability engineering. Without active monitoring, a single bit flip in the configuration memory can cause indefinite functional failure, since the corrupted bit persists until the device is explicitly reconfigured. This stands in contrast to upsets in user logic elements such as flip-flops or block RAM, where new data naturally overwrites the corrupted state on subsequent clock cycles, making the fault transient in nature. Configuration memory in SRAM-based FPGAs is organized into frames, which represent the smallest addressable unit for readback and write-back operations. This organization is common across Xilinx FPGA families. Each frame spans a column of the device and contains bits that collectively define the behavior of multiple logic resources in that column, including LUT functions, routing multiplexer settings, and control signals. When an SEU occurs in the configuration memory, it alters the function or routing of the affected resource. Depending on which bit is upset, the resulting failure mode can range from a silent output error to a complete behavioral change in the implemented design, and the relationship between the physical bit location and the

observable system-level effect is what makes systematic fault injection analysis necessary for thorough reliability characterization [3].

2.1.5 Xilinx Soft Error Mitigation (SEM) IP Core

The SEM IP core, developed by AMD (formerly Xilinx) and documented in PG187 [2], is an FPGA-embedded IP that interfaces directly with the internal configuration infrastructure of UltraScale and UltraScale+ devices through the FRAME_ECC primitive. The SEM IP supports several operating modes, including observation, where the IP passively monitors the configuration memory for upsets without taking corrective action; classification, where detected errors are categorized by type and location; correction, where single-bit upsets are automatically corrected through a scrubbing mechanism; and injection, where controlled bit flips are deliberately introduced into the configuration memory to emulate SEU effects. In this work, only the injection capability is used, as the objective is to systematically emulate SEUs in the configuration memory of the design under test and characterize the resulting failure behaviour, rather than to detect or correct errors at runtime.

Error Injection Capabilities – SEU Emulation

A powerful feature of the SEM IP is its integrated error injection functionality, which enables controlled fault injection for validation of error detection mechanisms and fault tolerance architectures. The error injection interface accepts commands specifying the target frame address and bit position, allowing precise injection of upsets into arbitrary configuration bits. This capability supports comprehensive pre-deployment testing without requiring radiation facilities or external fault injection equipment [2].

According to PG187, error injection operates by temporarily modifying the SEM IP's internal golden reference for a specified frame, causing the scrubber to detect a non-existent error during subsequent frame reads [2]. Alternatively, the injection mechanism can directly corrupt the target configuration bit through a configuration write operation. Both approaches enable realistic fault injection that exercises the complete error detection and correction pathway, validating that the SEM IP and user error handling logic operate correctly.

XAPP1298 describes systematic fault injection methodologies using the SEM IP for design validation [14]. These methodologies involve iterating through all essential bits (namely potential critical bits, see Section 2.1.6), injecting a fault into each bit sequentially, monitoring application behavior, and classifying whether the injected fault causes a functional failure. This process enables empirical determination of true critical bits, the subset of essential bits whose upset produces observable errors, providing valuable data for reliability modeling and selective protection strategies.

The error injection capability also supports validation of Triple Modular Redundancy (TMR) for some FPGA devices and other fault tolerance implementations. By injecting faults into TMR voter logic or redundant computation paths, designers can verify that the redundancy scheme correctly masks single-point failures. XAPP1298 provides case studies demonstrating SEM IP-based validation of TMR processors, where systematic fault injection revealed design weaknesses including incomplete voter coverage and common-mode failure paths [14].

Performance Characteristics and Resource Utilization

The SEM IP's resource utilization and performance characteristics vary depending on device family, configuration options, and operating mode. PG187 provides detailed resource consumption data for UltraScale+ implementations [2]. A typical SEM IP instance configured for Correction mode with essential Bits support consumes approximately:

- 600-800 LUTs for control logic and interface circuitry
- 300-400 flip-flops for state machines and data buffering
- 2-4 block RAMs for golden configuration storage (varying with design size and essential bit count)
- 1 FRAME_ECC primitive for hardware ECC acceleration

The SEM IP operates at clock frequencies ranging from 50 MHz to 100 MHz on UltraScale+ devices, with higher frequencies enabling faster scrubbing and reduced MTTD. At 100 MHz operation, a complete classification pass of a large UltraScale+ device (e.g., VU9P with ~148 million configuration bits) requires approximately 40-60 milliseconds for full-device scrubbing, or 2-5 milliseconds when using essential bits mode with a typical design utilizing

5% of device resources [45]. The time needed for a full cycle of a single bit error injection, which is the mode that have been used in this work, is 130 milliseconds.

Successful integration of SEM IP into FPGA designs requires careful consideration of several factors. XAPP1298 provides comprehensive guidelines for SEM IP deployment into Zynq UltraScale and UltraScale+ devices which will be explained briefly in chapter 3 since this work is implemented on Zynq UltraScale+ ZCU102.

Limitations and Advanced Considerations

While the SEM IP provides robust SEU mitigation for configuration memory, several limitations must be recognized. As documented in both PG187 and XAPP1298, the IP cannot protect against multi-bit upsets within a single ECC word, which represent an increasing threat in high-flux radiation environments or advanced technology nodes where charge sharing can cause spatially-adjacent upsets [30, 48]. Additionally, the SEM IP does not protect user logic state, flip-flops, block RAMs, and distributed RAMs require separate mitigation techniques such as TMR or scrubbing. The same FPGA resources also require separate injection techniques as the SEM IP does not perform bit flips in those resources

The scrubbing process introduces a vulnerability window between upset occurrence and error correction. During this window, bounded by the scrub cycle time an upset bit remains in the corrupted state and may cause functional errors if accessed by user logic. For critical applications, XAPP1298 recommends combining SEM IP with design-level fault tolerance techniques like TMR to provide instantaneous masking while configuration scrubbing provides background correction [14].

Recent research has explored advanced scrubbing strategies that improve upon the SEM IP's fixed-sequence scanning. Adaptive scrubbing algorithms that dynamically adjust scan patterns based on observed error rates, thermal conditions, or application criticality can reduce average MTTD and improve protection efficiency. However, these techniques currently require custom logic external to the standard SEM IP core [12].

2.1.6 ICAP and PCAP Considerations for SEM IP on Zynq UltraScale+

The Internal Configuration Access Port (ICAP) is a dedicated primitive available in Xilinx FPGAs that provides access to the configuration memory from within the programmable logic itself. Unlike external configuration interfaces that are used during initial programming of the device, ICAP allows user logic or IP cores running on the PL to read and write the configuration memory at runtime. This makes it a key component for functions like partial reconfiguration, readback, and soft error mitigation. In the context of the SEM IP, ICAP is the interface through which the controller scans the configuration memory frames, detects bit flips, and performs corrections or injections. Without access to ICAP, the SEM IP cannot operate.

As noted in XAPP517 [9], since there is only one ICAP primitive per device, any design that needs to share it between the SEM controller and other user functions must carefully coordinate access, the SEM controller must first be moved to the IDLE state before the user logic takes over ICAP, and control must be returned to the SEM controller afterward without issuing a DESYNC command, otherwise the controller may fail to resume correctly [XAPP517].

One important aspect of integrating the SEM IP on Zynq UltraScale+ devices is managing the configuration port access between the PS and the PL. On these devices, when the system boots, the PS takes control of the configuration logic through the Processor Configuration Access Port (PCAP). This is the default path used by the PS bootloader to load the bitstream into the PL. After boot is complete, the PS and PCAP remain in control of the configuration logic, which means the ICAP, which is the internal configuration port on the PL side, is blocked and cannot be used. Since the SEM IP relies on the ICAP to scan and access the configuration memory, this creates a conflict that must be resolved before the SEM controller can function properly.

To solve this, configuration logic access must be explicitly transferred from PCAP to ICAP. As described in XAPP1298, this is done through the PS using the EMIO GPIO interface of the Zynq UltraScale+ block. Specifically, the PS drives the ICAP arbitration signals, cap_req, cap_gnt, and cap_rel, to grant the SEM IP access to the ICAP. Once the grant is given, the

SEM controller initializes and begins scanning the configuration memory. The PS then monitors the SEM IP status signals, such as status_correction and status_uncorrectable, to track the results of the scan and any detected errors. In our implementation, this handshake between the PS and the SEM IP is handled during the initialization phase through the second UART interface, which allows the host PC to trigger the ICAP grant sequence and confirm that the SEM controller has initialized successfully before the injection campaign begins.

2.1.7 Essential Bits and Critical Bits in FPGA Designs

The vast configuration memory space of modern FPGAs contains significant redundancy, with many bits controlling unused resources or providing functionally equivalent routing alternatives. Essential bits represent the subset of configuration memory that actively contributes to the implemented design's functionality. Xilinx formally defines essential bits as "configuration bits that, if altered, could change the functionality or behavior of the design" [1].

The essential bits for a specific design constitute a small fraction of total configuration memory in a way that typical circuits utilize only 2-8% of available configuration memory, with the remainder controlling unused logic blocks, routing resources, and I/O banks [45]. It is important to note that not all essential bits are critical bits: essential bits are those that affect the implemented design in any way, while critical bits are the subset whose upset produces an observable error at the design outputs. This distinction is significant for fault injection campaigns, since targeting the injection effort to the essential bit population rather than the full configuration memory substantially reduces the number of injections required for comprehensive coverage, making systematic reliability characterization practical within realistic time constraints.

Xilinx provides automated essential bits identification through the Vivado design tools, which generate Essential Bits Database (EBD) files during bitstream generation. The EBD file specifies exactly which configuration frame address-bit pairs are essential for the design, enabling targeted protection strategies [4]. The generation process analyzes the placed and

routed design, identifying all configuration bits associated with used logic elements, active routing interconnects, block RAM contents, and I/O configurations.

The Essential Bits Database (EBD) file, which forms the foundation for targeted fault injection campaigns, can be generated during the Vivado bitstream creation process through two primary methods. The first method involves setting a design property through a Tcl command in the Vivado command console or script:

set_property BITSTREAM.SEU.ESSENTIALBITS YES [current_design].

This command must be executed before the write_bitstream command in the Vivado design flow, typically after place and route completion [4]. Alternatively, and more commonly for automated and reproducible design flows, the essential bits generation can be enabled by adding a constraint to the design's XDC (Xilinx Design Constraints) file. The EBD generation is permanently configured for the project and will automatically execute during every bitstream generation without requiring manual intervention by including the following line in the constraints file[4]:

```
set_property BITSTREAM.SEU.ESSENTIALBITS YES [current_design]
```

This constraint file approach is particularly advantageous for version-controlled projects and collaborative development environments, as it ensures consistent EBD generation across different users and build sessions.

When either method is enabled, Vivado analyzes the fully placed and routed design during bitstream generation, comparing the design-specific configuration against the device's default configuration state to identify all modified bits. The tool then generates an .ebd file alongside the standard .bit bitstream file, containing a comprehensive list of frame addresses and bit positions for all essential configuration bits. This EBD file uses a compact ASCII format where each line specifies a configuration frame address followed by the bit positions within that frame that are essential to the design. For UltraScale and UltraScale+ devices, the EBD file typically represents only 2-8% of the total device configuration memory, significantly reducing the fault injection search space from hundreds of millions of bits to a manageable subset of several hundred thousand to a few million essential bits [1]. The automated

extraction of this database eliminates the need for manual essential bits identification and ensures that fault injection campaigns focus exclusively on configuration regions that can actually impact design functionality.

2.1.7.1. Parsing and Utilizing the Essential Bits Database

The Essential Bits Database (EBD) file generated by Vivado employs a structured ASCII format that requires specific parsing procedures to extract usable fault injection targets. According to AMD's documentation, the EBD file consists of multiple lines, each beginning with a keyword that identifies the type of information contained in that line [5]. The primary keywords include Bit, which specifies individual essential bit locations, and Block, which defines blocks of essential bits within configuration frames. Header lines provide metadata about the design and device, including the part name, design name, and EBD file format version. Comment lines, denoted by a hash symbol (#), provide human-readable descriptions and are ignored during automated parsing. The critical information for fault injection resides in the Bit lines, which follow the format: Bit <frame_address> <word_offset> <bit_offset>, where the frame address uniquely identifies a configuration frame within the device's configuration memory space, the word offset specifies the 32-bit word within that frame (ranging from 0 to 92 for UltraScale+ frames containing 93 words), and the bit offset indicates the specific bit position within that word (0 to 31) [5].

To utilize the EBD file for systematic fault injection, automated parsing scripts must extract and interpret these bit specifications to generate fault injection commands compatible with the SEM IP core. The SEM IP error injection interface requires faults to be specified using frame addresses and bit positions within frames, necessitating conversion from the EBD file's word-and-bit format to the SEM IP's frame-and-bit format. According to the AMD documentation, this conversion involves calculating the absolute bit position within a frame using the following formula [5]:

$$\text{Absolute Bit Position} = (\text{Word Offset} \times 32) + \text{Bit Offset} \tag{2.1}$$

where the Word Offset specifies the 32-bit word within the frame (ranging from 0 to 92 for UltraScale+ frames containing 93 words), and the Bit Offset indicates the specific bit position within that word (0 to 31).

For example, an EBD entry specifying Bit 0x00020100 45 7 indicates that the essential bit is located in configuration frame at address 0x00020100, at word offset 45, bit offset 7. Using the conversion formula:

$$\text{Absolute Bit Position} = (45 \times 32) + 7 = 1440 + 7 = 1447 \tag{2.2}$$

This corresponds to bit position 1447 within that frame [5].

A Python-based automation scripts has been developed in this work to systematically parse the entire EBD file, extracting all Bit entries, performing the necessary address conversions, and constructing a comprehensive database of injectable fault targets. This parsed database then serves as the input to fault injection manager, which sequences through each essential bit location, issuing appropriate SEM IP error injection commands via the UART interface. The structured nature of the EBD format ensures that automated parsing is robust and reliable, enabling comprehensive fault coverage across all design-essential configuration bits without manual intervention or risk of human error in target specification.

2.1.7.2 Critical Bits and Fault Analysis

While essential bits represent the configuration memory actively used by a design, not all essential bits equally threaten application functionality when upset. As shown in figure 2.2, critical bits constitute the subset of essential bits whose corruption causes observable application-level malfunctions, incorrect outputs, system failures, or violation of functional requirements. The distinction between essential and critical bits is not merely academic, it has profound implications for reliability assessment, mitigation strategy design, and understanding real-world failure risks in deployed systems.

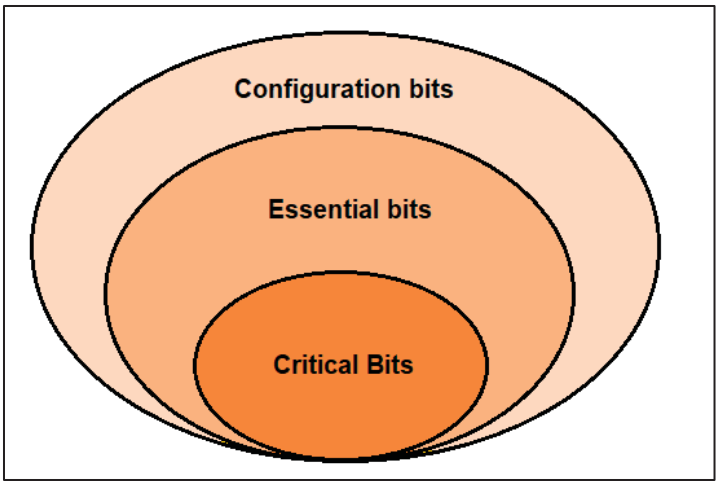


Figure 2. 2 Critical bits are a subset of essential bits

Application Malfunction vs. Bit Essentiality

An essential bit, by definition, is any configuration bit that differs from the device's default state for a given design implementation. This includes all bits controlling used logic resources, active routing paths, block RAM contents, and I/O configurations [1]. However, essentiality does not imply criticality. A configuration bit may be essential to the design's implementation yet non-critical to its functional correctness. For example, consider a dual-port block RAM where only one read port is actively used by the application. Configuration bits controlling the unused second port are essential (they differ from default values) but non-critical (their corruption cannot cause functional failures since the port is never accessed).

The critical distinction emerges when examining application-level behavior under fault injection. Foucard et al. (2010) conducted comprehensive fault injection campaigns on FPGA-implemented processors, injecting upsets into all essential bits and monitoring application execution [19]. Their results revealed that approximately 40-60% of essential bit upsets caused no observable malfunctions during benchmark execution. Applications continued to produce correct outputs, met timing requirements, and completed execution successfully despite configuration corruption. These non-critical essential bits represent configuration overhead, routing redundancy, unused LUT inputs, clock distribution to inactive logic, or other implementation artifacts that do not affect functional behavior [19].

Conversely, critical bits, though representing only a fraction of essential bits, directly threaten application correctness. When critical bits are upset, applications exhibit various malfunction modes: incorrect computational results that violate algorithmic specifications, control flow errors causing programs to execute wrong code paths or enter infinite loops, communication failures where data interfaces produce corrupted or missing transmissions, timing violations where outputs are generated late or not at all, and catastrophic crashes requiring system reset or reconfiguration. Understanding which essential bits are critical enables targeted protection strategies that maximize reliability while minimizing resource overhead.

2.1.7.3 Conjunctive Critical Bits

A related phenomenon observed in fault injection campaigns is that of conjunctive critical bits, configuration memory bits where a single bit flip alone is not sufficient to cause an observable functional failure, and the failure only manifests when two or more specific bits are simultaneously upset. This behaviour arises from the way Vivado maps logic onto FPGA LUT resources. Each 6-input LUT on UltraScale+ devices implements a Boolean function defined by 64 bits stored in its INIT parameter in the configuration memory. When a design does not use all six inputs of a LUT, which is common after synthesis, the unused entries in the INIT table are don't-care positions. Vivado fills those positions during place and route according to its own optimization heuristics, meaning that a single bit flip in a don't-care entry changes the LUT's truth table but the affected entry is never exercised by the actual input stimulus, so the output remains correct [19]. The bit appears non-critical under single-bit injection, even though it is marked as essential in the EBD file. A second flip in a related LUT or in the same LUT's exercised entries can then remove that masking and expose the error. Cherezova et al. observed this phenomenon systematically, noting that SEUs can generate additional components of logical functions or reduce them without affecting system outputs, and that such modifications account for a measurable fraction of all LUT fault injections [15]. This natural LUT masking effect means that single-SEU injection campaigns will miss conjunctive critical bits entirely, since they never flip two bits simultaneously. In

practice, conjunctive CBs are discovered incidentally when a sequential single-bit campaign happens to flip the second bit of a pair before the SEM IP scrubber corrects the first. Their proportion relative to the total critical bit population is therefore likely underestimated by any single-bit campaign.

