Automated Reliability Optimization for SRAM-based FPGAs

DISSERTATION
zur Erlangung des Grades eines Doktors
der Ingenieurwissenschaften

vorgelegt von
Christian Fibich, MSc

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen
Siegen 2026

Betreuer und erster Gutachter
Prof. Dr.-Ing. habil. Roman Obermaisser
Universität Siegen

Zweiter Gutachter
Prof. Roberto Passerone
Universität Trient – Università degli Studi di Trento, Italien

Tag der mündlichen Prüfung
3. Juli 2026

Mitglieder der Promotionskommission:
Prof. Dr.-Ing. habil. Roman Obermaisser
Prof. Roberto Passerone
Prof. Dr. Bhaskar Choubey
Prof. Dr.-Ing. Kai Hahn

Vorsitzender: Prof. Dr. Bhaskar Choubey

Automated Reliability Optimization for SRAM-based FPGAs

DISSERTATION
to obtain the degree of Doctor
of Engineering

submitted by Christian Fibich, MSc

submitted to the School of Science and Technology
of the University of Siegen
Siegen 2026

Acknowledgments

I would like to express my deepest gratitude to my advisor, Prof. Dr.-Ing. Roman Obermaisser, for his unwavering support, professional guidance, and invaluable feedback throughout this journey. His expertise and encouragement were instrumental in helping me navigate the challenges of this research endeavor, which is documented in this dissertation.

I am also profoundly thankful to FH-Prof. DI Dr. Martin Horauer at the University of Applied Sciences Technikum Wien for his continued support and constructive feedback, which greatly enriched my work.

I wish to express my sincere appreciation to Prof. Roberto Passerone, the external reviewer of this dissertation, and to the members of the examination committee for their time and consideration.

My heartfelt thanks go to my current and former colleagues at the Department of Electronic Engineering & Entrepreneurship at the University of Applied Sciences Technikum Wien. Our engaging discussions often opened new perspectives, sparked ideas, and provided fellowship during the highs and lows of this dissertation project.

A venture such as this does not stay neatly contained temporally or spatially within academic boundaries. I am deeply grateful to my dear partner Aenna, my friend Harald, and my parents Maria and Felix, for their tireless support, understanding, and encouragement from the first day on. Completing this work would have been impossible without you. I also extend my gratitude to all my friends and family members not explicitly named here for their patience and support throughout this journey.

This work has been partially supported by the Austrian Federal Ministry for Digital and Economic Affairs (BM:DW) and the National Foundation for Research, Technology and Development via the projects “Josef Ressel Center for Innovative Platforms for Electronic-Based Systems (INES)” and “Josef Ressel Center for Verification of Embedded Computing Systems (VECS)” managed by the Christian Doppler Society (CDG). Furthermore, this work has been partially supported by the Municipal Department MA23 for Economic Affairs, Labour and Statistics of the City of Vienna via the project “Stadt Wien Kompetenzteam für Drohnentechnik in der Fachhochschulausbildung” (MA23 35-02). I am also grateful for the support received by the Faculty Electronic Engineering & Entrepreneurship, the Department Electronic Engineering & Entrepreneurship, and the Research Group Embedded Systems at the University of Applied Sciences Technikum Wien.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer, nicht angegebener Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt. Es wurden keine Dienste eines Promotionsvermittlungsinstituts oder einer ähnlichen Organisation in Anspruch genommen.

Affidavit

I hereby declare in lieu of an oath that I have drawn up the present work without any undue assistance by third parties and without using any aids other than the ones specified. The data and concepts taken, either directly or indirectly, from any other sources have been marked, indicating the source. The work has not been submitted to any other examining authority neither in Germany nor abroad and neither in the same nor in any similar form. Use of the services of any PhD mediation institute or of any similar organisation has not been made.

Abstract

Today's most densely integrated Field Programmable Gate Arrays (FPGAs) store their configuration in integrated volatile SRAM cells and thereby offer virtually limitless reconfigurability. However, randomly occurring bit flips in an SRAM-based FPGA's configuration memory can alter the implemented circuit. Furthermore, bit flips in user-defined storage elements such as flip-flops and block memory can corrupt the design's state. Both effects may compromise system functionality, making SRAM-based FPGAs especially vulnerable to bit flips.

As a mitigation measure, on-chip spatial redundancy is often introduced. This, however, increases FPGA resource usage and thereby system cost. Selectively protecting the design parts that are most vulnerable to bit flips allows to balance cost and reliability. This requires identifying especially vulnerable parts of the design to co-optimize a design's bit-flip resilience with system cost.

Existing reliability prediction approaches for FPGA designs often focus on logic resources, overlooking the significant and harder-to-predict effects of interconnect bit flips. Additionally, they do not account for process, voltage, and temperature variations on the outcome of configuration memory bit flips. These gaps in fault modeling limit current design-space exploration methods, preventing them from evaluating the impact of fault mitigation measures on bit-level reliability. This dissertation addresses these challenges with the following contributions.

1. This work provides an in-depth analysis of configuration memory bit flips in SRAM-based FPGAs. Based on this analysis, it develops and validates criticality prediction models to classify bitstream bits as benign or malignant when flipped. This reduced the need for time-consuming fault injection experiments by 60% to 98% with minimal false negatives in four case studies. In this course, this dissertation provides the first systematic study of how chip temperature and operating voltage influence configuration-level fault injection outcomes in FPGAs, which revealed significant inter- and intra-device variations in bit flip effects.

2. Based on the fault modeling work discussed above, an automated module-based design-space exploration approach for partial redundancy insertion is proposed. It co-optimizes FPGA resource usage and vulnerability to single-bit flips by encoding the module selection problem for genetic algorithms. In this course, it defines fitness metrics based on cost and vulnerability. Evaluated on six case-study designs, the approach generates Pareto-optimal partially redundant design variants, improving the cost-effectiveness of solutions by 13 to 59 percentage points compared to random starting sets. This method provides a diverse range of design variants along the cost-reliability trade-off. This enables the designer to select a variant that best fulfills the application's, mission's, or mission phase's requirements.

3. A profiling approach is proposed for FPGA designs generated by high-level synthesis. The approach aims at identifying critical user-defined storage elements by profiling the design's memory access patterns at the high-level language or intermediate representation level. Fault injection evaluations of case-study designs based on well-known benchmarks revealed a potential link between this memory access behavior and the in-hardware vulnerability of storage elements. This enables cost-effective design variants with partial bit-flip protection for user-defined storage elements. Furthermore, this high-level analysis may guide the design-space exploration process mentioned above, accelerating convergence.

In summary, this dissertation contributes multiple methods for automating reliability improvements for SRAM-based FPGA designs. These methods are based on empirical analyses of FPGA vulnerabilities and address effects previously overlooked in existing research.

Zusammenfassung

Field Programmable Gate Arrays (FPGAs), deren Konfigurationsspeicher auf Statischem RAM (SRAM) basiert, sind heutzutage in vielen Anwendungen verbreitet. Flüchtige Speicher wie SRAM oder Flip-Flops sind jedoch anfällig für zufällig auftretende Bitfehler, welche zur Umkehrung des gespeicherten Wertes führen können. In SRAM-basierten FPGAs können sowohl der Konfigurationsspeicher, der die Schaltungsstruktur enthält, als auch der Datenspeicher, in dem der Schaltungszustand abgelegt ist, von Bitfehlern betroffen sein. Da diese Fehler zum Fehlverhalten der im FPGA realisierten Schaltung führen können, ist ihre Behandlung in sicherheitskritischen Systemen essenziell. Die Beiträge dieser Disseration befassen sich mit diesem Thema.

Zunächst (1) stellt die Arbeit ein Verfahren zur statischen Zuverlässigkeitsabschätzung für den Konfigurationsspeicher von SRAM-basierten FPGAs vor. Dieses Verfahren bewertet die Auswirkungen von einzelnen Bitfehlern im Konfigurationsspeicher auf die Zuverlässigkeit der Schaltung. Unterstützt wird dies durch eine Methode zur empirischen Charakterisierung des Fehlerverhaltens der Ziel-FPGA-Technologie. Eine Implementierung des Verfahrens wurde durch Fehlerinjektionsexperimente an vier Fallstudien-Designs für FPGAs der *7 Series* von AMD erfolgreich validiert. Im Vergleich zum Herstellerwerkzeug konnte die Anzahl der durch Fehlerinjektion zu prüfenden Konfigurationsbits je nach Design um 60% bis 98% reduziert werden. Zudem wurde erstmals die Auswirkung von Process-Voltage-Temperature (PVT)-Effekten auf Fehlerinjektionsergebnisse systematisch untersucht, wobei insbesondere bei FPGAs der *7 Series* von AMD eine signifikante Abhängigkeit von Temperatur und Betriebsspannung festgestellt wurde.

Aufbauend auf der statischen Zuverlässigkeitsabschätzung für Konfigurationsbits präsentiert die Arbeit (2) ein Verfahren für die ressourceneffiziente Steigerung der Zuverlässigkeit von SRAM-basierten FPGA-Designs. Hierbei handelt es sich um ein „Design Space Exploration (DSE)"-Verfahren, welches den Ressourcenverbrauch von teilweise dreifachredundanten Schaltungsvarianten mit deren Zuverlässigkeit unter Konfigurations-Einzelbitfehlern kooptimiert. Das Verfahren erzeugt Pareto-optimale Schaltungsvarianten des Original-Designs in Bezug auf diese Ziele. Zu seiner Validierung wurde das vorgestellte Verfahren für FPGAs der *7 Series* von AMD implementiert und auf sechs Fallstudien-Designs angewandt. Dabei konnte die Ressourceneffizienz der Fehlertoleranzmechanismen zwischen 13 und 59 Prozentpunkten gesteigert werden, gemessen als Vergrößerung des durch die Pareto-Front nach 16 Generationen abgedeckten Hypervolumens im Vergleich zur zufälligen Anfangsgeneration. Auch hierbei wurden PVT-Effekte berücksichtigt.

Ergänzend zur Fehlertoleranz in Bezug auf den Konfigurationsspeicher SRAM-basierter FPGAs stellt die Arbeit (3) ein Verfahren zur Analyse der Zuverlässigkeit bezogen auf den Datenspeicher dieser Bausteine vor. Dieses Verfahren zielt auf Designs ab, welche mittels High-Level Synthese (HLS) erstellt werden. Durch die Verfolgung von Speicherzugriffen in Exekutionen des Designs auf hoher Abstraktionsebene wird die Zuverlässigkeitsrelevanz einzelner Datenspeicher-Elemente vorhergesagt. Das Verfahren wurde durch Fehlerinjektionsexperimente an verschiedenen Designs evaluiert. Die Ergebnisse deuten auf einen möglichen Zusammenhang zwischen dem zeitlichen Verhältnis der Speicherzugriffe und der Zuverlässigkeitsrelevanz der Speicherelemente hin. Überlegungen zur Verbesserung der Treffsicherheit des Verfahrens werden diskutiert.

Zusammenfassend leistet diese Dissertation Beiträge zur Weiterentwicklung von Analyseverfahren und zur automatisierten, ressourceneffizienten Implementierung von Fehlertoleranzmechanismen in SRAM-basierten FPGAs. Die vorgestellten Methoden werden durch empirische Evaluierungen der Zuverlässigkeit von SRAM-basierten FPGAs gestützt, wobei auch Effekte berücksichtigt wurden, denen in der bisherigen Forschung kaum Aufmerksamkeit geschenkt wurde.

Publications Arising from this Work

Parts of this dissertation have been published in the following journals and proceedings.

FPGA Configuration Fault Modeling and Effects of Process, Voltage and Temperature (PVT) Variations on Fault Injection Results

[1] C. Fibich, M. Horauer, and R. Obermaisser, "Device- and Temperature Dependency of Systematic Fault Injection Results in Artix-7 and iCE40 FPGAs," *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2021, pp. 1600–1605, https://doi.org/10.23919/DATE51398.2021.9474161.

[2] C. Fibich, M. Horauer, and R. Obermaisser, "Bitstream-Level Interconnect Fault Characterization for SRAM-based FPGAs," *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 2023, pp. 1–2, https://doi.org/10.23919/DATE56975.2023.10136911.

[3] C. Fibich, M. Horauer, and R. Obermaisser, "Characterization of Interconnect Fault Effects in SRAM-based FPGAs," *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, Tallinn, Estonia, 2023, pp. 65–68, https://doi.org/10.1109/DDECS57882.2023.10139343.

[4] C. Fibich and L. Lovric, "Leveraging Open-Source Bitstream Documentation for FPGA Soft-Error Robustness Analysis", *2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS)*, Ischia, Italy, 2025, pp. 1–7, https://doi.org/10.1109/IOLTS65288.2025.11116854.

Bitstream-Level Cost-Reliability Design-Space Exploration (DSE)

[5] C. Fibich, M. Horauer, and R. Obermaisser, "Automated Bitstream-Level Cost-Reliability Design-Space Exploration for SRAM-Based FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 45, no. 1, pp. 218–231, Jan. 2026, https://doi.org/10.1109/TCAD.2025.3573225.

Single Event Upset (SEU) Vulnerability Analysis of User-Defined Storage Elements in High-Level Synthesis (HLS)

[6] C. Fibich, M. Horauer, and R. Obermaisser, "HLShield: a reliability enhancement framework for high-level synthesis," *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*, Toulouse, France, 2017, pp. 1–10, https://doi.org/10.1109/SIES.2017.7993378.

[7] C. Fibich, M. Horauer, and R. Obermaisser, "Vulnerability analysis of storage elements in HLS-generated designs using high-level profiling," *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, Milan, Italy, 2017, pp. 190–194, https://doi.org/10.1109/ICSRS.2017.8272819.

[8] C. Fibich, M. Horauer, and R. Obermaisser, "Reliability-Enhanced High-Level Synthesis using Memory Profiling and Fault Injection," *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, Vancouver, BC, Canada, 2019, pp. 1363–1370, https://doi.org/10.1109/ISIE.2019.8781158.

Contents

List of Abbreviations **xv**

1 Introduction **1**

2 Background **5**

2.1 Soft Errors in Digital Storage Elements 5

2.1.1 Dynamic RAM 7

2.1.2 Static RAM 8

2.1.3 Electrically Erasable PROM 9

2.1.4 Emerging Memory Technologies 11

2.2 FPGA Technology and Applications 12

2.3 FPGA Reliability 16

2.4 Reliability Enhancement Techniques for FPGAs 19

3 Literature Review **23**

3.1 Requirements & Challenges 23

3.1.1 Tool-Level Requirements 24

3.1.2 Configuration-Bit-Level Requirements 27

3.2 Bit-Level Fault Modeling and Prediction 30

3.2.1 STAR 30

3.2.2 SEVAX 33

3.2.3 Rad-SyMPaR/VERI-Place 35

3.2.4 PyXEL 36

3.2.5 ACME and ACME-2 38

3.2.6 BAFFI 39

3.2.7 BFAT 41

3.2.8 Bricas et al. 44

3.2.9 Other Related Work 46

3.2.10 Summary 50

3.3 Cost-Aware FPGA Reliability Improvement Approaches 54

3.3.1 Reliability-aware FPGA Placement and Routing 55

3.3.2 Selective Redundancy Insertion 57

3.3.3 Redundancy Optimization and Design-Space-Exploration 62
3.3.4 High-Level Synthesis-based Reliability Improvement Approaches 69
3.3.5 Summary 74
3.4 Motivation of the Research Gap for this Dissertation 77

4 Bitstream-Level Fault Modeling and Criticality Analysis 81
4.1 Low-Level Bitstream Parsing and Design Analysis 85
4.1.1 Bitstream Decoding and Analysis 86
4.1.2 Extraction of Placement and Routing Information 90
4.2 Static Bitstream-Level Fault Analysis 94
4.2.1 Modeling Faults in Look-Up Tables 94
4.2.2 Modeling Interconnect Faults 96
4.2.3 Modeling Faults in Other Resources 115
4.3 Characterization of Interconnect Fault Consequences 117
4.3.1 Experimental Setup 118
4.3.2 Results 129
4.3.3 Summary 142
4.4 Analysis of PVT Influence on Fault Injection Results 143
4.4.1 Temperature Effects on Systematic Fault Injection Results 144
4.4.2 Temperature Influence on Open Faults in AMD 7 Series FPGAs 154
4.4.3 Supply Voltage & Inter/Intra-Chip Variation in AMD 7 Series FPGAs 158
4.4.4 Summary 164
4.5 Static Criticality Prediction for AMD 7 Series FPGAs 165
4.5.1 Validation Experiments 168
4.5.2 Results & Discussion 171
4.5.3 Summary 175
4.6 Bitstream Patching for AMD 7 Series FPGAs 176
4.6.1 Validation Experiments 179
4.6.2 Results & Discussion 180
4.6.3 Summary 183
4.7 Conclusion 184

5 Design-Space-Exploration for FPGA Reliability Optimization 187
5.1 Proposed Design Space Exploration Approach 188
5.1.1 Genetic Encoding of the Partial TMR Selection Problem 189
5.1.2 Genetic Algorithm Parameters and Operations 191
5.1.3 Fitness Evaluation 195
5.2 Implementation 197
5.2.1 Genetic Optimization Procedure 197
5.2.2 Fitness Assignment 201
5.3 Evaluation Strategy 206

5.4 Results . . . 210
5.4.1 Precision of Criticality Prediction . . . 211
5.4.2 Quality of Optimization Results . . . 212
5.4.3 Execution Time . . . 216
5.4.4 Evaluation of 32-Bit CPU Designs . . . 217
5.5 Discussion . . . 220
5.6 Summary . . . 225

6 Selective SEU Mitigation for Storage Elements in High-Level Synthesis 231
6.1 A Profiling-Based Approach . . . 232
6.2 Implementation . . . 236
6.2.1 High-Level Representation Profiling . . . 237
6.2.2 Intermediate Representation Profiling . . . 239
6.2.3 Reliability-Enhanced High-Level Synthesis with PandA Bambu . . . 241
6.3 Evaluation . . . 243
6.3.1 High-Level Representation Profiling . . . 244
6.3.2 Intermediate Representation Profiling . . . 248
6.4 Results and Discussion . . . 251
6.5 Summary . . . 259

7 Discussion 263
7.1 Configuration Memory Fault Modeling . . . 263
7.2 Design-Space Exploration for Selective Fault Mitigation . . . 265
7.3 High-Level Storage Element Reliability Profiling . . . 267
7.4 Key Limitations . . . 268

8 Conclusion 271

List of Abbreviations

5MR	Quintuple Modular Redundancy
ALU	Arithmetic Logic Unit
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ATPG	Automated Test Pattern Generation
AES	Advanced Encryption Standard
BCR	Benefit-Cost-Ratio
BEL	Basic Element
BIST	Built-In Self Test
BRAM	Block Memory
BTI	Bias Temperature Instability
bit/s	Bits Per Second
CBRAM	Conductive Bridge ReRAM
CDFG	Control and Data Flow Graph
CE	Clock Enable
CFG	Control Flow Graph
CLB	Configurable Logic Block
CMF	Common-Mode Fault
CMOS	Complementary Metal-Oxide-Semiconductor
CMOS	Complementary MOS
CORDIC	Coordinate Rotation Digital Computer
CPU	Central Processing Unit
CRAM	Configuration Memory
CRC	Cyclic Redundancy Check
DD	Displacement Damage
DFG	Data Flow Graph
DFT	Design For Test
DICE	Dual Interlocked Storage Cell
DMR	Dual Modular Redundancy
DRAM	Dynamic RAM
DSE	Design-Space Exploration
DSP	Digital Signal Processing
DUT	Design under Test
DVFS	Dynamic Voltage and Frequency Scaling
DWC	Duplication With Comparison
DLL	Delay-Locked Loop
.dcp	AMD Design Checkpoint
ECC	Error-Correcting Code
EDAC	Error Detection and Correction
EDA	Electronic Design Automation
EDIF	Electronic Design Interchange Format
EEPROM	Electrically Erasable PROM
.ebd	AMD Essential Bits Mask
.ebc	AMD Essential Bits Value
eV	Electron Volts
FeRAM	Ferroelectric RAM
FET	Field Effect Transistor
FF	Flip-Flop
FI	Fault Injection
FIFO	First In – First Out
FIR	Finite Impulse Response
FIT	Failures in Time
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
FU	Functional Unit
GCC	GNU Compiler Collection
GD	Generational Distance
HR	Hyper-area Ratio
HDL	Hardware Description Language

HLL High-Level Language

HLS High-Level Synthesis

HPWL Half-Perimeter Wire Length

HRS High-Resistance State

HCI Hot Carrier Injection

I²C Inter-Integrated Circuit Bus

ICAP Internal Configuration Access Port

IGD Inverse Generational Distance

IP Intellectual Property

IoU Intersection-over-Union

IR Intermediate Representation

ITC International Test Conference

ISCAS IEEE International Symposium on Circuits and Systems

JSON Javascript Object Notation

JTAG Joint Test Action Group

LFSR Linear Feedback Shift Register

LRS Low-Resistance State

LU Least-used

LUT Look-Up Table

MAC Multiply-Accumulate

MBU Multi-Bit Upset

MCKP Multiple Choice Knapsack Problem

MIFTF Mean number of Injected Faults To Failure

MLC Multi-Level Cell

MRAM Magneto-Resistive RAM

MSB Most Significant Bit

MTTF Mean Time To Failure

MU Most-used

MCNC Microelectronics Center of North Carolina

NCD Native Circuit Description

NMR *N*-Modular Redundancy

ONO Oxide-Nitride-Oxide

ONVG Overall Non-Dominated Vector Generation

OxRRAM Oxygen Vacancy Resistive RAM

PCIe PCI Express

PCRAM Phase-Change RAM

PIP Programmable Interconnect Point

PLB Programmable Logic Block

PLL Phase-Locked Loop

PMIC Power Management IC

P&R placement and routing

PROM One-time Programmable ROM

PVT Process-Voltage-Temperature

QSPI Quad SPI

QoR Quality of Results

RAM Random Access Memory

RBD Reliability Block Diagram

ReRAM Resistive RAM

RHBD Radiation Hardened by Design

RRAM Resistive RAM

RTL Register Transfer Level

RVF Register Vulnerability Factor

SEB Single-Event Burnout

SECDED Single Error Correction, Double Error Detection

SEE Single-Event Effect

SEFI Single-Event Functional Interrupt

SEGR Single-Event Gate Rupture

SEL Single-Event Latchup

SEM Soft Error Mitigation

SERDES Serializer-Deserializer

SET Single-Event Transient

SEU Single-Event Upset

SLC Single-Level Cell

SoC System on Chip

SONOS Silicon-Oxide-Nitride-Oxide-Silicon

SPI Serial Peripheral Interface

SRAM Static RAM

SSA Single Static Assignment

STT Spin-Transfer-Torque

SHA Secure Hash Algorithm

Tcl Tool Command Language

TID Total Ionizing Dose

TMR Triple Modular Redundancy

TMR Tunneling Magneto-Resistance

UART Universal Asynchronous Receiver/Transmitter

VHDL Very High Speed Integrated Circuit Hardware Description Language

VPR Versatile Place and Route

WAL Waveform Analysis Language

XDL Xilinx Design Language

XML eXtensible Markup Language

Chapter 1

Introduction

Since their inception in the 1980s, Field-Programmable Gate Arrays (FPGAs) have seen widespread adoption in various fields of applications, including embedded systems, robotics, and telecommunication equipment, as well as recently in accelerators for training and inference of large-scale artificial intelligence (AI) models.

This widespread use of FPGAs is driven by their immense flexibility, consisting of a vast array of configurable combinational and sequential logic resources paired with a programmable interconnect network. These are supplemented by special-purpose resources such as memory blocks, arithmetic-logic units, and even entire cutting-edge systems-on-chip. While these resources are pre-fabricated in silicon in an FPGA, they assume their design-dependent functionality at run-time only when the chip is supplied with a configuration memory image that defines the precise function of each combinational or sequential resource and the paths through the interconnect network.

State-of-the-art FPGAs utilize different technologies for loading and storing these configuration settings: one-time programmable cells, such as fuses or anti-fuses, Electrically Erasable PROM (EEPROM) or Flash cells, or volatile memory cells, including Static RAM (SRAM) cells or Latches. Today's most advanced FPGA devices employ the latter approach – loading their configuration at power-on from a non-volatile memory into an integrated SRAM – and thereby offer virtually limitless reconfigurability as well as advanced features such as run-time partial reconfiguration.

Loading their configuration once at boot time and subsequently relying on the configuration SRAM cells to hold their state continuously to configure the FPGA to perform the functionality intended by the designer leaves the circuits implemented in the FPGA vulnerable to bit flips in these memory cells. External effects such as radiation are well known to cause upsets in memory, leading to memory cells flipping to their opposite stored value. While the effect of these bit flips in data memories of CPU-based systems can be mitigated using checksums (e.g., Error-Correcting Code (ECC)), bit flips occurring in the configuration memory cells of an FPGAs cause immediate changes to the implemented circuit, potentially directly resulting in system malfunction.

Depending on the application's requirements, using an SRAM-based FPGA in a safety-critical system may require the consideration of fault mitigation measures in the system design. A combination of several approaches is usually applied to mitigate the effects of these faults: This entails a combination of on-chip redundancy (i.e., replication of blocks of logic) to transparently mask immediate circuit malfunctions caused by these faults, periodic checking and correction of the configuration memory content ("scrubbing"), and higher-level redundancy such as chip-, board-, or device-level approaches.

In on-chip redundancy approaches, logic is duplicated or triplicated for the purpose of detecting

or masking, respectively, the changes in one of the circuit copies introduced by bit flips in the configuration memory. While this somewhat increases circuit reliability, it comes at the cost of higher resource usage than the original circuit. If this circuit fits within the target device, the placement and routing task becomes more difficult due to higher device utilization, potentially impacting the timing properties of the implemented design.

Not all configuration memory bit flips, however, immediately negatively impact the design implemented in the affected device: The fault may concern a memory cell associated with an unused FPGA resource, which does not necessarily affect the circuit implemented in the device. Alternatively, the effect of its flipping may be absorbed by the design – e.g., because the change in a lookup table is benign. Finally, the FPGA technology may also mask a configuration bit flip, which can occur because the circuit change within a configurable resource does not significantly affect the circuit behavior. Thus, a potentially beneficial way of introducing redundancy may be to selectively replicate those parts of the design that are more severely affected by a single bit flip, e.g., due to a high concentration of configuration bits with severe bit-flip consequences. This necessitates ways to determine which configuration bits in a design fall into which category – benign or malignant when flipped. Furthermore, it requires automated methods for improving a design's reliability in the presence of these bit flips while keeping costs (i.e., FPGA resource usage, and, by this proxy, delay and power consumption) reasonable.

As configuration bits commonly control functionality within the FPGA at the transistor level, the consequences of configuration bit flips also manifest themselves at this low technological level. At this level, the electrical consequences of a bit flip may cause FPGA behavior to deviate from well-defined digital logic values, thereby opening the door to Process-Voltage-Temperature (PVT) variations that can lead to different erroneous behaviors from chip to chip. Cost-effectively protecting against the immediate effects of configuration-memory bit flips requires the selective introduction of redundant logic that takes these effects into account.

Finally, as mentioned above, configuration memory cells are not the only storage elements in an FPGA that are susceptible to randomly occurring bit flips: a comprehensive solution for building cost-effective reliability-enhanced FPGA designs necessarily must also address bit flips in user-defined storage elements.

In this context, the driving goals behind the writing of this dissertation are, as developed from an analysis of the state of the art:

G1: Develop an effective and precise way of modeling the effects of individual configuration-level bit flips to predict their impact on the design's functionality, therefore enabling the directed introduction of fault mitigation measures.

G2: Utilize the insights provided by an effective configuration-level fault modeling approach to automate the process of selectively introducing fault mitigation measures.

G3: While the goals above mainly concern the mitigation of configuration bit flips, an automated way to analyze and protect a design against bit flips in its user-defined storage elements – those holding design state and data – is desirable as well for comprehensive protection. Thus, a further goal is to open up the potential for time-efficient comprehensive soft error mitigation for designs generated by modern High-Level Synthesis (HLS) approaches by developing approaches to predict the soft-error sensitivity of user-defined storage elements in these designs from a high-level design representation.

This dissertation is structured as follows to address these goals: Chapter 2 introduces the reader to relevant background in the field of soft errors, FPGA technology, and soft-error susceptibility,

and general approaches to improve FPGA soft-error resilience. Building on this understanding, Chapter 3 provides a thorough analysis of the state of the art in this field and motivates the subsequent contributions of this work by highlighting its gaps. The contributions laid out in the subsequent chapters of this dissertation address these gaps as follows.

- Chapter 4 provides an in-depth discussion, characterization methods, and empirical analysis of the bit-level effects of interconnect faults in several state-of-the-art FPGA technologies. This analysis includes the impacts of chip temperature, supply voltage, and manufacturing variations (i.e., PVT effects) on the effects of single-bit flips injected into the configuration memory. Based on these analyses, it proposes criticality prediction models for discriminating a design's bitstream into bits that cause benign and malignant consequences when flipped. These models are validated using exhaustive fault injection on several case-study designs.
- Based on the fault modeling work discussed above, Chapter 5 proposes an automated module-based Design-Space Exploration (DSE) approach towards partial redundancy insertion that co-optimizes cost (i.e., bitstream size) and vulnerability (i.e., number of configuration bits vulnerable to bit flips). This involves encoding the module selection problem for use with NSGA-II, a well-known genetic optimization algorithm, as well as defining and evaluating methods to assign cost- and vulnerability-derived fitness to individual design variants. An implementation of the proposed DSE approach is evaluated with respect to convergence, effectiveness, and overall run-time, as well as compared to related work with respect to the benefit-to-cost ratio of the generated solutions on a set of case-study designs, including two well-known 32-bit processor cores: The NEORV32[1] implementation of the RISC-V open instruction set architecture, and the LEON3[2] SPARCv8 Central Processing Unit (CPU).
- Chapter 6 proposes a profiling approach for digital logic designs generated using HLS that highlights especially important user-defined state storage elements – flip-flops and memory blocks – using simulation as part of the HLS process. At its core, the respective chapter of this dissertation proposes several methods for applying the concept of the Register Vulnerability Factor (RVF) to designs generated by HLS, as well as methods for deriving it for individual user-defined storage elements. Implementations of the approach that profile a design at two levels of abstraction during the HLS process – at the level of the high-level language, and at the level of an intermediate representation – are evaluated on a set of case-study designs sourced from well-known benchmark suites under fault injection.

The impact of these contributions with respect to the state of the art as well as their limitations are discussed in Chapter 7. Finally, Chapter 8 concludes this work and provides directions for future research in the thematic context of this dissertation.

[1] See `https://web.archive.org/web/20250128134944/https://github.com/stnolting/neorv32`, accessed 2026-01-05

[2] See `https://web.archive.org/web/20250123185022/https://www.gaisler.com/products/leon3`, accessed 2026-01-05

Chapter 2

Background

This chapter introduces the reader to relevant concepts within the context of this dissertation. Section 2.1 discusses the soft-error phenomenon, its causes, and its effects on current and emerging memory technologies. Section 2.2 introduces FPGA technology and provides examples for FPGA-based systems in applications related to safety, while Section 2.3 describes concepts related to FPGA reliability. Finally, Section 2.4 details common concepts to improve the reliability of those systems and sets the focus for the remainder of this dissertation.

2.1 Soft Errors in Digital Storage Elements

Understanding the causes of soft errors in digital memories is crucial for the implementation of techniques and approaches to mitigate these sources of errors occurring within an FPGA design.

It is generally understood that radiation, by (1) introducing charge directly, or (2) introducing ionized paths into the device, is a major contributor to bit flips occurring in digital memories [9]. However, as laser-based fault injection [10, 11] shows, the contributing effects can be caused by other mechanisms as well.

In space, cosmic radiation, solar radiation, and charged particles trapped within the Earth's magnetic field may hit electronic equipment deployed there [12]. In contrast to this harsh environment, a smaller amount of these particles and radiation reaches the Earth's surface. However, cosmic rays colliding with oxygen and nitrogen molecules in the atmosphere may generate secondary radiation that may reach equipment deployed closer to the Earth's surface [9]. The most important kinds of radioactive particles in analyzing radiation effects on electronic systems are photons, electrons, neutrons, heavy ions (such as alpha particles), and protons [12].

High-radiation environments on Earth may be found in nuclear fuel processing facilities as well as nuclear power stations, medical equipment, scientific equipment such as particle accelerators, and industrial or agricultural irradiation facilities. However, with shrinking technology sizes, chips operating outside of these environments may also experience radiation-induced soft errors due to the effects of cosmic rays colliding with Earth's atmosphere.

The effects of radiation on an electronic system – such as a memory – are commonly classified into two categories [12]:

- Slowly developing defects related to accumulating doses of radiation. These are further subdivided into (1) *Total Ionizing Dose (TID) effects* that shift the threshold voltage of Field Effect Transistors (FETs) and lower the gain of bipolar junction transistors (BJTs) by accumulating

defects and trapped charge carriers, and (2) *Displacement Damage (DD) effects*, where bombardment with energetic particles causes defects in the crystal lattice of the semiconductor material, particularly affecting bipolar transistors, solar cells, and charge-coupled devices (CCDs). In literature, the dose used in experiments is often reported in the unit *rad*, denoting the quantity of energy deposited per unit of mass of the irradiated material [12]. Experiments exploring TID effects are often done using a reference radiation source such as the isotope Co^{60}, which is widely used in various industrial and medical applications.

[13] measured doses of between 376 rad/year and 42 krad/year in geostationary orbit, depending on the amount of shielding of the employed dosimeters. Assuming a typical lifespan of 15 years for a geostationary satellite [14], its electronics would thus experience total doses of 5.64 – 630 krad.

- Instantaneous effects of the interaction between radioactive particles and the system, referred to as Single-Event Effect (SEE). These effects can further be subdivided into *hard* and *soft* errors, with the former permanently damaging the device under consideration, while the effects of the latter are generally reversible – although a power cycle may be required in some cases. They are mainly caused by an ionizing particle hitting a semiconductor device, thereby generating free charge carriers that then may affect, for example, information stored as charge [9].

  Figure 1 illustrates a possible outcome of a high-energy neutron hitting a FET and generating an ionized path in the semiconductor bulk.

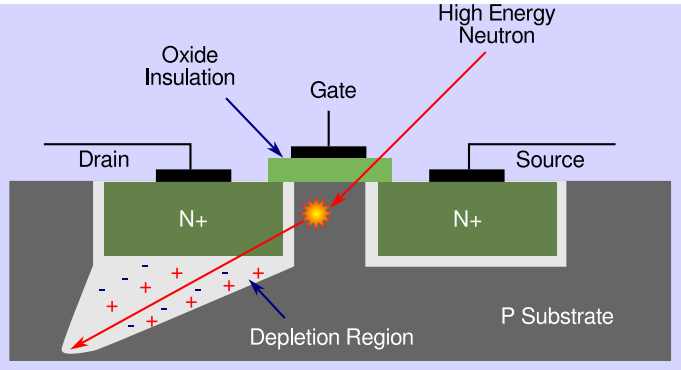


Figure 1: Effects of high-energy neutron impact on a FET [15]

Depending on the event's individual effect, the following subcategories of SEE can be differentiated [9]. Most relevant to memories, a reversible bit flip in an individual storage element in a RAM or flip-flop is referred to as a Single-Event Upset (SEU), while an error affecting multiple bits is called a Multi-Bit Upset (MBU). Other types of SEE are Single-Event Transients (SETs) (short voltage spikes generated in combinational logic or in analog hardware that may affect downstream components), Single-Event Latchups (SELs) (latch-up in CMOS devices caused by radioactive particles), as well as Single-Event Burnouts (SEBs) and Single-Event Gate Ruptures (SEGRs) (hard errors especially relevant to power transistors). Finally, more complex faulty behavior resulting from a radiation event is referred to as Single-Event Functional Interrupt (SEFI).

Radiation used in SEE-related experiments is often described using (1) the type of particle (e.g., neutron, proton, heavy ion) and (2) the acceleration energy in Electron Volts (eV). Examples are low-energy (25 meV) *thermal neutrons* and higher-energy *10 MeV neutrons* [12]. Experiments are carried out using particle accelerators [16].

Problems related to radiation-induced events as discussed above are often considered to be predominantly confined to harsh radiation environments such as outer space, primarily outside of the protection of Earth's atmosphere and magnetosphere. However, in 2022, the British Broadcasting Corporation (BBC) published an article[3] describing a collection of "real-world" computer errors attributed to radiation events on or near the Earth's surface. Examples include the Belgian 2003 federal election[4], where a soft error is suspected to be responsible for adding a suspicious 4096 unaccounted-for votes to one candidate's vote tally, and an aviation incident in 2008, where spurious data spikes from an angle-of-attack sensor caused sudden aircraft maneuvers, leading to several passenger and crew injuries. In its investigation report, the Australian Transport Safety Bureau (ATSB) included a soft error as a possible root cause [17].

Due to their charge-based nature, conventional volatile solid-state memory technologies (SRAM, and Dynamic RAM (DRAM)), as well as floating-gate or Silicon-Oxide-Nitride-Oxide-Silicon (SONOS) transistors used to build EEPROM and Flash memories, are susceptible to ionizing radiation. The impact of radiation on storage elements using these technologies is discussed in the following. Furthermore, radiation effects on a selection of emerging memory technologies (Resistive RAM (ReRAM) and Magneto-Resistive RAM (MRAM)) are briefly explored.

Especially relevant to this dissertation is the impact on SRAM cells due to their use in SRAM-based FPGAs, both as configuration technology as well as data storage in the form of block memories. In their operating principle, latches and flip-flops are similar to SRAM cells and thus are subject to similar effects. However, their layout is typically less dense, and typical FPGAs offer orders of magnitude fewer of these resources compared to Block Memory (BRAM) and Configuration Memory (CRAM). Thereby, latches and flip-flops expose a comparatively small soft-error sensitive cross-section.

2.1.1 Dynamic RAM

Chronologically, DRAM was one of the first storage technologies where radiation effects were investigated. After the possibility of cosmic rays affecting low-power TTL flip-flops[5] in satellites had been discussed in 1975 [18], a mechanism describing possible effects of alpha radiation on the recently introduced DRAM cells was proposed in 1978 [19].

The authors describe the concept of a critical charge Q_{crit} that, when delivered to the respective storage element, can flip its stored state from '0' to '1' or vice-versa [19]. Precisely, a radioactive particle hitting a bit cell's access transistor may drain its stored charge, thereby flipping a '1' to a '0'. Additionally, a model may consider if the cell is in storage state or connected to the sense amplifier in the instant it is hit by a radioactive particle [20]. Devices where Q_{crit} is larger than the charge to be delivered by a single alpha particle are inherently immune to alpha-induced errors. However, as device sizes and supply voltages shrink, Q_{crit} is also reduced, and the expected soft error rate increases [19].

The authors of [19] also discuss possible causes for observed SEUs at sea level and cite alpha particles emanating from IC packaging materials as a notable source for these errors. They also note the possibility of cosmic radiation propagating to sea level and causing bit flips [19]. This effect was investigated in 1994, with measurements using 288 KBit DRAMs taken at different elevations (underground, sea level, 1.6 km, and 3.1 km). The authors propose a method to differentiate

[3] See `https://web.archive.org/web/20250625082043/https://www.bbc.com/future/article/20221011-how-space-weather-causes-computer-errors`, accessed 2026-01-05

[4] See `https://web.archive.org/web/20250725082613/https://technews.acm.org/articles/2004-6/0728w.html#item13`, accessed 2026-01-05

[5] Of Texas Instruments' SN54Lxx series

between the number of soft errors observed due to alpha-emitting packaging and those due to cosmic radiation. While a clear positive correlation between elevation and cosmic-ray-related soft errors was found, these errors have also been observed at sea level [21].

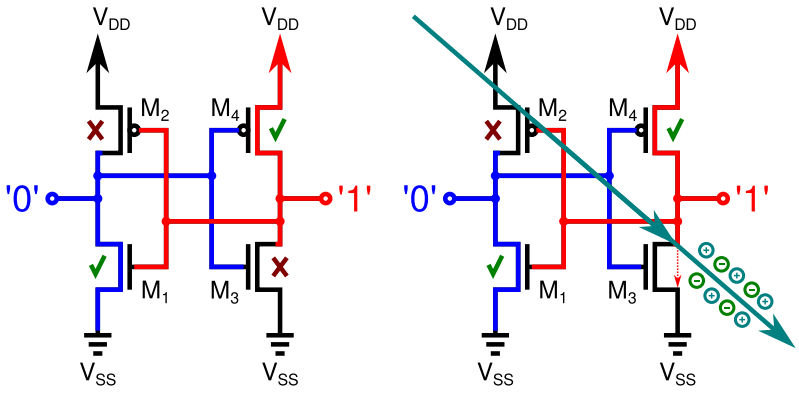
To protect applications requiring DRAM from the effects of bit flips, some Radiation Hardened by Design (RHBD) DRAM cell implementations have been proposed (e.g., [20]). These cells, however, are not widely used. Alternatives are to (1) use cells with larger Q_{crit}, limiting the density of DRAM in applications expected to be exposed to larger amounts of radiation, combined with (2) the use of Error Detection and Correction (EDAC) to mitigate the effect of bit flips. For recent dense DRAM memories, manufacturers have started to include transparent on-die ECC to reduce the observed bit error rate [22]. Also, Single Error Correction, Double Error Detection (SECDED) codes, such as extended Hamming codes, are available for circuit-level soft-error mitigation in memories [9].

With DRAM, not only the radiation sensitivity of the bit cells themselves, but also the complex control logic has to be considered. However, RHBD cells (e.g., radiation hardened flip-flops) may be used to implement this logic [23].

DRAM cells are not only susceptible to single-event errors as discussed above. Continued exposure may degrade the device's retention time due to TID effects, requiring more frequent refresh operations to compensate. Furthermore, increased operating current and total functional failure due to TID effects have been reported [24].

2.1.2 Static RAM

When radiation-induced errors were first discussed in the 1970s and 1980s, they were predominantly a problem of DRAMs due to the small charges stored in these memories' bit cells. However, with shrinking transistor size and operating voltage, the Q_{crit} of SRAM cells has also decreased into regions where radiation became more and more relevant for SRAM reliability [9].



(a) Functional State

(b) Particle Striking Drain Region of M_3, causing an ionized track to GND

Figure 2: SRAM SEU mechanism (based on [12])

In SRAM, the most common radiation-induced error is an SEU that flips a bit in memory. Figure 2a shows the four transistors making up the latch at the center of a 6T CMOS SRAM cell. The cell is in a state where NMOS transistor M_1 and PMOS transistor M_4 conduct. If a radioactive particle hits the drain region of the NMOS transistor M_3 in its off state, it generates a trace of charge carriers (see Figure 1), causing a transient current from the drain of M_3 to V_{SS}. Depending on the size of this pulse, this may be enough to turn on M_2 (and turn off M_1), and the SRAM cell flips to the complementary state [12].

A multitude of radiation-hardened SRAM cells have been proposed. These cells employ additional transistors (leading to 10T, 12T, and 14T cells) to improve the cell's stability in the presence of particle strikes and thus lower their soft error rate [25]. An interesting example is the Dual Interlocked Storage Cell (DICE) latch, a 10T memory cell that increases radiation tolerance without relying on relative transistor sizing. It leads to an 81% area overhead and 20% longer fall time when used as part of a flip-flop. The basic cell may also be used to implement SRAM bit cells [9]. Cross-coupled resistors – forming a low-pass filter and thus blocking the effect of a transient pulse from flipping the stored bit – have also been used in Complementary MOS (CMOS) SRAM cells [26].

As they are built from transistors, SRAM cells are vulnerable to TID and DD effects, degrading these devices in addition to SEE.

In addition to radiation hardening – which inevitably and drastically lowers memory density – ECC schemes such as those used in DRAM may also be used. However, these are only well-suited for conventional byte- or word-oriented memories. When used to form an FPGA's configuration memory, however, each SRAM cell's output directly controls a circuit element, enabling an SEU to directly affect the circuit implemented by the device configuration. This may cause logical errors (e.g., erroneous Look-Up Table (LUT) content) and opens or short circuits in interconnect resources, among other effects [27]. Thus, safety-related designs implemented in SRAM-based FPGAs usually require a combination of fault mitigation measures, including ECC and redundancy. FPGA vendor AMD regularly publishes a *reliability report* detailing the experimentally determined SEU susceptibility (according to the relevant JEDEC standard JESD89A [28]) of their devices. According to the most recent published iteration, for the widely-used 28 nm Artix 7 and Zynq 7000 parts, a configuration memory soft error rate of 74 FIT/MBit is reported, where one FIT represents one failure per one billion device hours [29]. For context, the configuration of the largest Artix 7 part (xc7a200t) comprises roughly 77 MBit [30] of configuration memory, leading to a maximum soft error rate of 5698 FIT/device if the entire configuration would be used. As real-world FPGA designs depend only on a fraction of the configuration bits to remain set correctly, however, the real-world soft error rate experienced by such designs is much lower.

2.1.3 Electrically Erasable PROM

In EEPROM and the related Flash Memories, information is stored non-volatile as charge in a floating storage gate or Oxide-Nitride-Oxide (ONO) charge-trapping structure sandwiched between the control gate and the channel. Charges are placed in these structures using hot carrier injection or quantum tunneling. If a charge is stored in these structures, this shifts the threshold voltage of the transistor, which is sensed during readout.

Radiation may affect an EEPROM cell when it can shift the threshold voltage of the transistor far enough to corrupt (1) its stored information state or (2) the value being read out during the particle hit [31]. This can happen as a result of continued bombardment with charged particles, gradually neutralizing the charge placed in the storage structure. This gradually shifts the cell's threshold voltage until it corresponds to corrupt information when read out [31], which can also be modeled as a shift of threshold voltage to an intermediate level between the voltage used to indicate a "1" and the voltage used to indicate a "0" [26]. Due to smaller margins, this effect increases in severity with the number of levels used, making Multi-Level Cell (MLC) memory more severely impacted than Single-Level Cell (SLC) memory [32]. TID-related effects might also impact the endurance of the memory [31].

In addition to these TID-related effects, single-event effects have also been investigated [31], which

seem to become more and more severe with ever-shrinking cell sizes and the shift towards MLC memories [33]. Single-event effects are becoming a problem with EEPROM memories due to the lower charge necessary to shift a scaled-down transistor's threshold voltage [33].

Furthermore, while older, less aggressively scaled EEPROM memories may have been less radiation-sensitive, the peripheral logic surrounding the memory array is implemented using conventional CMOS technology for registers, flip-flops, and buffers. These structures are susceptible to single-event effects in the same way SRAM is, potentially leading to SEFI behavior [31].

Hardening floating-gate devices against radiation may be done by (1) implementing differential sensing, (2) increasing the on/off ratio by increasing the amount of charge stored in the floating gate, and (3) adapting the fabrication process by including hardened oxide layers to shield the floating gate [26]. The gate dielectric in SONOS memories inherently is relatively radiation-hard. Radiation-hardening of these devices primarily focuses on hardening peripheral logic [26].

In [34], different *emerging* memory technologies (Ferroelectric RAM (FeRAM), MRAM, ReRAM, Phase-Change RAM (PCRAM)) were compared to a 32 nm Samsung SLC Flash device (K9FAG08U0M) and a Cypress CY14B1011 charge-trapping SONOS device, primarily to assess TID effects. The "conventional" Flash device began to experience data corruption after a dose of approximately 100 – 150 krad, with being completely corrupted after receiving a dose of 400 krad. However, the SONOS did not experience any data loss [34]. Radiation experiments were conducted while the devices were turned off. A Co^{60} radiation source was used [34]. In experiments carried out during operation (i.e., powered-on memory devices), the SONOS device was able to operate beyond exposure to 1 Mrad.

Microchip (formerly Microsemi) offers radiation-tolerant FPGA devices such as the RT PolarFire family based on 28 nm SONOS technology and the RTG4 family based on 65 nm Flash technology [35]. Limiting factors in the application of EEPROM-based FPGAs are (1) density and (2) re-programmability: Microchip's largest RT PolarFire device provides 481,000 4-input LUTs and flip-flops [35], which is exceeded by AMD's mid-range Kintex UltraScale+FPGA family that provides up to 842,000 6-input LUTs and 1,685,000 flip-flops, while the high-end Virtex UltraScale+ family offers even higher densities. In the relevant device datasheet, Microchip specifies up to 2000 programming cycles for PolarFire devices' embedded configuration Flash cells, where *"programming, zeroization, and verify operations all count as a programming cycle"*. This limits the flexibility with which these devices can be used – potentially limiting their in-field updateability, and, in essence, precluding them from using dynamic reconfiguration techniques that are common practice in SRAM-based FPGAs. While SRAM-based FPGAs require non-volatile configuration storage as well, commonly used Quad SPI (QSPI) Flash devices such as Infineon's S25FLxxxL series supported by AMD Vivado [36] provide a minimum of 100,000 cycles per sector [37].

A future upgrade path towards unrestricted programmability of configuration storage may be provided by emerging memory technologies: Lattice Semiconductor announced support for MRAM devices for configuration storage[6], among which Avalanche Technology's space-grade AS3xxx204/8 QSPI devices are specified by the manufacturer to endure at least 10^{16} write cycles [38]. With a minimum endurance of 500,000 write cycles per 4-byte sector, Fujitsu/RAMXEED's MB85AS12MT ReRAM device [39] still offers an improvement over the usual endurance of SPI Flash while providing reduced read (0.7 mA at 10 MHz) and write (2.5 mA) currents when compared to an Infineon S25FLxxxL Flash device of similar intended role (10 mA when reading at 10 MHz, 17 mA when writing and erasing [37]).

[6] See `https://web.archive.org/web/20250725104204/https://www.latticesemi.com/en/Blog/2025/01/16/16/41/From-Flash-to-MRAM`, accessed 2026-01-05

2.1.4 Emerging Memory Technologies

Many proposed future memory technologies (at different stages of commercialization) are not charge-based in the same way as SRAM, DRAM, or EEPROM. They store their information using electric polarization (FeRAM), magnetization (MRAM), crystalline/amorphous phase of a dielectric (PCRAM), or presence/absence of conductive filaments in a dielectric (ReRAM). This makes them potentially less sensitive to the radiation effects discussed in the previous chapters, except for their continued reliance on CMOS technology for their peripheral circuitry.

In this section, MRAM and ReRAM are briefly discussed as candidates for future FPGA configuration memory technologies. These technologies are discussed, for example, by [40] (Spin-Transfer-Torque (STT)-MRAM) and [41] (HfO_2-based Oxygen Vacancy Resistive RAM (OxRRAM)). In general, their SEU-immunity and projected density make them ideal candidates for next-generation non-volatile FPGAs. However, FPGAs based on these technologies are out of reach at the time of writing of this dissertation. Thus, considering the current state of the art, the discussion in this section pertains mostly to non-volatile configuration storage devices as a Flash-EEPROM replacement as discussed above.

Magneto-Resistive RAM

In MRAM, the remanent magnetization of a *free* layer relative to the magnetization of a *fixed* layer of a Tunneling Magneto-Resistance (TMR) “sandwich” is used to store information. Between the free and fixed layers, the cell contains a tunnel barrier; the resistance of the entire stack is high when the two layers are magnetized in opposite directions and low if they are magnetized in parallel due to different magnetic effects on electrons with opposite spins. The magnetization of the free layer is altered either by generating an internal magnetic field using currents through wordlines and bitlines, or by directly transferring spin torque via a spin-polarized current passing through the free layer.

MRAM devices are less sensitive to radiation than charge-based memories such as SRAM, DRAM, and FLASH, as there exists no known mechanism that would allow charged particles to interact with the magneto-resistive property of the MRAM bit cells in a single-event fashion [33]. However, since MRAM relies on a very thin tunneling barrier between the pinned and free magnetic layers with highly uniform interfaces, accumulated radiation damage may degrade the bit cells over time. These effects reduce the on/off ratio of the cell due to intermixing between the materials forming the barrier and the magnetic layers [32].

In [16], SEE in commercial MRAM devices MR10Q010 (1 MBit) and MR25H00 (4 MBit) by Everspin were investigated under neutron radiation at two different levels of energy (thermal neutrons and 14.8 MeV neutrons). In contrast to FeRAM and ReRAM, no proton radiation experiments seem to have been conducted on MRAM in [16]. Apart from two SEFI events, no upsets are reported for MRAM, underlining its relative radiation insensitivity in line with earlier experiments. The two observed SEFIs are assumed to have been caused by CMOS peripheral circuitry. Compared to FeRAM and ReRAM, MRAM was observed to be more radiation tolerant in total [16].

Resistive RAM

Similar to MRAM, ReRAM also stores information by altering the resistance of a bit cell. Here, however, ions are migrated into or out of the material instead of changing the active material’s magnetization. In filamentary ReRAM, these ions reversibly form (or leave behind) conductive channels, thereby switching the cell from a High-Resistance State (HRS) to a Low-Resistance

State (LRS) and vice-versa. ReRAM is described as not especially susceptible to SEE, with bit-flips virtually only being possible by transients caused by SETs in the selection transistor, causing currents large enough to disturb the cell [32]. It is reported to tolerate TID in the range of up to 10-100 kgr (i.e., 1 Mrad to 10 Mrad), above which radiation may cause defects and phase transitions [32]. These effects may increase leakage current and negatively affect switching properties (e.g., linearity in MLC ReRAMs) [32].

An in-depth review of radiation effects on ReRAM devices is provided in [42], differentiating between (1) TID effects and SEE effects and (2) reporting data for both Conductive Bridge ReRAM (CBRAM) and OxRRAM devices.

In CBRAM devices, the LRS and HRS could still be clearly differentiated after being irradiated to determine TID effects. Additionally, neither the current needed to SET/RESET CBRAM cells nor their retention characteristics (measured as the decrease in the resistance in the HRS or increase of the resistance in the LRS with accumulating write cycles) seems to be affected by the received dose of radiation [42]. However, in CBRAM cells manufactured in a different process, failure to switch between resistance states has been observed [42]. Radiation also appears to affect the impedance characteristics in devices irradiated before forming has been carried out [42]. TID experiments on TiO_2, TaO_x, and HfO_2-based OxRRAM devices are also summarized in [42]. The evaluated TiO_2 cells retained their pre-programmed LRS and HRS up to 23 Mrad [42]. In TaO_x cells, however, TID effects may switch cells in HRS to LRS, while cells in LRS retain their state when irradiated with x-rays [42]. HfO_2 cells were shown not to be TID-sensitive for radiation doses up to 7 Mrad [42].

SEE in both CBRAM and OxRRAM could appear due to radiation affecting the selection transistor in a One Transistor, One Resistor (1T1R) ReRAM cell. This may happen due to SETs turning on the selection transistor, leading to voltage pulses that may inadvertently and partially set or reset the ReRAM cell, depending on the biasing applied to the memory array at the moment of the particle hit. This essentially leads to an erroneous re-programming of cells from an LRS to an HRS and vice-versa [42]. A possible way to mitigate these SEEs is to construct memories without using transistor-based cells, e.g., One Resistor (1R) or One Selector, One Resistor (1S1R) cells. Additional SEFIs due to the CMOS peripheral circuitry is possible [42].

In [16], experiments on two commercially available ReRAM devices, Fujitsu's MB85AS4MT (4 MBit) and MB85AS8MT (8 MBit), have been conducted using 1 MeV protons, thermal neutrons, and 14.8 MeV neutrons. In static tests (i.e., programming the device with static content, irradiating, then checking the pattern) under proton radiation, some hard fails (permanent stuck-at 0 of all memory cells) of the 4 MBit device were observed. Under neutron radiation, mainly SEFIs were observed in which communication with the memory over its SPI interface was no longer possible. A single stuck-at-0 SEU was observed on the 4 MBit device under thermal neutron radiation. The 8 MBit device is reported to be more sensitive to SEFI than the 4 MBit device.

2.2 FPGA Technology and Applications

Figure 3 shows a typical modern heterogeneous FPGA architecture [43]: The FPGA's predefined Configurable Logic Blocks (CLBs) providing basic combinational and sequential resources – usually LUTs and flip-flops – are aligned in a grid surrounded by routing resources (i.e., connection and switch boxes). In heterogeneous devices, certain columns (or rows) in that grid of CLB are replaced by special-function or hard Intellectual Property (IP) blocks such as CPU cores, interface or transceiver blocks (e.g., for PCI Express (PCIe) or multi-gigabit Ethernet), memory blocks, and Digital Signal Processing (DSP) blocks (i.e., fast multipliers or Multiply-Accumulate (MAC)

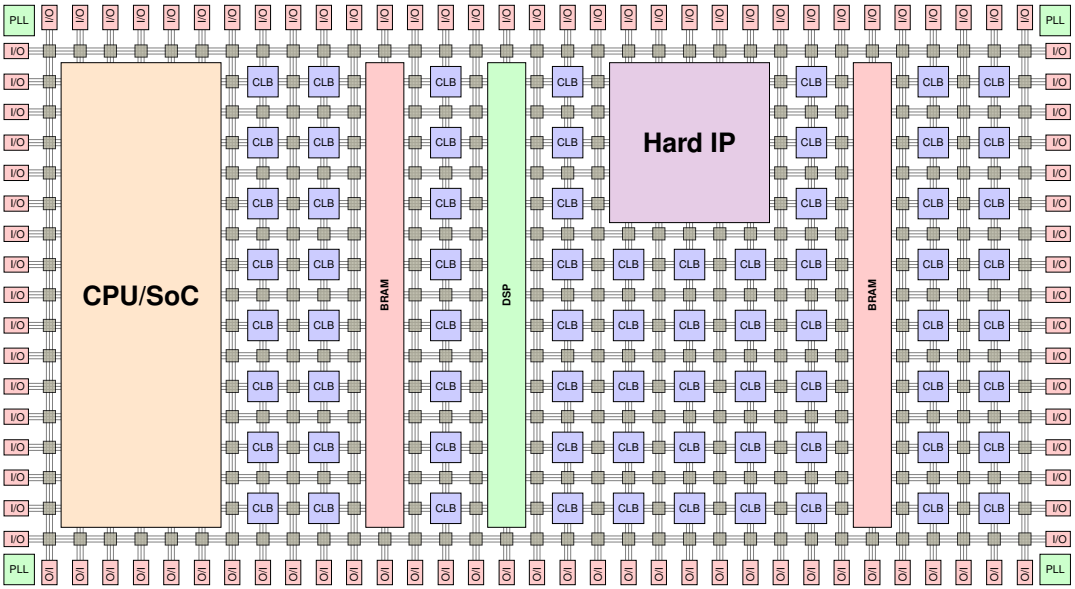


Figure 3: Typical Island-style FPGA Architecture (based on [43])

units). At the edge of this grid of basic and advanced FPGA resources lie configurable input-output blocks, usually providing tri-state drivers, multiple I/O standards, programmable delay lines, pull-up and pull-down, as well as on-chip termination circuitry. Clocking resources such as configurable Phase-Locked Loops (PLLs) and Delay-Locked Loops (DLLs), as well as delay-sensitive blocks such as DRAM controllers, are integrated in the vicinity of these input-output cells.

In the last decade, more and more *SoC FPGAs* have been released by device vendors. These devices combine an FPGA fabric with a high-performance System on Chip (SoC), usually providing multiple CPU cores and periphery within one package. Examples for this trend are AMD's Zynq, Zynq UltraScale(+), and Versal device families[7], Altera's/Intel's *SoC* variants of its Cyclone V, Arria, Stratix, and Agilex families [44], and Microchip's PolarFire and SmartFusion (2) SoCs [8]. As an example for the performance promised by these devices, even the low-end Zynq 7000S device family provides an ARM Cortex A9 application-class CPU core that can be clocked up to 766 MHz [45].

As mentioned above, a flexible interconnect network is responsible for connecting the FPGA's resources. Modern FPGA offer a variety of interconnect options:

- A global routing network is responsible for supplying high-frequency and high-fanout signals such as clock and control signals across the chip with minimal skew between the FPGA's regions. This is often supplemented by regional routing networks that supply certain subdivisions of an FPGA with clock or control signals. This also involves special global or regional routing resources such as clock buffers, multiplexers, or dividers.
- Long interconnect lines transport logic signals within the FPGA fabric across large distances – i.e., several to tens of columns or rows.
- Short interconnect lines carry these signals between cells in their respective neighborhood, e.g., to the neighboring row or column, or two to six rows or columns away.

[7] See `https://web.archive.org/web/20250818183835/https://www.amd.com/en/products/adaptive-socs-and-fpgas.html`, accessed 2026-01-05

[8] See `https://web.archive.org/web/20250818185139/https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas`, accessed 2026-01-05

- Local interconnect lines connect the signals carried within the FPGA's routing network to and from the individual logic resources within a CLB or special-purpose block.

To flexibly connect between those interconnect lines, modern FPGAs usually provide a wealth of routing resources – usually programmable switches, buffers, or multiplexers whose output depends on the settings of configuration bits [43]. The configuration of these elements can be responsible for upwards of 80%–90% FPGA's configuration memory [46, 47].

Due to their flexibility, FPGAs have found widespread adoption in Application Specific Integrated Circuit (ASIC) prototyping [48, 49] as well as communications [50, 51], signal and image processing [52, 53, 54], autonomous driving and robotics [55, 56, 57], and scientific [58, 59, 60, 61, 62] applications. Recent advances in artificial intelligence have further increased interest in FPGA devices for embedded AI [63, 64, 65] and for the acceleration of large-scale AI models [66, 67, 68].

In the remainder of this, a selection of concrete case studies **CS1–CS4** for FPGA usage in commercial and scientific applications is presented. In each of the applications discussed above, FPGA malfunction can lead to severe consequences – from invalid scientific results and costly mission failure to lives lost.

CS1: Audi's zentrales Fahrerassistenzsteuergerät (zFAS)[9] is an Advanced Driver Assistance System (ADAS) introduced by Audi for their A8 model in 2017[10]. The system is reported [55] to provide SAE Level 3 assisted driving functions, i.e., conditional driving automation under the supervision of a fallback-ready user [69]. Alongside an Nvidia Tegra K1 GPU, a Mobileye EyeQ3 SoC, and an Infineon Aurix microcontroller, zFAS features an Intel Cyclone SoC FPGA for facilitating communication between the aforementioned processing elements [55].

In SAE Level 3 assisted driving applications, the ADAS is tasked with *"sustained lateral and longitudinal vehicle motion control"* as well as *"object and event detection and response"* [69]. As examples, SAE J3016 mentions (1) traffic jam assistance, i.e., autonomously driving the vehicle in stop-and-go traffic by accelerating and braking appropriately to stay within the vehicle's lane and keep a safe distance from other vehicles, and (2) highway overtaking assistance, appropriately accelerating and steering the vehicle as well as avoiding collisions with other road users to pass a slower vehicle upon activation by the driver. While SAE J3016 requires a driver using a Level 3 ADAS to be ready at all times to respond to a request to intervene upon a detected failure of the ADAS, and to take over the driving task in case of an obvious system malfunction, a certain level of system reliability still must be guaranteed to implement a trustworthy driving assistant capable of giving ample warning to the user in case of malfunction. This also concerns potential FPGA components such as the Intel SoC FPGA – an SRAM-based device – found in Audi's zFAS unit.

CS2: Light Detection and Ranging (LiDAR) sensors use an active light source to measure the distance to obstacles, either in 2D (e.g., 360° LiDAR) or in a (limited) 3D space. A common principle is Time-of-Flight (ToF), in which distance is measured using the time until a modulated light pulse emitted by the LiDAR is reflected onto a light sensor.

One application for these sensors lies in navigation for autonomous robots, where LiDAR based Simultaneous Localization and Mapping (SLAM) algorithms allow to continuously build a map of the surroundings, within which the robot can locate itself and thereby uses it for navigation [70]. Furthermore, LiDAR technology is also applied in ADAS for object detection and collision avoidance [71]. The consequences of failure of these sensors may range from

[9] Translated to English: *"central driver assistance control device"*

[10] See `https://web.archive.org/web/20211028115404/https://www.audi-mediacenter.com/en/press-releases/awards-for-audi-11469`, accessed 2026-01-05

autonomous ground robots such as vacuum-cleaning robots causing property damage by colliding with their surroundings to loss of human lives due to an ADAS or ground robot navigation subsystem failing to detect a person or obstacle in the path of driving.

This case study is based on three commercially available LiDAR sensors: (1) The Velodyne Puck VLP-16 [11], (2) the Ouster OS1-64 3D LiDAR with 45° vertical field of view[12,13], and (3) the Livox MID-360 3D LiDAR with 59° vertical field of view[14,15]. According to the reverse-engineering articles referenced above, all of these sensors employ FPGAs: Velodyne Puck VLP-16 employs an Intel/Altera Cyclone III SRAM-based FPGA, Ouster OS1-64 both an AMD Zynq 7020 SoC FPGA and an Artix 7 FPGA (both devices are SRAM-based), and Livox MID-360 uses an SRAM-based AMD Zynq 7020 SoC FPGA as well. It is assumed that devices of similar role and specification profile make use of FPGA technology as well, making FPGA reliability relevant to LiDAR sensors and their applications.

CS3: The Compact Muon Solenoid (CMS) and A Toroidal LHC ApparatuS (ATLAS) experiments, integrated into CERN's Large Hadron Collider (LHC) particle accelerator, detect particles emanating from collisions. Both contain particle detectors that enclose the LHC's accelerator ring at beam collision points. These particle detectors generate a relatively high rate of events and resulting data: Collisions happen in bursts of tens of events with the beam crossing frequency of 40 MHz, with each collision generating 1–2 MB in data. To match the capabilities for data processing and storage, this must be reduced such that only the most "interesting" results are recorded and further processed. To that end, both CMS and ATLAS employ a multi-stage *trigger* system that analyzes the data output by a subset of the detectors' frontend units [72].

This trigger system must be flexible enough to allow it to be configured to the parameters of the physical events under consideration. To that end, for example, FPGAs are used at several points of CMS's first-stage trigger's (Lvl-1) pipeline [72, 59]. In particular, CMS's *Global Trigger* stage of the first-stage trigger, aggregating data from the previous trigger pipeline stages to converge on a final trigger decision, was upgraded to use AMD Virtex 7 FPGAs for Run II (2015–2018) of the LHC. This upgraded global trigger hardware, comprising six FPGA boards, was kept in place for Run III (2022–2026), with trigger improvements being done by adapting the employed algorithms[16]. Using AMD's Virtex 7 SRAM-based FPGA in proximity to a particle accelerator generating high-energy collisions raises the question of radiation tolerance and soft-error mitigation, which is explored, for example, in [73].

CS4: As part of the *Mars 2020* mission, NASA launched the *Ingenuity* drone to demonstrate the

[11] Reverse Engineering Report: `https://web.archive.org/web/20250217172447/https://electronics360.globalspec.com/article/16089/teardown-velodyne-lidar-puck-vlp-16-sensor`, accessed 2026-01-05

[12] Specification: `https://web.archive.org/web/20250724001032/https://ouster.com/products/hardware/os1-lidar-sensor`, accessed 2026-01-05

[13] Reverse Engineering Report: `https://web.archive.org/web/20241104044641/https://electronics360.globalspec.com/article/17802/techinsights-teardown-ouster-os1-64-lidar-sensor`, accessed 2026-01-05

[14] Specification: `https://web.archive.org/web/20250414224237/https://www.livoxtech.com/mid-360`, accessed 2026-01-05

[15] Reverse Engineering Report: `https://web.archive.org/web/20250825193110/https://fastriver.dev/2024/12/08/mid360-teardown/`, accessed 2026-01-05

[16] See `https://web.archive.org/web/20250823141907/https://ep-news.web.cern.ch/content/cms-trigger-system-run-3`, accessed 2026-01-05

first powered atmospheric flight on an extraterrestrial body, completing 72 flights by 2024[17].

The drone's high-level flight controller runs on a dual-core lockstep ARM Cortex R5 CPU, which is monitored by a radiation-tolerant FPGA. In case of a mismatch between the two primary flight control CPUs, the FPGA may switch over to an identical backup flight controller. The FPGA further implements low-level attitude control, facilitates communication with the drone's sensors (gyroscope, accelerometer, rangefinder, and inclinometer), and implements low-level control of the drone's actuators (servos via Pulse-Width Modulation (PWM) and flight motors via Space-Vector PWM (SVPWM)), as well as communication between flight and navigation computer based on a Qualcomm Snapdragon SoC [60, 74].

The employed FPGA is a Flash-based Microchip ProASIC 3L; additional soft-error mitigation has been implemented for storage elements by triplicating selected flip-flops. This makes the integral FPGA the most radiation-tolerant component of the digital flight architecture [74].

2.3 FPGA Reliability

To mitigate the effects of malfunctioning FPGAs, and thus to reduce the risk of critical system failures, it is imperative to understand which factors may lead to failure of the designs implemented on these devices.

In a similar way to other complex integrated circuits, FPGAs may fail in the field due to manufacturing defects. These defects are expected to be caught when the devices are tested after manufacturing. Due to their complexity, achieving satisfactory fault coverage in FPGAs proves challenging targets for conventional testing methods [75]. As a commercial example, AMD publishes an overview of their test strategy for FPGA devices[18]. AMD tests the digital logic within their FPGA devices using a combination of Automated Test Pattern Generation (ATPG), logic Built-In Self Test (BIST) and fault-graded test vectors. Using these methods, a coverage of more than 98% of single stuck-at faults and at least 80% of at-speed transition faults is reached. Furthermore, the 500 most delay-critical paths are tested using delay-specific ATPG, IP blocks such as transceivers, PLLs, and mixed-signal IPs are tested using application-specific test methods, while embedded memory blocks are tested using the Modified Algorithmic Test Sequence (MATS) algorithm [76] for all port widths. Thus, a state-of-the-art process that is not specific to FPGA exists for managing this potential source of FPGA failure. Manufacturing defects are thus not considered within the scope of this dissertation. However, variations induced by the manufacturing process (i.e., PVT) may interact with the FPGA's susceptibility to soft errors while not being identified within the testing process.

Another group of effects that may cause FPGAs to fail at runtime concerns device aging. Commonly, effects such as Bias Temperature Instability (BTI) and Hot Carrier Injection (HCI) are considered to be the main culprits in FPGA aging. Both effects raise the threshold voltage and lower the switching speeds of transistors within the FPGA [77]. Eventually, these effects could lead to malfunctioning FPGA logic elements or timing violations, leading the design implemented in the FPGA to fail unpredictably. In [77], both methods to artificially and selectively age FPGA components, as well as to characterize the aging process, are described. Once a device has been characterized, and an aging profile has been developed, continuous characterization allows for estimating the age – and thus aging-related fault probability – of the FPGA. A similar approach is taken in [78] to detect

[17] See `https://web.archive.org/web/20250729162706/https://science.nasa.gov/mission/mars-2020-perseverance/ingenuity-mars-helicopter/`, accessed 2026-01-05

[18] See `https://web.archive.org/web/20240411154444/https://www.xilinx.com/support/quality/quality-design.html#test`, accessed 2026-01-05

recycled “counterfeit” FPGAs [78]. While certain aging-related faults may be masked by soft-error mitigation measures, aging is considered out of scope for this dissertation.

Effects of the printed circuit board (PCB) carrying the FPGA, as well as effects of failing external components such as power supply, clock generator, or configuration storage, are not specific to FPGAs. These effects are thus not considered to lie within the scope of this dissertation. This also applies to systematic design flaws such as Hardware Description Language (HDL) coding errors.

The scope of this dissertation is, as discussed above, soft errors caused by the effects and mechanisms described in Section 2.1. Bit flips due to soft errors can cause disruptions in an FPGA’s programmed functionality in two major ways:

1. A bit flip occurring in a configuration memory changes the circuit implemented by the FPGA. This bit flip may induce a change in a logic function (implemented in a LUT), change the configured behavior of another FPGA logic element such as a flip-flop (e.g., it might change reset or clock polarity), or alter the routing of one or more signals with the FPGA’s interconnect. The effect varies depending on which configuration bits are affected by the bit flip. An SEU affects one configuration bit, while an MBU may affect multiple. This is a semi-permanent fault that stays active unless the configuration bit flip is corrected. The fault can be benign when, for example, an unused logic element is affected, or it is affected in a non-disruptive way. It may also be masked by redundancy that is naturally part of the design or introduced to improve reliability.

   This kind of vulnerability is largely restricted to SRAM-based FPGAs, as Flash-based devices are less susceptible to radiation-induced bit flips due to the inherently lower soft-error sensitivity of the configuration cells as described in Section 2.1.3.

2. A bit flip occurring in the user-defined memory (i.e., flip-flops, memory blocks, First In – First Out (FIFO) buffers, shift registers, and distributed Random Access Memory (RAM)) corrupts the design’s state and data content. This error can be masked when the flipped value is overwritten before it has the chance to propagate through the design. It may also be masked by inherent or deliberately introduced redundancy, such as the triplication of flip-flops as done in the *Ingenuity* Mars drone [74] or by ECC in RAM blocks.

User-configured hard-IP cores and FPGA periphery (such as the configuration interface) may also contain storage elements (RAM blocks and flip-flops) that are susceptible to soft errors. However, these elements are assumed to be negligible in area when compared to the user-defined storage elements, and, in particular, the configuration memory in SRAM-based devices. Thus, the contribution of these elements is not considered within the scope of this dissertation.

The contribution of SEUs to the FPGA design’s failure rate can thus be described as shown in Equation (2.1). Due to the technological difference among the cells implementing these memories, they contribute to the overall design’s bit-flip-related failure rate $\lambda_{\mathrm{design}}$ at different levels. Here, N_{FF}, N_{BRAM}, and N_{CRAM} describe the number of SEU-susceptible storage elements (flip-flops/latches, block memory bits, and configuration memory bits, respectively) present in the design.

$$\begin{aligned}\lambda_{\mathrm{design}} = {} & N_{\mathrm{FF}} \cdot \lambda_{\mathrm{FF}} \cdot p_{\mathrm{prop,FF}} + \\ & N_{\mathrm{BRAM}} \cdot \lambda_{\mathrm{BRAM}} \cdot p_{\mathrm{prop,BRAM}} + \\ & N_{\mathrm{CRAM}} \cdot \lambda_{\mathrm{CRAM}} \cdot p_{\mathrm{prop,CRAM}}\end{aligned} \tag{2.1}$$

The technology-dependent failure rate for these elements is captured by λ_{FF}, λ_{BRAM}, and λ_{CRAM}. This failure rate depends on the assumed radiation profile (e.g., the type of radiation and its energy)

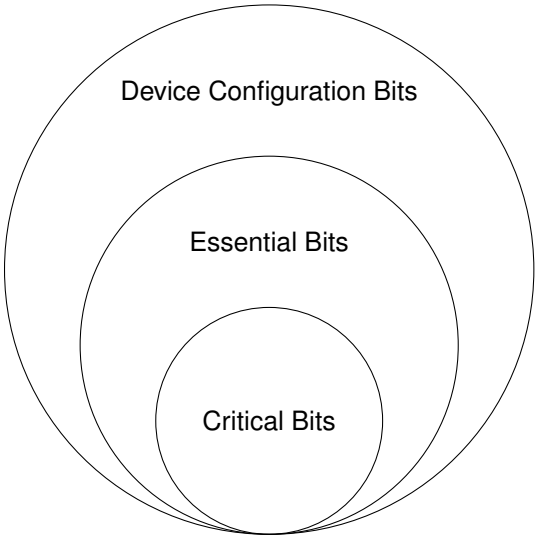


Figure 4: Relationship of Device Configuration Bits, Essential Bits, and Critical Bits [88]

and the interaction between radiation and the storage elements. FPGA vendors frequently publish this data as part of a *"reliability report"* or similarly entitled document. For example, AMD publishes FIT data for BRAM, CRAM, and UltraScale URAM in their *"Device Reliability Report"* [29], while both Intel and Microchip only publish this data under Non-Disclosure Agreements (NDAs) via their respective web portals. However, an archived version of a Microsemi reliability report is available for the SmartFusion2 and IGLOO2 devices [79]. Radiation-related SEU sensitivity is also frequently explored in literature: [80, 81, 82, 83] empirically characterize the sensitivity of various AMD/Xilinx devices to several kinds of radiation, with [80, 84, 85] performing similar studies on Intel devices. CologneChip's GateMate technology's radiation sensitivity is characterized in [86], while [87] is concerned with certain Lattice FPGAs.