2.1.8 Deep Learning and Neural Network Inference Fault Tolerance

Deep learning models implemented on FPGAs demonstrate remarkable resilience to certain types of configuration upsets due to the inherent fault tolerance of neural network computations. Reagen et al. (2016) conducted extensive fault injection studies on convolutional neural networks (CNNs) deployed on FPGAs, revealing that many weight quantization and activation function errors have minimal impact on classification accuracy [37]. Neural networks trained with dropout or other regularization techniques exhibit natural redundancy, no single weight or neuron is critical to overall network function. When configuration bits controlling weight storage or multiply-accumulate operations are upset, the resulting computational errors often fall within the noise tolerance envelope of the trained model.

However, this fault tolerance is highly non-uniform across network components. Li et al. (2017) demonstrated that configuration upsets affecting early convolutional layers cause larger accuracy degradation than upsets in later fully connected layers, because early layers extract low-level features critical to all subsequent processing [26]. Similarly, batch normalization parameters and activation function implementations are highly critical, their corruption can cause catastrophic accuracy collapse even when weight corruption is tolerated. Configuration bits controlling data format (floating-point vs. fixed-point representation), quantization thresholds, and numerical precision are universally critical because their upset fundamentally changes computational semantics.

The practical implication is that deep learning FPGA accelerators require selective criticality-aware protection. Protecting all essential bits imposes unnecessary overhead, while protecting only the most critical bits, early layer parameters, normalization factors, and datapath precision controls, can achieve >99% fault coverage with <20% of full protection

cost [26]. Understanding which configuration bits affect neural network inference accuracy versus those that cause complete computational failure is essential for designing efficient fault mitigation strategies in AI accelerator systems. To explore workload-dependent criticality and the fault tolerance characteristics of different application domains, this work includes fault injection testing of the RISC-V processor executing deep learning inference workloads. Chapter 4 shows the result on this work.

2.2 Literature Review

Overview

The assessment of FPGA reliability in radiation environments urges the need of study in two critical aspects: the physical beam testing or emulation strategies for SEU/fault injection, and developing effective methodologies to evaluate design vulnerability through fault injection. This dual challenge has driven extensive research into both radiation hardening techniques and automated testing frameworks. The Soft Error Mitigation (SEM) IP core from Xilinx represents a significant advancement in integrated SEU protection, while parallel developments in automated fault injection have enabled comprehensive reliability assessment of complex FPGA-based systems. This section reviews the alternative methods in both SEU fault injection and automated fault injection, positioning the contributions of this work with gaining Xilinx SEM IP and python-based automation, and compares the key frameworks in Table 2.2.

2.2.1 ICAP-Based Fault Injection Platforms

The Internal Configuration Access Port (ICAP) is the most widely used interface for automated fault injection on Xilinx FPGAs. Because ICAP operates within the FPGA fabric and can be clocked at high frequencies, it provides injection latencies on the order of microseconds per fault, several orders of magnitude faster than external JTAG interfaces, making it suitable for comprehensive campaigns covering large essential bit populations.

Zhang et al. [47] proposed a pipelined ICAP platform for Xilinx Virtex-6 FPGAs that overlaps the readback of the next frame with the writeback of the current one, increasing throughput significantly over sequential approaches. Their platform also supports accumulated multiple-SEU injection and automatic bitstream restoration between campaign cycles, two capabilities essential for reproducible testing. Ferlini et al. [18] published a complementary methodology in 2023 that embeds the ICAP injection controller as an on-chip LEON3 soft-core, eliminating round-trip host communication latency and reducing a 20-hour simulation campaign to 20 minutes of hardware emulation. Liu et al. [28] extended ICAP injection to the user SRAM domain in 2025, proposing a dual-circuit model that inserts saboteur circuits to emulate SEUs in flip-flops and block RAM at random operational moments, capturing the temporal sensitivity of the design rather than only its static configuration memory footprint. DA-FIS [16], also from 2025, introduced a configurable LFSR-based fault generator for adaptive real-time injection, enabling the platform to adjust injection targets dynamically based on observed system behaviour during a running campaign.

2.2.2 JTAG-Based and External Interface Approaches

External JTAG access to the configuration memory requires no fabric resources and is non-intrusive, making it applicable to any Xilinx FPGA without design modifications. The trade-off is throughput: frame-level JTAG injection typically incurs latencies of 300 to 720 milliseconds per fault [20], making exhaustive campaigns across large essential bit populations impractical. FREtZ [20], an open-source JTAG-based framework from the University of Piraeus, addresses this by exposing a structured TCP client-server API that supports concurrent fault injection and DUT monitoring, as well as injection into both configuration memory and user registers through the same interface. Smit et al. [41] extended FREtZ in 2024 to support temporally targeted injection at specific clock cycles or program counter locations within a soft-core processor's execution, reducing the effective injection space to the code regions of interest and enabling more thorough coverage with fewer injections.

2.2.3 Dynamic Partial Reconfiguration-Based Fault Injection

Dynamic Partial Reconfiguration (DPR) enables localized injection into isolated FPGA design regions while the rest of the device continues running, and allows multiple DUT variants to be swapped in and out of reconfigurable regions without modifying the static injection infrastructure. Ben Jrad et al. [21] combined DPR via ICAP with the Xilinx Isolation Design Flow to perform spatially constrained injection and applied fault distribution laws derived from ground-based radiation experiments, improving the physical accuracy of the emulated fault patterns. Wilson et al. [44] at Brigham Young University demonstrated the scalability of DPR-based injection in 2023 by deploying 20 parallel Automatic Test Equipment (ATE) instances on Xilinx 7-series FPGAs, collectively performing over 15 million injections across 10 DUT configurations, including five open-source RISC-V processors in both unmitigated and TMR-protected variants, in 10 days. Their results showed TMR sensitivity reductions of 20 to 500 times depending on the processor and TMR granularity. While DPR provides the highest throughput and the most versatile multi-DUT testing capability, it requires non-trivial floorplan preparation and is best suited to comparative studies rather than single-design reliability characterization.

2.2.4 HDL Simulation and Bit-Accurate Approaches

HDL-level fault injection targets the structural VHDL or Verilog description of a design before synthesis, offering early-stage vulnerability screening without requiring hardware. Tuzov et al. [43] presented BAFFI at DATE 2023, an open-source bit-accurate fault injector within the DAVOS toolkit that maps essential bits to individual netlist elements, down to individual LUTs or registers, without requiring manual Pblock definitions. On UltraScale+ devices BAFFI uses PCAP for low-latency on-chip injection, making it one of the few tools to explicitly support this platform. Cherezova et al. [15] published an LUT-based SEU model in 2025 that modifies INIT parameters in structural VHDL, achieving 94.76 percent fault coverage on Versal FPGA implementations of processor cores and AI inference workloads. Rajkumar and Åberg [36] addressed the challenge of critical bit identification at DATE 2024,

proposing a guided injection strategy that converges on the critical bit population more efficiently than exhaustive random injection, without requiring proprietary bitstream reverse engineering.

2.2.5 The ACME/ACME-2 Tools and SEM IP-Based Frameworks

A major enabler for SEM IP-based injection campaigns is efficient essential bit targeting. Aranda et al. [6] developed the ACME (Automatic Configuration Memory Error-injection) tool in 2019, which translates a user-defined Pblock region containing the DUT into SEM IP injection addresses derived from the Vivado-generated Essential Bits Database (EBD) file, reducing campaign runtime by up to 32.5 times compared to full-device injection. ACME-2 [8] refined this in 2022 by incorporating FPGA architectural information to filter spurious boundary bits introduced by the linear correlation model, improving measurement precision. Ruano et al. [39] provided a comprehensive end-to-end tutorial in 2021 demonstrating the full SEM IP injection workflow with ACME integration and MATLAB automation, covering a finite impulse response filter as the DUT.

Several studies have applied the SEM IP specifically as a fault injection engine. Oliveira et al. [33] used it to characterize the Rocket Chip RISC-V processor in 2020, finding it more sensitive than LEON3 due to its more complex microarchitecture. Siecha et al. [40] applied SEM IP-based emulation in 2025 to a Time-to-Digital Converter on a ZedBoard, injecting faults into the essential bit population and comparing DNL and INL metrics between the faulty and golden designs. Aranda et al. [7] used the SEM IP with ACME in 2020 to analyze critical bits of a RISC-V processor for space applications and showed that the critical bit population is workload-dependent, different programs result in different critical bit fractions, a finding directly relevant to the two-workload RISC-V campaigns in this thesis.

2.2.6 Processor-Based System Fault Injection Studies

Several researchers have applied fault injection techniques specifically to processor-based systems implemented in FPGAs, providing critical insights into the vulnerability patterns and

critical bit distributions of different processor architectures. Understanding these processor-specific reliability characteristics is particularly relevant to this work, as the automated SEM IP-based fault injection framework is applied to a RISC-V32I processor implementation to validate the methodology.

Höller et al. (2015) conducted extensive fault injection campaigns on embedded processors using simulation-based injection, characterizing failure modes across various benchmark applications [22]. They classified upsets into categories including crashes, hangs, wrong outputs, and silent data corruption, providing statistical distributions of failure types across different processor workloads.

Azambuja et al. (2013) investigated SEU effects on ARM Cortex-A9 processors in Xilinx Zynq devices using partial reconfiguration-based fault injection [10]. Their methodology targeted specific processor components including the register file, instruction cache, and data cache, revealing that different processor structures exhibit vastly different critical bit fractions. The study demonstrated that only 5 to 15 percent of essential bits in processor implementations cause functional failures when upset, highlighting the importance of critical bit identification for efficient fault injection campaign targeting.

Papadimitriou and Gizopoulos (2023) provided a foundational microarchitectural analysis of silent data corruptions (SDCs) in modern out-of-order processors, published in IEEE Transactions on Computers [34]. Their work systematically characterized which microarchitectural structures, including the reorder buffer, register file, load/store queues, and branch predictor, are most likely to generate SDCs at the program output rather than detectable crashes or hangs. A key finding is that the magnitude of SDCs varies significantly across hardware structures and is not uniformly distributed, meaning that some processor components are disproportionately responsible for silent, undetectable corruption. This structural SDC characterization is directly relevant to FPGA-based processor fault injection studies: when a configuration memory upset alters the function of a logic resource implementing one of these high-SDC structures, the resulting failure is more likely to produce a corrupted output that passes undetected by a simple pass/fail monitor. This reinforces the importance of output validation beyond crash detection, as implemented in the UART monitoring channel of the framework presented in this thesis.

2.2.7 Progressive Bit Search (PBS) technique Fault Injection

Recent research has extended fault injection methodologies to address the unique challenges of evaluating single event effects in deep neural networks deployed in radiation environments. Lyu et al. (2025) developed a specialized framework for systematic vulnerability assessment of DNNs, introducing a Progressive Bit Search (PBS) technique to efficiently identify sensitive layers within neural network architectures [29]. Unlike exhaustive fault injection that requires testing all essential bits, PBS employs a gradient-based hierarchical search strategy that progressively narrows the search space by identifying configuration regions with the highest impact on network inference accuracy.

The PBS methodology operates in two stages: intra-layer search and cross-layer matching. In the intra-layer search phase, the algorithm computes gradient-based sensitivity metrics for individual weight parameters, sorting them by absolute gradient values to identify the most vulnerable bits. The cross-layer matching phase evaluates the cumulative effect across all network layers, selecting the layer with maximum loss contribution for targeted bit flipping. Implemented within the PyTorch framework on YOLOv5 object detection networks, their approach demonstrated over 10× improvement in fault injection evaluation efficiency compared to random fault injection methods. The PBS technique could render the neural network ineffective after modifying just one parameter in stable conditions, whereas random bit flipping required significantly more iterations to achieve comparable accuracy degradation [29].

Lyu et al. combined PBS with selective Triple Modular Redundancy applied only to identified sensitive layers, achieving a 24× improvement in single event soft error resilience while maintaining resource overhead below 10%. This selective hardening approach contrasts with full-design TMR that imposes 200-250% resource costs, demonstrating that application-aware fault injection can inform highly efficient mitigation strategies. However, the PBS technique's effectiveness depends critically on selecting appropriate gradient metrics and may be sensitive to the specific input dataset used during the progressive search phase. Additionally, the approach is inherently application-specific, optimized for DNN workloads,

and requires domain expertise to implement the gradient-based search heuristics, limiting its generalizability to arbitrary FPGA designs [29].

The software-based fault injection in PyTorch, while enabling rapid iteration and efficiency improvements, operates at a higher abstraction level than hardware-accurate FPGA fault injection. Software simulation cannot capture configuration memory SEU effects, timing-dependent failures, or routing-specific vulnerabilities that manifest only in physical implementations. This represents a fundamental difference from hardware-based approaches like the SEM IP methodology, which directly injects faults into actual FPGA configuration memory to characterize real device behavior under SEU conditions.

2.2.8 Radiation Beam Testing

Physical radiation testing using particle accelerators provides the gold standard for SEU characterization, offering direct measurement of upset rates under realistic radiation conditions. Quinn et al. (2015) conducted comprehensive neutron beam testing of Xilinx Zynq-7000 devices at the Los Alamos Neutron Science Center (LANSCE), measuring configuration memory cross-sections and characterizing failure modes in processor and programmable logic [35]. Their methodology involved exposing FPGAs to atmospheric-like neutron spectra while monitoring system behavior through custom test applications.

While radiation testing provides the most accurate characterization of SEU effects, it suffers from significant practical limitations. Accelerator facilities have limited availability, requiring months of advance scheduling and costing thousands of dollars per beam day. The statistical nature of radiation testing necessitates long exposure times to accumulate sufficient upset events for reliable analysis, particularly for rare failure modes or designs with low critical bit fractions. Additionally, beam testing provides limited controllability, researchers cannot selectively target specific configuration bits or replay identical fault scenarios for detailed debugging. Moreover, the costs and scheduling challenges make iterative testing of multiple design variants impractical for most projects.

2.3 Comparison of Fault Injection Methodologies

Table 2.2 compares the key frameworks reviewed in this chapter across the dimensions most relevant to the methodology selection for this thesis.

The comparison reveals that each existing methodology carries trade-offs that limit its practical applicability for systematic, reproducible, and low-cost reliability characterization of FPGA designs.

Physical radiation beam testing, while the most physically accurate method, is prohibitively expensive and logistically constrained. Accelerator facilities require months of advance scheduling and charge thousands of dollars per beam day, making iterative testing across multiple design variants or workloads infeasible for most projects. Radiation testing also offers no controllability over which specific bits are upset, making it impossible to replay identical fault scenarios for debugging or to isolate the effect of individual configuration regions. For these reasons, beam testing is reserved for final validation of mature designs rather than for the iterative reliability characterization needed during development.

ICAP-based platforms achieve the lowest injection latencies, but require the injection controller to be instantiated inside the FPGA fabric, consuming logic resources and potentially affecting the placement of the design under test. DPR-based platforms are the most scalable for multi-DUT comparative studies but impose significant design overhead: the designer must define Pblock regions and isolation constraints before implementation, which changes the design flow, affects routing, and requires expertise in partial reconfiguration that is not part of a standard FPGA design practice. Both ICAP and DPR approaches therefore require design preparation steps that couple the injection infrastructure to the DUT, reducing their generality.

JTAG-based approaches avoid all fabric resource consumption and require no design modifications, but their injection latency of 300 to 720 milliseconds per fault is several orders of magnitude slower than internal interfaces. For a design with tens of thousands of essential bits, a comprehensive JTAG-based campaign would take weeks to complete, making it impractical for systematic reliability assessment.

Table 2.2 Comparison of SEU Fault Injection Methodologies for SRAM-Based FPGAs

Methodology	Injection Latency	Automation Scope	Scalability	Resource Overhead	Integration Complexity	Cost
Radiation Beam Testing [Quinn et al., 33]	N/A (uncontrolled)	None — no software control over which bits are upset; no campaign automation	Not scalable; limited beam time; months of scheduling per test	None on FPGA; requires accelerator facility	Very High — requires access to particle accelerator; specialized test setup	Very High ($K/day)
ICAP-Based Injection [Zhang et al., 1] [Ferlini et al., 2] [DA-FIS, 4]	~10–100 µs/fault	High — automated frame-level injection; automatic bitstream restore; campaign scripting	High — microsecond latency enables millions of injections; multi-board scaling possible	Moderate — on-chip ICAP controller consumes fabric LUTs and routing resources	Moderate — requires on-chip controller instantiation; may affect DUT placement	Low
JTAG-Based Injection [FREtZ, 5] [Smit et al., 6]	~300–720 ms/fault	Moderate — non-intrusive; supports concurrent DUT monitoring via TCP client-server	Poor — 300–720 ms/fault makes exhaustive campaigns across large bit populations impractical	None — no fabric resources consumed; requires only JTAG cable	Low — no design changes; standard JTAG interface; easy to deploy on any board	Low
DPR-Based Injection [Ben Jrad et al., 7] [Wilson et al., 8]	~10–100 µs/fault	Very High — multi-DUT ATE platform; 15M+ injections across 10 DUTs demonstrated	Very High — multiple partial regions tested in parallel; highly scalable for comparative studies	High — requires dedicated DPR floorplan partitions; isolation design flow overhead	Very High — Pblock constraints, isolation design flow, DPR-compatible floorplanning required	Moderate (multiple boards)
HDL Simulation [Tuzov et al. (BAFFI), 9] [Cherezova et al., 10]	N/A (simulation time varies)	Very High — bit-accurate; targets individual LUTs/FFs; automated campaign scripting	Moderate — limited by simulation speed; not representative of physical layout effects	None on hardware — runs entirely in simulation environment	Moderate — pre-synthesis only; does not reflect post-P&R critical bit distribution	Very Low (no hardware)
Guided Injection (PBS) [Rajkumar & Åberg, 11] [Lyu et al., 31]	~µs/fault (hardware) or N/A (software)	High — gradient-based search narrows injection space; no exhaustive bit sweeping needed	High — 10× efficiency improvement over random injection demonstrated on DNNs	Low to Moderate — depends on implementation (software or hardware-assisted)	Moderate — requires application-specific gradient metrics; domain expertise needed	Low to Very Low

Table 2.2 Continued Comparison of SEU Fault Injection Methodologies for SRAM-Based FPGAs

Methodology	Injection Latency	Automation Scope	Scalability	Resource Overhead	Integration Complexity	Cost
SEM IP-Based Injection [Aranda et al., 12,13,17] [Ruano et al., 14] [Oliveira et al., 15] [Siecha et al., 16]	~100–500 ms/fault (UART-limited)	Moderate — ACME/MATLAB scripting available; but no closed-loop DUT monitoring; manual campaign management	Moderate — UART bandwidth limits throughput; limited to Zynq-7000 and 7-series devices	Low — no fabric resources consumed beyond the SEM IP itself; design-agnostic	Low — no RTL modifications; no floorplan constraints; standard SEM IP instantiation	Low
This Work (SEM IP, Python/UART, Zynq UltraScale+)	~130 ms/fault (UART-limited)	High — fully automated Python campaign loop; real-time closed-loop UART monitoring; auto-halt on failure detection	Moderate — UART bandwidth limits throughput; scalable to any SEM IP-compatible Xilinx design without modification	Low — no fabric resources beyond SEM IP; no on-chip controller; no floorplan constraints	Low — no RTL modification; no floorplan required; plug-and-play UART interface; first framework on Zynq UltraScale+	Low

HDL simulation and software-level methods such as the PBS technique offer very low cost and no hardware requirement, but they fundamentally cannot capture the post-implementation, placement- and routing-dependent effects that determine which physical configuration bits are actually critical for a specific FPGA implementation. A simulation-based result reflects the pre-synthesis model, not the actual device behavior. Similarly, the PBS technique operates entirely within a software deep learning framework and cannot model configuration memory SEUs, timing-dependent failures, or routing-specific vulnerabilities that only emerge in a physical implementation.

Among all reviewed approaches, SEM IP-based fault injection offers the most suitable combination of properties for design-agnostic, non-intrusive, hardware-accurate reliability assessment on Xilinx FPGAs. The SEM IP requires no fabric resources beyond its own instantiation, needs no modifications to the RTL under test, operates on the actual synthesized and placed-and-routed design capturing all physical layout effects, and accesses

the configuration memory through a vendor-supported dedicated interface. However, a thorough review of all published SEM IP-based fault injection studies confirms that none has used the SEM IP injection interface as the primary engine for an automated, closed-loop reliability characterization campaign on the Zynq UltraScale+ platform. Studies targeting the UltraScale+ family, such as Yang et al. and Pagonis et al., use DPR-based injection or employ the SEM IP only in its scrubbing and correction role. Studies using the SEM IP injection interface systematically, such as Siecha et al. [40] and Oliveira et al. [33], target older Zynq-7000 and 7-series devices. Furthermore, none of the reviewed SEM IP-based works implements a closed-loop automation in which the host monitors the DUT output in real time during each injection and halts the campaign automatically upon detecting a failure, instead relying on post-hoc log analysis.

2.4 Research Gap and Positioning of This Work

The reviewed frameworks confirm that automated SEU fault injection for SRAM-based FPGAs is well developed for ICAP, JTAG, DPR, and HDL simulation interfaces. However, the SEM IP injection interface on the Zynq UltraScale+ remains unaddressed by any published general-purpose automation framework. Existing SEM IP studies either focus on runtime error correction rather than injection, perform injection semi-manually for a specific single design, or operate on 7-series and Zynq-7000 devices not directly applicable to the UltraScale+ family.

This thesis addresses that gap by building an automation layer on top of the SEM IP's existing command interface. The framework's design-agnostic UART communication model distinguishes it from ICAP-based platforms that require design-specific control logic inside the FPGA. The framework is also the first to apply SEM IP-based automated injection to a fixed-point deep learning workload running on a RISC-V soft processor on this platform. Importantly, this work is an extension of prior fault injection research conducted at ÉTS. Souari et al. [42] presented an optimization of SEU emulation on SRAM FPGAs based on sensitiveness analysis, demonstrating that targeting the injection effort to sensitive

configuration regions substantially reduces campaign time without sacrificing fault coverage. Laouej [25] subsequently developed improved methods for emulating radiation-induced faults on FPGAs, establishing a methodological foundation for SEM IP-based reliability studies at ÉTS. The present framework builds on this lineage by closing the feedback loop between injection and DUT monitoring: it observes the design-under-test's UART output in real time during each injection and halts the campaign automatically upon failure detection, rather than requiring post-hoc log analysis. The platform also extends this prior work by adding Zynq UltraScale+ support and applying the methodology to processor-based and deep learning workloads not previously covered by earlier ÉTS tooling.

2.5 Summary

This chapter provided the background and literature context necessary for the work presented in Chapters 3 and 4. The physics of soft errors were introduced, explaining how energetic particles deposit charge at sensitive nodes in semiconductor devices, causing transient bit flips. For SRAM-based FPGAs, configuration memory SEUs are particularly severe: unlike user logic upsets which are transient, configuration memory upsets permanently alter the implemented circuit until explicitly corrected, making them the primary reliability concern in radiation-prone environments.

The AMD SEM IP core was described as the fault injection engine used in this work, operating through a UART command interface. The concepts of essential bits and critical bits were defined, along with conjunctive critical bits. The literature review covered the main FPGA fault injection methodologies and prior ÉTS work, and a research gap analysis identified the absence of a general-purpose SEM IP automation framework for the Zynq UltraScale+ platform applicable to processor-based and deep learning workloads — the gap this thesis addresses.

CHAPTER 3

AUTOMATION OF SEM IP SEU EMULATION

3.1 Automation of SEM IP

In this work, an automated SEU fault injection framework was developed using the AMD SEM IP core on a Zynq UltraScale+ FPGA. The goal was to systematically inject faults into the configuration memory and identify which essential bits, when flipped, cause an observable malfunction in the design under test. To detect these malfunctions, multiple instances of the target module are placed in parallel on the FPGA. Their outputs are continuously compared, and if a mismatch is detected between them, an interrupt is triggered, signaling that the injected fault caused one of the modules to behave incorrectly. This approach allows the framework to automatically identify critical bits without any manual observation. The following sections describe the UART communication and Python automation layer, the ICAP/PCAP configuration port considerations specific to the Zynq UltraScale+, and the PS-side initialization sequence required before the injection campaign can begin.

3.1.1 UART Communication and Python-based Automation

The first UART is connected to the Programmable Logic (PL) side and is dedicated to the SEM IP. This interface is used to send injection commands to the SEM IP and to receive feedback, including the addresses of injected errors and the status of each operation. The second UART is also connected to the PL and handles the initialization process, including communication with the ICAP and PCAP interfaces as described in PG187 and XAPP1298. Together, these two channels give the host PC full control over the injection process without requiring any manual interaction with the board after setup. Figure 3.1 illustrates the topology of the automation frame work and the UARTs lines.

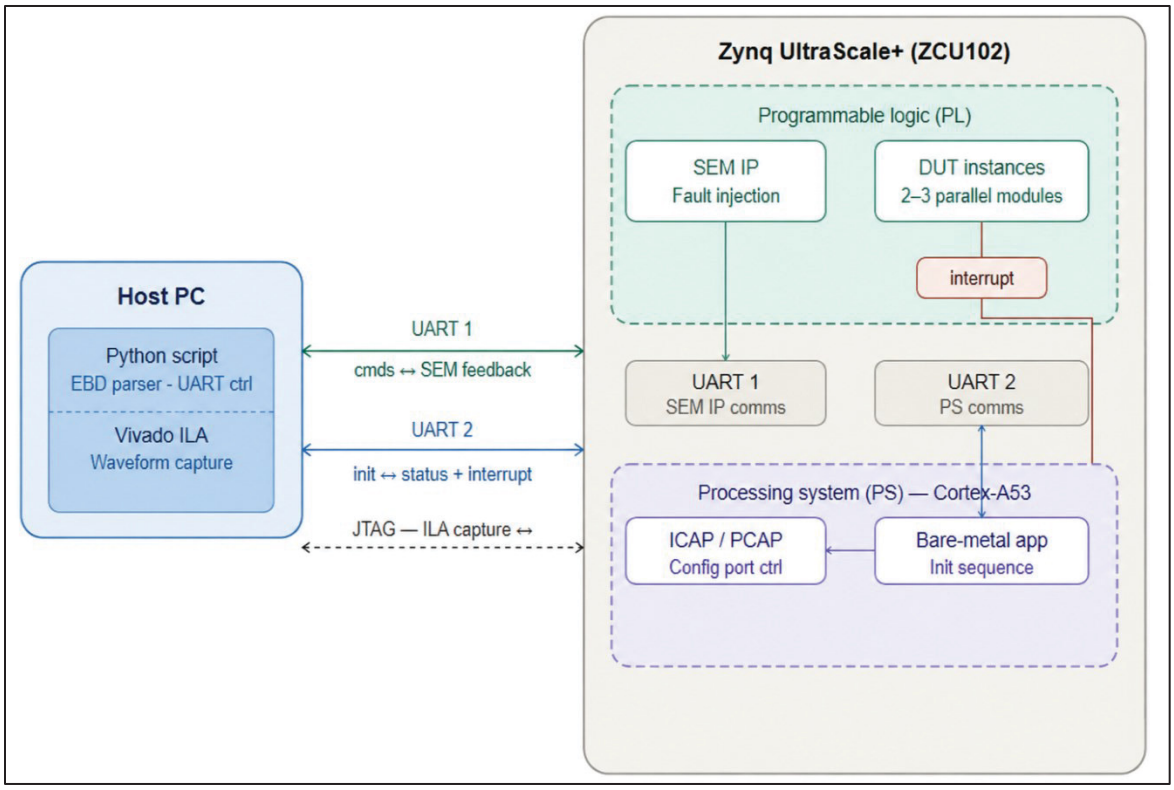


Figure 3.1 Host PC to Zynq UltraScale+ connection setup: UART 1 carries injection commands and SEM IP feedback; UART 2 handles PS-side initialization; JTAG is used for ILA waveform capture

To automate the fault injection campaign, a Python script was developed on the host PC. The script first parses the Essential Bits Description (EBD) file generated by Vivado, which contains the list of all essential bits in the design and their corresponding CRAM addresses. The user can specify a starting address, so the injection campaign can begin from any point in the EBD file. This is useful when a campaign needs to be resumed after an interruption, or when targeting a specific region of the design.

Once the campaign starts, the script iterates through the essential bits and sends injection commands to the SEM IP one at a time through the UART interface. Each error injection takes approximately 130 milliseconds. After each successful injection, the SEM IP sends a feedback response confirming the operation, and the script logs the address before moving to the next bit.

The campaign stops automatically when an interrupt is triggered. As will be explained in more detail in the following sections, the design under test includes two or three instances of the target module running in parallel. Their outputs are continuously compared, and if a mismatch is detected, indicating that one of the modules is producing wrong results due to an injected fault, an interrupt is generated. When either of these conditions is met, the Python script stops the campaign and reports the CRAM address at which the event occurred. This address is then recorded as a critical bit, since the fault injected at that location caused an observable malfunction in the system. Figure 3.2 describes the flow chart of the python campaign.

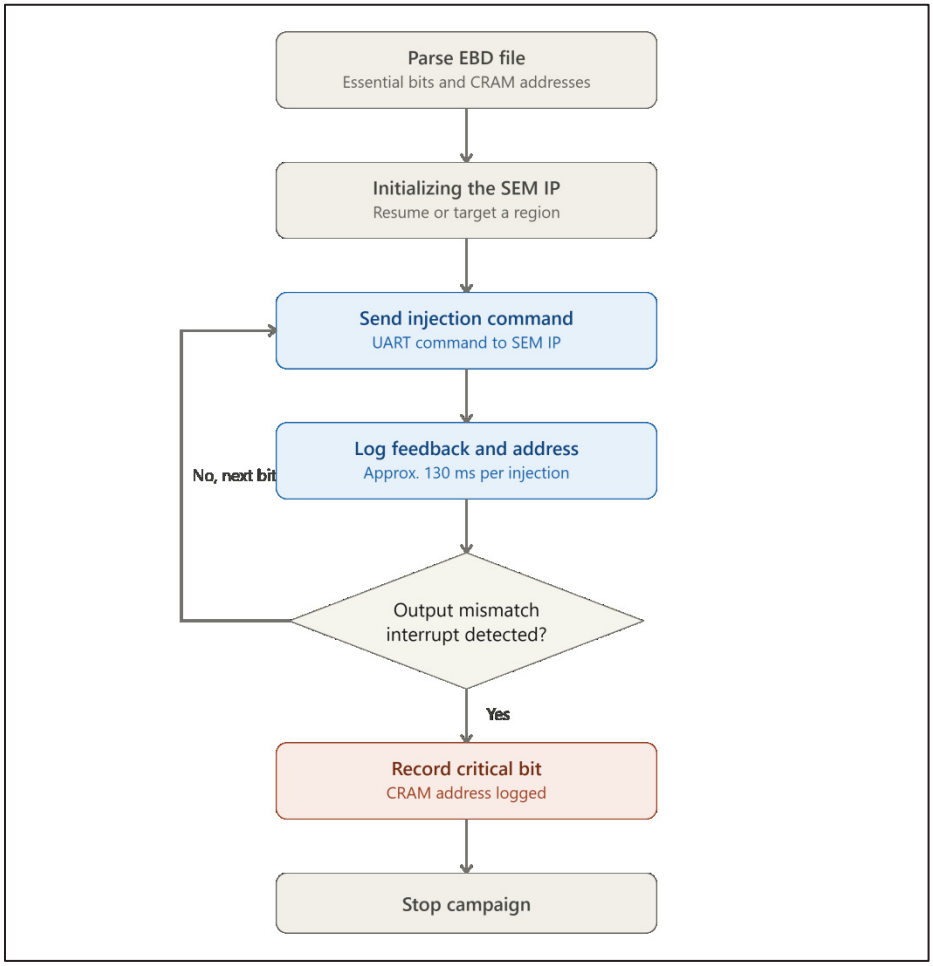


Figure 3.2 Main operations of the Python-based fault injection automation script

3.1.2 PS-Side Initialization of the SEM IP on Zynq UltraScale+

As described in the previous section, transferring configuration port control from PCAP to ICAP is a prerequisite for SEM IP operation on the Zynq UltraScale+. The mechanism by which this transfer is performed is described in XAPP1298, and it is carried out entirely in software running on the Processing System. Specifically, a bare-metal C application runs on one of the ARM Cortex-A53 cores of the Application Processing Unit (APU) of the Zynq UltraScale+ MPSoC. The APU is chosen over the real-time Cortex-R5F cores because the initialization sequence requires direct register-level access to the PCAP control registers and the EMIO GPIO interface, which is straightforward to implement in a bare-metal Vitis application targeting the A53.

The initialization sequence executed by this PS application proceeds as shown in figure 3.3. First, the application disables the PCAP interface, releasing the PS's hold on the configuration logic and making the ICAP available to the PL. Next, it enables the BUFGCE clock gating cell that feeds the 100 MHz clock to the SEM IP controller in the PL. This clock must be gated off when the PS holds PCAP control and switched on before the SEM controller is allowed to start. Once the clock is running, the PS drives the ICAP arbitration signals through the EMIO GPIO interface of the Zynq UltraScale+ block: it asserts cap_req to request ICAP access, waits for the SEM IP to assert cap_gnt confirming the grant, and confirms that cap_rel is deasserted to indicate the SEM controller now has exclusive access to the ICAP. With this handshake complete, the SEM controller initializes, scans the configuration memory, and begins reporting status through its UART interface. The PS application then polls the SEM status signals, specifically status_correction and status_uncorrectable, to monitor the controller's operating state and to respond to any events such as error detection or uncorrectable errors that require intervention.

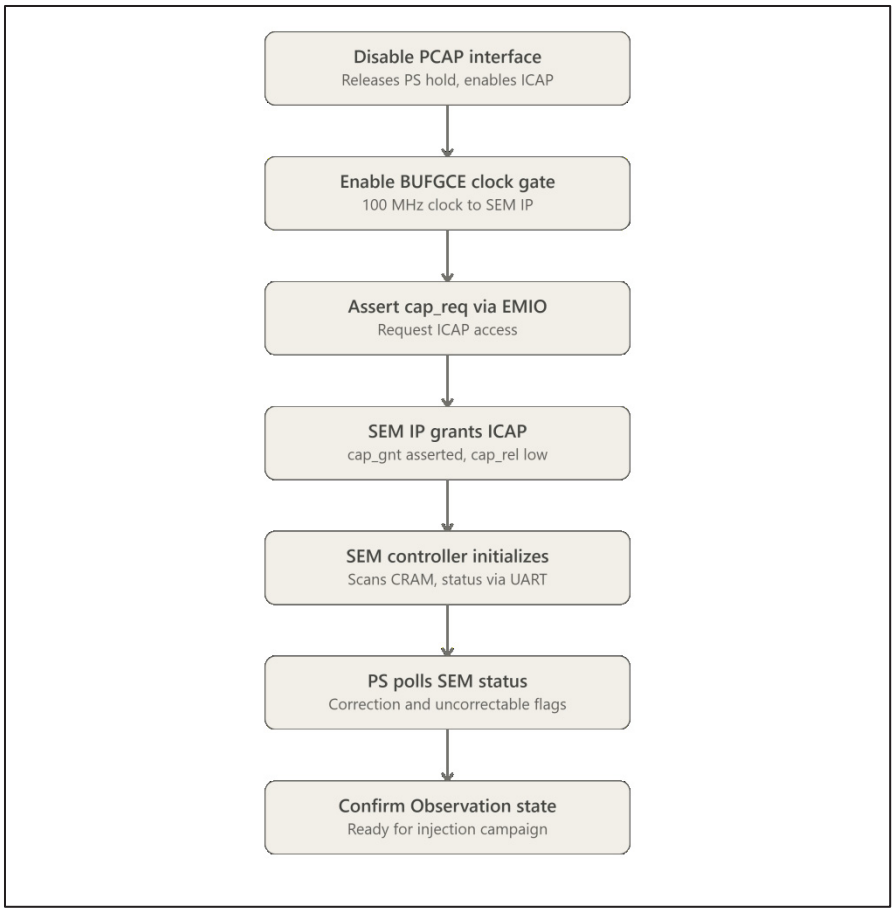


Figure 3.3 PS-side SEM IP initialization sequence on Zynq UltraScale+

In our implementation, this entire initialization handshake is triggered and monitored through the second UART interface connected from the FPGA board to the host PC, allowing the Python automation script to confirm that the SEM controller has completed initialization and is in the Observation state before beginning the fault injection campaign.