As the third and final component of Equation (2.1), $p_{\mathrm{prop,FF}}$, $p_{\mathrm{prop,BRAM}}$, and $p_{\mathrm{prop,CRAM}}$ represent the average fault propagation probability for each storage element category.

A design's soft-error-related failure rate may thus be reduced by (1) reducing the *number* of radiation-sensitive storage elements, (2) reducing the *probability* that a bit flip particular element causes an unmasked failure, or (3) by moving to a device *technology* that is inherently less susceptible to SEUs.

For an accurate prediction of the impact of bit flips in the configuration memory, an estimate for the number of radiation-sensitive configuration bits (N_{CRAM} in Equation (2.1)) is required. Assuming that *all* of the device's configuration bits are radiation sensitive leads to overly pessimistic reliability predictions. This is due to it being unlikely that the correct functionality of the design depends on *all* device configuration bits: no realistic design will make use of all device resources, especially its routing resources.

In this context, Xilinx/AMD introduced the *essential* and *critical* subsets of the device's configuration bits [88]. Xilinx/AMD categorizes the subset of device configuration bits that *"are associated with the circuitry of the design"* as *essential* bits. In turn, the subset of *essential* bits that actually cause incorrect design behavior when upset are categorized as *critical* bits. The relationship between these configuration bit categories is illustrated in Figure 4. AMD's ISE and Vivado FPGA design tool optionally report, as part of their *essential bits* feature, the number of essential bits estimated by an proprietary, non-disclosed, algorithm. This provides a less pessimistic estimate of N_{CRAM} for AMD's FPGAs devices.

Furthermore, AMD's FPGA tools can be instructed to output an AMD Essential Bits Mask (.ebd) file, which is a binary mask of the device's bitstream indicating the location of these essential bits, and an AMD Essential Bits Value (.ebc) file, which contains the values of these configuration bits. In literature, these files are commonly referred to by their file extension (.ebc and .ebd).

2.4 Reliability Enhancement Techniques for FPGAs

This section outlines options to improve the resilience of designs implemented using SRAM-based FPGA against single-event upsets.

Improving the tolerance of SRAM-based FPGA against these events can be divided into on-chip and off-chip methods. Off-chip methods include the introduction of shielding materials to attenuate radiation before reaching the chip die, as well as the deliberate placement of the radiation-sensitive components (including FPGAs) in the most radiation-shielded regions of the target system's enclosure. Depending on the application's requirements, the thickness (and thus effectiveness) of radiation shielding must carefully be counter-balanced with size and weight constraints. While even thin shielding is very effective against alpha and beta radiation [89], shielding against higher-energy ionizing radiation such as gamma rays and cosmic neutrons is more difficult [89, 90]. [90] suggests that especially the resilience against cosmic particles cannot be feasibly addressed by shielding, and should be addressed on the design or process level. An obvious design-level off-chip way of improving a system's radiation tolerance would be to use multiple identical devices with external voting – i.e., chip-level *N*-Modular Redundancy (NMR) – assuming that only one of the three chips is ever affected by a soft error. While conceptually simple, this method at least triples component count, circuit board area, and power usage of the FPGA components. This also significantly increases the system's weight.

Given that many applications of FPGAs are in size-, weight- and power-constrained embedded systems, and the effectiveness of shielding against certain types of radiation is limited within these constraints, this work focuses on on-chip methods to improve the reliability of a single FPGA. Also, the suitability of a single power and clock supply is assumed in this work.

A self-evident way of improving the soft-error tolerance of an FPGA-based system using on-chip methods at the design level is to choose a less soft-error susceptible chip technology. This includes FPGAs that use Flash or PROM cells to store their configuration. This, however, forgoes the advantages of SRAM-based FPGAs: SRAM-based devices can be reconfigured quickly, making even run-time (partial) reconfiguration possible. This allows design updates and to swap out designs, design implementations, or even parts of the design, depending on the current mission's or mission phase's requirements. On the other hand, as mentioned in Section 2.1.3, Microchip, for their Flash-based PolarFire devices, specifies only up to 2000 programming cycles [91]. One-time Programmable ROM (PROM)-based devices, such as Microchip's Antifuse-based Axcelerator and SX-A devices, can only be programmed once. This work assumes that the application requires the reconfigurability or density offered by an SRAM-based FPGA.

Future developments may also make FPGAs based on emerging memories such as ReRAM or MRAM possible and commercially viable. These devices would unify the advantages of SRAM-based devices (dense integration as well as fast and near-limitless reconfigurability) with the resilience of Flash-based devices. However, there is not yet a commercially available FPGA device based on these technologies on the horizon, not to mention available at the moment of writing this dissertation.

In place of these methods, several ways of improving the soft-error robustness of SRAM-based

FPGAs have been developed. The remainder of this section provides an overview of the most widespread concepts. These concepts have been developed over the approximately four decades within which FPGA have been available, and are founded on precise modeling and analysis of the reasons FPGA devices fail due to soft errors. The current state of the art considering fault modeling for FPGAs is discussed in more detail in Section 3.2. A more in-depth discussion of state-of-the-art fault mitigation concepts themselves can be found in Section 3.3.

- Soft errors within the configuration of an FPGA device can be corrected by re-configuring the device. In some applications, the functionality provided by the FPGA is only required on demand, or there is sufficient temporal or spatial redundancy that allows the application to tolerate temporary or partial malfunctions of the FPGA design. In these applications, periodically taking the FPGA device offline for reconfiguration – *"scrubbing"* [92] – is an effective way to mitigate the effects of configuration bit flips. While this so-called "blind scrubbing" is effective at correcting bit flips, it may take the FPGA device offline for a considerable amount of time.

  Alternatives are scrubbing techniques that use checksums stored within the FPGA device such as AMD's frame-wise ECC [92], and approaches that combine on-chip redundancy (i.e., replicated logic) with scrubbing. As an example, [93] proposes duplicating portions of the design and comparing the copies' outputs. If they mismatch, a scrubbing cycle is triggered. Another possibility to trigger scrubbing cycles are on-chip monitors generated from Property Specification Language (PSL) or System Verilog assertions repurposed from Assertion-based Verification (ABV) [94] or re-using monitors present for security purposes [95].

- Scrubbing does, however, not protect the state of the design: while most FPGA devices allow triggering a reset of all flip-flops, and reconfiguring the device from its bitstream can re-initialize all memory blocks, this destroys pre-existing state or data stored in these elements. Furthermore, not all applications may tolerate the design being non-functional until a fault in the configuration is periodically blindly scrubbed, or detected and corrected by selective scrubbing. To remedy these aspects, approaches introducing spatial redundancy have been proposed.

  The concept of *N*-Modular Redundancy (NMR), often in the form of Triple Modular Redundancy (TMR), is a well-known redundancy concept in reliability engineering [96]. In this concept, the design as a whole or in part is instantiated multiple times (e.g., thrice in TMR, five times in Quintuple Modular Redundancy (5MR)). The concept's underlying assumption is that only one of the copies is affected by a soft error. Thus, the fault's consequences can be masked by a majority vote among the three, five, or more (sub-)circuit copies [97]. A voter can trivially be extended to output a mismatch signal in addition to the voting result, thus delivering a possible trigger signal for a (slower) scrubbing mechanism to prevent the accumulation of soft errors within the configuration.

  As a drawback, replicating parts of logic naturally leads to increased FPGA resource usage: The redundant circuit requires at least three times the amount of logic resources of the original design; the design's routing is complicated by instantiating redundant logic as well. Both of these effects impact the circuit's power dissipation as well as its timing properties. Thus, approaches that selectively replicate those parts of a design that are assumed to be most critical to the design's correct functionality are of high interest to provide cost-efficient fault mitigation.

- Precisely modeling the effects of bit flips in the configuration of SRAM-based FPGA devices reveals that not all functionally equivalent configurations of the same FPGA device contain

the same number of bits that cause the system to fail if they are flipped by a soft error. This means that some configurations for the same design are less susceptible to soft errors than others. The effect can be explained by the influence of alterations in placement and routing on the configuration: While in one configuration, two signals transporting critical information may be routed in a way where they can be short-circuited by a single-bit flip, another configuration may route these signals farther apart. Furthermore, logic resources such as flip-flops FPGA devices are often organized in *slices*, i.e., groups of resources which share certain configuration bits. An example from AMD's 7 Series FPGAs is that all configurable register cells within a slice share the configuration bits that decide if the register acts as a flip-flop or a latch [98].

Thus, reliability-oriented placement and routing approaches (see [99]) based on device-specific fault modeling have been introduced. These approaches can, on their own, not mitigate all potential soft errors – they merely *reduce* the number of soft-error-sensitive configuration bits. To protect the irreducible subset of configuration bits that define the design's functionality, they are supplemented by redundancy (e.g., TMR) and scrubbing approaches.

- Finally, user-defined data – i.e., the state of the FPGA design – is stored in latches or flip-flops as well as memory blocks. As discussed in Section 2.3, these storage elements are susceptible to soft errors as well, albeit usually fewer in number than configuration bits in SRAM-based FPGAs.

  Faults in these elements can be mitigated in a similar way to logic by replicating the individual storage elements and voting over their outputs. This is, for example, done in NASA's Ingenuity Mars drone [74]. As memories or register banks get wider, however, conventional parity-based techniques for protecting against bit flips in memory become more efficient. For example, calculating, storing, and verifying parity bits according to an extended Hamming code allows for detecting double-bit flips and correcting single-bit flips [96]. Modern FPGA technologies such as AMD's 7 Series and UltraScale families, Lattice's ECP5 devices, as well as Intel's Agilex family integrate ECC encoders and decoders within the silicon of their memory blocks [100, 101, 102, 103]. In other devices, ECC encoding, decoding, and checking logic must be implemented in the FPGA fabric, decreasing maximum frequency and introducing additional configuration bits possibly susceptible to soft errors.

In general, the application's requirements define the selection and possible trade-offs between the general categories of soft-error mitigation approaches that are applied to the FPGA design under consideration. The acceptable level of (un)reliability and requirements for the development process can be derived from standards relevant to the intended field of application such as IEC 61508 [104] for general safety-related applications, ISO 26262 [105] for automotive applications, as well as DO 254/EUROCAE ED 80 [106] concerning hardware development for aerospace applications. Other limiting factors, such as system cost – increase in FPGA resource usage, power dissipation, and weight, as well as decrease in clock frequency – dictate the necessity of a possible trade-off, and are also related to economic decisions regarding the related product: While spatial redundancy encompassing the entire system (e.g., triplicating the entire system) is highly efficient at mitigating faults, it may prove prohibitively expensive for all but the most safety-critical applications. In other applications, partial redundancy or other trade-offs that trade higher fault rate for lower system cost may be acceptable [107].

Chapter 3

Literature Review

Achieving application-adequate and cost-effective design reliability in a complex and inherently vulnerable technology like SRAM-based FPGAs builds on a broad set of analysis techniques and concepts. These tools have been subjects of research and development almost since the introduction of FPGA, leading to a broad selection of pre-existing approaches in today's state of the art. In this chapter, Section 3.1 first discusses challenges in this field based on the problem description in Chapter 1 and the background described in Chapter 2. Furthermore, it introduces requirements for a comprehensive reliability improvement toolflow for SRAM-based FPGAs in the context of the goals discussed in Chapter 1. The following sections then discuss the state of the art regarding bit-level fault modeling (see Section 3.2) and reliability-improvement techniques and approaches (see Section 3.3) for these devices. While these approaches form the basis for the tools and concepts proposed this dissertation, it aims to improve on the state of the art by fulfilling the requirements formulated in this chapter. In this regard, Section 3.4 summarizes the state of the art and introduces the concrete research gaps this dissertation aims at filling.

3.1 Requirements & Challenges

The aim of this dissertation is to propose a comprehensive and cost-effective toolflow that improves the reliability of an FPGA design in the presence of SEU in the FPGA's memory elements. As the configuration memory exposes the largest error cross-section, the initial focus is set on mitigating bit flips in this category of storage elements. However, also mitigating bit-flips in the contents of user-defined storage elements (flip-flops, LUTRAM, and BRAM) is addressed.

This section argues a set of requirements that an approach or toolflow in this category should fulfill to provide (cost-)effective protection against SEU. These requirements are specified on two levels:

- *Tool-Level Requirements* refer to a reliability-improvement toolflow that takes an FPGA design as its input and provides one or more variants of this design with improved reliability properties. The requirements in this category argue which overall properties this toolflow should fulfill and which challenges it needs to address the shortcomings of current reliability improvement tools. These requirements focus on more widespread applicability, area-efficiency, and effects not previously considered when introducing fault tolerance measures such as the effect of alterations in placement and routing as well as inter- and intra-device variation on design reliability. The requirements of this level are summarized in Table 1 and discussed in Section 3.1.1. They are related to goals **G2** and **G3** formulated in Chapter 1.

- *Bit-Level Requirements* specify, in more detail, the requirements towards a configuration bit-level reliability analysis approach underlying this reliability-improvement toolflow: The effects of the considered configuration-memory faults manifest themselves in the form of alterations of the bit-level configuration of the FPGA. Thus, an accurate configuration-bit-level fault analysis approach is an important prerequisite for developing an effective reliability-improvement toolflow for SRAM-based FPGAs. The related requirements are summarized in Table 2 and discussed in Section 3.1.2. They are relevant to goal **G1** discussed in Chapter 1.

Sections 3.1.1 and 3.1.2 discuss and motivate the requirements in these categories. Following these sections, the remainder of this chapter discusses works directly related to fulfilling these requirements, works describing significant effort related to a subset of these requirements, and works that, while tangential to these requirements, still discuss efforts relevant to a better understanding of underlying effects and approaches:

Section 3.2 discusses approaches aiming to model and predict the effects of bit flips in the configuration of SRAM-based FPGAs and other work related to that field.

Section 3.3 reviews approaches targeting cost-aware approaches to FPGA fault tolerance at different levels of abstraction. In this context, it covers reliability-aware placement and routing approaches, approaches to selectively introduce redundancy, reliability improvement approaches, and approaches aiming to improve reliability during the HLS process.

At the end of each section, works directly related to fulfilling the requirements discussed above are assessed regarding their degree of fulfillment. Finally, based on this assessment, Section 3.4 expounds on the research gaps this dissertation fills.

3.1.1 Tool-Level Requirements

Table 1 introduces a number of requirements that are necessary for a comprehensive and cost-efficient reliability improvement tool(-flow) for SRAM-based FPGA as relevant to goals **G2** and **G3** of this dissertation as introduced in Chapter 1). These requirements are discussed in the remainder of this section.

Table 1: Requirements for Reliability Improvement Toolflows

Requirement	Description
REQ:TOOL:MASKING	The tool introduces redundancy to mask configuration bit flips.
REQ:TOOL:DSE	The tool performs reliability-related design space exploration.
REQ:TOOL:AREA_RELIABILITY	The tool explores the cost-reliability design space.
REQ:TOOL:PARETO	The tool provides a “family” of Pareto-optimal solutions.
REQ:TOOL:PNR	The tool considers placement and routing effects on reliability.
REQ:TOOL:BIT	The tool takes bitstream properties into account (see Section 3.1.2)
REQ:TOOL:PVT	The tool acknowledges PVT effects on its results. (see Section 3.1.2)

***REQ:TOOL:MASKING*:** This dissertation focuses on applications where this transparent masking is absolutely necessary. This excludes applications with sufficient error detection capabilities and

temporal leeway within the application to allow for temporal redundancy (e.g., detecting a faulty result, repairing the FPGA's configuration and re-calculating results).

As discussed in Section 2.3, soft errors in the configuration can be corrected by *scrubbing*, i.e., periodic checking and reloading of the configuration from a less SEU-sensitive memory (e.g., an external configuration flash). The total correction time may take up to tens of milliseconds for large devices [108]. To allow a design to function until *scrubbed*, and to prevent effects of faults that cannot easily be corrected by scrubbing, redundant logic – i.e., spatial redundancy – is introduced into the design (e.g., TMR).

This is done by first instantiating multiple copies (TMR or 5MR) of the design. Subsequently, majority voting is performed across these copies' outputs. Under the assumption that any one bit flip only affects one of the design copies, the negative consequences of a bit flip (i.e., one of the copies malfunctions) can be masked by the other copies by out-voting the non-functional copy.

Thus, the first requirement for a reliability-improvement tool, as argued by this dissertation, is that it shall introduce spatial redundancy into the design to improve its reliability by masking potential errors. This is opposed to approaches that – instead of introducing redundancy – aim to improve reliability by carefully avoiding error-prone circuit constellations (such as bridging faults or Common-Mode Faults (CMFs)) during placement and routing or finding beneficial synthesis optimization settings. While these approaches may somewhat improve the reliability of a design at a given "operating point", every design contains an irreducible subset of configuration bits necessary to implement the design's functionality. This subset contains bits that define the functions realized by LUTs or would cause disconnections in the design's routed nets if flipped. Faults in these bits cannot be mitigated without introducing redundant logic that masks the ensuing malfunction. Nevertheless, a redundancy-introducing approach may be combined with a placement and routing approach that considers the configuration-level soft-error sensitivity of the implemented solution.

***REQ:TOOL:DSE*:** The amount of redundancy introduced into the design is expected to increase design cost in terms of FPGA resource utilization. As this correlates with higher FPGA utilization, power consumption and critical path delay are also affected. If partial redundancy is applicable to the design – which is the case for all but the most safety-critical designs, where absolute fault masking must be guaranteed – introducing redundancy into parts of the circuit that are highly relevant to the system's correct operation (i.e., parts of the circuit that, after implementation, contain a large number of critical bits) is more cost-effective compared to introducing redundancy into randomly-selected or less critical parts of the circuit. The different selections of replicated design partitions, or otherwise introduced selective fault tolerance mechanisms, form a design space in which the cost and benefit of each variant can be quantified along multiple dimensions. These include the remaining number of critical bits, FPGA resource utilization, power draw, and critical path delay. Careful selection of these partitions can drastically improve the benefit gained from "invested" costs. Thus, an optimizing reliability improvement toolflow should *explore* this *design space* spanned by reliability improvement on the one hand, and the associated cost on the other hand.

***REQ:TOOL:AREA_RELIABILITY*:** Resource utilization is the primary cost to consider in FPGA design: a design that takes up too many resources may become non-placable or non-routable for the target FPGA. When selecting parts of a design for replication as part of a selective redundancy scheme, there exists a trade-off between resource utilization and reliability: The more logic is protected by replicating it, the more resources the resulting FPGA design will consume. On the other hand, a design that has a large portion of its logic protected will also exhibit higher

fault tolerance. Additionally, non-uniform criticality of subsets of the design's logic to its correct functionality – for example, considering a module that consists of very few logic resources but is absolutely critical to the design's correct functionality – leads to the emergence of beneficial and detrimental subsets of the design when selected for replication. A beneficial subset, when replicated, eliminates a disproportionally large portion of the design's critical bits while only slightly increasing resource utilization. A detrimental subset, on the other hand, does little to reduce the design's critical bits but is expensive in terms of resource utilization. Thus, resource- or area-efficient reliability optimization should be the primary goal of a cost-effective reliability-focused DSE toolflow. The toolflow, thus, should aim at generating the most reliable solution possible at a given overhead measured in terms of design size. This allows the toolflow to provide the designer with a solution that satisfies the application's reliability requirements while keeping the associated resource utilization in the range that can be implemented on the target FPGA device.

***REQ:TOOL:PARETO*:** Instead of receiving a single, optimized design variant as an output of this toolflow, it may be preferable to choose among several design variants along a cost-reliability curve. Combined with partial reconfiguration abilities, this allows dynamically choosing a design variant with a certain level of fault tolerance and – for example – power dissipation as required by the current mission phase. Thus, a design-space exploration tool should generate a family of Pareto-optimal design variants differentiated by their respective area cost and criticality.

***REQ:TOOL:PNR*:** Redundancy is often introduced at a relatively high level (HLS or Register Transfer Level (RTL)) in approaches from literature. However, its effects have to prove useful at the lowest level in the FPGA: The introduced redundancy must effectively reduce the critical bits in the FPGA design in order to improve the its resilience against randomly ocurring bit flips. However, changes at the formerly mentioned higher levels may not directly translate to the expected reliability improvements on the FPGA level.

For example, consider a design whose RTL description is altered by triplicating a selected module and adding a majority gate for each output. Applying high-level reliability improvement approaches targeting SEUs in the configuration memory then raises the following questions:

- *How will these changes in the design affect*
  (1) *the netlist*,
  (2) *the placement of the replica and their voter cells, and*
  (3) *the remaining non-triplicated logic?*
- *How will these changes alter the routing of the nets in the non-triplicated logic?*

To answer these questions satisfyingly, a reliability improvement approach must be able to consider the effects a certain replicated subset has on the placement and routing of the design.

***REQ:TOOL:BIT*:** While – in the best case, considering that no common-mode faults are present in the design – all critical bits of the selected module are now removed from the selected partition of design, the changes necessary to the overall placement and routing of the non-triplicated logic may introduce other critical bits into the design. This may happen by packing certain combinations of logic elements in the same FPGA slice that were not packed together before, making the combination of packed-together cells vulnerable to slice-level configuration bit flips (e.g., bits that control the behavior of all LUTs or flip-flops in the same slice). Furthermore, the placement

changes necessary to accommodate the triplicated modules may lead to longer or shorter routing on non-triplicated wires. Changes in routing may also cause particularly incompatible nets to close together – in a way that a single-bit error can erroneously short-circuit them – thus making shorts with critical consequences to the design between these nets more likely. It can be seen from the examples above that the tool must consider reliability on the bitstream level in making decisions about which modules to replicate and evaluate the usefulness of the proposed partial replication variants of the original design. In this context, the primary goal of the tool should be to reduce the absolute number of critical bits present in a design, not only the *fraction* of critical bits. This is because the absolute number of these bits defines the vulnerable cross-section the design exposes to potentially harmful radiation.

***REQ:TOOL:PVT*:** Consequences of particular faults in the FPGA may be, for example, (1) causing wire segments to float, (2) causing short circuits whose results depend on relative driver strength, or (3) introducing additional delay to nets. These faults violate the conditions under which the specified digital operation of the FPGA is guaranteed, opening the door to possible PVT effects in the case of a fault – and thus temperature-related, voltage-related, intra-, or inter-chip variation of fault outcomes. If a tool uses fault injection for characterization, PVT effects may thus distort this analysis: If the design is evaluated on one device, the results cannot necessarily be generalized to all other devices of the same model. The tool should be aware of this possibility and offer ways to analyze and minimize the effects of PVT on its generated partial replication solutions, aiming at generating results that are transferable to other FPGA device instances of the same type.

3.1.2 Configuration-Bit-Level Requirements

An important precondition for several tool-level requirements as formulated in Section 3.1.1 is to be able to consider reliability on the bitstream level. Table 2 introduces a set of requirements in order to do so effectively and to address a wide range of challenges in this area – thereby addressing goal **G1** of this dissertation as introduced in Chapter 1. The remainder of this section describes and motivates these requirements.

Table 2: Formulated Requirements for Bit-Level Reliability Analysis Approaches

Requirement	Description
REQ:BIT:MAPPING	Mapping of bitstream to netlist is performed.
REQ:BIT:MODELING:LOGIC	Bit-level fault models for logic resources.
REQ:BIT:MODELING:INT	Bit-level fault models for interconnect.
REQ:BIT:MODELING:SPECIAL	Bit-level fault models for special-function blocks.
REQ:BIT:BITWISE_PREDICTION	Reliability of a bitstream is predicted.
REQ:BIT:CHARACTERIZATION	Fault models are based on device characterization.
REQ:BIT:VALIDATION	Modeling is validated on real-world devices.
REQ:BIT:PVT	Modeling takes PVT effects into account.
REQ:BIT:DEVICES	More than one up-to-date device type is considered.
REQ:BIT:RUNTIME	Justifiable analysis tool runtime.

***REQ:BIT:MAPPING*:** The tool must be able to correlate the design's bitstream and netlist elements to analyze the effects of design alterations and validation experiments using fault injection. This includes pinpointing bits responsible for disrupting multiple cells or modules in the design and,

as a subset, bits that are possibly responsible for CMFs by affecting multiple TMR replica of the same design. The tool thus needs to consider (1) the bitstream, (2) information about the design's placement and routing, and (3) the design's netlist to achieve this mapping.

This is only possible to a sufficient degree of precision if the tool is able to determine the circuit-level effects of a bit flip occurring in a given bit in the design's bitstream by applying a circuit-level fault model. Examples of the application of this degree of modeling are:

- Bit flips in under-utilized LUTs: An FPGA technology generally provides LUTs with a given number of inputs. If logic functions over fewer inputs need to be implemented, parts of the LUT's inputs are driven to static logic levels – usually by disconnecting the respective inputs from the interconnect network. Thus, when only single-bit errors are considered, a certain portion of the LUT's configuration bits become "don't care" bits: A fault in those bits will never be able to affect the LUT's outputs.

- Bit flips in fractured LUTs: Some FPGA technologies provide a second output for each LUT that, instead of a function over all n inputs, outputs a function over a subset of $n-1$ inputs. By statically driving the n^{th} input to '1' (as a "selection" input), it is possible to implement two functions over $n-1$ inputs in a single n-input LUT. Bit-level fault modeling has to consider the propagation of the fault via both of the LUT's outputs.

- Bit flips in interconnect resources: Depending on the configuration encoding and the circuit-level implementation of the FPGA's interconnect resources and general interconnect structure, as pointed out by [27], different topological changes can occur in a design as a consequence of a configuration bit flip. These include *Open*, *Conflict*, *Bridging*, or *Antenna* faults. Each of these changes may affect the circuit differently, possibly benignly under one condition but maliciously under another.

REQ:BIT:LOGIC*, *REQ:BIT:INT*, and *REQ:BIT:SPECIAL*:** This required degree of modeling implies considering the bit-level effects on the reliability of added fault mitigation features on the basic combinational and sequential logic implemented in the FPGA (REQ:BIT:LOGIC***), as well as the interconnect network of the implemented circuit (***REQ:BIT:INTERCONNECT***). As most FPGA designs make use of special-function blocks, also the effect of special-function blocks such as BRAM and DSP resources should be considered (***REQ:BIT:SPECIAL***).

***REQ:BIT:BITWISE_PREDICTION*:** These fault models enable the identification of the configuration bits that are expected to be critical to the design's functionality. This allows for direct reliability improvement measures or for assessment of the criticality-related fitness of certain design variants. In the latter use-case, for example, in the form of applying bitstream-level analysis in the fitness function of a DSE approach, it is required that the approach performs bitstream-level predictions.

***REQ:BIT:CHARACTERIZATION*:** Most FPGA families, however, are not documented to a degree that allows to perfectly predict the behavior of configuration faults from a model taking a combination of bitstream, placement and routing, and architectural information into account. Thus, a bitstream-related reliability-improvement tool should be based on systematic device characterization experiments. These experiments allow for identifying the behavior of certain device resources (especially those that behave non-digitally in some fault cases) to form more precise predictions of the consequences of individual faults.

***REQ:BIT:VALIDATION*:** The effects of modeling and reliability prediction can significantly influence the decision if a bit is regarded as critical or non-critical. Thus, the outcome of this modeling plays a large role in guiding the selective introduction of redundancy into the design. Consequently, it is necessary that any modeling or prediction done by this tool is systematically and extensively validated before being applied. This includes verifying if its predictions about the criticality of certain bits hold or under which conditions they do not hold.

***REQ:BIT:PVT*:** To allow a reliability improvement tool (see Section 3.1.1) to consider PVT effects in a satisfactory way, these effects must be understood when modeling the bit-level effects of SEUs in the configuration memory. Furthermore, as characterization and validation potentially involve fault injection experiments carried out on real FPGA devices, these effects must be already addressed in this phase of the construction of device-focused fault models. Otherwise – i.e., when not considering the possibility of PVT effects in characterization experiments – bias may be introduced into the constructed fault models.

REQ:BIT:DEVICES*:** While bitstream modeling implies strong dependence on the actual FPGA device type a particular tool is geared towards, an approach that does not constrain the user to a particular device type is preferable. Thus, requirement ***REQ:BIT:DEVICES argues that more than one device type should be considered by the bitstream-level analysis. These device types should be up-to-date, that is, neither tools nor silicon should be discontinued by the manufacturer at the time of writing. This implies that the tool has to cope with the effects of miniaturization of device structures in the deep sub-micron range. This includes, for example, larger PVT effects in the interconnect.

***REQ:BIT:RUNTIME*:** Finally, the complexity of the task of analyzing several thousands of bits and their consequences in the routing graph of the circuit may impact tool runtimes. However, as it may be necessary to run this tool as part of a fitness function of a DSE approach, tool runtime significantly impacts the scalability of the approach. As static analysis may need to be supplemented by fault injection experiments – which necessarily take more time than static analysis – to achieve accurate criticality estimates within the fitness function, expeditious static analysis is especially important. Thus, the static analysis tool should provide a runtime that is justifiable for use in a fitness function of a DSE approach.

3.2 Bit-Level Fault Modeling and Prediction

This section describes works that focus on the modeling of faults in the configuration memory of SRAM-based FPGAs and the prediction of the criticality to a circuit's functionality based on these models. In particular, Sections 3.2.1 to 3.2.8 describe full-featured tools and approaches in this field of application. Core properties, including the year(s) of publication, target FPGA technologies, intended application, and availability status for these approaches, are summarized in Table 3.

Table 3: Summary of Bitstream-Level Analysis Approaches Discussed in this Section

	Approach		
Property	STAR [109, 110, 111, 112]	SEVAX [113]	Rad-SyMPaR/VERI-Place [114, 115, 116]
Year	2003-2007	2014	2022, 2023
Devices	AMD Virtex AMD Virtex 2 AMD Spartan 2	AMD Virtex 5 AMD Virtex 6	AMD Ultrascale Microchip RT-ProASIC3 and RT4G NanoXplore NG
Scope	Criticality Estimation Reliability-oriented Impl.	Criticality Estimation	Reliability-oriented Impl.
Availability	Closed	Open-Source	Closed
Discussion	Section 3.2.1	Section 3.2.2	Section 3.2.3

	Approach		
Property	PyXEL [46, 117]	ACME/ACME-2 [118, 119]	BAFFI [120]
Year	2018, 2020, 2023	2019, 2022	2023
Devices	AMD 7-Series AMD Ultrascale	AMD 7-Series AMD Ultrascale	AMD 7-Series AMD Ultrascale
Scope	Criticality Estimation Reliability-oriented Impl.	Fault Injection	Fault Injection
Availability	Closed	Closed	Open-Source
Discussion	Section 3.2.4	Section 3.2.5	Section 3.2.6

	Approach	
Property	BFAT [121, 122, 123]	Bricas et al. [124]
Year	2023, 2024	2024
Devices	AMD 7-Series	AMD 7-Series
Scope	Post-Radiation Analysis	Criticality Estimation
Availability	Open-Source	Closed
Discussion	Section 3.2.7	Section 3.2.8

In addition to the approaches shown in Table 3 and discussed in Sections 3.2.1 – 3.2.8, Section 3.2.9 discusses other related works that do not offer full-featured criticality prediction.

Finally, Section 3.2.10 summarizes the discussed works, evaluates the approaches shown in Table 3 according to the bit-level requirements introduced by this dissertation (see Table 2), and describes gaps left in the state of the art in this regard.

3.2.1 STAR

STAR [109, 110, 111] is a toolflow that aims at statically predicting the effects of configuration bit flips effects and thereby the bitstream's critical bits. The author's motivation is to accelerate analysis times compared to fault injection. Its general focus is the determination of bits capable of causing

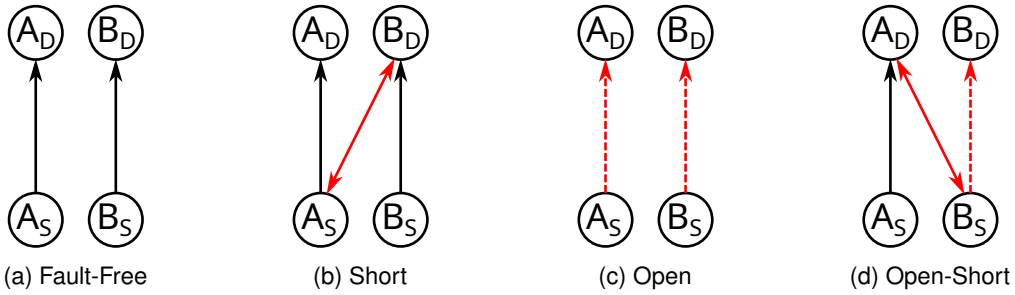


Figure 5: Interconnect Faults Affecting Multiple Nets [110, 111]

CMFs in TMR designs. For this purpose, the effects of SEUs that affect multiple TMR replica are determined. This includes identifying bits configuring CLBs that contain cells from multiple TMR replica and routing-related SEU that may cause effects in nets may belong to different replica.

However, *STAR* focuses more on identifying CMF-related interconnect bits than on analyzing CLB-related bits. To identify these interconnect bits, the circuit-level effects of each bit flip are deduced from a functional model of the associated interconnect element. This model is described in [109], and is based on the vendor's configuration-to-netlist tool NCD2VHD. This tool converts an FPGA configuration given in a vendor-specific format – Native Circuit Description (NCD) – back to VHDL. This allows the authors to initially generate a configuration for a placed and routed circuit in the form of an NCD file, injecting an error in the configuration stored in this file, and converting it back to a netlist using NCD2VHD. In this way, a structural model for the interconnect faults that are possible in the target technologies is derived.

STAR analyzes different FPGA technologies (AMD Virtex, Virtex-II, and Spartan-II). In these technologies, SEUs in interconnect-related configuration bits can affect a single net or multiple nets. The authors concentrate on faults affecting multiple nets as relevant to their goal of identifying potential sources for CMFs. As shown in Figure 5, an original configuration of two nets (see Figure 5a) may be affected by an SEU such that shorts between the two nets (see Figure 5b), opens in the two nets (see Figure 5c), or an open in one net and a short in the other net (see Figure 5d) emerge. In the case that each of these nets is part of a different TMR replica, this may violate the single-fault assumption underlying TMR, thereby causing an unmitigated fault. In order to identify the configuration bits capable of causing these faults, the authors consider the routing graph of the circuit and color each vertex according to its membership in one of the three TMR replica. By cross-referencing the analyzed routing resources with the ones present in the design's routing graph, they then consider the changes made to the routing graph by each fault (see Figure 5). Thereby, *STAR* is able to predict SEU criticality in a TMR design: If a single fault affects two nets of different TMR replica, it is considered critical. Otherwise, the fault is considered mitigated by TMR.

The authors validate their predictions by implementing TMR versions of several circuits (*add8*, *add16*, *mul8*, *filter*, and *8051*). The resulting bitstreams are then analyzed for bits that violate the TMR functionality as discussed above. Fault injection on randomly sampled configuration bits is then performed. The resulting number of identified critical bits is subsequently compared to the predicted one. This comparison shows that the static analysis performs well in identifying bits capable of causing TMR violations. In contrast, fault injection lags behind regarding discovered critical bits. The authors argue that this is due to the random sampling missing some critical bits. They perform a second set of fault injection on all the predicted bits and verify that all of them are actually critical. The proposed static analysis approach achieves speedups between 122× and 1244× depending on the analyzed design. This results in static analysis runtimes of 2-6 minutes

for the *add8*, *add16*, *mul8*, and *filter* designs and 238 minutes for the *8051* CPU. With the number of configuration bits for *add8*, *add16*, *mul8* published in [111], *STAR*'s analysis speed can be determined in Bits Per Second (bit/s) as shown in Table 4. Table 4 also reports the analysis time and speed reported by [111] for Fault Injection (FI) experiments. In [112], these predictions are additionally validated against radiation.

Table 4: Analysis Performance Results for *STAR* [111]

Design	# Bits	Analysis Time (min)		Analysis Speed (bit/s)	
		STAR	FI	*STAR*	FI
add8	9785	1.85	978.5	88.2	0.17
add16	11963	3.57	1196.5	55.8	0.17
mul8	17448	5.18	1744.8	56.1	0.17

Table 5: Evaluation of *STAR* [109, 110, 111, 112] against Bit-Level Requirements

Requirement	Evaluation	Explanation
REQ:BIT:MAPPING	●	To allow identification of CMF bits in TMR designs, STAR identifies the netlist elements affected by an SEU.
REQ:BIT:MODELING:LOGIC	◑	CLBs are only analyzed structurally (i.e., co-located cells), but not functionally (e.g., underutilized LUTs).
REQ:BIT:MODELING:INT	●	Interconnect faults (open, short, open-short, . . .) are extensively modeled.
REQ:BIT:MODELING:SPECIAL	○	Special-function blocks such as BRAM or 18×18 multiplier blocks that are present in the AMD Virtex-II and Spartan-II technologies [125, 126] are not addressed.
REQ:BIT:BITWISE_PREDICTION	●	*STAR* employs bitwise CMF prediction for TMR designs.
REQ:BIT:CHARACTERIZATION	○	Interconnect modeling is based on the vendor's configuration-to-netlist tool NCD2VHD, not characterization.
REQ:BIT:VALIDATION	◑	Validation by exhaustive fault injection as described in [110, 111] and by radiation experiments [112].
REQ:BIT:PVT	○	*STAR* does not address PVT.
REQ:BIT:DEVICES	○	AMD Virtex, Virtex-II, and Spartan-II are obsolete since 2013 [127]
REQ:BIT:RUNTIME	●	Design-dependent, 55-85 bit/s for *add8*, *add16*, *mul8*[111]

As a conclusion on *STAR*, Table 5 evaluates the tool's properties and capabilities against the requirements formulated in Section 3.1. It can be seen that despite *STAR* being able to predict critical interconnect bits, modeling of SEUs affecting CLBs, as well as related to BRAM and multiplier blocks are not sufficiently addressed in order to assess the criticality of all individual configuration bits. Furthermore, *STAR*'s modeling of interconnect faults is solely based on the device vendor's configuration-to-netlist back-annotation tool NCD2VHD as no characterization experiments have been performed. This (1) opens the possibility for faults not modeled by the manufacturer to affect the designs in unknown ways, and (2) provides only structural knowledge

about the faults, but not their behavior on the circuit level. While the predictions regarding CMF in TMR designs are validated using configuration-level fault injection and radiation experiments, this is only done on TMR designs that are expected to mask a significant portion of faults inherently. Furthermore, the authors do not address the possibility of PVT affecting the fault modeling or the outcomes of the faults. Finally, while the concepts described by the authors form the groundwork of several bit-level criticality prediction approaches, the device-specific descriptions and evaluations in [109, 110, 111, 112] themselves are limited to devices that have been out of production for more than a decade at the time of writing of this dissertation (AMD Virtex, Virtex-II, and Spartan-II) [127].

3.2.2 SEVAX

The soft error vulnerability analysis framework *SEVAX* is proposed in [113]. *SEVAX* aims at estimating the soft error vulnerability in terms of the number of potentially critical configuration bits in Xilinx Virtex-5 and Virtex-6 FPGAs. *SEVAX* uses the proprietary Xilinx Design Language (XDL) [128], a low-level design representation supported by AMD's obsolete *ISE* design tool, but not by their current *Vivado* design tool [129]. *SEVAX* extends BYU's *RapidSmith* Java framework with additional SEU analysis classes in order to generate an estimate of the number of critical configuration bits of a design. Different levels of analysis (pre-placement, post-placement, post-routing) are supported.

SEVAX applies different models to estimate the number of critical configuration bits of a design: For LUTs, the authors consider the usage of the LUT (i.e., the number of used inputs) to estimate the number of critical bits[19]. For BRAMs and DSPs, a static number of bits is returned for each utilized block of each type. For slice configuration bits (affecting all cells of a given type in a slice, e.g., all LUTs, flip-flops, multiplexers, or IO blocks), they assume a fixed number of bits per slice instantiating each resource type according to experimental data. For interconnect, they propose both a post-placement analysis and a post-routing analysis. In a post-placement analysis, where the exact routing of each net is yet unknown, the size of each net's bounding box is used to estimate the number of critical bits per net. In post-routing interconnect analysis, however, the individual Programmable Interconnect Points (PIPs) used by each net is available. Here, *SEVAX* iterates over all nets, and estimates the number of open-, short- and antenna-sensitive bits. Analyzing *SEVAX*'s source code[20], the following analyses are performed:

- The number of bits capable of causing open faults is estimated by summing up the numbers of PIPs making up each individual net (i.e., the number of open-sensitive bits equals the number of used PIPs).
- The number of bits capable of causing shorts is estimated by enumerating all PIPs used by the design that are able to short-circuit two nets – i.e., each PIP unused by the design that has both its input and output wires used, albeit by different nets and PIPs.
- The number of bits capable of causing antenna faults is estimated by counting all unused PIPs with unused input or output wires that are either able to drive the currently considered PIP's output wire, or are driven by the PIP's input wire.

Thus, despite estimating a number of critical configuration bits, only a rather coarse model for PIPs is applied (one that assumes that one individual configuration bit controls each PIP). Furthermore, the logical effects of bit flips on the signal(s) transported by the affected wire(s) – e.g., wired-AND,

[19] That is, for a j-input LUT, *SEVAX* estimates 2^j critical bits

[20] See `https://web.archive.org/web/20250912115932/https://github.com/unipieslab/sevax`, accessed 2026-01-05

wired-OR, etc. – are not considered. To consider these effects would require fault characterization experiments carried out on the targeted FPGA families, which are not reported in [113].

AMD's design tools are, in principle, capable of generating an estimate of SEU-sensitive bits via their *Essential Bits* feature – albeit pessimistically [88]. However, AMD's essential bits have the advantage of relating to individual configuration bits, allowing further analysis or targeted fault injection. On the other hand, *SEVAX* only aims at estimating the total *number* of potentially critical bits, not bit-wise criticality prediction. In the evaluation section of [113], *SEVAX*'s output is compared to the number of essential bits estimated by AMD ISE, divided by logic and interconnect bits, for several benchmark designs ranging from 225 to 5652 LUTs.

While *SEVAX* estimates a lower number of critical bits than AMD ISE, no validation experiments are carried out, such as using fault injection on a reference design. While [113] does not report the runtime it takes *SEVAX* to analyze a full design, the tool is publicly available. *SEVAX*'s runtime has been recorded using *bash*'s `time` builtin function on a Debian Linux 12 Workstation using an AMD Ryzen 7 5800X 8-core processor running at 3.8 GHz (bursting up to 4.7 GHz) and 64 GiB RAM. *SEVAX* performed (1) post-routing analysis on the placed and routed `barrel64` design supplied by the authors, with *SEVAX* determining the number of expected critical bits configuring LUTs and interconnect (subdivided into open, short, and antenna bits), as well as (2) essential bit analysis by dividing the 95948 essential bits reported by AMD ISE into interconnect, logic, CLB, DSP, and clock bits has been selected as the benchmark design. The runtime consumed by these tasks was measured three times, resulting in *real* runtimes of {14.5 s, 14.7 s, 14.7 s} and consumed *user* time of {17.3 s, 17.2 s, 17.3 s}. Relating the *real* time to the number of essential bits in the design gives a time analysis speed of approximately 6556 bit/s, which appears to be sufficient for use in DSE fitness functions. It must be noted, however, that *SEVAX* arrives at its post-routing estimates purely by counting affected PIPs and does not apply bit-level fault modeling.

Table 6 summarizes the evaluation of *SEVAX* against the bit-level requirements established in Section 3.1.

Table 6: Evaluation of *SEVAX* [113] against Bit-Level Requirements

Requirement	Evaluation	Explanation
REQ:BIT:MAPPING	○	No individual mapping between configuration bits and the netlist is performed.
REQ:BIT:MODELING:LOGIC	◑	LUTs are partially functionally modeled by considering their number of inputs to estimate critical bits.
REQ:BIT:MODELING:INT	◑	Interconnect faults (open, short, open-short, ...) are modeled on a coarse, not necessarily bit-accurate level.
REQ:BIT:MODELING:SPECIAL	◑	Bits belonging to special-function blocks such as BRAM or DSP multiplier blocks are considered in routing and logic bit estimation. However, no fault modeling occurs.
REQ:BIT:BITWISE_PREDICTION	○	No per-bit consequences are predicted.
REQ:BIT:CHARACTERIZATION	○	Interconnect modeling is based on the vendor's PIP connectivity information alone, not characterization.
REQ:BIT:VALIDATION	○	As no per-bit consequences are predicted, per-bit validation is not performed either.
REQ:BIT:PVT	○	*SEVAX* does not address PVT.
REQ:BIT:DEVICES	○	Xilinx has discontinued several Virtex-5 and Virtex-6 in 2023 [130]. Furthermore, the underlying XDL toolflow is not supported by the current AMD Vivado design tool [129].
REQ:BIT:RUNTIME	●	Evaluations by the author of this dissertation resulted in *SEVAX* runtimes in the range of tens of seconds for a substantial design.

3.2.3 Rad-SyMPaR/VERI-Place

In [114], the authors propose the framework *Rad-SyMPaR* ("Radiation Synthesis, Mapping, Place and Route"). *Rad-SyMPaR* is capable of directing synthesis and implementation to generate robust circuits concerning configuration-related bit flips. As part of this process, *Rad-SyMPaR* estimates the soft-error rate of the design from several design stages (netlist-level, post-routing, and bitstream-level). The bit-level soft-error prediction component integrated in *Rad-SyMPaR* is referred to as *VERI-Place*. This component appears to be – at least considering a significant overlap in authorship – related to *STAR* (see Section 3.2.1).

[114] briefly covers the interconnect architecture and interconnect-related configuration bit mapping of devices supported by *Rad-SyMPaR*: Microchip RT-ProASIC3 and RTG4, AMD Ultrascale+, and NanoXplore NG. This forms the basis of modeling the circuit-level consequences of interconnect bit flips, as described in slightly more detail for AMD 7 Series devices in [115]. Also, mapping and modeling of resources such as LUTs, BRAM, BRAM interconnect, and flip-flops are supported [115], while other resources such as DSP blocks are not mentioned. However, it is unclear to what extent the inner workings of these resources (e.g., the number of "used" vs. "don't care" configuration bits in under-utilized LUTs) is modeled. While fault modeling is not described in explicit detail in [115, 114], it is assumed due to the personnel overlap that *VERI-Place* employs the same

mitigation strategies pioneered by *STAR*. The bit-level soft-error prediction component *VERI-Place* is validated experimentally in [116]. However, the validation focuses on the overall soft error rate predicted by VERI-Place rather than a validation of the criticalities of individual configuration bits as predicted by *VERI-Place*. Neither publication describing *Rad-SyMPaR* [115, 114] or *VeriPlace* [116] report experimental results on the run-time of the bitstream-level soft-error-rate prediction, while [116] describes it being *"orders of magnitude faster than fault injection"*. It must be noted that neither *Rad-SyMPaR* nor *VERI-Place* are publicly available tools.

Table 7 summarizes an evaluation of *Rad-SyMPaR/VERI-Place* against the requirements for bit-level tools defined in Section 3.1.

Table 7: Evaluation of *Rad-SyMPaR/VERI-Place* [114, 115, 116] against Bit-Level Requirements

Requirement	Evaluation	Explanation
REQ:BIT:MAPPING	●	*VERI-Place* extracts bitstream-to-netlist mapping from vendor-generated placement and routing information.
REQ:BIT:MODELING:LOGIC	◑	CLBs are assumed to be structurally analyzed as in *STAR*. It is unclear if, e.g., LUTs are functionally modeled (e.g., underutilized LUTs).
REQ:BIT:MODELING:INT	●	Interconnect faults (open, short, open-short, ...) are assumed to be modeled in the same way as in *STAR*.
REQ:BIT:MODELING:SPECIAL	◑	BRAM is explicitly supported, DSP blocks are not mentioned.
REQ:BIT:BITWISE_PREDICTION	●	*VERI-Place* predicts bit-level criticality and soft error rate.
REQ:BIT:CHARACTERIZATION	○	No characterization experiments are described in [114, 115, 116].
REQ:BIT:VALIDATION	◑	Validation of predicted fault rate [116], but not of bit-wise predictions.
REQ:BIT:PVT	○	*VERI-Place* does not address PVT.
REQ:BIT:DEVICES	●	AMD 7 Series and UltraScale+, Microchip RT-ProASIC3 and RTG4, NanoXplore NG.
REQ:BIT:RUNTIME	–	No analysis of runtime is performed in [114, 115, 116]

3.2.4 PyXEL

PyXEL is a tool described by [46] to work on bitstreams and design representations generated by AMD's ISE and Vivado design tools. Its task is to analyze and modify bitstreams. This allows, for example, to enable or disable specific PIPs in the FPGA. To demonstrate the capabilities of their tool, the authors of [46] perform a somewhat systematic fault injection analysis to characterize the effect of configuration bit flips on one hundred PIPs in an Artix 7 FPGA. The authors implement a simple test design that routes two test signals via specific PIPs. They then perform fault injection by modifying the placed and routed design via *PyXEL* to enable or disable PIPs assumed capable of causing short-circuits between the two nets or opens in them. These characterization experiments reveal that the analyzed Artix 7 FPGAs exhibits stuck-at-1 in the case of an open fault and mostly wired-AND and wired-OR behavior in the case of two conflicting wires. The authors, however, do

not describe in detail how they derived their bitstream-level models that allow conflict or open faults to be injected. On the level of implementation described in [46], PyXEL, however, is not described to be able to map the bitstream back to the design's netlist or to predict the criticality of individual bits.

A later publication [117] describes *PyXEL* as a tool capable of orchestrating fault identification campaigns, analyzing and decoding the bitstream, and performing custom placement and routing to improve design robustness. In terms of bitstream analysis and decoding, the authors note that *PyXEL* is able to map the configuration bits found in the bitstream to FPGA tiles and to decode these tile configurations to reveal their functionality. The authors especially note that *PyXEL* is able to decode interconnect bits to PIPs with an accuracy of at least 98% for AMD Ultrascale+ FPGAs. It remains unclear from [117] if *PyXEL*, given a bitstream of a design, is capable of predicting the critical bits according to fault models. However, as it is described to be able to emulate specific faults such as open routing and corruption of LUTs content, as well as localize faults observed during radiation campaigns, it may be capable of prediction. Neither [46] nor [117], however, describe how precisely the authors arrive at their bit-level fault models.

In a case study described in [117], *PyXEL* is used to direct fault injection experiments to determine the fault sensitivity of a case-study design. Faults are injected into both a conventionally placed TMR variant and an isolated TMR variant of the same design – a Coordinate Rotation Digital Computer (CORDIC) IP core connected to an ARM AXI on-chip bus. The placement of the isolated TMR variant was also generated by *PyXEL*.

Table 8 summarizes the evaluation of *PyXEL* against the requirements for bit-level analysis approaches described in Section 3.1. To conclude, *PyXEL* can be seen as a comprehensive approach capable of bitstream analysis, bitstream manipulation, and fault injection. While characterization experiments are conducted in [46] for AMD 7 series devices, only 100 PIPs of the 3700 present in each interconnect tile in these FPGAs are characterized, leaving room for non-characterized behavior. Furthermore, antenna faults are not considered by *PyXEL*. Except for the characterization experiments described in [46], however, not much of the approach's inner workings or its underlying fault models is revealed by [46, 117]. Whether *PyXEL* is capable of predicting the criticality of individual configuration bits remains unclear. While interconnect seems to be modeled sufficiently, little is known about *PyXEL*'s modeling of logic resources and special-function blocks, and its source code is not publicly available. In the same way as the previously discussed approaches, *PyXEL* does not address PVT effects in its analyses or characterization experiments. It shall be noted that as in the case with *Rad-SyMPaR/VERI-Place*, a considerable personnel overlap exists among authors of publications concerning *PyXEL*, *Rad-SyMPaR/VERI-Place*, and *STAR*.

Table 8: Evaluation of *PyXEL* [46, 117] against Bit-Level Requirements

Requirement	Evaluation	Explanation
REQ:BIT:MAPPING	●	*PyXEL* is described as providing an "automatized flow for associating configuration data with hardware resources", and module-aware fault injection, requiring bitstream-to-fault mapping.
REQ:BIT:MODELING:LOGIC	◑	While "LUT corruption" is supported for fault injection, the extent of LUT modeling in the case of bitstream analysis is unclear.
REQ:BIT:MODELING:INT	●	Interconnect bits are associated with PIPs, within which open and short faults are modeled.
REQ:BIT:MODELING:SPECIAL	○	BRAM or DSP blocks are not mentioned in [46, 117]
REQ:BIT:BITWISE_PREDICTION	○	It remains unclear from [117] if *PyXEL* is capable of bit-wise criticality prediction.
REQ:BIT:CHARACTERIZATION	◑	[46] describes somewhat systematic characterization experiments for open and short faults. However, only a small subset of the target architecture's PIPs were tested, and antenna faults were not considered.
REQ:BIT:VALIDATION	◑	While *PyXEL* is described to correctly associate 98% of PIPs in evaluated benchmark designs, no reliability prediction and thus no bit-by-bit validation experiments are reported in [46, 117].
REQ:BIT:PVT	○	*PyXEL* does not address PVT.
REQ:BIT:DEVICES	◑	PyXEL supports AMD 7 Series [46] and UltraScale+ [117] devices, but no devices from other vendors.
REQ:BIT:RUNTIME	●	PyXEL is described to be able to associate bitstream to PIPs in UltraScale+ FPGAs in "only a few minutes per design"[117]

3.2.5 ACME and ACME-2

The tools *ACME* ("Automatic Configuration Memory Error-injection") [118] and *ACME-2* [119] aim at translating the *essential* bits data provided by AMD Vivado as an .ebd file (see Section 2.3) into bit addresses to be used for fault injection. ACME [118] filters the .ebd file by extracting just the configuration frames (frame, bit, and word addresses) where the DUT is placed in the target FPGA using a *"pBlock"* placement constraint. ACME-2 [119] improves the filtering of the .ebd file to use both the corresponding configuration frame addresses and a word offset within the frame. In the context of the targeted AMD FPGA technologies, this allows more precise fault injection. *ACME* and *ACME-2* furthermore use statistical fault injection: They establish a confidence level and an acceptable error margin, and accordingly derive the necessary sample size of bits to be tested using fault injection to estimate the critical share of the design's essential configuration bits. However, neither *ACME* nor *ACME-2* perform in-depth modeling of the effect of individual bit flips on logic or interconnect level or mapping of bits (or bit flips) to the effects in the design's

netlist. The tools, however, map configuration bit addresses to coarse placement regions to perform targeted fault injection experiments. AMD's *pBlock* placement constraint toolflow works based on hierarchical modules assigned to a placement region. Thus, it may be argued that *ACME* performs a severely limited degree of mapping of bitstream to netlist.

Table 9: Evaluation of *ACME* [118] and *ACME-2* [119] against Bit-Level Requirements

Requirement	Evaluation	Explanation
REQ:BIT:MAPPING	◑	*ACME* and *ACME-2* coarsely map bit addresses to FPGA coordinates of pBlocks, and thus to the netlist cells placed there.
REQ:BIT:MODELING:LOGIC	○	No bit-level modeling of faults is done.
REQ:BIT:MODELING:INT	○	No bit-level modeling of faults is done.
REQ:BIT:MODELING:SPECIAL	○	No bit-level modeling of faults is done.
REQ:BIT:BITWISE_PREDICTION	○	No criticality prediction is done.
REQ:BIT:CHARACTERIZATION	○	As no bit-level modeling of faults is performed, there is no necessity for characterization experiments.
REQ:BIT:VALIDATION	○	No validation experiments are performed.
REQ:BIT:PVT	○	*ACME* and *ACME-2* do not address PVT.
REQ:BIT:DEVICES	◑	*ACME* and *ACME-2* support AMD 7 Series, UltraScale, and UltraScale+ devices, but no devices from other vendors.
REQ:BIT:RUNTIME	–	No analysis of runtime is performed in [118, 119]

No experiments are conducted in [118, 119] to validate that ACME or ACME-2 exclude only non-critical bits from fault injection experiments. This may be due to the fact that the task of these tools mainly is to extract the information out of the *essential bits* file generated by AMD Vivado, not reliability prediction per se. However, the effect of filtering is verified experimentally in [118]: By performing the same number of random fault injections on (1) bits sampled without *ACME*'s filtering and (2) bits sampled with the help of *ACME*, it can be seen that significantly more critical bits of a *pBlock* of interest can be determined with the *ACME*'s help. In [119], *ACME* and *ACME-2* are compared to each other. Despite offering no bit-level fault modeling or criticality prediction, Table 9 summarizes *ACME*'s capabilities with respect to the bit-level requirements from Section 3.1.

3.2.6 BAFFI

The tool *BAFFI* ("Bit-Accurate FPGA Fault Injection") is proposed in [120]. This tool analyzes an FPGA bitstream for Xilinx 7 Series and Ultrascale+ devices. From this bitstream, it extracts a mapping between configuration bits and netlist elements for LUTs, flip-flops, LUTRAM (i.e., distributed RAM: LUTs used as RAM), and BRAM. These resources can then be targeted using fault injection in a feature referred to as "bit-accurate fault load generation". On the other hand, *BAFFI* supports a mode of fault-load generation similar to *ACME* and *ACME-2* (see 3.2.5), in which to-be-injected faults are generated for every bit in the *essential bits* file generated by the AMD Vivado implementation tool, filtered by the location of a *pBlock* to be targeted.

While *BAFFI* is able to map bits from the bitstream to netlist elements for LUTs, flip-flops, LUTRAM, and BRAM, it does not offer provisions for targeted interconnect fault injection. Bits configuring other resources, including DSP blocks and the important category of interconnect-related configuration

bits, may be targeted using *BAFFI*'s *pBlock*-filtered fault load generation mode. However, they are neither cross-referenced to nets in the netlist nor is their effect on the implemented circuit modeled. In general, *BAFFI* provides no provisions for predicting the criticality of certain configuration bits or for modeling the effect of their flipping on the circuit's functionality. This also includes "don't care"-bits in LUTs that are not modeled by *BAFFI*. *BAFFI* is publicly available as part of the *DAVOS* toolkit[21].

While the overall duration of a *fault injection experiment* is evaluated in [120], this includes the duration of executing the tested design's test application and thus does not serve as an evaluation of *BAFFI*'s bitstream analysis runtime. However, as *BAFFI* is publicly available, this data has been reproduced by the author of this dissertation[22]. Only the runtime of the "fault-load generation" part, as relevant to this work's bit-level configuration fault modeling part, is reported here. As no designs have been supplied with *BAFFI* by the authors, the evaluation is performed using four benchmark designs by the author of this dissertation. All designs were implemented for an AMD Artix 7 xc7a35t FPGA and were available to *BAFFI* as an AMD Design Checkpoint (.dcp) file – i.e., a proprietary archive format used by AMD Vivado to store a design's implementation state.

For all designs, the runtime includes the steps of (1) loading the design from an AMD Vivado design checkpoint and generating fault lists for (2) all bits, (3) LUT bits only, (4) flip-flops, and (5) BRAM. The resulting data can be seen in Table 10. It can be seen that *BAFFI* analyzes all designs in approximately one minute and that analysis time is unrelated to the number of essential bits. However, it needs to be noted that *BAFFI* neither performs complex modeling of interconnect (or other) faults nor traces faults through the design's routing graph, which benefits fast execution times.

Table 10: Runtime Analysis of *BAFFI* [120]

Design	Target FPGA	Essential Bits	Runtime (min)				Speed (bit/s)
			Run 1	Run 2	Run 3	Avg.	
benchmark_1	xc7a35t	21351	1.11	1.07	1.14	1.11	321.3
benchmark_2	xc7a35t	16050	1.12	1.89	1.15	1.39	192.8
benchmark_3	xc7a35t	73660	1.07	1.13	1.11	1.10	1111.5
benchmark_4	xc7a35t	62365	1.07	1.67	1.12	1.12	928.9

Table 11 summarizes an evaluation of *BAFFI* against the requirements proposed in Section 3.1. The authors validate *BAFFI*'s ability to pinpoint specific resources by comparing the results of fault injection experiments carried out using *BAFFI* with simulation-based fault injections on RTL and netlist level. The experiments target three soft-core CPUs: An 8051 core, an AVR core, and an AMD MicroBlaze core. In particular, faults are injected by generating (1) bit flips in flip-flops, BRAM, and LUTRAM for all three cores and (2) bit flips in logic-related LUTs for the 8051 core. Statistical sampling is employed for all experiments according to a given confidence level and margin of error. Thus, limited validation for *BAFFI*'s ability to pinpoint configuration-memory bit flips in the bitstream exists: The LUT-related failure rate for the 8051 estimated from fault injection experiments directed by *BAFFI* matches that derived from gate-level fault simulation.

[21] See `https://web.archive.org/web/20250912140105/https://github.com/IlyaTuzov/DAVOS`, accessed 2026-01-05

[22] Commit 33f3bd9 of *BAFFI* was analyzed; see `https://web.archive.org/web/20250912140226/https://github.com/IlyaTuzov/DAVOS/commit/33f3bd90f7b7331ec20a7397149edb2faaa14be2`, accessed 2026-01-05

Table 11: Evaluation of *BAFFI* [120] against Bit-Level Requirements

Requirement	Evaluation	Explanation
REQ:BIT:MAPPING	◑	*LUT*, flip-flops, BRAMs and LUTRAMs are mapped to the netlist. Other resources (e.g., interconnect and DSP blocks) are excluded.
REQ:BIT:MODELING:LOGIC	○	No bit-level modeling of faults is done.
REQ:BIT:MODELING:INT	○	No bit-level modeling of faults is done.
REQ:BIT:MODELING:SPECIAL	○	No bit-level modeling of faults is done.
REQ:BIT:BITWISE_PREDICTION	○	As no bit-level modeling of faults is done, bit-level criticality cannot be predicted.
REQ:BIT:CHARACTERIZATION	○	As no bit-level modeling of faults is performed, there is no necessity for characterization experiments.
REQ:BIT:VALIDATION	◑	Limited validation on bit flips in LUTs is performed.
REQ:BIT:PVT	○	*BAFFI* does not address PVT.
REQ:BIT:DEVICES	◑	*BAFFI* supports AMD 7 Series and UltraScale+ devices, but no devices from other vendors.
REQ:BIT:RUNTIME	●	Evaluations by the author of this dissertation resulted in *BAFFI* runtimes in the range of several minutes per design.

3.2.7 BFAT

Originally intended for the analysis of faults recorded during radiation experiments and developed in part in parallel to the writing of this dissertation, [121] proposes the fault analysis tool *BFAT*. The source code for this tool is available publicly[23]. In the theme of post-radiation fault analysis, *BFAT* works backward from a list of (combinations of) critical bits' addresses observed during radiation fault injection for AMD 7 series FPGAs. It then applies bitstream fault modeling techniques to determine the resources, netlist elements, and consequences associated with each fault. For interconnect-related bits, *BFAT* applies a model described in more detail in [131] to determine if the fault's consequences will be an open, short, or other interconnect fault. Furthermore, it identifies the cells connected to the interconnect resource affected by the fault. For LUT bits, the LUT is modeled, including unused (statically driven) inputs to determine "don't care" bits. Other resources are identified and linked to netlist elements but not modeled in detail.

Incrementally improved work based largely on [121] is published in [122] by the same authors as well as in a corresponding PhD dissertation [123] by the first author of [121, 122]. *BFAT* is discussed here as well, and based on the results applying *BFAT*'s models to the observed faults, techniques to improve the TMR separation are proposed.

However, as *BFAT* appears to be targeted at "root-cause" analysis of bits already observed to be critical, neither [121] nor [122] apply and validate it in the "forward" direction: That is, [121, 122] do not report experiments validating that bits for which *BFAT* cannot find any detrimental effect on the analyzed circuit are not critical under fault injection.

Regarding the characterization of interconnect faults, [131] performs small-scale experiments by generating several routing configurations that allow manually observing the effects of conflict faults

[23] See `https://web.archive.org/web/20250909162749/https://github.com/byuccl/bfat`, accessed 2026-01-05

by setting the input wire's values via switches and observing the output wires using LEDs. The goal of these experiments appears to be to identify whether short-type interconnect faults affect both involved wires or only one of them. The conclusion from these characterization experiments is that in general, in AMD 7 Series FPGAs, only the "target", i.e., the wire downstream of a PIP, is affected in general interconnect. In global interconnect (e.g., clock signals), both wires (one of them being necessarily upstream of the fault) are affected by a short. However, in neither the general nor global interconnect case, an analysis regarding the logical effect on these wires is reported – such as the Wired-AND or Wired-OR behavior as observed in [46]. It is assumed that *BFAT*'s fault models are partly based on these limited characterization experiments assessing the behavior of a handful of PIPs.

In the experimental section of [121], the authors replicate faults observed during radiation experiments on a VexRiscV CPU using partial-reconfiguration-based fault injection after analysis using *BFAT*. They implicitly acknowledge the possibility of PVT effects in their experiment design by playing back the faults observed in radiation experiments three times on three identical FPGA instances (for a total of nine runs). Of the 27 faults observed in radiation experiments, only 17 caused a failure in all three devices in all three runs. Four caused a failure only on a subset of the devices, and one caused a failure on all devices but not in all runs. However, while the possibility of PVT appears to be considered in these experiments, *BFAT* does not offer provisions for including these effects in its modeling.

While no analysis of *BFAT*'s runtime is published in [121, 122, 123], its runtime can be assessed fairly easily, given the tool's source code is publicly available. This has been done by the author of this dissertation[24]. This analysis has been carried out using five benchmark designs: (1) *bfat_demo_1* is supplied by *BFAT*'s authors and contains the VexRiscV SoC design used in [121, 122], and (2) four benchmark designs by the author of this dissertation. Along with *bfat_demo_1*, the authors also supplied a list of faults recorded during radiation analysis, including 30 SEUs and 6 MBUs. These faults were used for the runtime analysis of *bfat_demo_1*. Vivado's *Essential Bits* feature was used for the other designs to generate a list of faults analyzed. Three runtime measurements were taken for each design using *bash*'s `time` builtin on a Debian Linux 12 Workstation using an AMD Ryzen 7 5800X 8-core processor running at 3.8 GHz (bursting up to 4.7 GHz) and 64 GiB RAM. The *user* time is used as tool runtime. Furthermore, the tool's analysis speed was calculated as the quotient of the number of analyzed bits and the elapsed *user* time, resulting in a *Bits Per Second (bit/s)* metric.

As shown in Table 12, analyzing a design's essential bits in their entirety takes *BFAT* a considerable amount of time. Analyzing the fault lists supplied by the authors for the design *bfat_demo_1* led to an average speed of only 0.08 bit/s, possibly due to the large – and thus more complicated – xc7a200t FPGA device used. Analyzing the designs *benchmark_1*, *benchmark_2*, and *benchmark_4*, *BFAT* reaches an average analysis speed of approximately 1.3 bit/s. This falls into the speed range of fault injection toolflows: For example, [120] reports FPGA-based fault injection rates (referred to as "*FFI*") between 0.87 and 38 bit/s depending on the complexity of the tests to be conducted under fault injection.

The design *benchmark_3* is an outlier compared to the other benchmarks provided by the author of this dissertation: Here, *BFAT* both reaches an average analysis speed of 10 bit/s and produces a considerably smaller share of identified faults within the analyzed bits. Analyzing the tool's source code, the author of this dissertation identified a bug in the tool's bitstream parser. This bug results in incorrect bit data being analyzed if user-defined bitstream IDs – the `USR_ACCESS` value in AMD terminology – starting with the same 3-bit pattern as a "Type-2" bitstream packet, usually containing

[24] Commit 02fdfd2 of *BFAT* was analyzed; see `https://web.archive.org/web/20250912123423/https://github.com/byuccl/bfat/commit/02fdfd216ed9bdda8f3ffcc7bc6259ce92c204d8`, accessed 2026-01-05

Table 12: Runtime Analysis of *BFAT* [121, 122, 123]

Design	Target FPGA	Analyzed Bits	Identified Faults	Runtime (min)				Speed (bit/s)
				Run 1	Run 2	Run 3	Avg.	
bfat_demo_1	xc7a200t	42	42	8.9	9.0	8.9	8.9	0.08
benchmark_1	xc7a35t	21351	20076	299.3	294.3	297.4	297.0	1.20
benchmark_2	xc7a35t	16050	15164	218.0	214.0	214.3	215.5	1.24
benchmark_3	xc7a35t	73660	34883	124.1	120.8	122.4	122.4	10.03
benchmark_4	xc7a35t	62365	60244	787.8	786.8	786.2	786.9	1.32
*benchmark_3**	xc7a35t	73660	68671	949.1	948.8	951.6	949.8	1.29

the device's configuration [30]. The author of this dissertation supplied *BFAT*'s authors with a patch for this bug[25]. With this patched tool version[26], *benchmark_3* was analyzed once again, reported as *benchmark_3** in Table 12. It shall be noted that, before the contribution made by the author of this dissertation, *BFAT* evidently did not process all supplied bitstreams correctly. Furthermore, it did not notify the user of any inconsistencies while producing an incorrect list of potentially critical bits not present in the "essential bits" file generated by AMD Vivado.

In summary, as seen in Table 13, *BFAT* appears to be a promising candidate for bit-level fault analysis, even though it is at the time of writing not geared towards "forward-working" fault criticality prediction[27], which is additionally hampered by slow tool runtimes.

[25] See `https://web.archive.org/web/20250909162406/https://github.com/byuccl/bfat/pull/53`, accessed 2026-01-05

[26] Commit 00b1a15a containing the patch by the author of this dissertation was used; see `https://web.archive.org/web/20250909162728/https://github.com/cfib/bfat/commit/00b1a15a41a46fe2466557174d677c731c54ec00`, accessed 2026-01-05

[27] The related *issues* are open since 2022; see `https://web.archive.org/web/20250912131927/https://github.com/byuccl/bfat/issues/30` and `https://web.archive.org/web/20250912132340/https://github.com/byuccl/bfat/issues/44`, accessed 2026-01-05

Table 13: Evaluation of *BFAT* [121, 122, 123] against Bit-Level Requirements

Requirement	Evaluation	Explanation
REQ:BIT:MAPPING	●	Netlist elements affected by each fault are identified.
REQ:BIT:MODELING:LOGIC	●	LUT-content faults are modeled including under-utilized LUTs.
REQ:BIT:MODELING:INT	●	Open and Short interconnect faults are fully modeled, including shorts with unconnected nodes.
REQ:BIT:MODELING:SPECIAL	◑	BRAM, DSP, and IO-related bits are associated with netlist-level resources, but not modeled.
REQ:BIT:BITWISE_PREDICTION	◑	Prediction is possible in principle but is not yet implemented as of the time of writing and is hampered by long tool runtimes.
REQ:BIT:CHARACTERIZATION	◑	Small-scale, manually administered characterization experiments are reported in [131].
REQ:BIT:VALIDATION	○	No validation experiments are reported.
REQ:BIT:PVT	○	*BFAT* does not address PVT.
REQ:BIT:DEVICES	◑	*BFAT* supports AMD 7 Series devices and no devices from other vendors.
REQ:BIT:RUNTIME	○	Own evaluations resulted in *BFAT* runtimes in the range of several hours per design.

3.2.8 Bricas et al.

Developed in parallel to the writing of this dissertation, a bitstream-based fault prediction approach is presented in [124]. The approach is targeted at AMD 7 Series devices and based – in the same way as *BFAT* – on the bitstream documentation for these devices provided by *Project XRAY*[28]. It models open and short-circuit (referred to as “bridge” faults with used and unused wires in [124]) in the target device’s interconnect, routing multiplexers within CLBs, faults in LUTs according to the usage of its inputs, as well as LUT and flip-flop “parameters” (i.e., use of LUTs as shift registers, LUTRAM, or combinatorial function and use of register cells as flip-flops or latches). BRAM and DSP resources are not explicitly modeled.

To derive the behavior of interconnect faults, the authors perform limited characterization experiments similar to those done in the course of this dissertation [3] by routing wires from a LUT input buffer to a LUT output buffer via different, specific PIPs to be tested. The authors tested the effect on 50 different interconnect multiplexers (referred to as “PIP junctions” in [124]). In these experiments, bridge faults with used and unused wires were evaluated. However, as the authors note, each interconnect tile in AMD 7 Series devices provides 172 interconnect multiplexers. In total 3636 direct connections between 182 input wires and 172 output wires (i.e., the output wires of the interconnect multiplexers) can be configured in an interconnect tile provided by this technology. Additionally, several “pseudo PIPs” are used to configure certain defined connections in the absence of any other configuration of the respective interconnect resource. This includes driving the 64 input multiplexers (`IMUX`, `FAN_ALT`, and `BYP_ALT` wires) of logic tile input multiplexers to a static ’1’. There are 3636 possible SEU-induced open faults, and, due to the structure of

[28] See `https://web.archive.org/web/20250115214812/https://f4pga.github.io/prjxray-db/`, accessed 2026-01-05

the interconnect tile, 21900 possible SEU-induced shorts. It is not reported in [124] how many of these faults form the basis of the fault characterizations. This limited characterization causes some deficiencies in the derived model for the interconnect multiplexers in the targeted FPGA technology, modeled as a block of “hidden logic” shown in a figure, but not discussed, in [124]. Furthermore, the data is only presented in a very condensed form (i.e., the percentage of faults exhibiting a certain behavior), while a differentiation by source and destination wire types may be beneficial.

Nevertheless, it is assumed that the characterization experiments in [124] are more extensive than those described in [131]; they are assumed to be comparable to those described in [46]. The authors develop a fault model for PIPs from these characterization experiments. This model assumes that shorts between two used wires result in a wired-AND behavior, that shorts between a used and an unused wire have no effect, and that open faults cause a stuck-at-1 fault. This, however, does not consider the effects determined in earlier works such as [3] by the author of this dissertation, where shorts with unused wires have been observed to be critical under certain conditions (e.g., shorts with the outputs of unused LUTs, certain outputs of unused BRAM and DSP resources). The authors furthermore develop a fault model for intra-slice multiplexers from the configuration bit patterns.

The authors combine their fault models and activation conditions with RTL simulation results to predict the criticality of each configuration bit flip. The circuit topology resulting from each fault to be analyzed is first recreated, and the net values are imported from the simulation. Afterward, the fault is propagated through the circuit’s netlist to determine whether it reaches a primary output of the circuit over the analyzed simulation duration, or is logically blocked by the values carried by the other nets in the analyzed simulation trace. This is only possible if bitstream-to-netlist mapping is performed by this approach.

The authors compare the numbers of critical bits predicted by their tool to those identified using partial-reconfiguration-based fault injection to validate their predictions. This is done based on a Finite Impulse Response (FIR) filter design implemented on an AMD Artix 7 xc7a50t FPGA. The essential bits reported by AMD Vivado formed the basis for the fault injection experiments. According to the authors, their estimated number of critical bits lies within 1% of the bits reported by fault injection. However, they do not provide detailed bit-by-bit validation of their criticality predictions. Furthermore, neither PVT effects nor environment temperature during their characterization or validation experiments is discussed. However, the authors report the execution time of their approach, which lies in the order of 10 minutes for small designs and grows linearly with the number of cells.

Table 14 summarizes the fulfillment of the bit-level requirements established in Section 3.1 by the approach presented in [124]. Overall, it is a remarkably capable and well-performing fault prediction approach with few shortcomings related to modeling special resources such as DSPs, limited characterization, and validation experiments. Furthermore, like the other considered bit-level analysis and prediction approaches, it also lacks consideration of PVT effects. Nevertheless, the simulation-based analysis provided by this approach allows the integration of information about logical masking, thereby increasing prediction accuracy. However, this requires simulation traces to be available, and the generalizability of the masking-related prediction may depend on the simulation coverage achieved by the testbench.