The key outcome of this architecture is that, once the PS-side initialization code is in place and the SEM IP is instantiated in the PL, the framework becomes fully design-agnostic with respect to the user logic under test. Any RTL design can be implemented in the remaining PL resources alongside the SEM IP, and the same initialization sequence, the same Python automation script, and the same essential bit injection workflow apply without modification. The only design-specific input required from the user is the EBD file generated by Vivado for that particular implementation, which captures the essential bits for the specific placed-

and-routed design. This generality is demonstrated in the following section, where the framework is applied to a simple sequential benchmark circuit, the ITC'99 b01 FSM [35], as an initial proof-of-concept validation of the complete injection and fault detection pipeline.

3.1.3 Injection Verification Through Readback

To confirm that fault injection command issued through the SEM IP actually results in a physical bit flip in the configuration memory, a readback verification procedure was carried out using Vivado's readback capability. Vivado allows the user to capture the current state of the FPGA configuration memory into a Readback Data file (.rbd), which contains the full content of the configuration frames at the moment of capture. This file can be read directly and compared against a known reference to detect any differences in the configuration memory content.

In this work, two .rbd files were captured for the same essential bit address: one before issuing the injection command through the SEM IP, and one immediately after. The two files were then compared bit by bit. The comparison revealed exactly one bit difference at the location corresponding to the injected CRAM address, confirming that the SEM IP successfully flipped the targeted bit in the configuration memory. This one-to-one correspondence between the commanded injection address and the observed difference in the readback data provides direct hardware-level evidence that the fault injection mechanism operates as intended and that the campaign results reflect real configuration memory upsets rather than software-level artifacts.

This verification step was performed during the initial validation of the framework and was not repeated for every injection in the full campaign, as the SEM IP feedback protocol already confirms each injection through the UART status response. However, the readback comparison serves as a ground-truth reference that validates the reliability of the SEM IP injection interface on the Zynq UltraScale+ platform.

3.2 Reliability Assessment of a Simple RTL Design Using the Automated SEM IP Platform

3.2.1 The b01 Benchmark FSM

To validate the automated fault injection framework before applying it to more complex designs, the ITC'99 benchmark circuit b01 was chosen as the design under test. The b01 is a finite state machine originally used as a standard benchmark for testing combinational and sequential circuits. It has 9 primary inputs, 2 primary outputs, and 8 internal states, making it compact enough to allow exhaustive analysis while still being representative of real sequential logic. Three identical instances of the b01 module, referred to as b01-1, b01-2, and b01-3, were instantiated in parallel on the same Zynq UltraScale+ device. All three instances share the same input stimulus, generated by a configurable linear feedback shift register (cLFSR) implemented by Jeremy Royer [38], which produces a pseudo-random sequence of input vectors that exercises the state machine continuously during the injection campaign. Figure 3.4 shows the b01 states.

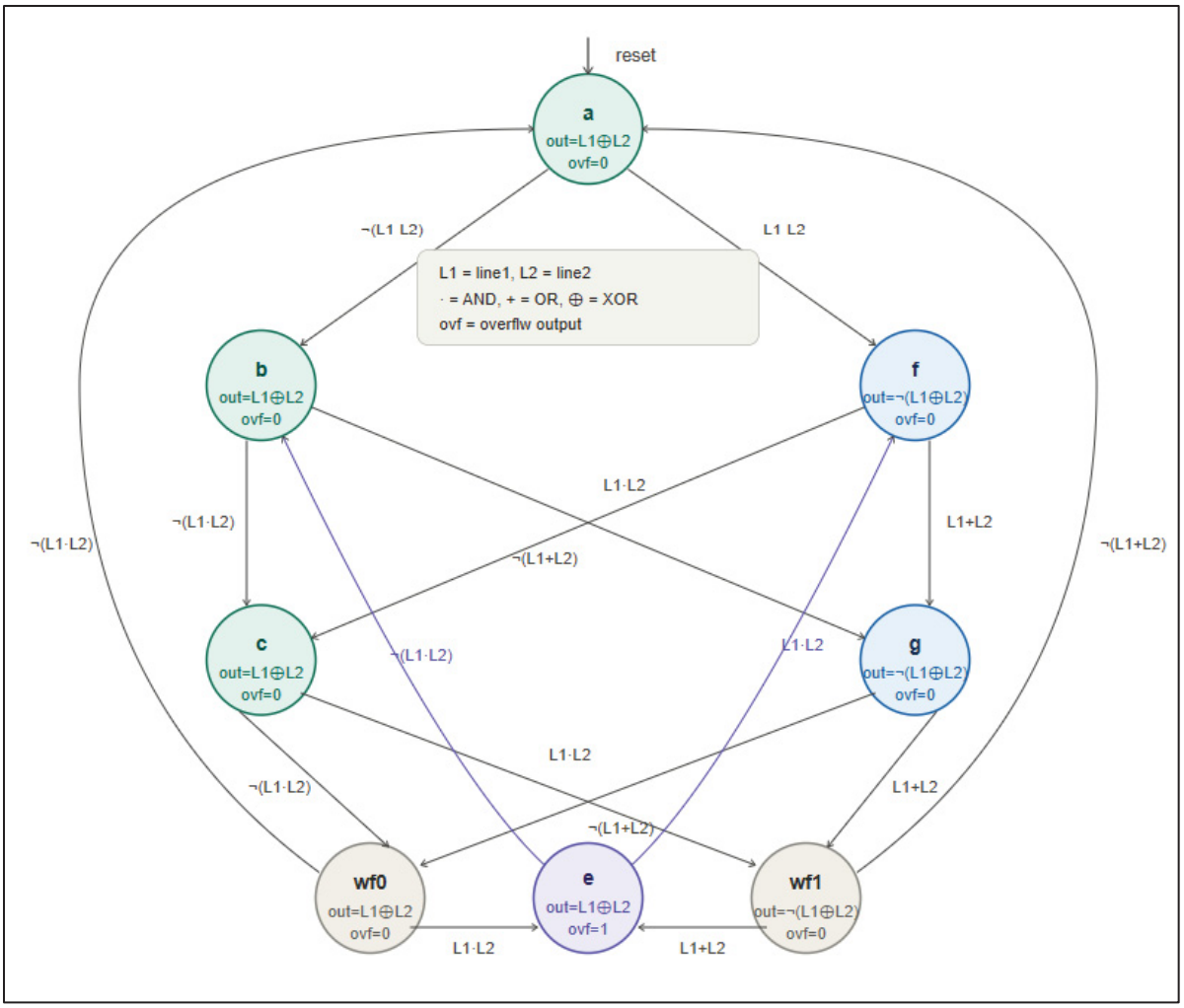


Figure 3.4 State transition diagram of the b01 benchmark FSM (8 states). Green states share the same transition condition structure as state a; blue states invert the XOR output; state e (purple) is the only state that asserts overflw = 1

3.2.2 Three-Module Parallel Architecture and Mismatch Detection

The core idea behind using three parallel instances rather than two is to enable majority voting: if one module produces an incorrect output due to a configuration memory upset, the other two remain unaffected and continue producing the correct result. A comparator circuit monitors the 2-bit outputs of all three b01 instances simultaneously. Any divergence between the outputs of the instances triggers an interrupt signal that is sent to the Python automation script on the host PC. When this interrupt is received, the script records the CRAM address

of the most recently injected essential bit and classifies it as a critical bit, since it caused a detectable behavioral difference between the three otherwise identical modules.

The output of each b01 module consists of two signals, referred to as outp (output) and overflw (overflow). Each module's outputs are concatenated into a 4-bit bus using a Concat IP block, as follows: the LSB corresponds to Line1, Bit 1 to Line2, Bit 2 to outp, and MSB to overflw. This 4-bit bus per module is connected to both the comparator logic and to the Integrated Logic Analyzer (ILA), which allows the output state of all three modules to be captured and recorded at the moment the mismatch interrupt fires. Figure 3.5 shows the vivado block diagram of the SEM IP framework along with the b01s design.

The ILA was configured to trigger on the mismatch interrupt signal, capturing a window of output samples around the moment the fault manifests. This capture provides a temporal record of the transition from correct to faulty behavior, which is the basis for the fault signature analysis described in the following section.

3.2.3 Fault Signature Definition and Classification

For each critical bit identified during the campaign, the ILA waveform captured at the moment of the mismatch interrupt constitutes what is referred to in this work as a fault signature. The fault signature records the 4-bit output bus of each of the three b01 instances over a time window that includes the last correct output cycle and the subsequent faulty output cycles after the configuration memory upset takes effect. By examining which of the four output bits diverges between the faulty module and the two healthy modules, it is possible to classify the nature of the failure and to identify which output signal of the b01 FSM is most affected by a given critical bit.

Each fault signature is characterized by two attributes: which b01 instance was affected (b01-1, b01-2, or b01-3), and which output bit or combination of output bits became corrupted. In the ILA traces, the two healthy modules continue producing matching outputs throughout the capture window, while the faulty module diverges on one or more bits. The specific bit or bits that diverge identify the corruption signature for that critical bit.

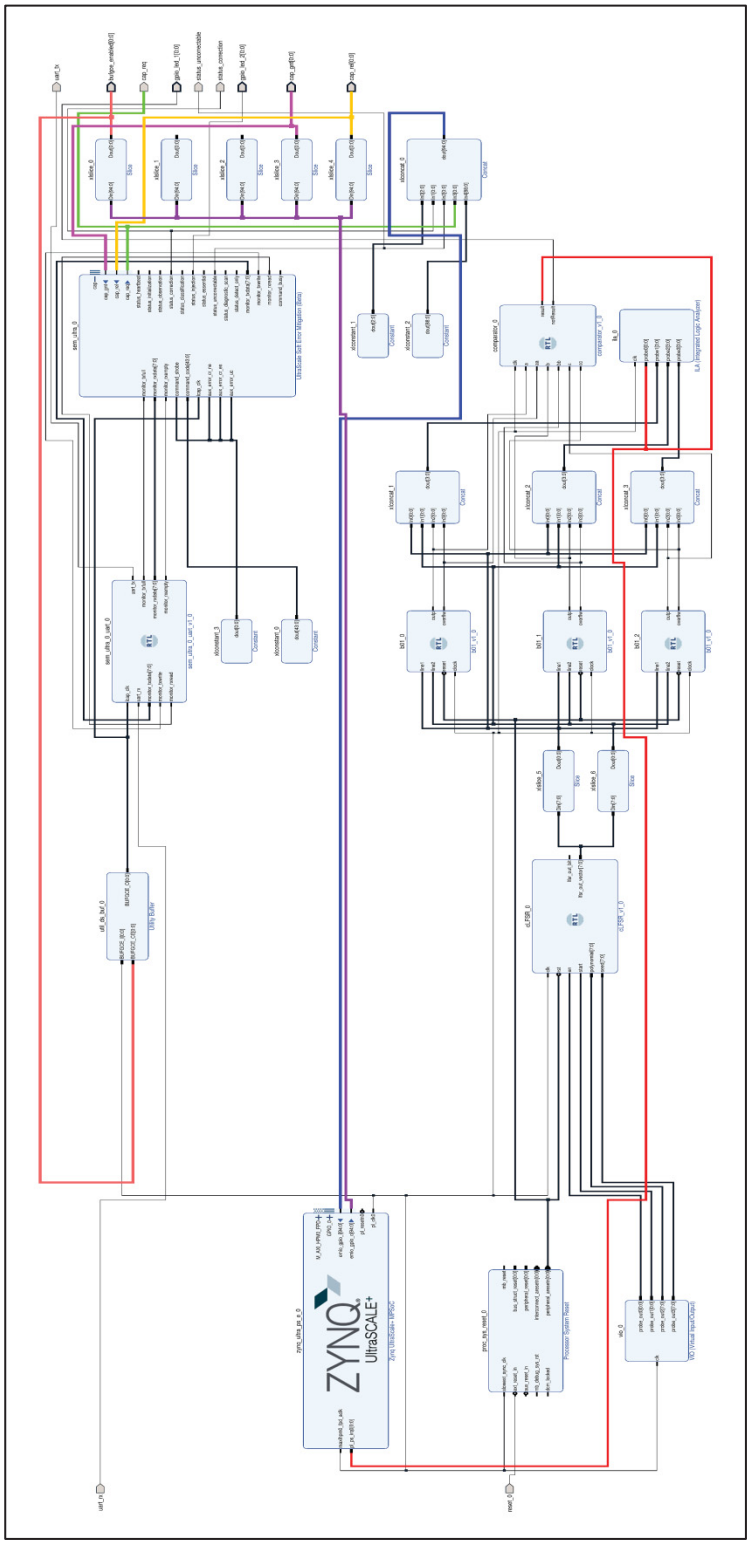


Figure 3.5 ILA probe connections for the three b01 instances. Each 4-bit bus encodes LSB (Line1), Bit 1 (Line2), Bit 2 (outp), and MSB (overflw)

Three distinct corruption patterns were observed across the 100 critical bits identified in this campaign as shown in table 3.1:

Table 3.1 Classification of critical bits by corrupted output bit(s) across 100 identified critical bits

Faulty Output Bit(s)	Count	Percentage
MSB and Bit 2 (outp + overflw)	58	58%
Bit 2 only (outp)	23	23%
MSB only (overflw)	19	19%

The most common failure mode, occurring in 58 out of 100 cases, involves simultaneous corruption of both the MSB (overflw) and Bit 2 (outp) output signals. This suggests that a significant portion of the critical configuration bits are associated with logic resources that drive multiple output signals of the FSM, or that the FSM state encoding is such that transitions corrupted by a single bit flip affect multiple outputs concurrently. The remaining failures affect either outp alone (23 cases) or overflw alone (19 cases), indicating single-output corruption patterns associated with routing or LUT resources more narrowly dedicated to one output path.

Table 3.2 Distribution of critical bits by faulty module instance

Faulty Module	Count	Percentage
b01-1	42	42%
b01-2	20	20%
b01-3	38	38%

Table 3.2 indicates that the distribution of affected module instances is not uniform: b01-1 was the faulty module in 42 cases, b01-3 in 38 cases, and b01-2 in 20 cases. This non-uniform distribution is expected and reflects the placement-dependent nature of FPGA fault injection. The 100 critical bits were identified from approximately 300 configuration memory

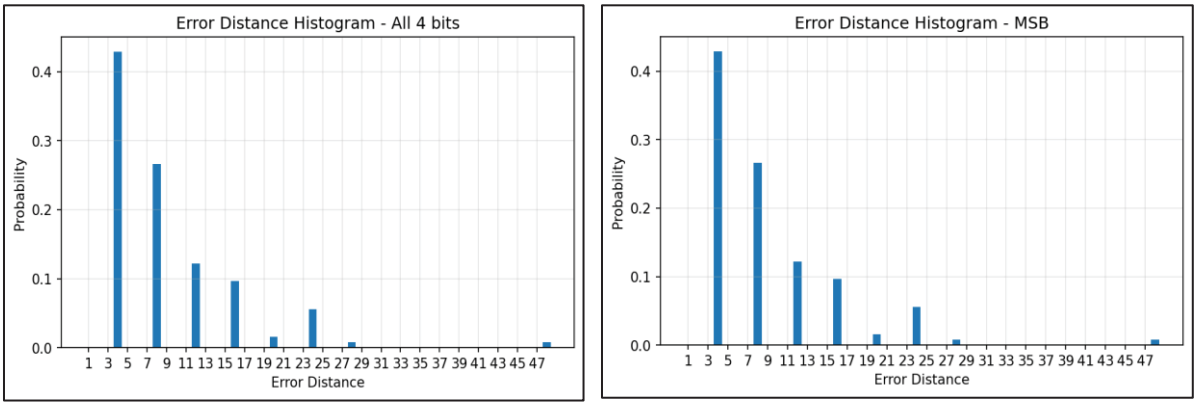
bit flips, covering EBD indices ranging from 129,911 to 134,923 — a span of roughly 5,000 essential bits within the design's CRAM footprint.

Examining the CRAM addresses reveals a clear regional partitioning between the three instances. The lower address region (frame prefix 0x0002C8, covering EBD indices approximately 129,911 to 133,919) is dominated by b01-1 and b01-3, which account for 25 and 19 critical bits respectively in this region, while b01-2 contributes only 2. The upper address region (frame prefix 0x0002C9, covering EBD indices approximately 134,767 to 134,923) shows a more even distribution, with b01-1 contributing 17, b01-2 contributing 18, and b01-3 contributing 19 critical bits. This spatial separation confirms that each b01 instance occupies a distinct physical placement region on the device, and that the EBD address space reflects this physical layout: the three instances' sensitive CRAM bits are concentrated in non-overlapping sub-regions, with b01-2's critical bits almost entirely confined to the upper address region.

3.2.4 Representative Fault Signature Examples

To illustrate the fault signatures recorded by the ILA, three representative examples are presented in Figure 3.6 to 3.8, one from each of the three corruption categories identified in Table 3.1.

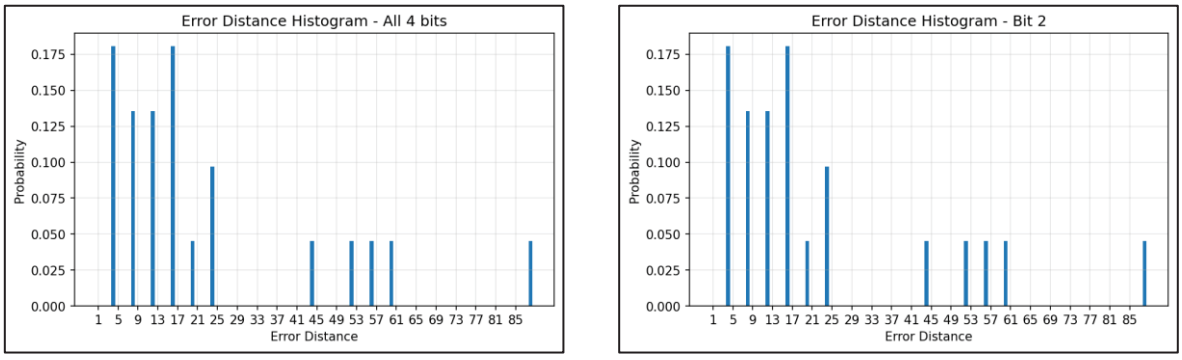
Type 1 — MSB and Bit 2 corrupted simultaneously (Critical Bit 1, b01-1). This is the most common failure pattern, accounting for 58% of critical bits. The overall ILA trace shows the moment the mismatch interrupt fires, with one module's 4-bit output diverging from the other two. The zoomed MSB channel confirms that the overflw signal of b01-1 transitions to an incorrect value while Bit 1 and LSB remain correct on all three modules. Figure 3.6 shows the histogram of error distance.



Overall — all 4 bits, all 3 modules *MSB channel — faulty module diverges*

Figure 3.6 Critical bit 1 ILA signature (Type 1): MSB and Bit 2 corrupted in b01-1. Non-faulty bits: Bit 1, LSB

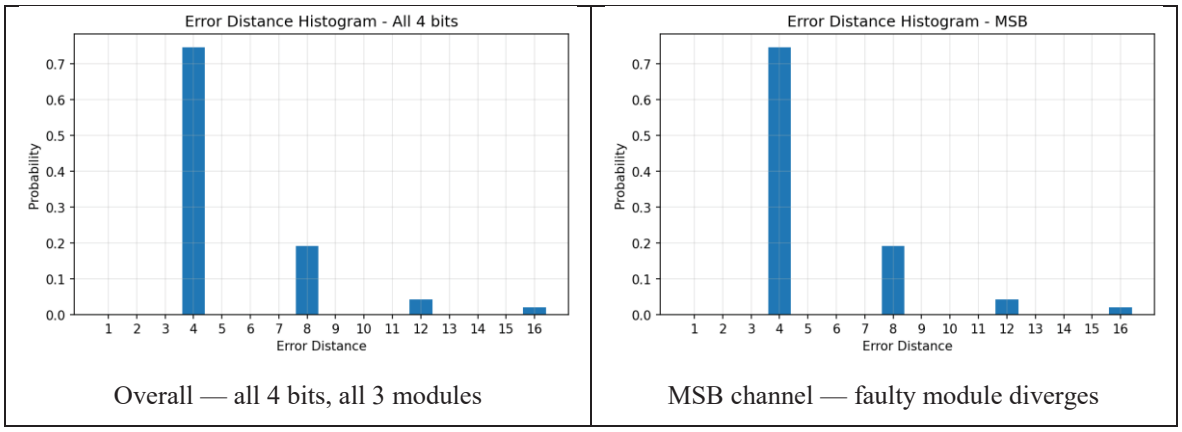
Type 2 — Bit 2 only corrupted (Critical Bit 7, b01-3). In this pattern, only the outp signal of the affected module diverges. The overflw signal, Bit 1, and LSB all remain identical across the three modules. This indicates that the upset bit is associated with a LUT or routing resource that exclusively drives the outp output path of the FSM.



Overall — all 4 bits, all 3 modules Bit 2 channel — faulty module diverges

Figure 3.7 Critical bit 7 ILA signature (Type 2): Bit 2 only corrupted in b01-3. Non-faulty bits: MSB, Bit 1, LSB

Type 3 — MSB only corrupted (Critical Bit 34, b01-1). Here only the overflw signal of the affected module deviates from the reference outputs. The outp, Bit 1, and LSB channels remain consistent across all three instances. This pattern is produced by configuration bits that exclusively affect the logic resources computing the overflow condition of the FSM.



Overall — all 4 bits, all 3 modules

MSB channel — faulty module diverges

Figure 3.8 Critical bit 34 ILA signature (Type 3): MSB only corrupted in b01-1. Non-faulty bits: Bit 2, Bit 1, LSB

3.2.5 Conjunctive Critical Bits

Of the 100 critical bits identified, 98 are single-bit critical bits, where a single configuration memory bit flip alone causes a detectable output mismatch, while the remaining 2 are conjunctive critical bits, where two bits must be flipped simultaneously to produce a failure. Table 3.3 summarizes this distribution.

Table 3.3 Single-bit vs. conjunctive critical bit distribution (100 CBs)

CB Type	Count (out of 100)	Percentage
Single-bit (non-conjunctive)	98	98%
Conjunctive (two-bit pair)	2	2%

It should be noted that the 100 critical bits reported here were not obtained by exhaustively sweeping all essential bits in the b01 design. The total number of essential bits in the three-

instance b01 implementation is 926,501, therefore, the number of critical bits for this design must be more than 100. The injection campaign was stopped after identifying 100 critical bits, as the primary objective at this stage was to validate the framework's ability to detect and classify critical bits rather than to obtain a complete critical bit fraction for this design.

3.3 Summary

This chapter presented the automated SEU fault injection framework developed for the Zynq UltraScale+ platform and its validation on the ITC'99 b01 benchmark FSM. The framework operates through a Python-based host script that communicates with the SEM IP over UART, parsing the Essential Bits Description file to sequentially inject faults into the configuration memory and monitoring the design under test for output mismatches via a three-module parallel architecture. The PS-side initialization sequence, including PCAP disable and ICAP handshake, was described as a prerequisite for SEM IP operation on the Zynq platform.

The b01 case study confirmed the framework's ability to detect and classify configuration memory SEUs. Three distinct fault signature types were identified through ILA captures: simultaneous corruption of the MSB and Bit 2 (Type 1, 58% of critical bits), corruption of Bit 2 only (Type 2, 30%), and corruption of the MSB only (Type 3, 12%). Of the 100 critical bits identified, 98 are single-bit critical bits and 2 are conjunctive two-bit pairs. The campaign was stopped at 100 critical bits, as the primary objective at this stage was to validate the detection and classification methodology rather than to exhaustively characterize the full critical bit fraction of the design.

CHAPTER 4

RELIABILITY ASSESSMENT OF A RISC-V SOFT-CORE PROCESSOR

4.1 RISC-V Architecture

RISC-V is an open-standard instruction set architecture (ISA) based on reduced instruction set computing principles. Unlike proprietary ISAs such as ARM or x86, RISC-V is freely available under open licenses, meaning anyone can implement it in hardware or use it as a software target without paying licensing fees or signing non-disclosure agreements. This openness has made it increasingly attractive for both academic research and industry, particularly in embedded systems, custom silicon, and FPGA-based processor implementations.

The base integer ISA comes in several variants defined by register width. RV32I uses 32-bit registers and is the simplest complete variant. The letter I stands for the base integer instruction set, which covers integer arithmetic, logical operations, load and store memory access, branches, and jumps. Additional functionality is provided through optional standard extensions: M adds integer multiplication and division, F and D add single and double precision floating-point, and C adds compressed 16-bit instructions. A processor implementing the base I extension alone is a fully functional general-purpose core capable of running arbitrary software including bare-metal C programs, which is exactly the case for the implementation used in this work. The basic architecture of the RISC-V is shown in figure 4.1.

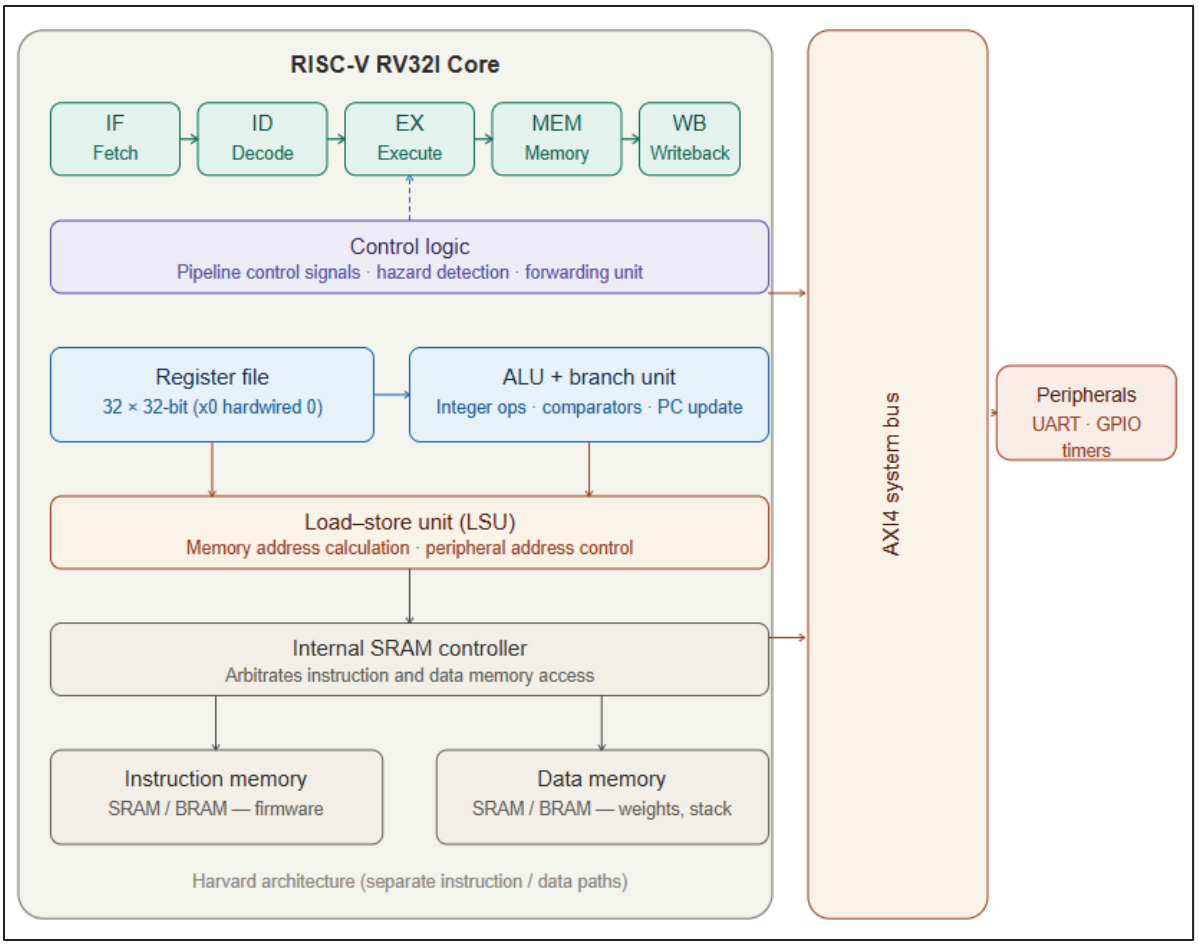


Figure 4.1 Block diagram of the RISC-V RV32I processor core, showing the five-stage pipeline, control logic, register file, ALU, LSU, SRAM controller, Harvard-architecture memories, and AXI4 system bus. Adapted from Lim et al. [27]

In this work, the Potato processor is used, a compact RV32I implementation written in VHDL. It implements the base integer ISA without the M, F, or D extensions, meaning all arithmetic is handled in software and there is no hardware floating-point unit. This constraint is relevant to the deep learning case study described later in this chapter, where it motivates the use of fixed-point quantization for the neural network weights and activations. The Potato core is connected to on-chip block RAM for instruction and data memory and communicates with the outside world through a UART peripheral mapped into its memory address space.

Reliability assessment of RISC-V processors is relevant given their growing deployment in environments where radiation-induced soft errors are a concern. Small satellites and

CubeSats [33] increasingly rely on COTS FPGA boards running RISC-V soft-cores for on-board data processing, where the terrestrial neutron flux and low-Earth orbit cosmic ray environment both contribute to SEU rates in SRAM-based FPGAs. Understanding which configuration bits are critical for a given RISC-V implementation, and how faults in those bits manifest at the output level, is a necessary step toward designing fault-tolerant systems based on this architecture. As mentioned before (section 2.2.5), prior work by Aranda et al. [7] has shown that RISC-V processors implemented on Xilinx 7-series FPGAs exhibit workload-dependent critical bit populations, meaning the program being executed affects which bits matter.

4.2 RISC-V Toolchain Setup and Firmware Deployment

The firmware for the Potato core was developed on an Ubuntu virtual machine, while Vivado ran on Windows on the host machine. This separation was necessary because the RISC-V GNU toolchain is Linux-native. The toolchain was built from source by cloning the official riscv-gnu-toolchain repository and configuring it for the RV32I ISA with the ILP32 ABI, which matches the Potato core. The key point in the configure step is to explicitly specify --with-arch=rv32i and --with-abi=ilp32, since the default configuration targets a different ABI. After compilation, the toolchain binaries are added to the shell PATH through the .bashrc file so that riscv32-unknown-elf-gcc is available system-wide.

4.2.1 Toolchain Installation

Following the command shown in algorithm 4.1, the RISC-V GNU toolchain will be cloned into the device.

Algorithm 4.1 RISC-V GNU toolchain installation

```
git clone https://github.com/riscv/riscv-gnu-toolchain
sudo apt-get install autoconf automake autotools-dev curl python3 python3-pip \
    libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo \
    gperf libtool patchutils bc zlib1g-dev libexpat-dev ninja-build git cmake \
    libglib2.0-dev libslirp-dev
sudo chmod +wrx /opt/
cd riscv-gnu-toolchain
./configure --prefix=/opt/riscv-toolchain --with-abi=ilp32 --with-arch=rv32i
make
```

After compilation, the lines shown in algorithm 4.2 are added to ~/.bashrc to make the toolchain available in every terminal session:

Algorithm 4.2 Toolchain PATH configuration in ~/.bashrc

```
export PATH="$PATH:/opt/riscv-toolchain/bin"
export RISCV="/opt/riscv-toolchain"
```

A Python virtual environment is also required for the upload and automation scripts. It is created and activated from the project root folder as shown in algorithm 4.3.

Algorithm 4.3 Python virtual environment setup

```
python3 -m venv .venv
source .venv/bin/activate
python3 -m pip install -r requirements.txt
```

4.2.2 Bootloader and Application Compilation

The Potato core uses a two-stage boot process. When the FPGA is programmed and the processor starts, it runs a bootloader stored in a ROM implemented as an AXI ROM IP in the Vivado block design. The bootloader initializes the UART and waits to receive a binary

application image over the serial connection. This design means the application firmware, the Fibonacci program or the image classifier, does not need to be compiled into the bitstream, and can be changed and re-uploaded without re-synthesizing the hardware.

Before compiling the applications, the bootloader must be updated to the version used in this project, since the default bootloader in the repository is different. The steps are:

1. Replace software/bootloader/main.c with the project version of that file.
2. Replace software/bootloader/Makefile with the project version.
3. Recompile the bootloader by running make in the software/bootloader/ directory.
4. In the Vivado project, recreate the AXI ROM IP using the newly generated bootloader.coe file. It is recommended to recreate the IP rather than just updating the file path, since the IP tends to cache the old file.
5. Regenerate the bitstream in Vivado.

Algorithm 4.4 to 4.6 cover these steps. The git submodules must also be initialized before compiling anything:

Algorithm 4.4 Git submodule initialization

```
git submodule update --init
```

The bootloader is then compiled from its directory:

Algorithm 4.5 Bootloader compilation

```
cd software/bootloader/
make
```

Each application is compiled from its own subdirectory. For the Fibonacci program:

Algorithm 4.6 Fibonacci application compilation

```
cd software/hello/
make
```

4.2.3 Firmware Upload and Execution

With the toolchain installed and the bootloader compiled into the bitstream, the firmware image is prepared and uploaded to the running RISC-V core over UART. A terminal application is needed for this step, Tera Term is used in this work. The upload procedure is as follows.

First, the binary image for the bootloader is prepared by padding the compiled binary to 128 KB, which is the fixed image size the bootloader expects as shown in algorithm 4.7.

Algorithm 4.7 Binary image preparation for bootloader upload

```
cd software/hello/
cat hello.bin /dev/zero | head -c128k | pv -s 128k -L 14400 >
image_for_bootloader.bin
```

The upload steps are then:

1. Connect the FPGA board to the PC via the USB port connected to the FTDI chip that interfaces with the RISC-V UART.
2. Open Tera Term and create a new serial configuration (Setup > Serial Port). Set the speed to 115200, data to 8 bits, parity to none, stop bit to 1, and flow control to none. The correct COM port may require a few attempts to identify.
3. Program the FPGA with the bitstream.
4. The Tera Term terminal should display the message ** Potato Bootloader - waiting for application image **. If this message does not appear, try a different COM port and reprogram the FPGA.
5. Send the image file via File > Send File in Tera Term. Browse to image_for_bootloader.bin and make sure the Binary option is checked before clicking Open.

6. Once the transfer completes, the terminal should display ** Riscv - fibonacci sequence ** followed by the continuous Fibonacci output, confirming the application is running correctly.

4.3 Two-Core Parallel Architecture and Fault Detection Methodology

The fault injection methodology for the RISC-V reliability assessment follows the same principle used in the b01 proof-of-concept in Chapter 3, but adapted to a processor-based system. Two identical Potato RV32I cores, referred to as RISC1 and RISC2, are instantiated in the PL of the Zynq UltraScale+ FPGA. Both cores execute the same firmware, loaded simultaneously over UART at startup, and both run continuously in parallel, writing their output to their respective UART TX lines.

A comparator circuit in the PL monitors the TX output of both cores simultaneously. When the two TX signals diverge, meaning one core is transmitting a different byte from the other, the comparator triggers an interrupt that is routed to the PS. The bare-metal application running on the ARM Cortex-A53 detects this interrupt and immediately sends a notification to the host PC over the second UART, signaling the Python automation script that a critical bit has been found. The script records the CRAM address of the last injected essential bit, logs the failure mode based on the UART output already captured from both cores, and resumes the campaign from the next address.

Simultaneously, the Integrated Logic Analyzer (ILA) connected to both UART TX lines captures the byte stream around the moment the interrupt fires, providing a hardware-level record of the divergence between the two cores. This capture is used to verify the mismatch and to confirm the failure mode observed on the faulty core. The ILA trigger is set to the mismatch interrupt signal, so the capture always shows the exact bytes being transmitted at the point of fault manifestation.