Table 14: Evaluation of [124] against Bit-Level Requirements

Requirement	Evaluation	Explanation
REQ:BIT:MAPPING	●	Netlist elements affected by each fault are identified for simulation-based prediction.
REQ:BIT:MODELING:LOGIC	●	LUT-content faults are fully modeled.
REQ:BIT:MODELING:INT	●	Open and Short interconnect faults are modeled, including shorts with unconnected nodes (i.e., input antenna faults)
REQ:BIT:MODELING:SPECIAL	○	Faults affecting DSPs are mentioned as future work, BRAM is not mentioned.
REQ:BIT:BITWISE_PREDICTION	●	Simulation results are combined with fault models to predict criticalities.
REQ:BIT:CHARACTERIZATION	◑	Limited characterization experiments for interconnect faults are reported.
REQ:BIT:VALIDATION	◑	The predicted *number* of critical bits is validated, while bit-by-bit criticality prediction validation is not reported.
REQ:BIT:PVT	○	[124] does not address PVT.
REQ:BIT:DEVICES	◑	[124] supports AMD 7 Series devices and no devices from other vendors.
REQ:BIT:RUNTIME	●	Runtimes on the order of tens of minutes are reported for small designs.

3.2.9 Other Related Work

This section describes works related to fault modeling and injection while not constituting full-fledged approaches to predicting or even analyzing a full design for its fault tolerance in contrast to section 3.2. Nevertheless, these analyses contain valuable insights that are integrated in part into the toolflow described in this dissertation.

Fault Modeling

The basics of FPGA fault modeling are elaborated in [27] based on *STAR* (see Section 3.2.1). The authors divide SEUs in FPGAs into three fault classes:

1. SEUs in user-defined storage elements such as flip-flops and latches.
2. SEUs concerning the content but not the functionality of BRAMs.
3. SEUs affecting configuration memory.

The effects of faults in the initialization content of LUTs, as well as interconnect-related faults with consequences of generating open, conflict, input/output antenna, bridge and tolerant faults are discussed. The influence of the FPGA technology and configuration encoding – e.g., the influence of decoded and non-decoded multiplexers – on the possibility of an SEU causing either of these interconnect fault types is discussed. Interconnect-related fault types (e.g., open, antenna, conflict) are also discussed in [132] for AMD Virtex FPGAs. The authors aim to predict the number of sensitive PIPs from the number of PIPs used by the design using curve fitting. Thus, no analytical modeling is performed, causing the approach to be unable to pinpoint individual potentially critical bits.

Despite not offering full bitstream analysis or criticality prediction approaches, several works preceding tools such as *PyXEL* (see Section 3.2.4) or *BFAT* (see Section 3.2.7) discuss modeling and characterization approaches that may have been relevant in developing these tools. In [133], a fault injection campaign is conducted on an AMD Virtex 4 FPGA where several peripherals for a hard-macro PowerPC CPU are implemented. The paper includes a fault model for Virtex 4 interconnect faults based on an assumed transistor-level model of the device's interconnect multiplexers. The authors also divide the critical configuration bits according to their function (logic, interconnect-open, interconnect-short, etc.) as is done by Section 3.2.1 for older FPGA technologies. Concerning interconnect faults, fault modeling for two-stage pass-transistor-based interconnect multiplexers is discussed, forming a basis for modeling the interconnect of more advanced FPGA technologies. The authors also divide short circuit faults in the interconnect into "buffer shorts" and "bridging shorts", where the former is related to input antenna behavior (the erroneously connected source is not driven by the configuration). The latter is a "conflict" fault in the terminology of STAR's and [27]. In the experiments conducted by the authors, they discovered that the particular tested FPGA technology exhibits wired-AND behavior if the second stage of the multiplexer is affected and wired-OR behavior if the first stage is affected, hinting at transistor-level implementation differences between the two stages. Using a model similar to [133], in [134], a model of a two-stage FPGA routing multiplexer from an unnamed FPGA technology is described. Using this model, the authors demonstrate how a single-bit flip can cause multiple shorts in this multiplexer. In a TMR design whose implementation does not consider these effects, this may cause TMR violations if it affects more than one of its three replica. The experimental part of the work analyzes the proposed mitigation technique for CMFs under fault injection via JTAG and radiation fault injection.

A further modeling approach targeting AMD's Virtex 4 architecture is discussed in [135]. The authors describe a way of analyzing the design's bitstream (via bitstream) to parse out open, short, and antenna-vulnerable bits, assuming that a single configuration bit controls each PIP. They analyze bitstreams for several implemented benchmark circuits from the collections published at the IEEE International Symposium on Circuits and Systems (ISCAS) in 1985 and 1989 (commonly referred to as "ISCAS'85" and "ISCAS'89" benchmarks) [136, 137]. The authors report the number of interconnect bits that, when flipped, cause either of the discussed interconnect fault effects. The most common interconnect fault effect in all analyzed designs is *antenna* followed by *open*. The least common type of fault reported in [135] are bits that cause short circuits between two nets when flipped. Combining these results with the assumption that the consequences of its flipping influence the criticality of a bit may lead to more precise criticality prediction. This is underlined by a limited fault-injection-based validation and characterization experiment reported by the authors: The individual criticalities for five bits in the c17 ISCAS'85 [136] design are provided, containing four critical and one non-critical bit. The cause for the benignity of the non-critical bit may be due to (1) logical masking, which may be identified by static or simulation-based (see Section 3.2.8) design analysis, or (2) technological masking that may be identified using more large-scale characterization experiments.

While most works related to bit-level interconnect fault modeling and fault injection focus on *open* and *short* faults – with some dividing short faults into *input antenna* and *conflict* faults – a seldom considered consequence of interconnect faults are *output antenna* faults. Antenna faults do not necessarily directly impact the design on the logical level as they do not short-circuit or cut a connection. Instead, they merely configure an additional wire segment to be connected to the input (*input antenna*) or fanout (*output antenna*) of an original route. This additional wire segment increases the fan-out of an interconnect resource already part of the affected net and thus increases the capacitance to be driven. This potentially worsens the timing properties of the affected route.

In this context, [138] discusses conditions that lead to additional wire segments being connected to a connection by a fault in the configuration in this way for AMD Virtex 5 FPGAs. This increases the delay of the original connection between two other points by introducing additional parasitic transmission lines (paths). Measurements indicate the delays of generic interconnect wires lie in the range of 20–128 ps, and for IO-blocks in the range of 0–400 ps for the analyzed Virtex 5 technology. The authors assume that the target technology allows an SEU to connect an additional wire segment as an output antenna.

Based on the principles discussed in [138], an updated publication [47] discusses how these delay changes may affect AMD 7 series devices, represented by a Zynq SoC FPGA in this case. To quantify the additional delay of individual segments (and segment types), [47] constructs routing paths to be measured and integrates them into ring oscillators. The frequencies of these ring oscillators allow the authors to measure the additional delay introduced by the possible faulty configurations.

Both works [138, 47] offer another perspective on the cause for the discrepancy between *essential* bits that are, in principle, related to the design implemented in the FPGA and those that are actually *critical* (see Figure 4). As these faults are strongly related to timing, PVT effects may impact circuit functionality. Furthermore, manufacturing variations in the interconnect elements may lead to different magnitudes of these effects being observed either intra-chip between different instances of the same type of interconnect resources or variations between different instances of the same FPGA model. However, for these faults to be possible on a large scale due to SEUs, single configuration bits need to be able to connect wire segments of significant length. This appears not to be the case, considering AMD's Series 7 and Ultrascale devices employ decoded two-stage multiplexers. While a configuration bit flip in the first stage may still connect an intra-interconnect-tile wire segment, these wire segments are assumed to be rather short and thus have a limited effect on timing. Nevertheless, these effects may be worth considering in circuits operating at the very edge of safe synchronous operation – for example, with very tight timing constraints or if the devices are subject to very high temperatures.

Related Bitstream-Level Fault Injection Approaches and Experiments

Usually, in descriptions of configuration-memory-related fault injection experiments, data is gathered and reported from one injection per bit on a single device. Contrary to this, [139] reports the numbers of sensitive bits of an AMD MicroBlaze CPU core determined using six identical fault injection experiments on a radiation-hard AMD Virtex 5 FPGA. The authors run the same fault injection experiment six times (on the same device) to determine the number of configuration bits that are not critical in every run. They report a difference of 41 critical bits between the worst and best of these six runs. The variation observed by the authors in their six fault injection runs may hint at PVT effects playing a role in the results of fault injection experiments in earlier FPGA technologies such as Virtex 5. However, [139] does not extend this experiment to multiple devices, which would potentially have revealed even more significant inter-chip PVT effects affecting fault injection results.

Commonly, fault injections are described to be performed on one particular FPGA device instance or a pair of a device-under-test and a golden device. When performing a large number of fault injections, such as in the case of exhaustive fault injection or random fault injection sampling a large number of bits, this impacts the runtime performance of the experiment. Thus, [140] proposes the *ParFlip* approach to use multiple FPGA boards in parallel to speed up fault injection. The bits to be tested are split among these boards, thereby parallelizing the fault injection to multiple devices that individually handle their fault injection load. Afterward, the results from the different boards are

merged, resulting in a sensitivity map for the entire design. The *ParFlip* approach targets AMD Zynq FPGAs but appears to be generally applicable. The authors also discuss the scalability of their approach, showing the increase in fault injection speed when (1) more boards are added and (2) certain bits causing the system to lock up (e.g., clock-related bits) are filtered from the bitstream. With one device, they achieve approximately 10 injections per second. The authors demonstrate their approach by performing fault injection experiments on several benchmark designs of different size. They, however, do not discuss the possibility of PVT variations that potentially result in different numbers of critical bits being derived from fault injection depending on the assignments of bits to be tested to the different devices.

The *TURTLE* fault injection platform proposed in [141] allows random, targeted, sequential, and multi-cell faults. It is used in several fault injection evaluations [142, 143, 144] by changing authors involving the group of Wilson and Wirthlin at Brigham Young University, Utah, USA. In the *Conclusion* of [141], the authors suggest using multiple *TURTLE* boards in parallel to speed up fault injection in a similar way to [140]. This suggestion is applied in [144], where the authors use 20 FPGA boards in parallel to speed up fault injection. Neither [141] nor [143], however, discuss the possibility of PVT variations resulting in different criticality assignments on different devices. In later experiments by overlapping authors such as [143, 121], experiments are conducted not only once but three times and are repeated on three identical FPGAs, which may be interpreted as an implicit acknowledgment of the possibility of PVT effects.

Both a partial reconfiguration-based as well as a radiation-based fault injection campaign is conducted on a VexRiscv CPU core implemented in an Artix 7 FPGA in [143]. The CPU core was protected using TMR and ran a benchmark application under Linux during fault injection. With partial reconfiguration-based fault injection, a 13$\times$ reliability improvement was measured over the unprotected design, while with radiation fault injection, only a 10$\times$ improvement was noted. The authors attribute this to (1) the possibility of MBU occurring during radiation fault injection and (2) user memories being corrupted during radiation fault injection, a behavior that was not tested during reconfiguration-based fault injection. During radiation fault injection, scrubbing was also performed with a scrub cycle of 5 seconds, within which the entire configuration is read back, compared with the original configuration, and corrected. On average, 6 bits needed to be corrected per scrub cycle. The fault injection results from radiation testing of the TMR-protected CPU implementation were then replayed with reconfiguration-based fault injection. By injecting each of these bits 20 times, it was observed that critical bits in one run do not necessarily appear critical in all other runs. The authors use these multiple runs to calculate a *sensitivity rate* that indicates in how many of the runs each bit was critical. In total, 54 bits from the 100371 bits flipped during the radiation campaign were observed critical in at least one fault injection run. Most bits (33, 61.1%) were critical in (almost) every run (32 bits in 20 out of 20 runs, 1 bit in 19 out of 20 runs). Nine bits (16.7%) were critical only in ten of the 20 runs. Therefore, 12 bits (22.2%) were critical in less than half of the performed runs. This clearly indicates, once again, that PVT effects may influence fault injection results to a significant degree.

Finally, an alternative to bitstream-level modeling-based criticality prediction approaches, such as described in Sections 3.2.1–3.2.8, is proposed in [145]: The authors assume that critical bits are clustered closely in the FPGA configuration in terms of their configuration bit address. Thus, starting from a random configuration bit – given that it is critical during fault injection – it is assumed that further critical configuration bits can be identified by performing fault injection experiments on "close-by" bits. The approach is intended to speed up fault injection experiments whose goal is to identify as many critical bits as possible using fault injection. It constructs a hierarchical tree of the FPGA's configuration using its configuration address (die, block, row, column). The root node defines the entire fabric itself, with the different hierarchical layers addressing blocks, rows, and

columns of the FPGA. The tree is constructed by selecting a start node, expanding it by adding a child node, randomly exploring from there, and finally back-propagating the choice to the algorithm, which then selects a new start node. The start node is selected by visiting the node according to a certain metric called *Upper Confidence Bound*, based on the number of times the parent node and current node have been visited, and a reward function that is based on fault injection: A reward is given when the fault injection results in an error, thereby having found a critical bit. This allows the exploitation of correlations between "neighboring" bit addresses to find additional critical bits. The experimental evaluation by the authors shows that identifying, for example, 95% of critical bits of their benchmark design on an AMD Ultrascale FPGA happens up to 1.9 times faster than random fault injection. However, in their design, injecting faults up to that point misjudged criticality to be twice as high as it actually is (estimated sensitivity: 18.1%, actual sensitivity: 9.3%). Finding 99% of all critical bits was only possible at a 1.2$\times$ speedup while finding all critical bits required injecting faults into almost all bits.

3.2.10 Summary

In the previous sections, a considerable number of approaches to analyze and predict the criticality of FPGA designs on the bitstream level have been discussed. Table 15 contrasts the evaluation of each of the discussed bitstream-level analysis approaches against the requirements introduced in Section 3.1.2. As in the preceding sections, a filled circle, half-filled, and empty circle represent a fulfilled requirement, a partly fulfilled requirement, and a requirement not fulfilled by the approach, respectively. Missing data that is neither available nor could be reproduced by the author of this dissertation is symbolized by a dash and counted as a non-fulfilled requirement.

As a rough evaluation of the overall capabilities of each approach, scores are shown in Table 15 in the line labeled "Approach Score" below the listed requirements. These have been calculated by summing up the number of fulfilled, partly-fulfilled, and non-fulfilled requirements, assigning an equal weight to each requirement. According to this evaluation, several capable approaches in the form of *Bricas et al.* followed by *Rad-SyMPaR*, *PyXEL*, and *BFAT* exist in today's state of the art. It is to be noted that both *BFAT* and the approach presented by *Bricas et al.* were developed in parallel to this dissertation.

Table 15 also, via its column "SotA score", allows to judge the capabilities of today's state of the art by combining the properties of all analyzed approaches. It can be seen that approaches performing bitstream mapping, as well as modeling of interconnect faults and logic, with runtimes acceptable for use in DSE fitness functions, are comparatively well-served by today's state of the art. However, some weak spots are under-served by today's approaches.

Most obviously – concerning ***REQ:BIT:PVT*** – none of the discussed approaches addresses the possible impact of PVT effects on the accuracy of characterization, validation, or criticality prediction results. This is despite several works observing variations between several subsequent fault injection runs [139, 143] and between repetitions of the same experiments on different device instances [121]. Some approaches [140, 141] suggest splitting up exhaustive fault injection experiments on multiple devices to speed up fault injection experiments. In the proposed fault injection approaches, this is done by evaluating certain subsets of configuration bits on certain devices according to a predefined mapping. In the presence of PVT effects, this mapping may severely impact the critical configuration bits reported by the fault injection experiment due to some bits possibly being critical only on a subset of the employed devices.

Secondly, the discussed approaches that model the effects of interconnect faults on the configuration bit level do this mostly based on assumptions about the circuit-level implementation of

Table 15: Bitstream-Level Analysis Approaches: Summary of Requirement Evaluation

Requirements (see Section 3.1)	Approach: STAR	SEVAX	Rad-SyMPaR/VERI-Place	PyXEL	ACME/ACME-2	BAFFI	BFAT	Bricas et al.	SoTA Score
REQ:BIT:MAPPING	●	○	●	●	◑	◑	●	●	6.0 / 8.0
REQ:BIT:MODELING:LOGIC	◑	◑	◑	◑	○	○	●	●	4.0 / 8.0
REQ:BIT:MODELING:INT	●	◑	●	●	○	○	●	●	5.5 / 8.0
REQ:BIT:MODELING:SPECIAL	○	◑	◑	○	○	○	◑	○	**2.0** / **8.0**
REQ:BIT:BITWISE_PREDICTION	●	○	●	○	○	○	◑	●	**3.5** / **8.0**
REQ:BIT:CHARACTERIZATION	○	○	○	◑	○	○	◑	◑	**1.5** / **8.0**
REQ:BIT:VALIDATION	◑	○	◑	◑	○	◑	○	◑	**2.5** / **8.0**
REQ:BIT:PVT	○	○	○	○	○	○	○	○	**0.0** / **8.0**
REQ:BIT:DEVICES	○	○	●	◑	◑	◑	◑	◑	**3.5** / **8.0**
REQ:BIT:RUNTIME	●	●	–	●	–	●	○	●	5.0 / 8.0
Approach Score	5.0	2.5	5.5	5.0	1.0	2.5	5.0	6.5	

interconnect elements formed by the following sources.

1. Patents by device vendors (e.g., [134, 131], and, thus *BFAT*).
2. Analysis of the encoding of certain interconnect configurations into configuration bits, e.g., leading to the assumption of encoded or decoded multiplexers.
3. Characterization experiments that empirically evaluate the device's behavior under the faults to be modeled.

Current approaches are only seldom based on characterization experiments. The approaches that perform characterization experiments do this in a severely limited form: A few select individual PIPs are analyzed in [131], while [46] analyzes tens and [124] evaluates hundreds of interconnect elements in devices containing tens of thousands of these elements. Neither of these approaches report systematic characterization results, which this dissertation argues to be crucial for the understanding of fault effects in devices where no transistor-level documentation is available as requirement ***REQ:BIT:CHARACTERIZATION***.

While several approaches are capable of performing bit-level prediction of the criticality of certain bits (see ***REQ:BIT:BITWISE_PREDICTION***), the respective publications do not provide information about the – desirable – systematic validation of these predictions. The following four possibilities exist for the validity of the prediction of the criticality of each individual configuration bit.

1. The bit is predicted to be critical and is actually critical under evaluation (*true positive*).
2. It is predicted to be non-critical and is actually non-critical under evaluation (*true negative*).

3. It is predicted to be critical but turns out to be non-critical under evaluation (*false positive*).

4. It is predicted to be non-critical but turns out to be critical under evaluation (*false negative*).

While a perfectly accurate approach may only generate *true positives* and *true negatives*, an optimistic approach may generate *false negatives*, and a pessimistic approach may generate *false positives*. For example, the *Essential Bits* feature built into AMD's Vivado bitstream generator may be seen as a quite pessimistic bit-level criticality prediction approach. While a number of publications describing bit-level prediction approaches describe validation experiments, none of the discussed publications describes systematic bit-by-bit validation against fault injection experiments in evaluated designs. For example, the approach by *Bricas et al.* [124] reports the total *number* of critical bits predicted by their approach and compares this number to the number of critical bits determined by fault injection. While the predicted and evaluated numbers are within close range, they do not report bit-by-bit validation as required per ***REQ:BIT:VALIDATION***. While this may be acceptable if only a design's total number of critical bits is relevant, more precise validation may be required if per-module criticalities are relevant – such as in selective redundancy toolflows.

The most abundant resources in SRAM-based FPGAs, logic elements contained in their CLBs – first and foremost LUTs – and interconnect elements are addressed essentially by all discussed approaches capable of forming bit-wise predictions or predicting the total number of critical bits, special-purpose resources such as BRAM or DSP elements are only addressed in a small subset of these approaches. However, relatively few bits may be concerned: For example, AMD's 7 Series FPGAs *Project XRAY* documents 446 unique configuration bits per DSP tile[29] and 434 per BRAM tile[30] as compared to 1002 bits per CLB tile[31] and 1598 per INT tile[32] while an Artix 7 35T FPGA provides 45 DSP tiles, 50 BRAM tiles, and 33280 CLB tiles. Nevertheless, a bitstream-level criticality analysis and prediction approach should – as a minimum – be able to identify the netlist element (a net driving or driven by a special-purpose resource or a cell mapped to this resource) to determine the potential masking of a fault via TMR. Only two approaches capable of bitwise prediction are explicitly capable of the mapping of these special-purpose resources to netlist elements as per ***REQ:BIT:SPECIAL***: *Rad-SyMPaR* and *BFAT*.

Finally, concerning the supported device families listed in Table 3, and concerning ***REQ:BIT:DEVICES***, most approaches target FPGA devices manufactured by AMD (formerly Xilinx), with the only exception being RadSyMPaR/VeriPlace which also addresses devices manufactured by Microchip and NanoXplore. However, the Microchip RT-ProASIC3 [146] and RTG4 [147] devices are Flash-based devices, which places them outside the scope of this dissertation. Both Microchip devices, as well as the NanoXplore NG [148] device, are space-grade radiation tolerant FPGA devices, and thus rather focused on a limited range of applications. In terms of FPGAs manufactured by AMD, most identified approaches target the current 28 nm Series 7, 20 nm UltraScale, and UltraScale+ technologies. While released to the market in 2011 [149], 2013 [150], and 2015 [151], respectively, AMD has extended the life cycle for 7 Series devices to 2040, and for UltraScale+ to 2045[33]. This makes these devices relevant for current

[29] See `https://web.archive.org/web/20250113214443/https://f4pga.github.io/prjxray-db/artix7/segbits_dsp_l.db`, accessed 2026-01-05

[30] See `https://web.archive.org/web/20250912133142/https://f4pga.github.io/prjxray-db/artix7/segbits_bram_l.db`, accessed 2026-01-05

[31] See `https://web.archive.org/web/20250912141130/https://f4pga.github.io/prjxray-db/artix7/segbits_clblm_l.db`, accessed 2026-01-05

[32] See `https://web.archive.org/web/20250113214512/https://f4pga.github.io/prjxray-db/artix7/segbits_int_l.db`, accessed 2026-01-05

[33] See `https://web.archive.org/web/20250723193920/https://www.amd.com/en/blogs/2024/amd-supports-new-long-lifecycle-fpga-designs-thro.html`, accessed 2026-01-05

and near-to-mid-future applications. However, other well-known FPGA vendors, most notably Intel/Altera, but also Lattice, Achronix, Efinix, are underrepresented among these approaches. Comparing Intel/Altera and AMD, one reason might be the relative "open-ness" of AMD's FPGA tooling, extended by APIs and frameworks that allow extracting detailed placement and routing information such as RapidSmith[34] and RapidWright[35] that are, at least in part, supported by the device vendor. Frameworks of this kind are less common for Intel/Altera FPGAs. Additionally, while both a near-complete documentation of configuration bit functionality for AMD's 7 Series devices in the form of *Project XRAY*[36], and similar efforts for UltraScale devices in the form of *Project URAY*[37] exist, approaches for Intel's devices are far more limited. The only publicly available documentation approach for Intel devices is *Mistral*[38], which is an incomplete bitstream documentation approach focused on Intel's Cyclone V devices. Certain SRAM-based Lattice FPGA devices are documented in a similarly complete way to AMD 7 Series devices – such as the iCE40 family via *IceStorm*[39], MachXO (including MachXO2, 3, and 3D) and ECP5 via *Project Trellis*[40], and Nexus (e.g., Crosslink NX) via *Project Oxide*[41]. These devices occupy a niche in the low-power FPGA market segment and thus may be a welcome addition to the portfolio of bitstream analysis and prediction approaches.

The aim of this dissertation concerning bitstream-level criticality analysis and prediction is to fill the gaps left by current approaches as well as by related work from literature as follows:

Based on previous work, a bitstream-level criticality prediction approach for the express purpose of assessing the fitness of partial TMR variants in a DSE toolflow is developed. This approach is based on systematic characterization experiments that aid in forming accurate predictions of the behavior of the target FPGA technology under single-bit-flips in its configuration, thereby superseding the state of the art. This is especially relevant for bit flips affecting interconnect resources, which the transistor-level implementation of the respective interconnect resources may strongly influence. As modern FPGAs contain a variety of interconnect wire types of different purposes (e.g., clock, global, and generic routing as well as interfacing with logic elements) and lengths (e.g., local, short, and long lines), this interconnect characterization takes wire types and purposes into account. Common fault models such as *Open*, *Conflict* and *Input Antenna* are characterized and validated against results from literature.

Based on this characterization and analysis of the target device's configuration, predictions about the device's behavior under the influence of single-bit configuration errors are formed and mapped to netlist-level resources. All faults – affecting CLB, interconnect, BRAM, and DSP resources – shall be traced to the respective netlist element. Faults affecting interconnect resources are traced to the respectively influenced cells downstream (or upstream – depending on the fault model) of the fault. The effects of faults affecting the truth tables implemented in LUTs are modeled by taking the LUT's static and dynamic inputs into account. In addition, depending on the target technology,

[34] See `https://web.archive.org/web/20250912135627/https://rapidsmith.sourceforge.net/`, accessed 2026-01-05

[35] See `https://web.archive.org/web/20250815131553/http://www.rapidwright.io/`, accessed 2026-01-05

[36] See `https://web.archive.org/web/20250115214812/https://f4pga.github.io/prjxray-db/`, accessed 2026-01-05

[37] See `https://web.archive.org/web/20250909163755/https://github.com/f4pga/prjuray-db`, accessed 2026-01-05

[38] See `https://web.archive.org/web/20250909164258/https://github.com/Ravenslofty/mistral`, accessed 2026-01-05

[39] See `https://web.archive.org/web/20250211084008/https://github.com/YosysHQ/icestorm`, accessed 2026-01-05

[40] See `https://web.archive.org/web/20250126223514/https://github.com/YosysHQ/prjtrellis-db`, accessed 2026-01-05

[41] See `https://web.archive.org/web/20250912133817/https://github.com/gatecat/prjoxide-db`, accessed 2026-01-05

the ability to *fracture* LUTs (i.e., the ability to use an n-input LUT as two $(n-1)$-input LUTs) must also be considered in this context. The effect on the – potentially partially redundant – circuit can then be predicted from these potentially affected cells.

These predictions are extensively and systematically validated, deriving the number of correctly and incorrectly predicted criticality-related consequences of bit flips. The results of this validation may be further studied to improve future prediction accuracy.

During all stages of performing characterization experiments, development of rules for criticality prediction, and validation experiments, the effects of PVT on the target FPGA device are considered. This is done by (1) performing all experiments under controlled temperature conditions, (2) undertaking efforts to determine the extent of PVT effects in the target FPGA technology, and (3) integrating the determined PVT effects into the developed prediction rules.

While the approach targets AMD's 7 Series devices, basic interconnect fault modeling and characterization experiments are demonstrated on two additional FPGA technologies for which bitstream documentation is available: Lattice's iCE40 and ECP5 FPGAs. This is done to both demonstrate the transferability of the described fault modeling approaches and to serve as a point of reference to compare technologies by different manufacturers to identify effects found in both technologies, which also benefits the generalizability of the described approach.

Finally, care is taken to implement the approach in a way that its runtime and memory usage – in contrast to, for example, *BFAT* – does not preclude use in a fitness function for a DSE approach.

3.3 Cost-Aware FPGA Reliability Improvement Approaches

While bitstream-level analysis allows to pinpoint especially critical parts of an FPGA design's configuration and potentially to predict its overall reliability or susceptibility to configuration memory bitflips, achieving the reliability level required by the application necessitates approaches and tools to reduce the number of configuration bits that – when flipped – affect the design's correct functionality. That is if a single SRAM-based FPGA device is to be used – as opposed to solutions using multiple devices or less radiation-sensitive configuration memory such as Flash or Antifuse memory – a way to reduce the number of critical SRAM configuration bits in a design is needed. The best single-device reliability may be achieved by instantiating the same FPGA core design three times and physically separating them inside the FPGA, as well as possibly triplicating all inputs and outputs with external voters, and possibly supplying the three copies from three different clock sources. This approach, in essence, treats the FPGA as three smaller devices sharing one power supply and configuration interface. However, this greatly increases the number of FPGA resources consumed by the design or, in other words, severely limits the design size that can be implemented in a given FPGA device. More cost-aware approaches may be chosen in applications that do not require this level of fault tolerance. This section discusses four levels of abstraction on which the reliability of an SRAM-based FPGA design may be improved: Section 3.3.1 discusses low-level approaches that directly aim at reducing the sensitivity of a design on the configuration bit level. Section 3.3.2 describes works to select an especially beneficial subset of an FPGA design's modules or cells for replication using TMR. Efforts to apply optimization techniques to achieve cost-effective reliability improvements in FPGA designs are evaluated in Section 3.3.3. Approaches that leverage information available during the HLS process to generate intrinsically more reliable designs are the focus of Section 3.3.4. Finally, Section 3.3.5 summarizes the approaches from these four levels and develops the gap this dissertation fills in this context.

3.3.1 Reliability-aware FPGA Placement and Routing

Once an association can be made between certain especially fault-sensitive configuration patterns and the placement and routing constructs generating these configuration patterns, placement and routing algorithms can be implemented to penalize these constructs. This is possible by using information from approaches discussed in Section 3.2. Instead of only considering FPGA resource usage and delay as costs during placement and routing, approaches in this category also consider reliability properties. This allows the optimization inherently carried out in placement and routing to improve reliability in addition to design density and wire length. This is, for example, especially useful when placing and routing a TMR design with minimal physical separation between the replica, as a reliability-aware placement and routing algorithm may be able to avoid situations where an SEU may cause a CMF affecting two or more replica. As these methods do not introduce additional logic into the FPGA design, they can be a cost-effective way to improve design reliability to a certain extent. The cost of these approaches mainly comes in the form of less dense placement and routing, leading to slightly worse FPGA utilization for the number of FPGA area consumed by the design, and slightly longer wire lengths, increased critical path delay, and thus reduced operating frequencies.

In this context and based on the analyses carried out by *STAR*, the same authors in [99, 152] propose a routing algorithm – *RoRA* (Reliability-oriented Routing Algorithm) – that aims to improve reliability. This algorithm is geared towards avoiding SEUs affecting multiple TMR replica in Xilinx Spartan II and Virtex II FPGAs. Validating their approach's usefulness using fault injection, the authors find that while TMR reduces the number of critical bits roughly by the factors 7–10, *RoRA* leads to an additional improvement in the range of factors 41–85. Comparable numbers are reported by [153], where the authors test different grades of TMR separation: intermixed TMR domains, separation in placement without spacing, and separation by variable numbers of CLB columns. They determined that most of the time, physical separation of the three TMR copies (i.e., no mixed placement) drastically decreases the fault sensitivity compared to mixed-placement TMR. However, mixed-placement TMR improves the reliability $6\times - 22\times$ while increasing separation negatively impacts maximum frequency in most cases. In a similar way to *STAR*, *PyXEL* [117] and works by the authors of *BFAT* such as [134] aim at modifying the placement and routing of designs to reduce the possibility of CMFs. A related but different approach is taken by *FeDRA* [154] that aims at reducing the number of configuration frames used by a design. While this does not necessarily directly improve an FPGA design's reliability, it potentially reduces the time necessary to validate its relevant configuration checksums and the time to scrub its configuration: As the target FPGA's (AMD 7 Series) finest reconfiguration granularity is a 101-word configuration frame, reducing the number of frames reduces the time needed to reconfigure a design – e.g., during scrubbing.

Another approach to improve overall design reliability without introducing redundancy (e.g., via TMR) is to modify the design to work around the weaknesses of the target device. In [155], the authors first perform characterization experiments to determine the effects of SEU on the interconnect in AMD Virtex 5 FPGAs. Based on these experiments, the authors identified that approx. 82% of interconnect SEUs behave as "switch-to-one" faults (i.e., cause the affected wire to be always driven to '1'), and 17% of faults behave as the complementary "switch-to-zero". The remaining 1% of faults behave in an "always inverting" manner – inverting the signal carried by the affected interconnect resource. Based on earlier work [156], they additionally model Virtex 5 interconnect multiplexers as binary encoded multiplexers, meaning that an SEU may select a different input pin to drive the output instead of the pin originally defined by the bitstream. To increase the likelihood of such a fault being masked, the originally and erroneously selected

signals feeding into the MUX must be as logically "similar" as possible. The authors propose an algorithm that modifies the truth tables of the LUTs driving these inputs to minimize the logical difference between these signals. They evaluate the approach using configuration-level fault injection targeting combinational benchmark circuits published by the Microelectronics Center of North Carolina (MCNC) [157][42] placed together with golden circuits. The authors report an approximately 28.6% reduction in critical interconnect bits. Notably, this result is achieved with no additional overhead regarding the resource utilization of the target FPGA. Thus, (interconnect) fault characterization may not only lead to more precise criticality predictions as mentioned in Section 3.1, but also enable zero-overhead reliability improvements by mere bitstream patching.

Finally, the most convenient case for developing criticality prediction or fault tolerance measures tailored to a certain FPGA technology is that the transistor-level implementation for this technology is available. This is assumed in [158], where the authors formulate expressions to calculate fault probabilities based on a circuit-level model of a custom FPGA technology's interconnect elements. Based on these probabilities, they define a reliability-oriented placement and routing scheme they implement using Versatile Place and Route (VPR). They include probabilities P_{ON} and P_{OFF} that model the probabilities of fault propagation when the SEU affects a configuration bit that is originally on (i.e., switches on a transistor in a route defined by the bitstream) or off (i.e., blocks a route that is not defined by the bitstream). These probabilities depend on the transistor-level implementation of the circuit. For well-sized transistors, P_{ON} is assumed to be 50% (i.e., one of the logic levels is affected by the open fault), bit P_{OFF} is assumed to be only 25% (i.e., there is a wired-AND or wired-OR, but both the original signal and the conflicting signal must be in the "correct" state for it to affect the output). This can be used to calculate the expected number of critical interconnect bits, with this number being used as the SEU cost component of the routing cost function for VPR. The authors also acknowledge the possibility of partial TMR: If their routing approach does not lead to sufficient SEU improvements, they introduce partial TMR into the circuit and no longer regard the bits corresponding to nets protected by TMR as critical. The parts of the circuit to be protected using partial TMR are randomly chosen. Afterward, they perform routing and criticality analysis again and increase the TMR amount if necessary. They compare their proposed combination of reliability-oriented routing and partial TMR to full TMR, STMR [159] and *BLTmr* [160, 161] on several MCNC benchmark circuits [157]. The SEU vulnerability of the designs is tested using 1000 fault injections into the configuration in a simulation of their custom FPGA technology. The designs achieve comparable vulnerability to STMR and BLTmr and only marginally higher vulnerability than full TMR regarding the critical share of configuration bits. However, the approach proposed by [158] achieves this by consuming significantly fewer FPGA resources. While their model of interconnect multiplexers is quite similar to the one assumed by, e.g., *BFAT* (See Section 3.2.7) and *Bricas et al.* (See Section 3.2.8) for AMD Series 7 FPGAs, no real-world implementations of the particular assumed technology exist. This also precludes the authors from validating their approach in hardware. This, furthermore, means that the degree to which the tool's predictions and subsequently implemented placement and routing is robust towards PVT effects cannot be evaluated.

As the approaches in this category possess similar properties, they are evaluated against the tool-level requirements discussed in Section 3.1.1.

While the reliability-aware implementation approaches discussed above aim to reduce a design's vulnerable cross-section by avoiding potentially harmful constellations during an SEU, they do not introduce redundancy into the design. Given that a design needs to implement a certain specified functionality, nevertheless, there is an irreducible set of configuration bits that cannot be protected against the number of bit flips by careful placement and routing. Bits belonging to this set are, for

[42] commonly referred to as "MCNC" or "MCNC'91" benchmarks

Table 16: Evaluation of Reliability-aware FPGA Placement and Routing Approaches [99, 152, 155, 158] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	○	Redundancy is usually introduced in an earlier design step.
REQ:TOOL:DSE	◑	Limited DSE – optimization inherent to P&R.
REQ:TOOL:AREA_RELIABILITY	◑	Wire length optimization inherent to P&R.
REQ:TOOL:PARETO	○	None of the discussed approaches provides a Pareto front of solutions.
REQ:TOOL:PNR	●	Placement and Routing is at the core of these approaches.
REQ:TOOL:BIT	●	Approaches generally work on the bitstream level.
REQ:TOOL:PVT	○	None of the discussed approaches considers PVT.

example, configuration bits defining the functions realized by LUTs or interconnect bits that, when turned off by an SEU, cause *Open* faults in the design's nets. To protect against the consequences of bit flips in these bits, fault tolerance mechanisms such as TMR need to be introduced into the bitstream.

By their very nature, the approaches discussed above fulfill the tool-level requirement of considering the effects of placement and routing (***REQ:TOOL:PNR***) formulated in Section 3.1.1. Furthermore, they do this mostly by considering bit-level effects, fulfilling ***REQ:TOOL:BITLEVEL***.

However, they do not introduce masking capabilities into the design and thus do only offer limited DSE capabilities regarding the cost-vs-reliability design space in general and the area-vs-reliability design space in particular. This means that they do not fulfill ***REQ:TOOL:MASKING***. The requirements ***REQ:TOOL:DSE*** and ***REQ:TOOL:AREA_RELIABILITY*** are only partly fulfilled. As usually only one optimized design is generated by these approaches, the requirement of generating a result family from which to choose the desired implementation – ***REQ:TOOL:PARETO*** – is not fulfilled. Finally, while in principle possible, neither of the approaches discussed in this section considers the potential effects of PVT while performing reliability-aware design implementation – e.g., by avoiding additional configuration constellations known to cause issues at certain temperatures or during their validation experiments.

The evaluation of these approaches is summarized in Table 16. Nevertheless, reliability-aware implementation approaches may prove advantageous in combination with other approaches to improve design reliability, e.g. by optimizing module-level TMR by minimizing the possibility of TMR violations.

3.3.2 Selective Redundancy Insertion

Instead of replicating the entire FPGA design, selective redundancy approaches apply a certain evaluation approach or heuristic to select and replicate parts of the design. This may happen on gate level, module level, or at RTL, with different implications to the implemented masking. While some of the approaches described in this category employ optimization techniques to solve sub-problems, they are, in general, not geared towards large-scale design-space exploration as the approaches described in Section 3.3.3.

An approach that selectively introduces redundancy into a design selects a subset of cells or modules that are part of the design. This selection process is usually guided by a heuristic that suggests which of the design's cells or modules to replicate to effectively increase the design's reliability. One such heuristic is applied in the earlier works on partial TMR for FPGAs presented as *STMR ("Selective TMR")* in [159]: The authors calculate and propagate signal probabilities – also commonly used in the calculation of testability metrics [162] – at the combinational gates of a targeted circuit to identify gates that are especially sensitive to a soft error. They define SEU-sensitive gates as gates where a logical value complementary to the calculated signal probability (i.e., the consequence of a soft error affecting the gate's input net or a gate upstream) appearing at any one of its inputs likely will propagate through the gate and thus forward the soft error's consequences to an output of the circuit. Thus, the authors calculate the probability for every gate to (not) mask the consequences of a soft error preceding the gate. The sensitive gates identified in this way are then triplicated and connected to a majority voter. In [159], a voter implementation using the internal tri-state buffers provided by the targeted AMD Virtex FPGA family is proposed. Due to the lack of internal user-accessible internal tri-state buffers in most modern FPGAs, this description is nowadays less relevant. To validate their approach, the authors implement a subset of the MCNC benchmark designs [157] (1) in an original form without TMR, (2) a form including selective redundancy using *STMR*, and (3) a form that triplicates all gates of the design. They simulate their circuits with 1000 test vectors each and randomly introduce soft errors in this simulation by inverting the selected signal for multiple durations. Durations of 3, 5, and 20 ns were simulated, where 20 ns was the duration each test vector was applied for. Thus, soft errors lasting for 3 and 5 ns correspond to SETs while errors lasting for 20 ns constitute permanent errors that are to be expected in configuration memory upsets. As the authors evaluate their approach on the netlist level and not on the bit level, the effect of the introduced circuit modifications on the placement and routing of the circuit – influencing, e.g., packing of gates into potentially CMF-prone CLBs or effects on wire length or number of traversed PIPs – is unknown. Thus, *STMR's* effect on bit-level configuration-memory SEU sensitivity cannot be accurately assessed from the published data.

While *STMR* introduces masking capabilities into the design by selectively protecting cells it deems SEU-sensitive using TMR, no optimization or design-space exploration is performed. The algorithm proposed in [159] generates one circuit variant with selectively replicated cells according to its sensitivity analysis. Trade-offs between reliability and other properties, such as area or timing, do not play a role in its decision-making. As the approach works purely on the netlist level, also the bit-level implementation of the design or its placement or routing do not affect which parts of the circuit are replicated. Thereby, also PVT effects are not considered in these decisions. A summary of this evaluation can be found in Table 17.

Another partial TMR approach, *BLTmr*, is proposed in [160] and extended in [161]. This approach applies TMR on the netlist level and applies two heuristics to prioritize cells for triplication: On the one hand, it prioritizes cells that are part of feedback structures where the output of one state element (e.g., a flip-flop) feeds back into the logic cone driving its input. The approach identifies these structures by finding cycles in the directed graph representing the design's netlist. The rationale behind this heuristic is that state elements outside a feedback structure (e.g., in a feed-forward-only pipeline structure) might well be influenced by an SEU as long as it persists in the configuration memory. However, once the configuration memory has been corrected by scrubbing, the design's state corrects itself after all state values affected by the SEU have been shifted out of the pipeline. The time it takes for the circuit to become fully operational again after scrubbing thus depends only on the depth of this pipeline. In a circuit containing feedback structures, however, once the state of the circuit has been corrupted or reset due to this corruption, restoring an operational

Table 17: Evaluation of *STMR* [159] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	●	*STMR* selectively replicates sensitive cells using TMR.
REQ:TOOL:DSE	○	No design-space exploration or optimization takes place as *STMR* replicates all cells deemed sensitive.
REQ:TOOL:AREA_RELIABILITY	○	
REQ:TOOL:PARETO	○	
REQ:TOOL:PNR	○	Placement and Routing does not play a role in *STMR*'s replication decisions.
REQ:TOOL:BITLEVEL	○	Bit-Level effects do not play a role in *STMR*'s replication decisions. Validation takes place at the gate level.
REQ:TOOL:PVT	○	*STMR* does not consider PVT effects.

state by applying a certain input sequence may be quite complex – a problem similar to the ones faced by sequential ATPG [163]. This increases the time for which the circuit outputs incorrect values or is otherwise non-operational, which decreases the circuit's availability. Prioritizing cells in these feedback structures when applying gate-level partial TMR decreases the likelihood of SEUs affecting feedback structures, which helps to avoid these lengthy re-initializations of the circuit state. Among cells part of the discussed feedback structures, cycles (and cells) furthest from the primary outputs are prioritized, as these are assumed to possibly corrupt the state of a larger set of storage elements [161]. On the other hand, the approach prioritizes contiguous sections of triplicated gates in the graph representing the circuit. This decreases the number of voters – and thus single points of failure – inserted into the circuit. By ranking the cells in the circuit according to these heuristics, the authors further propose implementing incremental TMR – replicating cells until a certain specified budget of FPGA resources is exhausted. In [161], the approach is validated using a "synthetic" design and a "DSP kernel" design. 200 variants with incremental amounts of gate-level TMR are generated for both designs. These variants are then evaluated using bitstream-level fault injection to determine (1) the number of overall sensitive (i.e., critical) configuration bits and (2) the number of configuration bit flips in feedback structures – requiring a reset procedure to be carried out. These experiments show that the proposed approach effectively reduces bit flips in the latter category. An overhead far smaller than the replication of the entire design is sufficient to eliminate these especially critical bit flips. However, it can be seen, especially in the "synthetic" design, that the total number of critical configuration bits does not monotonically decrease with the number of replicated netlist elements: At multiple points in the criticality-vs-overhead plot, increasing overhead – i.e., replicating more cells – *increases* criticality. This might be a consequence of *BLTmr*'s sole focus on logic cells, while the effect of replicating these cells on the placement and routing of the circuit is not considered.

This is underlined by a more recent evaluation based on the work described in [160, 161]: a comparison of several replication subsets of the cells of the B13 ITC'99 benchmark design[43] is carried out in [142, 165].

This evaluation underlines that there exist more and less beneficial selections of cells for replication due to bit-level effects: Replicating all flip-flops in the B13 design using TMR does more harm than good, in effect *increasing* the number of critical bits compared to the original design. The authors theorize that this is because flip-flops in the target technology (AMD 7 Series FPGAs) are

[43] This design is included in the collection of benchmark circuits first published at the 1999 International Test Conference (ITC) (i.e., "ITC'99") [164].

configured by relatively few configuration bits. Thus, replicating the corresponding cells has little effect in masking configuration bit flips. However, the added voters and altered placement and routing introduce new vulnerabilities. On the other hand, protecting certain feedback structures allows a disproportional reduction in critical bits when considering how many additional FPGA resources are necessary for implementing these replications.

Thus, *BLTmr* completely fulfills ***REQ:TOOL:MASKING*** from Section 3.1.1, and partly fulfills ***REQ:TOOL:BITLEVEL***, ***REQ:TOOL:DSE***, ***REQ:TOOL:AREA_RELIABILITY***, and ***REQ:TOOL:PARETO***. However, as, it does neither consider the effects of placement and routing nor PVT, it does not fulfill ***REQ:TOOL:PNR*** and ***REQ:TOOL:PVT***. This evaluation is summarized in Table 18.

Table 18: Evaluation of *BLTmr* [160, 161] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	●	*BLTmr* selectively replicates cells using TMR.
REQ:TOOL:DSE	◑	*BLTmr* iteratively generates circuit variants with increasing redundancy and thus increasing cost.
REQ:TOOL:AREA_RELIABILITY	◑	
REQ:TOOL:PARETO	◑	This also allows, in a limited way, to generate families of design variants with different cost-reliability profiles.
REQ:TOOL:PNR	○	Placement and Routing does not play a role in *BLTmr*'s replication decisions.
REQ:TOOL:BITLEVEL	◑	Persistent and Non-Persistent bits are extracted; no further bit-level effects are considered.
REQ:TOOL:PVT	○	*BLTmr* does not consider PVT effects.

While the approaches discussed above introduce redundancy by exactly replicating the chosen (sub)circuits that each generate identical output signals given the same inputs, the concept of *approximate TMR* [166] (based on [167]) uses potentially smaller *approximate logic circuits* as replica. These functions generate the same output as the original function for a particular subset of input vectors. Different outputs are generated for input vectors outside this subset. By pairing the original circuit with two *approximate* replica – one approximating the original function for all cases where it outputs *'1'* and one approximating the original function for all cases where it outputs *'0'* – faults in the original circuit can be masked to some degree. The degree of fault tolerance is larger, the larger the overlap is between all replica, i.e., the set of input vectors for which both approximate replica produce the same output as the original function. In the other cases, where one of the replica generates an output mismatching the original output, this is masked by the combination of the original circuit and either the 0-approximating or 1-approximating circuit overruling the mismatching circuit. The overlapping output set's size correlates with fault tolerance and FPGA resource usage as a replica closer to the original circuit is expected to be larger (up to the size of the original circuit). Thus, the desired degree of fault tolerance can be selected based on the size of the resources spent on the two approximate circuits. As discussed in [166], ideally, knowledge about the behavior of the FPGA under faults is considered when generating the approximate replica of the original circuit. As an approximation, [166] proposes fault simulation of stuck-at faults at every net of the considered circuit, counting the number of simulated clock cycles for which the circuit's output differs from the fault-free version. Signals are considered especially critical when a simulated stuck-at fault produces an erroneous output value in at least 15% of the simulated

clock cycles. While critical signals are protected using conventional TMR, non-critical signals are protected using approximate TMR. By choosing the actual size of the set of cells protected by approximate TMR, a trade-off between the degree of protection vs. the degree of FPGA resource usage can be utilized. To validate their approach, the authors of [166] implement the ITC'99 benchmark circuit B13 [164] in seven variants: an original, un-mitigated variant, a full-TMR variant using triplicated voters, a full-TMR variant using single voters, and four variants making use of different degrees of approximate TMR, ranging between 85% and 96% of resource usage of full TMR. They test these variants both under radiation and configuration-level fault injection and measure the Mean Time To Failure (MTTF) (via radiation experiments) and the Mean number of Injected Faults To Failure (MIFTF) (via fault injection). They do not test for the criticality of individual bit flips but for the number of accumulated bit flips their circuit can tolerate. The results of these experiments show that all approximate TMR variants improve MTTF or MIFTF. In the form as presented in [166], approximate TMR does not consider the bit-level effects of configuration bit flips when analyzing the criticality of signals in the circuit (and thus also cannot consider bit-level effects of placement and routing). It thus does not fulfill ***REQ:TOOL:BITLEVEL***, ***REQ:TOOL:PNR***, and ***REQ:TOOL:PVT***, while on the other hand partly fulfilling ***REQ:TOOL:MASKING*** by the insertion of *approximate* replica. While all variants using approximate TMR perform well when compared to conventional TMR in the evaluation presented in [166], it must be noted that the compared circuits still appear to contain quite a large amount of mitigation, using upwards of 85% of the resources of the full TMR design, while the original design uses 33% of flip-flops and 18% of the LUTs of the full TMR design. Evaluations with smaller amounts of replication – i.e., more un-mitigated cells – may allow better comparison with module-level partial "conventional" TMR. Furthermore, due to the accumulating fault injection experiments, the actual resilience of the generated designs to SEUs in terms of individual critical bits is unknown. Thus, the effect of the altered placement and routing due to the TMR insertion cannot be analyzed.

While the presented approach, in principle, would allow the implementation of a design-space exploration tool, this is not done in [166], thus not fulfilling ***REQ:TOOL:DSE***, ***REQ:TOOL:AREA_RELIABILITY***, and ***REQ:TOOL:PARETO***.

In principle, however, approximate TMR could also be combined with a DSE approach as a way to introduce (module-level) redundancy. This would provide a further alternative to non-replicated implementations or implementations using TMR or 5MR. The evaluation of the tool against the tool-level requirements formulated in Section 3.1 is shown in Table 19.

Table 19: Evaluation of *Approximate TMR* [166] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	◑	*Approximate* replica are introduced.
REQ:TOOL:DSE	○	DSE or Optimization is not part of the proposed approach.
REQ:TOOL:AREA_RELIABILITY	○	
REQ:TOOL:PARETO	○	
REQ:TOOL:PNR	○	Placement and Routing do not affect the approach's replication decisions.
REQ:TOOL:BITLEVEL	○	Bit-level effects do not affect the approach's replication decisions.
REQ:TOOL:PVT	○	PVT effects are not considered.

While this section mainly focuses on *automated* introduction of selective redundancy, guided by heuristics and by parameters entered by the designer, as a more "conventional" approach to selective TMR is to apply knowledge about the application at hand. If the designer can, by their

experience in the field of application, suggest parts of the design that might be especially vulnerable to soft errors, *selective TMR* can also mean just following these expert recommendations. An example is the evaluation presented in [168], where the authors compare the performance of two neural networks in different versions under fault injection and radiation experiments. They compare an unmitigated version, a full-TMR version with hardware voters, and a full-TMR version with software voters to a partial-TMR version replicating only the layers of the neural network deemed especially critical by the authors. In their evaluation, they accept certain minuscule disturbances of the neural networks' outputs due to injected faults as tolerable. In contrast, faults that affect the classification output by the tested *Iris flower* and *MNIST* handwriting recognition networks are considered critical. The authors show that in these networks, selectively protecting the hidden layer in the Iris Flower network and a fully connected layer close to the outputs in the MNIST network drastically improves the fault tolerance of these networks while consuming far fewer FPGA resources than full TMR. Instead of replicating entire layers, more recent approaches such as [169] suggest replicating resources for individual channels within layers for an even finer-grained introduction of fault tolerance. This approach is evaluated on three quantized neural networks trained on the *CIFAR-10* image dataset. Evaluating variants of these networks with different degrees of added redundancy allows the authors to explore the design space regarding FPGA resource utilization (in terms of used LUT) vs. worst-case classification error. While these approaches are rather application-specific and do not consider criticality on the bitstream level – and thus cannot directly take placement and routing or PVT effects into account – they nevertheless demonstrate the benefit that can be gained by "intelligently" selecting subcircuits for replication using TMR. Due to its application-specific nature, an evaluation of [168] or [169] against the requirements from Section 3.1 is omitted.

3.3.3 Redundancy Optimization and Design-Space-Exploration

When there are multiple options to improve a design's reliability, be it different selections or combinations of sub-circuits or cells for replication or different options for implementation (reliability-optimized or conventional placement and routing), that have different associate costs and reliability improvements inherent to them, a design space can be explored for the solution(s) that best fit the target application. The costs of a variant can be quantified in terms of FPGA resources used by the design, as well as metrics often correlating with resource usage such as critical path delay or power dissipation. The remaining number of critical configuration bits, an estimated radiation cross-section, or MTTF can be used as metrics for the reliability properties of a design (implementation) variant. A method of evaluating these properties is required to implement optimization or DSE approaches, either by static prediction (see Section 3.2) or empirical evaluation – for example, in the form of fault simulation or fault injection experiments.

The partial TMR approach proposed in [170], *NuTMR* ("non-uniform TMR"), is not specifically targeted at FPGA. Similar to [159], the authors argue for different TMR implementations for sub-circuits according to the observability of their outputs and the related probability that they are responsible for an error propagating to the circuit's outputs. The authors formulate the problem of which parts of the circuit to replicate using TMR considering the logical cones driving each circuit output and each gate's input: At each gate, in essence, three options for replication exist: (1) do not replicate the gate and the cones feeding its inputs, (2) replicate the gate itself, connecting the cones driving its inputs to the respective inputs of all copies, and vote over the gate's outputs, or (3) replicate the gate along with the cones driving its inputs and vote over the replicated cone's outputs. As each cone consists of gates for which these three options exist, this describes a recursive problem with a large solution space. While the minimum option to implement a given circuit is

no TMR at all, the maximum option is to recursively implement TMR at every circuit level. Thus, while each cell in the first level (closest to the primary output) is triplicated, cells in the second level are instantiated ninefold, in the third level 27-fold, and so on. The approach presented in [170] enumerates all possible circuit variants for each cone. It then sorts the possible solution candidates for the cone by their area and fault probability and eliminates variants dominated by others in either of these metrics. Thus, a Pareto front along area and fault probability is generated for each cone. The fault probability is evaluated using fault simulations: The circuit is simulated with multiple randomly generated test vectors in a fault-free form and a form affected by randomly generated faults. A simulated fault inverts the logical value output by a randomly selected gate. The fault is masked if the fault-free and faulty cases deliver the same results. The fault probability achieved by a replication variant is calculated as the ratio of non-masked faults to simulated test vectors. As many intersecting logical cones exist in any circuit, the authors discuss ways to identify, order, and optimize these cones. Finally, for the remaining cones after these steps, the approach generates a set of Pareto-optimal cones that can be combined to form Pareto-optimal combinations of solutions for the entire circuit. Thus, the approach both explores the area-reliability design space and delivers a "family" of Pareto-optimal solutions, fulfilling ***REQ:TOOL:DSE***, ***REQ:TOOL:PARETO*** and ***REQ:TOOL:AREA_RELIABILITY***.

The authors demonstrate the effectiveness of their approach by comparing both the area (when implemented using a 45 nm ASIC cell library) as well as the simulation-derived fault probability of (1) the original version, (2) a conventional TMR version, and (3) their *NuTMR* version of 11 combinational ISCAS'85 [136] and ISCAS'89 [137] benchmark designs. While consuming significantly less area than full TMR, *NuTMR* achieves reliability improvements in all the evaluated circuits. While the fault probability is higher than with TMR in most circuits, in some, reliability is on par with TMR. In three circuits, *NuTMR* achieves better results than conventional TMR while consuming significantly less area. While presented in [170] for ASIC technology, the approach is – in principle – also applicable to circuits implemented in FPGAs. However, both the area evaluation as well as the fault simulation evaluation of the cone variants would need to include bit-level evaluations (number of used configuration bits vs. number of critical configuration bits), as these are the relevant elements in SEUs affecting an SRAM-based FPGA's configuration. As faults in these devices can affect routing and cause CMFs between wires routed or cells placed "close" to each other, this evaluation is difficult to do on a cone-by-cone granularity without placing and routing the entire design for the target FPGA. Thus, while *NuTMR* fulfills the requirements mentioned above as well as ***REQ:TOOL:MASKING*** by introducing TMR, it does not fulfill the FPGAs-related requirements ***REQ:TOOL:PNR*** and ***REQ:TOOL:BITLEVEL***, and thereby also does not consider configuration-fault-related PVT effects (***REQ:TOOL:PVT***). Furthermore, while recursive TMR has been proposed for intrinsically unreliable cells in IC fabrication processes [171], it may only offer limited benefits over TMR when protecting against SEU. The evaluation of *NuTMR* against the tool-level requirements laid out in Section 3.1 is summarized in Table 20.

In [172], the cost-reliability DSE problem is formulated as the task of finding an optimal *grouping* of an FPGA design's modules, where each of these groups is assigned both a reliability improvement technique and a reconfigurable fault containment region on the target. The groups define the granularity of TMR application, as a group may comprise an entire module or a subset of its cells. Reliability improvement techniques include Duplication With Comparison (DWC)/Dual Modular Redundancy (DMR) and TMR or the option not to replicate the module at all. All three properties are encoded into a genome, containing both a per-module group assignment and a mapping of these groups to (1) reliability improvement techniques and (2) a reconfigurable region in the FPGA. A fitness function is formulated that considers (1) the number of FPGA resources necessary to implement the solution, (2) the minimum areas required by the coarse-grained reconfigurable

Table 20: Evaluation of *NuTMR* [170] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	●	Cell- or module-level TMR is introduced.
REQ:TOOL:DSE	●	The tool aims at generating a set of area-optimized TMR variants of a starting design.
REQ:TOOL:AREA_RELIABILITY	●	
REQ:TOOL:PARETO	●	
REQ:TOOL:PNR	○	Placement and Routing do not affect the approach's replication decisions.
REQ:TOOL:BITLEVEL	○	Bit-level effects do not affect the approach's replication decisions.
REQ:TOOL:PVT	○	PVT effects are not considered.

fault containment regions, (3) the number of fault containment regions required to implement the solution, as well as (4) a metric for the system's fragmentation as an estimate of the utilization of the fault containment regions. These metrics are estimated from an initial synthesis run and thus do not consider bit-level properties or the effects of placement and routing. Thus, the approach does not fulfill ***REQ:TOOL:BITLEVEL*** and ***REQ:TOOL:PNR***, as well as is precluded from considering PVT effects (***REQ:TOOL:PVT***). Building on these formulations, the authors describe a two-dimensional optimization approach using NSGA-II [173] that generates a Pareto front of solutions (***REQ:TOOL:PARETO***). The design space explored by this approach considers as one dimension the total size of the implemented system in terms of FPGA slices and as its second dimension, the average size of each reconfigurable fault containment region. Smaller fault containment regions can be reconfigured (i.e., scrubbed) faster, meaning faster recovery time from detected faults. Thus, it does not directly optimize for the reliability of a solution while still introducing reliability-improvement techniques into the design, fulfilling ***REQ:TOOL:MASKING***. However, validation of the reliability of the found solutions using fault simulation or injection is not reported in [172]. The evaluation above is summarized in Table 21.

Table 21: Evaluation of [172] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	●	TMR is introduced at different granularities.
REQ:TOOL:DSE	●	The approach aims to generate a set of repairability-optimized TMR variants of a starting design.
REQ:TOOL:AREA_RELIABILITY	○	The approach optimizes for *area* and *fault containment region size*, but not for reliability.
REQ:TOOL:PARETO	●	Due to its use of NSGA-II [173], sets of Pareto-optimal solutions are generated by the approach.
REQ:TOOL:PNR	○	Placement and Routing do not affect the approach's replication decisions.
REQ:TOOL:BITLEVEL	○	Bit-level fault effects do not affect the approach's replication decisions.
REQ:TOOL:PVT	○	PVT effects are not considered.

In [174], a DSE approach to select combinations of (1) cell subsets to be replicated as proposed by *BLTmr* [160, 161] and (2) TMR and voter configurations (single voters vs. replicated voters

or cascaded TMR) for a given design is proposed. The approach aims to minimize resource utilization, critical path delay, and power draw. The authors argue that these results strongly depend on placement and routing. Thus, they recommend as part of their DSE toolflow first to implement all possible variants of a design (i.e., explore all parameter combinations), then eliminate dominated variants, and finally select the optimal parameter configuration. This is demonstrated on six benchmark designs of different sizes, implemented for the AMD Virtex 5 FPGA technology: the two combinational designs *c17* and *c3540* from the ISCAS'85 design collection [136], the two sequential RTL designs *s713* and *s838* from the ISCAS'89 collection [137], and the two sequential RTL designs *b17* and *b22-1* from the ITC'99 collection [164]. For the combinational design, the advanced techniques for suggesting triplication subsets and voter insertion suggested by *BLTmr* are not applicable, as these are only defined for sequential designs. Thus, here, the non-replicated version of the two circuits is compared to variants using different voter replication and TMR strategies. The non-replicated design and 32 combinations of parameters are evaluated for each of the other designs. As the size of the parameter space is not design-dependent and manageable in size (33 variants per design), no heuristic or meta-heuristic, such as a genetic algorithm, is employed to select specific variants. While the approach targets generating redundancy configurations of a given circuit, the design space exploration does not consider circuit reliability as a metric. It merely focuses on (1) the number of slices used by the design, (2) the maximum achievable frequency as predicted by the vendor's static timing analysis, and (3) the predicted dynamic power draw as predicted by the vendor's implementation tool. Thus, while fulfilling the requirement of introducing redundancy (***REQ:TOOL:MASKING***), as well as outputting a Pareto front of solutions (***REQ:TOOL:PARETO***), the requirement for optimizing design-space-exploration (***REQ:TOOL:DSE***) is only partly fulfilled as variants are compared, but no optimizing approach is employed. The approach does not quantify the reliability of each variant, neither on the bit level nor on a higher level. Thus, the requirement for bit-level reliability analysis (***REQ:TOOL:BITLEVEL***) is not fulfilled. This also precludes the approach from considering the bit-level effects of altered placement and routing. However, via the slice usage and maximum frequency, placement and routing inform the results generated by the approach, partly fulfilling ***REQ:TOOL:PNR***. Finally, ***REQ:TOOL:PVT*** does not affect the tool's evaluation. This evaluation against the tool-level requirements from Section 3.1 can be found in Table 22.

Table 22: Evaluation of [174] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	●	Design subsets as proposed by *BLTmr* [160, 161] are replicated.
REQ:TOOL:DSE	◑	All 33 variants per design are explored; no optimization is performed.
REQ:TOOL:AREA_RELIABILITY	○	Variants are compared on area, maximum frequency, and predicted power usage, but not on their reliability.
REQ:TOOL:PARETO	●	Since all variants are explored, the true Pareto front can be constructed.
REQ:TOOL:PNR	◑	Placement and routing influence the slice-level resource usage and maximum frequency metrics.
REQ:TOOL:BITLEVEL	○	Bit-level fault effects do not affect the approach's evaluation.
REQ:TOOL:PVT	○	PVT effects are not considered.

An alternative to introducing redundancy into an FPGA design to improve its resilience against configuration memory bit flips is proposed in [175]: Selecting an optimal combination of synthesis and implementation tool parameters that generate an inherently more reliable implementation than when using the implementation tool's default parameters. The optimization of the parameter settings is proposed to be carried out using the NSGA-II [173] genetic algorithm. A set of 30 parameters for the AMD Vivado synthesis and implementation tools is considered. Each of these parameters has two or more options. Examples are shown in Table 23.

Table 23: Examples for Vivado synthesis [176] and implementation [177] parameters considered by the DSE approach proposed in [175]

Factor	Synthesis/Implementation Parameter	Settings
x_{01}	`directive`	0: `Default*` 1: `RuntimeOptimized` 2: `AreaOptimized_high` 3: `AreaOptimized_medium` 4: `AlternateRoutability` 5: `AreaMapLargeShiftRegToBRAM` 6: `AreaMultThresholdDSP` 7: `FewerCarryChains`
x_{21}	`opt_design.is_enabled`	0: `false` 1: `true*`
x_{30}	`post_route_phys_opt_design.directive`	0: `Explore` 1: `AgressiveExplore` 2: `AddRetime` 3: `Default*`

Each parameter's setting is encoded into a chromosome to implement single- or multi-objective optimization using NSGA-II. During the fitness evaluation for the genetic algorithm, each candidate in the current population is first implemented using AMD Vivado. The bitstream and metadata, such as the maximum frequency of the design as predicted by static timing analysis, are extracted for each implemented candidate. The number of critical bits is determined by fault injection experiments carried out in hardware. To speed up this process, iterative statistical fault injection [178] is employed. This fault injection approach works as follows: The minimum sample size for achieving a given margin of error and confidence level depends on the probability p of a randomly chosen configuration bit being critical or not. Statistical fault injection starts with the worst-case p of 50%: it is considered equally likely to choose a critical or non-critical configuration bit randomly. The true probability, however, may differ significantly from this number. For example, the FPGA vendor AMD estimates a design-dependent share of approximately 5–10% of bits predicted by its *Essential Bits* feature being actually critical [29]. As this percentage is unknown at the start of the fault injection campaign, the worst case (p = 0.5) is assumed for the initial sample size calculation with a relaxed margin of error and relaxed confidence level requirements. The expected p is adapted from this initial injection, and the minimum sample size is recalculated, applying stricter requirements for the margin of error and confidence level. This process is repeated until the required accuracy is achieved. The authors show in [178] that they could speed up fault injection experiments by up to a factor of 6.1 while still achieving a margin of error of 0.1 % when estimating the total failure rate of the design. In their evaluation and using their design containing 784447 bits, this still corresponds to a margin of error of 784 critical bits. Furthermore, the criticality of configuration bits may be non-uniform: for example, some interconnect faults – e.g., open faults – may be more critical than

others – e.g., antenna faults. This would need to be considered when taking a random sample of configuration bits, for example, by taking a "stratified" sample of each category of configuration bits.

The authors also propose an adapted version of their fault injection approach that is bounded by experimentation time and allows to quantify the margin of error achieved during any point of the incremental fault injection experiments. This allows to abort experimentations once a given margin of error is reached. This is used in [175] to further reduce the fault injection effort, as fault injection experiments are stopped once the criticality-related domination relationship between the considered candidates can be established clearly enough. This is the case once the confidence intervals of the candidates' numbers of critical bits no longer overlap. For the single-objective optimization, only this number of critical bits is used as a fitness metric. At the same time, severely sub-optimal variants (in terms of criticality or maximum frequency) are eliminated from the population. The multi-objective optimization proposed by the authors additionally considers maximum achievable frequency as a second optimization dimension.

In single-objective mode, the approach reduced the number of critical bits by up to 52% compared to the default configuration in the evaluation carried out in [175]. In multi-objective mode, the number of critical bits was reduced by up to 25% while simultaneously improving the maximum clock frequency by 17%.

The approach shows that not only redundancy but also the *way* a circuit is implemented can have a significant impact on circuit reliability. For example, analyzing the "winning" chromosomes, the authors of [175] deduce that parameters instructing the synthesis or implementation tool to optimize the generated logic heavily appear to correlate with more reliable designs. The authors assume this is due to its smaller "attack surface" exposed by these optimized circuits. It is, thus, interesting that the authors, in their multi-objective optimization, do not consider FPGA utilization as a fitness metric.

In summary, the approach proposed in [175] fulfills the requirements ***REQ:TOOL:DSE*** and ***REQ:TOOL:PARETO*** by optimizing for reliability and maximum frequency, and generating a family of solutions on different points along the corresponding Pareto front. However, since the approach does not introduce masking into the circuit, its reliability cannot be optimized beyond a certain design-specific number of irreducible configuration bits defining its functionality. If the application requires reliability beyond that point, the approach must be paired with redundancy insertion. Thus, ***REQ:TOOL:MASKING*** is not fulfilled by [175]. Nevertheless, by implementing each candidate solution, exporting it as a bitstream, and finally testing each candidate under fault injection, the approach fulfills the requirements ***REQ:TOOL:BITLEVEL*** and ***REQ:TOOL:PNR***. The used statistical fault injection delivers a rather coarse estimate of the number of critical bits of each candidate when variants are differentiated clearly enough along this axis, thus requiring subsequent fault injection experiments by the designer if the precise number of critical bits needs to be known. It furthermore is solely based on AMD's *Essential Bits* estimation without additional fault modeling. Finally, the approach does not consider PVT effects (thus not fulfilling ***REQ:TOOL:PVT***): The fault injection experiments are carried out on two AMD Zynq FPGA boards in parallel, without consideration given to the influence of the individual FPGA device or environmental conditions such as the effect of chip temperature on fault injection results. The above evaluation is summarized in Table 24.

A redundancy-introducing design-space approach is proposed in [179], extending on the work published in [180]. The approach works on RTL module level by pre-generating design variants, where each of these variants has exactly one of its replicable modules replicated using a fault tolerance technique (TMR, 5MR) to quantify the costs and benefits of replicating each module. Furthermore, a simplex variant without fault tolerance features is generated. These variants are synthesized, implemented for an AMD Virtex 5 FPGA, and subsequently tested under fault

Table 24: Evaluation of [175] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	○	The approach aims to forgo replication by optimizing synthesis/P&R parameters.
REQ:TOOL:DSE	●	NSGA-II is used to explore beneficial combinations of these synthesis/P&R parameters.
REQ:TOOL:AREA_RELIABILITY	○	The approach optimizes critical bits one-dimensionally or co-optimizes with maximum frequency.
REQ:TOOL:PARETO	●	NSGA-II generates Pareto fronts of solutions.
REQ:TOOL:PNR	●	Effects considered through bit-level fault injection and timing analysis.
REQ:TOOL:BITLEVEL	●	Bit-level reliability is considered through statistical fault injection.
REQ:TOOL:PVT	○	PVT effects are not considered.

injection using the *FT-EST* framework by the same authors published in earlier work [181]. This framework predominantly injects faults into LUTs, thus limiting the available information about the routing-related reliability impacts of each combination. *FT-EST* injects cumulative faults at a specified rate to determine the MTTF of the pre-generated design variants. Thus, no bit-by-bit information on the criticality of individual configuration bits is retrieved.

Using this pre-generated information, the design space exploration part of the toolflow follows. The tool considers the design space of all combinations of design modules, which can be implemented as simplex systems or made redundant using the above-mentioned fault tolerance techniques. This design space is first constructed by the tool by enumerating (1) all the relevant variants, (2) variants selected by a particular heuristic, or (3) guided by a single- or multiple-objective optimization approach. For example, in [180], a Multiple Choice Knapsack Problem (MCKP) solver is used to generate the relevant variants to evaluate. Once these variants have been generated, the respective metrics of the entire system are predicted by combining the reliability (MTTF), resource usage, and power dissipation determined from the pre-generated module variants. From this prediction, the Pareto-optimal variants are identified and presented as output solutions, from which the user may choose a suitable implementation. As a case study, the authors apply their DSE approach to a partial reconfiguration controller (*GPRDC*, [182]), consisting of ten major components (i.e., RTL modules). As detailed above, the authors first evaluate redundant versions of the individual components independently. Only simplex and TMR versions for each module are considered in this case study, resulting in a design space of 2^{10} possible combinations of modules and design variants. By combining the pre-generated MTTF, resource usage, and timing properties, the metrics for each of these variants are statically estimated. This allows the construction of Pareto fronts regarding the three possible combinations of these metrics from which one solution is selected.

The DSE approach presented in [179] relies on combining data measured in isolation for each of the design's modules. This approach is significantly less time-consuming than implementing and evaluating each individual combination. However, each change to the design – e.g., the insertion of redundant logic and voters – is assumed to cause alterations in placement and routing. It is unclear if combining the data measured in isolation accurately represents the resulting design's reliability due to the cross-module effects of these alterations. This also includes possible TMR

violations introduced during the routing process.

Thus, while the approach fulfills the requirements ***REQ:TOOL:DSE*** and ***REQ:TOOL:PARETO***, and – partly – ***REQ:TOOL:BITLEVEL*** by considering bit-level fault injection results, it does not fulfill ***REQ:TOOL:PNR***. Also, the possibility of PVT effects affecting the initial reliability estimation of each individual design module is not considered. Finally, while the approach bases the calculation of its Pareto fronts on its prediction of design metrics from the combination of pre-generated per-module data, the prediction accuracy is only validated for one design variant of the *GPDRC* design. A summary of this evaluation can be found in Table 25.

Table 25: Evaluation of [180, 179] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	●	TMR or 5MR is introduced selectively.
REQ:TOOL:DSE	●	A MCKP solver can be employed to combine pre-generated variants.
REQ:TOOL:AREA_RELIABILITY	●	Predicted MTTF and configuration bits are used as metrics for reliability and FPGA area, respectively.
REQ:TOOL:PARETO	●	The approach can construct Pareto fronts from the design space.
REQ:TOOL:PNR	○	The replication variants for each module are pre-generated in isolation. Placement and routing effects due to the combination of these variants can thus not be considered.
REQ:TOOL:BITLEVEL	◑	The pre-generated variants are evaluated using fault injection. However, MTTF is evaluated instead of per-bit criticality. Fault injection predominantly focuses on LUTs.
REQ:TOOL:PVT	○	PVT effects are not considered.

3.3.4 High-Level Synthesis-based Reliability Improvement Approaches

As an alternative to low-level (e.g., netlist-level or bit-level) reliability analysis and introduction, several approaches from the literature propose introducing these measures at a higher level of abstraction. In this diverse set of approaches, the considered design's control or data flow is often analyzed for structures deemed especially vulnerable to faults. The HLS process then considers these evaluations, e.g., in its resource allocation/binding step by binding one operation to multiple (potentially shared) resources or by selecting more reliable operator implementations for some operations. Additional complexity may be introduced when adapting the degree of resource sharing, area of the design, or latency and critical path delay in combination with system reliability.

Several approaches [183, 184, 185, 186] rely on a library of operator (or Functional Unit (FU)) implementations with different area, latency, and reliability properties that have been characterized beforehand. This pre-characterization is more suitable to ASIC designs, where the transistor-level implementations of the respective FUs can be individually synthesized and characterized. In FPGAs, the design's precise placement affects the packing certain logic cells into slices that share certain configuration bits. Furthermore, the precise routing of its signals affects the number of employed interconnect resources and the individual routing's influence on the subset of nets that may be shorted, as well as the location of opens or shorts. Thus, any changes to the design may

affect the outcome of these implementation steps. This makes it difficult to predict the impact of combining particular pre-characterized FU implementations on the design's reliability.

These approaches aim at predicting the combined reliability of a resource binding by treating it as a Reliability Block Diagram (RBD) [96], calculating the resulting system reliability by combining the series-parallel structures of its operators. All of these approaches aim to co-optimize the generated solution's reliability with its area and other properties. [185] stands out as an approach that uses a genetic algorithm to optimize the reliability properties of the scheduling and binding steps.

In addition to general circuit reliability, the approach described in [186] concerns itself with the minimization of the *active time* data needs to be stored in (protected) memory elements in the design, thereby not only addressing faults occurring in the logic of the design but also SEUs in its memory elements. This concept of *active time* is also considered in [187, 188], where it is referred to as *lifetime*. Variables with a long lifetime are seen as inherently more critical, as the more time passes between updating the variable and the last time its value is needed, the more likely it is that a SEU (occurring randomly but at a constant rate) affects the value. [187, 188] aim at extracting especially vulnerable variables from the design's Control and Data Flow Graph (CDFG), and subsequently adapt the register binding of these variables in a way that as many vulnerable variables are bound to a set of register units protected by TMR or ECC that is kept as small as possible.