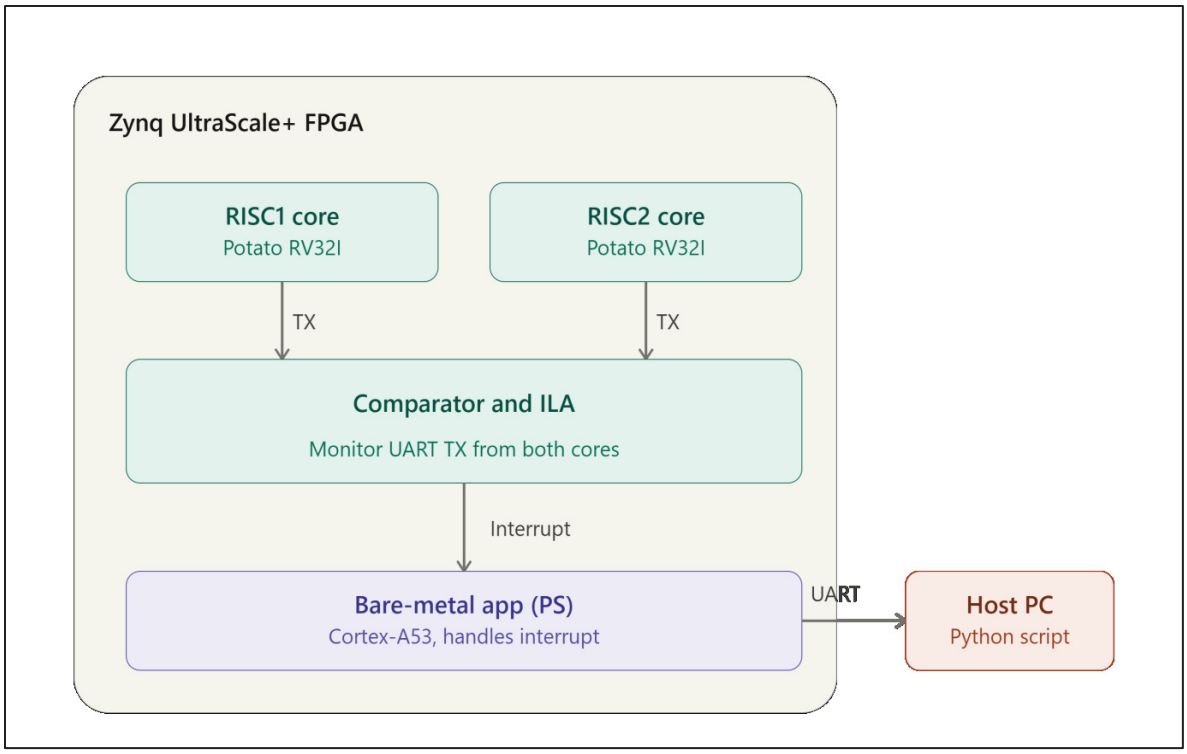


Figure 4.2 Two-core parallel architecture and fault detection methodology for the RISC-V reliability assessment

This architecture shown in figure 4.2 is identical for both workloads described in this chapter: the Fibonacci program and the quantized deep learning image classifier. The only difference between the two case studies is the firmware loaded into the cores. The hardware, two Potato cores, the comparator, the ILA, and the Python automation script, remain unchanged.

4.4 Fibonacci Program: Application and SEU Emulation Results

4.4.1 Application Description

The Fibonacci program runs an infinite loop that computes and prints the Fibonacci sequence over UART. Each iteration of the loop prints the next number in the sequence followed by a comma and a line break after every N values. Both cores produce identical output under normal conditions. A sample of the correct output as captured from both cores during a healthy campaign cycle is shown in algorithm 4.8.

Algorithm 4.8 Sample correct Fibonacci output captured over UART

```
** Riscv - fibonacci sequence 10416**
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,
6765,10946,17711,28657,46368,75025,121393,196418,317811,514229,
832040,1346269,2178309,3524578,5702887,9227465,14930352,24157817,
39088169,63245986,102334155,165580141,267914296,433494437,...
```

Any divergence between the two UART streams, whether in the numeric values, the formatting characters, or the timing of transmission, causes the comparator to trigger the interrupt and the Python script to log the current CRAM address as a critical bit.

4.4.2 Injection Campaign and Address Coverage

The fault injection campaign was run across multiple ranges of essential bit indices in the EBD file, selected arbitrarily to sample different regions of the configuration memory rather than sweeping it exhaustively. The ranges were chosen at various points across the EBD address space with the goal of observing whether critical bit density varies across different memory regions, without any prior knowledge of which regions would be more or less sensitive. A region around index 3,000,000 to 3,005,406 was found to be completely clean, producing no critical bits. In total, 43,448 bit flips were performed across all tested ranges, out of 4,944,721 essential bits in the full design (two Potato cores plus all surrounding logic). Of those 43,448 injections, 100 produced a detectable output mismatch, yielding a critical bit density of 100/43,448 = 0.23% within the tested regions. This figure reflects the density of critical bits across the arbitrarily sampled portions of the configuration memory and does not represent a global critical bit fraction for the full design, which would require an exhaustive sweep of all 4,944,721 essential bits.

Table 4.1 Tested EBD index ranges and number of critical bits found per range

EBD Index Range	Critical Bits Found	Proportion of CB/Essential Bit	Estimated Time to Find CBs (min)	Affected Core
50,000 – 59,158	10	0.10%	29	RISC2
100,000 – 105,241	10	0.10%	21	RISC2
200,000 – 200,351	5	1.42%	6	RISC2
250,000 – 250,837	5	0.59%	7	RISC2
300,000 – 301,964	5	0.25%	9	RISC2
400,000 – 404,284	5	0.11%	14	RISC2
520,000 – 522,190	5	0.22%	10	RISC2
630,000 – 631,871	5	0.26%	9	RISC2
700,000 – 702,648	5	0.18%	11	RISC2
850,000 – 862,839	2	0.01%	30	RISC2
2,000,000 – 2,000,274	5	1.82%	6	RISC2
2,100,000 – 2,100,228	3	1.31%	4	RISC2
2,400,000 – 2,400,091	5	5.49%	6	RISC2
2,700,000 – 2,700,786	5	0.63%	7	RISC2
2,900,000 – 2,900,078	5	6.41%	6	RISC2
4,000,000 – 4,000,018	5	26.32%	6	RISC1
4,200,000 – 4,200,012	5	38.49%	6	RISC1
4,300,000 – 4,300,060	7	11.48%	8	RISC1

Table 4.1 Continued Tested EBD index ranges and number of critical bits found per range

EBD Index Range	Critical Bits Found	Proportion of CB/Essential Bit	Estimated Time to Find CBs (min)	Affected Core
4,400,322	1	-	-	RISC1
4,500,000 – 4,500,498	2	0.40%	3	RISC1

IP Address Regions: A notable observation from the address distribution is that critical bits 1 through 80 all affect RISC2, while critical bits 81 through 100 all affect RISC1. This is consistent with the placement-dependent nature of FPGA fault injection: each RISC-V core occupies a distinct physical region of the device, and the EBD file organizes essential bits spatially, so the two cores' critical bits appear in separate non-overlapping address regions.

Proportion of Critical Bits to Essential Bits: The CB/Essential Bit ratio column in Table 4.1 reveals that the density of critical bits is far from uniform across the configuration memory. In the lower index ranges associated with RISC2, the ratio stays mostly below 0.5%, with the 50,000–59,158 and 100,000–105,241 ranges both at around 0.10%, meaning roughly one in every thousand essential bits in those regions caused a detectable failure when flipped. The 850,000–862,839 range drops even lower to 0.01%, suggesting that this region of the CRAM contains many essential bits that implement logic which is either non-critical to the UART output path or whose corruption is masked by the surrounding logic.

However, the picture changes significantly in higher index ranges. The 2,400,000–2,400,091 and 2,900,000–2,900,078 ranges reach 5.49% and 6.41% respectively, meaning roughly one in every fifteen to twenty essential bits in those narrow windows is critical. The most striking ratios appear in the RISC1 region: the 4,000,000–4,000,018 range reaches 26.32%, and the 4,200,000–4,200,012 range reaches 38.49%, meaning that in those small but concentrated windows, nearly half of the essential bits are critical. These high-density regions likely correspond to CRAM frames that implement particularly sensitive parts of the processor.

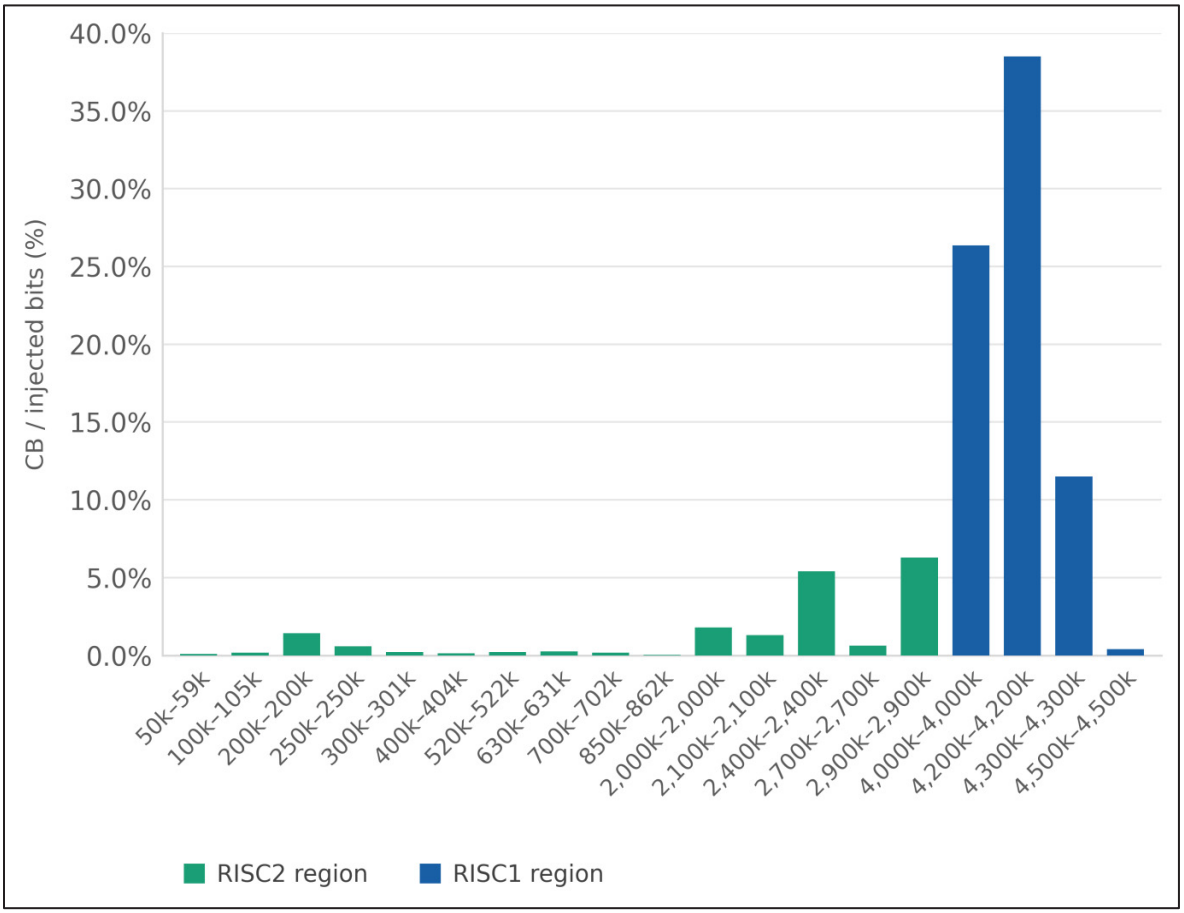


Figure 4.3 CB density spectrum across tested EBD index ranges. Bar colour indicates risk level (green = low, blue = high). Dashed line separates RISC2 (lower addresses) from RISC1 (higher addresses) placement regions

This non-uniform distribution shown in figure 4.3 has a practical implication for reliability assessment. A uniform random injection strategy that samples essential bits evenly across the full EBD address space would allocate the same number of injections to low-density regions like 850,000–862,839 as to high-density regions like 4,200,000–4,200,012, leading to an underrepresentation of the most vulnerable CRAM areas in the results. This suggests that the true critical bit fraction for the complete Potato RV32I implementation on this device is likely not uniform, and that a full exhaustive sweep would reveal significant spatial

clustering of critical bits around the logic resources that implement the processor's core datapath.

Estimated Time to Find Critical Bits: The estimated time to find the critical bits in each range, also shown in Table 4.1, was calculated based on two components: the injection time of approximately 130 ms per essential bit, and a fixed overhead of approximately one minute per critical bit found, which accounts for the manual steps required after each mismatch interrupt, reading and saving the ILA capture, noting the failure mode from the Tera Term output, resetting the board, reloading the firmware, and restarting the injection campaign from the next address. For a range of N essential bits containing K critical bits, the estimated time is therefore (N × 0.130 / 60) + K minutes.

The time estimates reveal a clear inverse relationship between CB density and the time needed to find a given number of critical bits. In the low-density ranges such as 50,000–59,158 and 850,000–862,839, the campaign spent approximately 29 and 30 minutes respectively to find 10 and 2 critical bits, because most of that time was consumed sweeping through the large number of non-critical essential bits in between. In contrast, high-density ranges such as 2,400,000–2,400,091 and 4,200,000–4,200,012 required only 6 minutes each to find 5 critical bits, because the critical bits were tightly clustered and the sweep through non-critical bits was short. The most extreme case is 4,200,000–4,200,012, where 41.66% of essential bits were critical, meaning the injection campaign encountered a critical bit roughly every 2 to 3 injections, keeping the overhead-per-CB-found very low relative to the sweep time.

The total estimated time across all 20 tested ranges is approximately 198 minutes, or just over 3 hours, to identify 100 critical bits. This figure highlights both a strength and a limitation of the current framework. The strength is that the 130 ms injection latency, while slower than ICAP-based platforms, is fast enough to make targeted campaigns in high-density regions very efficient, six minutes to find five critical bits in a 90-bit window is a practically useful throughput. The limitation is that sweeping low-density regions is expensive in time relative to the yield: the 50,000–59,158 range spent 29 minutes and found only 10 critical bits in 9,158 essential bits. This suggests that an adaptive injection strategy, one that uses an initial coarse sweep to identify high-density windows and then focuses

subsequent effort on those windows, would substantially reduce the total time required to characterize the complete critical bit population of the design.

4.4.3 Failure Mode Classification

The failure modes observed across the 100 critical bits were classified based on the UART output of the affected core captured by Vivado ILA at the time the interrupt fired. Five distinct failure categories were identified, described below.

Stops — the affected core halts completely and produces no further UART output. This was the most common failure mode. It indicates that the upset bit is associated with logic resources critical to the processor's ability to complete an instruction fetch, decode, or execute cycle. In several cases, the core stops mid-line, with the last transmitted character being a partial Fibonacci number or a formatting character.

Wrong Fibonacci numbers — the core continues running and printing output, but one or more values in the sequence are incorrect. In some cases, the correct sequence resumes after a few iterations, while in others the output remains permanently wrong. This mode indicates corruption of a data path resource such as an ALU path or a register connection where the processor's control flow is unaffected but its arithmetic results are corrupted. Examples include critical bits 71, 76–80, and 82–85.

Periodic wrong output — the core produces a repeating pattern of incorrect output interspersed with otherwise correct output, typically on every cycle of the Fibonacci print loop. This suggests the upset bit affects a resource exercised once per loop iteration rather than on every clock cycle — for example, a path in the output formatting logic or a counter tracking loop progress. Examples include critical bits 14, 18–19, 38–42, and 50.

Wrong characters or formatting — the core continues running but the UART output contains garbled characters, wrong line formatting, or corrupted punctuation rather than wrong numeric values. Examples include critical bits 24, 34, 37, and 96.

Invalid instruction trap — the core generates a POTATO_MCAUSE_INVALID_INSTR exception, indicating that the upset bit has corrupted an opcode or an instruction memory

read path such that the processor attempts to execute an illegal instruction. This was observed in critical bit 68.

Table 4.2 summerizes all the failure modes with distribution of the critical bits.

Table 4.2 Summary of the failure mode distribution across the 100 critical bits

Failure Mode	Count	Percentage
Stops (core halts)	62	62%
Wrong Fibonacci numbers	16	16%
Periodic wrong output	14	14%
Wrong characters / formatting	7	7%
Invalid instruction trap	1	1%

4.4.4 Conjunctive Critical Bits

Of the 100 identified critical bits, 8 were conjunctive, meaning a single bit flip at that CRAM address alone does not cause a mismatch, and the failure only manifests when two specific bits are flipped simultaneously. These cases are noted for critical bits 5, 6, 13, 44, 46, 92, 93, and 98. As shown in table 4.3, the remaining 92 are single-bit upsets where flipping one essential bit is sufficient to trigger the comparator interrupt. The near-total dominance of single-bit critical events is consistent with the b01 findings and validates the single-SEU injection model used by the SEM IP as appropriate for characterizing this processor implementation.

Table 4.3 Single-bit vs. conjunctive critical bit distribution (Fibonacci workload, 100 CBs)

CB Type	Count (out of 100)	Percentage
Single-bit (non-conjunctive)	92	92%
Conjunctive (two-bit pair)	8	8%

4.5 Deep Learning Inference on RISC-V: Application and SEU Emulation Results

4.5.1 Quantized Deep Learning Model for Image Classification

The deep learning application used in this case study is a binary image classifier trained on a subset of the CIFAR-10 dataset [24], specifically the ship and cat classes. The input images were converted to grayscale and resized to 32×32 pixels, reducing each image to a 1,024-element vector. Before any training took place, all pixel values were converted to Q16 fixed-point format. This means the entire pipeline, data preprocessing, forward pass, weight updates, and final evaluation, operated on Q16 values from the start. There was no separate floating-point training phase followed by post-training quantization. The model was trained to convergence directly on the quantized data, so the learned weights and decision boundaries are inherently adapted to the Q16 number representation and the arithmetic approximations described below.

Dataset and training configuration. The dataset was split into a training set of 8,000 images, a validation set of 2,000 images (validation split of 0.2), and a separate test set of 2,000 images. Training was run for 50 epochs with a batch size of 32 and a learning rate of 0.001 using the Adam optimizer [23]. The loss function was binary cross-entropy and the monitored metric was accuracy.

Architecture. The network is a fully connected two-layer architecture: an input layer of 1,024 neurons (one per grayscale pixel), a hidden layer of 12 neurons with ReLU activation, and a single output neuron with sigmoid activation. The small hidden layer size of 12 neurons was chosen deliberately to keep the weight memory footprint small enough to fit in the Potato core's block RAM and to keep inference time short enough for the UART-based fault injection monitoring loop. Figure 4.4 illustrates the model architecture.