The approaches described above are not directly targeted at use in FPGAs, and no FPGA-related experimental evaluations of the approaches are mentioned in the corresponding publications. However, the fault injection evaluation carried out in [189] shows that also the configuration-level reliability of an FPGA design is influenced by the HLS process: The authors analyzed which effects the combination of (1) enabling or disabling *pipelining* and (2) enabling or disabling the use of DSP blocks in Xilinx FPGAs during HLS had on the number of essential and critical bits of the FPGA design (see Figure 4) under fault injection. They showed that disabling pipelining (leads to smaller logic and thus less "attack surface" for configuration bit flips) combined with the use of DSP blocks (again leads to smaller logic implemented in configurable resources) positively influenced design reliability. Thus, in HLS-generated FPGA designs, the HLS process may itself contribute to the implemented design's reliability. In this context, several reliability-aware HLS approaches focusing explicitly on FPGAs have been proposed [190, 191, 192, 193, 194].

In [190, 191], an HLS approach aimed at mitigating SEUs in the configuration memory of FPGAs is described. This approach, on the one hand, works on the level of an intermediate representation of the high-level code. On the other hand, it also forward-annotates triplication from HLS to a floorplan generation and routing tool. It then makes use of approaches similar to RoRa [99, 152] (See Section 3.3.1) to respect the bit-level properties of these triplications. Thus, [190, 191] can be seen as an approach to combine high-level reliability analysis and fault mitigation with low-level reliability-aware implementation algorithms.

The high-level part of the approach functions by triplicating the operations found in the application's Data Flow Graph (DFG), which are scheduled and bound to triplicated resources in such a way as not to map two replica of the same operation to the same hardware resource, thus allowing hardware faults in any one of the hardware replica to be masked by the other two. Furthermore, the high-level part of the approach already attempts to reduce the number of routing resources (or interconnect complexity). This is done by considering the number of input signals shared by multiple resources when mapping them to replica. This property is introduced as a cost component during the various steps of the HLS process.

Finally, the hardware triplication properties are forward-annotated to the low-level part of the toolflow, including a floorplanning and routing tool. This tool honors this information by integrating

reliability metrics, such as a required spatial gap between replicas, into the cost function of the P&R tool.

The evaluation and validation of the approach, however, only happens statically: The number of potential TMR conflicts, as well as the area cost (number of CLBs) and delay cost, are compared. These numbers are compared to several reference points: (1) no TMR, (2) post-synthesis TMR with voters at every CLB, and (3) post-synthesis TMR with voters for feedback loops (see [160, 161]). The HLS-level TMR, as proposed by [191], offers the best resource usage and critical path delay in this comparison. However, since the reliability of the variants is not compared, the reliability advantage of the proposed method cannot be quantified. As the approach introduces TMR as well as aims at optimizing reliability and area usage, the approach fulfills ***REQ:TOOL:MASKING***, ***REQ:TOOL:DSE***, and ***REQ:TOOL:AREA_RELIABILITY***. Due to its focus on reducing interconnect complexity already during the HLS process, as well as forward-annotating vulnerability gap constraints, it addresses ***REQ:TOOL:PNR***. However, it does only indirectly consider bit-level effects (i.e., via the introduction of a vulnerability gap), thus partly fulfilling ***REQ:TOOL:BITLEVEL***. Finally, the approach generates a single solution instead of a set of solutions with different area-reliability properties, nor does it consider PVT effects, thus not fulfilling ***REQ:TOOL:PARETO*** and ***REQ:TOOL:PVT***. A summary of this evaluation is shown in Table 26.

Table 26: Evaluation of [190, 191] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	●	Operation FUs are replicated using TMR.
REQ:TOOL:DSE	●	The high-level part performs optimization.
REQ:TOOL:AREA_RELIABILITY	●	Area and reliability are optimization goals.
REQ:TOOL:PARETO	○	One design candidate is generated by the approach.
REQ:TOOL:PNR	●	Placement and routing are addressed by the low-level part of the approach, the high-level part already attempts to reduce routing complexity.
REQ:TOOL:BITLEVEL	◑	Bit-level fault effects are addressed indirectly by introducing a “vulnerability gap”.
REQ:TOOL:PVT	○	PVT effects are not considered.

Whereas the approach described by [190, 191] focuses on the entire design, *StitchUp* described in [192] prioritizes the control flow of the design. After generating an LLVM[44] Intermediate Representation (IR) of the design described in C code using the front-end of the HLS tool *LegUp*, *StitchUp* analyzes its CDFG to separate instructions related to the control flow from those just affecting data flow. The original LLVM IR is synthesized using the *LegUp*’s backend, resulting in a RTL description of the design as well as the schedule generated by *LegUp*. In parallel, the control-flow-related instructions are converted into LLVM IR again and synthesized to RTL with *StitchUp*’s own backend. *StitchUp*’s backend also takes the schedule generated by *LegUp* into account to generate a control flow state machine that is temporally equivalent to one implemented by *LegUp*. The two RTL designs are then merged, resulting in a design where the logic related to the control flow has been duplicated. By comparing the outputs of the two control flow implementations, control-flow-related faults can be detected (i.e., DMR /DWC). The authors evaluate *StitchUp* by implementing several benchmark designs in three variants for the AMD Zynq 7000 device family: (1) without replication, (2) with full-design DMR, and (3) control-flow-related DMR using *StitchUp*. They compare

[44] *“The name ‘LLVM’ itself is not an acronym; it is the full name of the project.”*, see `https://web.archive.org/web/20250917221536/https://www.llvm.org/`, accessed 2026-01-05

resource usage (in terms of LUTs, Flip-Flops (FFs), and DSP blocks), power dissipation, as well as resilience via configuration memory fault injection results. The resource overhead incurred by *StitchUp* naturally depends on the application's control-flow-orientedness. The authors show that cryptography-related designs (Advanced Encryption Standard (AES), Blowfish, Secure Hash Algorithm (SHA)) contain relatively small control-flow-related logic in their examples. In contrast, in other designs, such as floating-point adders, dividers, multipliers, and sine functions, *StitchUp* replicates nearly the entire design, resulting in overhead comparable to DMR.

In terms of resilience, the total number of errors is subdivided into data-only-errors (design finishes calculation but returns erroneous data), execution time errors (design never finishes), and caught errors (detected by DMR or *StitchUp*), and their relative proportions are compared. Results indicate that both duplicating the entire design (i.e., DMR) as well as *StitchUp* eliminate all execution-time errors. Both approaches also catch some errors that otherwise would manifest as data-only errors. However, the reported data being entirely relative (i.e., the share of error types of all recorded errors during fault injection), the overall design SEU susceptibility regarding critical bits is not reported. Furthermore, reporting area in terms of the number of essential bits (as opposed to the number of logic cells) would have been more appropriate when evaluating an approach addressing configuration-level SEUs as this number would also include the effect of a more complicated interconnect.

While it seems intuitive in principle to instantiate the control-flow-related logic extracted by *StitchUp* twice to implement control flow TMR, this approach is neither examined in [192] nor in the authors' subsequent works. Thus, in its published form, *StitchUp* neither introduces fault-masking into the design nor performs optimization or design-space exploration, as the entire control-flow-related logic is replicated. Furthermore, the approach is based on the assumption that the control-flow-related logic is the most important part of the design and that it needs to be replicated in its entirety. However, protecting a subset of this logic may mask errors while consuming fewer logic resources in the case of control-flow-intensive designs such as the floating-point functions mentioned above. Finally, neither the replication decisions nor the implementation are able to consider placement and routing, as well as bit-level effects (e.g., effects due to changed routing), and thus are also not able to consider PVT effects on SEU susceptibility. Thus, while *StitchUp* aims at trading off reliability and area, and thus fulfills ***REQ:TOOL:AREA_RELIABILITY***, and partly fulfills ***REQ:TOOL:MASKING*** due to it being easily expanded to implement control-flow TMR, it does not fulfill the other tool-level requirements discussed in Section 3.1. A summary of this evaluation is shown in Table 27.

Table 27: Evaluation of *StitchUp* [192] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	◑	*StitchUp* introduces DMR, but extension to TMR would be trivial
REQ:TOOL:DSE	○	No optimization is carried out, the entire control flow is duplicated.
REQ:TOOL:AREA_RELIABILITY	●	Area and reliability are traded-off by only protecting the control flow.
REQ:TOOL:PARETO	○	One design candidate is generated by the approach.
REQ:TOOL:PNR	○	Placement and routing do not play a role in the replication decisions made by *StitchUp*.
REQ:TOOL:BITLEVEL	◑	Bit-level fault effects do not play a role in the replication decisions made by *StitchUp*.
REQ:TOOL:PVT	○	PVT effects are not considered.

The approach *TLegUp*, introduced in [193], introduces TMR into a target design during the HLS process. The type of replication (simplex or TMR), voter type (combinational or registered), and voter locations are provided in the form of user-defined constraints. The approach first inserts voters into the LLVM IR (to be able to take sequential voters into account when generating the design's schedule) and subsequently synthesizes RTL for this IR. *TLegUp* supports inserting voters into datapaths, Finite State Machines (FSMs), memory, or at the top level of modules. Finally, the approach introduces replications into the generated RTL design description. As it introduces TMR, the approach fulfills the requirement ***REQ:TOOL:MASKING***. Since the entire design is triplicated, and voter insertion is user-directed, this approach does not fulfill the requirements ***REQ:TOOL:DSE***, ***REQ:TOOL:AREA_RELIABILITY***, and ***REQ:TOOL:PARETO***. [193] compares the effectiveness of combinational and sequential voters in *TLegUp* in reducing SEU susceptibility by performing configuration-level fault injection experiments. However, the results are not compared to a reference implementation that only introduces TMR on RTL. Furthermore, *TLegUp* does not address bit-level effects or effects of placement and routing, thus not fulfilling requirements ***REQ:TOOL:PNR***, ***REQ:TOOL:BITLEVEL***, and ***REQ:TOOL:PVT***. This is partly addressed in [194], which extends *TLegUp* with function-level and instruction-level partitioning to implement fault containment regions that can be physically separated. A summary of this evaluation is shown in Table 28.

Table 28: Evaluation of *TLegUp* [193] against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	●	*TLegUp* introduces TMR.
REQ:TOOL:DSE	○	No optimization is carried out; the entire design is triplicated.
REQ:TOOL:AREA_RELIABILITY	○	
REQ:TOOL:PARETO	○	
REQ:TOOL:PNR	○	Placement and routing do not play a role in *TLegUp*.
REQ:TOOL:BITLEVEL	○	Bit-level fault effects do not play a role in *TLegUp*.
REQ:TOOL:PVT	○	PVT effects are not considered.

The following approaches build upon *TLegUp* or its underlying concepts: [195] uses designs generated by *TLegUp* in a case study describing intelligent scrubbing mechanisms that allow recovering the circuit state after a configuration bit flip. [196] proposes an approach alternative to replication, where each FU is simulated in an RTL simulator with multiple random input vectors to determine an expected range of output values, thereby allowing the generation of checker units that supervise the design during operation. These checkers can be linked to the design's control flow FSM to select acceptable ranges depending on scheduling. [197] defines error probabilities for data errors affecting the output more than 20 dB, 30 dB, and 40 dB instead of considering every discrepancy of a design's output from its fault-free form as a potentially critical error as is done in most other tool flows evaluated in this dissertation. Based on these definitions and characterization of the components of an HLS-generated design in simulation on the level of its LLVM IR representation, they harden the design selectively using TMR to improve the cost-effectiveness of SEU hardening.

While it is difficult to address the bit-level and placement and routing requirements during the HLS process, design analyses can proceed faster at the higher levels of abstraction these tools use. This could be employed to perform fast but coarse-grained sensitivity analysis that may inform a lower-level reliability improvement flow or tool as is shown via the forward-annotation of replicated units in [190, 191]. The exact logic-level reliability in SRAM-based FPGA designs, however, strongly depends on the implementation on the bit level, e.g., via placement and routing or bit-level PVT effects. Furthermore, while HLS tools have gained a certain adoption in, for example, signal processing and machine learning acceleration (see hls4ml [198] and finn-hlslib[45]), many designs are still done conventionally using (System)Verilog or Very High Speed Integrated Circuit Hardware Description Language (VHDL) on RTL or employ RTL generators (e.g., Chisel (Scala)[46], SpinalHDL (Scala)[47], MyHDL (Python)[48] or Amaranth (Python)[49]). Solely employing an HLS-based approach to introduce redundancy or otherwise improve reliability would exclude designs implemented using these toolflows. Conversely, many low-level approaches apply to HLS-generated designs, as they are often generated as a RTL description or synthesized to a netlist. Many of the discussed approaches work on the netlist level (or lower), which allows them to take the placement and routing better into account.

3.3.5 Summary

The sections above discussed a diverse set of reliability-improvement approaches that are in one way or another aware of the cost of introduced reliability improvements, for example, in the form of increased area usage (i.e., FPGA resource utilization), increased critical path delay, or increased power dissipation. A summary of the degree of fulfillment of the tool-level requirements introduced in Section 3.1 of the discussed approaches can be seen in Table 29. As the analyzed approaches in the category of reliability-aware placement and routing (see Section 3.3.1) are of similar quality related to these requirements, they have been summarized into a single column.

In a similar way to the bit-level fault analysis approaches described in Section 3.2.10, for each approach, an *approach score* indicating the total fulfillment of the formulated requirements (under

[45] See `https://web.archive.org/web/20250912151839/https://github.com/Xilinx/finn-hlslib`, accessed 2026-01-05

[46] See `https://web.archive.org/web/20250807221323/https://www.chisel-lang.org/`, accessed 2026-01-05

[47] See `https://web.archive.org/web/20250830230033/https://github.com/SpinalHDL/SpinalHDL`, accessed 2026-01-05

[48] See `https://web.archive.org/web/20250807082707/https://www.myhdl.org/`, accessed 2026-01-05

[49] See `https://web.archive.org/web/20250806171143/https://github.com/amaranth-lang/amaranth`, accessed 2026-01-05

the assumption that all of these are weighted uniformly), as well as a *SoTA* score showing the fulfillment of the respective requirement by the summarized state of the art are shown in Table 29. This reflects this dissertation's assessment of the individual approaches, as well as its assessment of the technological progress toward individual goals. Furthermore, as approaches of different types have been analyzed in this section, an *average approach type score* is reported in Table 29 to quantify the assessment of the suitability of the different approach types for the field of application described by the tool-level requirements formulated in this dissertation.

It can be seen from Table 29 that no individual approach fulfills all requirements, with the approaches published in [180] and [190, 191] performing best in this evaluation. Their shortcomings are shared by many of the analyzed approaches: It is difficult to accurately address the effects of changes in placement and routing that are necessarily introduced into the design when it is altered by introducing partial redundancy when using pre-generated or pre-analyzed design variants. The placement and routing process is addressed using a variety of heuristics [199], making its outcome hard to predict on a bit-by-bit level necessary to estimate the resulting design area and reliability accurately. Furthermore, bit-level effects such as specific interconnect fault models or bit-level LUT fault modeling are not implemented or addressed by most of the analyzed approaches, which further prevents optimization algorithms – especially relevant for DSE approaches, see Section 3.3.3 – from considering these effects. Finally, reliability analysis at the bit-level would also allow an approach to consider PVT effects, such as certain faults having different effects depending on environmental conditions such as temperature or operating voltage or across different instances of the same FPGA model. However, none of the analyzed approaches that carry out bit-level analysis address these effects or acknowledge their existence or influence on the criticality evaluation that they carry out. Thus, also considering the *SoTA* score shown in Table 29, a reliability improvement approach aiming at improving on the state of the art should especially (1) address co-optimization of resource usage and criticality, and in the course of this co-optimization, aim at (2) considering the effects of bit-level fault models, (3) placement and routing, and especially (4) PVT on the reliability of the generated solution(s). As per the requirements formulated in Section 3.1, an approach addressing these points should be based on a form of area-reliability design-space exploration.

Furthermore, Table 29 shows that high-level approaches have difficulties addressing low-level effects such as placement and routing and bit-level fault effects unless paired with a low-level placement and routing approach as done in [190, 191] which introduces other issues. Nevertheless, an approach of this kind may benefit from a reliability analysis that is more efficiently executed at a higher level of design abstraction. While configuration-level fault effects such as the effects of short-circuiting two closely routed signals or common-mode faults between two LUTs placed within the same slice – which may contain configuration bits that affect all flip-flops in this slice – may be better addressed in a low-level approach, a fast high-level analysis may deliver suggestions for replications that address faults that are rather insensitive to placement or routing effects, such as the susceptibility of the *data content* of user-defined storage elements. In RTL designs, this information may alternatively be gathered using conventional logic simulation, albeit at a lower speed of analysis.

Table 29: Summary of Cost-Aware FPGA Reliability Improvement Approaches

	Approach (Type)												
	P&R-based Section 3.3.1	Selective Replication Section 3.3.2			Design-Space Exploration Section 3.3.3					High-Level Synthesis Section 3.3.4			
Requirements (see Section 3.1)	[99, 152], [155], [158]	[159]	[160, 161]	[166]	[170]	[172]	[174]	[175]	[179]	[190, 191]	[192]	[193, 194]	SotA Score
REQ:TOOL:MASKING	○	●	●	◑	●	●	●	○	●	●	◑	●	9.0 / 12.0
REQ:TOOL:DSE	◑	○	◑	○	●	●	◑	●	●	●	○	○	6.5 / 12.0
REQ:TOOL:AREA_RELIABILITY	◑	○	◑	○	●	○	○	○	●	●	◑	○	**4.5 / 12.0**
REQ:TOOL:PARETO	○	○	◑	○	●	●	●	●	●	○	○	○	**5.5 / 12.0**
REQ:TOOL:PNR	●	○	○	○	○	○	◑	●	○	●	○	○	**3.5 / 12.0**
REQ:TOOL:BITLEVEL	●	○	◑	○	○	○	○	●	◑	◑	○	○	**3.5 / 12.0**
REQ:TOOL:PVT	○	○	○	○	○	○	○	○	○	○	○	○	**0.0 / 12.0**
Approach Score	3.0	1.0	3.0	0.5	4.0	3.0	3.0	4.0	4.5	4.5	1.0	1.0	
Avg. Approach Type Score	3.0	1.5			3.7					2.17			

3.4 Motivation of the Research Gap for this Dissertation

In the previous sections, the current state of the art concerning approaches towards bit-level fault modeling and prediction for SRAM-based FPGAs (see Section 3.2) and cost-aware reliability improvement approaches (see Section 3.3) has been discussed in depth. An evaluation of these approaches against the requirements formulated at the beginning of this section (see Section 3.1) revealed that significant gaps exist in the current state of the art for reliability improvement approaches targeting SRAM-based FPGAs. These gap can be summarized as follows.

Gap 1: The (selective) introduction of redundancy into SRAM-based FPGA designs is a well-known approach to improve the resilience of designs implemented in these devices. However, the state of the art does not offer a comprehensive DSE approach that co-optimizes resource usage and configuration-level reliability that is coupled with bit-level reliability analysis. In this context, bit-level analysis is crucial for SRAM-based devices to correctly evaluate the benefits of "spending" additional FPGA resources on replicating parts of the design concerning overall reliability. This goes hand-in-hand with the analysis of the effects of RTL or netlist alterations due to the insertion of redundancy and voters on design placement and routing, which in SRAM-based devices can significantly impact overall design reliability. Furthermore, with ever-shrinking feature sizes, the magnitude of PVT effects, manifesting themselves in inter-chip and intra-chip variation, increases [200]. These effects may especially affect the behavior of faults that violate the cleanly defined digital behavior of an FPGA (e.g., floating wire segments due to open faults and fault-induced short circuits). These properties and effects are not sufficiently considered in today's cost-aware reliability improvement approaches (see Table 29). A DSE approach that does not adequately cover bit-level effects and placement and routing may generate selective-redundancy variants that – while attempting to mask errors in one design unit – may inadvertently introduce more critical bits elsewhere [165]. A DSE approach that does not consider PVT effects might generate design implementations that perform well on one device but less so on another, otherwise identical, device of the same model. An approach considering these effects has – to the best knowledge of the author – not been published at the time of writing this dissertation. This gap motivates the development of the goal **G2** laid out in Chapter 1.

Gap 2: To adequately consider configuration bit-level effects on design reliability – and thus also treat the effects of placement and routing as well as PVT effects – a bit-level fault modeling approach is necessary. This approach needs to perform the three essential steps of (1) associating bits in the bitstream with resources in the netlist, (2) modeling the effects of SEUs affecting these bits according to a technology-specific fault model, and (3) analyzing the criticality of these bit flips in the context of the netlist and potentially introduced fault tolerance measures. The first two steps of this analysis can be carried out statically. In contrast, the third step may require dynamic analysis as some faults may be state- or data-dependent, being only triggered if the circuit is brought into a certain state – thus requiring fault simulation or fault injection experiments. Furthermore, the static fault models may not be accurate enough, or the fault effects not predictable enough for purely static analysis, requiring fault injection experiments on multiple real-world specimen devices and potentially under varied operating conditions to address PVT effects. The current state of the art provides several fault modeling approaches targeting a variety of FPGA devices. As shown in Section 3.2.10, however, these approaches require improvement. This especially concerns the following dimensions:

- Bit-by-bit fault consequence prediction and criticality analysis: Several contemporary ap-

proaches aim solely at predicting the *number* of critical bits – and thus the error cross-section – but not the individual fault effects, which is necessary when assessing which parts of the design to replicate and which to leave as-is.

- Systematic empirical characterization of the effects of bit flips violating defined digital FPGA behavior is required to formulate accurate fault models. Characterization also may reveal categories of faults to be assessed further for PVT effects. This requires the design and execution of systematic interconnect fault injection experiments, as modern FPGAs provide various types of interconnect resources and wire types that may behave differently in the presence of faults.
- Modeling or – at least – mapping to netlist resources of faults in special-purpose resources such as BRAM or DSP cells, as bit-level reliability assessment approaches frequently neglect these.
- Systematic bit-by-bit validation of the static criticality predictions made by the approach. This can be done by performing exhaustive fault injection experiments on a set of benchmark designs and comparing their results to predictions made by the approach. This allows an assessment of the specificity and sensitivity of these predictions and may highlight areas for future improvement.

These shortcomings form **Gap 2**, which motivates the development of the goal **G1** as described in Chapter 1.

Gap 3: While design (reliability) analyses conducted at a high level of abstraction form the basis of HLS-based reliability improvement approaches, these approaches on their own are ill-equipped to address low-level reliability effects related to the precise placement and routing implemented for the target FPGA, as well as bit-level and PVT-related effects. Furthermore, conducting design analysis solely in the course of an HLS process precludes designs using different design entry methods (e.g., using in the form of descriptions in an HDL on RTL) from benefiting from these reliability improvements. Netlist-level DSE approaches are preferable to improve tolerance concerning errors affecting the design's logic or routing resources (i.e., its cell's functionality and the design topology). However, high-level approaches of this kind may still benefit from the faster analyses that can be carried out in a higher level of abstraction – especially if the data content of the user-defined storage elements (flip-flops and BRAMs) are concerned. As current HLS-based approaches are predominantly focused on improving the reliability of only the portions of a design that are implemented using HLS, there is a lack of high-level analysis approaches that can be combined with netlist-level DSE approaches to improve total design reliability. This motivates the development of goal **G3** as summarized in Chapter 1.

This dissertation describes, to the best knowledge of its author, the first comprehensive approach to take the aforementioned gaps into account to address the requirements formulated at the beginning of this section, resulting in the goals **G1–G3** of this dissertation laid out in Chapter 1. The following chapters detail the contributions made in the course of addressing these goals in this work and are structured as follows:

Goal G1: With respect to the goal **G1**, Chapter 4 describes bitstream-level fault modeling and criticality prediction based on PVT-aware characterization experiments for the AMD 7 Series as well as Lattice iCE40 and ECP5 FPGA families. The toolflow for AMD 7 Series is extensively validated using three variants (degrees of applied partial TMR) of a set of benchmark designs.

Goal G2: Chapter Chapter 5 addresses the goal **G2** by proposing a concept and implementation of a DSE toolflow that co-optimizes reliability and resource usage by generating Pareto-optimal combinations of a hierarchical design's modules using a genetic algorithm, as well as a combination of static analysis (see Chapter 4) and fault-injection-based fitness evaluation. Furthermore, an in-depth evaluation of the approach and its implementation is performed on a set of benchmark designs, consisting of designs generated by HLS and RTL designs.

Goal G3: Finally, Chapter 6 addresses the goal **G3** developed from it by describing a high-level analysis approach to determine especially critical storage elements in a design during its HLS process by simulating an intermediate representation of the design. The analysis approach and related proposed metrics are evaluated on a set of benchmark designs.

Chapter 4

Bitstream-Level Fault Modeling and Criticality Analysis

As argued in Section 3.1.2, bitstream-level reliability analysis and prediction are central requirements for the cost-efficient implementation of reliability improvements within a given FPGA design. For example, the reliability information extracted or predicted by this process may be used to assess the cost and benefits gained from a particular design variant incorporating module-based partial TMR. Ideally, this reliability analysis and prediction step receive the design's bitstream (i.e., the individual configuration for each FPGA resource) and information concerning its placement and routing. By combining this information with pre-existing knowledge about the particular FPGA technology and technology-dependent fault models, it first determines the expected effect on the design for each considered fault. This includes the (potentially) affected subset of the design's logic cells and the nature of the faults. This information is then further combined with (1) design-related masking information – e.g., the TMR replica membership of the affected cells – and (2) technology-dependent criticality prediction rules. The outcome of the bitstream-level reliability analysis and prediction step is a prediction of (potential) criticality for configuration bits in the design's bitstream. Depending on the accuracy of this prediction, it may be used directly as a measure of the design's reliability or to guide further, more time-consuming reliability analysis such as fault simulation or fault injection.

This section proposes a bitstream-level reliability analysis and prediction approach guided by the bit-level requirements formulated in Section 3.1.2. Figure 6 depicts an overview of the proposed approach. Where applicable, it contains references to the relevant sections of this chapter.

The proposed approach starts with the results of a previous implementation of the target design. That is, logic synthesis, placement and routing, and bitstream generation have been completed before this analysis, resulting in a bitstream representing the design's FPGA configuration and a logical representation of the decisions made by the implementation tool during the placement and routing process (i.e., the correspondence between the design's netlist cells and their placement locations, and between the design's nets and the interconnect resources employed by the router). In addition to the bitstream and P&R information, implementation tools may also support (coarse) reliability analysis.

For example, AMD Vivado offers an *Essential Bits* feature [88, 108] that generates a bitstream mask (referred to as the "Essential Bits File"). This mask indicates which bits in the bitstream are related to the design (i.e., may become critical) and which bits relate to unused resources that may not affect the design if their configuration changes due to a fault. While not strictly required, this

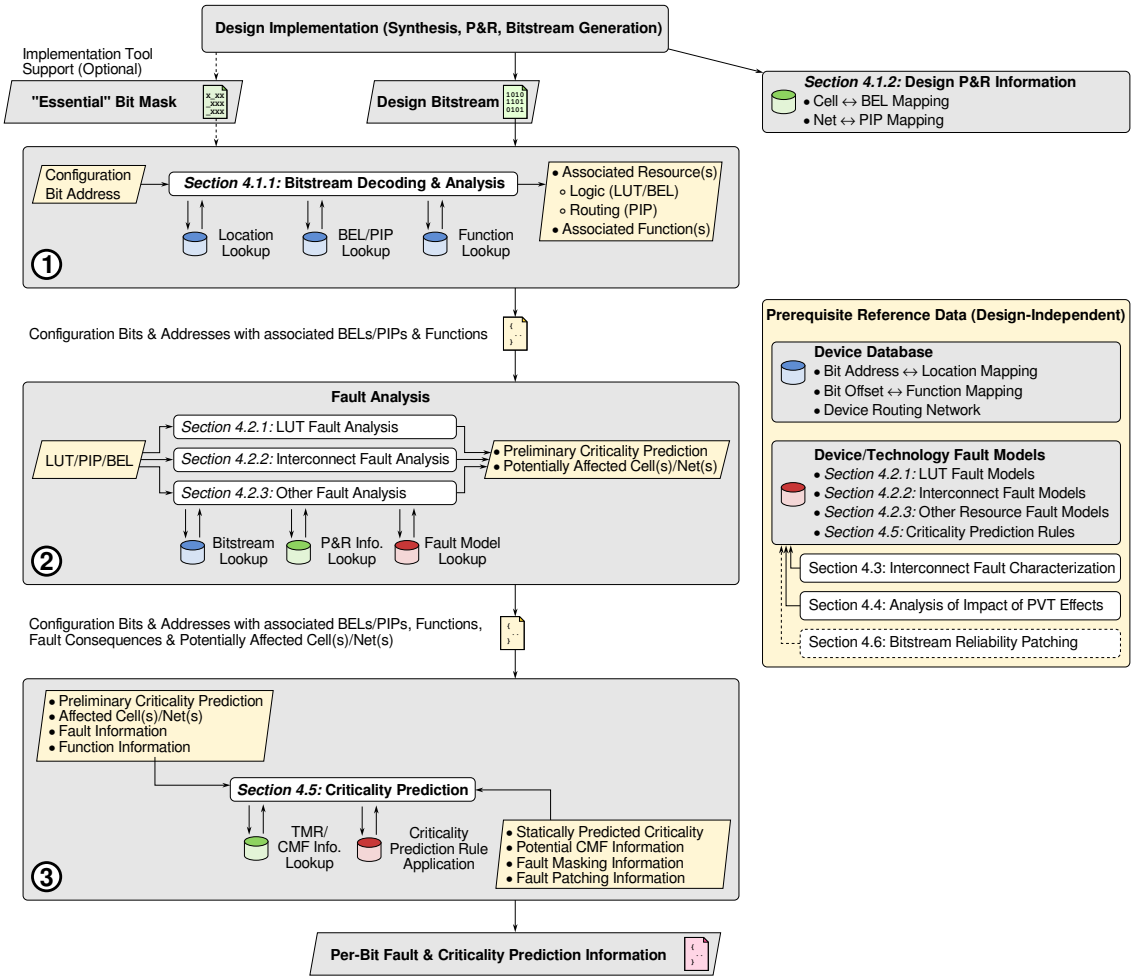


Figure 6: Flowchart of the Proposed Per-Bit Criticality Prediction Approach

information can be used to prioritize certain bits in the bitstream for analysis.

Furthermore, the approach requires design-independent information about the technology and device to proceed. This information can be retrieved with the help of a *device database* as described in the following.

As a first step, a device database containing information to identify the physical location within the device that is affected by a particular configuration bit is required. As described in Section 2.2 modern “island-style” devices commonly are organized in a grid of similar collections of basic logic or routing elements, often referred to as “tiles”. From a bit’s configuration address (or offset), the physical tile location can often be extracted directly (e.g., with the tile coordinates being a part of the address) or with the help of a reference table part of the device database.

Having identified the affected FPGA subdivision (e.g., tile), the device database must then allow the identification the affected resource (e.g., a particular LUT, FF or interconnect multiplexer). The remainder of the configuration bit address (e.g., an offset) commonly has the same meaning for each instance of a given tile type. For example, a particular offset addresses the same LUT configuration bit for each logic tile. The device database is commonly de-duplicated by providing an offset-to-resource (and function) mapping for each device tile type. Using this part of the device database, the affected resource and the function of the affected configuration bit (e.g., LUT initialization or switching between LUT and distributed RAM mode) is identified.

Finally, the device database must also contain information about the device's routing network. On the one hand, this means a mapping of configuration bits to routing configurations, i.e., which configuration bits to set to configure individual routes. Furthermore, information on which connections are possible between which tiles and how these connections are structured (i.e., unidirectional vs. bidirectional/tristate) must be available.

In addition to the device database, the approach requires knowledge about the behavior of each fault type that is possible under the assumed fault model, including information on how PVT effects may affect the reliability of the design and the potential ability of certain faults to be patched by modifications to the bitstream alone (without significant re-implementation).

Making use of the technology and device-related information described above, the reliability analysis and prediction approach proposed in this dissertation consists of three major steps. These steps are depicted in Figure 6. They are briefly summarized and discussed within the bit-level requirements formulated Section 3.1.2 in the following.

1. In the first step, **low-level bitstream parsing and design analysis**, the design's bitstream is decoded into configuration bit addresses, followed by the association with device location (tile) and resource/function by cross-referencing these addresses with the device database (see Section 4.1.1). At this point, the design's bitstream may be pre-filtered by reliability analysis functionality provided by the implementation tool such as an *Essential Bits* file. This results in an augmented list of the design's configuration bit addresses with their associated logic or interconnect resources and these resources' affected function(s). This forms the basis for addressing ***REQ:BIT:MAPPING***. Furthermore, the required placement and routing information is extracted in this step (see Section 4.1.2).

2. In the second step, static **bit-level fault analysis** (see Section 4.2), the effect of each individual fault is assessed. This is done by considering the context of (1) the device configuration, (2) placement and routing information, as well as (3) technology-dependent fault models. The association between configurable FPGA element and the targeted design forms the second step of addressing ***REQ:BIT:MAPPING***.

   In this dissertation, configuration faults are modeled as single-bit flips persisting for the entire execution duration of the targeted design. Thus, the fault analysis step assesses the effects of each individual (essential) configuration bit flipping from the value defined by the bitstream to its complementary value. Depending on the type of resource and its function affected by the fault, this bit flip must be considered in a greater context of configuration bits, i.e., all bits that configure that resource or function.

   The description of the applied fault models is split into three subsections in Section 4.2 that address ***REQ:BIT:MODELING:LOGIC***, ***REQ:BIT:MODELING:INTERCONNECT***, and ***REQ:BIT:MODELING:SPECIAL***, respectively:

   (a) Section 4.2.1 models faults in the initialization content of LUTs.

   (b) Section 4.2.2 models faults affecting interconnect resources.

   (c) Section 4.2.3 describes efforts towards modeling faults in other FPGA resources.

   Especially for interconnect-related faults, developing an accurate fault model requires prior identification of the target device's behavior under the faults to be modeled. The related *fault characterization* described in Section 4.3 employs purpose-built fault injection experiments to determine this behavior. Both a general approach towards these characterization experiments as well as the results of applying this process to AMD 7 Series, Lat-

tice iCE40, and Lattice ECP5 FPGAs are described. This addresses the requirement ***REQ:BIT:CHARACTERIZATION***.

Furthermore, as PVT effects have been observed as an influence on fault injection results [139, 201], PVT characterization is beneficial. To address ***REQ:BIT:PVT***, Section 4.4 describes an experimental setup for identifying the presence and effect of PVT variations in fault injection results, and discusses results obtained on AMD 7 Series and Lattice iCE40 devices.

It is also possible that the fault characterization indicates that specific faults may be easily "patched" by changing the initialization of certain cells directly in the bitstream. For example, in [156, 155] the logical function of LUTs is inverted wherever possible to avoid especially error-prone configurations. Section 4.6 proposes and evaluates an approach for patching bitstreams for AMD 7 Series devices to improve resilience against certain interconnect faults.

In the form described in this dissertation, the **bit-level fault analysis** step augments the list of the design's configuration bit addresses. For each of these addresses, it assesses information about (1) the type of fault caused by an SEU affecting each bit, (2) the design's logic cells potentially affected by the bit flip, and (3) a preliminary criticality prediction for each individual fault.

In the described form, this process is purely static, as only data concerning the design's structure (device configuration, placement and routing information) is used in this assessment. However, if dynamic design information concerning its run-time behavior exists, it may be integrated into this analysis step. An approach developed in parallel to the writing of this dissertation [124] aims at integrating simulation traces into fault analysis.

In the described form, this process is geared towards single-bit errors. To assess multi-bit upsets, first, a list of multi-bit errors likely to occur according to a given fault assumption (considering, e.g., physical configuration memory layout, energy of considered radiation or charge of involved particles, or radiation characterization data) needs to be generated. The effects of these faults on the bitstream can then be modeled in a similar way to single-bit errors, with special consideration needing to be applied to multi-bit upsets affecting the same resource (e.g., a LUT or an interconnect multiplexer).

3. Section 4.5 describes the final proposed step, **reliability prediction**, that addresses ***REQ:BIT:BITWISE_PREDICTION***). In this step, the information gathered about each fault is used to form a prediction of the criticality of the fault to the design's correct functionality. The data for each fault considered by this decision-making process includes the affected resource and function type and the set of (potentially) affected netlist cells. By combining this with information on the membership of these cells in TMR replica groups, the prediction step also identifies TMR-violating CMF conditions. Static criticality prediction rules formed from fault modeling and characterization then allow to form an assessment of the criticality of each single bit flip. To ensure the validity and applicability of this process, both the accuracy of the formed predictions (***REQ:BIT:VALIDATION***) as well as the performance (***REQ:BIT:RUNTIME***) must be evaluated.

The outcome of this assessment process is a list of single-bit flips in the configuration memory that are considered to be able to critically impair the design's functionality. This results in a reliability prediction method fulfilling the requirements formulated in Chapter 3, thereby being fit for use in higher-level cost-effective reliability enhancement approaches such as bit-level DSE for partial TMR. The detailed steps necessary to accomplish the proposed assessment approach are discussed in the following sections.

This chapter's core contributions to the state of the art are:

1. A systematic characterization of interconnect fault consequences on multiple device technologies using differing interconnect architectures and configuration encoding (see Section 4.3). While interconnect fault characterization was described in previous works [46, 131, 124], this was only done on a small scale and exclusively for AMD devices.

2. An approach to evaluate PVT effects on the results of systematic fault injection experiments in AMD 7 Series and Lattice iCE40 devices, as well as results of an in-depth evaluation of these effects (see Section 4.4). The influence of environment temperature is investigated on AMD and Lattice devices, while the impact of supply voltage is demonstrated on AMD devices. While a small number of previous works either implicitly acknowledge the possibility of these effects by repeating experiments on the same device [139] or multiple devices [121], other works implicitly generalize from one device to all other devices of the same type by suggesting to speed up fault injection experiments by parallelization [140, 141].

3. A well-validated bitstream-based criticality prediction flow for single-bit errors incorporating post-implementation information as well as TMR CMF analysis, providing adequate performance for use in a fitness function of a DSE approach (see Section 4.5). While earlier works [121] and works published in parallel to the writing of this dissertation [124] propose similar approaches, no bit-by-bit validation of their predictions (i.e., discussion of false-positives and false-negatives generated by these approaches) have been published to date. While an implementation of the approach is presented for AMD Artix 7 devices, the underlying modeling, characterization, and PVT analysis concepts apply to any FPGA technology.

4. A discussion and evaluation of bitstream patching for reliability improvements in AMD 7 Series FPGA by exploiting characteristics of the employed FPGA technology. Related work on custom FPGA architectures [156] addresses bridge faults in binary encoded multiplexers by inverting certain LUTs to reduce boolean differences in the waveforms generated in these potentially bridged signals, requiring in-depth knowledge of the design's functionality. The approach presented in Section 4.6 of this dissertation for AMD 7 Series devices is a purely static, zero-overhead patching method that does not alter (parts) of the design's functionality. It merely changes the initialization of unused resources, thereby increasing the number of benign interconnect faults.

Finally, Section 4.7 concludes this chapter. It thereby assesses the described contributions according to the requirements put forward in Section 3.1, contrasts them to the assessment of the relevant state of the art found in Section 3.2, and discusses directions for further possible improvements.

4.1 Low-Level Bitstream Parsing and Design Analysis

The goal of the bit-level fault modeling and analysis approach discussed in this chapter is to identify the consequences of each bit flip in the configuration of a target FPGA device implementing a given design. It is assumed that each configuration bit in the device's configuration memory is either unused or associated with a particular logical or routing-related configurable element in the FPGA's fabric. Furthermore, it is assumed that each configuration bit is associated with a particular address that (1) denotes its position in the configuration bitstream and (2) allows the association with the logic or routing-related element it configures. This address may also further be

used in fault injection operations, as is done, for example, with AMD's Soft Error Mitigation (SEM) IP core [108].

4.1.1 Bitstream Decoding and Analysis

An essential first step is to extract the device configuration, that is, how each configuration bit in the device is set for it to perform the functionality constituting the given design. This data is commonly found in the FPGA configuration file (often referred to as "bitstream") generated by the implementation tool from a representation of the design's placement and routing.

In SRAM-based FPGAs, this configuration file is commonly loaded into the FPGA's configuration logic from a non-volatile source such as an external Flash chip. This configuration logic commonly not only supports loading the configuration data directly into the device's configuration memory. Often, it provides additional device management functionality such as reading a device ID, triggering a reboot, and controlling the commonly present configuration Cyclic Redundancy Check (CRC) logic. The device vendor specifies which sequence of operations on this configuration interface is necessary to bring the FPGA device into a mode where it accepts data to be loaded into configuration memory.

For example, in AMD 7 Series devices, the bitstream consists of *configuration packets* that are addressed to registers internal to the configuration logic. A packet header contains an opcode (i.e., read, write, or no-operation), a destination register address, and a payload length, followed by the quantum of data specified in the header. The sequence of writes to the configuration logic's internal registers that needs to be followed to configure the device correctly is specified in [30]. At minimum, a bitstream in this technology consists of the following steps:

1. A synchronization step that includes bus width detection and is concluded with a predefined 32-bit synchronization word.
2. Writing the vendor-defined device ID to the configuration logic's IDCODE register, with the device only unlocking the configuration memory if the correct device ID is provided – thereby preventing the inadvertent load of configuration data intended for a different device.
3. Writing the start address in the configuration memory where the configuration data shall be loaded to the configuration logic's *Frame Address Register*,
4. Writing the configuration data to the configuration logic's *Frame Data Input Register*, thereby writing directly to the device configuration.
5. Performing a CRC check and de-synchronization to finalize the transaction.

The exact details of these operations are omitted here for brevity and can be found in [30]. For other devices, the corresponding information is commonly published by the device vendor. As examples, Lattice's iCE40 and ECP5 devices follow a similar configuration pattern as described above for AMD 7 Series devices [202, 203].

The description above indicates that a first step in bitstream parsing is the identification and extraction of the relevant portion of the bitstream file containing the device configuration data – stripping synchronization and protocol information. This device configuration data is then presented as a sequence of configuration words or bits. The information indicating the assignment of each configuration bit to a certain configuration memory storage element in the FPGA is implicit to the bitstream (with the exception of a starting frame address, e.g., in AMD 7 Series devices). Thus, as a next step, a bitstream decoder must replicate the addressing behavior of the FPGA's

configuration logic in order to associate each configuration bit with a unique bit address that can subsequently be used to associate the bit with logic or routing-related resources in the FPGA.

Third-party bitstream documentation projects provide this functionality for the FPGA technologies applied in this work:

- For AMD 7 Series devices, Project XRAY[50] provides the `bitread` tool that receives a binary bitstream as its input, identifies the synchronization word and subsequent configuration memory write access (see above), and finally outputs the bit addresses of each bit in the bitstream. Alternatively, *BFAT* (see Section 3.2.7) provides a Python script that includes similar functionality. Each bit is associated with a *frame address*, identifying the smallest reconfigurable portion of AMD 7 Series FPGAs consisting of 101 32-bit words, as well as a *word* and *bit* address to identify the respective 32-bit word within this frame, and subsequently one bit within this word. The frame addresses employed by Project XRAY and *BFAT* are *physical frame addresses* [108], containing coarse information about the physical location of the configured frame (among others the frame's clock region, tile row, and tile column). This is also the frame address format employed in the functional bitstream documentation information provided by Project XRAY.
- For Lattice iCE40 devices, bitstream decoding functionality is provided by the tool `iceunpack` provided by Project IceStorm[51]. This tool separates the configuration data into configuration words associated with each tile in the FPGA, with the further bitstream documentation information referring to the bits stored in these words[52].
- For Lattice ECP5 devices, the `libtrellis` and `pyTrellis` Application Programming Interfaces (APIs) provided by Project Trellis[53] allow loading bitstreams and iterating over their contained configuration bits in C++ programs or Python scripts, respectively.

Once each bit is associated with a configuration bit address by the means discussed above, bitstream documentation is used to identify the associated physical location (e.g., tile or site), as well as the logical or routing-related resources related to this bit. However, doing so for all bits in the bitstream may be prohibitively costly, as FPGAs may contain millions of configuration bits, of which only a fraction is ever used by the design – or able to cause faulty behavior if flipped.

As noted in Section 2.3, AMD ISE and Vivado provide the *Essential Bits* feature, allowing the generation of a mask for the bitstream, identifying the configuration bits that are in any way related to the design as deemed by the device vendor. Of these essential bits, only a fraction may behave critically if flipped [108]. Thus, this "essential bits data" is used in the work carried out on AMD FPGAs for this dissertation to filter the bitstream before any further analysis as is common practice in related work (see Chapter 3).

The open-source NextPNR implementation tools employed for Lattice iCE40 and ECP5 do not output this data in a similar way. In these devices, a heuristic can be employed: for example, the placement and routing data of the design (see Section 4.1.2) may be consulted to identify in which FPGA regions the design's cells have been placed and nets have been routed, and prioritize these regions when identifying potentially design-associated bits. This heuristic, however, must additionally consider that some configuration bits outside FPGA tiles utilized by the design may

[50] See `https://web.archive.org/web/20250630071412/https://github.com/f4pga/prjxray`, accessed 2026-01-05

[51] See `https://web.archive.org/web/20250211084008/https://github.com/YosysHQ/icestorm`, accessed 2026-01-05

[52] See, e.g., `https://web.archive.org/web/20250426192831/https://prjicestorm.readthedocs.io/en/latest/logic_tile.html`, accessed 2026-01-05

[53] See `https://web.archive.org/web/20250126225938/https://github.com/YosysHQ/prjtrellis`, accessed 2026-01-05

affect the design. This is, for example, the case with bits affecting the global routing network as well as long and bidirectional wires that may be driven from multiple locations within the FPGA.

The exact process to implement the association between bit address and related FPGA resource depends on the format provided by the device's configuration documentation (i.e., the device database). FPGAs are commonly structured regularly, e.g., as a grid of logic "tiles", often organized in columns or rows of similar tiles as described in Section 2.2. Thus, a common format for the configuration documentation in third-party bitstream documentation projects separates the information on (1) the association of configuration frame addresses (and offsets within the frames) with positions in the device's grid of tiles including the tile type associated with each frame and (2) the association of each configuration bit with a configurable resource and its corresponding function for each tile type. This format is, for example, used for AMD 7 Series devices[54] and UltraScale+[55] as well as Lattice ECP5[56].

In the following, the association process is demonstrated for an exemplary configuration bit in an AMD Artix 7-35T (xc7a35t) FPGA.

The bitstream documentation for AMD 7 Series devices views the device as a grid of tiles subdivided by clock regions. Each column of similar tiles within one clock region is associated with multiple frames (with this number depending on the column type) bearing subsequent frame addresses. Each frame consists of 101 subsequent 32-bit words (i.e., 3232 bits) [30]. The configuration for a particular tile consists of a "window" (i.e., a collection of words) into multiple subsequent 101-word frames starting at a particular word offset. A given configuration bit in the bitstream is addressed by a 3-tuple (F, W, B) with F being the frame address, W the word offset within this frame, and B the bit offset within this word. The configuration documentation for each tile type, however, is given as a 2-tuple (O_F, O_B). Here, O_F refers to the index of the frame within all frames configuring this particular column of tiles. O_F is the difference of the bit's frame address F and the tile column's first frame address F_0 (referred to as *base address* in the device database). In a similar way, O_B refers to the index of the bit within all words configuring this particular tile (i.e., the difference of the bit's index within the frame ($W \cdot 32 + B$) and the tile's first bit offset $W_0 \cdot 32$ (referred to as *offset* and given as word indices W_0 in the device database).

For example, consider the bit address $(120\mathrm{a}_h, 3, 20)$. The following steps must be performed to arrive at the bit's tile(s) as well as offsets O_F and O_B. The procedure is illustrated in Figure 7.

1. For the frame address $F = 120\mathrm{a}_h$, identify the corresponding *base address* F_0 from the *tilegrid* part of the bitstream documentation. This can be done efficiently by pre-processing the *tilegrid* information into an interval tree that stores a collection of all tiles (and respective information such as base word offsets and numbers of words) associated with each encountered frame interval. The 101 words in each frame address a maximum of 50 tiles. Querying this interval tree for AMD Artix 7 yields $F_0 = 1200_h$ as a *base address* and all tiles associated with this F_0. Furthermore, this allows the calculation of $O_F = \mathrm{a}_h$, i.e., 10_d.

2. To identify the affected tile, all tiles whose configuration "window" contains the word offset W must be identified among the tiles addressed by F. In the case of the bit address $(120\mathrm{a}_h, 3, 20)$, the word offset 3 matches two tiles: `CLBLM_L_X36Y51` of type `CLBLM_L` with a base word offset $W_0 = 2$, configured by a two-word window into 36 subsequent frames

[54] See `https://web.archive.org/web/20250115214812/https://f4pga.github.io/prjxray-db/`, accessed 2026-01-05

[55] See `https://web.archive.org/web/20250909163755/https://github.com/f4pga/prjuray-db`, accessed 2026-01-05

[56] See `https://web.archive.org/web/20250126223514/https://github.com/YosysHQ/prjtrellis-db`, accessed 2026-01-05

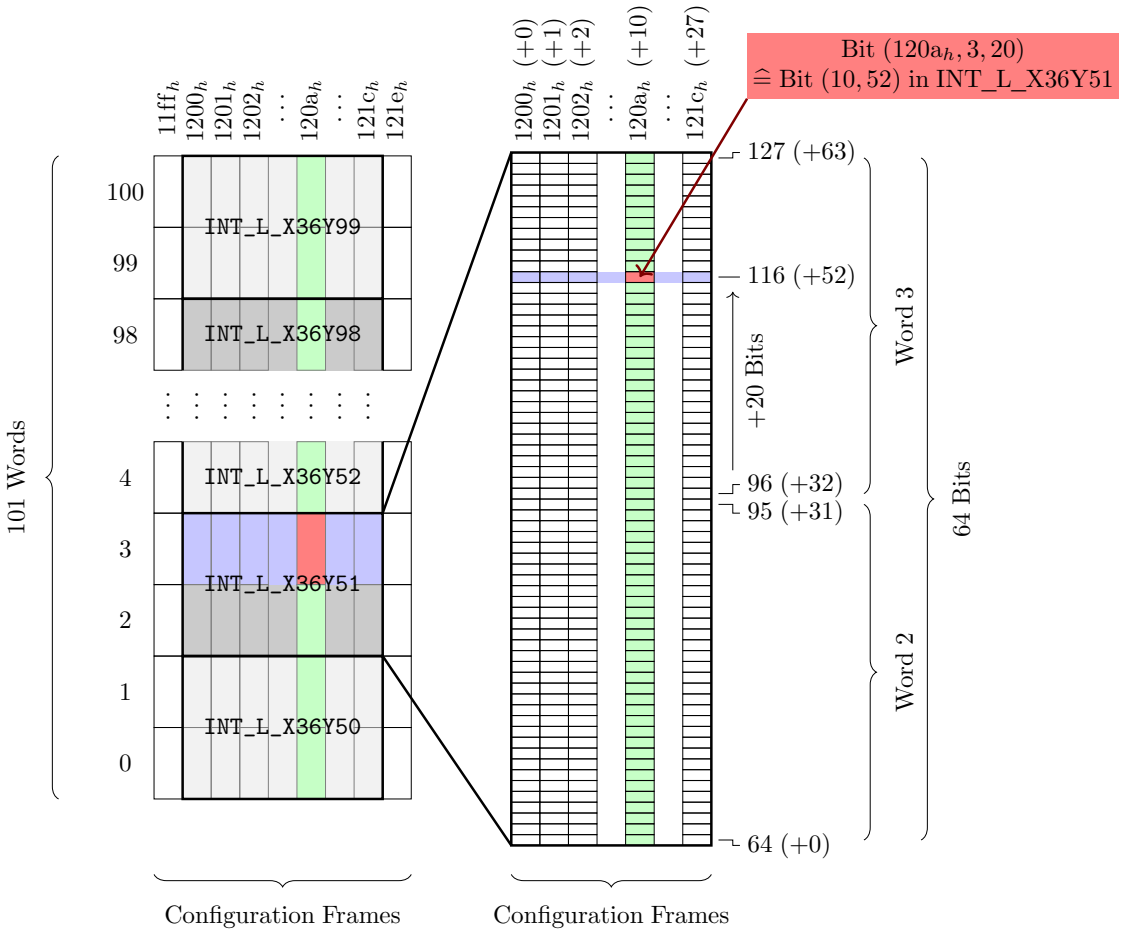


Figure 7: Decoding the Bit Address $(120a_h, 3, 20)$ in an AMD Artix 7 xc7a35t FPGA

(starting at $F_0 = 1200_h$), as well as the tile `INT_L_X36Y51` of type `INT_L` with the same F_0, W_0, and number of (vertical) words as `CLBLM_L_X36Y51`, but configured only by 28 (horizontal) subsequent frames. The matching W_0 allows to calculate the base bit offset of the tile within the frame $B_0 = W_0 \cdot 32 = 64$.

3. O_B is calculated as the difference between the configuration bit's offset within the frame and the base bit offset B_0. For the example above $O_B = W \cdot 32 + B - B_0$ yields $O_B = 52$. Thus, the 2-tuple $(10, 52)$ identifies the configuration bit's offset within both addressed tiles.

Finally, by querying the tile documentation of both identified candidate tile types `CLBLM_L` and `INT_L`, bit $(10, 52)$ is found in the documentation for `INT_L`. The corresponding excerpt of Project XRAY's documentation for the tile type `INT_L`[57] is shown in Listing 1. Listing 1 shows that bit $(10, 52)$ (in the considered `INT_L` tile) is related to the configuration of four connections (i.e., PIPs) to be made by the interconnect multiplexer driving the wire `NE2BEG3`: If either `LOGIC_OUTS_L21`, `NL1BEG_N3`, `NW2END3`, or `NW6END3` shall drive `NE2BEG3` in the tile `INT_L_X36Y51`, this bit must be set to "1" in combination with bits $(15, 52)$, $(12, 52)$, $(14, 52)$, or $(13, 52)$, respectively.

[57] See `https://web.archive.org/web/20250113214512/https://f4pga.github.io/prjxray-db/artix7/segbits_int_l.db`, accessed 2026-01-05

```
[...]
INT_L.NE2BEG3.LOGIC_OUTS_L21 10_52 15_52
INT_L.NE2BEG3.NL1BEG_N3 10_52 12_52
[...]
INT_L.NE2BEG3.NW2END3 10_52 14_52
INT_L.NE2BEG3.NW6END3 10_52 13_52
[...]
```

Listing 1: Excerpt of Project XRAY's Configuration Documentation for INT_L Tiles in AMD 7 Series FPGAs

Repeating the process described above for each bit allows an association of the bit address and affected resource (and the resource's function) to be made. For logic resources (e.g., LUTs, FFs, DSP blocks, and BRAMs), the process is similar to the illustrative example described above. In some cases, which are implicit in the bitstream documentation, one configuration bit can influence multiple cells simultaneously. An example for this condition are the `LATCH` bits for each slice (`X0` and `X1`) in `CLBLL` and `CLBLM` tiles (bits $(30, 32)$ and $(31, 32)$). These bits influence all four register cells capable of both latch and flip-flop behavior in the respective slice. They simultaneously configure all of these cells either as a flip-flop or a latch. When associating configuration bits with affected FPGA resources, this information must be considered.

An analogous process for retrieving the functions associated with each configuration bit address must be implemented if other devices are to be supported. In the case of Lattice iCE40 and ECP5, this process is somewhat similar, as these devices' configurations are organized similarly.

When combined with placement and routing information (see Section 4.1.2), the data retrieved in this step allows to identify the netlist cells or nets associated with the fault. Furthermore, by combining the information of which bit is associated with which (kind of) FPGA resource(s) with low-level fault models for these resources, the individual consequences of each single bit flip can be derived. Section 4.2 further describes this process, supplemented with characterization information described in Sections 4.3 and 4.4.

4.1.2 Extraction of Placement and Routing Information

The information extracted from the bitstream (see Section 4.1.1) must be complemented with placement and routing information to identify the netlist elements affected by individual bit flips. This, subsequently, allows more precise fault modeling and criticality analysis.

Placement information contains the relationship between cells instantiated in the design's netlist and the physical cells (commonly referred to as Basic Elements (BELs)) utilized in the target FPGA device and is the result of all placement and placement-related optimization operations carried out during implementation (see *Cell* $\leftrightarrow$ *BEL mapping* in Figure 6). This also includes information for co-locating combinational functions when the technology provides fracturable LUTs.

Routing information contains all physical wire segments and routing resources ("PIPs") for each net in the design's netlist (see *Net* $\leftrightarrow$ *PIP mapping* in Figure 6).

To be helpful in this regard, this information must include all information to uniquely identify each physical element utilized by the FPGA design. The implementation of this extraction process strongly depends on the FPGA implementation tool used, as this defines in which format this data is available. In the remainder of this subsection, examples for this extraction process are shown for AMD Vivado and the open-source implementation tool NextPNR.

P&R Information Extraction from AMD Vivado

While the (obsolete) ISE implementation tool for AMD FPGAs provided the option to export a post-routing netlist in the reasonably open textual XDL format [128], this is no longer supported by the current Vivado implementation tool targeting state-of-the-art devices (see Section 3.2.2). While ISE supports all AMD 7 Series devices [204], full functionality and support of subsequent device series is only present in Vivado, suggesting the use of this implementation tool.

AMD Vivado allows archiving the current state of a design (pre-synthesis, post-synthesis, post-placement, post-routing) in the form of a *"design checkpoint"* (referred to as a .dcp file in the following by its file extension). This file is, in essence, a ZIP archive of the design containing several files in a proprietary format holding this information. While the format of these files is not publicly documented, there are multiple ways of extracting placement and routing information from .dcp files:

- Vivado allows to load a design from a .dcp file. It is then possible to extract placement and routing information from the implemented design via Vivado's integrated Tool Command Language (Tcl) scripting interface. For this purpose, Vivado provides Tcl commands such as `get_bels` and `get_pips` to retrieve the physical elements of the FPGA that can be filtered by the design's cells or nets [204]. While this allows to extract all placement and routing information, the complexity of modern AMD FPGAs entails intricacies that requires, for example, conditional retrieval of certain elements such as route-through LUTs and intra-CLB routing. Additionally, the limited run-time performance of Vivado's Tcl interpreter limits the suitability of this approach to very small designs – even designs consisting of several hundred cells may contain thousands of PIP.
- As an alternative, AMD provides the *RapidWright* API[58] that allows to parse and manipulate .dcp files using Java or Python (via a Jython[59] interface). This API provides an object-oriented view of the design, and allows to navigate the physical (post-routing) netlist of the design. Thereby, *RapidWright* offers a more convenient way of accessing and extracting placement and routing information than Vivado alone.

This dissertation, thus, employs *RapidWright*'s Jython interface to implement placement and routing information extraction for AMD 7 Series FPGAs. For this purpose, the implementation in this dissertation performs the following steps:

1. Load the .dcp file into *RapidWright*, which in turn constructs a `Design` object that allows to programatically access both the logical (Electronic Design Interchange Format (EDIF)) and physical (post-implementation) netlists.
2. Extract the name and hierarchical path of all cells into Javascript Object Notation (JSON) format for later reference (e.g., for TMR insertion based on the design's module hierarchy) from the netlists stored in the .dcp file.
3. Iterate over FPGA *Sites* instantiated by the design – a *Site* is AMD's terminology for a configurable logic location in the FPGA (e.g., a part of a CLB or BRAM tile). Within each site, retrieve the *Cell* $\leftrightarrow$ *BEL* mapping. This contains both logic cells instantiated by the input design as well as cells instantiated by the router as "route-throughs": To improve routability, Vivado's router utilizes both (1) LUTs configured with identity functions to route

[58] See `https://web.archive.org/web/20250815131553/http://www.rapidwright.io/`, accessed 2026-01-05

[59] See `https://web.archive.org/web/20250218033156/https://www.jython.org/`, accessed 2026-01-05

one of the LUT's inputs into CLB tiles and (2) latches configured with permanently active clock and clock enable signals to route signals via the latch. Furthermore, the internal routing of CLB tiles in Vivado is handled differently than general routing via interconnect tiles. This information is retrieved from the instantiated FPGA sites as well. For quick *PIP* $\leftrightarrow$ *Net* reference when identifying nets affected by certain faults in PIPs, this information is supplemented by information on all PIPs in interconnect tiles utilized by the design organized by tile name. All information discussed above is exported as a nested JSON dictionary that contains information organized first by FPGA tile, followed by the tile's FPGA sites. This site information, in turn, consists of the logic cells placed in BELs found in this site, as well as intra-site routing multiplexers and information about BELs utilized as route-through cells as described above.

4. Iterate over the design's nets, retrieving information about each PIP and, again, intra-site routing element and route-through element used by these nets. Care is taken to extract all intra-site routing including intra-site multiplexers, as well as driving BEL and driven BEL pins. This data is stored as a second JSON dictionary that is organized by the design's nets' names. It contains (1) the PIPs used by the net, (2) the site pins driving and driven by the net, (3) site-local routing information as a list of related sites, and (4) metadata such as the net's hierarchical name and a flag if the net is a static net (GND or VCC) for each net name, and thus provides the *Net* $\leftrightarrow$ *PIP* mapping discussed above.

P&R Extraction from Yosys & NextPNR

The procedure described above for AMD Vivado is necessary as it does not allow to export the placement and routing information required for further design analysis in a documented machine-readable format. In contrast, [205] presents an open-source FPGA toolchain that consists of Yosys[60] as a synthesis tool and NextPNR[61] as an implementation tool. When implementing a design using NextPNR, the necessary information is included automatically within the generated post-routing netlist.

Yosys defines a JSON netlist format[62] which is augmented by NextPNR with placement and routing information. An example for the structure of a post-routing netlist generated by NextPNR is shown in Figure 8.

The netlist is present as nested JSON dictionaries where each module is itself represented as a dictionary containing sub-dictionaries to hold (1) implementation settings, (2) attributes carried over from RTL synthesis, (3) assignments of ports to internal wire IDs, (4) the cells (i.e., gates) instantiated by the module, and (5) the nets inside the module. In a post-routing netlist, only the toplevel module persists as the design is flattened during implementation.

The placement information can be extracted from the `cells` subdictionary of the toplevel module dictionary. Each entry in this dictionary represents an instantiated cell, which is augmented by NextPNR with physical placement information. In the example in Figure 8, the placement for a LUT and an FF to the slice A of the tile `X32/Y26` in a Lattice ECP5 LFE5U-25 FPGA. Furthermore, each cell's `connections` subdictionary contains a mapping of each (used) cell pin to a net in the `netnames` dictionary holding routing information. Here, the *F* output of the LUT is connected to the *DI* input of the FF, which is indicated by the connection of both pins to a wire with ID 3627.

[60] See `https://web.archive.org/web/20250104053342/https://yosyshq.net/yosys/`, accessed 2026-01-05

[61] See `https://web.archive.org/web/20250211084002/https://github.com/YosysHQ/nextpnr`, accessed 2026-01-05

[62] See `https://web.archive.org/web/20250221164638/https://yosyshq.readthedocs.io/projects/yosys/en/v0.48/cmd/write_json.html`, accessed 2026-01-05

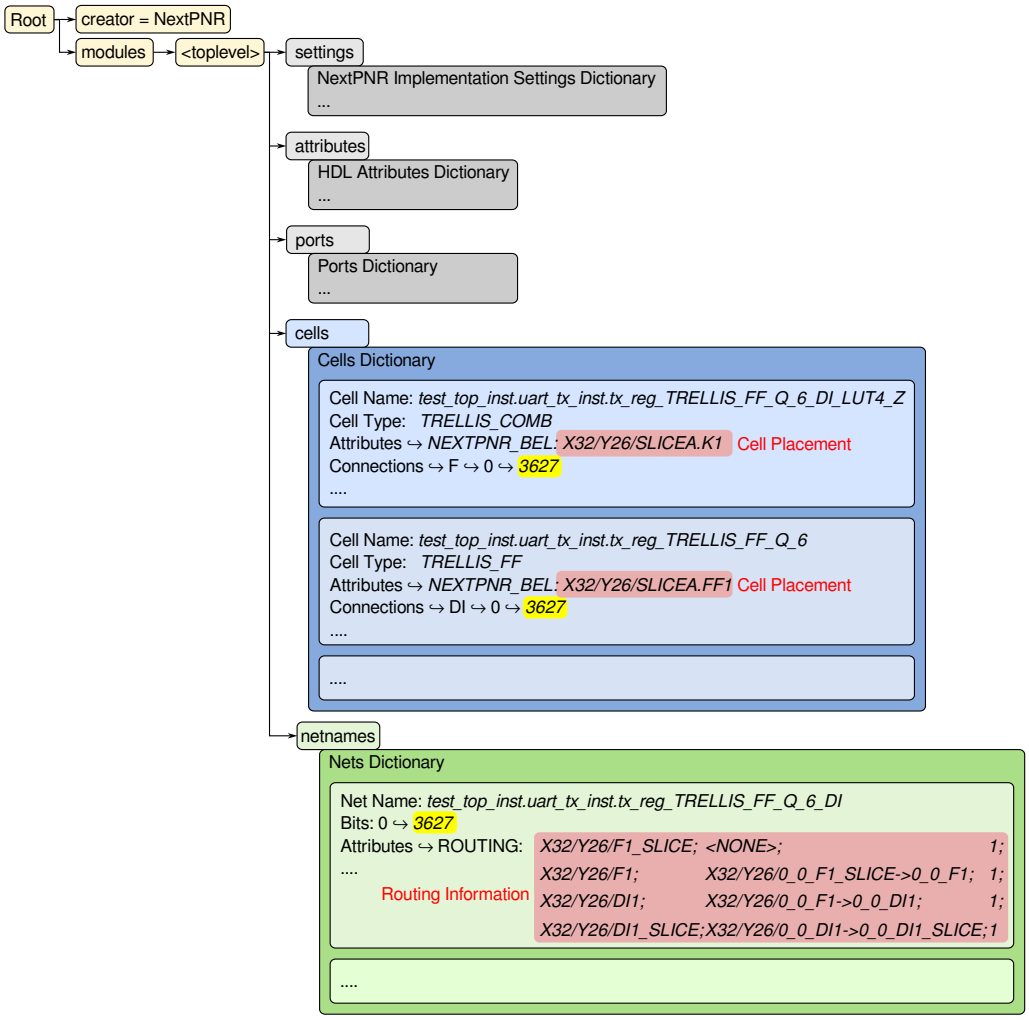


Figure 8: NextPNR JSON Post-Routing Netlist Example

The respective routing information can be found in the netnames subdictionary of the toplevel module dictionary. In a post-routing netlist, this dictionary contains an entry for each wire ID, which is augmented by NextPNR with routing information. This information is, in essence, a collection of 3-tuples (W, P, S) where W stands for a unique physical wire in the FPGA, P stands for the PIP driving this wire segment, and S is a "strength" value for the constraint represented by W and P[63]. The order of tuples does not carry routing information; the order of wire segments needs to be derived by combining this routing information with the device database of the target FPGA. Furthermore, the information that `F1_SLICE` represents the output of the LUT `SLICEA.K1` and that `DI1_SLICE` represents the input of the FF `SLICEA.FF1` is implicit. Supplementing this information, a graph representation can be constructed in a similar way as described for AMD Vivado for each net, which allows to identify the net fanout affected by a bit-flip affecting a particular PIP.

This procedure is possible for all devices supported by NextPNR, including Lattice iCE40 and ECP5 devices used in this dissertation. Further devices fully supported by NextPNR include Lattice Nexus, Gowin LittleBee, and NanoXplore NG-Ultra[64].

[63] This value is always set to "1" by NextPNR after routing is complete

[64] See `https://web.archive.org/web/20250211084002/https://github.com/YosysHQ/nextpnr`, accessed 2026-01-05

4.2 Static Bitstream-Level Fault Analysis

After each relevant configuration bit has been associated with one or more FPGA resources in the target device, the next step is to model the consequences of each assumed fault. In this dissertation, configuration bit flips are modeled as single-bit errors.

While multi-bit errors are expected to happen in the field, in a recent evaluation of the radiation susceptibility of AMD's 7 series FPGAs [206], 89.7% of the observed configuration faults were SEUs, underlining the importance of the single bit flip as a fault model for configuration faults in FPGAs. While considering multi-bit upsets would require a significantly larger number of faults to be modeled, the fault modeling process described in this section is also applicable to multi-bit upsets.

This step of the proposed bit-level fault assessment approach augments the existing mapping of configuration bits to FPGA resources (and their functions) with the expected consequences of a fault, where applicable. Furthermore, it aims to associate each configuration bit with the netlist cells affected by it flipping. This is done by querying the placement and routing information extracted from the design (see Section 4.1.2).

Determining the consequences of a fault requires considering the type of affected resource and its affected function. Functional information about the affected resource is available as part of the bitstream documentation necessary to implement this approach, supplemented by information published in the vendor's data sheets and application notes on the device's architecture. The description of the modeling approaches for different types of resources in this section has been split into modeling faults in LUTs (see Section 4.2.1), in the interconnect (see Section 4.2.2), and other resources (see Section 4.2.3).

4.2.1 Modeling Faults in Look-Up Tables

Bit flips in the initialization content of LUTs potentially change the logical function(s) realized by these elements. This is demonstrated on the basis of Figure 9a which shows an exemplary implementation of a 4-input LUT based on the transistor-level implementation provided in [43][65]. This LUT has been extended to be "fracturable" [207] into two three-input LUTs as shown in Figure 9. A larger number of outputs (i.e., a more fine-grained fracturability) is possible. For example, the OpenFPGA project proposes a 6-input LUT that can be fractured into four 4-input LUTs, two 5-input LUTs or one 6-input LUT[66].

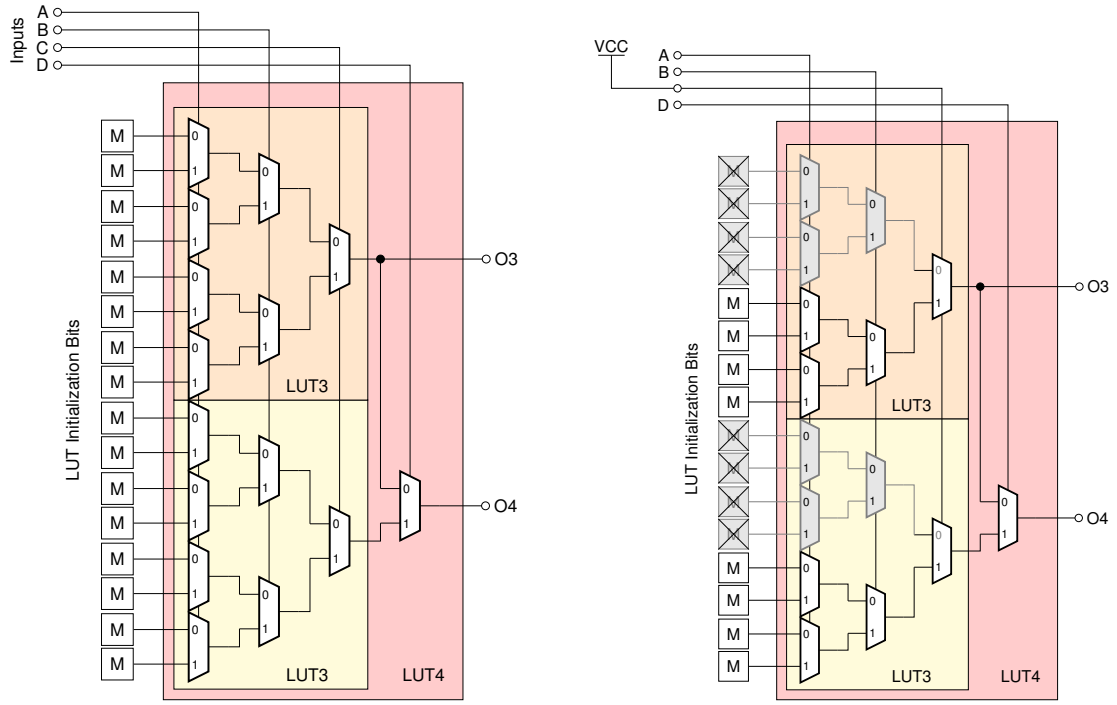
The LUT shown in Figure 9 serves as a model for a widespread type of LUT resource found in a large variety of FPGA devices with certain variations. As an example for variations on this design, AMD's 7 Series FPGAs as well as UltraScale FPGAs provide 6-input LUTs with two outputs ($O5$ and $O6$) which are fracturable into two 5-input LUTs [98, 208]. In contrast, both Lattice's iCE40 and ECP5 families provide 4-input LUTs with one output, which thus cannot be fractured [209, 210].

In this example, the output $O3$ outputs the function of inputs A, B, C, while the output $O4$ outputs the function over all of the LUT's four inputs. Different cases must be observed when assessing the consequences of bit flips in the LUT's initialization content:

- If the LUT is used in a non-fractured way (i.e., D is driven dynamically), a bit flip in the first

[65] In [43], a transistor-level implementation of this LUT is given as a four-level pass-transistor-based multiplexer with integrated inverters to generate an additional inverted version of its inputs A, B, C, D and internal buffers between the second stage (controlled by $B/\overline{B}$) and the third state (controlled by $C/\overline{C}$). These implementation details, however, are not directly relevant to a bitstream-based fault model for configuration errors.

[66] See `https://web.archive.org/web/20250222090512/https://openfpga.readthedocs.io/en/latest/manual/arch_lang/circuit_model_examples/#native-fracturable-lut`, accessed 2026-01-05



(a) General Structure (b) *Don't-Care* Bits due to Static-"1" Input C

Figure 9: Example Implementation of a 4-Input LUT (based on [211, 43]) that can be fractured into two 3-Input LUTs

half of the LUT's initialization content affects both outputs, while a bit flip in the second half of the initialization content only affects $O4$.

- If the LUT is fractured into two three-input LUTs sharing the same set of inputs by tying D to "1", a bit flip in the first half of the initialization only affects $O3$, while a bit flip in the second half only affects $O4$.

- If the LUT is under-utilized – i.e., realizing a function of less than four inputs while tying the unused inputs to a static logic level (often "1") – there are certain subsets of configuration bits that are never accessed under normal operation. This condition is illustrated in Figure 9b. Its existence for each LUT can be derived (1) from the placement and routing information by querying the net each of the LUT's inputs is driven from, or (2) from the configuration itself under the assumption that a LUT input with no configured driver is pulled to a static logic level (e.g., "1"). Under the assumption that a single-bit error is not able to affect the inputs driven to a static logic level, the thusly derived "don't care"-bits may be considered as non-critical bits when assessing potential fault consequences.

Depending on the FPGA technology under consideration, LUT may also be configured as distributed memory or as a shift register (e.g., SRL16/SRL32 in AMD 7 Series and Ultrascale FPGAs). The respective configuration bits bear the potential to drastically affect the function of the LUT and thus are assumed to affect cells connected to all of the LUT's outputs.

If a cell is used as distributed RAM or shift register, a bit flip in the LUT's data content (which may be stored in the respective configuration elements for the LUT's initialization value) can affect the design differently. As these faults do not alter the structure of the design, but its state instead, these faults are out of scope of the modeling considerations described in this section.

The cells affected by a bit flip in the configuration of a LUT used as logic or memory are determined by cross-referencing placement information. On the other hand, cells affected by a single-bit error in the configuration of a LUT used as route-through are determined by querying the intra- and inter-tile routing information for the fanout of each of the considered LUT's outputs.

4.2.2 Modeling Interconnect Faults

Bit flips in the configuration bits affecting interconnect resources have been subject of previous studies [27, 132]. Figure 10 shows five basic effects of interconnect faults caused by configuration memory bit flips in SRAM-based FPGAs as described by [27].

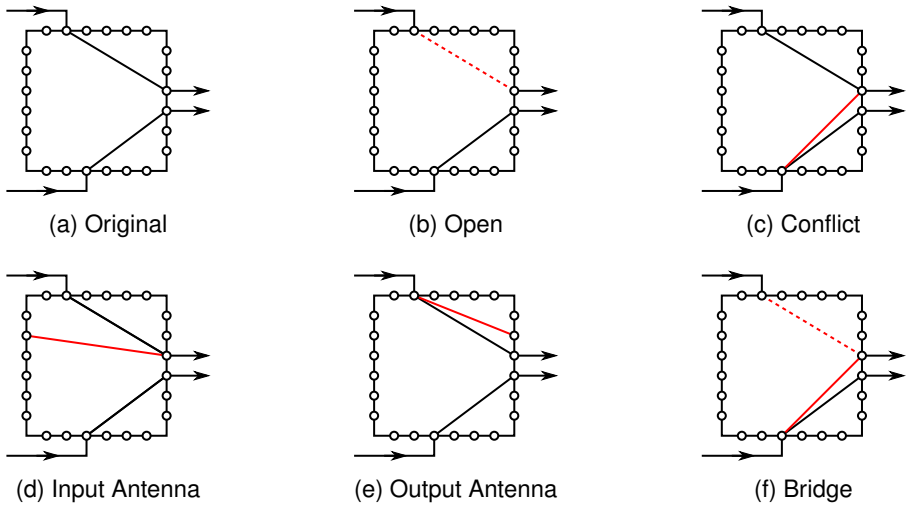


Figure 10: Basic Effects of Interconnect Bit Flips [27]

Figure 10a shows the original configuration of a routing tile in the FPGA. Two signals are routed on two different paths through this tile, configured by the original settings of the corresponding configuration bits. The other inputs and outputs of the routing tile are assumed to be unused. Errors in this configuration may cause the following effects to emerge:

- In an *Open* fault (see Figure 10b), the original driver of one interconnect resource is disconnected. This causes the original route to float or to be (weakly) pulled to an implementation-defined level.
- In a *Conflict* fault, a connection between two unrelated routes is newly formed by the fault (shown in Figure 10c), connecting an additional "Incident" (i.e., fault source) wire.
- In an *Input Antenna* fault, the fault causes another, previously unused (as opposed to *conflict* faults), driver to be connected as an "Incident" (i.e., fault source) wire to the driven part of the route as shown in Figure 10d.
- An *Output Antenna* fault (seen in Figure 10e) causes the driver of the route to be connected to a previously unused wire segment in addition to the originally driven route.
- A *Bridge* fault (see Figure 10f), removes an original connection. In turn, however, the fault connects the driven part of the route to the driver of another route used in the design.

Finally, a *Tolerant* fault does not change a route topologically in any way, while *Unrouted* faults are introduced to match all changes not described in the modes above. Which of these models can

be the effect of a single bit flip depends on the circuit structure of the PIPs and the encoding of their connections in configuration memory [27, 132]. This work focuses on *Open*, *Input Antenna*, *Conflict* and *Bridge* faults, as these faults bear the possibility to disrupt the design logically. *Output Antenna* faults, on the other hand, impact the delay of an interconnect wire, potentially leading to timing violations if they affect nets with minimal slack. These faults are covered in-depth in [47].

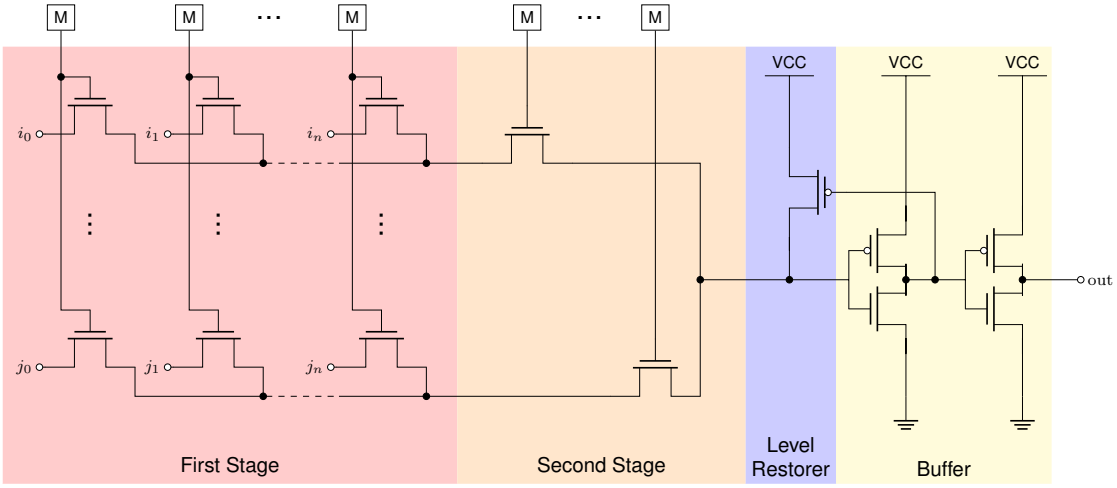
A common way to implement FPGA interconnect elements is to implement multiplexers based on pass transistors or transmission gates. As their select signal is driven by configuration memory cells, these multiplexers are also referred to as "stored-select multiplexers" [212].

To form wide multiplexers (e.g., 16:1, 24:1, or 32:1), as needed in complex FPGA interconnects, a decoded two-stage design can be employed [213]. Examples for these implementations are depicted in Figures 11a and 11b. While multiplexers based on pass transistors require fewer transistors and reduce routing effort (requiring only the true output of each configuration SRAM cell to be routed), they require a level restoring buffer to counteract signal degradation [213]. Two-stage multiplexers are implemented from $n \times m : 1$ decoded single-stage multiplexers. Here, n is the number of inputs of the first-stage multiplexers, and m is the number of inputs in the second-stage multiplexer. As the first-stage multiplexers may share configuration bits, $n + m$ SRAM cells are needed to store the routing configuration. For example, a 24:1 multiplexer can be implemented as a $6 \times 4 : 1$ or a $8 \times 3 : 1$ multiplexer, requiring 10 or 11 SRAM cells, respectively.

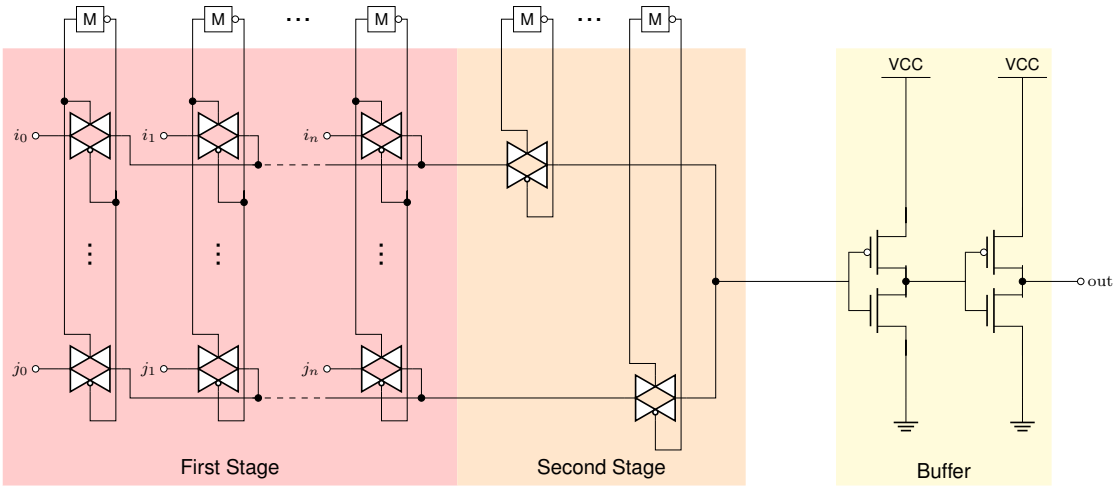
In FPGA technologies requiring narrower multiplexers (e.g., 8:1), binary coded multiplexers can be a more efficient choice, requiring $\lceil log_2(n) \rceil$ configuration bits. This reduced number of configuration bits is traded off against increased logic complexity (e.g., due to a larger number of pass gates and inverters) and potentially worsened delay properties. On transistor level, binary encoded multiplexers may be implemented using pass transistors, transmission gates, or pure CMOS logic. An implementation example is depicted in Figure 11c.

Deducing the multiplexer implementation style (encoded vs. decoded) is a crucial first step in modeling the effects of single bit flips in interconnect-related configuration bits. For example, in interconnect resources implemented as binary encoded multiplexers as shown in Figure 11c, a single configuration bit flip leads to the multiplexer switching over to a different driver for a route instead of the original driver, constituting a *Bridge* fault (see Figure 10f). In contrast, in a decoded multiplexer (see Figures 11a and 11b), each configuration bit switches off or on at least one transistor when it changes state. An *Open* fault (see Figure 10b) is the consequence of a configuration bit originally set to "1" flipping to "0", thereby turning its corresponding transistor(s) off. When a bit that is originally set to "0" flips to "1", one (in the second stage) or multiple (in the first stage) short-circuits arise. This causes either an *Input Antenna* (see Figure 10d) or *Conflict* fault (see Figure 10c), depending on the additionally configured source wire being used by the design or not. An architecture similar to Figure 11b was patented by AMD (then Xilinx) in 2016 [214]. The patented implementation adds buffers at the outputs of the first-stage multiplexers, and a NAND pulldown structure at the output of the second-stage multiplexer to provide a defined logic level when all configuration bits are set to "0".

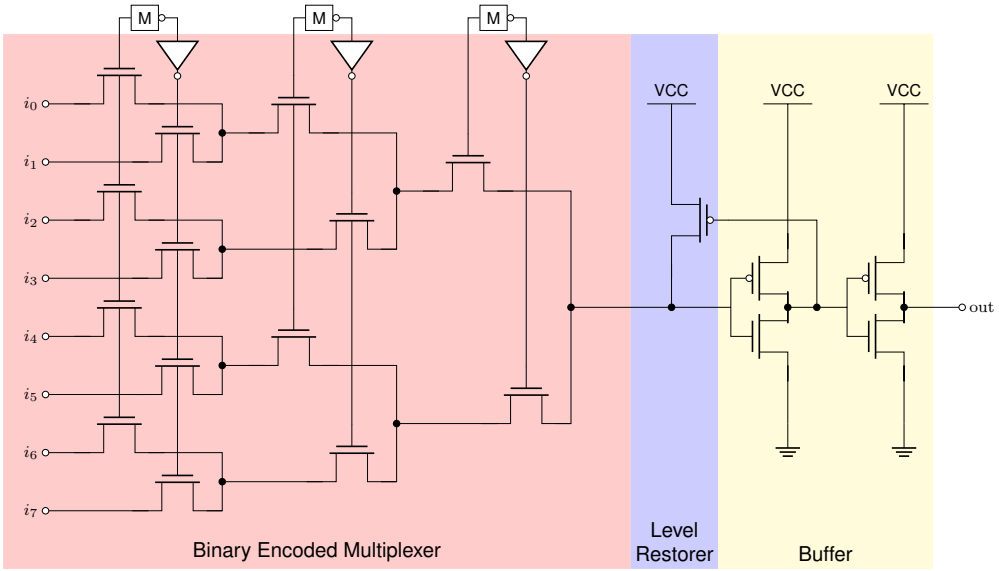
Having access to bitstream documentation for the target device, the employed multiplexer implementation style can be inferred from the configuration patterns for the considered interconnect resource's valid configuration patterns. All configuration patterns having a similar population count (i.e., numbers of configuration bits set to "1") suggests a decoded multiplexer, with the maximum number of "1"-bits indicating the number of employed stages. This is less straightforward to deduce if some of the multiplexer's configuration bits are inverted or if the multiplexers provides "fast" inputs that are directly connected to the second stage as done in Intel/Altera's Stratix-II technology [215]. On the other hand, a population count that widely varies among valid configuration patterns indicates the use of an encoded multiplexer.



(a) Decoded Two-Stage Pass-Transistor-Based Multiplexer



(b) Decoded Two-Stage Transmission-Gate-Based Multiplexer



(c) Binary Encoded Pass-Transistor-Based Multiplexer

Figure 11: Examples of Interconnect Multiplexer Implementations [213]

In addition to bitstream documentation, the systematic evaluation of the effects of particular types of faults in a target device further requires knowledge about the device's interconnect structure. An important design decision for implementing an FPGA's interconnect is structuring the interconnect as either (1) a tri-state interconnect where multiple drivers potentially connect to a single (thus bidirectional) interconnect wire, or (2) a single-driver unidirectional architecture. [216] argues the latter option provides significant area, power, and delay advantages. However, more physical interconnect wires may be necessary to provide connectivity to all FPGA tiles. Modern devices often employ point-to-point architectures for most intra- and inter-tile routing. They supplement this with bidirectional wires for special purposes, e.g., crossing long distances within the FPGA [43].

Devices that use internal tri-state signals inherently provide the possibility of a fault causing bus contention by a bit flip that enables a second driver. An example for an interconnect architecture based on tri-state signals is shown in Figure 12, where a small square indicates each configurable element.

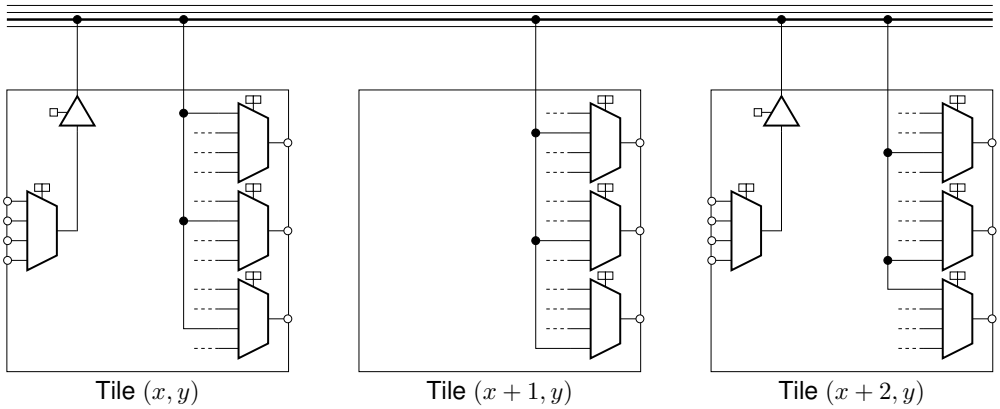


Figure 12: Exemplary Bidirectional Interconnect Architecture based on Tri-State Signals

If the tri-state wire shown in this figure is used to route a signal from tile (x, y) to tile $(x + 1, y)$, a single-bit error can:

- Disable the active buffer in tile (x, y), causing an *Open* fault,
- Enable the inactive buffer in tile $(x + 2, y)$, causing a *Conflict* fault between tiles (x, y) and $(x + 2, y)$, or
- Depending on the employed internal multiplexer architectures (see Figure 11), cause intra-MUX *Open*, *Bridge*, *Conflict*, *Input Antenna* or *Output Antenna* faults. These faults must be differentiated by taking the multiplexers' connectivity, original configuration, and post-routing design information into account:
  - In a decoded multiplexer (see Figures 11a and 11b),
    - Turning off pass transistors or transmission gates configured by the fault-free bitstream may cause an *Open* fault.
    - Turning on an additional pass transistor or transmission gate, so that a previously unused source wire is added as another input, causes an *Input Antenna* fault.
    - Turning on an additional pass transistor or transmission gate, so that an additional signal within the design is added as another input, causes a *Conflict* fault.
  - In an encoded multiplexer (see Figure 11c),

    - Altering the multiplexer's selected input so that the output is driven from an unused source wire may cause an *Open* fault.
    - Altering the multiplexer's selected input so that the output is driven from a different signal within the design constitutes a *Bridge* fault.
  - If the "default" configuration of a multiplexer selects a certain signal (e.g., VCC), any alteration in the multiplexer's configuration may cause *Open* or *Bridge* faults.

In unidirectional interconnects, wires can only be driven by one interconnect resource. An example is shown in Figure 13. It is to be noted that the buffer driving the interconnect wire from the output of the multiplexer in tile (x, y) is always enabled in contrast to the configurable tri-state drivers shown in Figure 12. In these interconnects, contention is limited to the internals of the affected decoded multiplexers. Similar faults as discussed above for bidirectional interconnect architectures are possible within the multiplexers.

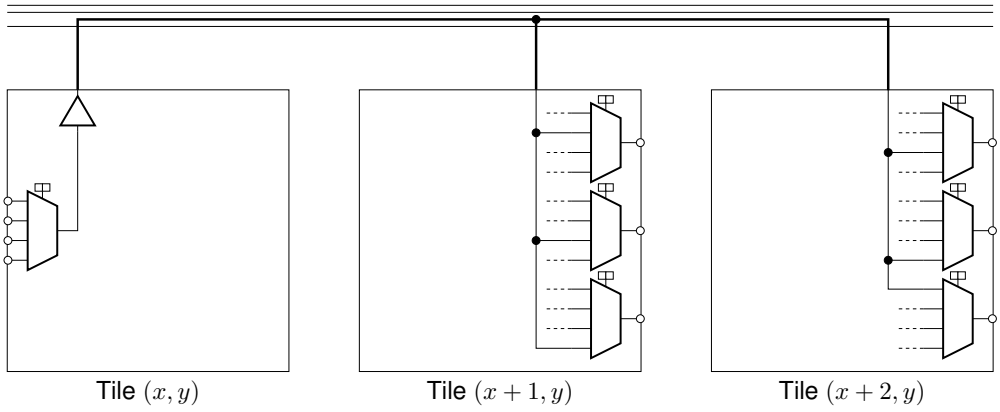


Figure 13: Exemplary Unidirectional Interconnect Architecture

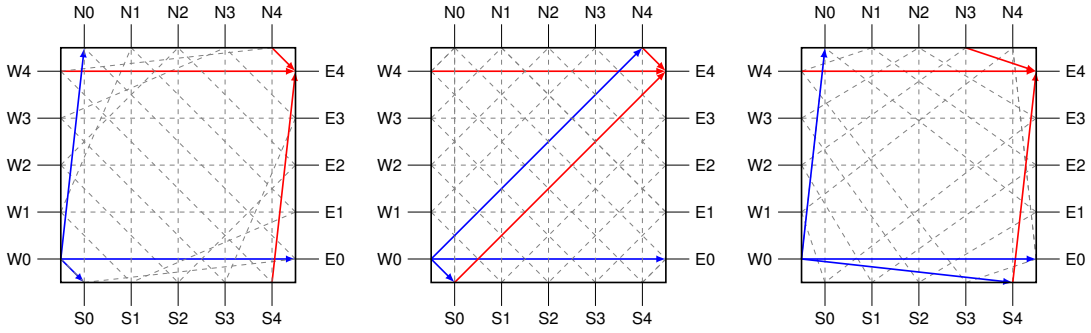
In both types of devices, information about the connectivity of each interconnect resource's source wires is necessary to identify, e.g., conflict faults correctly. This includes all interconnect resources that can be driven from a single source wire. This is influenced both by the FPGA's routing segmentation (i.e., the number and length of the wire segments employed in the interconnect) as well as the layout of the interconnect switch boxes [217].

Common switch box layouts are modifications of the basic interconnection patterns shown in Figure 14. In this illustration, $W0$ is an input wire of the switch box, while $E4$ is an output wire. The illustration shows the options for routing a signal entering the switch box at $W0$ and the routing options for a signal exiting it at $E4$. The nodes driving the switch boxes' outputs (e.g., $E4$) may be implemented using one of the multiplexer options shown in Figure 11.

Modified Wilton switch boxes are a popular choice as this layout eases routing [43]. The academic open-source FPGA CAD flow *Verilog-to-Routing (VTR)* uses modified Wilton switch boxes per default[67].

Thus, after modeling the consequences of a fault localized to a particular interconnect resource (i.e., within a single multiplexer), the device's switch box layout and interconnect architecture must be considered when deducing the upstream and downstream connectivity of the fault. In the ideal case, a device database not only contains the bitstream-to-resource and bitstream-to-function mapping but also connectivity information. Ideally, the information described above is present as a data structure that can be queried for all wire segments, potential drivers, and potential

[67] See `https://web.archive.org/web/20260105164234/https://docs.verilogtorouting.org/en/v9.0.0/arch/reference/#fpga-device-information`, accessed 2026-01-05



(a) Disjoint (b) Universal (c) Wilton

Figure 14: Basic FPGA Switchbox Interconnection Patterns [217]

fanouts connected to a particular interconnect resource's inputs and outputs. In conjunction with the placement and routing information, this information is necessary to refine the assessment of each fault's potential consequences.

The remainder of this section shows how interconnect faults can be modeled based on the above information and procedure. This is done by illustrating the interconnect fault modeling procedure on three FPGA families: AMD 7 Series, Lattice iCE40, and Lattice ECP5.

AMD 7 Series

AMD's 7 Series FPGAs are organized in columns of "tiles" containing IO resources, logic resources (i.e. CLBs containing LUTs, Flip-Flops, Carry Blocks etc.), BRAM, DSP blocks, and columns of interconnect blocks in between. Each of these columns is composed of several rows of tiles of the same kind. Particular (parts of) columns are replaced by special hard macros such as monitoring circuitry, the Internal Configuration Access Port (ICAP), and PCIe or CPU hard IP blocks.

In AMD's 7 Series technology, all interconnect tiles are paired with a specific logic tile (a CLB, block memory, etc.) next to it. If this block is left or right of the interconnect tile, the tile is referred to as `INT_L` or `INT_R`. In `INT_L` tiles, certain wire names are extended by the suffix "`_L`" to indicate their location. These interconnect tiles contain PIP that can be configured to connect subsets of the tile's input wires (see Table 30) to its output wires (see Table 31). The shown wire names have been exported using AMD's *RapidWright* API for an `INT_L` tile in an `xc7a35tcpg236-1` device, but are the same in (1) other interconnect tiles and (2) other 7 Series devices. The wires shown in Tables 30 and 31 have been categorized according to the *Cost Code* reported by *RapidWright* which serves as a proxy for the wire's function and connectivity. The meaning of these cost codes is shown in Table 32. The cost codes `BOUNCEACROSS`, `BOUNCEIN`, and `PINBOUNCE` signify wires that are capable of "bouncing" the signal back into the interconnect, i.e., to serve as a destination and source wire for connections simultaneously. The cost code `HVCCGNDOUT` signifies a connection to a static driver. However, documentation information by Project XRAY indicates that only connections to GND correspond to physical source wires. In contrast, connections to VCC are made by using default configuration patterns in certain multiplexers. This is referred to as "pseudo-PIP" in the remainder of this section. General interconnect wires that route connections between interconnect tiles can be found in the `BENTQUAD`, `DOUBLE`, `HQUAD`, `SINGLE`, and `VQUAD` cost codes. The cost code differentiates general interconnect wires by length, and, for `*QUAD` wires, by direction. Finally, it can be seen in Table 32 that while most wires in AMD's 7 Series devices are unidirectional and thus can only be driven by one routing resource, wires belonging to the `HLONG`, `VLONG`, and `VLONG12` cost codes are bidirectional.

Table 30: Input Wires of AMD 7 Series Interconnect Tiles

Cost Code	Number	Name Pattern(s)
BENTQUAD	18	`[NE|NW|SE|SW]6END[0-3]`, `NW6END_S0_0`, `SW6END_N0_3`
BOUNCEACROSS	15	`BYP_BOUNCE[2,3,6]`, `BYP_BOUNCE_N3_[2,3,6,7]`, `FAN_BOUNCE[2,4,6,7]`, `FAN_BOUNCE_S3_[0,2,4,6]`
BOUNCEIN	9	`BYP_BOUNCE[0,1,4,5]`, `FAN_BOUNCE[1,3,5]`, `GFAN[0,1]`
DOUBLE	38	`[EE|NE|NN|NW|SE|SS|SW|WW]2END[0-3]`, `[NE|NN|NW|SS|SW|WW]2END_S3_0`
GLOBAL	18	`GCLK(_L)_B[0-11]`, `GCLK(_L)_B[6-11]`
HLONG	3	`LH[0,6,12]`
HQUAD	9	`[EE|WW]4END[0-3]`, `WW4END_S0_0`
HVCCGNDOUT	2	`GND_WIRE`, `VCC_WIRE`
OUTBOUND	24	`LOGIC_OUTS(_L)[0-23]`
PINBOUNCE	16	`BYP_ALT[0-7]`, `FAN_ALT[0-7]`
SINGLE	38	`[EL|ER|NR|SL|WL|WR]1END[0-3]`, `NL1END[0,1,2,_S3_0]`, `SR1END[1,2,3,_N3_3]`, `EL1END_S3_0`, `ER1END_N3_3`, `NL1BEG_N3`, `SR1BEG_S0`, `WL1END_N1_3`, `WR1END_S1_0`
VLONG	3	`LV(_L)[0,9,18]`
VLONG12	2	`LVB(_L)[0,12]`
VQUAD	10	`[NN|SS]6END[0-3]`, `NN6END_S1_0`, `SS6END_N0_3`

The interface to the neighboring logic tile consists of the wires shown for a CLB tile as an example in Table 33. In `INT(_L)` tiles neighboring CLB tiles, `LOGIC_OUTS(_L)[0-7]` are driven by eight flip-flops, outputs `LOGIC_OUTS(_L)[8-15]` are driven by the eight LUTs in the neighboring tile, and outputs `LOGIC_OUTS(_L)[15-23]` are driven by output multiplexers that either propagate the output of a (fractured) LUT, carry logic, or a secondary set of flip-flops. Depending on the neighboring tile type, the logic output wires are driven by different outputs of logic elements in the adjacent tile. In an `INT(_L)` tile neighboring a BRAM tile, depending on the vertical offset between the tiles, the logic outputs are driven by BRAM or FIFO buffer status outputs or data output lines. This applies similarly to DSP tiles, where `LOGIC_OUTS_(L)[...]` wires are connected to the DSP's outputs, and the DSP's inputs are connected to `IMUX_(L)[...]` wires. Each BRAM and DSP tile connects to five adjacent interconnect tiles. Other special-function logic, such as the PCIe hard IP found in AMD 7 Series devices, as well as other logic blocks, are connected similarly. Depending on the adjacent logic tile type and the interconnect tile's offset relative to it, a selection of the wires shown in Table 33 may also be left unconnected. Table 34 shows the number of interconnect tiles connected to each logic tile type in an AMD Artix 7 xc7a35t FPGA as reported by AMD Vivado.

The interconnect tiles contain resources to drive each output endpoint from several input endpoints. Conversely, the input endpoints can themselves drive several different output endpoints. However, as interconnect tiles in AMD 7 Series devices are designed as Wilton switch blocks [47], not every combination of input and output endpoints is possible. This information is published by the third-party bitstream documentation initiative Project XRAY as part of the supplied device databases as a JSON file containing inter-tile connections[68].

[68] See `https://web.archive.org/web/20250115210456/https://f4pga.github.io/prjxray-db/artix7/xc7a50t/tileconn.json`, accessed 2026-01-05

Table 31: Output Wires of AMD 7 Series Interconnect Tiles

Cost Code	Number	Name Pattern(s)
BENTQUAD	16	`[NE|NW|SE|SW]6BEG[0-3]`
BOUNCEACROSS	9	`BYP_BOUNCE[2,3,6,7]`, `FAN_BOUNCE[0,2,4,6,7]`
BOUNCEIN	9	`BYP_BOUNCE[0,1,4,5]`, `FAN_BOUNCE[1,3,5]`, `GFAN[0,1]`
DOUBLE	32	`[EE|NE|NN|NW|SE|SS|SW|WW]2BEG[0-3]`
GLOBAL	12	`GCLK(_L)_B[6-11]_[EAST|WEST]`
HLONG	2	`LH[0,12]`
HQUAD	8	`[EE|WW]4BEG[0-3]`
PINBOUNCE	16	`BYP_ALT[0-7]`, `FAN_ALT[0-7]`
PINFEED	68	`BYP(_L)[0-7]`, `CLK(_L)[0,1]`, `CTRL(_L)[0,1]`, `FAN(_L)[0-7]`, `IMUX(_L)[0-47]`
SINGLE	32	`[EL|NL|WL]1BEG[0-2,_N3]`, `[ER|SR|WR]1BEG[1-3,_S0]`, `[NR|SL]1BEG[0-3]`
VLONG	2	`LV(_L)[0,18]`
VLONG12	2	`LVB(_L)[0,12]`
VQUAD	8	`[NN|SS]6BEG[0-3]`

Based on these documentation efforts, an entirely open-source RTL-to-bitstream toolchain for (among others) AMD 7 Series FPGAs has been implemented[69]. Furthermore, the bitstream documentation has been used in several academic works (e.g., [121, 124]) to gain insight into the inner workings of these devices.

Tables 35 and 36 show examples of configurations for all the PIPs that drive one output wire in an `INT_L` interconnect resource, provided by Project XRAY's bitstream documentation for `INT_L` tiles[70]. These tables show the configuration bits $b_0, \ldots, b_8$ and $b_0, \ldots, b_9$, respectively, that need to be set to "1" to connect a particular input wire to the resource's output wire. To improve clarity, bits that need to be set to "0" are indicated by a dot instead of a zero.

To drive the multiplexer's output wire, for example, from I_4 in Table 35, the configuration bits $b_0, \ldots, b_8$ must be set to the pattern that corresponds to that connection. Thus, in this case, bits b_0 and b_6 must be set to "1", while the others remain at "0".

In the examples in Tables 35 and 36, it is indicated that an input wire may come from the interconnect (I_n) or from the neighboring function tile (L_n). Considering the configuration layout, it can thus be seen that there is no difference between PIPs that are driven by wires from the interconnect, and PIPs that are driven by the outputs of function tiles.

In the interconnect tiles' overall configuration, groups of PIPs are controlled by 7, 8, 9, or 10 configuration bits. Configuration bits are only shared among the PIPs that drive one particular output endpoint, not among PIPs that drive different endpoints. This indicates that these groups of PIPs are the connections possible using individual interconnect multiplexers.

Considering the configuration pattern of the possible drivers laid out in Table 35, it can be seen that this structure fits well with the assumption that this interconnect point is implemented as a decoded multiplexer (see Figures 11a and 11b). Each configuration bit in the first stage of multiplexers (either $b_0, \ldots, b_4$ or $b_5, \ldots, b_8$) controls either 4 or 5 pass transistors or transmission gates, respectively.

[69] See `https://web.archive.org/web/20250121184318/https://f4pga-examples.readthedocs.io/en/latest/`, accessed 2026-01-05

[70] See `https://web.archive.org/web/20250113214512/https://f4pga.github.io/prjxray-db/artix7/segbits_int_l.db`, accessed 2026-01-05

Table 32: *Cost Code* in AMD 7 Series Interconnect Wires

Unidirectional			
Cost Code	In	Out	Functionality
BENTQUAD	✓	✓	Diagonally connect `INT` tiles across two rows and four columns (i.e., "six tiles")
BOUNCEACROSS	✓	✓	"Bounce" within tile and connect to previous/next row
BOUNCEIN	✓	✓	"Bounce" within tile to adjacent logic tile's `IMUX` and `CTRL_(L)[0,1]` inputs
DOUBLE	✓	✓	Connect `INT` tiles across two columns or rows
GLOBAL	✓	✓	Access to global routing network (e.g., clock routing)
HQUAD	✓	✓	Connect `INT` tiles across four columns
HVCCGNDOUT	✓	–	Connect to static "0" and "1" drivers.
OUTBOUND	✓	–	Driven from outputs of adjacent logic tile
PINBOUNCE	✓	✓	"Bounce" within tile to adjacent logic tile's `FAN(_L)[...]` and `BYP(_L)[...]` inputs
PINFEED	–	✓	Drive inputs of adjacent logic tile
SINGLE	✓	✓	Connect two adjacent `INT` tiles
VQUAD	✓	✓	Connect `INT` tiles across four rows
Bidirectional			
Cost Code	In	Out	Functionality
HLONG	✓	✓	Connect `INT` tiles across 9 and 18 columns bidirectionally
VLONG	✓	✓	Connect `INT` tiles across 9 and 18 rows bidirectionally
VLONG12	✓	✓	Connect `INT` tiles across 12 rows bidirectionally

The remainder of the bits control a second stage that then selects a wire to drive the multiplexer's output from the outputs of the first stage. The assumption that the interconnect resources of AMD 7 Series FPGAs are constructed this way is also used in the state-of-the-art [124, 131, 218].

A slightly different pattern is used in several endpoints according to Project XRAY's bitstream documentation. This pattern is shown in Table 36. Here, bits $b_6, \ldots, b_9$ are assumed to control the second state of a two-stage multiplexer. The first stage appears to be decoded similarly to both stages in Table 35. In contrast, a different pattern is applied in the second stage, where more than one bit has to be set to complete a connection as shown on the left side ("Original Interpretation") of Table 36. This is interpreted by [124] as "hidden logic" with unknown effects on the interconnect multiplexer if flipped. The underlying circuit-level implementation, however, need not be completely different from the implementation for the configuration pattern shown in Table 35: When assuming that an inverted value is stored in bits b_8 and b_9 as shown on the right side of Table 36 ("Modified Interpretation"), a configuration pattern consistent with a pass-transistor or transmission-gate multiplexer emerges. The only difference to a conventional decoded pass-transistor or transmission-gate-based multiplexer on the circuit level is that the inverted output of the memory cells corresponding to b_8 and b_9 are used.

Project XRAY's bitstream documentation[71] shows that these configuration patterns are present in all multiplexers driving wires of the types `BYP_ALT[...]`, `FAN_ALT[...]`, and `IMUX_[...]`. These wires can, on the one hand, be driven by interconnect wires present in the tile by setting a valid

[71] See `https://web.archive.org/web/20250115214920/https://f4pga.github.io/prjxray-db/artix7/tile_int_l.html` and `https://web.archive.org/web/20250115214920/https://f4pga.github.io/prjxray-db/artix7/tile_int_r.html`, accessed 2026-01-05

Table 33: Connections between Interconnect Tiles and Adjacent CLB Tiles in AMD 7 Series

Num.	Wire Name Pattern	Direction	Function
24	`LOGIC_OUTS_(L)[`i`]`	Logic →Int.	Connect CLB tile's outputs to interconnect network
48	`IMUX_(L)[`i`]`	Int. → Logic	Connect interconnect network to the CLB tile's inputs
8	`BYP_(L)[`i`]`	Int. → Logic	Bypass LUT in neighboring CLB tile
8	`FAN_(L)[`i`]`	Int. → Logic	Clock Enable for FFs, LUTRAM/SRL data inputs
2	`CTRL_(L)[`i`]`	Int. → Logic	Set/Reset for FFs
2	`CLK_(L)[`i`]`	Int. → Logic	Clock input for adjacent logic tile

Table 34: Interconnect Tiles connecting to Logic Tile Types in xc7a35t FPGA

Tile Type(s)	Connected INT Tiles	
	Absolute	Relative (%)
CLB	4075	72.1
BRAM	375	6.6
DSP	300	5.3
IO	250	4.4
Clock Management	250	4.4
High-Speed Transceiver (`GTP`)	50	0.9
PCI Express	50	0.9
Configuration (e.g., ICAP)	50	0.9
Clock Buffer (`CLK_HROW`, `CLK_BUFG`)	32	0.6
Monitor (XADC)	25	0.4
Unconnected (`VFRAME`, `CLK_{FEED,REBUF_BUFG,PMV,MTBF2}`)	193	3.4
Total	5650	

configuration pattern. On the other hand, Project XRAY states that the multiplexer's output is driven to VCC per "default", i.e., when no configuration pattern is set[72]. At the technology level, this behavior can be explained under the following two assumptions:

1. The multiplexer enters a "default" state that pulls the output to VCC when all configuration bits $b_0, \ldots, b_9$ are set to zero.
2. The second-stage transistors connected to the configuration cells storing b_8 and b_9 are turned on when these bits are set to zero.

In this way, driving the multiplexer's output wire to VCC is accomplished by leaving the first stages' outputs undriven (as $b_0, \ldots, b_5$ are all-zero) while driving the multiplexer's output from the outputs of two undriven first stages (via $\overline{b_8}$ and $\overline{b_9}$) in parallel. This implies that the first multiplexer stages use a form of weak pull-up, pull-down or level-restoring buffer to keep their undriven output at a stable level. This enables the second stage to reliably drive the output wire to VCC.

Based on these assumptions, the following consequences are possible for a single interconnect bit flip in AMD 7 Series FPGAs:

- If any bit is flipped from 1 to 0 in resources similar to Table 35, it is assumed to cause the respective pass transistor to disconnect, thus causing an *Open* fault (cf. Figure 10b).

[72] See `https://web.archive.org/web/20250914163820/https://f4pga.readthedocs.io/projects/prjxray/en/latest/architecture/interconnect.html#vcc-drivers`, accessed 2026-01-05

Table 35: Example of a decoded interconnect multiplexer's configuration in AMD's 7 Series FPGAs

Source Wire	Configuration Bits								
	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
I_0	1	.	.	.	.	1	.	.	.
I_1	.	1	.	.	.	1	.	.	.
I_2	.	.	1	.	.	1	.	.	.
I_3	.	.	.	1	.	1	.	.	.
L_0	.	.	.	.	1	1	.	.	.
I_4	1	.	.	.	.	.	1	.	.
I_5	.	1	.	.	.	.	1	.	.
I_6	.	.	1	.	.	.	1	.	.
I_7	.	.	.	1	.	.	1	.	.
I_8	.	.	.	.	1	.	1	.	.
I_9	1	.	.	.	.	.	.	1	.
I_{10}	.	1	.	.	.	.	.	1	.
I_{11}	.	.	1	.	.	.	.	1	.
I_{12}	.	.	.	1	.	.	.	1	.
I_{13}	.	.	.	.	1	.	.	1	.
L_1	1	.	.	.	.	.	.	.	1
L_2	.	1	.	.	.	.	.	.	1
L_3	.	.	1	.	.	.	.	.	1
L_4	.	.	.	1	.	.	.	.	1
L_5	.	.	.	.	1	.	.	.	1

- If in these resources any bit is flipped from 0 to 1, this causes an additional driver to be enabled inside the multiplexer. For example, if I_4 is configured as a driver in Table 35, originally only configuration bits b_0 and b_6 are set. If a bit flip causes b_1 to be set in addition, this causes at least a conflict between I_4 and I_5 in this multiplexer. If, however, a bit flip causes b_5 to be set in addition, this causes at least a conflict between I_0 and I_4 in the second stage of the multiplexer. Depending on if I_5 or I_0 are used by the design, this constitutes either an *Input Antenna*-like fault (cf. Figure 10d) or *Conflict* (cf. Figure 10c) fault.

  Additional contentions internal to the multiplexer are also possible, depending on the structure of the resource.

- If a bit is flipped in resources encoded like Table 36, the consequences depend on the location of the bit. If this happens in the first stage of the interconnect multiplexer, this is assumed to cause the same consequences as in the fully decoded resource as described above (i.e., *Input Antenna* or *Conflict*). However, if it is flipped in the second stage, a single-bit flip can never cause another valid pattern to emerge. Here, the inversion of b_8 and b_9 must be considered to deduce the correct effect of a bit flip, which may be an *Open*, *Input Antenna*, or *Conflict* fault.

- Another possibility exists for resources encoded like Table 36: If the resource is used as VCC driver, all bits $b_0, \ldots, b_9$ are set to "0" by the bitstream. Setting b_8 or b_9 to "1" may degrade the performance of the VCC driver. However, since the fanout of these wires is limited, this fault may often manifest benignly. Additionally, if the resource is used as a VCC driver, flipping $b_0, \ldots, b_7$ may cause a (used or unused) wire segment to be connected, potentially disrupting

Table 36: Example of a partially decoded interconnect multiplexer's configuration in AMD's 7 Series FPGAs

Source Wire	Configuration Bits														
	Original Interpretation										Modified Interpretation				
	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	$b_0, \ldots, b_5$	b_6	b_7	$\overline{b_8}$	$\overline{b_9}$
I_0	1	.	.	.	.	.	.	.	1	.	Unchanged from Original Interpretation	.	.	.	1
I_1	.	1	.	.	.	.	.	.	1	.		.	.	.	1
I_2	.	.	1	.	.	.	.	.	1	.		.	.	.	1
I_3	.	.	.	1	.	.	.	.	1	.		.	.	.	1
I_4	.	.	.	.	1	.	.	.	1	.		.	.	.	1
I_5	.	.	.	.	.	1	.	.	1	.		.	.	.	1
I_6	1	.	.	.	.	.	.	.	.	1		.	.	1	.
I_7	.	1	.	.	.	.	.	.	.	1		.	.	1	.
I_8	.	.	1	.	.	.	.	.	.	1		.	.	1	.
I_9	.	.	.	1	.	.	.	.	.	1		.	.	1	.
L_0	.	.	.	.	1	.	.	.	.	1		.	.	1	.
I_{10}	.	.	.	.	.	1	.	.	.	1		.	.	1	.
I_{11}	1	.	.	.	.	.	1	.	1	1		1	.	.	.
I_{12}	.	1	.	.	.	.	1	.	1	1		1	.	.	.
I_{13}	.	.	1	.	.	.	1	.	1	1		1	.	.	.
I_{14}	.	.	.	1	.	.	1	.	1	1		1	.	.	.
L_1	.	.	.	.	1	.	1	.	1	1		1	.	.	.
I_{15}	.	.	.	.	.	1	1	.	1	1		1	.	.	.
I_{16}	1	.	.	.	.	.	.	1	1	1		.	1	.	.
I_{17}	.	1	.	.	.	.	.	1	1	1		.	1	.	.
I_{18}	.	.	1	.	.	.	.	1	1	1		.	1	.	.
I_{19}	.	.	.	1	.	.	.	1	1	1		.	1	.	.
L_2	.	.	.	.	1	.	.	1	1	1		.	1	.	.
I_{20}	.	.	.	.	.	1	.	1	1	1		.	1	.	.

the VCC-driving behavior of the two unused first stages connected to the output in parallel by $\overline{b_8}$ and $\overline{b_9}$.

As seen in the example configurations in Tables 35 and 36, all valid configurations consist of at least two configuration bits set to "1". Assuming that in a default configuration – when the destination wire is not driven by the considered group of PIPs – the configuration is set to all-zeroes, a single-bit flip is not enough to configure a complete route in such a group. Thus, only *Output Antenna* faults internal to the multiplexers are possible.

The severity at which multiplexer-internal *Conflict* faults as described above impact a circuit depends on additional factors:

- As previously discussed, AMD's 7 Series FPGAs provide a variety of different wire types, defined by their source, as well as their wire length, and their directionality. These different wire types can be associated with different driver types or strengths, which may influence the outcome of a conflict between two wires of a different kind.
- As described in existing work [46], wires affected by *Open* faults, when considered statically, exhibit stuck-at-1 behavior. However, as noted in [1], when monitoring the disconnected part

of a wire that has been subject to an *Open* fault, it takes some temperature-dependent amount of time (up to multiple milliseconds) for the wire to return to "'1" if it was driven "0" in the moment of disconnection (see Section 4.4.2 for an in-depth discussion of this behavior). This hints at a level-restoring buffer (or "half-latch") circuitry responsible for keeping a consistent "1"-level among unused wire segments. Thus, it is deemed beneficial in this dissertation to distinguish between conflicts of a design's signal with an unused wire (i.e. *Input Antenna*-like faults) and conflicts between two wires used by signals of the design (i.e. *Conflict* faults) when analyzing the sensitivity of a given bitstream to interconnect bit flips. The latter is possible in AMD's 7 Series FPGAs because most input wires that feed into an interconnect tile can drive multiple output wires via multiple interconnect resources. If a signal is routed via another, used, PIP but is also present as an input in the PIP group subject to a bit flip, a conflict between the two signals is possible.

It is assumed that in both cases, multiple drivers are in conflict. In the case of the former *Input Antenna* fault, one is a level-restoring buffer, while in the latter *Conflict* fault both are "normal" drivers. Differences may also be caused by the *location* where the fault happens (the first or second stage of an interconnect multiplexer).

The expected effects in the *Input Antenna* case are that the static driver of the unused wire is either strong enough to impact one or both logical values ("0", "1") transported by the net, or that it is too weak to affect the net. When an accurate circuit-level representation of the multiplexer is not available, the definitive in-hardware effects must be determined experimentally (see Section 4.3).

In the *Conflict* case, there may be an effect on one, both, or neither of the involved nets, depending on the relative driver strength and directionality of the connection newly formed by the fault. In [131], it has been observed in this context (for a limited number of interconnect resources) that when two general interconnect wires are involved, only one of them is impacted by the fault. If one general and one global interconnect wire are involved, both are impacted. Furthermore, the conflict may affect one logical value ("0" or "1") or both logical values of either net as in the case of *Input Antenna Faults*. The in-hardware effects must be determined experimentally in this case as well.

Finally, there are other less common fault modes that can be triggered by a single bit flip in AMD's 7 Series FPGAs, that are not the main focus of this work:

- Potential timing effects of *Output Antenna Faults* are discussed extensively in [47]. However, (1) a single bit flip cannot directly connect a long wire segment, thus the effect in 7 Series FPGAs is expected to be small and (2) the effects measured in [47] are in the sub-nanosecond range – *Output Antenna Faults* may thus be mitigated by slightly increasing the slack requirements on the critical path.

- *Conflict* faults do not only affect the directly routed connection but may also cause consequences on other input wires, depending on if a bit controlling the first or second stage of the responsible interconnect multiplexer flips. If a bit in the second stage flips, only the output wire should be affected. However, a bit flip in the first stage may short-circuit multiple pairs of signals. Depending on the buffering of these signals, an "upstream" effect may be caused on these wires.

- In addition to the point-to-point and point-to-multipoint wires between two interconnect tiles, the considered AMD devices also provide "long" wires that can be driven from multiple interconnect tiles. These are named `LH[0,12]`, `LV[0,9,18]`, and `LVB[0,12]` and span 12 to 18 columns or rows of tiles. As these wires are driven by "normal" interconnect resources as are the unidirectional wires, a *Conflict* or *Input Antenna* fault can occur in a resource that

Table 37: Configuration Examples for Intra-CLB Interconnect Resources in AMD 7 Series FPGAs

(a) `[ABCD]OUTMUX`: Slice Output Multiplexer

Source Wire	Config. Bit			
	b_0	b_1	b_2	b_3
I_1	1	.	.	.
I_2	.	1	.	.
I_3	.	.	1	.
I_4	1	.	.	1
I_5	.	1	.	1
I_6	.	.	1	1

(b) `[ABCD]5FFMUX`: Secondary FF Input Multiplexer

Source Wire	Config. Bit	
	b_0	b_1
I_1	1	.
I_2	.	1

(c) `PRECYINIT`: Carry Input Multiplexer

Source Wire	Config. Bit		
	b_0	b_1	b_2
Static “0”	.	.	.
Static “1”	1	.	.
I_1	.	1	.
I_2	.	.	1

is used by the design to drive these wires, as with any other interconnect wire. However, since these wires can be driven from multiple tiles, there exists the additional possibility that a bit flip activates a previously unused interconnect resource as an additional driver to an already used wire. This would then possibly lead to an additional *Input Antenna* or *Conflict* fault on the considered wire. These faults are referred to as *Long Wire Conflict* faults in the remainder of this work.

In AMD’s 7 Series FPGA, routing resources are not restricted to interconnect (`INT_[R,L]`) tiles. A small amount of interconnect resources can also be found in CLB tiles, mainly multiplexers driving the CLB tiles’ `[A,B,C,D]X` outputs, and inputs of the CLB tiles’ register cells (configurable as flip-flop or latch). Faults in these resources can be modeled in a similar way to the faults in interconnect resources in interconnect tiles. Examples for the configuration of intra-CLB are shown in Table 37[73]. Examples of two-stage interconnect resources with a decoded first stage and an encoded second stage are shown in Table 37a. In these multiplexers, a bit flip in the decoded stage ($b_0, \ldots, b_2$) causes similar faults as discussed above for generic interconnect resources found in interconnect tiles. A bit flip in the encoded stage, however, causes an *Open* or *Bridge* fault by selecting a different input wire without causing a short circuit. Furthermore, also single-level multiplexers (see Tables 37b and 37c) are present in CLB tiles. Similar observations regarding intra-slice routing have been made in [124].

Lattice iCE40 UltraPlus

Lattice’s iCE40 UltraPlus FPGAs are structured as a grid of similar tiles, either *logic* tiles containing eight LUT-flip-flop combinations (referred to as Programmable Logic Blocks (PLBs)) or *RAM* tiles containing block memory. These tiles also contain interconnect resources to connect (1) to eight

[73] Adapted from `https://web.archive.org/web/20250115224420/https://f4pga.github.io/prjxray-db/artix7/tile_clbll_l.html`, accessed 2026-01-05

global interconnect networks and (2) neighboring tiles. This grid of rows and columns is surrounded by *IO* tiles that provide access to the FPGA's pins, global buffers, as well as special tiles containing additional hard IP such as PLLs, additional RAM, MAC cells, and Serial Peripheral Interface (SPI) and Inter-Integrated Circuit Bus (I^2C) controllers [209].

Third-party documentation about Lattice's iCE40 and iCE40 Ultra Plus architecture and configuration are available from Project IceStorm [219], that also includes a functioning, entirely open-source HDL-to-bitstream toolflow for these devices [205]. According to this documentation, the interconnect in these FPGAs provides several classes of wires:

- Eight global interconnect networks intended for clock and control signals
- *Span-4* wires bidirectionally connect either tiles across up to 5 columns (Span-4 Horizontal) or tiles across up to 5 rows (Span-4 Vertical).
- *Span-12* wires bidirectionally connect either tiles across up to 13 columns (Span-12 Horizontal) or tiles across up to 13 rows (Span-12 Vertical).
- *Global-to-local* and *Global-to-LUTFF* wires that can be driven from the global interconnect networks. The former can connect the global networks to local wires, and the latter can connect the global networks specifically to the shared control inputs of the 8 flip-flops in a logic tile.
- *Tile-local* wires that can connect *global-to-local*, *span*, or neighboring LUT output wires to the tile's logic cells and vice-versa
- Input and output wires of logic cells, for example in the case of a *logic* tile:
  - Control inputs that are shared among all 8 flip-flops in a tile (clock, set/reset, clock enable)
  - LUT inputs that can either be driven from *tile-local* wires or directly from carry-in of the tile or other LUT outputs in the same tile
  - LUT carry output wires that can drive local LUT inputs or the carry-out of the tile
  - Output wires of the tile's 8 LUT-flip-flop combinations available in the same tile and the eight neighbor tiles.

Wires of the *global-to-local* and *global-to-LUTFF* category, as well as tile-local wires and most LUT input wires are only driven by a single interconnect resource. This resource selects if the target wire is driven at all, and if this is the case, by which source wire.

Table 38 shows an example for the different configurations that drive a *tile-local* wire in tile X12Y16 in an iCE40 up5k FPGA. In all configurations where the target wire is driven, bit b_4 is set to "1", making it a *buffer enable* bit. The other four configuration bits, $b_0, \ldots, b_3$, are binary encoded to select one of the 16 source wires to drive the target wire. Thus, it is assumed that this interconnect resource is implemented as an encoded multiplexer (e.g., Figure 11c) controlled by $b_0, \ldots, b_3$ with a subsequent buffer, logic gate, pass transistor or transmission gate controlled by b_4. If the routing resource is not used, all five bits are assumed to be set to zero, and the circuitry controlled by b_4 ties the wire to a defined value or sets it to a high-impedance state.

A single bit flip in these resources can thus have the following consequences:

- If the routing resource is not used, and one bit of $b_0, \ldots, b_3$ flips from "0" to "1", one wire of `X12Y16/sp12_h_r_8` ($b_0, \ldots, b_3$ are set to 0001), `X12Y16/neigh_op_bot_0` (0010),

Table 38: Driver configurations for *tile-local* wire `X12Y16/local_g0_0` in iCE40 up5k

Source Wire	Source Wire Type	Configuration Bits				
		b_0	b_1	b_2	b_3	b_4
`X12Y16/sp4_r_v_b_24`	Span-4 Vertical	.	.	.	.	1
`X12Y16/sp12_h_r_8`	Span-12 Horizontal	.	.	.	1	1
`X12Y16/neigh_op_bot_0`	Neighbor LUT out	.	.	1	.	1
`X12Y16/sp4_v_b_16`	Span-4 Vertical	.	.	1	1	1
`X12Y16/sp4_r_v_b_35`	Span-4 Vertical	.	1	.	.	1
`X12Y16/sp12_h_r_16`	Span-12 Horizontal	.	1	.	1	1
`X12Y16/neigh_op_top_0`	Neighbor LUT out	.	1	1	.	1
`X12Y16/sp4_h_r_0`	Span-4 Horizontal	.	1	1	1	1
`X12Y16/lutff_0/out`	Same-Tile LUT out	1	.	.	.	1
`X12Y16/sp4_v_b_0`	Span-4 Vertical	1	.	.	1	1
`X12Y16/neigh_op_lft_0`	Neighbor LUT out	1	.	1	.	1
`X12Y16/sp4_h_r_8`	Span-4 Horizontal	1	.	1	1	1
`X12Y16/neigh_op_bnr_0`	Neighbor LUT out	1	1	.	.	1
`X12Y16/sp4_v_b_8`	Span-4 Vertical	1	1	.	1	1
`X12Y16/sp12_h_r_0`	Span-12 Horizontal	1	1	1	.	1
`X12Y16/sp4_h_r_16`	Span-4 Horizontal	1	1	1	1	1

`X12Y16/sp4_r_v_b_35` (0100), or `X12Y16/lutff_0/out` (1000) may receive an additional load (the MUX input) instead of the original wire `X12Y16/sp4_r_v_b_24` (0000). Other than this change in *Output antennas*, no impact on the circuit is assumed as the *buffer enable* bit b_4 remains set to "0".

- If the routing resource is not used, and bit b_4 flips from "0" to "1", the wire `X12Y16/local_g0_0` is driven from `X12Y16/sp4_r_v_b_24` (0000). However, as in such configuration the wire `X12Y16/local_g0_0` will not have been routed to any other endpoints, this will likely not impact the circuit further.

- If the routing resource is used, and one bit of $b_0, \ldots, b_3$ flips, the selected wire of the affected routing resource will change. This causes the original route to be disconnected and another source wire to be selected. Depending on the use of this new source wire, this will either constitute an *Open* fault if the original source wire is not driven or a *Bridge* fault that drives the target wire from some other cell of the circuit than intended by the design.

- If the routing resource is used, and bit b_4 flips from 1 to 0, the route becomes disconnected in an *Open* fault.

Since the considered local wire cannot be driven by any other routing resource, a bit flip cannot cause a conflict fault on this wire.

On the other hand, wires of the *Span-4* and *Span-12* category and a subset of LUT input wires are driven by two or more interconnect resources. In this case, all potentially driving interconnect resources are assumed to have an *off* configuration where their driver is set to a high-impedance state (i.e., tri-stated).

As an example, Table 39 shows the interconnect resources and associated configurations that can drive a *Span-4* horizontal wire in tile `X12Y16` of an iCE40 up5k FPGA. The destination wire can be driven from five different interconnect resources:

Table 39: Driver configurations for wire `X12Y16/sp4_h_l_37` in iCE40 up5k

Source Wire	Source Wire Type	Configuration Bits								
		b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
`X8Y16/sp4_h_l_37`	Span-4 Horizontal	.	.	1						
`X8Y16/sp4_v_t_42`	Span-4 Vertical	.	1	.						
`X8Y16/sp4_h_l_41`	Span-4 Horizontal	.	1	1						
`X8Y16/sp4_v_t_37`	Span-4 Vertical	1	.	.						
`X8Y16/sp4_v_b_0`	Span-4 Vertical	1	.	1						
`X8Y16/sp4_h_l_44`	Span-4 Horizontal	1	1	.						
`X8Y16/sp4_v_b_6`	Span-4 Vertical	1	1	1						
`X8Y16/lutff_0/out`	LUT out				1					
`X9Y16/sp12_h_r_2`	Span-12 Horizontal					1				
`X10Y16/lutff_4/out`	LUT out						1			
`X12Y16/sp4_h_r_0`	Span-4 Horizontal							.	.	1
`X12Y16/sp4_v_b_7`	Span-4 Vertical							.	1	.
`X12Y16/sp4_h_r_4`	Span-4 Horizontal							.	1	1
`X12Y16/sp4_v_b_0`	Span-4 Vertical							1	.	.
`X12Y16/sp4_v_t_37`	Span-4 Vertical							1	.	1
`X12Y16/sp4_h_r_9`	Span-4 Horizontal							1	1	.
`X12Y16/sp4_v_t_43`	Span-4 Vertical							1	1	1

- A binary encoded multiplexer-type resource in tile `X8Y16` that can select one out of eight Span-4 wires to drive the considered wire, controlled by bits $b_0, \ldots, b_2$
- Three buffer-type resources in tiles `X8Y16`, `X9Y16`, and `X10Y16` that are controlled by one configuration bit each (b_3, b_4, b_5)
- Another binary encoded multiplexer-type resource in tile `X12Y16` than can select one out of eight source wires, controlled by configuration bits $b_6, \ldots, b_8$.

It is assumed that in a default configuration, when the configured circuit does not use the wire, all bits $b_0, \ldots, b_8$ will be set to “0”.

A source wire for the configuration pattern 000 is missing in the encodings for the two multiplexer-type resources in tiles `X8Y16` and `X12Y16`. It is assumed that this all-zero configuration disables the tri-state driver of the considered resource. If one of these interconnect resources is used, a single bit flip may turn off the driver if a configuration of 001, 010, or 100 is used, and the bit set to “1” flips to “0”, causing an *open* fault. If any other bit flips or the original configuration contains more than one bit set to “1”, a different source wire is selected, causing a *bridging* fault.

If the interconnect resource is unused (i.e., all three configuration bits are “0”), a single-bit flip will turn on the driver, selecting one of the source wires corresponding to the 001, 010, or 100 configurations. This may constitute an *Output Antenna* fault if the destination wire is entirely unused, but the source wire is used by the design. If the destination wire is driven by another interconnect resource (for example, to connect unrelated tiles using this *Span-4* or *Span-12* wire), the result of the bit flip above can be either a *Input Antenna* fault (if the source wire is unused) or a *Conflict* fault (if the source wire is used). The wires corresponding to configurations 011, 101, 110, and 111 will never be part of these faults because a single-bit flip cannot activate an unused routing resource and select these source wires.

Table 40: Connections between Logic and Interconnect in Lattice ECP5 FPGAs

Num.	Wire Name Pattern	Direction	Function
8	F[0-7]	Logic →Int.	Connect LUT outputs or BRAM/DSP interface to interconnect resources.
8	Q[0-7]	Logic →Int.	Connect FF outputs or BRAM/DSP interface to interconnect resources.
32	[A,B,C,D][0-7]	Int. → Logic	Connect interconnect to LUT inputs or BRAM/DSP interface.
8	M[0-7]	Int. → Logic	"Multipurpose" Inputs, connecting to, e.g., multiplexers
2	LSR[0,1]	Int. → Logic	Set/Reset for FFs; Can be connected to slices via MUXLSR[0-3]
2	CLK[0,1]	Int. → Logic	Clock for FFs; Can be connected to slices via MUXCLK[0-3]
4	CE[0-3]	Int. → Logic	Clock Enable for FFs (one per slice)

In the interconnect resources controlled by a single bit in tiles X8Y16, X9Y16, and X10Y16, a bit flip may turn off a used driver, causing an *Open* fault. If the driver is unused, on the other hand, it may be enabled by a single-bit flip. The outcome of this fault once again depends on the use of the wire in the original circuit, constituting once again either an *Output Antenna*, *Input Antenna* or *Conflict* fault, depending on whether the source wire and destination wires are used by the design.

Lattice ECP5

FPGAs in Lattice's ECP5 family are structured as a grid of tiles that contain both logic and routing interleaved with rows of block memory and multiplier blocks (i.e., "DSPs"). The logic contained in these tiles can either be programmable functional units (i.e., tiles that contain four slices comprising two LUTs and two flip-flops each) or interface logic to block memory or DSPs [210]. Each tile, furthermore, contains routing resources to connect (1) to other tiles, and (2) to the inputs and outputs of the logic located in the tile.

The third-party bitstream documentation initiative *Project Trellis*[74] provides a device database[75] as well as C++ and Python APIs (libtrellis and pyTrellis, respectively)[76] that provide programmatic access to this database.

According to the documentation published by *Project Trellis*[77], the general interconnect in ECP5 devices is based on unidirectional interconnect wires (cf. Figure 13) of different lengths. The logic (or BRAM/DSP interface) contained within the tiles is connected to interconnect resources via several input and output wires shown in Table 40. Interface wires in tiles connecting to BRAM/DSP are denoted with a prefix of "J" (i.e., a BRAM output is denoted JF0) but is configured using the same configuration pattern as the corresponding LUT output in an equivalent tile containing logic.

In addition to the wires interfacing with logic (or an adjacent BRAM/DSP block), the general

[74] See https://web.archive.org/web/20250126214546/https://prjtrellis.readthedocs.io/en/latest/, accessed 2026-01-05

[75] See https://web.archive.org/web/20250126223514/https://github.com/YosysHQ/prjtrellis-db, accessed 2026-01-05

[76] See https://web.archive.org/web/20220706213931/https://prjtrellis.readthedocs.io/en/latest/libtrellis/overview.html, accessed 2026-01-05

[77] See https://web.archive.org/web/20250126214602/https://prjtrellis.readthedocs.io/en/latest/architecture/general_routing.html, accessed 2026-01-05

Table 41: Wire Lengths in Generic Interconnect in Lattice ECP5 FPGAs

Num.	Length	Name	Purpose	Example
8	0	X0	Tile-local interconnect for increasing routing options	H00L0100
8	1	X1	Tile-local and connection to neighboring tile	V01S0001
32	2	X2	Connect to neigboring tile and the tile after	H02E0101
16	6	X6	Connect to tiles at ±3 and ±6 columns/rows	V06N0003

interconnect of ECP5 devices consists of horizontal and vertical interconnect wires of different lengths that connect to tiles across 0, 1, 2, or 6 columns or rows. Horizontal wires follow a naming pattern of H0[0,1,2,6][L,R,E,W]0[i]0[0,1,2,3], where L (left) and R (right) indicate the direction for H00 wires and E (east) and W (west) indicate it for the other wire lengths. Vertical wires follow the pattern V0[0,1,2,6][T,B,N,S]0[i]0[0,1,3], where T (top) and B (bottom) indicate the direction for V00 wires, and N (north) and S (south) apply to other lengths of vertical wires. Wires of lengths X1, X2, and X6 are driven within a single tile but appear as a source wire in two tiles as shown in Table 41.

In addition to the general interconnect, ECP5 devices contain 16 global networks to distribute clock and control signals. These wires are driven from special multiplexer tiles[78], and are not the target of this modeling approach. However, these global signals appear in general interconnect resources as source wires to choose from as G_HPBX[00-16]00.

Each tile contains 48 interconnect resources driving inputs to the tile's logic and 64 resources driving general interconnect wires. Additionally, four MUXLSR multiplexers allow selecting for each slice which of the two LSR signals (or a static "1") shall be connected to the local set/reset input. The same is the case for four MUXCLK multiplexers selecting the clock among CLK0, CLK1 and a static "1" for each slice. Finally, each of the eight F[0-7] LUT outputs may either be driven by the LUT directly, or by a wider function output (F5A, FXA, F5B, FXB, F5C, FXC, F5D, FXD). In total, each tile contains 128 interconnect resources that are configured by one (F[0-7]), two (MUX[CLK,LSR][0-3]), eight ([HV]06[...]), nine ([HV]00[...], [HV]01[...], [HV]02[...], CE[0-3], LSR[0,1]), or ten ([A,B,C,D][0-7], CLK[0,1]) configuration bits.

The configuration pattern for the multiplexers driving F[0-7] (see Table 42b for an example) indicate that this is implemented using a binary encoded 2-to-1 multiplexer. In contrast, the pattern for multiplexers driving MUX[CLK,LSR][0-3] (see Table 42c for an example) indicates a single-level decoded multiplexer, e.g., using pass transistors or transmission gates. Finally, general interconnect resources and ones driving slice logic inputs follow the pattern shown in Table 42a. This pattern is identical to the patterns observed in AMD 7 Series devices, as shown in Table 35. Thus, the ECP5 technology likely also uses decoded multiplexers based on pass transistors or transmission gates.

The similarities to AMD 7 Series FPGAs concerning interconnect resource configuration layout and interconnect structure make Lattice ECP5 devices vulnerable to the same categories of interconnect faults. However, as the ECP5 technology does not use (1) bidirectional wires and (2) "pseudo-PIPs" for driving wires to VCC, there are fewer special cases to consider. Thus, most single-bit flips in interconnect-related configuration bits in ECP5 FPGAs cause *Open* faults (if a bit flips from "1" to "0") or *Input Antenna* or *Conflict* faults if a bit flips from "0" to "1"). Additionally, the multiplexers driving F[0-7] from either a LUT directly or a wide-function output are prone to *Bridge* faults.

[78] See https://web.archive.org/web/20250126214604/https://prjtrellis.readthedocs.io/en/latest/architecture/global_routing.html, accessed 2026-01-05

Table 42: Exemplary Interconnect Configuration Patterns for Lattice ECP5 FPGAs

(a) Multiplexer driving General Interconnect Wire H01W0000

Source Wire	Configuration Bits								
	b_0	b_1	b_2	b_4	b_3	b_5	b_6	b_7	b_8
Q3	1	.	.	.	1	.	.	.	.
Q4	.	1	.	.	1	.	.	.	.
Q2	.	.	1	.	1	.	.	.	.
Q5	.	.	.	1	1	.	.	.	.
G_HPBX0500	1	.	.	.	.	1	.	.	.
Q0	.	1	.	.	.	1	.	.	.
G_HPBX0100	.	.	1	.	.	1	.	.	.
Q1	.	.	.	1	.	1	.	.	.
F3	1	.	.	.	.	.	1	.	.
F4	.	1	.	.	.	.	1	.	.
F2	.	.	1	.	.	.	1	.	.
W3_H06E0103	.	.	.	1	.	.	1	.	.
Q7	1	.	.	.	.	.	.	1	.
F0	.	1	.	.	.	.	.	1	.
Q6	.	.	1	.	.	.	.	1	.
F1	.	.	.	1	.	.	.	1	.
F6	1	.	.	.	.	.	.	.	1
F7	.	1	.	.	.	.	.	.	1
F5	.	.	1	.	.	.	.	.	1
E3_H06W0103	.	.	.	1	.	.	.	.	1

(b) Multiplexer Driving LUT Output F0

Source Wire	Config. Bits
	b_0
F0_SLICE	.
F5A_SLICE	1

(c) Multiplexer Driving MUXCLK0

Source Wire	Config. Bits	
	b_0	b_1
CLK1	1	.
CLK0	.	1

4.2.3 Modeling Faults in Other Resources

In addition to LUTs and interconnect, most FPGAs provide other resources, among the most important are (1) flip-flops or latches, (2) memory blocks in different widths and port configurations, (3) arithmetic blocks often containing multipliers or MAC logic (i.e., DSP blocks), (4) clock management blocks containing, for example, PLLs, and (5) I/O blocks interfacing with the device's pins. Furthermore, more complex blocks ranging from microcontroller-style peripherals (e.g., I^2C hard IP in Lattice's iCE40-up5k) to high-speed transceivers including Serializer-Deserializer (SERDES) logic and complex interfaces such as PCIe (both found in AMD 7 Series devices) may be included.

A first step for assessing the effects of faults affecting these elements is done during bitstream analysis when the netlist cells corresponding to the respective resource have been identified. Bits concerning resources unused by the post-implementation netlist may be less often critical than those concerning resources actively used.

While each LUT initialization bit corresponds to a single LUT (or a co-located LUT pair when the initialization bit may affect both outputs of a fracturable LUT), single configuration bits can also influence multiple FPGA resources. One example for this effect are the bits CEUSEDMUX, CLKINV, FFSYNC, LATCH, and SRUSEDMUX in CLB tiles in AMD 7 Series FPGAs. These bits configure clock enable usage, clock polarity, synchronous/asynchronous reset, FF/Latch, and set-reset usage

for all eight[79] register cells in a AMD 7 Series CLB[80]. Thus, a single bit flip in the CEUSEDMUX configuration bit can affect the clock enable signal for all eight flip-flops, switching it between a static "1" value and a signal received via the interconnect. All potentially affected cells must be identified during bitstream parsing in these cases. A similar effect can be seen in PLC2 tiles in Lattice's ECP5 technology, where the per-slice bits SLICE[A,B,C,D].GSR affect both flip-flops in each slice by connecting or disconnecting them from the global set-reset signal[81].

The effects of these faults can be modeled by cross-referencing the affected configuration bit with bitstream documentation and, if available in the necessary depth, with the vendor's documentation for the affected logic elements.

In this way, the assumed effects can also be integrated into adapted simulation models for the affected resources for a gate-level fault simulation. Once again taking the CEUSEDMUX configuration bit in AMD 7 Series FPGAs as an example, modeling this fault for gate-level analysis consists of the following steps:

1. Identify the FPGA location (tile and slice) affected by the particular CEUSEDMUX bit
2. Identify the flip-flops cells placed within this FPGA location
3. Model the effect of the CEUSEDMUX bit flip in a gate-level netlist for fault simulation:
    - If the bit flips from "0" to "1", the signal connected to the "CE" input of the slice now drives the clock-enables of the flip-flops instead of a static "1". If no signal is connected, it is likely still pulled to "1", leading to no effect on the netlist. If, however, it is connected to a signal, the simulation netlist must be adapted to reflect this change.
    - If the bit flips from "1" to "0", the clock-enables of the affected flip-flops are now driven by a static "1", which must be reflected in the simulation netlist.
4. Perform a simulation run to determine the criticality of the effects of the bit flip.

As the design needs to be simulated on gate level, the scalability of this simulation-based approach to larger designs may be limited.

Considering all bits documented for AMD xc7a35t FPGA by Project XRAY, 76% pertain to the inter-tile interconnects (in INT_[R,L] tiles), 18% concern LUT initialization, 3% concern other CLB tile configuration (i.e., flip-flop/latch, LUT/LUTRAM, carry block configuration), 2% pertain to intra-CLB routing multiplexers, 0.3% are found in BRAM tiles, and another 0.2% in DSP tiles. The remaining 0.5% configure the remaining tile types (e.g., clock management, I/O blocks, PCIe and high-speed transceiver blocks. Thus, the focus of this dissertation lies on LUT and interconnect (inter-tile and intra-CLB) faults (responsible for 96% of configuration bits), while the modeling of faults affecting other resources is mainly restricted to identifying the affected netlist cells. Modeling these faults and supplementing this modeling with fault simulation may nevertheless prove useful in future work to further reduce the number of false-positively predicted critical bits.

[79] of which only four are affected by the LATCH bit; the other four register cells cannot be configured as latch

[80] See https://web.archive.org/web/20250115224420/https://f4pga.github.io/prjxray-db/artix7/tile_clbll_l.html, accessed 2026-01-05

[81] See https://web.archive.org/web/20231107175527/http://yosyshq.net/prjtrellis-db/ECP5/tilehtml/PLC2.html, accessed 2026-01-05

4.3 Characterization of Interconnect Fault Consequences

As discussed in Section 4.2.2, the exact consequences of an SEU in the configuration bits responsible for the interconnect strongly depend on the actual circuit implementation of the interconnect resources. Modern FPGA's interconnects provide a variety of different types of interconnect (for example, wires connecting logic blocks, BRAM, DSPs, and other resources to the general interconnect, local routing of varying degrees – short wires connecting neighboring logic resources, longer wires to connect resources in the extended neighborhood, and long wires to carry signals across the chip – as well as global routing for clock, and possibly control signals. Examples are shown in Table 32 for AMD's Artix 7 as well as Tables 40 and 41 for Lattice's ECP5 technology. As the PIPs connecting to those different types of interconnect may differ in their circuit structure or properties (e.g., driver strength), we argue that systematic analysis of the effects of faults in each type of resource is needed to draw conclusions about their criticality. This is the primary motivation for proposing the systematic experiments described in this section. These experiments analyze a significant subset of all routable connections in a given interconnect resource (or an entire interconnect tile) to identify individual resources, or wires, that are relevant to overall design criticality.

While knowledge about the logical mapping between configuration bits and routing configurations, as shown in Section 4.2.2, can be used to deduce which bit flip may cause which kind of topological alteration of the design's interconnect, the effect this has on its functionality depends on the actual physical implementation of the affected interconnect resource(s). This, however, does not directly suggest how the interconnect resources are implemented – and thus how they perform in the presence of faults – on the circuit level. For example, consider the interconnect multiplexer shown in Figure 15, which is based on patents by AMD [211, 214] and corresponds to the configuration layout in Table 35. Here, configuration bits $b_0, \ldots, b_4$ each control four transmission gates (or pass transistors) in the first stages, while $b_5, \ldots, b_9$ each control one transmission gate or pass transistor in the second stage.

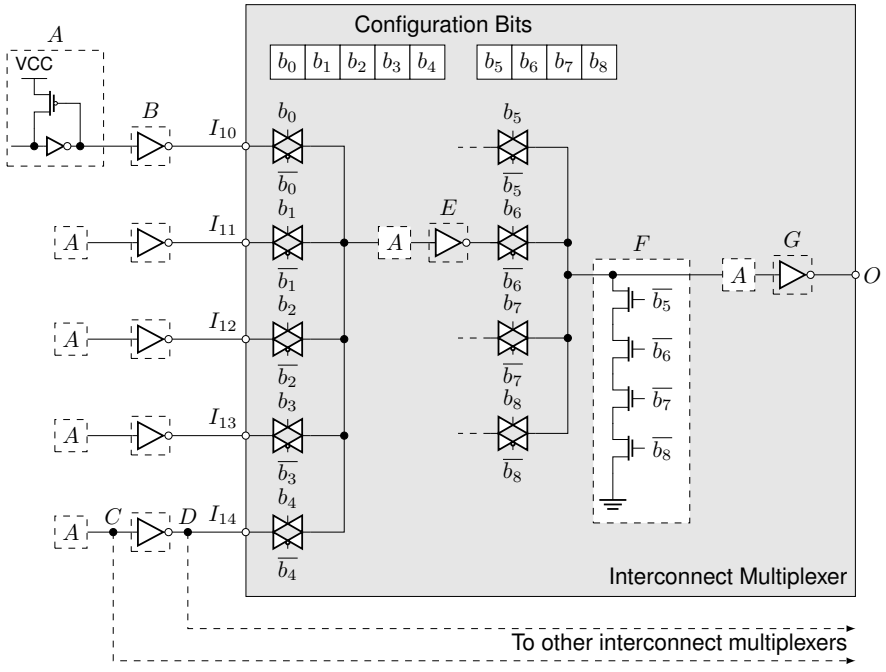


Figure 15: Exemplary Decoded Interconnect Multiplexer (based on [211, 214])

The drive strength of unused input wires may depend on which (weak) hold circuitry ("A" in Figure 15) is used at the source (e.g., a logic element in a function tile or a driver in another interconnect tile). An example is the half-latch discussed in [155] in the context of AMD's Virtex-5 FPGAs.

Furthermore, the buffer driving the particular interconnect wire could influence the outcome of a fault – especially considering the transistor sizing that may lead to differing drive strengths for "0" and "1"[82]. Considering the variety of wire types in modern FPGA interconnects, differences in implementation and sizing depending on wire types are likely. For example, one could assume that drivers for physically longer wires (e.g., a wire that connects two tiles separated by multiple rows/columns in a AMD 7 Series FPGA) may be stronger than drivers that only drive a local interconnect wire. If both of these drivers are involved in a conflict fault, the stronger wire may always "win" against the weaker one, allowing to judge the effects of particular faults from the bitstream alone.