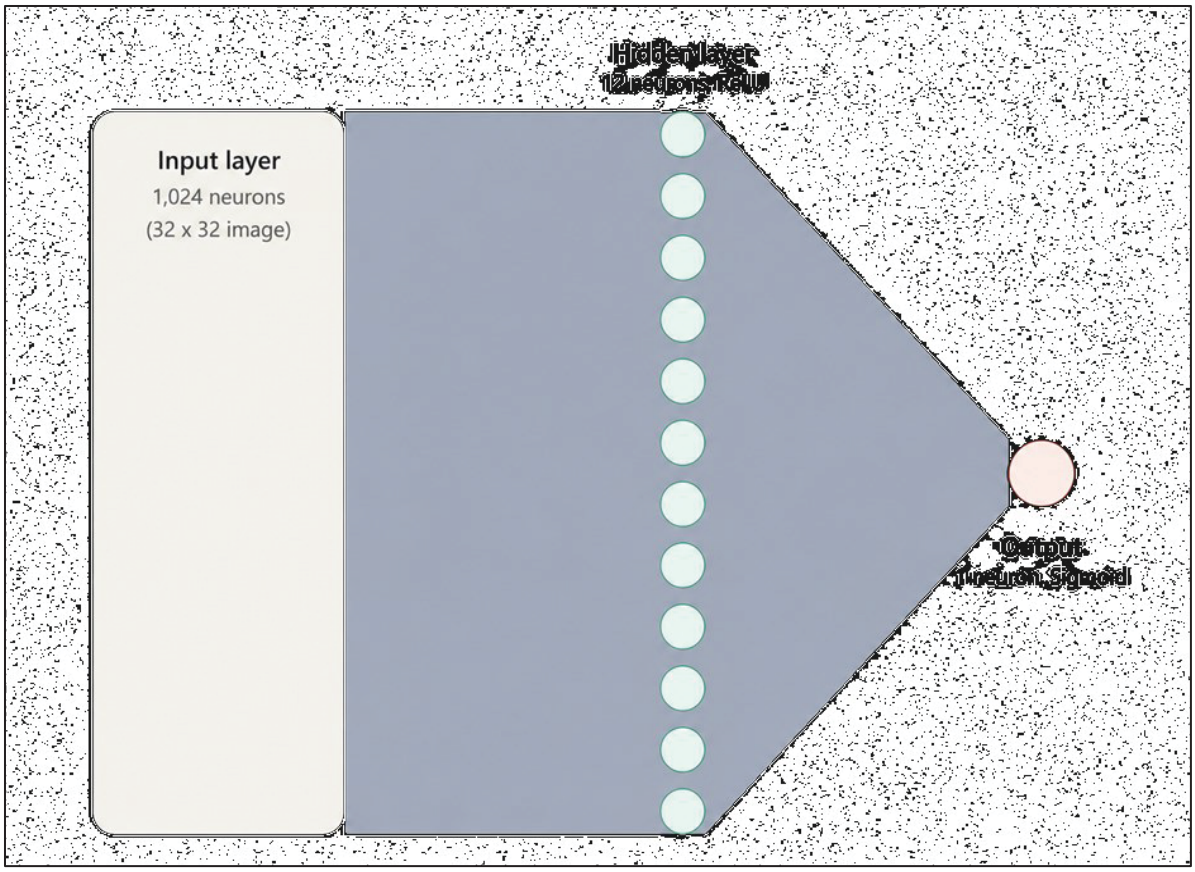


Figure 4.4 Fully connected two-layer architecture of the quantized image classifier: 1,024-neuron input layer, 12-neuron ReLU hidden layer, and a single sigmoid output neuron

Quantization format. The Q16 format uses one sign bit, seven integer bits, and eight fractional bits. This format provides sufficient dynamic range for both the pixel values and the trained weight values while keeping all multiplications within 32-bit integer arithmetic, which the base RV32I ISA supports without any extensions.

Custom fixed-point multiplication. Multiplying two Q16 values produces a result scaled by 2^16, so a right shift by 16 is required after each multiplication to restore Q16 scaling. To prevent overflow during accumulation, the multiplication is performed in 64-bit integer arithmetic before the shift and then clipped to the int32 range to simulate the overflow behaviour of the RISC-V core:

$$r = \frac{a \times b}{2^{16}} \tag{4.1}$$

$$r' = \max(-2^{31}, \min(r, 2^{31} - 1)) \tag{4.2}$$

This operation was used consistently in both the Python training pipeline and the bare-metal C inference kernel on the Potato core, ensuring bit-identical arithmetic between simulation and hardware.

Custom sigmoid approximation. The standard sigmoid function cannot be computed on RV32I without a software floating-point library. A five-segment piecewise-linear approximation was implemented and used during both training and inference. The five segments cover: below −5 (output ≈ 0), between −5 and −2.5, between −2.5 and 1, between −1 and 2.5, between 2.5 and 5, and above 5 (output ≈ 1).

Equation (4.3) gives the explicit form of this piecewise-linear approximation, where X is the real-valued input to the sigmoid and f(X) is the approximated output. Within each segment, the slope and intercept were chosen by fitting a line to the true sigmoid function over that interval, minimizing the maximum absolute error within each segment, which remains below 0.025 across the full input range. The largest error occurs near |X| ≈ 1.7, where the curvature of the sigmoid is highest relative to the chosen slope.

For the RV32I implementation, all values are represented in Q16 fixed-point format, where a real value X is stored as the integer x = round(X · 2^16). Equation (4.4) gives the corresponding fixed-point form, sigmoid_Q16(x), which produces a Q16-scaled output directly from the Q16 input using only integer multiplication, addition, and division (implemented as a right shift where the divisor is a power of two, except where noted). The approximation is therefore computed entirely with integer comparisons, additions, multiplications, and divisions, all available in the base RV32I ISA. This formulation was used identically in both the Python training pipeline and the bare-metal C inference, ensuring that the network's behaviour during training matches its behaviour on the RISC-V core exactly.

$$\text{sigmoid}(x)=\begin{cases} 0, & X < -5 \\ 0.0304(X+5), & -5 \le X < -2.5 \\ 0.0760 + 0.1287(X+2.5), & -2.5 \le X < -1 \\ 0.2691 + 0.2310(X+1), & -1 \le X \le 1 \\ 0.7309 + 0.1287(X-1), & 1 < X \le 2.5 \\ 0.9240 + 0.0304(X-0.5), & 2.5 < X \le 5 \\ 1 & X > 5 \end{cases} \tag{4.3}$$

$$\text{sigmoid}_{Q16}(x)=\begin{cases} 0 & x < -327680 \\ 4980(x + 327680) / 163840 & -327680 \le x < -163840 \\ 4980 + 12649(x + 163840) / 98304 & -163840 \le x < -65536 \\ 17629 + 30277(x + 65536) / 131072 & -65536 \le x \le 65536 \\ 47906 + 12649(x - 65536) / 98304 & 65536 < x \le 163840 \\ 60555 + 4980(x - 163840) / 163840 & 163840 < x \le 327680 \\ 65536 & x > 327680 \end{cases} \quad (4.4)$$

Model accuracy. Since the model was trained directly on Q16 data using the custom fixed-point multiplication and piecewise sigmoid approximation throughout, there is a single accuracy figure: the model achieves 95% accuracy on the 2,000-image Q16 test set. There is no separate floating-point baseline to compare against, as the model was never trained or evaluated in floating-point.

4.5.2 Firmware Logic and Image Transmission Protocol

The inference firmware follows a request-response protocol over UART. After booting and loading the application image through the bootloader, the RISC-V core prints a prompt over UART indicating it is ready to receive an image. The host PC then sends the 32×32×3 image data (3072 bytes encoded as Q16 fixed-point values) over UART. Once the full image is received, the core runs the inference loop, printing some of intermediate hidden layer values as they are computed, which was used for detecting faulty intermediate parameters when injecting SEUs, and finally prints the classification result as either ship or cat.

For all fault injection experiments in this section, a ship image from the CIFAR-10 test set was used as the fixed test input. This means the correct classification result for every injection cycle is ship (true positive). Any output of cat is a misclassification caused by the injected fault. The intermediate values printed during inference, referred to as mid values in the data, are the accumulated dot products at each layer of the network before the final classification threshold is applied. These mid values are also useful for diagnosing fault behaviour, since a wrong mid value that does not change the final classification decision represents a case where the fault perturbed an intermediate computation but the network's inherent numerical resilience absorbed the error before the output.

4.5.3 First 102 Critical Bits: Same Address Space as Fibonacci

The first phase of the deep learning injection campaign targeted the same CRAM addresses as the Fibonacci campaign described in Section 4.4, plus two additional addresses (CB 101 and 102) that appeared in the same address region. This was done deliberately to answer the question of whether the critical bit population of the Potato RISC-V core is workload-dependent, that is, whether bits that are critical for Fibonacci are also critical for deep learning inference, and whether the failure modes are similar. Table 4.4 shows the result for this first batch.

Table 4.4 Tested EBD index ranges for the first 102 deep learning CBs, with CB density and estimated time

EBD Index Range	CBs Found	Proportion of CB / Essential Bits	Est. Time (min)	Affected Core
50k – 59k	12	0.12%	33.7	RISC2
100k – 106k	10	0.17%	23.0	RISC2
200k – 200.3k	5	0.50%	7.2	RISC2
250k – 250.8k	5	0.50%	7.2	RISC2
300k – 302k	5	0.25%	9.3	RISC2
400k – 404k	5	0.10%	15.8	RISC2
520k – 522k	5	0.17%	11.5	RISC2
630k – 632k	5	0.25%	9.3	RISC2
700k – 703k	5	0.17%	11.5	RISC2
850k – 863k	2	0.02%	30.2	RISC2
2.0M	5	1.00%	6.1	RISC2
2.1M	3	0.60%	4.1	RISC2
2.4M	5	1.00%	6.1	RISC2
2.7M	6	0.60%	8.2	RISC2
2.9M	5	1.00%	6.1	RISC2
4.0M	4	0.40%	6.2	RISC1
4.2M	5	1.00%	6.1	RISC1
4.3M	7	0.70%	9.2	RISC1
4.4M	1	0.10%	3.2	RISC1
4.5M	2	0.20%	4.2	RISC1

Table 4.5 indicates the failure mode distribution for the first phase of the results.

Table 4.5 Failure mode distribution for first 102 CBs (deep learning workload)

Failure Mode	Count	Percentage
Stops (core halts)	73	71.6%
Periodic wrong intermediate values	15	14.7%
Wrong characters / formatting	6	5.9%
Exception trap (MCAUSE)	4	3.9%
Core restarts	**4**	**3.9%**

4.5.4 Comparison with the Fibonacci Program

Of the 100 addresses shared with the Fibonacci program, 96 are present in both the Fibonacci and deep learning first-100 CB lists, confirming a 96% overlap in the critical bit population across the two workloads. This is a strong indicator that the vulnerability of this RISC-V implementation is determined primarily by its hardware structure rather than by the specific program being executed. The remaining 4% difference reflects bits associated with execution paths that are exercised differently between Fibonacci arithmetic and neural network inference.

The 8 cases with different critical bit addresses for Fibonacci and deep learning firmware, are summarized in Table 4.6.

Among the 96 common addresses, 45 show a different failure mode between the two workloads, even though the same CRAM bit is upset in both cases. Table 4.7 shows the most informative of these mode changes, specifically those where the DL workload produces an observable classification result despite the fault, while Fibonacci produced only a crash or garbled output. This shift in failure mode for the same bit might be a consequence of the different execution paths: the DL inference loop exercises different arithmetic resources and has different data flow patterns than the Fibonacci loop, so a bit that disrupts a counter or a UART formatting path in Fibonacci may instead perturb an intermediate neural network accumulator in a way that propagates through the remaining layers and still produces a (possibly incorrect) classification output.

Table 4.6 Addresses present in one workload's CB list but not the other (first 100 CBs)

CRAM Address	EBD Index	Present In	Fibonacci Effect	Deep Learning Effect
00011CC870	57,146	Deep Learning only	Not critical	Periodic wrong mid value
00011CCA50	57,242	Deep Learing only (CB102)	Not critical	Periodic wrong mid value (true)
0004A12256	4,300,056	Deep Learing only	Not critical	Stops
Conjunctive	[105190 - 105201]	Deep Learing only	Not critical	Periodic wrong mid value (true)
Conjunctive	[105160 - 105209]	Fibonacci only	Stops	Not critical
0004860097	4,000,004	Fibonacci only	Prints zeros	Not critical
000496F327	4,200,010	Fibonacci only	Misses numbers	Not critical
0004A12255	4,300,057	Fibonacci only	Wrong numbers	Not critical

Table 4.7 Selected common CBs where failure mode differs between workloads

CRAM Address	EBD Index	Fibonacci Failure	Deep Learning Failure	Deep Learning Classification
000168599B	520,057	Wrong periodic numbers	Periodic wrong mid value	True (ship)
0001685A5C	520,200	Periodic error	Periodic wrong mid value	True (ship)
0001709803	700,847	Stops	Periodic wrong mid value	True (ship)
00017B499C	850,155	Wrong numbers + format	Periodic wrong mid value	True (ship)
000312B2CF	2,700,632	Wrong numbers	Periodic wrong mid value	False (cat)
000312B2D0	2,700,631	Wrong numbers	Periodic wrong mid value	False (cat)
0001622718	401,789	Prints ","	Continuously prints "["	Stops
000312B3AF	2,700,786	Periodic error	Core restarts	No decision

4.5.5 Second Campaign Batch: Classification-Focused Injection

After completing the address-matched first phase, a second injection campaign was run targeting new CRAM regions specifically selected to find critical bits where the inference firmware continues executing and produces a classification decision rather than halting. In total, 98 additional critical bits were found in this phase, bringing the overall DL CB total to 200. All 98 CBs in the second batch affect RISC2 and produce the failure mode "periodic wrong intermediate values", the core continues running and printing inference results, but the hidden layer accumulator values are periodically incorrect.

Because the second batch was obtained by densely sampling specific CRAM regions known to be related to the DNN inference datapath rather than by sweeping contiguous index ranges, the CB density and estimated time figures for this batch would reflect only those targeted high-density windows and would not be representative of the broader essential bit population. For this reason, the density and timing analysis is limited to the first 102 CBs in Table 4.4 above, while Table 4.8 below presents the classification results across all 200 CBs.

Table 4.8 Classification result distribution across all 200 deep learning CBs

Classification Result	Count	Percentage	Interpretation
No decision — core stops or errors	93	46.5%	Permanent failure; firmware cannot complete inference
Always correct — True (ship)	66	33.0%	Fault corrupts mid values but network absorbs it; correct output
1 cycle wrong, then self-corrects	20	10.0%	Transient effect; SEM scrubbing corrects bit before next cycle
Always wrong — False (cat)	19	9.5%	Permanent misclassification; fault flips decision boundary
Some cycles wrong, some correct	2	1.0%	Intermittent effect; fault near a metastable accumulator path

The most important observation from Table 4.9 is that 46.5% of all CBs cause the firmware to stop or generate an exception, producing no classification result at all. A further 33% of CBs corrupt intermediate computation values but do not change the final output — the network classifies the ship image correctly despite the fault. This resilience is a property of the neural network architecture: a single corrupted accumulator value in a hidden layer often has insufficient influence on the final classification score to flip the decision, particularly when the correct class has a large score margin over the incorrect class.

Only 9.5% of CBs cause a permanent misclassification, always outputting cat for the ship input. These are the most dangerous from a system safety perspective, as the fault produces a silent data corruption: the firmware continues running and providing outputs that appear normal, but the outputs are consistently wrong. The 10% of CBs that produce a single wrong inference cycle followed by a return to correct output suggest that the affected configuration bits influence only a transient portion of the computation, where the fault alters the result of one inference pass but does not permanently corrupt the logic driving subsequent cycles. The remaining majority of CBs either produce no misclassification at all or cause intermittent errors, suggesting that the classifier's output is relatively robust to isolated single-bit configuration faults, with most upsets affecting logic that does not deterministically drive the final binary decision. However, these observations are drawn from a partial campaign covering a small fraction of the full EBD address space; a confident reliability assessment of the complete design would require exhaustive injection across all essential bits.

4.5.6 Conjunctive Critical Bits — All 200 Deep Learning CBs

Table 4.9 Single-bit vs. conjunctive CB distribution across all 200 deep learning CBs

CB Type	Count (out of 200)	Percentage	Batches Affected
Single-bit (non-conjunctive)	176	88%	Both
Conjunctive — Batch 1 (stops)	7	3.5%	Batch 1 only
Conjunctive — Batch 2 (wrong mid values)	17	8.5%	Batch 2 only
Total conjunctive	24	12%	Both

Across all 200 deep learning CBs, 24 are conjunctive, 12% of the total population, compared to 8% in the Fibonacci campaign (6 out of 100). The increase is almost entirely driven by the second batch: only 7 of the 24 conjunctive CBs come from the first 102, while the remaining 17 come from the 98 CBs in the second batch. This higher conjunctive density in the second batch is a direct consequence of where those CBs are located in the configuration memory. The second batch targeted CRAM regions associated with the neural network weight storage and the inference datapath. In these regions, flipping either bit of a conjunctive pair alone does not trigger any detectable malfunction, the output of both cores remains identical and no mismatch interrupt is raised. A failure only becomes observable when both bits are simultaneously upset. For example, one of the identified conjunctive pairs involves CB109 (CRAM address 0x0001261A2F, EBD index 105,304) and the bit at EBD index 105,300: flipping either bit alone produced no detectable failure, while flipping both bits simultaneously caused periodic wrong intermediate values in the inference computation, confirming that the two bit flips must occur for a configuration memory to produce an observable effect on the design output.

The failure mode also differs clearly between the two batches. All 7 conjunctive CBs in Batch 1 cause the processor to stop, no inference output is produced and there is no classification result. In contrast, all 17 conjunctive CBs in Batch 2 produce the failure mode "periodic wrong intermediate values", meaning the firmware continues executing and produces a classification decision despite the fault. This behavioural difference suggests that the two batches' conjunctive pairs affect fundamentally different aspects of the processor's operation, though a precise characterisation would require cross-referencing the CRAM addresses against Vivado's placement data, which is beyond the scope of the present campaign.