Another influence on the consequences of *Conflict* faults may be the buffering of interconnect wires – once per interconnect tile ("C" in Figure 15) or for each interconnect multiplexer individually ("D" in Figure 15): Assume that in Figure 15 route $I_{11} \rightarrow O$ is configured initially, and I_{14} is routed via a different interconnect resource. A single bit flip in b_4 then turns on the transmission gate for I_{14} in this interconnect resource. If each wire is buffered in each interconnect resource individually (I_{14} fans out to the different interconnect resource at point "C"), I_{14} may interfere with the logical value driven by I_{11}. However, the original route for I_{14} itself may not be impacted because it fans out before the individual buffer. If the interconnect resource is unbuffered, or each input is buffered only once for all interconnect resources (the fan-out occurs at point "D"), both initially configured routes may be affected by the short circuit induced by the bit flip in b_4.

Furthermore, the outputs of each first-stage multiplexer may be unbuffered, weakly buffered (e.g., using another half-latch – "A"), or fully buffered ("E").

Finally, the behavior of an undriven wire segment in the design depends on which circuit structure is present to pull the wire segment to a defined logic level if not in use. The subcircuit labeled "F" in Figure 15 depicts an implementation as a NAND-style pulldown structure found in [214]. If none of the transmission gates in the second stage are turned on, pull-down transistors (controlled by the same configuration bits) pull the input of the output buffer low, causing the output signal to be driven to high if unused (which is consistent with behavior observed by [46]). An alternative to prevent the wire from floating would be using another half-latch ("A"). Finally, the drive strength of buffer "G" in Figure 15 may affect the consequences of interconnect faults in other interconnect resources or tiles, where wire O is available as a source (and thus available as an incident wire for *Input Antenna* or *Conflict* faults.

As information about the circuit implementation of FPGA devices is generally not disclosed by vendors, the circuit-level effects of the faults determined by bitstream-level modeling have to be determined experimentally. This section describes an approach to do so systematically.

4.3.1 Experimental Setup

The focus of the characterization experiments are the effects of bit flips that form new connections (*Input Antenna*, *Conflict*, *Bridge*) rather than break existing ones. However, it may also be used to characterize *Open* faults as is done for Lattice ECP5. The effect of such faults on the involved signals is systematically evaluated for the different types of source and destination wires a given FPGA type provides. It is assumed that if the considered fault type can impact the concerned

[82] i.e., weak "0"/strong "1" or strong "0"/weak "1"

signal(s), this is visible at the end of the route as an impacted logical value that mismatches the one transported by the “Golden Route”.

For all three FPGA types discussed in Section 4.2.2, there are certain faults that may cause *Input Antenna* faults that potentially affect a single signal by connecting an additional upstream wire segment. Furthermore, there is potential in both technologies for *Conflict* faults that cause a short circuit between two previously unrelated nets. Finally, as the iCE40 UltraPlus technology uses encoded multiplexers, here also *Bridge* faults are expected that disconnect one involved signal and cause a conflict in another. This may also be the case in specific interconnect resources in AMD 7 Series and Lattice ECP5 technologies, whose interconnect encoding predominantly exposes them to other fault types.

The circuit shown in Figure 16 has been implemented in a variant containing only the signals “Route under Test 1” and “Golden Route” to experimentally evaluate faults that impact only one signal (i.e., *Input Antenna* and *Open* faults). A “signal generator” controlled by the circuit’s control portion drives two routes via input buffers implemented in 1-input LUTs. The “golden route” is routed so that it is unaffected by the fault injection operations carried out on the other route, the “Route under test 1”. This generator can apply a static 0, static 1, or a toggling logic value to the two routes. The outputs of both routes are then buffered again using 1-input LUTs. Their output logic values are compared by a 2-input LUT that generates a “mismatch” signal. A counter increments at every clock cycle where this “mismatch” signal is active. This count serves as a result of each fault injection experiment: If the count is non-zero after injecting a particular fault, the fault caused the considered signal to change value when compared to the “golden route” in at least one clock cycle.

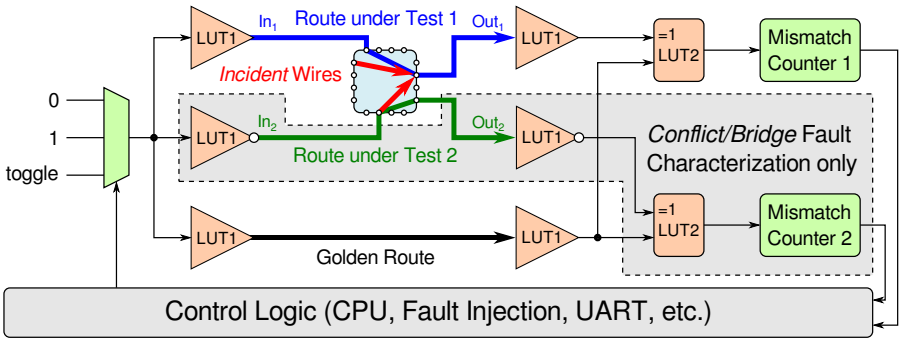


Figure 16: Characterization Circuit for *Input Antenna*, *Open*, *Conflict*, and *Bridge* Interconnect Faults

The control logic tests the routes with all three possible inputs (static ‘0’, static ‘1’ and toggle), reads out and resets the mismatch counter, and communicates results back to the controlling PC. On FPGAs that allow access to the configuration at run time (i.e., that are dynamically and partially reconfigurable), such as AMD’s 7 Series FPGA via ICAP, the control logic can also inject the required faults into the configuration. On FPGAs not providing this functionality, the PC generates a faulty bitstream and subsequently programs it into the FPGA e.g., via IEEE 1149.1 (“Joint Test Action Group (JTAG)”), for each fault pattern to be tested. This potentially leads to a lower fault injection rate, as an entire bitstream is downloaded for each test.

To evaluate faults with two involved signals (i.e., *Conflict* and *Bridge* faults), a second “route under test” along with a second comparator LUT and mismatch counter, is added to the circuit shown in Figure 16. Its buffer LUTs are inverted compared to the first “route under test”. This generates a complementary logic value in the relevant portion of this second “route under test” compared to the first, causing worst-case conditions for *Conflict* faults, increasing the possibilities for effects to be observed. Table 43 shows expected effects of *Conflict* faults for the output segment (“Out_1”)

of “Route under Test 1” in Figure 16. Effects may be observed on both routes if the interconnect resource used to route “Route under test 1” (subject to the injected fault) can propagate fault effects back “upstream” to the output portion of “Route under test 2”. Thus, similar effects to those described for Out_1 in Table 43 may be observed for Out_2. These conditions can be deduced similarly for *Input Antenna* and *Open* faults from the reported mismatch counts.

Table 43: Possible Effects of *Conflict* Faults on Segment Out_1 of “Route under Test 1”

(a) No Effect

Out_1	In_2 \ In_1	0	1
	0	0	1
	1	0	1

(b) Wired AND

Out_1	In_2 \ In_1	0	1
	0	0	0
	1	0	1

(c) Wired OR

Out_1	In_2 \ In_1	0	1
	0	0	1
	1	1	1

(d) Forced By In_2

Out_1	In_2 \ In_1	0	1
	0	0	0
	1	1	1

Random effects during manufacturing may cause the properties of transistors (e.g., dopant density, oxide thickness, as well as length and width) to vary, leading to inter-chip and intra-chip variation of, e.g., threshold voltages, gate capacitances, and drive strengths. Furthermore, the width and thus resistance of (metal) interconnect wires may also be affected by these variations [220]. The effects of the interconnect faults to be tested may be sensitive to these variations: The performance of a level-restoring buffer may be impacted (affecting the outcome of *Open* faults as well as the relative drive strengths of the wires involved in *Conflict* faults. Thus, to achieve reliable results, the discussed experiments should be conducted (1) on multiple FPGA devices, (2) with a variety of involved interconnect wires and in different locations within the FPGA, and (3) under controlled environmental conditions (i.e., supply voltage and chip temperature). Ideally, comparing results across multiple devices and varied environmental conditions yields information about the influence of PVT variations on the behavior of the target devices under single-bit errors.

Implementation for Characterizing *Input Antenna* Faults

For evaluating the effects of *Input Antenna*-like faults, a variant of the circuit shown in Figure 16 containing *Route under Test 1* and *Golden Route* is implemented for each route to be tested. In each of these implementations, the route is constrained via the FPGA implementation tool’s constraint mechanism to a particular path to be tested. Subsequently, a set of bitstreams, each containing one particular route to be tested, must be generated and subjected to fault injection, observing and recording the output of the route with each of the test patterns (“0”, “1”, toggle). This procedure depends on the structure of the tested FPGA interconnect.

In the following, the implementation of these experiments for AMD 7 Series, Lattice iCE40, and Lattice ECP5 devices is discussed.

AMD 7 Series FPGAs In these devices, the (general) interconnect resources are found in `INT_L` and `INT_R` tiles. As discussed in Section 4.2.2, these devices’ interconnect mainly consists of unidirectional wires. Interconnect resources are implemented as multiplexers that select which input wire of the tile drives a particular output wire. Thus, a route under test is characterized by a particular PIP in a given interconnect tile, i.e., a specific pair of start and end nodes that can be connected using a single interconnect resource. This data can be extracted, e.g., from the data published by Project XRAY[83]. Additionally, the `route_via` Tcl function from the Project XRAY

[83] See `https://web.archive.org/web/20250113214512/https://f4pga.github.io/prjxray-db/artix7/segbits_int_l.db`, accessed 2026-01-05

`utils.tcl` library[84] can be used to generate the necessary routing constraints comfortably using AMD Vivado. Once a bitstream containing the route to be tested has been successfully generated (some pairs of endpoints cannot be freely routed with the design shown in Figure 16, such as connections to the control inputs of a logic tile's flip-flops), it can be subjected to fault injection. Referring to Tables 35 and 36, for each of these routes under test, all bits capable of causing *Input Antenna* faults must be tested. To that end, each of these bits must be flipped individually in a separate experiment. As AMD 7 Series FPGAs support runtime access to their configuration via an *Internal Configuration Access Port (ICAP)*, this can be done with one bitstream containing the original route under test. The corresponding faults can be injected (and corrected) at run time.

An example can be seen in Figure 17, where the route under test has been routed via the input wire `NE2END0` into an `INT_L` tile. Within this tile, it has been routed to the output wire `NE2BEG0`, and thus utilizes the interconnect multiplexer driving that wire. Shown in grey in the figure are alternative drivers for `NE2BEG0`. In the general AMD 7 Series Interconnect, only a single interconnect multiplexer may drive an output wire in one interconnect tile. Thus, the possible sources for both an *Input Antenna* or *Conflict* fault caused by a single-bit error are a subset of the wires shown here.

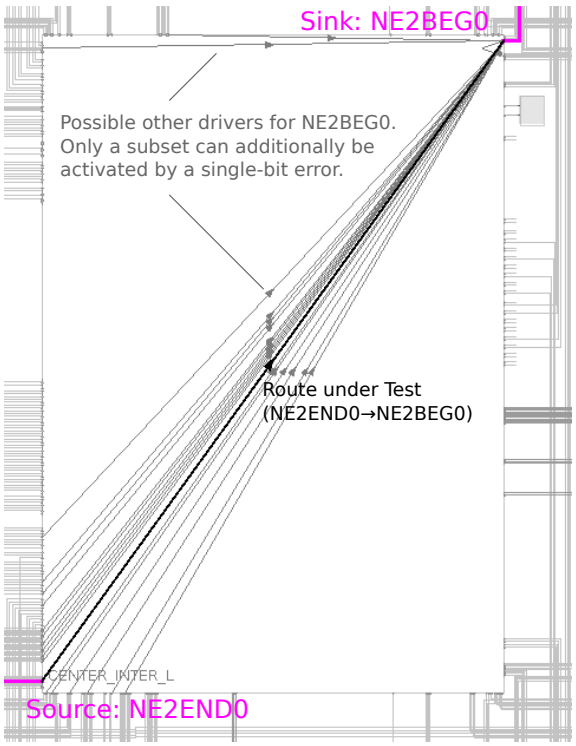


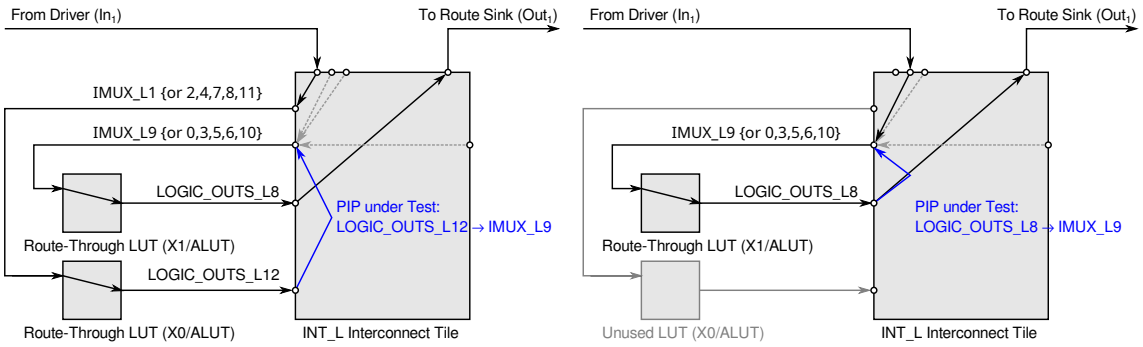
Figure 17: Example Route under Test for AMD 7 Series FPGA (Screenshot: AMD Vivado)

AMD 7 Series FPGAs provide two principial interconnect tile types: one that connects to its corresponding function tile via its left side (`INT_L`) and one that connects via its right side (`INT_R`). Both of these tile types have 182 source wires and 172 destination wires. However, not all output wires can be reached from all input wires, as the tiles are organized as Wilton switch matrices [47]. Of the 172 destination wires, 96 allow routing from 20 source wires, while 64 allow routing from 24. The remaining 12 destination wires can only be driven from 12, 16, or 18 possible sources. The Wilton switch matrix allows 3636 routable source-destination pairs. However, the single-route-under-test variant of the circuit shown in Figure 16 cannot be routed via all combinations:

[84] See `https://web.archive.org/web/20250124151547/https://github.com/f4pga/prjxray/blob/master/utils/utils.tcl`, accessed 2026-01-05

- All 32 paths originating in each of the eight flip-flops (`LOGIC_OUTS[0-7]` in an interconnect tile next to a `CLB` tile) cannot be used as routes-under-test using the circuit described above, as the CLB flip-flop cannot be bypassed. This results in 256 unroutable combinations[85]. In interconnect tiles connecting to tile types other than CLB tiles (e.g., BRAM and DSP tiles), these options for route-throughs do not exist.
- The same applies to two paths originating from the `GND_WIRE`, a static "0" source present in every interconnect tile (2 unroutable combinations).
- There are also output wires to which the route-under-test cannot be routed, as they correspond to specific inputs of the corresponding logic tile. This concerns 16 paths that can drive each of the two clock and reset inputs per logic tile (64 unroutable combinations).
- Finally, several source-destination pairs are logically incompatible. This is the case because routing the route-under-test through some particular source wires (direct LUT outputs `LOGIC_OUTS[8-15]`, as well as Slice MUX outputs `LOGIC_OUTS15` – `LOGIC_OUTS23` in an interconnect tile next to a `CLB` tile) or destination wires (`IMUX0` – `IMUX47`) require routing *through* a LUT – this works by initializing the LUT to a function that replicates the logic value on one of its input to its output. In routes that test a combination of a `LOGIC_OUTS[8-23]` source and an `IMUX` destination, thus, two route-through LUTs are required (see Figure 18a). However, as can be seen in Figure 18b, some combinations would require to first route the signal under test via a LUT to the appropriate `LOGIC_OUTS` source wire, then via the PIP to be tested, and finally via the same LUT as before (due to the `IMUX` wire to be tested being an input to the very same LUT). As the `IMUX` wire in this combination would then have two drivers, this is considered a non-routable combination and subsequently not tested. This concerns an additional 16 source-destination pairs.

With 338 connections unavailable for the route-under-test, at most 3298 of the 3636 possible source-destination pairs are testable with the described circuit in interconnect tiles next to CLB tiles. Fewer connections can be tested in interconnect tiles adjacent to other logic tile types. While not all combinations can be tested with the proposed circuit, hypotheses based on the testable routes can be formed that can be verified against criticality in fault injection results of actual circuits that, e.g., also contain flip-flops.



(a) Routeable (b) Non-Routeable

Figure 18: Combinations of `LOGIC_OUTS` source and `IMUX` destination in AMD 7 Series FPGAs

[85] While AMD Vivado uses register cells configured as latches as route-through cells during regular routing, this could not be forced to implement the experiments described in this section.

In CLB tiles, from the perspective of the injected *Input Antenna* fault, there are 3499 pairings of source wire and incident (fault source) wire to be tested. However, as pairs of these wires may appear as inputs of multiple interconnect resources, the bit with which this connection is made must also be considered. Conversely, depending on the state of the other configuration bits in the interconnect resource, the same bit may cause different faults depending on the context (this is visible from the example shown in Table 35 – bit b_1 causes a fault between I_0 and I_1 if b_0 and b_5 are set, but a fault between I_1 and I_2 when b_2 and b_5 are set. In total, with the generated bitstreams for an `INT_L` tile next to a `CLB` tile, there are 19958 of these "fault combinations" (3-tuples of original source wire, incident (fault source) wire, and bit causing the fault) to test.

However, some combinations of routes-under-test and *Input Antenna* faults to be injected are invalid, as shown in Figure 19. It can be seen in this example that a configuration can arise where the connection configured by a fault would cause the route-under-test to loop back on itself. While this may be an interesting fault mode to explore, it is considered out of the scope of this work due to its rarity – concerning 119 (0.6%) of the 19958 pairs in the generated bitstreams. Thus the "fault combinations" that would cause this kind of behavior are excluded, leaving 19839 combinations to be tested.

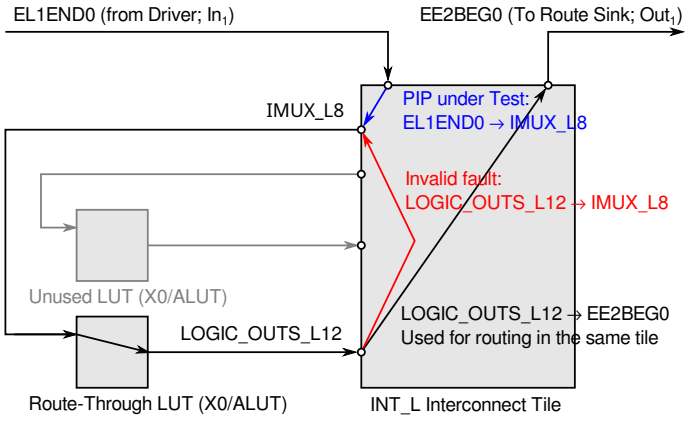


Figure 19: Invalid combination of Route-under-Test and Incident Wire in AMD 7 Series

Finally, as the `IMUX` and `LOGIC_OUTS` wires discussed above interface directly with the logic in the function tile *next* to the considered interconnect tile – and an influence of the connected logic to the behavior of unused wires in an *Input Antenna* fault is possible – multiple interconnect tiles were tested.

The procedure for characterizing *Input Antenna* faults was implemented for all compatible routes in the interconnect tiles of an AMD xc7a35t-cpg236 FPGA shown in Table 44. In tiles next to function tiles that contain other logic than LUTs (found in CLB tiles), such as BRAM or DSP, some additional connections become unroutable. This is due to the lack of LUTs available for pass-through routing of `LOGIC_OUTS` sources or `IMUX` sinks. This, however, does not mean that these wires cannot be used as *incident* route that is connected in an *Input Antenna*-like fault to another route (i.e., connected via fault injection to a routable combination).

Table 44: Interconnect Tiles Evaluated for *Input Antenna* Faults on AMD Artix 7 xc7a35t

Interconnect Tile	Neighbor Tile	Neighbor Description
INT_L_X26Y70	CLBLL_L_X26Y70	Logic tile
INT_R_X27Y70	CLBLM_R_X27Y70	Logic tile (+LUTRAM)
INT_L_X30Y70	BRAM_L_X30Y70	Block memory
INT_L_X30Y71	BRAM_L_X30Y70	Block memory
INT_L_X30Y72	BRAM_L_X30Y70	Block memory
INT_L_X30Y73	BRAM_L_X30Y70	Block memory
INT_L_X30Y74	BRAM_L_X30Y70	Block memory
INT_L_X34Y70	DSP_L_X34Y70	DSP
INT_L_X34Y71	DSP_L_X34Y70	DSP
INT_L_X34Y72	DSP_L_X34Y70	DSP
INT_L_X34Y73	DSP_L_X34Y70	DSP
INT_L_X34Y74	DSP_L_X34Y70	DSP
INT_L_X0Y80	LIOI3_X0Y79	I/O logic
INT_L_X18Y74	VFRAME_X47Y77	Empty tile
INT_L_X18Y76	MONITOR_BOT_X46Y79	XADC System Monitor
INT_L_X30Y100	PCIE_BOT_X71Y115	PCIe Hard Macro
INT_L_X42Y80	CMT_FIFO_L_X107Y85	I/O FIFO

As a means to identify potential influences of chip-to-chip variation on the experimental results, experiments were carried out on four identical Artix 7 xc7a35t devices (i.e., repeated using multiple boards) on four Digilent BASYS3 development boards[86]. The devices were from a mix of date- and lot codes: one device (device 1) with date code *1909* and lot code *D5834810A* and three devices (devices 2–4) with date code *1913* and lot code *D5851384A*.

The following experiment sequence was performed:

1. Configure the FPGA with the bitstream to be tested, containing the control logic and the route-under-test.
2. Select a bit to test, i.e., a bit that, when flipped, causes an additional segment to be connected in the assumed decoded interconnect multiplexer.
3. Check the route with each input value (0, 1, or 50 MHz toggle) in the fault-free circuit to identify potential effects remaining from the previous experiment. Each pattern is applied for a delay defined by the MicroBlaze CPU of approximately 10000 clock cycles at 50 MHz, i.e., $20\mu s$. The mismatch count output by the counter should return zero for each input value in these fault-free cases.
4. Flip the selected bit from 0 to 1.
5. Check the route with each input value again and record the mismatch count.
    - If the fault is benign, the mismatch counter for all input values will report zero.
    - If the fault only affects a specific logic value, the mismatch count for either static input value 0 or 1, as well as the mismatch count for the toggle input value, will be non-zero, while it will be zero for the other static input value.

[86] See https://web.archive.org/web/20241227120346/https://digilent.com/reference/programmable-logic/basys-3/start, accessed 2026-01-05

- If the fault mainly affects timing significantly (that leads to a violation of the input flip-flops setup/hold time or makes the data arrive even later), the mismatch count for the static input values will be minimal or even zero. In contrast, the mismatch count for the toggling pattern will be significant.

6. Flip the selected bit back to 0, correcting the introduced fault, and continue with Step 2 if more bits can be tested for the current route.

7. Otherwise, continue with Step 1 with the next bitstream (i.e., route to be tested).

The experiments were conducted at a controlled temperature environment (measured chip temperature range of 46 °C – 48 °C) to address potential temperature dependencies of faults.

Lattice iCE40 In the bidirectional interconnect of Lattice's iCE40 FPGAs, each interconnect wire can have multiple drivers in multiple logic tiles and drive multiple inputs in different tiles. Here, a route under test is characterized by the specific wire to be tested, and a pair of driver and driven interconnect resources. For each generated bitstream containing a particular route-under-test, the set of bits that can cause an *Input Antenna*-like fault must be tested.

Figure 20 shows an example for a route under test on iCE40 UltraPlus, which has been routed via the wire `X12Y16/sp4_h_l_37`. The route under test originates from another interconnect wire (`X8Y16/sp4_v_h_0`), that drives the considered wire in tile X8Y16. It drives a local wire in tile X10Y16. However, there are several alternative or additional drivers in tiles X8Y16, X9Y16, X10Y16, and X12Y16. A subset of these drivers are alternatives to the source chosen in X8Y16 (other inputs of the binary encoded multiplexer). Of the remainder, only a subset can be configured additionally as a result of a single-bit error (`X8Y16/lutff_0/out`, `X9Y16/sp12_h_r_2`, `X9Y16/sp12_h_r_2`, `X10Y16/lutff_4/out`, `X12Y16/sp4_h_r_0`, `X12Y16/sp4_v_b_7`, and `X12Y16/sp4_v_b_0`, see Table 39).

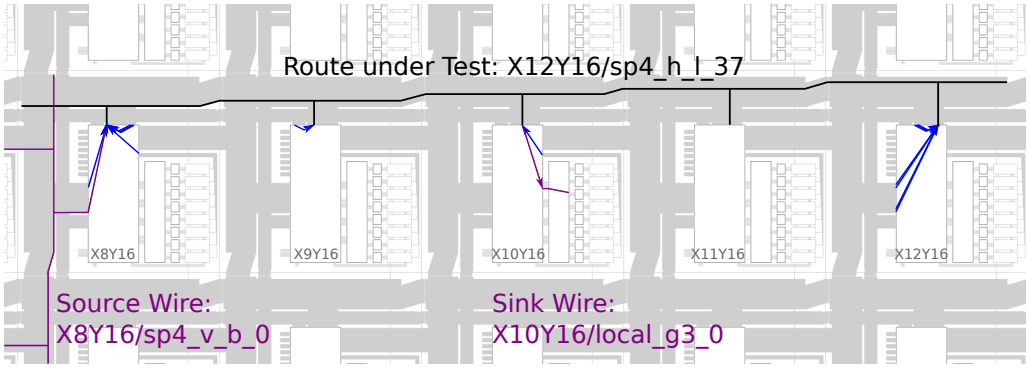


Figure 20: Example Route under Test for Lattice iCE40 UltraPlus FPGA (Screenshot: NextPnR)

As Lattice iCE40 FPGAs do not support run-time reconfiguration, bitstreams containing the respective fault to be tested must be generated beforehand by modifying the original bitstream. The device under test is then configured with these bitstreams one by one, and the experiments are carried out without the need to flip certain bits at run time. Ten iCE40 devies with identical markings[87], likely indicating identical date and lot, on board of ten iCEBreaker[88] development boards were used.

[87] Device markings: `ICE40UP5` ↩ `SG48I` ↩ `A9011RC9` ↩ `ED7K78YG11`

[88] See `https://web.archive.org/web/20250126163023/https://github.com/icebreaker-fpga/icebreaker`, accessed 2026-01-05

Lattice ECP5 Due to the similarity of ECP5's interconnect to the one found in AMD 7 Series, a similar approach was followed to characterize these devices' interconnect for *Input Antenna* faults. However, as the employed open-source NextPNR implementation tool does not support directed routing (i.e., a routing mode that can complete the routing of a partially constrained route), this needed to be implemented as an add-on to generate the testing circuits. Finally, in the same way as for Lattice iCE40, partial reconfiguration could not be utilized for fault injection. Instead, erroneous bitstreams were generated beforehand and were loaded into the target device via a IEEE 1149.1 ("JTAG") interface to perform the fault injection experiments. Loading full-device bitstreams severely limited the fault injection rate to approx. 0.04 faults per second (i.e., approx. 23 seconds per fault).

Thus, the scope of *Input Antenna* fault characterization experiments for ECP5 was reduced to one tile (`R16C48:PLC2`) of one tile type: the `PLC2` tile (containing LUTs and FFs). The `PLC2` tile was chosen for *Input Antenna* fault characterization as this tile type (1) is the most abundant in ECP5 FPGAs – `3036` tiles (70.4%) out of `4312` are `PLC2` tiles in an LFE5U-25F device; combined, they account for 87.3% of all interconnect bits – and (2) these tiles contain LUTs that can be used as route-throughs, extending the number of connections that may be tested using conflict faults. Each `PLC2` tile supports 2242 unique connections to route the characterization input wire (In_1) to the characterization output wire (Out_1), some necessitating using a LUT configured as a buffer to forward the signal – i.e., forming a route-through. This, however, is not possible in the same way with the circuit shown in Figure 16 for wires driving flip-flop clock and control inputs or the logic tiles' "miscellaneous" (`M[0-7]`) inputs, and wires driven by flip-flops or wide-function multiplexers. Thus, of the 2242 possible connections in the tile `R16C48:PLC2`, 1619 were routable for *Input Antenna* fault characterization.

While *Open* faults in AMD 7 Series have already been explored in literature [46, 124], no prior fault characterization for Lattice ECP5 FPGA exists. Thus, in addition to *Input Antenna* faults, the effects of *Open* faults were also characterized in the experiments. In this way, 3238 single-bit errors causing *Open* faults (i.e., two for each of the 1619 connections, see Table 42a) and 11594 causing *Input Antenna* faults could be evaluated.

Fault injection experiments were conducted on a pair of *iCESugar-Pro*[89] FPGA boards providing a LFE5U-25F-6BG256C FPGA. The experiments were conducted under controlled temperature conditions at an environment temperature of 35 °C. The two devices carried identical markings[90], likely indicating identical date and lot code.

Implementation for Characterizing *Conflict* or *Bridge* Faults

To evaluate the effects of *Conflict* or *Bridge* faults, the two-wire variant of the circuit shown in Figure 16 is implemented on the target device. Once again, routing constraints confine the signals to the routing resources to be tested. How the pair of signals are routed precisely to test conflict or bridging faults depends on the structure of the FPGA interconnect.

In the following, the implementation of these experiments for the same FPGA types as characterized for *Input Antenna* faults (AMD 7 Series, Lattice iCE40, and Lattice ECP5) is discussed.

AMD 7 Series A simplified view of an interconnect tile used in these devices is shown in Figure 21. In these interconnect tiles, most input signals can be routed to multiple output signals in a Wilton

[89] See `https://web.archive.org/web/20240926174830/https://github.com/wuxx/icesugar-pro`, accessed 2026-01-05

[90] `LFE5U-25F` ↩ `6BG256C` ↩ `A9421R96` ↩ `MALAYSIA 1943` ↩ `DR6K0HR88`

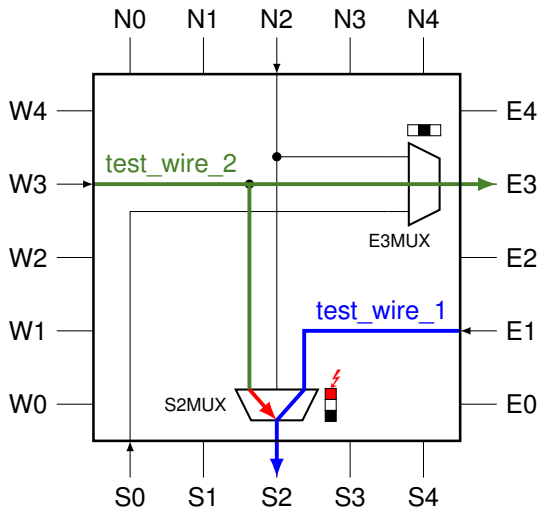


Figure 21: Exemplary Routing Configuration for Evaluating Conflict Faults in Wilton Interconnect Tile

switch matrix configuration [47]. This is implemented by the same signal being a configurable driver in multiple interconnect resources (i.e., interconnect multiplexers). While three-input multiplexers configured by three bits are used in the simplified example in Figure 21, AMD 7 Series FPGAs use two-stage decoded multiplexers. The respective configuration patterns must be observed when generating faults to be characterized.

To evaluate the effects of a contention between two signals, the following procedure is followed to prepare a routed design for every interconnect resource to be tested:

1. Route the first "route under test" (`test_wire_1` Figure 21) via a PIP to be tested and identify its configuration pattern. Its source and destination wire define this PIP. In the simplified example in Figure 21, the PIP `E1->S2` is selected, corresponding to the pattern 001 for `S2MUX`.

2. Considering the bit-level fault model for the targeted interconnect resource (`S2MUX` in the example), identify which source wires can be shorted to the selected PIP's destination wire by a single bit flip. These wires are candidates for the fault's "*incident*" wire.

3. Select one candidate among these wires and identify which interconnect resources can be driven from it. In the example in Figure 21, `W3` and `N2` are available as candidate *incident* wires. `W3` can be shorted to `S2` by changing the configuration pattern from 001 to 101, while it needs to be changed to 011 to short `N2` to `S2`[91].

4. Select one candidate interconnect resource, thereby defining the candidate destination wire and the PIP to be used in routing. In the example in Figure 21, `E3MUX` is identified as the only candidate interconnect resource, and thus `E3` is defined as the destination wire.

5. Attempt to route the second route-under-test, `test_wire_2` via the PIP defined in the previous step. In the example, this is `W3->E3`. If this is not possible, repeat with a different choice of candidate *incident* source wire or candidate interconnect resource until all possibilities are exhausted. If all possibilities are exhausted before generating a valid route, the PIP is not testable in this manner.

[91] Changing `S2MUX`'s configuration pattern to 000 would constitute an *Open* fault.

Once a routed design (and its bitstream) exist, the behavior of *Conflict* faults in the targeted interconnect resource can be evaluated by observing the mismatch counts reported for `test_wire_1` and `test_wire_2` under fault injection for the different input patterns (“0”, “1”, “toggle”) for both routes. This procedure is similar for any FPGA technology that predominantly uses an unidirectional interconnect structure and decoded multiplexers as interconnect resources. Thus, the same procedure can be followed to characterize Lattice ECP5 devices.

Characterization experiments for *Conflict* faults were carried out on the same set of AMD Artix 7 devices as the *Input Antenna* experiments described above.

Lattice iCE40 On an interconnect that is structured like the one found in Lattice’s iCE40 FPGAs, a different implementation of the test circuit shown in Figure 16 is necessary. Here, initially, the first route-under-test (`test_wire_1`) must be routed via the wire that should be the target of the injected *Conflict* or *Bridge* fault. Subsequently, the configuration patterns for all drivers that may additionally connect to this wire need to be searched for patterns that can be enabled by a single bit flipping from zero to one (i.e., the faults caused by an SEU). For example, consider the configuration patterns of the interconnect resources shown in Table 39. If the original routing path drives this wire from `X8Y16/lutff_0/out`, only the corresponding bit $b_{l,3}$ would be set to one, and all other configuration bits would be set to zero. Thus, a single bit flip may lead to:

- the patterns 001, 010 or 100 on bits $b_{l,0}...b_{l,2}$, corresponding to an additional driver from wires `X8Y16/sp4_h_l_37`, `X8Y16/sp4_v_t_42`, or `X8Y16/sp4_v_t_37`.
- the additional activation of a connection from `X9Y16/sp12_h_r_2` via bit $b_{l,4}$
- the additional activation of a connection from `X10Y16/lutff_4/out` via bit $b_{l,5}$
- the patterns 001, 010 or 100 on bits $b_{l,6}...b_{l,8}$, corresponding to an additional driver from wires `X8Y16/sp4_h_r_0`, `X8Y16/sp4_v_b_7`, or `X8Y16/sp4_v_b_0`.

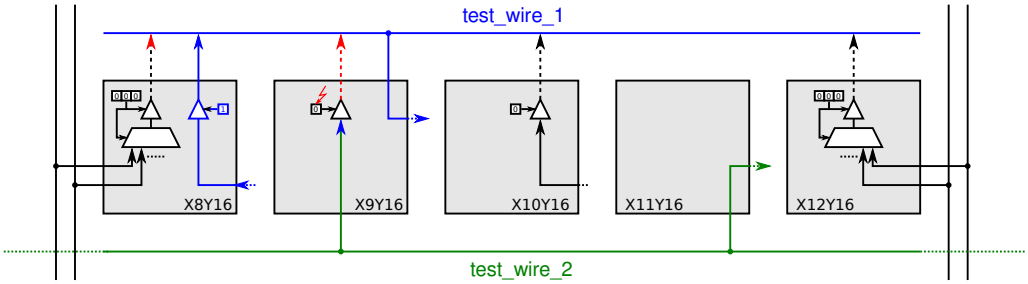


Figure 22: Routing for Evaluation of Conflict Faults in Simplified iCE40 Interconnect

To test the effects of conflict faults, the second route-under-test (`test_wire_2`) must be routed via each of these *potential* conflict wires, resulting in a bitstream for each combination.

These bitstreams then must be altered in the position that leads to possible conflict with that pair of wires. As Lattice iCE40 devices are supported by an open-source toolchain [205] that generates ASCII configuration files holding the configuration bit assignments for each tile, bitstream modification is a trivial task. The ASCII bitstreams are then converted to a bitstream according to the configuration format expected by iCE40 devices that can be loaded via SPI. This is done using the open-source `icepack` tool provided by Project IceStorm [219]. Characterization experiments for *Conflict* faults were carried out on the same set of Lattice iCE40 devices as the *Input Antenna* experiments described above.

Lattice ECP5 The ECP5 technology allows 15812 combinations of shorting In_2 to Out_1 (i.e., *Conflict* faults) in each `PLC2` tile. The same limitations of toolchain and fault injection rate with ECP5 and the employed FPGA boards apply to *Conflict* faults as described for the characterization of *Input Antenna* faults in ECP5 technology in Section 4.3.1. Thus, a random sample of 2003 routable combinations was selected from the 15812 possible conflict faults in a single `PLC2` tile (`R16C48:PLC2`). These combinations were was subjected to fault injection experiments to identify the effects of possible single-bit flips. While not delivering a complete device characterization, these results allow to assess the relative criticality of *Open*, *Input Antenna*, and *Conflict* faults on these devices.

Due to the structural similarities of Lattice ECP5's interconnect to AMD's 7 series interconnect, a similar approach was followed for routing the connections to be tested: (1) select a PIP for the $In_1 \rightarrow Out_1$ part of the route, route `test_wire_1` from the input buffer LUT to In_1 and from Out_1 to the output buffer LUT and (2) among the source wires of the interconnect multiplexer employed by `test_wire_1`, select a different source wire that is both present in another multiplexer and may be shorted to Out_1 via a single bit flip, and attempt to route $In_2 \rightarrow Out_2$ via that connection. The resulting characterization experiments involved 1176 routed connections for `test_wire_1` between 126 source and 88 destination wires.

The discussed experiments were conducted on the same pair of *iCESugar-Pro* boards as discussed with *Input Antenna* faults.

4.3.2 Results

This section presents the results of the experiments conducted to characterize interconnect faults on the targeted FPGA devices as well as the conclusions that may be drawn from them for fault analysis and criticality prediction using the experimental setup discussed in Section 4.3.1.

Input Antenna Faults

First, the characterization results and subsequently formed assumptions for criticality prediction are described for the targeted AMD Artix 7, Lattice iCE40, and Lattice ECP5 devices. While the interconnect in AMD Artix 7 devices was extensively characterized for *Input Antenna* faults, qualitative characterization results are provided for Lattice iCE40 and ECP5 devices to demonstrate how the characterization approach may be extended to other FPGA types.

AMD 7 Series Tables 45 and 46 summarize the results obtained for the *Input Antenna* case on AMD Artix 7 in tiles `INT_L_X26Y70` and `INT_R_X27Y70` (both adjacent to a CLB tile) by the respective *incident* wire (i.e., the fault source wire that is connected by the fault to the original destination). As a fault combination, the 3-tuple of (1) original connection, (2) incident wire, and (3) flipped configuration bit is considered. Incident wires that lead to mismatches in the route-under-test's output (Out_1) on any of the four devices and in any of the input values to the route-under-test are referred to as "critical" fault combinations. The "Evaluated" column shows in how many fault combinations the respective incident wire was tested. The "Union" ($\cup$) column shows in how many of these tests the fault was critical on *any* of the four devices, while the "Intersection" ($\cap$) column in how many it was critical on *all* of the devices. Furthermore, the data for each of the four evaluated devices is shown.

Table 45: Evaluated vs. Critical *Input Antenna* Faults by Incident Wire in `INT_L_X26Y70` in AMD Artix 7 xc7a35t

Incident Wire	Tested	Critical Combinations					
				Devices			
		∪	∩	1	2	3	4
BYP_BOUNCE1	77	6	3	6	6	3	6
GND_WIRE	6	6	6	6	6	6	6
LOGIC_OUTS_L8	158	158	158	158	158	158	158
LOGIC_OUTS_L9	195	195	195	195	195	195	195
LOGIC_OUTS_L10	163	163	163	163	163	163	163
LOGIC_OUTS_L11	195	195	195	195	195	195	195
LOGIC_OUTS_L12	195	195	195	195	195	195	195
LOGIC_OUTS_L13	163	163	163	163	163	163	163
LOGIC_OUTS_L14	195	195	195	195	195	195	195
LOGIC_OUTS_L15	163	163	163	163	163	163	163
NR1END1	122	6	1	6	2	1	6
WW4END1	114	6	1	6	3	4	6
170 Other Wires	18053	0	0	0	0	0	0
Total	19799	1451	1438	1451	1444	1441	1451

Table 46: Evaluated vs. Critical Faults *Input Antenna* by Incident Wire in `INT_R_X27Y70` in AMD Artix 7 xc7a35t

Incident Wire	Tested	Critical Combinations					
				Devices			
		∪	∩	1	2	3	4
BYP_BOUNCE1	77	6	0	6	2	0	5
GND_WIRE	6	6	6	6	6	6	6
LOGIC_OUTS8	163	163	163	163	163	163	163
LOGIC_OUTS9	195	195	195	195	195	195	195
LOGIC_OUTS10	163	163	163	163	163	163	163
LOGIC_OUTS11	195	195	195	195	195	195	195
LOGIC_OUTS12	195	195	195	195	195	195	195
LOGIC_OUTS13	163	163	163	163	163	163	163
LOGIC_OUTS14	185	185	185	185	185	185	185
LOGIC_OUTS15	163	163	163	163	163	163	163
LOGIC_OUTS16	163	1	0	1	0	0	0
LOGIC_OUTS23	163	1	0	1	0	0	0
NR1END1	122	6	1	4	5	1	5
SE6END1	108	1	0	1	0	0	0
SE6END2	108	1	0	1	0	0	0
WW4END1	114	6	0	6	4	0	0
166 Other Wires	17547	0	0	0	0	0	0
Total	19830	1450	1429	1448	1439	1429	1438

The results provided in Tables 45 and 46 show that there is only a small subset of the 182 tested incident wires that act critically. The wires LOGIC_OUTS_(L)[8-15] are involved in most critical faults, which lead to a mismatch in the original route in every tested case. This is also the case for one other wire, GND_WIRE, albeit with far fewer available fault combinations. In addition, four combinations with incident wire BYP_BOUNCE1, and two each with NR1END1 and WW4END1 were observed on all devices. Furthermore, there are some devices in which additional combinations with these source wires were critical. In tile INT_R_X27Y70, a small number of additional combinations with other source wires were sporadically observed critical. In total, approximately 7.3% of all evaluated *Input Antenna* faults were critical on any of the four devices in both INT_L_X26Y70 (1451 of 19799) and INT_R_X27Y70 (1450 of 19830).

The connections in which incident wires other than LOGIC_OUTS_L[8-15] caused critical *Input Antenna* faults are shown in Tables 47 and 48 for INT_L_X26Y70 and INT_R_X27Y70, respectively. It can be seen that most appear only as critical incident wires in the GFAN0 and GFAN1 interconnect resources. These resources can be driven from the clock network (GCLK[...]), a tile-local connection point (BYP_BOUNCE[...]), the tile-local ground wire (GND_WIRE) and two inter-tile wires (a connection from the tile below via NR1END1, and a connection from the tile four columns to the right via WW4END1). When the original wire is routed via a GCLK[...] signal to GFAN[0,1], it appears that configuring GND_WIRE, NR1END1, or WW4END1 as an incident wire can result in a critical fault. The incident wires LOGIC_OUTS[16,23] and SE6END[1,2] in INT_R_X27Y70 were only critical in connections of two particular BENTQUAD wires (SS6END[1,2] → SW6BEG[1,2]).

Table 47: Details for selected critical Incident Wires in INT_L_X26Y70 in AMD Artix 7 xc7a35t

Incident Wire	Original Routes
BYP_BOUNCE1	{GCLK_L_B0, GCLK_L_B4, GCLK_L_B8_WEST} → GFAN0
	{GCLK_L_B0, GCLK_L_B4, GCLK_L_B8_WEST} → GFAN1
GND_WIRE	{GCLK_L_B11_WEST, GCLK_L_B3, GCLK_L_B7_WEST} → GFAN0
	{GCLK_L_B11_WEST, GCLK_L_B3, GCLK_L_B7_WEST} → GFAN1
NR1END1	{GCLK_L_B1, GCLK_L_B5, GCLK_L_B9_WEST} → GFAN0
	{GCLK_L_B1, GCLK_L_B5, GCLK_L_B9_WEST} → GFAN1
WW4END1	{GCLK_L_B10_WEST, GCLK_L_B2, GCLK_L_B6_WEST} → GFAN0
	{GCLK_L_B10_WEST, GCLK_L_B2, GCLK_L_B6_WEST} → GFAN1

Table 48: Details for selected critical Incident Wires in INT_R_X27Y70 in AMD Artix 7 xc7a35t

Incident Wire	Original Routes
BYP_BOUNCE1	{GCLK_B0_EAST, GCLK_B4_EAST, GCLK_B8} → GFAN0
	{GCLK_B0_EAST, GCLK_B4_EAST, GCLK_B8} → GFAN1
GND_WIRE	{GCLK_B11, GCLK_B3_EAST, GCLK_B7} → GFAN0
	{GCLK_B11, GCLK_B3_EAST, GCLK_B7} → GFAN1
NR1END1	{GCLK_B1_EAST, GCLK_B5_EAST, GCLK_B9} → GFAN0
	{GCLK_B1_EAST, GCLK_B5_EAST, GCLK_B9} → GFAN1
WW4END1	{GCLK_B10, GCLK_B2_EAST, GCLK_B6} → GFAN0
	{GCLK_B10, GCLK_B2_EAST, GCLK_B6} → GFAN1
LOGIC_OUTS16	{SS6END2} → SW6BEG2
LOGIC_OUTS23	{SS6END1} → SW6BEG1
SE6END1	{SS6END1} → SW6BEG1
SE6END2	{SS6END2} → SW6BEG2

Finally, information about the consequences of the injected faults can be gained by analyzing which input value (“0”, “1”, or toggle) is affected by an injected fault. It was observed that when the incident wire is any one of LOGIC_OUTS(_L)[8-15], mismatches were only reported when the original route was either driven to the logic value “1” or the toggle pattern. The respective connections for LOGIC_OUTS wires are shown in Table 49, where it is visible that LOGIC_OUTS(_L)[8-15] connect directly to LUT outputs in interconnect tiles adjacent to CLB tiles. LOGIC_OUTS(_L)[0-7] connect to the adjacent tile’s flip-flop outputs, while LOGIC_OUTS(_L)[16-23] connect to the tile’s slice multiplexers that can be used to bypass the LUT, route the LUT’s fracturable output, the secondary FF output, wide function multiplexer, or carry logic (XOR) output. With GND_WIRE as incident wire, the effect was similar to *Input Antenna* faults with LOGIC_OUTS(_L)[8-15] as incident wires. However, faults were observed for the other (critical) incident wires when the original wire was transporting either “0” or the toggle value.

Table 49: CLB Function Output to LOGIC_OUTS Wire Mapping in AMD Artix 7

LOGIC_OUTS Wire	Driving Slice	Driving Function
LOGIC_OUTS(_L)[0-3]	0	{A,B,C,D}FF
LOGIC_OUTS(_L)[4-7]	1	{A,B,C,D}FF
LOGIC_OUTS(_L)[8-11]	0	{A,B,C,D}LUT
LOGIC_OUTS(_L)[12-15]	1	{A,B,C,D}LUT
LOGIC_OUTS(_L)[16-19]	0	{A,B,C,D}MUX
LOGIC_OUTS(_L)[20-23]	1	{A,B,C,D}MUX

In summary, from the characterization results of *Input Antenna* faults in interconnect tiles next to CLB tiles in AMD Artix 7 FPGAs described above, the following assumptions about their general behavior are made:

- If a previously unused wire segment is connected in an *Input Antenna* fault, the outcome greatly depends on the type of wire that is the source of this antenna connection. To a lesser part, the affected interconnect resource, via which the original signal is routed, also plays a role.
- The most critical incident wires in these interconnect tiles are the subset of LOGIC_OUTS wires that transport the LUT output value into the interconnect tile. In the conducted experiments, every single fault that (erroneously) connected this type of wire to an existing connection resulted in a disruption of the original signal, given that it was transporting a vulnerable logic value.
- Also critical – regardless of the type of incident wire – are faults affecting interconnect resources driving the GFAN0 and GFAN1 wires from a global clock signal (GCLK_[...]).
- Finally, different logical values are affected by *Input Antenna* faults depending on the type of incident wires and interconnect resources: When the incident wire is a LOGIC_OUTS or GND_WIRE wire, the original signal is only affected when it transports a value of “1”. In a small number of additional cases, concerning mainly faults within the GFAN0 and GFAN1 interconnect resources, the original signal is only affected if it transports “0”.
- The consequences of a fault in this category are, in most cases, benign (i.e., no change to the affected signal). In other cases, they affect either one of the transported logical values consistent with wired-AND or wired-OR behavior. Thus, while the change to the circuit

incurred by the fault may match the *Input Antenna* pattern shown in Figure 10d on a structural level, the behavior may better be described as *Conflict* fault (shown in Figure 10c) with a wire that is driven by a constant value, and whose output value does not propagate further into the design.

- There may be a small, device-dependent or PVT-sensitive subset of additional incident wires that *can* lead to critical *Input Antenna* faults on some devices or in some interconnect tiles. This behavior may lead to single configuration bits being classified as false-negative when applying the assumptions above to predict the criticality of *Input Antenna* faults.

Comparable results were observed with the remainder of interconnect tiles shown in Table 44: those connecting to other logic such as BRAM and DSP blocks instead of CLB tiles. Instead of `LOGIC_OUTS[8-15]` in `INT_L_X26Y70` and `INT_R_X27Y70`, there appears to be a tile-dependent subset of `LOGIC_OUTS[0-23]` wires that can become critical on all of the four FPGAs in all of the tested combinations. This ranges from one `LOGIC_OUTS` wire in `INT_L_X0Y80` next to an I/O Tile to all 24 in `INT_L_X34Y74` next to a DSP tile. Considering all interconnect tiles reported for an AMD Artix 7 xc7a35t FPGA by AMD Vivado, Table 34 shows the distribution of their neighbor tile types. As CLB, BRAM, and DSP tiles, in sum, make up 84% of the device's interconnect tile neighbors, this dissertation primarily considers these tile types for characterization. For interconnect tiles neighboring the abundant CLB tiles, this subset contains only `LOGIC_OUTS[8-15]`, i.e., the eight outputs of (unused) LUT within the adjacent CLB tile. The critical incident wires for BRAM and DSP tiles are marked with "Y" in Table 50a and Table 50b, respectively.

Apart from the wires critical in only one device, almost no wire is critical in the interconnect tile `INT_L_X18Y74`. This tile neighbors a tile that does not contain logic – referred to a `VFRAME` tile. The fact that there are no connections present to the `LOGIC_OUTS` wires is also displayed in Vivado's *Device* view. Thus, it is assumed that the `LOGIC_OUTS` wires can also be considered non-critical in other, non-tested, tiles that do connect to the neighbor tile's `LOGIC_OUTS` wires in a similar way. This data can be used to form predictions about the *Input Antenna* faults occurring in these interconnect tiles. For the remaining interconnect tiles (making up approx. 13% of an Artix 7 xc7a35t FPGA's interconnect tiles), similar assumptions of benignity can be made for regular incident wires. Concerning logic output wires (i.e., `LOGIC_OUTS`), the worst case (i.e., that all of these wires are critical incident wires) can be made without extensive pessimism.

The results observed across all tiles, and visible in Tables 45 and 46 indicate a degree of device-to-device variation between the four tested FPGA devices concerning which incident wires can cause errors in addition to the common subset of the `LOGIC_OUTS` wires. In some tiles, individual additional connections that can cause erroneous behavior of the route-under-test, were observed. This concerns mostly longer wires (`QUAD` and `BENTQUAD`, see Table 32) but in rare cases also shorter ones (`SINGLE`, `DOUBLE`), and, in the case of `INT_R_X27Y70` on one device the wires `LOGIC_OUTS[16,23]` driven by the CLB's output multiplexers. However, due to the scarcity of these effects, it is assumed that the predictions made from this characterization data do not lead to significant amounts of false negative assessments for individual interconnect bits.

Table 50: Critical LOGIC_OUTS Incident Wires (marked "Y") in Interconnect Tiles neighboring unused BRAM/DSP Tiles in AMD Artix 7 xc7a35t

(a) Neighboring Tile BRAM_L_X30Y70

Fault Source Wire	INT Tile (Row Offset)				
	+ 0	+ 1	+ 2	+ 3	+ 4
LOGIC_OUTS0	–	–	–	–	–
LOGIC_OUTS1	–	–	Y	–	–
LOGIC_OUTS2	–	–	–	–	–
LOGIC_OUTS3	–	–	–	–	–
LOGIC_OUTS4	–	–	–	–	–
LOGIC_OUTS5	–	–	Y	–	–
LOGIC_OUTS6	–	–	–	–	–
LOGIC_OUTS7	–	–	–	–	–
LOGIC_OUTS8	–	–	–	–	–
LOGIC_OUTS9	Y	Y	Y	Y	–
LOGIC_OUTS10	–	–	Y	–	–
LOGIC_OUTS11	–	–	Y	Y	–
LOGIC_OUTS12	Y	Y	Y	Y	Y
LOGIC_OUTS13	–	–	Y	–	–
LOGIC_OUTS14	Y	Y	Y	Y	Y
LOGIC_OUTS15	–	–	–	–	–
LOGIC_OUTS16	Y	Y	Y	Y	Y
LOGIC_OUTS17	–	–	–	–	–
LOGIC_OUTS18	–	Y	Y	Y	Y
LOGIC_OUTS19	–	–	Y	–	–
LOGIC_OUTS20	–	–	Y	–	–
LOGIC_OUTS21	Y	Y	Y	Y	Y
LOGIC_OUTS22	–	–	–	–	–
LOGIC_OUTS23	Y	Y	Y	Y	Y

(b) Neighboring Tile DSP_L_X34Y70

Fault Source Wire	INT Tile (Row Offset)				
	+ 0	+ 1	+ 2	+ 3	+ 4
LOGIC_OUTS0	Y	Y	Y	Y	Y
LOGIC_OUTS1	Y	Y	Y	Y	Y
LOGIC_OUTS2	Y	Y	Y	Y	Y
LOGIC_OUTS3	Y	Y	Y	Y	Y
LOGIC_OUTS4	Y	Y	Y	Y	Y
LOGIC_OUTS5	Y	Y	Y	Y	Y
LOGIC_OUTS6	Y	Y	Y	Y	Y
LOGIC_OUTS7	Y	Y	Y	Y	Y
LOGIC_OUTS8	Y	–	Y	Y	Y
LOGIC_OUTS9	Y	–	Y	–	Y
LOGIC_OUTS10	Y	Y	Y	Y	Y
LOGIC_OUTS11	Y	Y	Y	Y	Y
LOGIC_OUTS12	Y	Y	–	–	Y
LOGIC_OUTS13	Y	Y	Y	–	Y
LOGIC_OUTS14	Y	–	Y	Y	Y
LOGIC_OUTS15	Y	–	Y	Y	Y
LOGIC_OUTS16	Y	Y	Y	Y	Y
LOGIC_OUTS17	Y	Y	Y	Y	Y
LOGIC_OUTS18	Y	Y	Y	Y	Y
LOGIC_OUTS19	Y	Y	Y	Y	Y
LOGIC_OUTS20	Y	Y	Y	Y	Y
LOGIC_OUTS21	Y	Y	Y	Y	Y
LOGIC_OUTS22	Y	Y	Y	Y	Y
LOGIC_OUTS23	Y	Y	Y	Y	Y

Lattice iCE40 To characterize the behavior of iCE40 under *Input Antenna* faults, 922 different variants of the one-signal version of the circuit shown in Figure 16 were generated by routing the route under test on different paths from source to destination.

In total, 29107 fault injection experiments were performed, involving 3505 incident wires (In_1) and 1007 fault target wires (Out_1). `Span4`, `Span12`, and `logic_op/neigh_op/slf_op` wires were tested as source wires, while `Span4` and `Span12` were used as target wires.

Table 51: iCE40 *Input Antenna* Fault Characterization Summary

Tested Faults	Observed Critical on	
	$\geq$ 1 Device	All 10 Devices
29107	6426 (22.07%)	1440 (4.94%)

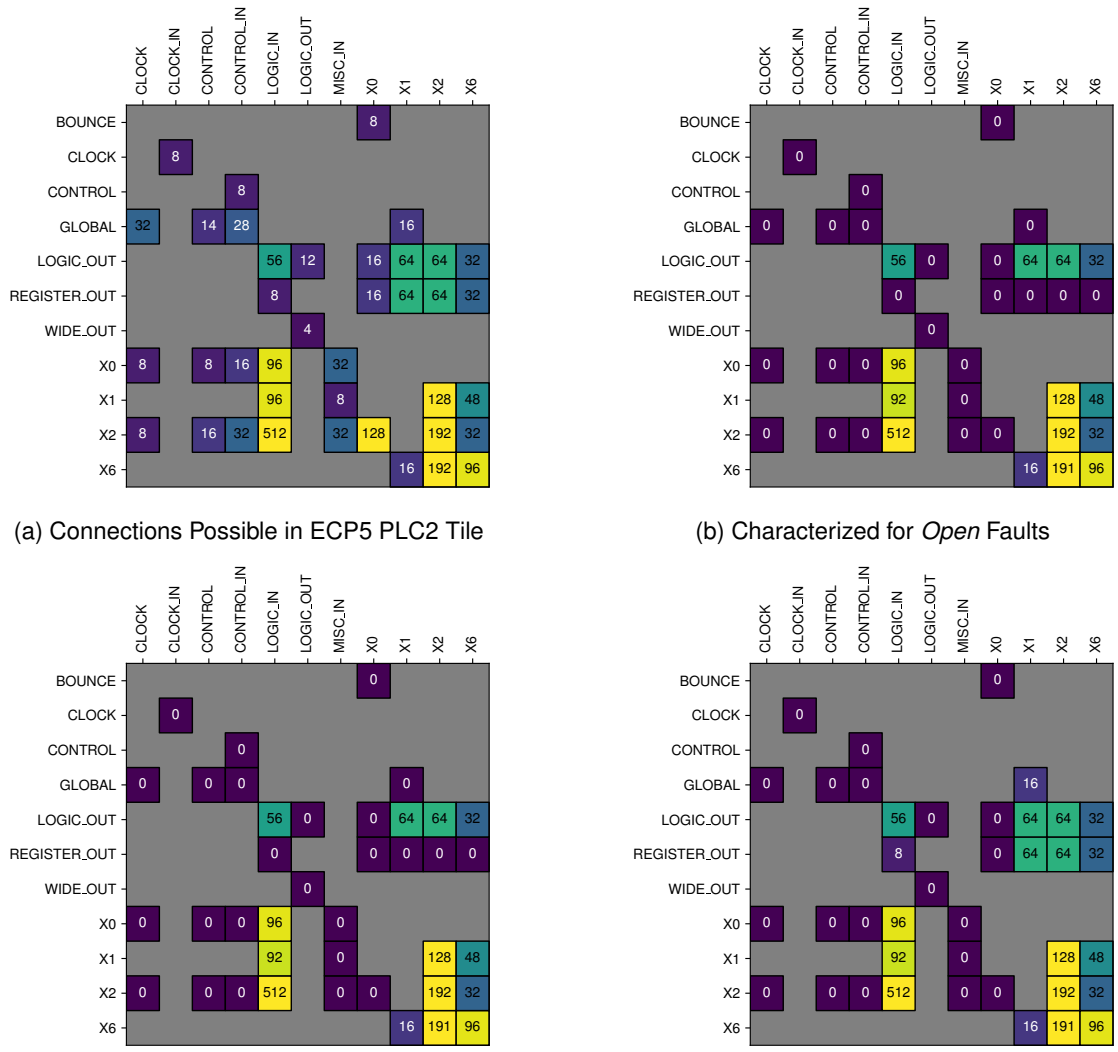
The results of this process are summarized in Table 51. In general, a large variation between the tested devices was observed regarding the criticality of individual injected faults. While this was also observed to some extent on the evaluated AMD devices, the effect is significantly higher on iCE40: overall, 22.1% of the *Input Antenna* faults were critical on at least one of the 10 tested devices, while only 4.95% were critical on all of them. Also, the signal values affected by the faults greatly varied between devices and individual interconnect resources of the same type. The most common effect – nevertheless observed only in 6.8% of injections – was that both “0” and “toggle” were affected by the fault on some devices, but not on all. Additionally, differences have been observed between different target wire types, where *Span12* behaved more uniformly than *Span4*.

The observed types of variability make predicting the effects of *Input Antenna* faults on Lattice iCE40 challenging. Future work should perform more in-depth analysis of affected wire types, and conduct experiments on a larger scale in order to identify patterns of (1) criticality and (2) inter-device variation to improve prediction quality.

Lattice ECP5 Figure 23 shows the number of connections possible in the characterized tile of type `PLC2` (see Figure 23a) organized by the type of source and destination wire. Organized in the same way, the number of different connections characterized for *Open* faults (see Figure 23b) and *Input Antenna* faults (see Figure 23c) are shown. Finally, the number of connections characterized for *Input Antenna* faults are organized by *incident* wire type (i.e., the type of unused wire that was erroneously connected due to the injected fault) and type of destination wire (see Figure 23d). It can be seen that certain types of connections were not tested (e.g., all originating from the output of a flip-flop – `REGISTER_OUT`, and all driving clock or control wires – `CLOCK`, `CLOCK_IN`, `CONTROL`, `CONTROL_IN`) due to the limitations lined out in Section 4.3.1.

The results of these experiments can be summarized as follows:

- On both devices, all *Open* faults caused a mismatch between the now-disconnected wire segment (Out_1) and the `golden_wire` (referring back to Figure 16). In all of the 3238 bits tested for the effects of *Open* faults, these mismatches only occurred when In_1 was driven to “0” and the “toggle” pattern. This suggests that, in the same way as AMD 7 Series devices, disconnected wire segments in Lattice’s ECP5 technology are pulled to a logical “1”.
- In 11562 of 11594 bits (i.e., 99.7%) tested for the effects of *Input Antenna* faults, no mismatches between Out_1 and the `golden_wire` were recorded. This, once more, mirrors the behavior of AMD 7 Series devices, where the overwhelming majority of *Input Antenna* faults do not affect the transported signal value. However, different to the AMD devices, no variation among the two devices were observed.



(a) Connections Possible in ECP5 PLC2 Tile

(b) Characterized for *Open* Faults

(c) Characterized for *Input Antenna* Faults

(d) Characterized for *Input Antenna* Faults (Vertical Axis: Incident Wire Type)

Figure 23: Available and Characterized Connections in ECP5 PLC2 Tile for *Open* and *Input Antenna* Faults (Horizontal Axis: Destination Wire Type, Vertical Axis: Source Wire Type)

- Mismatches between Out_1 and `golden_wire` were recorded in 32 of 11594 bits tested for *Input Antenna* faults. This exclusively concerned connections where the erroneously connected additional wire (i.e., the "incident wire") was in the `GLOBAL` category.

  While `GLOBAL` wires `G_HPBX[08-15]00` are only available in multiplexers driving `CONTROL` and `CLOCK` signals, `G_HPBX[00-07]00` are each available as a source wire in two multiplexers driving `X1` wires. As interconnect multiplexers are modeled as two-stage decoded multiplexers in this work, two possibilities exist (one within both stages) to erroneously connect one of these wires in an *Input Antenna* fault, resulting in the 32 tested bits in this category.

  In all of these 32 cases, Out_1 mismatched `golden_wire` whenever Out_1 should transport a static "1" or the "toggle" pattern. Thus, an *Input Antenna* fault with a `GLOBAL` wire appears to cause a "stuck-at-0" fault.

  Also among these cases, no variation was observed between the two tested ECP5 devices.

Thus, not only is the interconnect found in Lattice's ECP5 FPGAs structured similarly to AMD's 7 series devices, but also behaves similarly in the presence of *Open* and *Input Antenna* faults. While fewer devices (two vs. four) and interconnect tiles (one vs. 17) were tested, Lattice ECP5 appears to exhibit less device-to-device variation. A partial explanation may be that ECP5 devices are 40 nm devices, while AMD's 7 Series are 28 nm devices. With this also comes a slightly higher core supply voltage: iCESugar-Pro uses 1.1 V while Digilent's BASYS3 supplies its AMD Artix 7 FPGA with a VCC_{INT} of 1.0 V. However, even though Lattice's iCE40 FPGAs are also based on 40 nm technology, and use a core supply voltage of 1.2 V, they experience significant device-to-device variation as discussed above. Thus, neither feature size nor core supply voltage are sufficient on their own to explain ECP5's low observed variation in fault injection results.

Conflict Faults

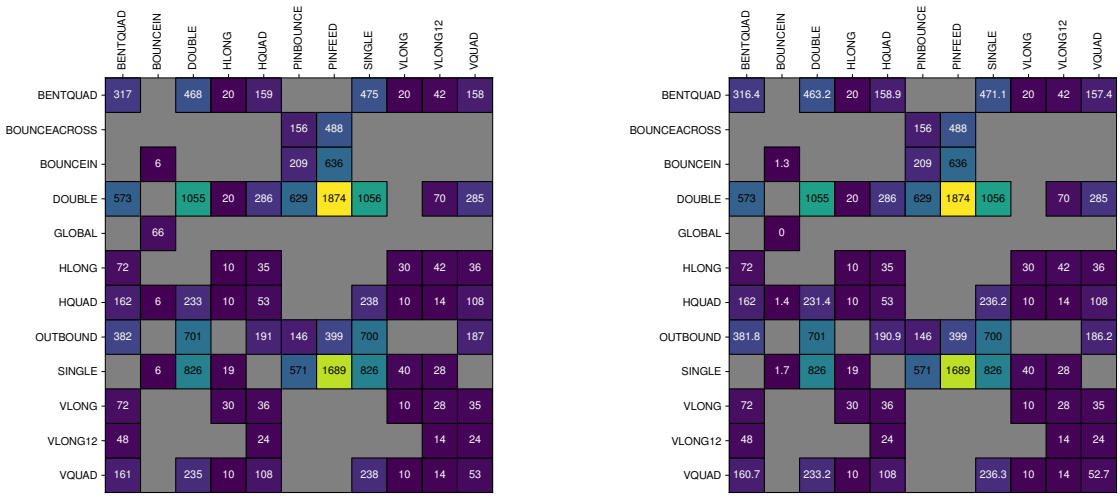
Following the characterization of *Input Antenna* faults, *Conflict* faults were characterized for the targeted AMD 7 Series, Lattice iCE40 and Lattice ECP5 devices. The results of this process are reported in the remainder of this section.

AMD 7 Series *Conflict* faults were evaluated in the interconnect tile `INT_L_X26Y70`. This tile was also evaluated in the experiments characterizing *Input Antenna* faults. Referring back to the implemented two-wire characterization circuit (see Figure 16), a total number of 17988 combinations of `test_wire_1` source, `test_wire_2` source, and the bit by which `test_wire_2`'s source is configured as an additional driver for `test_wire_1`'s destination wire have been tested. These experiments were carried out on twelve devices, including the same four previously used in the *Input Antenna* experiments. This set contained a mix of date- and lot codes: Four devices with date code *1445* and lot code *D5034186A*, one device with codes *1909*/*D5834810A*, four devices with codes *1913*/*D5851384A*, and three devices with codes *1913*/*D5851385A*. This diversity allows the identification of possible inter-device variations.

Figure 24a shows the 17988 tested combinations divided by the type of affected interconnect resource (i.e., the wire type for Out_1) on the horizontal axis vs. the type incident wire (i.e., In_2) on the vertical axis. As discussed in Section 4.3.1, depending on the actual circuit implementation of the interconnect resource, *Conflict* faults potentially influence both wires that are shorted together by a fault can be affected. Furthermore, the fault may only affect certain combinations of logical values (see Table 43). Figure 24b shows the most prominent observed effect, observed on average in 17889.8 cases among the twelve devices: In these cases, only Out_1 (i.e., the wire driven by the affected interconnect resource) was affected when carrying the logical value "1".

The relative shares of fault effects only affecting Out_1 is shown in Figure 25 organized similarly to Figure 24. In contrast, Figure 26 shows the share of critical fault combinations that (additionally) affect Out_2. In conjunction, this data allows the conclusion that in most fault combinations, only Out_1 is affected by the fault. Furthermore, in most cases, only the combination of Out_1 driven to "1" and the incident wire (in this case In_2 via `test_wire_2`) driven to "0" has an effect on the value at the output of the former. Only in rare cases other combinations are critical. An interesting rare case is that one fault was only observed as critical with the "toggle" input signal. This might constitute a fault where Out_1's timing is impacted severely enough in order to become visible to the mismatch checking circuit. The incident route-under-test, `test_wire_2` (via Out_2), is only seldomly affected by the short circuit introduced by the fault. This can be seen in Figure 26a (both Out_1 and Out_2 affected) and Figure 26b (only Out_2 affected).

The interconnect resources involved in the exceptional cases found on all tested devices (intersec-



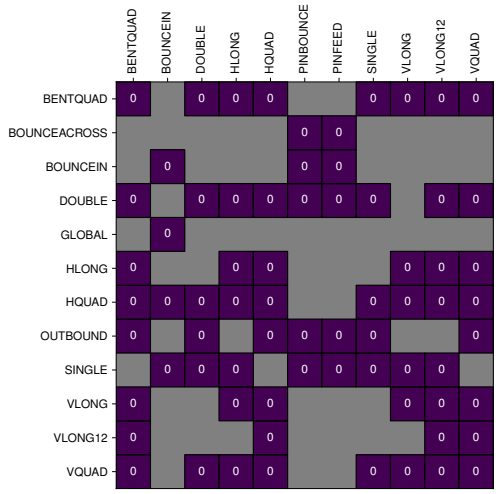
(a) Evaluated Combinations

(b) Faults only affecting "1" in Out_1

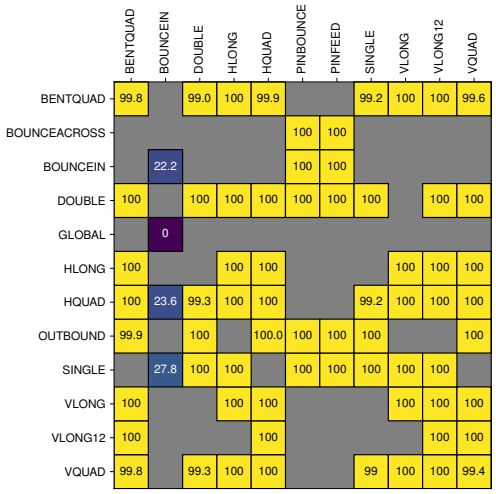
Figure 24: Evaluated Fault Combinations and Most Prominent Observed Effect for *Conflict* faults in twelve AMD Artix 7 xc7a35tcpg236-1 FPGAs by Fault Destination Wire type (Out_1; Horizontal Axis) and Incident Wire type (In_2; Vertical Axis)

tion) where also `test_wire_1` driven to "0" is affected are once again the `GFAN[0,1]` resources (i.e., wire type for Out_1 is `BOUNCEIN`), with In_1 driven by `GCLK_L[...]` (i.e., wire type `GLOBAL`) and the other wire by generic interconnect – `WW4END1` (`QUAD`) or `NR1END1` (`SINGLE`) or `BYP_BOUNCE1` (`BOUNCEIN`). The `GFAN[0,1]` resources are also involved in the cases where `test_wire_2` is affected by the fault, although these occur with any combination of sources for both wires, not only `GCLK[...]` signals. These observations are consistent with the ones described in [131, 218].

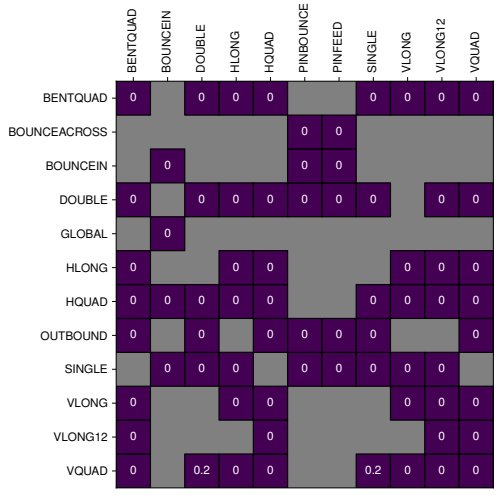
In summary, the potential for a *Conflict* fault to impact a design on an AMD 7 Series FPGA appears to be much higher than that of an *Input Antenna*-like fault. In almost all cases observed, a *Conflict* fault was able to impact `test_wire_1`, while *Input Antenna*-like faults only consistently affect their route-under-test (i.e., Out_1 wire segment) when the incident wire stems from a specific subset of the `LOGIC_OUTS(_L)[0-23]` wires that depends on the function tile interfacing to the considered interconnect tile. The analysis based on the transported logical values affected by the faults suggests that there may be no fundamental difference between an *Input Antenna* fault and a *Conflict* fault on AMD 7 Series FPGA. In a conflict between a high and a low value, the low value almost always "wins out" – showing wired-AND characteristics (see Table 43). The difference in criticality lies in the fact that unused wires (i.e., the incident wires *Input Antenna*-like faults) are in most cases driven with a static (weak) high value, and only in some cases (e.g., at the outputs of unused LUTs) with "0". If the incident wire is driven by another signal that is part of the design, the criticality of the individual *Conflict* fault will depend on the "compatibility" of the two involved signals in terms of the number of clock cycles where complementary logical values drive the two signals.



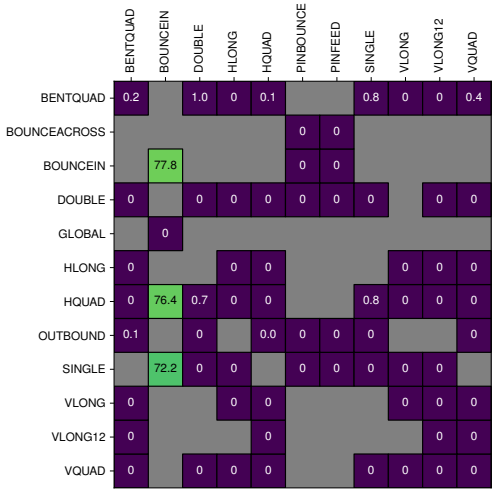
(a) Only “0” Affected



(b) Only “1” Affected

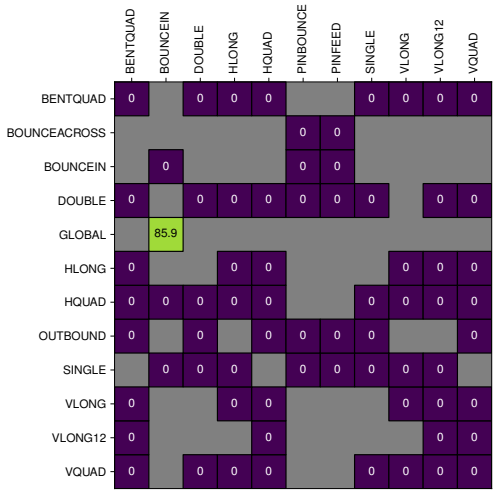


(c) Only “Toggle” Affected

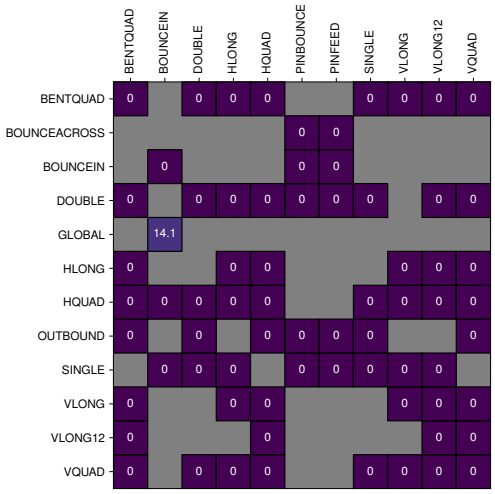


(d) Both “0” and “1” Affected

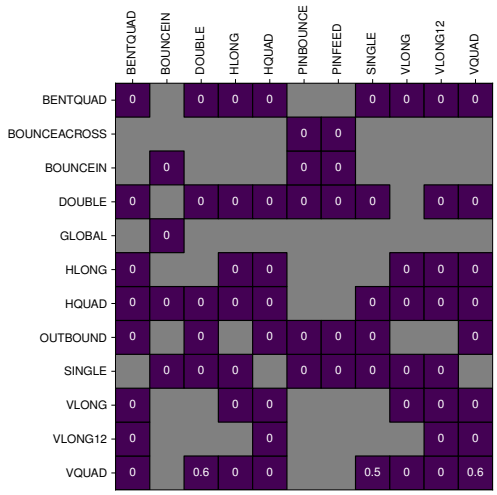
Figure 25: Share (%) of *Conflict* Faults in twelve AMD Artix 7 xc7a35tcpg236-1 FPGAs where only Out_1 is affected by Fault Destination Wire type (Out_1; Horizontal Axis) and Incident Wire type (In_2; Vertical Axis)



(a) Out_1 and Out_2 Affected



(b) Only Out_2 Affected



(c) No Effect

Figure 26: Share (%) of *Conflict* Faults in twelve AMD Artix 7 xc7a35tcpg236-1 not captured by Figure 25 by Fault Destination Wire type (Out_1; Horizontal Axis) and Incident Wire type (In_2; Vertical Axis)

Lattice iCE40 To preliminarily characterize the behavior of Lattice iCE40 FPGAs under *Conflict* faults, 4223 combinations of source wires (1317 of type `Span4`, 17 `Span12` and 2 `neigh_op`) with 880 `Span4` destination wires were tested by generating routing variations of the circuit shown in Figure 16.

Table 52: iCE40 *Conflict* Fault Characterization Summary

Tested Faults	Observed Critical on	
	$\geq$ 1 Device	All 10 Devices
4223	4158 (98.46%)	3644 (86.29%)

As can be seen in Table 52, *Conflict* faults behaved both more critical and more uniformly across the tested devices when compared to *Input Antenna* faults (see Table 51), as also observed on AMD Artix 7.

In most cases (66.6% of *Conflict* faults) the injected bit flip had an effect on the target route when it transported a logic "0" or the *toggle* signal uniformly on all 10 devices. In these cases, the erroneously connected driver would drive the target signal with "1", suggesting a predominant wired-OR (weak "0", strong "1") behavior, contrary to the *Conflict* effects observed on AMD Artix 7.

No significant differences between incident wire types (`Span4`, `Span12`, `neigh_op`) were observed. In general, only the target wires (i.e., Out_1) of the injected *Conflict* faults were affected. No "upstream" effect of the "incident" connection between In_2 and Out_1 back on the value transported via Out_2 (see Figure 16) was observed. In this regard, the results on Lattice iCE40 are similar to those on AMD Artix 7.

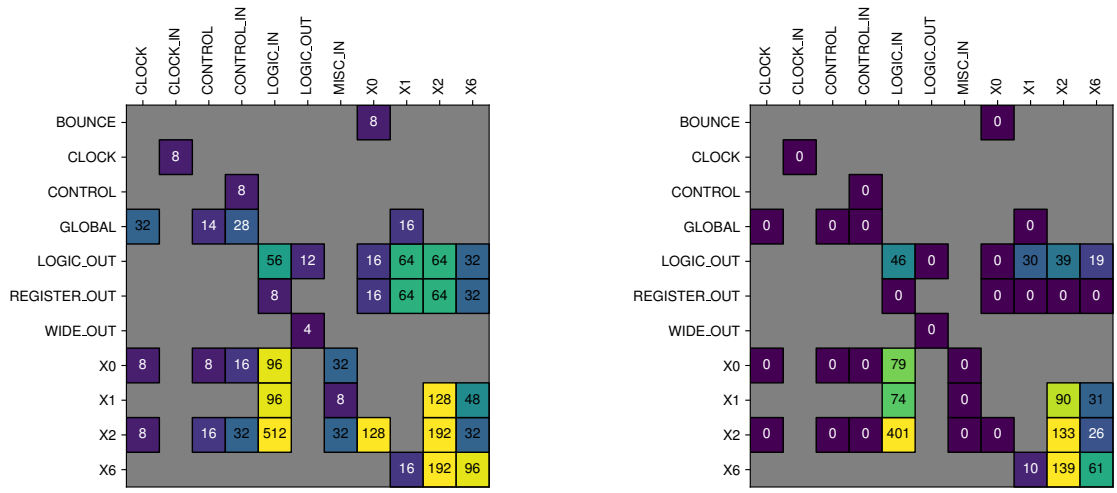
A criticality prediction approach for Lattice iCE40 FPGAs may thus consider all *Conflict* faults as potentially critical. When combining this static analysis with fault simulation, resources that transport "0" commonly during operation may be prioritized.

Lattice ECP5 In a similar way to the results on AMD Artix 7, Figure 27 summarizes the number of *Conflict* faults that are possible (see Figure 27a) and that were characterized (see Figure 27b) in the preliminary characterization experiments carried out in the context of this dissertation. All outputs of LUTs are summarized as `LOGIC_OUT`; in a similar way all flip-flop/register outputs (`REGISTER_OUT`), multipurpose inputs (`MISC_IN`), and wide-function outputs (`WIDE_OUT`) have been summarized. Furthermore, each slice in the ECP5 architecture can choose among multiple tile clock and flip-flop control signals (`CLOCK`, `CONTROL`) by forwarding them via a per-slice multiplexer, whose outputs were summarized as `CLOCK_IN` and `CONTROL_IN`. The general interconnect is present in the `X[0,1,2,6]` categories, while global wires are subsumed under the `GLOBAL` category. Finally, the `BOUNCE` category denotes that tile local (i.e., `X0`) wires can be used to "bounce" a signal to increase its intra-tile fanout.

It can be seen in Figure 27b that the 2003 characterized faults comprise a subset of the connections from (1) LUT outputs to the general interconnect, (2) LUT outputs to LUT inputs, (3) general interconnect to LUT inputs.

In all 2003 cases, on both tested devices, the injected conflict faults had similar outcomes:

- Only the value output by the interconnect resource affected by the fault (i.e., segment Out_1 in Figure 16) was ever affected by injected faults. The output of the second interconnect resource (i.e., Out_2) was never affected by a fault. Thus, similar to most cases observed on AMD 7 Series devices, *Conflict* faults appear to only affect the interconnect "downstream" of a fault.



(a) Possible Connections

(b) Characterized Connections

Figure 27: Available and Characterized *Conflict* Faults by Destination Wire Type (Out_1; horizontal axis) and Incident Wire Type (In_2; vertical axis) in ECP5 PLC2 Tile

- In every case, the value output via Out_1 mismatched the "golden wire" whenever it transported the "1" or "toggle" patterns, meaning that at the same time `test_wire_2` transported "0" (or an inverted toggle pattern). No mismatch occurred when `test_wire_1` transported the logic value "0" (meaning that `test_wire_2` transported "1" at that time). Thus, once again similar to AMD 7 Series, *Conflict* faults appear to manifest themselves as Wired AND (see Table 43b) between the two involved signals, only affecting the downstream interconnect resource.
- Similar to both AMD 7 Series and Lattice iCE40 devices, *Conflict* faults bear a larger potential to influence the correct functionality of the design, having observed critical effects in Out_1 in *every* tested conflict fault.
- In contrast to both AMD 7 Series and Lattice iCE40, neither variation among devices nor within the same device was observed. This might point towards a robust low-level interconnect implementation in ECP5 FPGA that is less vulnerable to PVT effects than the other evaluated devices.

This characterization provides insights into the behavior or *Conflict* faults on Lattice ECP5 devices, allowing the suggestion that all *Conflict* faults should be considered as potentially critical in a criticality prediction approach for these FPGAs. If this approach is to be combined with fault simulation, cases where a fault would affect a wire that transports "1" should be prioritized.

Nevertheless, a complete characterization including faults involving (1) flip-flop output wires, and (2) clock routing may be desirable to form more accurate predictions about the behavior of these devices under faults. To characterize the remaining faults, a different characterization circuit is needed that, e.g., allows the use of the output of a flip-flop as an *incident* wire.

4.3.3 Summary

This section presented a empirical characterization approach to identify the circuit-level consequences of interconnect bit flips, which is necessary in lieu of an accessible transistor-level schematic of the FPGA's interconnect resources. This approach improves upon existing methods

in literature [46, 131, 124] by facilitating the systematic analysis of these consequences, and by demonstrating the applicability to non-AMD FPGA devices. The approach consists of a benchmark circuit that is selectively placed and routed in order to include the interconnect resource to be tested in a "route under test" which is subsequently subjected to fault injection. By comparing the output value of this route to the expected fault-free output value, the circuit-level effects of *Open*, *Input Antenna*, and *Conflict* faults can be measured. For the characterization of *Conflict* faults, a second "route under test" is routed in a way to enable these kinds of faults in the resulting evaluation circuit.

Results for AMD Artix 7, Lattice iCE40, and Lattice ECP5 FPGA devices are provided. For AMD Artix 7, for which the most extensive characterization experiments were performed, the experiments revealed (1) the relative benignity of *Input Antenna* faults and the relative criticality of *Conflict* faults, (2) incident wire types responsible for the remaining critical *Input Antenna* faults, (3) Wired-AND as a consequence of most faults in both categories, and (4) hints towards inter-device variation of *Input Antenna* fault consequences. While *Input Antenna* faults exhibited small inter-device variation on AMD Artix 7, this variation was more extensive for interconnect faults on Lattice iCE40 FPGA. In these devices, the effects of *Input Antenna* faults greatly varied between devices, while they performed more uniformly for *Conflict* faults. Concerning interconnect fault consequences, Lattice ECP5 devices performed similarly to AMD Artix 7 devices with (1) both critical *Input Antenna* and *Conflict* faults acting as Wired-AND faults, and (2) only *Input Antenna* faults with select incident wires being critical, while all *Conflict* faults being observed to impact thre route under test. In contrast to both AMD Artix 7 and Lattice iCE40, no inter-device variation was identified in Lattice ECP5 devices. However, these observations may be impacted by the scope of experiments, with two ECP5 devices, but 16 Artix 7 and ten iCE40 having been evaluated.

The presented characterization results allow the formation of refined bitstream-based criticality prediction rules for AMD Artix 7, as most *Input Antenna* faults can be categorized as likely benign; the characterization results also highlight potentially malignant *Input Antenna* faults (see Section 4.5). In a similar way, a refined criticality prediction approach for Lattice ECP5 devices may be possible. Finally, the observed inter-device variation motivates further investigation into PVT-related effects when considering bitstream-level fault injection experiments (see Section 4.4) which may be integrated into criticality prediction as well.

4.4 Analysis of PVT Influence on Fault Injection Results

The results provided in Section 4.3.2 show that in AMD Artix 7 and Lattice iCE40, the results of injected *Input Antenna* faults (both device types) as well as *Conflict* faults (Lattice iCE40 only) varied from device to device[92]. These effects manifested themselves in varying logic-level effects of the injected faults on a route-under-test:

For example, in interconnect tiles next to CLB tiles in AMD 7 Series devices, only "incident wires" of the type `LOGIC_OUTS[8-15]` driven by unused LUTs were consistently able to disrupt the route-under-test when injecting *Input Antenna* faults. However, on some devices, sporadically, faults connecting general interconnect wires as "incident wires" were able to disrupt the route-under-test in the same way. Similar behavior was observed on the Lattice iCE40 devices. A cause for these device-to-device variations may be PVT effects, leading to slightly varied characteristics of interconnect wires, pass transistors or transmission gates, as well as buffers and half-latches in these devices.

Possibly related effects have been observed in literature: In [139], the authors report varying

[92] Conversely, no inter-device variation was observed in Lattice ECP5 FPGAs

criticality results in six subsequent bitstream-level fault injection runs on the same design carried out on the same device. The authors of [121] design post-radiation validation experiments so that each critical fault observed under radiation is aimed to be replicated using bitstream-level fault injection thrice on three different devices. While these works implicitly acknowledge the possibility of "random" effects – both variations within the same device across multiple repeated fault injection runs and variations between devices – they do not further investigate the extent and causes of these effects. On the other hand, multiple proposed approaches such as [141, 140] suggest that exhaustive fault injection experiments may be sped up by splitting up the bits to be tested across multiple devices – i.e., as is the title of [140], "Injecting FPGA Configuration Faults in Parallel". Thus, it is implicitly assumed by [141, 140] that the outcome of a fault injection experiment is independent of the device the experiment is conducted on. Without considering possible device-to-device variations, this might lead to misleading fault injection results if certain bits are tested on an especially error-prone or, worse, exceptionally fault-tolerant device. Furthermore, none of these works mention the operating conditions under which fault injection experiments are conducted, thereby assuming that criticality is independent of chip temperature and supply voltage.

This dissertation systematically investigates the effects and extent of inter-device variation, as well as the impact of temperature conditions and supply voltage on fault injection results. To that end, in Section 4.4.1, a synthetic benchmark FPGA design is evaluated on 16 AMD Artix 7 and 10 Lattice iCE40 devices and in four temperature steps. The fault injection results are compared, and fault categories especially vulnerable to inter-device and temperature-related criticality variation are identified. As an extension, Section 4.4.2 investigates the effects of environment (and chip) temperature on *Open* faults in AMD Artix 7 devices. Furthermore, Section 4.4.3 analyzes supply voltage's effects on criticality, intra-chip, and inter-chip variation when considering *Input Antenna* faults in AMD Artix 7 devices.

4.4.1 Temperature Effects on Systematic Fault Injection Results

To put the (implicit) assumptions of previous works discussed above [141, 140] to a test, the following experiment to be performed for a given Design under Test (DUT) and target FPGA device type is proposed:

1. **Bitstream Analysis:** As a first step, analyze the composition of the DUT's bitstream into functional categories, e.g., bits configuring logic (LUTs, carry blocks, sequential logic) and bits configuring interconnect. This requires detailed knowledge of the target's configuration structure which is publicly available for select FPGA families as discussed in Section 4.2.2.
2. **Configuration-Level Fault Injection:** Perform systematic fault injection experiments on multiple devices, adhering to the following constraints:
    - To minimize the influence of manufacturing variations, the target devices should be as similar as possible (i.e., in order of importance, same family, model, speed grade, temperature grade, date- and lot code).
    - As seen, for example, in [139], random processes appear to exist in FPGA devices that cause different sets of critical bits to be discovered in subsequent fault injection experiments on the same device. Thus, to reach higher confidence that all critical bits on a single device have been discovered, experiments per bit are performed for a set number of repetitions.
    - Variations between semiconductor devices may be amplified by differences in chip temperature. Thus, experiments are conducted under controlled temperature conditions,

and chip temperature should be monitored. As these variations may be temperature-related, tests are conducted at different temperatures.

3. **Intersection-Union Analysis:** To identify differences between devices, analyze the union and intersection sets of critical bits over all devices on which the experiments have been performed. In the ideal case – all bits to be found critical on any one device are also critical on all other devices due to a small impact of PVT effects on fault injection results – these sets will be equal. Otherwise, the ratio between intersection and union set size (i.e., *Intersection-over-Union (IoU)*) will indicate differences between the devices.

4. **Device- and Temperature-based Analysis:** To learn more about the device variations and temperature behavior, further analysis is performed on bits that are critical only on a subset of devices or only in specific temperature ranges. This may, e.g., highlight functional groups that are overrepresented in the respective analyses compared to their actual population in the bitstream.

Experimental Setup

Following the procedure described above, experiments have been conducted for AMD 7 Series and Lattice iCE40 devices. This section describes the experimental setup, including a synthetic DUT and the strategy for performing fault injection.

Target Devices and Toolchains: The experiments and analyses described above were performed on a set of 16 xc7a35tcpg236-1 AMD Artix 7 devices aboard Digilent BASYS-3 development boards[93], and a set of Lattice iCE40-up5k devices on 10 iCEBreaker[94] development boards. Care was taken in selecting the 16 employed AMD devices to choose devices with matching date-, lot-, and revision codes. The device markings of all 16 AMD FPGA indicate date code *1445*, lot code *D5034186A*, and revision code *ABX*[95]. Ten iCE40 devices with identical markings[96], likely indicating identical date and lot were used. This evaluation excludes Intel/Altera devices due to the lack of publicly available bitstream documentation. AMD Vivado 2017.2 was used to synthesize, place, route, and generate bitstreams for the Artix 7 platform. For iCE40, a fully open-source RTL-to-bitstream toolflow, consisting of Yosys, nextpnr [205], and the IceStorm toolbox [219] was used.

Design under Test: A synthetic test design, shown in Figure 28, was implemented as a target for systematic fault injection experiments. It consists of an 8-bit Linear Feedback Shift Register (LFSR), an 8-bit counter, both controlled by a state machine. The design was chosen as its small size allows for comparatively fast complete systematic fault injection runs while containing essential components found in almost all FPGA designs. It thus allows to judge the effect of device and temperature on fault injection results targeting bits configuring combinational logic, sequential logic, and interconnect. When the state machine receives a *start* strobe, it loads both the LFSR and the counter with initial values. The counter counts down a pre-configured number of clock cycles and causes the state machine to raise a *done* strobe once it has expired. The state of the LFSR

[93] See `https://web.archive.org/web/20241227120346/https://digilent.com/reference/programmable-logic/basys-3/start`, accessed 2026-01-05

[94] See `https://web.archive.org/web/20250126163023/https://github.com/icebreaker-fpga/icebreaker`, accessed 2026-01-05

[95] Device markings: `XC7A35T` ↩ `CPG236ABX1445` ↩ `D5034186A` ↩ `1C` ↩ `T8A817.R1-00`

[96] Device markings: `ICE40UP5` ↩ `SG48I` ↩ `A9011RC9` ↩ `ED7K78YG11`

depends on its initialization value – hard-coded in this design – and the number of clock cycles since initialization. It is forwarded as an output on the DUT's top level.

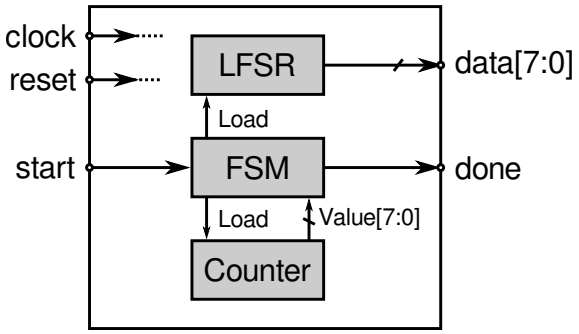


Figure 28: Synthetic Test Design for Systematic Fault Injection Experiments

Correct behavior of this DUT is thus characterized by both a correct latency between the assertion of the *start* and *done* strobes, as well as the correct LFSR output value when *done* is asserted. Each bit that, when flipped, causes the design to violate any of these properties is classified as a critical bit.

Analysis of the DUT's Bitstream: The synthesized DUT consists of 19 `FD{R|S}E` flip-flops and 47 LUTs when mapped to Artix 7. Mapped to iCE40, it consumes 19 `SB_DFF{ESR|SR|SS}` flip-flops, 26 `SB_LUT4`, and six `SB_CARRY` cells. Xilinx Vivado's bitstream generation tool reports 7539 *Essential* configuration bits for the implemented circuit. An "essential bits" feature is unavailable within the open-source iCE40 toolchain used for this work. The toolchain provides the program `icebox_explain` that describes the bits configured actively by the bitstream. All logic bits that may influence configured logic, as well as all routing bits that may add an unwanted driver to already used wires, have been added to the bits output by this tool. This allows generating a metric similar to Xilinx's "essential bits", resulting in 1839 configuration bits.

The configuration bits for both FPGAs have been classified into three categories by their functionality. The composition for Artix 7 is shown in Table 53[97], while Table 54 shows the data for iCE40. Furthermore, the tables show the interconnect-related configuration bits distinguished by the consequences of bit flips:

- Applying the interconnect fault model for AMD 7 Series devices described in Section 4.2.2, an interconnect bit flip in these devices may (1) split a wire into two disconnected parts (an *Open* fault), (2) force an additional driver on a wire (either a *Conflict* or *Input Antenna* fault), or (3) cause other effects such as creating new partial connections (i.e., *Other* faults). Particular interconnect resources in AMD 7 Series devices[98] appear to use inverted configuration bits in their second stage (see Table 36). Thus, the bare bit pattern is inconsistent with a two-stage decoded pass-transistor or transmission-gate-based multiplexer. Not applying these assumptions of inverting a subset of the second-stage configuration bits – and thus assuming an invalid bit pattern in the second stage disconnects the route entirely (i.e., an *Open* fault) – yields the interpretation shown in the columns entitled "Original Interpretation" in Table 53. Adding this assumption to the interpretation of the bitstream (see Table 36) yields the "Modified Interpretation" shown in Table 53.

[97] All bits in `CLBL[LM]_[LR]` tiles have been classified as "logic", despite a small amount of routing resources being present in CLB tiles (e.g., FF input MUXes)

[98] Driving `IMUX[...]`, `BYP_ALT[...]`, `FAN_ALT[...]`, `CLK`, `CTRL`, and `GFAN` wires

- On iCE40, applying the model for these devices described in Section 4.2.2, a single bit flip may (1) disconnect a driver that should be connected (an *Open* fault), (2) force an additional driver on a wire (*Conflict* or *Input Antenna* fault), or (3) select a different, possibly undriven, source wire by altering a multiplexer's select bits (i.e., a *Bridge* fault).

Due to the assumed electrical similarity of *Conflict* or *Input Antenna* faults, these have been summarized into one category in this analysis.

Table 53: Composition of Essential Configuration Bits in DUT on AMD Artix 7

Configuration Bit Category	Essential Bits			
	Original Interpretation		Modified Interpretation	
	Absolute	Relative	Absolute	Relative
Clock Routing & Other	131	1.7%	131	1.7%
Logic	2952	39.2%	2952	39.2%
Interconnect	4456	59.1%	4456	59.1%
Open	1352	17.9%	908	12.0%
Input Antenna/Conflict	2802	37.2%	3246	43.1%
Other	302	4.0%	302	4.0%
Sum	7539	100%	7539	100%

Table 54: Composition of Essential Configuration Bits in DUT on iCE40

Configuration Bit Category	Essential Bits	Relative
Global Routing	32	1.74%
Logic	767	41.71%
Interconnect	1040	56.55%
Open	190	10.33%
Input Antenna/Conflict	180	9.79%
Bridging	670	36.43%
Sum	1839	100.00%

Fault Injection Platform & Constraints: The DUT described above is instantiated in a fault injection platform that controls DUT execution and checks results. The actual implementation of the fault injection platform and process differ between the two development boards used:

On Artix 7, this platform can take advantage of the ICAP and perform fault injection at runtime. The platform uses a MicroBlaze CPU and AMD AXI-HWICAP partial reconfiguration controller. The entire design runs at 50 MHz clock frequency, with a worst-case setup slack of 6.3 ns reported by Vivado. The bitstream containing the DUT and the injection platform is written to the configuration flash IC on the BASYS3 board, allowing for fast recovery of the entire fault-free bitstream.

The platform implemented on the iCEBreaker board does not perform fault injection itself, as iCE40 lacks the partial/dynamic reconfiguration feature of Artix 7. Here, the fault injection is implemented by generating bitstreams offline on the host that differ in the state of one configuration bit from a fault-free bitstream file. The design implemented on the iCEBreaker board runs at 50 MHz clock frequency with a worst-case setup slack of 2.4 ns reported by NextPNR.

The DUT and the injector/controller hardware are floorplanned into different rows of clock regions (Artix 7) or tiles (iCE40) to minimize the influence of injected faults on the injector itself.

Fault Injection Sequence: Faults are injected *before* the DUT starts its operation. Thus, injected faults act as single permanent faults. Temporal information (i.e., especially critical operating phases) and effects that make only certain combinations of faults critical are thus not considered. Each fault injection experiment performs the following steps:

1. **Configuration:** On Artix 7, the FPGA boots from configuration flash and notifies the controlling PC that it is ready. In contrast, on iCE40, the PC configures the FPGA via IEEE 1149.1 Boundary Scan ("JTAG") with a faulty bitstream.
2. **Fault Injection:** On Artix 7 only, the PC requests fault injection into a particular configuration bit, identified by its frame address, word index, and bit position. The fault injection controller resets the DUT, and injects a fault at the requested location. Fault injection happens while the DUT's clock is stopped using a global clock buffer, as it requires readback, modification, and update of an entire configuration frame. On iCE40, fault injection is done beforehand by generating a faulty bitstream on the host PC.
3. **Reset & Start DUT:** The DUT is reset and started by asserting its *start* strobe
4. **Check Results:** Once the DUT completes its run (or the timeout of twice the correct latency expires) the controller checks the correctness of latency and LFSR output. The result of this check is reported back to the host.
5. **Reboot:** On Artix 7 only, the host instructs the FPGA to reload its configuration from flash to ensure a fault-free state for the next run.

Fault injection experiments were repeated eight times per "essential" configuration bit, surpassing the six repetitions proposed in [139]. A configuration bit is counted as critical if, during *any one* of these eight runs, the controller identified either latency, LFSR output, or both as incorrect.

Temperature Environment & Monitoring: To ensure controlled temperature conditions, the tests were performed in a *Revolutionary Science RS-IF-203 incubator/fridge* as a temperature chamber. This temperature chamber also provides the possibility to perform systematic temperature tests. While the device can hold temperatures between 5 °C and 65 °C, cooling performance below room temperature with a heat source on the inside is limited. Thus, temperature experiments were conducted under a chamber temperature range of 20 °C to 50 °C. On Artix 7, chip temperature was monitored using the *XADC* hard IP, which is not available in iCE40 devices.

Results

The fault injection experiments outlined above were conducted at four temperature steps, characterized by the set temperature of the RS-IF-203. As a measure of basic fault injection quality as done in [139], the average share of critical bits present in all eight repetitions per bit among all devices, as well as the standard deviation, are shown in Table 55.

Table 55: Average share of critical bits present in all eight injections per bit

FPGA	Temperature Step			
	20 °C	30 °C	40 °C	50 °C
AMD Artix 7	99.2% ± 0.3%	99.3% ± 0.4%	99.6% ± 0.2%	99.7% ± 0.2%
Lattice iCE40	95.9% ± 0.2%	96.0% ± 0.3%	96.0% ± 0.3%	95.7% ± 0.3%

Table 55 shows that between 95.7% (iCE40) and $>$ 99.7% (Artix 7) of critical bits appear in all eight injections per targeted bit. These bits would have been discovered by a single fault injection run per bit with a high probability. By performing seven additional runs per bit, the remaining critical bits were discovered. The results of these experiments are provided for Artix 7 in Table 56 and for iCE40 in Table 57 in the form of the union set and intersection set of critical bits (i.e., bits critical on *any* vs. *all* devices, respectively) for the four temperature steps.

Table 56: Sets of Critical Bits in 16 AMD Artix 7 FPGAs and four Temperature Steps

Union Set								
Configuration Bit Category	Original Interpretation				Modified Interpretation			
	Temperature Step				Temperature Step			
	20 °C	30 °C	40 °C	50 °C	20 °C	30 °C	40 °C	50 °C
Clock Routing & Other	26	27	27	27	26	27	27	27
Logic	386	383	379	380	386	383	379	380
Interconnect	851	799	717	662	851	799	717	662
Open	612	560	480	425	587	535	455	400
Input Antenna/Conflict	157	157	155	155	182	182	180	180
Other	82	82	82	82	82	82	82	82
Sum	1263	1209	1123	1069	1263	1209	1123	1069
Intersection Set								
Configuration Bit Category	Original Interpretation				Modified Interpretation			
	Temperature Step				Temperature Step			
	20 °C	30 °C	40 °C	50 °C	20 °C	30 °C	40 °C	50 °C
Clock Routing & Other	25	25	25	25	25	25	25	25
Logic	370	375	371	371	370	375	371	371
Interconnect	568	575	584	593	568	575	584	593
Open	361	366	378	388	336	341	353	363
Input Antenna/Conflict	154	151	150	148	179	176	175	173
Other	53	58	56	57	53	58	56	57
Sum	963	975	980	989	963	975	980	989

Table 57: Sets of Critical Bits in 10 iCE40 FPGAs and four Temperature Steps

Union Set				
Configuration Bit Category	Temperature Step			
	20 °C	30 °C	40 °C	50 °C
Global	4	4	4	4
Logic	182	182	181	182
Interconnect	769	770	768	770
Open	163	163	163	163
Input Antenna/Conflict	29	29	29	31
Bridge	577	578	576	576
Sum	955	956	953	956
Intersection Set				
Configuration Bit Category	Temperature Step			
	20 °C	30 °C	40 °C	50 °C
Global	4	4	4	4
Logic	177	178	178	177
Interconnect	660	668	670	667
Open	137	137	137	134
Input Antenna/Conflict	18	18	18	18
Bridge	505	513	515	515
Sum	841	850	852	848

Both on Artix 7 and iCE40, the results show that there is considerable difference between the union and intersection set sizes, indicating that different subsets of tested configurations are critical on each device. Considering the lowest temperature step, of the 1263 bits critical on any device, only 963 (or 76.2%) are critical on all 16 tested Artix 7 devices. On iCE40, 841 of 955 bits (or 88.1%) are critical on all devices at the same temperature step. These initial results confirm that PVT-related device-dependence of systematic fault injection results is possible.

Additional parallels between Artix 7 and iCE40 can be found when comparing the magnitude of differences between intersection and union sets across configuration bit categories. At the lowest temperature step on both device types, the criticality of configuration bits associated with clock routing, global configuration, and logic (tiles) are relatively consistent from device to device compared to interconnect-associated bits. For example, on Artix 7 95.9% of bits associated with logic (tiles) critical on any device are critical on all devices (370 of 386), but only 66.7% of interconnect-associated bits (568 of 851). This effect can also be seen on iCE40: 97.3% of logic-associated bits critical on any device are critical on all devices (177 of 182), but only 85.8% of interconnect-associated bits (660 of 769). Note that on AMD Artix 7, logic tiles contain a small amount of intra-tile routing that was – for the extent of this analysis – still categorized as "logic"-related bits.

In addition, the results on Artix 7 are temperature-dependent: When the experiments are repeated at 30 °C, 40 °C, and 50 °C environment temperature, it can be seen that the gap between union and intersection set starts to close, i.e., the devices start behaving more uniformly. This mainly affects the bits found in the "Interconnect" category, where in the highest temperature step, 89.6% of bits are critical on all devices (593 of 662). This especially concerns bits that cause *Open* faults. In contrast, this effect was not observed in the fault injection results for iCE40. There appears to be some fluctuation among both intersection and union set sizes between temperature steps, but not to the extent as visible on Artix 7.

Table 58 provides a more in-depth analysis of the temperature-dependent critical bits on Artix 7. This table shows the union set sizes for bits critical in all observed combinations of temperature steps. The bits were first divided by tile function, while the bits found in interconnect tiles were further analyzed by the consequence of a bit flip (as in Table 56). While there are some bits that *become* critical with higher temperature (shown in the first two rows), and there is a single bit that behaves somewhat erratically (critical in the 20 °C, 30 °C, and 50 °C steps, but not in the 40 °C step), the vast majority of temperature-dependent bits *cease* to be critical with higher temperature (shown in the center three rows). The CLB bits have been found to be related with the `AFFMUX` and `[ABC]OUTMUX`, i.e., intra-slice routing resources, while the bits subsumed under "other" are found in `CLK_BUFG_REBUF` and `CLK_HROW_TOP_R` tiles and are assumed to be clock routing-related. Not only are most of the temperature-related bits found in the interconnect, also most temperature-variable bits belong to the *Open* category. Finally, the row titled "Sum" shows the total number of bits observed critical across all temperature steps and devices.

On iCE40, based on the results shown in Table 57, the (unlikely) possibility remained that equal numbers of bits *become* and *cease* to be critical with changing temperature. As a consequence, the analysis described above was also performed on the results obtained on iCE40, shown in Table 59. There are a few critical bits that are unique to almost any combination of temperature steps, suggesting that no significant temperature dependency exists in the evaluated range on iCE40. Almost the entirety of bits with varying criticality are interconnect bits.

Table 58: Critical Bits per Temperature Step on any one of 16 Artix 7 Devices

Original Interpretation									
Temperature Step				Sum	Interconnect			Logic	Other
20 °C	30 °C	40 °C	50 °C		IA/Conflict	Open	Other		
●	●	●	●	1055	155	415	82	378	25
○	○	○	●	2	0	0	0	1	1
○	●	●	●	11	0	9	0	1	1
●	○	○	○	65	0	61	0	4	0
●	●	○	○	85	2	79	0	4	0
●	●	●	○	57	0	56	0	0	1
●	●	○	●	1	0	1	0	0	0
Sum				1276	157	621	82	388	28
Modified Interpretation									
Temperature Step				Sum	Interconnect			Logic	Other
20 °C	30 °C	40 °C	50 °C		IA/Conflict	Open	Other		
●	●	●	●	1055	180	390	82	378	25
○	○	○	●	2	0	0	0	1	1
○	●	●	●	11	0	9	0	1	1
●	○	○	○	65	0	61	0	4	0
●	●	○	○	85	2	79	0	4	0
●	●	●	○	57	0	56	0	0	1
●	●	○	●	1	0	1	0	0	0
Sum				1276	182	596	82	388	28

Table 59: Critical Bits per Temperature Step on any iCE40 Device

Temperature Step				Sum	Interconnect			Logic	Global
20 °C	30 °C	40 °C	50 °C		Open	IA/Conflict	Bridge		
●	●	●	●	960	163	32	579	182	4
○	○	○	●	2	0	1	1	0	0
○	○	●	●	3	0	2	1	0	0
○	●	●	●	5	0	3	2	0	0
●	○	○	○	1	0	1	0	0	0
●	●	○	○	3	0	1	2	0	0
●	●	●	○	4	0	1	3	0	0
6 other combinations				18	0	10	7	1	0
Sum				996	163	51	595	183	4

Tables 60 and 61 show the per-device criticality and the average number of critical bits across all devices with its absolute and relative standard deviations. The small IoU coefficient indicates that *different* bits are critical across the tested devices. However, the *number* of critical bits is relatively consistent across devices. For AMD Artix 7, the standard deviation falls below ±10 bits (approx. 1% of the average) in the temperature steps above 30 °C.

Table 60: Critical Bits per Device in 16 AMD Artix 7 FPGAs and four Temperature Steps

Description	Temperature Step			
	20 °C	30 °C	40 °C	50 °C
Device 1	1161	1086	1045	1037
Device 2	1101	1042	1038	1038
Device 3	1120	1055	1036	1039
Device 4	1136	1069	1033	1035
Device 5	1129	1060	1040	1036
Device 6	1108	1044	1025	1028
Device 7	1114	1056	1041	1037
Device 8	1149	1057	1033	1032
Device 9	1125	1053	1039	1036
Device 10	1123	1053	1032	1035
Device 11	1141	1058	1033	1035
Device 12	1152	1086	1033	1034
Device 13	1118	1067	1037	1036
Device 14	1126	1062	1031	1032
Device 15	1153	1076	1027	1021
Device 16	1152	1090	1045	1039
Intersection Set	963	975	980	989
Union Set	1263	1209	1123	1069
IoU	76.2%	80.6%	87.3%	92.5%
Average Std. Deviation	1131.8 ± 18.0 (1.6%)	1063.4 ± 14.6 (1.4%)	1035.5 ± 5.7 (0.6%)	1034.4 ± 4.5 (0.4%)

Table 61: Critical Bits per Device in 10 Lattice iCE40 FPGAs and four Temperature Steps

Description	Temperature Step			
	20 °C	30 °C	40 °C	50 °C
Device 1	908	911	913	910
Device 2	912	913	913	915
Device 3	891	898	899	896
Device 4	908	913	913	914
Device 5	911	908	902	902
Device 6	903	904	903	907
Device 7	909	907	907	910
Device 8	899	897	897	901
Device 9	907	908	911	910
Device 10	898	900	899	902
Intersection Set	841	850	852	848
Union Set	955	956	953	956
IoU	88.1%	88.9%	89.4%	88.7%
Average Std. Deviation	904.6 ± 6.7 (0.7%)	905.9 ± 5.9 (0.7%)	905.7 ± 6.5 (0.7%)	906.7 ± 6.2 (0.7%)

Finally, Figures 29 and 30 show the growth of the union set of critical bits for each temperature step as systematic fault injection is conducted on a growing number of devices (after initially testing on the first board). For both tested FPGA types, 1000 random sequences of adding board results (from the $16!$ and $10!$ possible sequences, respectively) were selected. On Artix 7, this shows an explicit temperature dependency once again, while on iCE40, the results are similar over the entire temperature range. The horizontal line in both figures indicates the 1% of the respective union set size of critical bits over all devices at each temperature step.

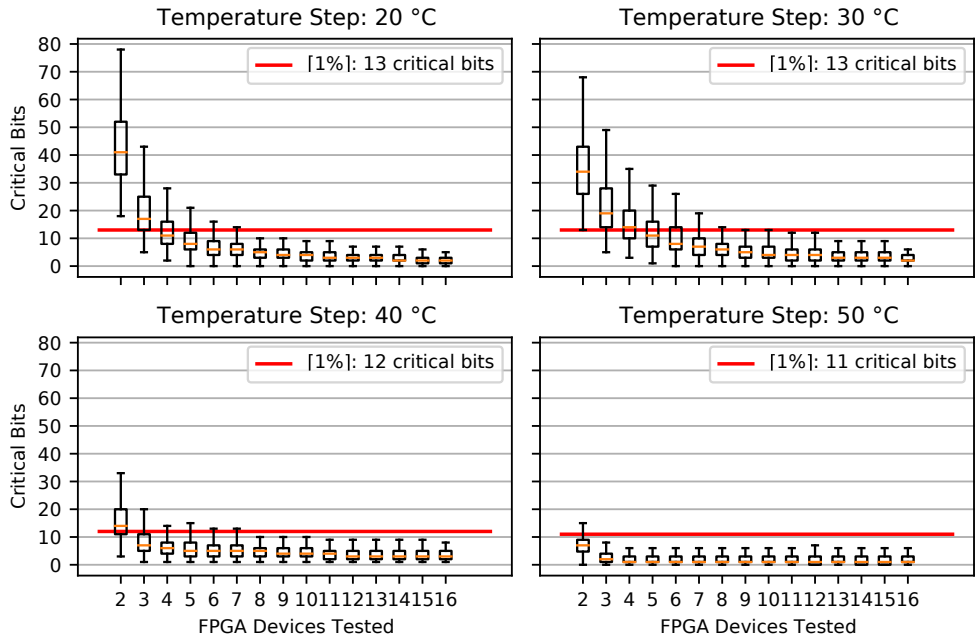


Figure 29: Additional identified critical bits vs. number of tested Artix 7 devices after first device

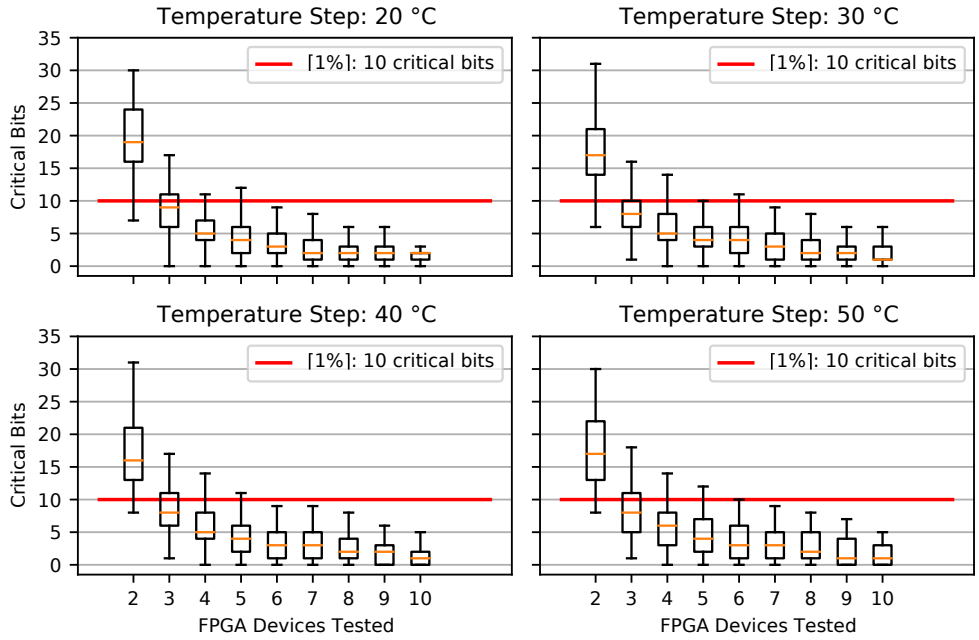


Figure 30: Additional identified critical bits vs. number of tested iCE40 devices after first device

Discussion

The results shown in the previous section show significant device dependence of systematic fault injection results on both evaluated FPGA families. In an evaluation using a synthetic DUT consisting of an LFSR, a counter, and an FSM, this variation is mainly found in interconnect-related bits: Predominantly *Open* faults in 16 AMD Artix 7 devices and both *Input Antenna/Conflict* as well as *Bridge* faults in ten Lattice iCE40 devices are concerned. Furthermore, on AMD Artix 7, significant variation in the total count of critical bits was observed at four different temperature steps (20 °C, 30 °C, 40 °C, and 50 °C). Here, once again, *Open* faults were concerned, warranting a closer investigation of these faults on AMD Artix 7 (see Section 4.4.2).

In general, the results in this section indicate that the effects of PVT variations should be considered in the design of future systematic fault injection experiments by evaluating the effects of individual bit flips on multiple devices and under controlled environmental (temperature, voltage) conditions. However, while the set sizes of critical bits vary across devices, the absolute *numbers* of critical bits appear to be relatively consistent across devices for AMD Artix 7. Thus, absolute numbers of critical bits retrieved by fault injection experiments on one device may be generalizable to other (similar) devices of the same type.

4.4.2 Temperature Influence on Open Faults in AMD 7 Series FPGAs

On AMD Artix 7, considerable variation among devices and temperature steps was observed for *Open* interconnect faults. Depending on the circuit-level implementation of interconnect resources, one possible consequence of an *Open* fault is a stuck-at condition: The disconnected wire segment is driven to a stable logic level once disconnected, for example, by a level-restoring buffer (see element “A” in Figure 15) or a NAND pulldown structure (see element “F” in Figure 15). Another possibility is that the wire segment is left floating, thus resulting in an essentially random interpretation of the signal value by the fan-out elements.

For AMD’s 7 Series, several earlier works identified stuck-at-1 as a common consequence of *Open* faults [124, 46]. However, inter-device variation of the criticality of these faults, as well as variation of the criticality of these faults with environment temperature (and thus chip temperature), indicates that this explanation does not fully suffice. Thus, this dissertation performs characterization experiments to identify possible causes for this variability qualitatively. The underlying assumption of these experiments is that while after a given time, *Open* faults behave as stuck-at-1 faults as observed by other works [124, 46], there exists a device- and temperature-dependent period within which the fault behaves otherwise (e.g., as a stuck-at-0 fault).

Experimental Setup

To investigate the assumptions discussed above, the experimental design shown in Figure 31 was implemented for an AMD Artix 7 xc7a35tcpg236-1 FPGA (i.e., the same type as used for the initial temperature- and device-dependency investigation described in Section 4.4.1).

Its essential components are (1) a fault injection, control, and monitoring block, which is comprised of a SoC based on a MicroBlaze CPU and interface logic. This system communicates with the outside world via Universal Asynchronous Receiver/Transmitter (UART), has access to the device configuration via ICAP, and interfaces with (2) the DUT portion of the design. The DUT consists of an input buffer LUT, a route under test that is routed via several interconnect resources, and an output buffer LUT. The controller drives this route to “0” or “1” via the `di` signal, performs a fault injection causing an *Open* fault and records the behavior of the `do` signal. The circuit is thus similar

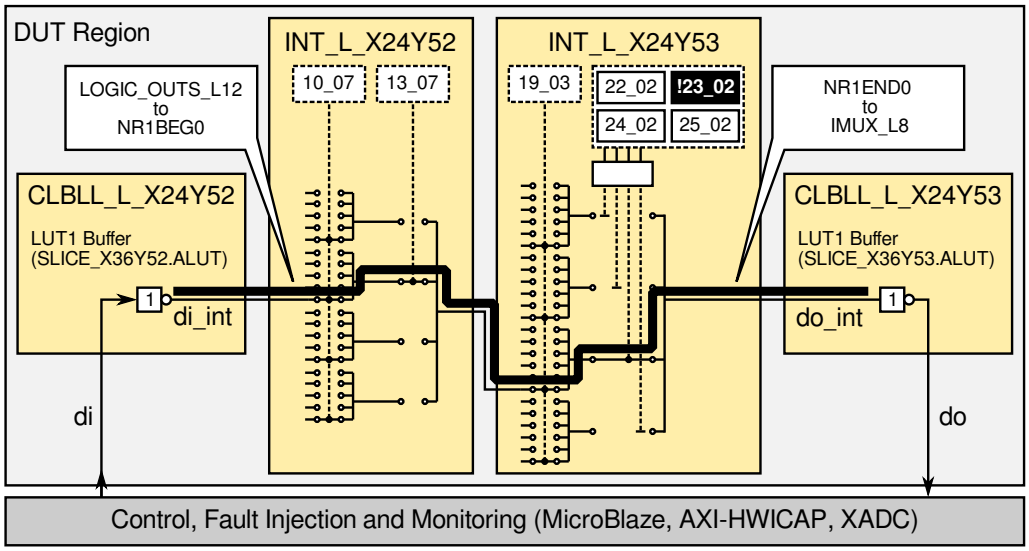


Figure 31: Experimental Setup for Investigation of *Open* Faults on AMD Artix 7

to the one shown in Figure 16. However, the interface logic and result analysis portion of the design are tailored towards the problem at hand: The `do` signal is synchronized to the SoC's clock via a two-stage synchronizer to limit the effects of setup/hold violations caused by the asynchronous nature of the assumed pull-up logic. For each transition of the `do` signal, a timestamp in terms of clock cycles after the injection of the fault is recorded in a FIFO buffer. The design is clocked with 50 MHz, and the DUT part floorplanned into a physical region in the FPGA where fault injection does not interfere with the controller. A snapshot of the final value of the signal under test is taken after the timestamp counter reaches the value 2^{20}, i.e., 1048576. As the MicroBlaze processor adds some additional processing delay, it is taken at 1048754 clock cycles (i.e., 20.97 ms at 50 MHz) after injection start.

This design allows to determine how the route under test reacts to disconnect faults, as shown in Figure 32: It is assumed that at a certain point in time $t_{\text{transition}}$ after an *Open* fault is injected via ICAP (at time t_{inject}), the newly disconnected wire segment will perform a transition. As earlier works [46] indicated, a "stuck-at-1" consequence is likely. Thus, a transition may only appear in signals that were driven to "0" before disconnection. The transition time, and thus the duration it takes for a signal to transition after fault injection, $T_{\text{disconnect}}$, is assumed to vary with environment temperature and device,

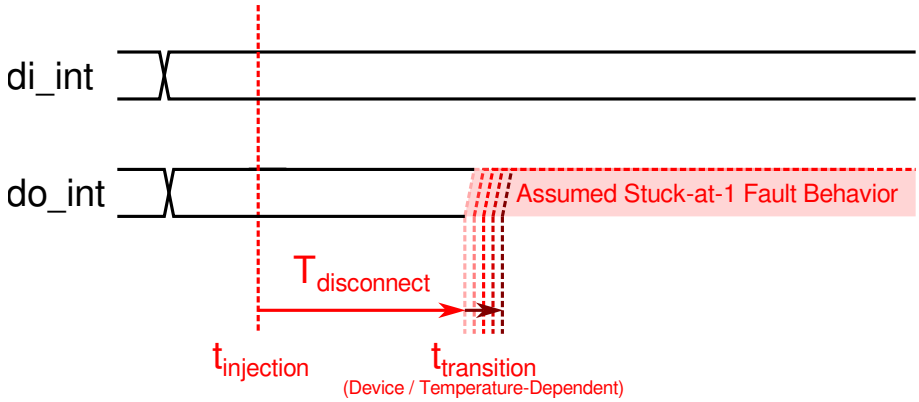


Figure 32: Behavior of Output of Route-under-Test after Injection of an *Open* Fault and its Correction

Using placement constraints, the input buffer is placed in LUT "A" of Slice 0 in tile CLBLL_L_X24Y52 and the output buffer in LUT "A" of Slice 0 in tile CLBLL_L_X24Y53. Note that the two LUTs are not placed in neighboring slices to force routing the connector signal via two interconnect tiles, INT_L_X24Y52 and INT_L_X24Y53. The signal is routed via INT_L_X24Y52/LOGIC_OUTS_L12, INT_L_X24Y52/NR1BEG0, INT_L_X24Y53/NR1END0, and INT_L_X24Y53/IMUX_L8.

With a 50 MHz clock constraint, Vivado reports a setup slack of 15.0 ns for the path from the controller, through the two buffers, and into the synchronizer. The entire data path delay is reported to be 4.8 ns, with logic causing 0.7 ns (14.8%) and routing causing 4.1 ns (85.2%).

Analyzing the essential bits file generated for the placed and routed design, the following bits are relevant as *Open* bits.

- Bits 10_07 and 13_07 of tile INT_L_X24Y52 enable the tile's PIP LOGIC_OUTS_L12→NR1BEG0.
- Bits 19_03 and 22_02 of tile INT_L_X24Y53 enable the tile'S PIP NR1END0→IMUX_L8. Furthermore, bits 24_02 and 25_02 must be set to "1". However, according to the *Modified Bitstream Interpretation* laid out in Table 36, flipping these bits from "1" to "0" does not cause an *Open* fault, but rather an *Input Antenna* or *Conflict* fault with either WW2END_N0_3 (bit 24_02) or NW2END0 (bit 25_02).

This suggests that the INT_L/NR1END0 → INT_L/IMUX_L8 circuit is made up differently than the INT_L/LOGIC_OUTS_L12 → INT_L/NR1BEG0 circuit. The effects of bit flips in the four bits capable of causing *Open* faults (10_07 and 13_07 in INT_L_X24Y52 and 19_03 and 22_02 in INT_L_X24Y53) were initially evaluated on a subset of twelve of the 16 devices used in Section 4.4.1 to identify mismatches between the devices (see Figure 29). Where mismatches between the devices were identified, the full extent of 16 devices was subjected to these experiments.

The fault injection experiments were carried out repeatedly while performing a temperature sweep.

Results

The results of the conducted experiments are summarized in Table 62. This table shows the input logic level the route-under-test was driven with along with the effect on the output after fault injection. If the output did not change within the monitored duration of 20.97 ms (see above), "no" output changes are recorded after fault injection. If the output changed after varying durations after injecting an *Open* fault, "variable" is recorded. It can be seen in Table 62 that in every case where the input of the route under test is driven to "1" no output changes were recorded after fault injection. Conversely, in cases where the route was driven with "0", the output changed to "1" after a device-, temperature- and resource-dependent duration in three of the four evaluated bits. However, with bit 22_02 in tile INT_L_X24Y53, this behavior was only observed on 15 of the 16 devices, and with bit 22_02 in tile INT_L_X24Y53, only on one device. The other device(s) exhibited no transition within the monitored timeframe.

Thus, there are two bits (19_03 in INT_L_X24Y53 and 10_07 in INT_L_X24Y53) that broadly exhibit variable behavior after disconnecting a PIP while its input is driven to "0" and two bits (22_02 in INT_L_X24Y53 and 13_07 in INT_L_X24Y53) that do not (or only sporadically). A potential explanation for this phenomenon is that these two groups of bits control different stages of the assumed two-stage decoded multiplexer, and that an *Open* fault occurring in bits controlling the first stage may behave differently from an *Open* fault occurring in bits controlling the second stage.

The results of the "variable" cases are shown in Figure 33 in more detail. The horizontal axis, here, displays the chip temperature as measured by the temperature sensor integrated into AMD 7 Series

Table 62: Analyzed Bits and Results from Disconnect Tests

Tile	Bit Address	Bit Offset	Input	Output Transition after FI
INT_L_X24Y52	0x00000c0a_004_07	10_07	Static “1”	no
INT_L_X24Y52	0x00000c0a_004_07	10_07	Static “0”	variable (16/16 devices)
INT_L_X24Y52	0x00000c0d_004_07	13_07	Static “1”	no
INT_L_X24Y52	0x00000c0d_004_07	13_07	Static “0”	no
INT_L_X24Y53	0x00000c13_006_03	19_03	Static “1”	no
INT_L_X24Y53	0x00000c13_006_03	19_03	Static “0”	variable (15/16 devices)
INT_L_X24Y53	0x00000c16_006_02	22_02	Static “1”	no
INT_L_X24Y53	0x00000c16_006_02	22_02	Static “0”	variable (1/16 devices)

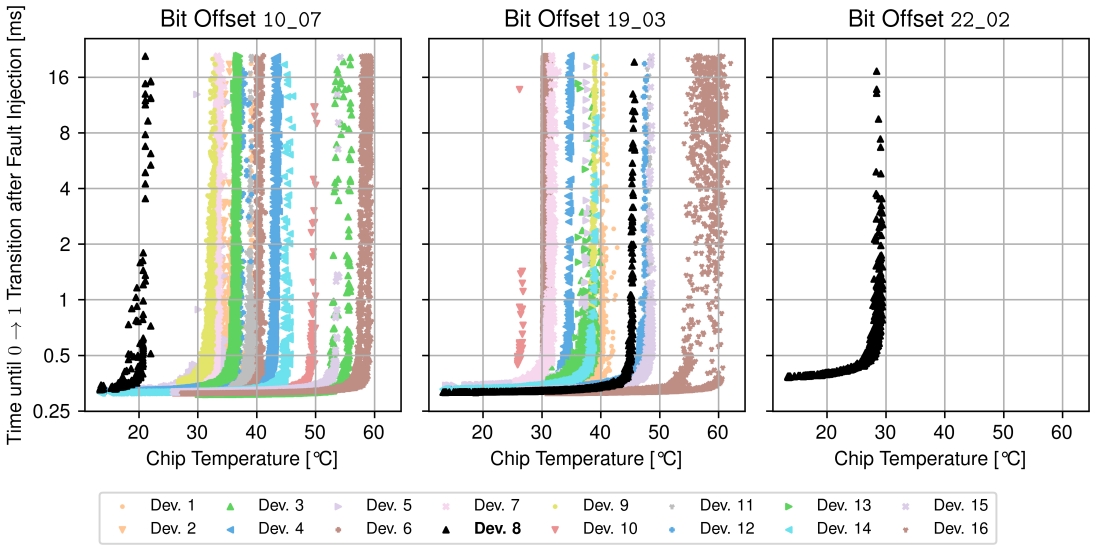


Figure 33: Temperature Behavior of Injected *Open* Faults on AMD Artix 7

devices via the likewise integrated XADC block. The vertical axis shows the duration after fault injection it took for the route-under-test’s output to change its value to “1” after being driven “0” in the moment of disconnection at each recorded temperature and for each of the 16 participating devices. The device exhibiting $0 \rightarrow 1$ transitions in three bits is plotted in black with triangle data points. No data points are plotted for a combination of device and temperature where no change after fault injection has been recorded within 20.97 ms after injection. Thus, it can be seen that at lower temperatures, the output of the route-under-test quickly changes to “1” after being disconnected. This duration non-linearly grows when repeating this experiment at higher temperatures. Above a certain device-, resource-, and bit-dependent “critical temperature”, this time rapidly increases until the wire does not change its value after being disconnected within the recorded $2^{20} + C$ clock cycles. While for bits 19_03 in INT_L_X24Y53 and 10_07 in INT_L_X24Y52 this behavior is shared among all 16 devices, only one device exhibts it for bit 22_02 in INT_L_X24Y53 as noted above. Furthermore, on the one device where the bit behaves variably, this bit’s “critical temperature” is relatively low.

Discussion

The effects of the fault characterization experiments described above can be summarized as follows: The consequences of a disconnect fault seem to depend on the state of the design. If a wire driven to “1” is disconnected by an injected *Open* fault, it will stay at “1”. However, if the wire is driven “0” during disconnection, the “0” value will be kept for some time before the value changes to “1” due to (assumed) discharge processes in the weak pullup elements in the interconnect resources. This time may systematically also depend on the multiplexer stage that is affected. A similar behavior was observed in [221] for half-latches in AMD Virtex 2 and Virtex 4 FPGAs, where these elements are used to provide logic constants. After a half-latch had been flipped by radiation, it required a considerable amount of time (up to one second) to return to its initially configured state. Due to the similarity of a half-latch and a level restoring buffer likely used in AMD 7 Series interconnect multiplexers, similar effects are likely to occur here. This behavior may lead to different effects of fault injection in the same bit depending on the state of the design, as the same bit can (temporarily) cause a stuck-at-0 fault when the corresponding interconnect bit is flipped while the signal is driven to “0”, but a stuck-at-1 fault when the signal is driven to “1”. Additionally, the variable (device-, temperature-, and multiplexer-dependent) time for a stuck-at-0 fault to develop into a stuck-at-1 fault may also affect the design, depending on how many clock cycles pass between fault injection and the continuation of the design’s operation.

4.4.3 Supply Voltage & Inter/Intra-Chip Variation in AMD 7 Series FPGAs

Another factor – in addition to process variations (P) and temperature (T) and – related to possible variations in fault injection experiments as observed in Section 4.4.1 is the supply voltage (V) at which fault injection experiments are conducted. Dynamically, higher supply voltages lead to lower gate delays, thus being one component of Dynamic Voltage and Frequency Scaling (DVFS) to improve digital circuit efficiency [222]. At a higher supply voltage, an identical gate capacitance is charged faster up to a threshold voltage than at a lower supply voltage, thus potentially also affecting half-latch circuits and thereby *Open* faults. Statically, variations in supply voltages may affect the relative drive strengths of interconnect drivers short-circuited by an *Input Antenna* or *Open* fault. Thus, in this section, the effect of varying supply voltages on the outcome of *Input Antenna* faults in AMD 7 Series FPGAs is investigated.

Furthermore, as shown in Section 4.4.2, different interconnect resources (and even different interconnect-related bits configuring the same interconnect resource) may behave differently in AMD 7 Series FPGAs. This suggests the possibility of significant intra-device variation of the outcome of fault injection experiments, which is investigated experimentally in this section alongside voltage-related variation.

Experimental Setup

AMD’s Artix 7 devices require four supply voltages [223]: The internal (core) supply V_{CCINT}, the auxiliary supply voltage V_{CCAUX}, the BRAM supply voltage the auxiliary supply voltage V_{CCBRAM}, and the output supply voltage V_{CCO}.

AMD generally bins Artix 7 devices by speed grade (–3, –2, –1). Additionally, the speed grades –1LI and –2L denote devices that have been binned for low-voltage operation: while conventional Artix 7 devices operate at a nominal V_{CCINT} of 1.0 V, low-voltage devices nominally operate at 0.95 V (–1LI) and 0.9 V (–2L). Across all variants, AMD’s recommended operating conditions for V_{CCINT} range from 0.87 V to 1.05 V [223].

Evaluating the effect of supply voltage levels on the outcome of bitstream-level fault injections requires a fault injection platform that is able to vary the FPGA's core supply voltage V_{CCINT}. Digilent's BASYS3 board, which was employed in the characterization experiments described in Section 4.3.1 and the temperature-based investigations in Section 4.4.1, uses an Analog Devices LTC3633 DC-DC converter[99] to generate a fixed V_{CCINT} of 1.0 V[100]. Thus, as an alternative, Digilent's Arty A7-35T board[101] was selected to perform the evaluations described in this section. This board uses a Renesas Electronics DA9062 Power Management IC (PMIC)[102] to generate all FPGA supply voltages. This PMIC's output voltages can be adjusted by accessing the `VBUCK[1-4][A,B]` registers via I^2C. As V_{CCINT} is connected to DA9062's `VBUCK1` switching regulator output, this voltage can be adjusted between 0.3 V and 1.57 V in a granularity of 10 mV.

Instead of the xc7a35tcpg236-1 device provided by the BASYS3 board, the Arty A7-35T board provides an xc7a35tcsg324-1LI device that operates typically at 0.95 V. The voltage steps 0.88 V, 0.92 V, 0.95 V, and 1.05 V – spanning the entire voltage range supported by Artix 7 devices [223] – were selected for the evaluations in this section. The experiments were carried out on four devices with identical date- and lot codes (date code *2129*, lot code *D6489789A*, revision code *ABX*). The four boards' PMICs were controlled via I^2C from a host PC, which also controlled fault injection experiments and stored result data. The experiments were conducted under controlled temperature conditions under an environment temperature of 35 °C.

For conducting fault injection experiments, the one-wire characterization circuit implemented for *Input Antenna* characterization described in Section 4.3.1 was ported to the Arty A7-35T FPGA platform. Routing variations of this circuit were generated to test a total of 21030936 *Input Antenna* faults in 1150 interconnect tiles comprising an eight-column-wide vertical slice of the AMD Artix 7 xc7a35t FPGA[103]. This slice was chosen to include a column of BRAM tiles (connected to column 30) as well as the PCIe macro that takes up a space of two by 25 interconnect tiles (column 28, row 100 to column 29, row 125; connected to columns 27 and 30 in that range).

Results

The results of the experiments described above are summarized in Figure 34. The basis for this visualization are the per-tile, per-device, per-voltage sets of critical *Input Antenna* faults, characterized by the involved source, destination, and incident wire, as well as the flipped configuration bit. An *Input Antenna* fault is considered critical if it affects the logical value transported by the route-under-test.

Figure 34a shows the union set sizes of these critical *Input Antenna* faults across the four evaluated devices concerning the number of tested faults per tile and voltage. It shows how many of the evaluated *Input Antenna* faults were critical on *any* of the four devices. In the same way, Figure 34b shows the intersection set sizes of these critical *Input Antenna* faults, i.e., how many of the evaluated faults were critical on *all* devices. The ratio between these set sizes, i.e., the *Intersection-over-Union (IoU)* ratio, can be seen in Figure 34c. Furthermore, all figures show the average value across all contained tiles and its standard deviation (abbreviated "SD").

[99] See `https://web.archive.org/web/20230522194459/https://www.analog.com/media/en/technical-documentation/data-sheets/3633fd.pdf`, accessed 2026-01-05

[100] See `https://web.archive.org/web/20241010230352/https://digilent.com/reference/_media/reference/programmable-logic/basys-3/basys-3-sch-public-rev-d.pdf`, accessed 2026-01-05

[101] See `https://web.archive.org/web/20250108171649/https://digilent.com/reference/programmable-logic/arty-a7/start`, accessed 2026-01-05

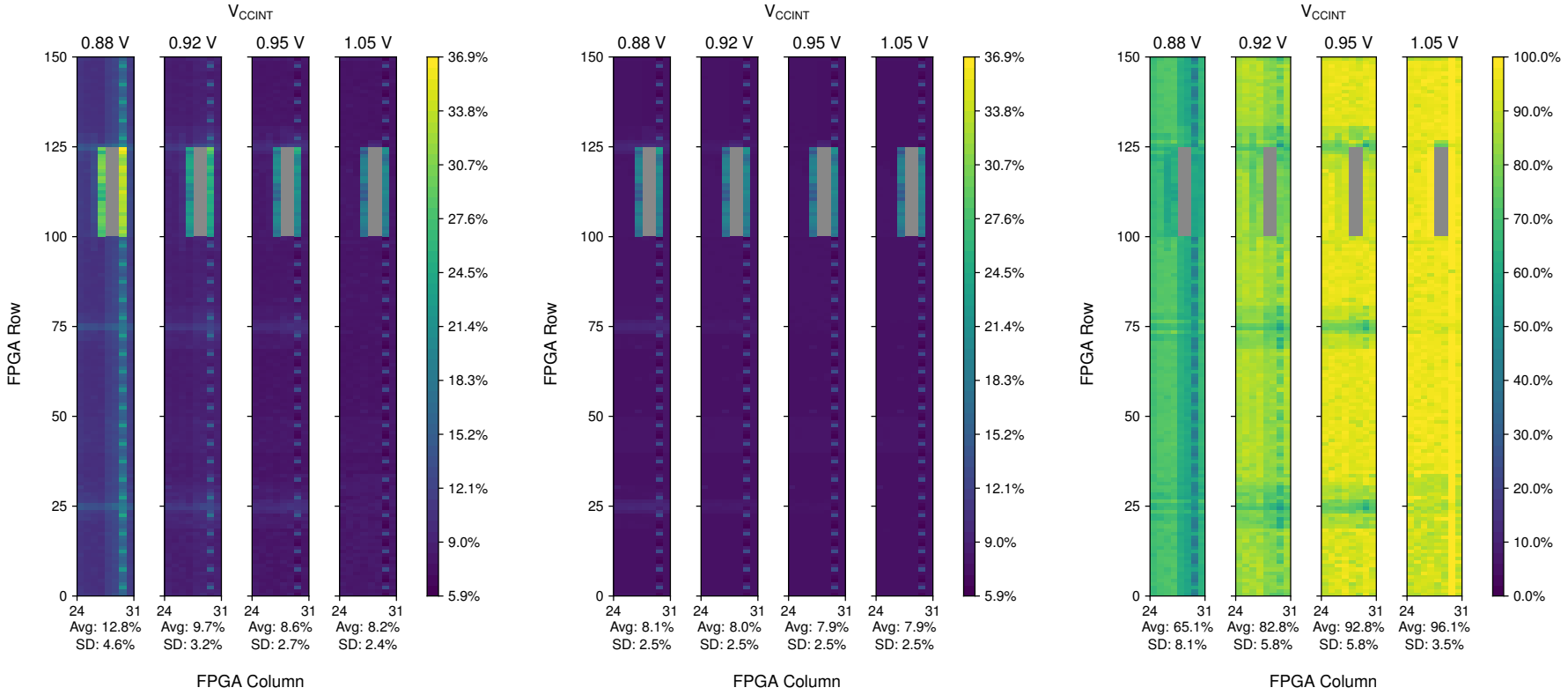
[102] See `https://web.archive.org/web/20250224214321/https://www.renesas.com/en/document/dst/da9062-datasheet`, accessed 2026-01-05

[103] Thus, approx. 20% of the device's 5650 interconnect tiles were evaluated (See Table 34)

A first observation is that the number of *Input Antenna* faults critical on *any* device (i.e., the union set size; see Figure 34a) shrinks with larger supply voltage, starting at an average of 12.8% of tested faults per tile at 0.88 V (lower end of the recommended supply voltage range), reducing down to an average of 8.2% at 1.05 V (upper end of the recommended supply voltage range). To a lesser extent, this also applies to faults critical on *all* devices, as the average intersection set size concerning the tested faults per tile (see Figure 34b) is reduced from 8.1% at 0.88 V to 7.9% at 1.05 V. These values approximate the 7.3% observed for CLB-related interconnect tiles observed in *Input Antenna* characterization as seen in Section 4.3. These effects are evident when comparing the ratio of intersection and union set sizes per tile and voltage (see Figure 34c), which shows a significant increase of device uniformity (i.e., IoU ratio nearing 100%) at higher supply voltages: while on average, only 65.1% of faults observed critical per tile on *any* device are critical on *all* devices at 0.88 V, this is the case for an average of 96.1% of faults per tile at 1.05 V.

Comparing criticalities at the same voltage across the evaluated tiles, it can be seen that there are location-related differences in both the union and intersection set sizes of critical bits: Tiles in rows 100–125 of columns 27 and 30 appear to contain an especially large amount of critical faults compared to, e.g., tiles in columns 24–26 and 31. Above and below this region in column 30, a regularly repeating pattern (with a period of five rows) of low and high criticality can be seen. These effects are present at all evaluated voltages.

A further effect that is visible in the results shown in Figure 34 is that there appears to be a region of slightly higher criticality and non-uniformity of that criticality in all columns in the rows surrounding rows 24–25, 74–75, and 124–125: In these rows, the ratio of union set size vs. tested faults (see Figure 34a, visible at 0.88 V–0.95 V) as well as the ratio of intersection set size vs. tested faults (see Figure 34a, visible at 0.88 V and 0.92 V) is slightly increased locally when compared to the surrounding tiles above and below. Especially at 0.92 V and 0.95 V, the tiles with the lowest IoU ratio are found in these regions (see Figure 34c). This means that in these regions, (1) more *Input Antenna* faults are critical than in “normal” interconnect tiles and that (2) these are comprised of device-variable subsets of faults (i.e., combinations of source, destination, and incident wires).



(a) % Critical on *Any* Device (Union Set)

(b) % Critical on *All* Devices (Intersection Set)

(c) Intersection-over-Union (IoU) Ratio

Figure 34: Criticality (Critical *Input Antenna* Faults vs. Evaluated Faults) vs. FPGA tile location and V_{CCINT} supply voltage

Discussion

The regular differences in per-tile criticality, best visible in Figure 34b, can be explained by the type of logic tile neighboring the individual interconnect tiles, as already observed in Section 4.3.2. As mentioned above, the shown slice of the xc7a35t FPGA includes a PCIe hard IP. This IP takes up the space of 25 rows by two columns of interconnect tiles, shown as the gray rectangular section (28, 100)–(29, 124) in Figure 34. The surrounding interconnect tiles in rows 100–124 of columns 27 and 29 connect to this hard IP via the interconnect tiles' logic interface wires (`LOGIC_OUTS[0-23]`, `IMUX[0-47]`, etc.; see Table 33). Furthermore, column 30 interfaces to a column of BRAM tiles via their equivalent logic interface wires. All other interconnect tiles in this section interface to CLB tiles containing LUTs and FFs.

Thus, different subsets of the logic output wires (i.e., `LOGIC_OUTS[0-23]`) are expected to be critical incident wires in *Input Antenna* faults depending on the neighboring logic tile. For interconnect tiles neighboring CLB tiles, it has generally been observed in characterization (see Tables 45 and 46), that `LOGIC_OUTS[8-15]` (connected to LUT outputs) are the most consistently critical incident wires. For BRAM tiles, this subset depends on the vertical offset of the interconnect tile to the neighboring (five interconnect tiles high) BRAM tile: While at offsets +0,+1,+3, and +4 the critical subset of `LOGIC_OUTS[0-23]` wires contains six to eight wires (thus comparable to CLB tiles), this increases to 14 wires at offset +2 (see Table 50a). This likely explains the regularly repeating criticality pattern with a period of five tiles observed in column 30. For the PCIe tile, no extensive characterization was performed in Section 4.3. However, similar effects and offsets are assumed to be present in the adjacent interconnect tiles in rows 100 – 124 of columns 27 and 29.

While incident wires of type `LOGIC_OUTS[0-23]` were observed consistently critical (related to the neighboring logic tile), the set of additionally critical incident wires consists of additionally critical *general* interconnect wires. It is assumed that slight local and device-related differences in either supply voltage (via the chip's power distribution network) or manufacturing variations in the transistor sizing in the respective drivers lead to these effects, which are also (to a small extent) visible in the characterization results when comparing Tables 45 and 46.

The regions with slightly increased criticality and increased device-to-device variation around rows 24–25, 74–75, and 124–125 may be related to the structure of the AMD Artix 7 FPGA device: AMD's 7 Series FPGAs are sectioned into *clock regions* that span 50 rows of interconnect tiles and a variable number of columns. Clock distribution circuits are assumed to be placed between the center pair of rows in each row of clock regions. The xc7a35t device is divided into six (two by three) clock regions, splitting the device vertically into thirds and horizontally into two halves. The clock regions are enumerated left-to-right and bottom-to-top (`X0Y0`–`X1Y2`). The tested slice of the FPGA is part of clock regions `X1Y0` (rows 0–49), `X1Y1` (rows 50–99), and `X1Y2` (rows 100–149). Thus, the aforementioned clock distribution circuits are assumed to be placed between rows 24–25, 74–75, and 124–125. Looking at (1) the device view provided by AMD Vivado and (2) a die microscopy[104] of the FPGA [224] as shown in Figure 35 indeed confirms the presence of (1) a row of `HCLK_...` tiles between these rows, and (2) horizontal "spines" disrupting the FPGA's columns at these positions. Thus, It is assumed that the effects seen in these rows may be caused by deliberate design measures to improve signal integrity (e.g., by adjusting drive strengths by tweaking transistor sizes) or inadvertent slight local dips in supply voltage due to the proximity to high-powered global drivers.

[104] The die microscopy [224] is from an xc7a50t device, while the evaluated device is an AMD xc7a35t FPGA. However, identical bitstream size [30] and static power consumption [223] suggest that its die is identical to the die used in an xc7a50t device. Furthermore, Project XRAY assumes identical fabric layouts for both devices (see `https://web.archive.org/web/20250207235526/https://f4pga.github.io/prjxray-db/artix7/mapping/devices.yaml`, accessed 2026-01-05)

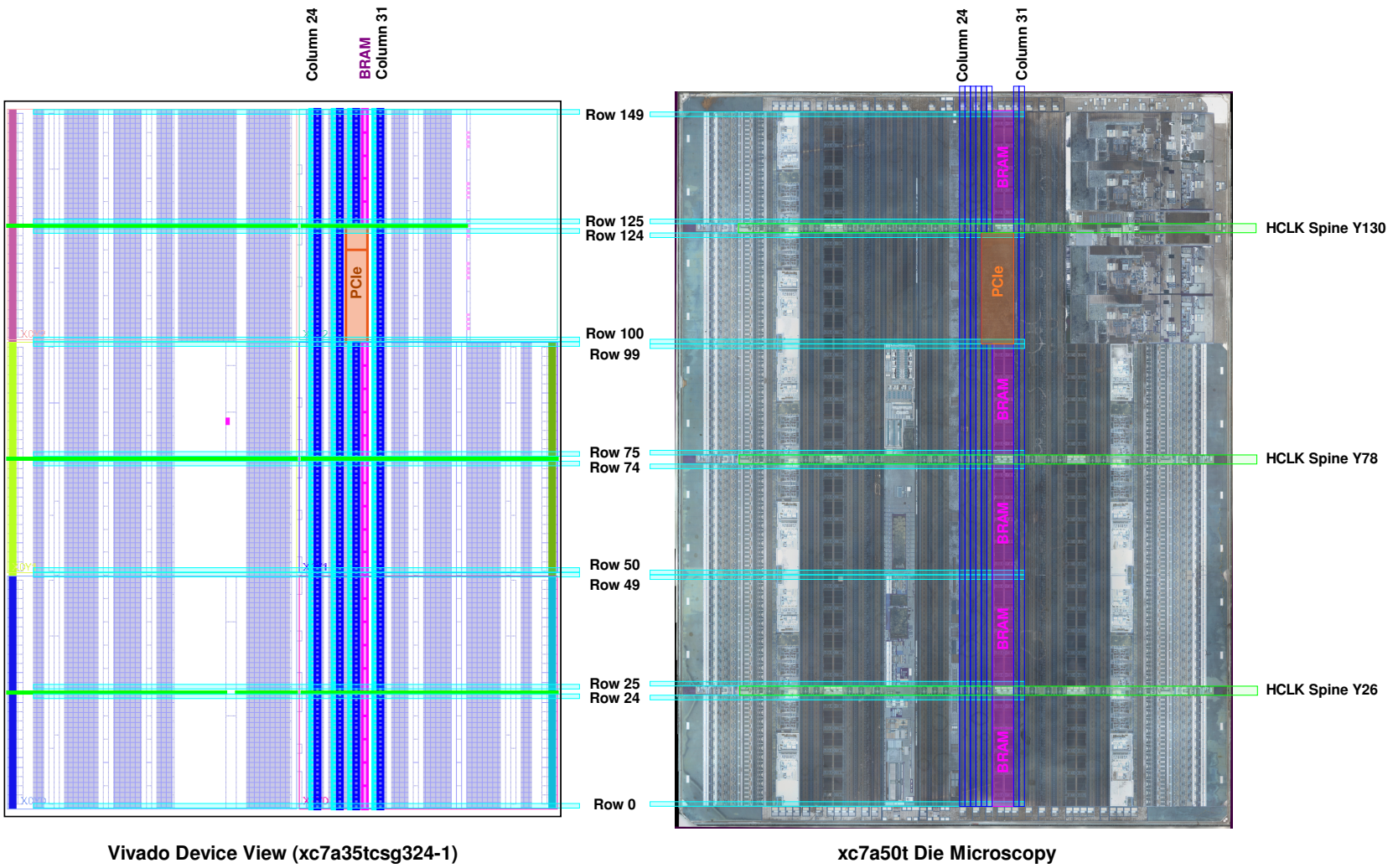


Figure 35: Selected Rows and Columns Overlayed on Artix 7 Vivado Device View & Die Microscopy [224]
Die Microscopy “xilinx_xc7a50t_s1-2_mit20x (l1)” by John McMaster.
Image source: `https://siliconprawn.org/map/xilinx/xc7a50t/single/xilinx_xc7a50t_unknown_s1-2_mit20x.jpg`
Licensed under CC Attribution 4.0 International – see `https://creativecommons.org/licenses/by/4.0/`

Thus, the results observed in this section allow a deeper insight into the sources of device-to-device variation in fault injection results observed in earlier experiments on AMD Artix 7 (e.g., shown in Tables 45 and 46 as well as in Section 4.4.1). They further indicate that the supply voltage at which fault injection experiments are carried out on these devices influences the number and distribution of critical *Input Antenna* faults. Fault injection results obtained at a higher supply voltage may not fully apply to the same design on the same device when powered using a lower supply voltage. Finally, future placement and routing algorithms could take the effects observed around the `HCLK` spines into account and assume a higher placement and routing cost for these locations to avoid placing cells or routing wires in these regions.