The practical implication is that a significant fraction of the conjunctive CBs in the second batch are theoretically more dangerous from a system reliability standpoint than the single-bit CBs that cause the processor to stop. A stopped processor is immediately detectable. A conjunctive upset that corrupts a hidden layer accumulator but still produces a classification

output, whether correct or incorrect, is a silent fault that may go undetected unless an independent reference output is available for comparison. However, the probability of two independent SEUs simultaneously hitting both bits of a specific conjunctive pair is extremely low in practice. The conjunctive fault scenario therefore represents a theoretical worst case rather than a practically dominant failure mode for this design.

4.6 Summary

This chapter presented the application of the automated fault injection framework to two RISC-V32I soft-processor workloads running in parallel on the Zynq UltraScale+ platform: a Fibonacci sequence generator and a fixed-point Q16 binary image classifier. The two-core parallel architecture and the UART-based mismatch detection methodology were described, along with the toolchain setup and firmware deployment process for the Potato RV32I core.

For the Fibonacci workload, 100 critical bits were identified across 20 arbitrarily sampled EBD index ranges, covering a total of 43,448 injections out of 4,944,721 essential bits. The dominant failure mode was processor halt (62%), followed by wrong output values (16%), periodic wrong characters (14%), wrong characters (7%), and trap exceptions (1%). The critical bit density within the sampled regions was 0.23%, with the highest densities observed near EBD indices 4,200,000 and 4,000,000, suggesting a non-uniform spatial distribution of sensitive configuration bits across the design's CRAM footprint.

For the deep learning workload, 200 critical bits were identified across two injection batches. The observed failure modes at the classification level were: no decision produced (46.5%), always correct output (33%), self-correcting single wrong cycle (10%), always wrong classification (9.5%), and intermittent errors (1%). Conjunctive critical bits accounted for 12% of the total, with the conjunctive fraction significantly higher in the second batch than the first. While conjunctive faults that produce silent misclassification are theoretically more dangerous than halting faults, their practical probability of occurrence under real SEU conditions is extremely low given the requirement for two independent bit flips to simultaneously hit a specific pair.

Across both workloads, a 96% overlap was observed between the critical bit addresses of the Fibonacci and deep learning campaigns, confirming that the sensitive CRAM regions are predominantly determined by the processor core's fixed logic rather than by the specific application firmware being executed.

CONCLUSION

Summary

This thesis addressed the problem of systematic and automated Single Event Upset reliability assessment for SRAM-based FPGA designs, with a focus on the Zynq UltraScale+ MPSoC platform and the Xilinx Soft Error Mitigation IP core. The work spanned three areas: background consolidation and literature positioning, framework development and proof-of-concept validation, and processor-level reliability characterization under two distinct workloads.

Chapter 2 established the physical and engineering foundations of SEU reliability in SRAM-based FPGAs. The chapter reviewed the mechanisms by which high-energy particles induce bit flips in configuration memory, the distinction between essential and critical bits, and the role of the SEM IP as both a correction and injection engine. A systematic review of published fault injection methodologies, covering ICAP-based, JTAG-based, DPR-based, HDL simulation, and SEM IP-based approaches, revealed that no prior work had used the SEM IP injection interface as the primary engine for a fully automated, closed-loop reliability characterization campaign on the Zynq UltraScale+ platform. This gap, combining platform novelty with the absence of real-time output monitoring in existing SEM IP frameworks, defined the motivation for the work.

Chapter 3 presented the design and implementation of the automated fault injection framework. A Python script running on a host PC drives the SEM IP over two UART interfaces: one for injection commands and SEM IP feedback, and one for the ICAP/PCAP initialization handshake managed by a bare-metal application on the ARM Cortex-A53. The script parses the Vivado-generated Essential Bits Database file and iterates through essential bit addresses at a rate of approximately 130 ms per injection. The key innovation relative to prior SEM IP-based works is the closed-loop monitoring channel: the DUT output is observed in real time after each injection, and the campaign halts automatically when a mismatch is detected between parallel module instances, logging the responsible CRAM address as a critical bit. Readback verification through Vivado confirmed that each injection physically flips the targeted bit in the configuration memory. The framework was validated

on the ITC'99 b01 FSM benchmark using three parallel instances driven by a shared cLFSR. One hundred critical bits were identified and classified into three signature types based on which output bits were corrupted. Six conjunctive critical bits were also observed, confirming that single-SEU campaigns miss a real fraction of the vulnerable bit population due to LUT don't-care masking.

Chapter 4 applied the framework to the Potato RV32I soft-core processor under two workloads. For the Fibonacci workload, 100 critical bits were identified across sampled EBD index ranges spanning a design with 4,944,721 total essential bits. The CB density was found to be highly non-uniform, ranging from 0.01% in sparse regions to 41.66% in concentrated windows near the RISC1 placement region, with core halt being the dominant failure mode at 62%. For the deep learning workload, a Q16 fixed-point binary image classifier trained on CIFAR-10 was deployed as bare-metal C on the same Potato core. The first 102 CBs targeted the same CRAM addresses as the Fibonacci campaign, revealing a 96% overlap in the critical bit population between the two workloads, indicating that the processor's vulnerability is primarily determined by its hardware structure rather than the software it executes. A second injection batch targeting the DNN inference datapath produced 98 additional CBs. Across all 200 DL CBs, 46.5% caused the firmware to halt, 33% corrupted intermediate values without affecting the final classification result, 9.5% produced permanent silent misclassification, and 10% showed transient errors consistent with SEM scrubbing correction within a single inference cycle. The 9.5% silent misclassification rate is of particular concern for safety-critical deployments, as these failures produce outputs that appear normal but are incorrect. The conjunctive CB rate in the DL campaign was 12%, double that of the Fibonacci campaign, driven by the higher density of symmetric LUT functions in the DNN arithmetic CRAM region.

Future Work

Several directions follow naturally from this work. The most immediate is an exhaustive sweep of all 4,944,721 essential bits in the RISC-V design to produce a definitive critical bit fraction, rather than the partial campaign presented here. An adaptive injection strategy, one

that uses a coarse initial sweep to identify high-density CRAM windows and focuses subsequent effort on those regions, would substantially reduce the time required for this exhaustive characterization, and is directly motivated by the non-uniform CB density observed in Chapter 3.

On the protection side, applying Triple Modular Redundancy to the Potato core implementation and re-running the injection campaign would provide quantitative validation of the TMR benefit on this specific platform and workload combination. The 96% CB overlap between the Fibonacci and deep learning campaigns suggests that processor-level TMR would be similarly effective for both workloads, though the 4% divergence means workload-specific selective hardening could offer additional gains in the DNN domain.

Finally, the framework currently requires manual intervention when the SEM IP encounters an uncorrectable error or critical bit, halting the injection campaign until the board is manually reset and reinitialized. A natural extension would be to integrate Dynamic Partial Reconfiguration to automatically reset and reinitialize the SEM IP when this condition is detected, without requiring a full device reprogram or user intervention. This would eliminate the last remaining manual step in the workflow and make the framework truly fully automated from start to finish, capable of running unattended injection campaigns of arbitrary length across the entire essential bit population.

BIBLIOGRAPHY

AMD Xilinx. 2017. "Essential Bits for UltraScale FPGAs." Application Note XAPP1246, v1.0.

AMD Xilinx. 2022. "Soft Error Mitigation Controller v4.1 LogiCORE IP Product Guide (PG187)." v4.1.

AMD Xilinx. 2023. "UltraScale Architecture Configuration User Guide (UG570)." v1.15.

AMD Xilinx. 2023. "Vivado Design Suite User Guide: Programming and Debugging (UG908)." v2023.2.

AMD Xilinx. 2024. "How to Use the SEM IP Error Report to Look Up Essential Bit Error Locations Using the EBD File." AMD Adaptive Support, Article 70684.

Aranda, L.A., Sanchez-Macian, A. and Maestro, J.A. 2019. "ACME: A Tool to Improve Configuration Memory Fault Injection in SRAM-Based FPGAs." IEEE Access, vol. 7, p. 128153-128161. DOI: 10.1109/ACCESS.2019.2939858.

Aranda, L.A., Wessman, N.-J., Santos, L., Sanchez-Macian, A., Andersson, J., Weigand, R. and Maestro, J.A. 2020. "Analysis of the Critical Bits of a RISC-V Processor Implemented in an SRAM-Based FPGA for Space Applications." Electronics, vol. 9, no. 1, p. 175. DOI: 10.3390/electronics9010175.

Aranda, L.A., Ruano, O., Garcia-Herrero, F. and Maestro, J.A. 2022. "ACME-2: Improving the Extraction of Essential Bits in Xilinx SRAM-Based FPGAs." IEEE Transactions on Circuits and Systems II, vol. 69, no. 3, p. 1577-1581. DOI: 10.1109/TCSII.2021.3105558.

J. Ayer Jr. 2011. “Dual Use of ICAP with SEM Controller.” Application Note XAPP517, v1.0. Xilinx, Inc.

Azambuja, J.R., Altieri, M., Becker, J. and Kastensmidt, F.L. 2013. "Heuristics for Temporal and Spatial Allocation for Checkpoint Distribution in Fault-Tolerant Embedded Systems." Journal of Electronic Testing, vol. 29, no. 1, p. 149-161.

Baumann, R.C. 2005. "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies." IEEE Transactions on Device and Materials Reliability, vol. 5, no. 3, p. 305-316. DOI: 10.1109/TDMR.2005.853449.

Berg, M. et al. 2008. "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA." IEEE Transactions on Nuclear Science, vol. 55, no. 4, p. 2259-2266. DOI: 10.1109/TNS.2008.2001422.

Caffrey, M., Graham, P., Johnson, E., Wirthlin, M. and Rollins, N. 2002. "Single-Event Upsets in SRAM FPGAs." In Military and Aerospace Programmable Logic Devices (MAPLD).

Carmichael, C. and Tseng, C.W. 2009. "Correcting Single-Event Upsets in Virtex-4 FPGA Configuration Memory." Xilinx Application Note XAPP1298, v1.0.

Cherezova, N., Jenihhin, M. and Jutman, A. 2025. "Fault Injection Tool for FPGA Reliability Testing: A Novel Method and Discovery of LUT-Specific Logical Redundancies." Electronics, vol. 14, no. 23, p. 4600. DOI: 10.3390/electronics14234600.

DA-FIS: A High-Speed Dynamic Adaptive Fault Injection Server Framework for Reliable FPGA-Based Embedded Systems. 2025. PeerJ Computer Science. DOI: 10.7717/peerj-cs.2996.

Dodd, P.E. and Massengill, L.W. 2003. "Basic Mechanisms and Modeling of Single-Event Upset in Digital Microelectronics." IEEE Transactions on Nuclear Science, vol. 50, no. 3, p. 583-602. DOI: 10.1109/TNS.2003.813129.

Ferlini, F., Viel, F., Seman, L.O., Pettenghi, H., Bezerra, E.A. and Leithardt, V.R.Q. 2023. "A Methodology for Accelerating FPGA Fault Injection Campaign Using ICAP." Electronics, vol. 12, no. 4, p. 807. DOI: 10.3390/electronics12040807.

Foucard, G. et al. 2010. "Investigation on the SEU Sensitivity of a SRAM-Based FPGA's Configuration Cells." In Proceedings of RADECS, p. 266-272.

FREtZ: FPGA Reliability Evaluation through JTAG. 2019. Open-source framework, University of Piraeus. In GitHub: https://github.com/unipieslab/FREtZ

Ghaffari, F., Sahraoui, F., Benkhelifa, M.E.A., Granado, B., Kacou, M.A. and Romain, O. 2014. "Fast SRAM-FPGA Fault Injection Platform Based on Dynamic Partial Reconfiguration." In Proceedings of the 26th International Conference on Microelectronics (ICM), p. 144-147. DOI: 10.1109/ICM.2014.7071827.

Höller, A., Macher, G., Rauter, T., Iber, J. and Kreiner, C. 2015. "A Virtual Fault Injection Framework for Reliability-Aware Software Development." In Proceedings of the IEEE/IFIP DSN Workshops (DSN-W), p. 69-74. DOI: 10.1109/DSN-W.2015.16.

D.P. Kingma and J. Ba. 2015. “Adam: A Method for Stochastic Optimization.” In Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA.

A. Krizhevsky, V. Nair and G. Hinton. 2009. “CIFAR-10 (Canadian Institute for Advanced Research).” Technical Report, University of Toronto.

M. Laouej. 2021. Amélioration des méthodes d’émulation des pannes par les radiations cosmiques sur les FPGA. Mémoire de maîtrise, École de technologie supérieure, Montréal, QC, Canada.

Li, G. et al. 2017. "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications." In Proceedings of SC, p. 1-12. DOI: 10.1145/3126908.3126964.

J. Lim, J. Son and H. Yoo. 2024. “Efficient Processing-in-Memory System Based on RISC-V Instruction Set Architecture.” Electronics, vol. 13, no. 15, p. 2971. DOI: 10.3390/electronics13152971. DOI: 10.3390/electronics13152971.

Liu, Y., Yao, L. and Zhang, W. 2025. "SEU Fault Injection Strategy for SRAM-Based FPGA User Memory Based on Dual-Circuit Model." Journal of Electronic Testing, vol. 41, p. 75-84. DOI: 10.1007/s10836-025-06162-w.

Lyu, H., Wang, B., Mei, B., Mo, R., Gao, W., Xue, P. and Yuan, Q. 2025. "A Fault Injection Method for Evaluating Single Event Soft Errors in Deep Neural Networks." In Proceedings of the 6th International Conference on Reliability Engineering and Electronic Design (ICREED), p. 1-6. DOI: 10.1109/ICREED65908.2025.11036247.

May, T.C. and Woods, M.H. 1979. "Alpha-Particle-Induced Soft Errors in Dynamic Memories." IEEE Transactions on Electron Devices, vol. 26, no. 1, p. 2-9. DOI: 10.1109/T-ED.1979.19370.

Normand, E. 1996. "Single Event Effects in Avionics." IEEE Transactions on Nuclear Science, vol. 43, no. 2, p. 461-474. DOI: 10.1109/23.490893.

Normand, E. 1996. "Single Event Upset at Ground Level." IEEE Transactions on Nuclear Science, vol. 43, no. 6, p. 2742-2750. DOI: 10.1109/23.556861.

Oliveira, A.B. et al. 2020. "Evaluating Soft Core RISC-V Processor in SRAM-Based FPGA Under Radiation Effects." IEEE Transactions on Nuclear Science, vol. 67, no. 7, p. 1503-1510. DOI: 10.1109/TNS.2020.2995729.

Papadimitriou, G. and Gizopoulos, D. 2023. "Silent Data Corruptions: Microarchitectural Perspectives." IEEE Transactions on Computers, vol. 72, no. 11, p. 3155-3167. DOI: 10.1109/TC.2023.3285094.

Quinn, H. et al. 2015. "Using Benchmarks for Radiation Testing of Microprocessors and FPGAs." IEEE Transactions on Nuclear Science, vol. 62, no. 6, p. 2547-2554. DOI: 10.1109/TNS.2015.2498313.

Rajkumar, T. and Aberg, J. 2024. "Guided Fault Injection Strategy for Rapid Critical Bit Detection in Radiation-Prone SRAM-FPGA." In Proceedings of Design, Automation and Test in Europe Conference (DATE). DOI: 10.23919/DATE58400.2024.10546801.

Reagen, B. et al. 2016. "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators." In Proceedings of ISCA, p. 267-278. DOI: 10.1109/ISCA.2016.32.

J. Royer. 2024. cLFSR: Configurable Linear Feedback Shift Register. Unpublished VHDL module, Laboratoire de Conception en Microélectronique (LaCIME), École de technologie supérieure, Montréal, QC, Canada.

Ruano, O., Garcia-Herrero, F., Aranda, L.A., Sanchez-Macian, A., Rodriguez, L. and Maestro, J.A. 2021. "Fault Injection Emulation for Systems in FPGAs: Tools, Techniques and Methodology, a Tutorial." Sensors, vol. 21, no. 4, p. 1392. DOI: 10.3390/s21041392.

Siecha, R.T., Alemu, G., Prinzie, J. and Leroux, P. 2025. "Analysis of the SEU Tolerance of an FPGA-Based Time-to-Digital Converter Using Emulation-Based Fault Injection." Electronics, vol. 14, no. 11, p. 2176. DOI: 10.3390/electronics14112176.

Smit, T.T., Forlin, B.E., Chen, K.-H., Souvatzoglou, I., Psarakis, M. and Ottavi, M. 2024. "An Enhanced Fault Injection Framework for FPGA-Based Soft-Cores." In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), p. 1-6. DOI: 10.1109/DFT63277.2024.10753564.

A. Souari, C. Thibeault, Y. Blaquière and R. Velazco. 2015. “Optimization of SEU Emulation on SRAM FPGAs Based on Sensitiveness Analysis.” In Proceedings of the IEEE 21st International On-Line Testing Symposium (IOLTS), Halkidiki, Greece, p. 36–39. DOI: 10.1109/IOLTS.2015.7229827.

Tuzov, I., de Andres, D., Ruiz, J.-C. and Hernandez, C. 2023. "BAFFI: A Bit-Accurate Fault Injector for Improved Dependability Assessment of FPGA Prototypes." In Proceedings of Design, Automation and Test in Europe Conference (DATE), p. 1-6. DOI: 10.23919/DATE56975.2023.10137300.

Wilson, A.E. and Wirthlin, M. 2021. "Fault Injection of TMR Open Source RISC-V Processors Using Dynamic Partial Reconfiguration on SRAM-Based FPGAs." In Proceedings of the 2021 IEEE Space Computing Conference (SCC), p. 1-8. DOI: 10.1109/SCC49971.2021.00008.

Wirthlin, M. et al. 2014. "Estimating TMR Reliability on FPGAs Using Markov Models." In Proceedings of the International Conference on Field Programmable Logic and Applications, p. 1-4.

Wirthlin, M. 2015. "High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond." Proceedings of the IEEE, vol. 103, no. 3, p. 379-389. DOI: 10.1109/JPROC.2015.2404212.

Zhang, R., Xiao, L., Li, J., Cao, X. and Qi, C. 2018. "A Fault Injection Platform Supporting Both SEU and Multiple SEUs for SRAM-Based FPGA." IEEE Transactions on Device and Materials Reliability, vol. 18, no. 4, p. 599-605. DOI: 10.1109/TDMR.2018.2874091.

Ziegler, J.F. and Lanford, W.A. 1979. "Effect of Cosmic Rays on Computer Memories." Science, vol. 206, no. 4420, p. 776-788. DOI: 10.1126/science.206.4420.776.