4.4.4 Summary

The evaluations carried out in this section revealed significant device-, temperature-, and voltage-dependent variations in fault injection results obtained via configuration-level fault injection on two FPGA models by two device vendors. While implicitly acknowledged by previous academic works [139, 121], no prior in-depth evaluation of these effects exists to this dissertation's author's best knowledge. The evaluations described in this section are published, in part, in [1, 2, 3, 4]. A recent publication on the topic [225] follows our argumentation that variability in fault injection results is likely caused by PVT effects.

On **AMD's Artix 7 FPGAs** devices especially interconnect-related faults are affected by these variations. Among interconnect faults, device-to-device variation has been observed both with *Open* and *Input Antenna* faults:

For *Open* faults, temperature sweep experiments revealed that device temperature influences the duration it takes a newly disconnected signal to be pulled to "1" after fault injection, given that it was driven to "0" in the moment of disconnection. At higher temperatures, signals have been observed to stay at "0" for a longer period. This time, however, appears to strongly depend on device and individual interconnect resource. In a synthetic benchmark design consisting of standard digital logic building blocks (an LFSR, a counter, and a state machine), *Open* faults have been observed to be the most temperature-dependent category of faults. With rising temperature, the number of *Open* faults critical on *any*, but not necessarily *all* of the four tested devices (i.e., the union set of critical *Open* faults) significantly shrank (by approx. 31% between 20 °C and 50 °C environment temperature), while the number of *Open* faults critical on *all* devices rose by 8% in the same range. Other faults (i.e., *Input Antenna/Conflict*, or faults affecting logic tiles) remained relatively stable within this range. Thus, with rising temperatures, the devices began to act more "uniformly" regarding the criticality of *Open* faults.

For *Input Antenna* faults, experiments at different supply voltages across the AMD Artix 7 FPGA's recommended core supply voltage range (0.88 V – 1.05 V) and across a larger cross-section of the device revealed both intra- and inter-chip variation in the criticality of *Input Antenna* faults. These faults are both more common and more diverse among devices at (1) lower supply voltages and (2) at particular locations within the device surrounding (assumed) clock distribution spines. Testing at higher supply voltages caused the devices to act more uniformly, i.e., a more extensive overlap between *Input Antenna* faults critical on *any* device vs. critical on *all* devices developed. Furthermore, testing at higher supply voltages caused fewer *Input Antenna* faults to have a malign effect on the observed route-under-test, indicating a higher tolerance for these faults at higher supply voltages.

While significant inter-device variations were also observed with **Lattice iCE40 FPGAs**, variations related to temperature (within the tested temperature range) were not as prominent in these devices

when compared to AMD's Artix 7 FPGAs. Still, inter-device variation must be considered when designing with these devices. In contrast to AMD Artix 7 and Lattice iCE40, as discussed in Section 4.3, no inter-device variation was observed in the experiments carried out on **Lattice ECP5 FPGAs**.

In general, previous academic works generalize fault injection results obtained on one device to all devices of the same type – to this dissertation's author's best knowledge, no previous works except [139, 121] perform repeated fault injection experiments or repeat experiments across devices. On the contrary, previous academic works have proposed parallelizing fault injection tasks on multiple identical FPGA devices [140, 141], thus implicitly assuming generalizability between devices. If these approaches are used to determine an overall *number* of critical bits, relatively accurate results may still be achievable despite inter-device variations. This is illustrated in Tables 60 and 61, which show that while different *sets* of bits are critical on different devices, the overall *count* remains fairly consistent: for 16 AMD Artix 7 devices, the standard deviation was 1.6% of the average critical bits across all devices at 20 °C and 0.4% at 50 °C. For ten Lattice iCE40 devices, across all temperature steps, the standard deviation was 0.7% of the average number of critical bits. Thus, for the sole purpose of determining a *number* of critical bits, generalization from one device to other devices appears to be appropriate. Nevertheless, as shown in the case of AMD 7 Series FPGAs, the conditions under which these experiments are conducted (temperature and supply voltage) may significantly impact these results.

If, however, the criticality of individual critical bits is to be precisely determined, e.g., to validate the introduction of specific fault tolerance measures such as reliability- and CMF-aware placement and routing, these inter-device variations should be considered by repeating tests on multiple devices. As shown in Figures 29 and 30, testing on eight to ten devices may be sufficient to discover less than 1% additional critical bits with the next evaluated device.

4.5 Static Criticality Prediction for AMD 7 Series FPGAs

This section describes how the information gained in the preceding sections concerning fault modeling, interconnect fault characterization, and PVT effects on interconnect faults can be used to form a criticality prediction approach as lined out in Figure 6.

After decoding the bitstream and placement and routing data (see Section 4.1.1), the first step towards predicting the consequences of a bit flip is to associate the bit with the potentially affected netlist elements as lined out in Section 4.2: First, the individual FPGA resource affected by a bit flip is identified from the configuration bit's frame address and offset in the configuration column. Afterwards, the bit's effect when flipped is considered based on its function:

- For LUTs initialization bits, the change in the respective LUT's initialization string is recorded; the actual effect is determined at a later stage when considering unused inputs and the usage of the direct (O6) and fractured (O5) output of the LUT (see Section 4.2.1)
- For other logic configuration bits such as those configuring the operating mode of LUTs (configuring it as a LUT, shift register (SRL) or distributed RAM (LUTRAM)) or flip-flops, the (potentially) affected FPGA elements are recorded. This must take into account that a single configuration bit can affect one or multiple FPGA elements:
  - The CLB slice configuration bits responsible for activating or deactivating flip-flops' clock enable, reset inputs, and clock inverter and the bits configuring slice flip-flops as

level-sensitive (i.e., latches) vs. edge sensitive (i.e., flip-flops) and asynchronously vs. synchronously reset affect all eight flip-flops in a slice in AMD 7 Series FPGAs [98].

– The CLB slice configuration bits configuring LUTs as distributed RAM or shift registers, as well as the configuration bits responsible for generating the write enable signal for these elements potentially affect all four LUTs in a slice in AMD 7 Series devices.

Furthermore, care has to be taken when assessing the cells potentially influenced by a bit flip affecting LUTs and FFs as they may not be associated with a netlist cell as such, but instead instantiated as route-through cells by the implementation tool. This is done to improve routability: A signal can be routed through a LUT simply by attaching it to one of its inputs and generating an appropriate initialization string (i.e., truth table) that forwards the signal value unchanged. In some cases, AMD Vivado also routes signals through flip-flops by using their ability to be configured as level-sensitive latches. In this case, all slice flip-flops are configured as latches, and clock as well as clock enable is configured appropriately to configure the latch to always be transparent, thereby allowing to forward its data input directly to its output.

- Configuration bits responsible for configuring DSPs and BRAMs tiles are subdivided into bits responsible for configuring (1) the tile's interconnect and (2) the logic element's functionality. The former bits are treated in a similar way to generic interconnect bits, while the latter are associated with a netlist cell. No further modeling of the effect of BRAM or DSP configuration bits is performed (see Section 4.2.3)

- Configuration bits affecting clock routing (i.e., bits found in the configuration of `HCLK_[R,L]`, `HCLK_CMT(_L)`, `CLK_HROW_...`, and `CLK_BUFG_REBUF` tiles are categorized as clock-related bit. Routing information is used to associate each clock routing bit with potentially affected cells.

- For both generic interconnect as well as intra-CLB interconnect resources, the faulty configuration is evaluated according to an assumed circuit-level model as described in Section 4.2.2 leading to a further classification into *Open*, *Input Antenna*, *Conflict*, or *Bridge* faults. Faults potentially causing conflicts in long bidirectional wires are categorized separately.

Once the architectural consequences of the fault, including all affected logic elements, routing resources, and interconnect wires, are known, the potential effect on the design is determined. This is done by enumerating all logic cells potentially affected by the fault:

- For SEUs affecting logic resources, this straightforwardly means identifying the netlist cells placed on the identified FPGA element by consulting placement information.

- For interconnect faults, all cells *downstream* of the fault are identified by consulting the routing graph of the affected net as proposed first in [99]. The necessary device-level routing graphs (i.e., containing information on which interconnect elements are used to route a particular net and how they are connected to do so) are extracted from placement and routing information, which, in turn, can be extracted using RapidWright from a *Design Checkpoint* on AMD 7 Series FPGAs (Section 4.1.2).

The mapping of configuration bits to the netlist and the netlist-level consequences of each fault determined in the previous step are subsequently combined with criticality and masking assumptions. On this basis, a prediction of bits that are likely to be susceptible to SEUs can be made. To estimate this criticality on the bit level, the following rules are applied to each bit:

R1: Due to their potential widespread effect, all bits related to clock resources and clock routing (in all types of tiles) are considered critical.

R2: A subset of configuration bits of LUTs with statically driven inputs are *"Don't Care"* bits and are thus considered non-critical. This is done according to the method described in Section 4.2.1, taking the utilization of both LUT outputs (fractured and direct) into account.

R3: Antenna-type interconnect faults in AMD 7 Series FPGAs are known to seldom produce erroneous outcomes [3, 124] and are thus only considered critical when (1) the error concerns global routing (i.e., connections originating from a `GCLK`-type wire, and connecting to a `CLK(_L)[0,1]` or `GFAN[0,1]` wire) or connects an incident wire driven by an unused LUT or other unused logic element. The latter connections are handled as follows: First, Project XRAY's interconnect documentation is queried if the concerned logic output wire is potentially driven by an (unused) FPGA element. This eliminates all unused `LOGIC_OUTS` incident wires, e.g., ones that are tied-off as the neighboring tile is empty (e.g., in interconnect tiles neighboring `VFRAME` tiles). Static reference tables for interconnect tiles neighboring CLB (i.e., outputs of unused LUTs connected to `LOGIC_OUTS(_L)[8-15]` wires) and BRAM tiles (unused BRAM cells connected to logic output wires as shown in Table 50a) based on fault characterization results are used to determine which *Input Antenna* faults to consider critical. As almost all logic output wires in CLB tiles neighboring DSP tiles containing unused DSP elements were observed critical in characterization as shown in Table 50b, all logic output wires connected to an unused DSP cell are considered potentially critical. The same is done for all other tile types not handled above, including PCIe tiles.

R4: In certain constellations, faults modeled as bridge faults on AMD 7 Series FPGAs appeared to never cause critical consequences in preliminary fault injection experiments, allowing them to be considered non-critical.

Essential bits not captured by these rules are considered as *potentially* critical. To estimate these bits' criticality in the context of (partial) TMR designs, we follow the general approach from [99]: If a fault affects a non-triplicated netlist cell, a non-triplicated TMR voter, or multiple replicas of the same triplicated module, it is considered critical as it may either cause a non-mitigated error or a CMF. If it only affects one replica of one or more replicated modules, it is considered masked by TMR and thus non-critical. Once again, clock-routing-related bits are treated differently, as clock resources were identified to be susceptible not only to "downstream" disruptions due to interconnect faults but also "upstream" disruptions affecting the clock signal distributed via interconnect multiplexers other than the one directly affected by the fault. Thus, additional CMFs are assumed for each of these interconnect resources. No influenced cell could be identified for a small subset of potentially critical bits. In some cases, malignant consequences can be virtually ruled out (e.g., bits associated with configuring logic resources in tiles where no cells are placed and no wires are routed), while any bit not fitting this category is considered critical. The sorting process described above is qualitatively illustrated by Figure 36. Its outcome is a set of bits predicted to be critical and a set of bits predicted to behave benignly when flipped.

These mapping and prediction efforts are agnostic towards the approaches taken in placement and routing (P&R) of the original design: As fault effects are extracted from the bitstream in combination with device-level fault modeling and the design's netlist, if the design is placed and routed in a more reliability-oriented way, simply fewer CMFs would be reported. It is thus applicable to both conventional P&R performed by vendor tools and reliability-aware P&R approaches [99, 226, 134, 117].

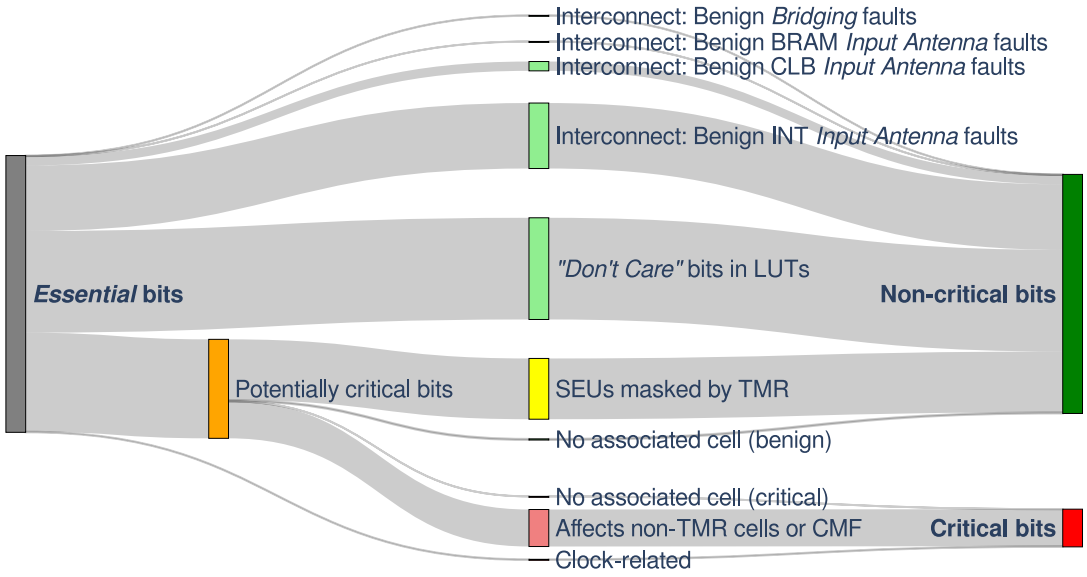


Figure 36: Illustration of Criticality Prediction Rules for AMD 7 Series

4.5.1 Validation Experiments

To evaluate the accuracy of the predictions formulated above for AMD Artix 7 FPGAs, the prediction approach and rules described above were applied to four diverse case study designs:

- *B13m*, a version of the widely-used *B13* ITC'99 benchmark design [164] modified by (1) moving the *B13*'s five processes (`p01`-`p05`) to individual hierarchical modules, and further wrapping each of *B13*'s gate-level cell instantiation by a hierarchical module to generate a hierarchical design, and (2) adding one LFSR as an input pattern generator and another one as an output signature generator,
- *Add32*, a design generated by the PandA Bambu HLS tool [227] from a C program that adds a predefined set of 32-bit integers,
- *Adler32*, another design generated by PandA Bambu from a C program that calculates the Adler32 checksum [228] of a predefined dataset, and
- *SERV*, an SoC based on the bit-serial RISC-V CPU SERV[105] running a program adding 32-bit integers.

Table 63 shows the designs' hierarchical makeup and the distribution of their logic resource. Both *Add32*'s and *Adler32*'s hierarchical structure is relatively flat, with most modules (and logic) being a descendant of one of three modules at the third hierarchical level. At this hierarchy level, and for relatively simple designs, PandA Bambu instantiates a central state machine, a datapath module, and a module to generate the interface signals. Most logic is concentrated in the datapath module, where the functional units (i.e., adders, multiplexers, etc.) are instantiated. The state machine orchestrates the operation of these modules at runtime given by the scheduling determined during HLS. Due to its higher complexity, *Adler32*'s datapath contains significantly more modules (and logic) than *Add32*'s. Finally, *SERV* has a deeper, more *organic* hierarchical structure due to it being based on a hand-written, well-modularized RTL description. The designs are implemented to be self-contained, requiring only an external *start* pulse to start a "self-test", for which a signature

[105] See `https://web.archive.org/web/20250213123216/https://github.com/olofk/serv`, accessed 2026-01-05

value is presented at the output after a certain number of clock cycles on the assertion of a *done* pulse by the DUT. This follows the interface generated by PandA Bambu for designs from the CHStone HLS benchmark suite [229].

Design	Levels	Modules	LUT	FF	Carry	BRAM	DSP	Wide MUX
B13m	5	114	101	83	–	–	–	–
Add32	5	18	132	126	7	–	–	–
Adler32	5	51	408	215	70	1	2	–
SERV	6	19	405	208	–	2	–	1

Table 63: Logic Resources of Case-Study Designs

The designs' resource utilization furthermore represents a microcosm of commonly used FPGA resources, not only making use of LUTs and FFs, but also utilizing DSP and BRAM resources. This allows to judge the potential influence of those special-function blocks on the approach's applicability.

These designs are first synthesized for AMD Artix 7 technology from their RTL source to hierarchical netlists using AMD Vivado 2019.1. The resulting netlists are then adapted using Synopsys Synplify Premier 2019 into three variants for each design to evaluate the effectiveness of the TMR-related reliability assessment proposed above: one variant with a small amount of logic (or none) protected by TMR, a variant with medium TMR protection, and a variant with a large amount of protection. TMR is introduced as block-TMR on the hierarchical module level, meaning that individual netlist submodules are triplicated and appropriate voters are generated.

The netlists are then instantiated in a pre-generated fault injection platform (present as an AMD Vivado *design checkpoint*, i.e., a pre-placed and routed "harness") adapted from the design-level temperature dependency evaluation described in Section 4.4.1. This platform contains a MicroBlaze SoC that monitors correct DUT execution (latency and output signature), monitors chip temperature by interfacing with the on-chip XADC block, and performs fault injection via ICAP. The DUT is floorplanned into a distinct region of the targeted AMD Artix 7 xc7a35tcpg236-1 FPGA to minimalize interference between DUT and fault injection and management logic. After placing and routing the DUT in its floorplan region of the fault injection platform using AMD Vivado, the resulting placed and routed design was both exported as another *design checkpoint* (.dcp) file for future placement and routing extraction, as well as a bitstream (".bit" file) for performing criticality analysis and fault injection experiments. Furthermore, as a starting point for (1) fault injection and (2) criticality analysis, an AMD Essential Bits Mask (.ebd) file was generated, containing a mask of bits determined by Vivado to be related to the design.

As described at the beginning of this section, these files form the basis for criticality analysis, which was performed for each of the twelve design variants. This first results in a mapping of *Essential Bits* to assumed fault consequences[106] and potentially affected cells, which is subsequently used to predict each fault's criticality.

Finally, fault injection experiments on the twelve design variants were conducted on two groups of twelve Digilent BASYS 3 boards as shown in Table 64. This table shows the device DNA[107], date code, and lot code for all employed FPGA[108] The two groups contain a mix of different date- and lot codes in order to assess the accuracy of the prediction approach with varying devices. It is to

[106] A maximum of 0.3% of configuration bits deemed *essential* by Vivado were not documented by Project XRAY. As no criticality prediction is possible for these bits, they were excluded from further analysis, as worst-case assumptions (all critical) may be made for these bits

[107] The *DNA* is a shortened serial number used by AMD to identify groups of up to 32 individual devices [30]

[108] In the second group (see Table 64b), a DNA collision between devices 8 and 12 is visible.

Table 64: Employed AMD Artix 7 xc7a35tcpg236-1 Devices for Prediction Validation

(a) Device Group 1

#	Device DNA	Date & Lot Code	
1	0020c292ce8c854	1913	D5851385A
2	002d4292ce8c85c	1913	D5851384A
3	00a8c292ce8c85c	1913	D5851385A
4	00c54002428c854	1909	D5834810A
5	0140c292ce8c854	1913	D5851385A
6	02ab4292ce8c85c	1913	D5851384A
7	02cd4292ce8c85c	1913	D5851384A
8	03265c809427054	1445	D5034186A
9	05284292ce8c85c	1913	D5851384A
10	05865c80942705c	1445	D5034186A
11	06865c809427054	1445	D5034186A
12	07c65c80942705c	1445	D5034186A

(b) Device Group 2

#	Device DNA	Date & Lot Code	
1	0000c292ce8c854	1913	D5851385A
2	01ab4292ce8c85c	1913	D5851384A
3	0328c292ce8c85c	1913	D5851385A
4	03484292ce8c854	1913	D5851384A
5	03c8c292ce8c85c	1913	D5851385A
6	048f4292ce8c854	1913	D5851385A
7	04a8c292ce8c854	1913	D5851385A
8	04ad4292ce8c85c	1913	D5851384A
9	0508c292ce8c854	1913	D5851385A
10	058d4292ce8c85c	1913	D5851384A
11	1028c292ce8c85c	1913	D5851385A
12	04ad4292ce8c85c*	1913	D5851384A

be noted that the first group (see Table 64a) contains devices that were used in characterization and PVT evaluation experiments. No such overlap exists for the second group of devices shown in Table 64b.

Fault injection experiments started by loading a bitstream containing the DUT and the monitoring and fault injection harness into the configuration Flash of all targeted Digilent BASYS3 FPGA boards. Single-bit-errors are then injected via partial reconfiguration using the following procedure:

1. The FPGA loads the fault-free bitstream from configuration Flash,
2. The bit to be targeted by fault injection is communicated to the FPGA via UART from a PC,
3. The correct operation of the DUT is verified by performing an iteration of its functionality.
4. The DUT is reset, and its clock is stopped to avoid glitches during reconfiguration.
5. A bit flip is injected using the FPGA's ICAP: The configuration frame containing the currently targeted bit is read into a buffer, the targeted bit is inverted, and the frame is written back.
6. The DUT and its clock are enabled again.
7. The DUT performs an iteration of its functionality.
8. Execution results are reported back to the control PC.
9. The fault-free configuration is re-loaded from Flash to ensure a clean state for the next experiment.

Designs targeted by the presented evaluation possess a self-test function with a pre-determined latency and signature value. A fault is considered critical if the design's self-test delivers an incorrect signature or its latency mismatches a simulation-determined value. This allows the designs-under-test to be treated as "black boxes" without detailed knowledge of bit-flips' effect on the circuit's behavioral conditions. As fault injection results may be impacted by PVT effects (see Section 4.4) the individual device used for fault injection is recorded, and the environment temperature is controlled. Experiments were conducted at a controlled environment temperature of 35 °C.

The sets of bits critical on *any* device are then compared to the bits predicted to be critical and non-critical by the proposed approach. This allows to deduce the following four sets of bits:

- True Positives (TP), i.e., bits predicted to be critical (C_P) that were observed to be critical during fault injection (C_{FI}), i.e., $C_P \cap C_{FI}$.
- True Negatives (TN), i.e., bits predicted to be non-critical (NC_P) that were observed to be non-critical during fault injection (NC_{FI}), i.e., $NC_P \cap NC_{FI}$.
- False Positives (FP), i.e., bits predicted to be critical (C_P) that were observed to be non-critical during fault injection (NC_{FI}), i.e., $C_P \cap NC_{FI}$.
- False Negatives (FN), i.e., bits predicted to be non-critical (C_P) that were observed to be critical during fault injection (C_{FI}), i.e., $C_P \cap C_{FI}$.

A perfect prediction approach would separate configuration bits only into True Positives (TP) and True Negatives (TN). However, since the prediction approach as discussed in this section does not take dynamic information about the design into account – which may be done as an extension by using, e.g., fault simulation as done in [124] – a certain amount of False Positives (FP) is expected. Even in the presence of a significant amount of false positives, however, the prediction approach may nevertheless be helpful to speed up fault injection experiments by allowing to prioritize potentially critical configuration bits (i.e., skip configuration bits that are unlikely to be critical). On the other hand, a significant amount of False Negatives (FN) may be detrimental as it would generate misleading results concerning the SEU sensitivity of a given design.

Finally, the time necessary to perform prediction is an important criterion when considering the use of the proposed bitstream prediction as part of, for example, a fitness function in a design-space exploration toolflow. Thus, each design was subjected to three iterations of timed criticality prediction on a Debian Linux 12 Workstation using an AMD Ryzen 7 5800X 8-core processor running at 3.8 GHz (bursting up to 4.7 GHz) and 64 GiB RAM.

4.5.2 Results & Discussion

As a result of the validation experiments described above, Table 65 shows the *Confusion Matrices* for each evaluated design in its three variants. Table 65a shows data obtained from fault injection testing on the first set of devices, while Table 65b shows the results from the second group that was not involved in characterization in any way. Columns labeled with C_{FI} denote bits determined to be critical during fault injection, while NC_{FI} denotes bits not observed critical during fault injection. Rows labeled C_p denote bits statically predicted to be critical, while bits predicted to be non-critical can be found in rows labeled NC_p. As described above, the cells at the intersection of a C_{FI} column and C_p row, and an NC_{FI} column and NC_p row are the true positive and true negative predictions, respectively. In contrast, cells at the C_{FI} columns and NC_p rows, and NC_{FI} column and C_p row denote false negative and false positive predictions.

It can be seen that a small number of false negatives ($NC_p \cap C_{FI}$) are generated, ranging from 0 in most cases to 6 bits in one case. In these rare cases, modeling the effects of bit flips on the circuit level – i.e., the assumption of benignity of specific interconnect faults or the prediction of CMF – does not hold.

Table 65: Confusion Matrices for Proposed Criticality Prediction for AMD 7 Series FPGAs

(a) Device Group 1

	Smallest			Medium			Largest		
B13m	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	3594	2729	6323	2432	2719	5151	453	1828	2281
NC_p	0	9727	9727	0	22979	22979	0	39328	39328
Σ	3594	12456	16050	2432	25698	28130	453	41156	41609
Add32	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	4242	4199	8441	3193	4901	8094	468	3119	3587
NC_p	0	12910	12910	3	31641	31644	0	61692	61692
Σ	4242	17109	21351	3196	36542	39738	468	64811	65279
Adler32	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	6476	17561	24037	4955	18439	23394	488	4949	5437
NC_p	1	49622	49623	6	99121	99127	0	215047	215047
Σ	6477	67183	73660	4961	117560	122521	488	219996	220484
SERV	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	11871	12272	24143	9132	10941	20073	3313	6415	9728
NC_p	0	38222	38222	0	83935	83935	3	156641	156644
Σ	11871	50494	62365	9132	94876	104008	3316	163056	166372

(b) Device Group 2

	Smallest			Medium			Largest		
B13m	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	3623	2700	6323	2452	2699	5151	465	1816	2281
NC_p	0	9727	9727	0	22979	22979	0	39328	39328
Σ	3623	12427	16050	2452	25678	28130	465	41144	41609
Add32	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	4279	4162	8441	3250	4844	8094	518	3069	3587
NC_p	0	12910	12910	0	31644	31644	0	61692	61692
Σ	4279	17072	21351	3250	36488	39738	518	64761	65279
Adler32	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	6540	17497	24037	5044	18350	23394	522	4915	5437
NC_p	0	49623	49623	2	99125	99127	0	215047	215047
Σ	6540	67120	73660	5046	117475	122521	522	219962	220484
SERV	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	11979	12164	24143	9232	10841	20073	3348	6380	9728
NC_p	0	38222	38222	0	83935	83935	0	156644	156644
Σ	11979	50386	62365	9232	94776	104008	3348	163024	166372

As observed in Section 4.4, faults critical on one device may not necessarily behave critically on all other devices. However, the absolute *number* of critical bits was observed to be relatively consistent across different devices. To quantify the impact of these effects on the four case-study designs, Tables 67 and 68 show an analysis of this effect on the twelve designs exhaustively tested to assess the accuracy of criticality prediction shown in Table 65. The columns $\cap$ and $\cup$ show the number of bits critical on *any* of the tested devices and bits critical on *all* of them, respectively. The mean value and its standard deviation (absolute and relative to the mean) are also shown in Tables 67 and 68. Finally, the amount of critical bits found in each device is shown. The cell color indicates the relative criticality identified in each device for the individual designs, with a relatively low number of critical bits colored purple. In contrast, a relatively high number is shown in yellow.

Finally, prediction times between 1.1 and 14.7 minutes were observed when conducting three iterations of timed prediction per design. As the designs are of different sizes (between 16050 and 220494 bits for which fault consequences need to be predicted) and different structures, Table 66 shows the mean value and standard deviation of the prediction speed, i.e., the number of bits predicted per second (expressed as bit/s).

Table 66: Mean Criticality Prediction Speed (Bits/Second) and Standard Deviation for Three Prediction Runs on Twelve Design Variants

	Variant		
Design	Smallest	Medium	Largest
B13m	251.9 ± 0.9	398.1 ± 14.7	520.6 ± 4.6
Add32	310.8 ± 0.7	467.1 ± 3.5	626.2 ± 4.6
Adler32	271.2 ± 2.7	393.8 ± 2.5	250.4 ± 1.1
SERV	434.6 ± 5.2	435.8 ± 6.3	527.5 ± 7.4

It can be seen that the prediction speed varies from design to design, with a mean value of 407.3 bit/s ± 116.2 bit/s (min: 249.2, max: 629.3). Thus, the average speed of BFAT (1.26 bit/s) is exceeded by at least by 197$\times$ (see Section 3.2.7[109]). Nevertheless, optimizing the implementation of the proposed prediction approach (e.g., by implementation in a compiled language instead of Python) may further improve the performance of the criticality prediction. This would improve the suitability for larger designs: Linearly extrapolating the mean observed prediction speed of the provided implementation (407.3 bit/s), prediction time using would exceed one hour for designs with more than 1.4 million essential bits. This does not take potential non-linearities more apparent in significantly more complex designs (e.g., higher-fanout nets) into account.

[109] "`benchmark_[1-4]`" in Table 12 refers to the "smallest" variants of the four designs evaluated in this chapter

Table 67: Criticality Data (Device Group 1)

Design	Variant	Tested	Critical Bits												
			∪	∩	Mean	Std. Deviation	Devices								
							1	2	3	4	5	6	7	8	9	10	11	12
B13m	Largest	41609	453	322	382.2	17.8 (4.7%)	381	371	403	344	370	417	392	379	379	382	385	383
Add32	Largest	65279	468	281	345.8	21.7 (6.3%)	332	340	377	318	332	395	359	337	349	331	336	343
Adler32	Largest	220484	488	336	415.1	17.6 (4.2%)	396	410	437	384	409	447	424	415	413	399	421	426
B13m	Medium	28130	2432	2144	2283.2	20.7 (0.9%)	2269	2279	2304	2246	2265	2330	2291	2279	2284	2277	2288	2286
Add32	Medium	39738	3196	2811	2960.1	22.6 (0.8%)	2943	2963	2978	2936	2952	3018	2951	2948	2957	2947	2947	2981
SERV	Largest	166372	3316	2978	3125.1	17.4 (0.6%)	3117	3121	3156	3093	3112	3157	3129	3125	3121	3120	3126	3124
B13m	Smallest	16050	3594	3327	3451.8	19.4 (0.6%)	3443	3434	3471	3423	3439	3498	3463	3454	3454	3441	3449	3452
Add32	Smallest	21351	4242	3808	3975.3	30.7 (0.8%)	3947	3974	4007	3962	3958	4058	3982	3957	3970	3951	3959	3979
Adler32	Medium	122521	4961	4363	4582.8	35.7 (0.8%)	4525	4576	4619	4578	4579	4671	4577	4565	4572	4558	4573	4600
Adler32	Smallest	73660	6477	5843	6088.7	32.0 (0.5%)	6045	6095	6116	6084	6064	6171	6091	6080	6090	6057	6081	6090
SERV	Medium	104008	9132	8512	8787.7	22.5 (0.3%)	8782	8774	8798	8766	8767	8848	8767	8783	8789	8785	8789	8804
SERV	Smallest	62365	11871	11193	11511.2	19.1 (0.2%)	11505	11517	11526	11495	11488	11548	11478	11513	11507	11509	11518	11531

Table 68: Criticality Data (Device Group 2)

Design	Variant	Tested	Critical Bits												
			∪	∩	Mean	Std. Deviation	Devices								
							1	2	3	4	5	6	7	8	9	10	11	12
B13m	Largest	41609	465	319	387.9	16.2 (4.2%)	356	407	399	397	392	404	396	378	392	381	393	360
Add32	Largest	65279	518	281	364.8	19.1 (5.2%)	345	402	386	353	361	375	381	341	360	351	377	346
Adler32	Largest	220484	522	340	427.9	18.7 (4.4%)	402	458	441	419	431	444	448	411	420	409	444	408
B13m	Medium	28130	2452	2132	2289.8	20.3 (0.9%)	2258	2320	2318	2283	2296	2308	2302	2274	2288	2273	2294	2263
Add32	Medium	39738	3250	2810	2977.6	21.6 (0.7%)	2974	3026	2981	2959	2985	2989	3001	2949	2968	2963	2982	2954
SERV	Largest	166372	3348	2962	3129.7	19.4 (0.6%)	3105	3167	3146	3135	3139	3135	3143	3096	3122	3126	3131	3111
B13m	Smallest	16050	3623	3309	3465.1	18.8 (0.5%)	3437	3483	3483	3458	3472	3490	3483	3457	3465	3457	3467	3429
Add32	Smallest	21351	4279	3812	3991.2	24.1 (0.6%)	3978	4041	3993	3970	3991	3980	4023	3964	3998	3986	4010	3960
Adler32	Medium	122521	5046	4362	4614.6	35.5 (0.8%)	4594	4682	4623	4563	4607	4613	4657	4563	4612	4608	4652	4601
Adler32	Smallest	73660	6540	5845	6117.6	31.3 (0.5%)	6095	6180	6139	6077	6117	6120	6156	6084	6107	6101	6143	6092
SERV	Medium	104008	9232	8464	8791.2	19.7 (0.2%)	8776	8833	8796	8781	8803	8797	8813	8765	8776	8782	8802	8771
SERV	Smallest	62365	11979	11161	11519.1	25.0 (0.2%)	11512	11554	11512	11526	11510	11511	11567	11478	11514	11509	11543	11493

4.5.3 Summary

The validation experiments' results indicated that the proposed criticality prediction approach can significantly reduce fault injection effort while incurring minimal false negatives. Considering the number of total and predicted-critical bits shown in Table 65, the proposed prediction approach leads to a significant reduction in bits to evaluate: In the maximal case (the *largest* variant of the *Adler32* design) AMD Vivado reports a total of 220484 essential bits. By injecting faults only in the 5437 bits predicted to be critical (C_p) by our approach, fault injection effort can be reduced by 98%. Even in the minimal case (the *smallest* variant of the *Add32* design), fault injection effort can be reduced by 60%.

While false positives are present in the evaluations, due to their small number compared to the correctly identified critical (and non-critical) bits, fault injection results acquired using this prediction can still be regarded as accurately representing the overall design's criticality for comparison between fitness results – especially when considering that device-to-device variations add noise to these results as well. While the first device group generated 13 false negatives, only two were generated when testing group 2. The relative uniformity of date and lot codes of group 2 may play a role in explaining this difference.

However, the prediction is still quite conservative and thus produces a considerable number of false positives ($C_p \cap NC_{FI}$). Two effects may play a part in these false positives: (1) faults in some resources have not been precisely modeled (e.g., DSP blocks); improving the quality of these fault models may reduce the number of false positives and (2) the prediction approach does not consider the design's functionality at runtime. Incorporating information about the design's runtime behavior (e.g., by including simulation results [124]) may reduce false positives.

Regarding inter-device variation of fault injection results, the results shown in Tables 67 and 68 illustrate once again that while there is considerable variation among the individual bits that are observed critical on the individual devices, the overall *numbers* of critical bits observed on the devices exhibits less variation. Except for design variants with a minimal amount of critical bits (the *Largest*) variants of *B13m*, *Add32*, and *Adler32*), the standard deviation is within ±1% of the mean number of critical bits. In both groups, specific devices consistently report more critical bits, possibly due to factors that let interconnect faults more often act critically in these devices than in other devices.

The similarity of results (concerning true positives, true negatives, and false negatives) when comparing the prediction results between the original device group (group 1), which partially consists of devices that played a role during interconnect characterization and PVT experiments, and the second device group (group 2) – which contains only devices that did not play a part in these experiments – indicate that no "overfitting" to the first device group took place during development of the criticality prediction rules. This indicates that the developed criticality prediction rules can be generalized to other AMD Artix 7 devices.

In terms of execution time, the implemented criticality prediction clearly outperforms the tool BFAT from the state of the art (see Section 3.2.7), making its use in fitness functions for design space exploration feasible.

4.6 Bitstream Patching for AMD 7 Series FPGAs

The characterization of *Input Antenna* interconnect faults Section 4.3 on AMD Artix 7 showed that one major contributor to critical *Input Antenna* faults appear to be the LOGIC_OUTS[0-23] source wires. These wires, in every interconnect tile in the AMD 7 Series technology, allow to connect the neighboring logic tile's outputs to the interconnect fabric. Most critical *Input Antenna* faults identified during characterization had an incident wire of this type. The subset of these 24 wires that is able to cause critical faults appeared to be dependent on the neighboring logic tile type, with different subsets being consistently critical for CLB, BRAM, DSP and other tile types. For logic tile types that span multiple rows of interconnect tiles, this subset depends on the vertical offset of the interconnect tile to the neighboring logic tile as well. Examples for CLB, BRAM, and DSP tiles can be seen in Tables 45, 46, 50a and 50b, respectively.

Furthermore, in all of these cases, an effect of the fault on the route-under-test was only visible when it transported "1" or the "toggle" pattern. Assuming, as observed in the characterization of *Conflict* faults, that faults that short-circuit connections (i.e., *Input Antenna* and *Conflict* faults) on AMD 7 Series predominantly manifest as Wired-AND conditions, this would imply that the LOGIC_OUTS[0-23] wires capable of critically affecting the route-under-test are driven to "0" when their associated resource is unused by the bitstream.

Characterization results for two interconnect tiles next to CLB tiles (see Tables 45 and 46) indicate LOGIC_OUTS[8-15] as the most consistent source of critical *Input Antenna* faults. These wires, as shown in Table 49, are directly connected to the outputs of LUTs in the adjacent CLB tile. Under the assumption that these wires are statically driven to "0" if the respective LUT is unused, inverting the initialization pattern of these resources in the bitstream should cause them to be driven to "1" instead.

Applying these patches can be done with the help of the bit-to-function mapping as well as placement-and-routing information derived during criticality prediction (see Section 4.5). Patching the affected faults requires first identifying all *Input Antenna* faults with an incident wire of type LOGIC_OUTS in interconnect tiles next to CLB tiles. In a second step, the LUTs connected to the respective LOGIC_OUTS wires must be identified by referencing the device's interconnect documentation. The combination of LUT name, slice, and CLB tile completely identifies the configuration bits to patch.

As seen from the example of LUTs, the prerequisite for patching *Input Antenna* faults in this way is that a set of configuration bit settings exist for an unused resource that set as many of its outputs (connected to the LOGIC_OUTS wires in the adjacent interconnect tile) to "1" instead of "0". Identifying this combination requires a per-resource model for the effects of each of their configuration bits on the resources' outputs.

A more complex resource for which patching is potentially possible are the DSP blocks in AMD 7 Series FPGAs. It can be seen in the characterization results shown in Table 50b that almost all LOGIC_OUTS wires connecting to an unused DSP tile are candidates for causing critical *Input Antenna* faults. Each DSP slice generates four CARRYOUT outputs, 48 P (result) outputs, one true PATTERNDETECT output, one inverted PATTERNBDETECT output, as well as an UNDERFLOW and an OVERFLOW output for a total of 56 outputs. A DSP tile contains two DSP slices, thus explaining the 112 critical *Input Antenna* faults involving LOGIC_OUTS wires connecting to the five interconnect tiles adjacent to the DSP tile shown in Table 50b. This indicates that in an unused DSP slice, all 56 outputs are driven to "0".

All outputs of a DSP slice can be either driven from a register or directly from the Arithmetic Logic Unit (ALU) integrated into the DSP block. Project XRAY's configuration documentation for DSP

slices contains multiple bits that are assumed to be active-low, enabling the corresponding register if set to “0”. For example, the bit DSP_L.DSP48.DSP_0.ZPREG[0] is assumed to enable the register for the 48-bit P output port (the ALU result port) when set to “0”. As unused configuration bits are assumed to be set to “0” by the AMD Vivado’s bitstream generator, and the effects of the P port being set to “0” were seen during characterization, the assumption follows that the corresponding registers are initialized to an all-“0” pattern at powerup. The bitstream documentation does not indicate configuration options for the initialization value of this register. However, inverting the concerned slice’s *.ZPREG[0] configuration bit to “1” drives the P port directly from the DSP slice’s ALU, which may be coerced to generate an all-“1” pattern as shown in the remainder of this section.

Several other registers optionally driving inputs to the DSP slice are configured in a similar way: ZALUMODEREG[0] for registering the ALU mode input, Z[AB]REG_2_[AB]CASCREG_1 for registering cascading inputs, ZCREG for registering the C input port, and ZOPMODEREG for registering the DSP operation mode input are all inverted. It is assumed that input ports for which the registers remain enabled drive “0” to their respective inputs, while all input ports for which registers are disabled are driven by the value of the connected input signal. As all logic inputs to an unused DSP slice are assumed to be driven by an unused interconnect resource, they are considered to be driven to “1” by the interconnect wire’s bus hold circuit. Certain inputs to the DSP slice can be inverted via configuration bits, such as ZIS_ALUMODE_INVERTED[0-3] for inverting the respective bits of the four-bit-wide ALUMODE input when set to “0”. In contrast, for some inputs, DSP tiles also allow to tie specific inputs to “0” or “1” via configuration bits. Finally, the values of the DSP slice’s 48-bit-wide PATTERN and MASK inputs can be configured via the bitstream directly.

By combining inverter configuration bits and static interconnect settings allowing to drive static levels to certain inputs, setting ALUMODE[3:0] to 0000_B configures the DSP-block’s ALU to calculate the following operation [230]:

$$\{CARRYOUT, P\} = Z + X + Y + CIN$$

CARRYOUT[3:0] is the ALU’s 4-bit-wide carry output port, while P[47:0] is the ALU’s 48-bit-wide result port. The input values of this operation are set as follows:

- Setting the configuration bits ZCARRYINREG and ZIS_CARRYIN_INVERTED both to “0” causes the single-bit signal CIN to be driven from the inverted “0”-initialized carry input register, and thus to “1”.
- The X, Y, and Z 48-bit wide inputs to the ALU are fed by multiplexers controlled by the OPMODE[6:0] input:
  - By setting OPMODE[1:0] to 11_B, X is set to a concatenation of the A[23:0] and B[23:0] input ports that are unconnected and thus assumed to be pulled to all-“1”s.
  - By setting OPMODE[3:2] to 10_B, Y is statically set to $\mathrm{FFFFFFFFFFFF}_H$.
  - Setting OPMODE[6:4] to 000_B sets Z to 0.[110].

Thus, the ALU combinationally calculates the operation

$$\{CARRYOUT, P\} = 0 + \mathrm{FFFFFFFFFFFF}_H + \{\mathrm{FFFFFF}_H : \mathrm{FFFFFF}_H\} + 1$$

This results in CARRYOUT[3:0] equaling 0001_B and P[47:0] equaling $\mathrm{FFFFFFFFFFFF}_H$.

[110] As OPMODE[6] as well as ALUMODE[3:2] can be set directly by the bitstream, the corresponding inversion bits are assumed unused

Finally, setting MASK[47:0] and PATTERN[47:0] both to $\mathrm{FFFFFFFFFFFF}_H$, the PATTERNDETECT output is set to "1" as the matching all-"1" pattern is present on output port P. As PATTERNDETECT and the inverted PATTERNBDETECT port are mutually exclusive, PATTERNBDETECT cannot be set to "1" at the same time. As driving the outputs OVERFLOW and UNDERFLOW requires that the output register for port P is activated, they cannot be set to "1" in this way.

In total, the configuration above sets 50 of the 56 output ports to "1". *Input Antenna* faults occurring with CARRYOUT[3:1], PATTERNBDETECT, OVERFLOW and UNDERFLOW as incident wires, however, will still be critical.

Other resources such as BRAM and PCIe blocks do not allow to set their outputs combinationally as is the case with the DSP slices.

BRAM: In these slices, the LOGIC_OUTS[...] wires driven to "0" are predominantly FIFO status signals (e.g., RDCOUNT[9:0] indicating the number of words present in the FIFO), which can only be set by interacting with the FIFO logic. Dynamic patching of unused BRAM cells would thus require setting appropriate control signals and generating clock cycles at the BRAM block's interface. While this is possible either by altering the netlist or RTL design or by run-time configuration access, this method would lead to significant overhead when compared to the purely static patching for CLB and DSP tiles.

PCIe: The AMD 7 Series PCIe block is a complex hard IP block whose internal state cannot be manipulated in a simple way via its logic inputs to change its outputs to "1". However, it was discovered in experiments carried out in the context of this dissertation that pulling two undocumented inputs to "0" (IMUX19 in the INT_R tile at offset +23 and IMUX12 in the INT_R tile at offset +24) appears to enable a scan interface (potentially to a wrapper boundary register such as described in [231]). Toggling the CLK[0,1] inputs in the INT_R tiles at offsets +12 (USERCLK,USERCLK2) and +11 (PIPECLK,DRPCLK) allows to shift in a pattern that successively sets the PCIe tile's outputs to "1". This is likely because an unused routing resource drives the scan data input signal which is thus pulled to "1" by hold circuitry.

The necessary setup is depicted in Figure 37. As the connections to IMUX19 and IMUX12 are undocumented and thus not routable by Vivado, the bitstream must be patched to connect these signals to GND_WIRE and thereby drive them to "0". Dynamic configuration access can be used to toggle the clock signals between a connection to GND_WIRE and a floating connection (i.e., pulled to "1" by the hold circuitry) without connecting a clock signal.

However, as the PCIe tile needs dynamic attention (i.e., generating clock cycles on its four clock inputs) for shifting in a less disruptive pattern at the output, it increases the demands placed on the design to be implemented, requiring at least a clock generator and counter unit for generating the shift clocks. Furthermore, as shown in Figure 37, specific routing resources are required to implement the patch described above, which are then unavailable for other routing. Other resources (e.g., in clock management or SERDES tiles) may also be patchable but require further experimentation.

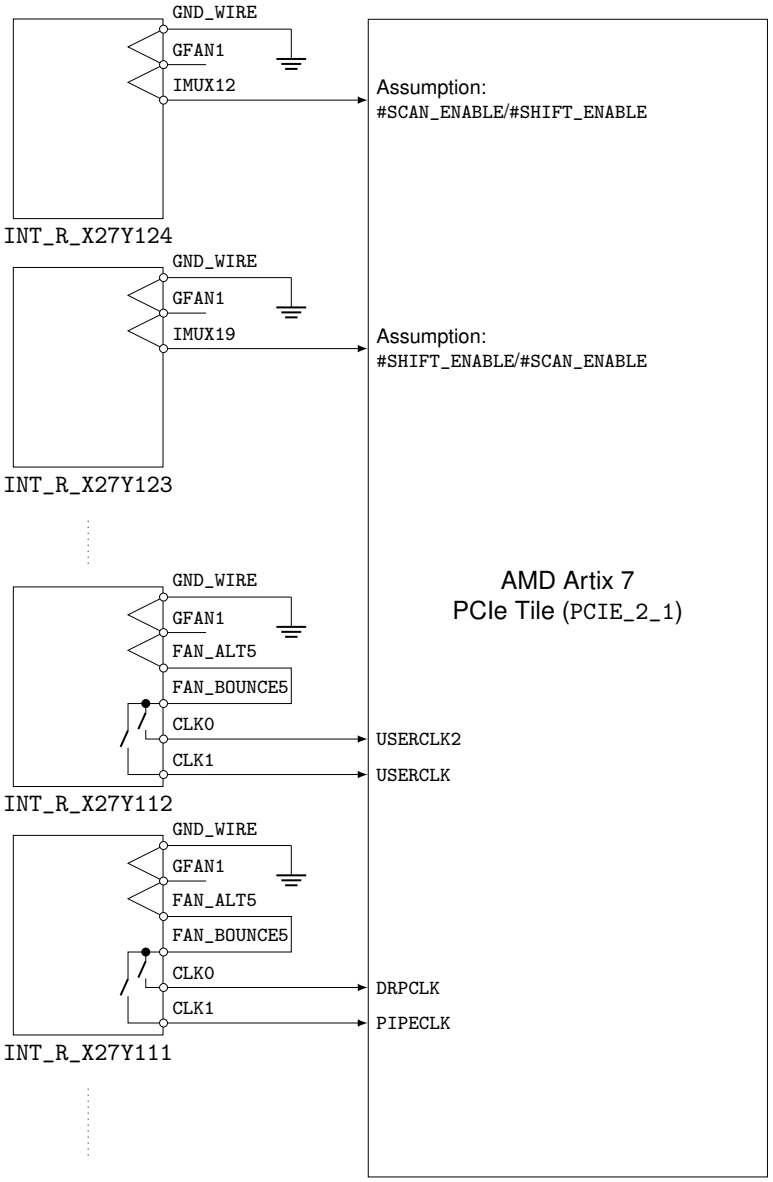


Figure 37: Dynamic Patching of Unused PCIe Tile in AMD Artix 7 FPGAs

4.6.1 Validation Experiments

Three experiments were designed to validate the bitstream patching approach for *Input Antenna* faults in AMD Artix 7 FPGAs:

- To initially validate the proposed patching concept, all concerned LUTs in the bitstreams generated for the *Input Antenna* characterization experiments in the CLB-adjacent tiles `INT_L_X26Y70` and `INT_R_X27Y70` in an AMD Artix 7 xc7a35t FPGA (see Section 4.3) were patched. This was done by (1) identifying the unused LUTs in the adjacent CLB tile and (2) setting the respective bitstream bits responsible for these LUTs' truth table to all-"1" patterns as described above. The *Input Antenna* characterization experiments were then repeated on the same set of four devices and under the same set of environmental conditions in order to compare the resulting critical set of incident wires in *Input Antenna* faults.

- To demonstrate the applicability of the patching approach to logic designs and to investigate the effect on the total number of critical bits in these circuits, the *smallest* variants of the four designs described in Section 4.5 were patched described above (i.e., unused LUTs and DSP blocks). The designs were then subjected to an exhaustive fault injection run on two groups of twelve devices each, and the resulting numbers of critical bits were compared.
- To demonstrate the principal viability of dynamically patching the outputs of a PCIe tile via the discussed undocumented boundary scan feature, a system for evaluating this method was constructed for an Arty A7 FPGA board. The system contains a MicroBlaze CPU, access to the device configuration via ICAP, and a monitor circuit for identifying the state of all 1152 outputs of a PCIe tile. In four FPGA devices, this system then dynamically modified the device's bitstream by partial reconfiguration to (1) activate the PCIe tile's discovered scan feature and (2) generate 2048 clock cycles on the tile's clock inputs simultaneously while also monitoring the state of the tile's outputs. The development of this count over the generated clock cycles then (1) demonstrates the discovered scan feature and procedure can produce high *population-count* patterns (i.e., patterns with a large number of outputs set to "1") at the PCIe tile's outputs, and (2) how many clock cycles are needed to generate the maximum population count at the output.

4.6.2 Results & Discussion

Comparing the non-patched systematic *Input Antenna* evaluation of tile `INT_L_X26Y70` neighboring a CLB tile (see Table 45) and the patched evaluation of the same tile (see Table 69) shows a significant reduction in critical *Input Antenna* faults: While in the non-patched version, 1451 faults were critical on any device, repeating the same experiments with patched bitstreams under similar conditions yielded only 24 critical *Input Antenna* faults. A similar reduction was observed for the tile `INT_R_X27Y70`: from 1450 critical *Input Antenna* faults before patching (see Table 46), to only 28 critical faults after patching (see Table 70) . Consistent with the underlying LUT-related assuptions, the `LOGIC_OUTS[8-15]` incident wires do no longer cause critical faults in both evaluated interconnect tiles after patching. This constitutes a 98% reduction in incident wires that are capable of causing critical *Input Antenna* faults.

Table 69: Evaluated vs. Critical *Input Antenna* Faults by Incident Wire in `INT_L_X26Y70` in AMD Artix 7 xc7a35t after Bitstream Patching

Incident Wire	Tested	Critical Combinations					
				Devices			
		∪	∩	1	2	3	4
`BYP_BOUNCE1`	77	6	3	6	6	3	6
`GND_WIRE`	6	6	6	6	6	6	6
`NR1END1`	122	6	1	6	2	1	6
`WW4END1`	114	6	1	6	3	4	6
178 Other Wires	19480	0	0	0	0	0	0
Total	19799	24	11	24	17	14	24

Table 70: Evaluated vs. Critical Faults *Input Antenna* by Incident Wire in INT_R_X27Y70 in AMD Artix 7 xc7a35t after Bitstream Patching

Incident Wire	Tested	Critical Combinations					
				Devices			
		∪	∩	1	2	3	4
BYP_BOUNCE1	77	6	0	6	2	0	5
GND_WIRE	6	6	6	6	6	6	6
LOGIC_OUTS16	163	1	0	1	0	0	0
LOGIC_OUTS23	163	1	0	1	0	0	0
NR1END1	122	6	1	3	5	1	5
SE6END1	108	1	0	1	0	0	0
SE6END2	108	1	0	1	0	0	0
WW4END1	114	6	0	6	4	0	0
174 Other Wires	18969	0	0	0	0	0	0
Total	19830	28	7	25	17	7	16

While a 98% reduction in potentially critical incident wires is impressive, the reductions reported above concern the theoretical maximum reduction in potentially critical fault sites, of which only a fraction will be present in an actual design. Table 71 shows the number of fault sites that can be removed from each case-study design used to evaluate the proposed bitstream patching approach. The reduction in fault sites is achieved by patching the associated number of LUTs and DSP blocks in its bitstream. These fault sites must be seen in the context of the total numbers of bits (see Table 73), where they make up only 2–4% of predicted-critical bits.

Table 71: Extents of Bitstream Patching for Case-Study Designs

Design	Removed Fault Sites	Patched LUTs	Patched DSP Blocks
B13m	242	150	2
Add32	348	216	3
Adler32	807	347	2
SERV	488	283	6

Furthermore, not all of these fault sites cause critical faults when the associated bit is flipped due to masking effects. The original criticality, as well as the criticality after re-evaluating the patched designs on device groups 1 and 2, are shown in Table 72. It can be seen that bitstream patching removes (almost) all of the patchable critical faults. However, two bits remain critical in the *SERV* design. When flipped, these bits cause *Input Antenna* faults that erroneously connect a DSP's UNDERFLOW and CARRYOUT1 outputs as incident wires to PIPs in this design. This behavior is expected considering the discussion of DSP patching at the beginning of this section, as neither of these signals can be set to "1" by static patching. Nevertheless, the patching effectively reduces the criticality of the patchable bits.

Table 72: Critical Faults in Patchable Bits in Case-Study Designs under Fault Injection

Design	In Original Bitstream	In Patched Bitstream	
		Device Group 1	Device Group 2
B13m	156	0	0
Add32	161	0	0
Adler32	247	0	0
SERV	304	2	2

To further analyze the effectiveness of bitstream patching as described above, the overall criticality of the designs before and after patching was analyzed in order to exclude the possibility that patching the faults discussed above leads to an increase in criticality elsewhere. Table 73 shows the results of this comparison. It can be seen that patching LUTs and DSPs reduced the number of *Critical* bits by 3.5% (*B13m*), 1.9% (*Add32*), 3.56% (*Adler32*), and 2.3% (*SERV*) on device group 1, and by 3.6% (*B13m*), 1.1% (*Add32*), 3.0% (*Adler32*), and 2.2% (*SERV*) on device group 2.

Table 73: Confusion Matrices for Criticality Prediction for AMD 7 Series FPGAs before and after Proposed LUT and DSP Patching

	Original (Group 1)			Patched (Group 1)			Original (Group 2)			Patched (Group 2)		
B13m	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	3594	2729	6323	3468	2855	6323	3623	2700	6323	3491	2832	6323
NC_p	0	9727	9727	0	9727	9727	0	9727	9727	4	9723	9727
Σ	3594	12456	16050	3468	12582	16050	3623	12427	16050	3495	12555	16050
Add32	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	4242	4199	8441	4160	4281	8441	4279	4162	8441	4231	4210	8441
NC_p	0	12910	12910	0	12910	12910	0	12910	12910	0	12910	12910
Σ	4242	17109	21351	4160	17191	21351	4279	17072	21351	4231	17120	21351
Adler32	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	6476	17561	24037	6246	17791	24037	6540	17497	24037	6347	17690	24037
NC_p	1	49622	49623	1	49622	49623	0	49623	49623	2	49621	49623
Σ	6477	67183	73660	6247	67413	73660	6540	67120	73660	6349	67311	73660
SERV	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ	C_{FI}	NC_{FI}	Σ
C_p	11871	12272	24143	11595	12548	24143	11979	12164	24143	11721	12422	24143
NC_p	0	38222	38222	1	38221	38222	0	38222	38222	0	38222	38222
Σ	11871	50494	62365	11596	50769	62365	11979	50386	62365	11721	50644	62365

Finally, Figure 38 shows the *population count* (i.e., "1"s count) among the `LOGIC_OUTS` outputs of a PCIe tile over a range of conducted shift operations after statically patching the bitstream to pull the undocumented shift enable inputs to "0". Experiments were conducted on four Artix 7 xc7a35t FPGAs. A maximum population count of 1129 was recorded on all devices after generating 1134 clock cycles. The example shows that the assumptions made about the PCIe block discussed above are accurate enough to set 1129 of 1152 outputs (98%) to "1", thereby decreasing the potential of critical *Input Antenna* faults in the vicinity of an unused PCIe tile in AMD Artix 7 FPGA.

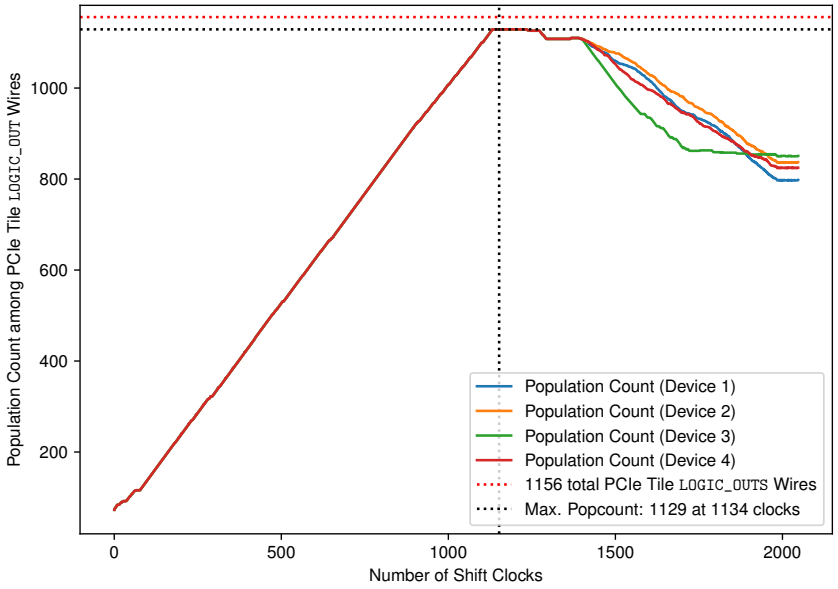


Figure 38: PCIe Tile Dynamic Patching Results: Population Count among LOGIC_OUTS Wires vs. Generated Shift Clock Cycles

4.6.3 Summary

This section illustrates that certain *Input Antenna* faults in AMD Artix 7 FPGAs that originate (via their incident wire) from unused logic resources can be mitigated at zero overhead – just by adapting the initialization of these logic resources. The working principle is demonstrated in systematic *Input Antenna* characterization experiments in two interconnect tiles adjacent to CLB tiles, where the proposed bitstream patch inverts the offending LUTs' initialization string, causing them to statically drive their output wire to "1" instead of "0". Furthermore, an approach is described to apply the same concept to DSP tiles. In combination, patching LUTs and DSPs in four benchmark designs resulted in reductions of critical bits of 1.1% to 3.6% when evaluating these designs under exhaustive fault injection on two groups of devices without significantly increasing criticality elsewhere. This is comparable to the average soft-error-rate reduction of 3.67% for a custom 6-input LUT FPGA technology[111] reported in [156], where designs are patched on the post-implementation netlist level by inverting specific LUTs to reduce the boolean difference of particular signals. All critical *Input Antenna* faults concerning LUT and almost all relating to DSP outputs were eliminated in these designs.

Not all resources can be patched statically in the way this is possible for LUTs and DSPs as their internal state is not directly affected by the FPGA's configuration. These cases (including BRAM and hard-IP blocks) require "dynamic" patching, i.e., run-time interaction with the logic resources to be brought to a state that generates fewer potentially critical fault sites for the surrounding logic. As an example, a method of dynamically patching the PCIe hard IP embedded in AMD Artix 7 devices is discussed and evaluated. However, due to the required ability to perform run-time interaction, dynamic patching incurs some overhead compared to static patching. Thus, its effect on actual designs was not evaluated in the fault injection evaluation of the four presented case study designs.

While the effect in terms of total design reliability is small, the proposed *static* patching method for LUT and DSP achieves these improvements while incurring no overhead. Furthermore, the

[111] AMD 7 Series FPGA likewise employ 6-input LUTs

evaluations performed in this section provide further evidence indicating that critical *Input Antenna* and *Conflict* interconnect faults are caused by wired-AND effects between the shorted wire segments. This behavior may be integrated into future fault simulation approaches to improve the accuracy of bitstream-level criticality prediction.

Other FPGA technologies or toolchains may offer similar opportunities for bitstream patching. However, the interconnect characterization on Lattice's ECP5 technology, whose interconnect is architecturally similar to that of AMD's 7 Series devices, revealed no equivalent opportunities concerning LUT patching: As LUTs are initialized using all-"1" initialization strings by the employed open-source *NextPNR-ECP5* implementation tool, no equivalent critical *Input Antenna* faults stemming from unused LUTs have been identified in the device characterization carried out in Section 4.3. However, patching DSP blocks in the ECP5 technology may be possible and lead to similar results.

4.7 Conclusion

This chapter presents considerations for a bitstream-to-netlist criticality prediction approach for single-bit faults in SRAM-based FPGAs as a building block for a comprehensive reliability improvement toolflow. Starting from an analysis of the target design's bitstream and post-implementation netlist, the approach separates configuration bits potentially critical for the design's correct functionality from those likely to behave benignly in the case of a bit flip. The presented approach is based on bitstream-derived modeling for faults in logic and interconnect resources, which, due to unpublished documentation by the device vendors, relies on (1) third-party bitstream documentation and (2) assumptions about the underlying circuit-level implementation.

This fault modeling is further evaluated and refined based on a proposed systematic fault characterization procedure geared towards deriving the in-circuit behavior of interconnect faults. Experiments conducted according to this procedure revealed the behavior of interconnect faults on AMD Artix 7, Lattice iCE40, and Lattice ECP5. While *Open* faults commonly manifest themselves as stuck-at faults, both *Input Antenna* and *Conflict* faults result in Wired-AND (AMD Artix 7, Lattice ECP5) and Wired-OR (Lattice iCE40) effects.

This characterization is also extended to the influence of inter- and intra-device variations as well as temperature and supply voltage on the outcome of fault injection experiments. To the author's best knowledge, both extensive and systematic characterization and analysis of PVT effects were previously not reported in literature. Significant PVT effects were discovered for AMD 7 Series FPGAs, where fault injection results may vary with (1) the device on which a fault injection experiment is performed, (2) the *location within the device* where the DUT is placed, (3) with environment temperature during tests, and (4) with supply voltage. Temperature (in the range of 20 °C– 50 °C, and thus well within the device's recommended temperature range [223]) mainly influenced the behavior of *Open* faults in these devices. Fewer critical bits were observed at higher temperatures and higher operating voltages in these devices. Lowering the operating voltage to and beyond the lower limit of the recommended operating range (i.e., "undervolting" which is commonly explored for energy efficiency improvement) appears to exacerbate both inter- and intra-device variation in criticality of *Input Antenna* faults in AMD 7 Series devices. In Lattice iCE40 devices, inter-device variation was also observed; however, no temperature-related component was identified within the evaluated temperature range. In a preliminary evaluation on Lattice ECP5 devices, no inter-device variation was identified in *Open*, *Input Antenna* or *Conflict* faults. Considering the possibility of inter-device variation in fault injection experiments is especially crucial, as state-of-the-art fault injection approaches have started to speed up fault injection by

parallelizing it on multiple FPGA devices [140, 141].

Combining the information gained from the steps described above allows for the implementation of a criticality prediction approach for the popular AMD 7 Series FPGA family. This approach can, from the bitstream, the vendor's *Essential Bits* pre-evaluation, and post-implementation netlist, form predictions of the criticality of each configuration bit. In evaluation on several case-study designs, this prediction was able to exclude between 60% and 98% of *Essential Bits* as benign when flipped while incurring minimal false negatives, thereby significantly reducing fault injection effort. It works at an average prediction speed of 400 bits/second (min: 249.2, max: 629.3), which constitutes an improvement of at least 197× vs. the average speed of the state-of-the art fault analysis tool BFAT [121] of 1.26 bit/s (see Section 3.2.7).

Finally, the extensive fault characterization enabled the implementation of a bitstream patching approach that is able to prevent malignant consequences in certain *Input Antenna* interconnect faults in AMD 7 Series FPGAs merely by altering the initialization state of unused LUTs and DSP elements. In evaluation, this patching approach reduced the number of critical faults in four benchmark designs by between 1.1% and 3.6% at zero resource overhead. In addition to this *static* patching, a *dynamic* patching approach for the PCIe block found in AMD 7 Series devices by enabling an undocumented Design For Test (DFT) mode was described.

As a summary, Table 74 evaluates the presented bitstream-level criticality analysis approach and its underlying analyses in the same format as done for approaches in the state of the art in Section 3.2. Comparing these evaluations, it can be seen that the approach(es) presented in this dissertation significantly outperform state-of-the-art approaches, especially by devoting attention to the systematic characterization of fault consequences in the targeted FPGA technologies, validation of the formed predictions, and by performing an in-depth analysis of PVT effects on fault injection results. Finally, most approaches limit themselves to AMD devices, so it was previously unknown whether their concepts are transferrable to other manufacturers' technologies. In this work, two additional in-production SRAM-based FPGA device families (Lattice iCE40 and Lattice ECP5) are considered, demonstrating the applicability of the described concepts to other devices.

Nevertheless, the following limitations of the presented approach offer room for improvement:

- While the prediction part of the approach generates few false negatives, it is still pessimistic – characterized by a significant portion of false positives. More precise fault models, e.g., for DSP and BRAM elements, may improve its accuracy.
- The presented approach only considers static information about the design. Additionally considering run-time information (e.g., from gate-level simulation runs) and combining it with characterization information (e.g., Wired-AND behavior of most interconnect faults) may further improve prediction accuracy. An example from the state of the art integrating this information into a prediction approach is shown in [124].
- While the approach considers devices by the market leader AMD and low-power, low-cost device vendor Lattice, neither the presented approach nor the state-of-the-art devoted appropriate attention to Intel/Altera devices – the second-largest FPGA vendor in terms of market share – or other device vendors. This may be due to missing device documentation. Potentially interesting targets for future exploration are Intel/Altera's Cyclone V series, for which Project Mistral[112] provides rudimentary device documentation, as well as the GateMate series by the European FPGA vendor CologneChip, for which Project Peppercorn[113] provides

[112] See `https://web.archive.org/web/20250909164258/https://github.com/Ravenslofty/mistral`, accessed 2026-01-05

[113] See `https://web.archive.org/web/20250831094915/https://github.com/YosysHQ/prjpeppercorn`, accessed 2026-01-05

Table 74: Assessment of Bit-Level Reliability Analysis Approach Proposed in this Dissertation

Requirement	Evaluation	Explanation
REQ:BIT:MAPPING	●	See Section 4.1.1 and Section 4.2.
REQ:BIT:MODELING:LOGIC	●	*Don't Care* Analysis for LUTs (see Section 4.2.1)
REQ:BIT:MODELING:INT	●	Modeled for AMD Artix 7, Lattice iCE40 and Lattice ECP5 technologies (see Section 4.2.2)
REQ:BIT:MODELING:SPECIAL	◑	Special-function resources are associated with the mapped cells (see Section 4.2.3).
REQ:BIT:BITWISE_PREDICTION	●	See Section 4.5.
REQ:BIT:CHARACTERIZATION	●	*Input Antenna* & *Conflict* faults characterized on AMD Artix 7, Lattice iCE40 and Lattice ECP5; *Open* faults characterized on Lattice ECP5.
REQ:BIT:VALIDATION	●	Validated on 24 AMD Artix 7 devices using twelve benchmark design variants with minimal false negatives; reducing fault injection effort by between 61% and 98%.
REQ:BIT:PVT	●	Evaluation of PVT effects on *Open* and *Input Antenna* faults indicated and analyzed inter- and intra-device variation with temperature and supply voltage in criticality of interconnect faults
REQ:BIT:DEVICES	●	Three in-production technologies (AMD Artix 7, Lattice iCE40, Lattice ECP5) are considered in bitstream analysis, fault modeling, and characterization.
REQ:BIT:RUNTIME	●	Mean Prediction Speed of 400 Bits/Second observed.
Approach Score	9.5	

device documentation. Additionally, documentation for QuickLogic's EOS-S3 SoC FPGA is provided by the device vendor[114].

- While the observed mean prediction speed of 400 bits/second outperforms the BFAT tool available in the state of the art, predicting fault consequences for a design containing 1.4 million essential bits would still require approximately one hour of prediction time. Thus, larger designs may still benefit from improvements in prediction speed.

[114] See `https://web.archive.org/web/20250831094901/https://github.com/QuickLogic-Corp/EOS-S3`, accessed 2026-01-05

Chapter 5

Design-Space-Exploration for FPGA Reliability Optimization

While full-design redundancy (e.g., TMR) has been shown to be effective at protecting a design against single-bit flips, it also incurs a large overhead in terms of FPGA resource usage. Partial TMR aims at exploiting a trade-off between resource usage and SEU susceptibility by optimizing the size and selection of the redundant portion of the design for maximal SEU mitigation at minimal resource consumption.

While related work in this field has addressed this problem to a certain extent as discussed in Section 3.3, existing reliability-improvement approaches, be it placement-and-routing-based, selective-replication-based, DSE-based or approaches based on high-level synthesis, in summary, do not sufficiently address area-reliability optimization to generate resource-effective protection against configuration memory bit flips in SRAM-based FPGAs. Even fewer approaches in today's state of the art integrate effects related to the design's precise placement and routing in the suggested (partial) replications to be introduced. While the effects of single-event upsets in the configuration bits in an SRAM-based FPGA manifest themselves first and foremost on the bit level, which requires bitstream-level modeling to guide optimization efficiently, this is only done by a fraction of the analyzed approaches. Finally, even though significant PVT-related influence on fault injection results – and, thus, on fault mitigation performance in validation experiments and possibly in the field – have been uncovered as described in Section 4.4, none of the identified related works addresses inter- or intra-device variations.

Motivated by this gap in the state of the art, this chapter proposes an automated way of exploring the area-reliability design space. This proposed approach includes the following contributions:

(1) A novel cost- and reliability optimizing DSE approach for module-based partial TMR, generating a set of Pareto-optimal TMR solutions,

(2) a demonstration of the real-world applicability of the approach using an implementation as a toolflow for AMD's 7 Series FPGA technology that takes PVT effects into account,

(3) an in-depth evaluation of the toolflow comparing variants of reliability-related fitness evaluation based on static bitstream analysis, (statistical) fault injection, and a combination of these approaches that takes care in performing evaluations to identify the impact of inter-device PVT-related variations on the quality of optimization results.

The remainder of this chapter is structured as follows: Section 5.1 presents the proposed DSE approach based on the NSGA-II algorithm, bitstream analysis, and fault injection, of which an

implementation for AMD Artix 7 FPGAs is presented in Section 5.2. The experimental evaluation of this approach is described in Section 5.3, while results are presented in Section 5.4. The results are discussed and compared to related work in Section 5.5. Section 5.6 concludes this chapter and outlines directions for further improvement.

5.1 Proposed Design Space Exploration Approach

The intention of the design flow proposed in this work is to allow the implemented circuit to temporarily continue to function in the presence of an SEU until it is detected and corrected by periodic configuration scrubbing by protecting a selected subset of the design's modules using TMR. In a conventional design flow, the designer must manually select a subset of the design's modules deemed especially critical to be replicated. Selecting which submodules of a design to protect using TMR is often based on experience, with the combinations of modules to replicate being selected manually.

The granularity of an individual design's RTL or HLS-based (sub)module hierarchy depends on the design entry style that is applied by the designer – i.e., how much logic is placed in each module, and how granularly these modules are structured. Additionally, the synthesis and implementation tools may apply cross-hierarchy optimization or flatten the design, which modifies the design's original hierarchical structure. Nevertheless, related approaches in the state of the art, such as [179, 232], operate on the module-hierarchy level. These approaches aim to identify trade-offs between (resource-related) costs and (soft-error-mitigation-related) benefits of selecting a subset of modules to replicate using TMR. By assuming that such a trade-off exists at the module level, they implicitly assume that the modular hierarchy of the design captures the necessary differences in module-level size and reliability impact. Both approaches do this with success: [232] demonstrates that selectively protecting different subsets of a NEORV32 RISC-V CPU's modules with TMR results in varying cost and soft-error sensitivity properties. It is further shown in [232] that introducing redundancy in certain module combinations is significantly more effective than in others. Similar observations are made in [179]. Thus, in line with the approaches described in [179, 232], the hierarchy defined by an RTL designer or an HLS tool serves as the primary clustering information for the DSE approach proposed in this chapter.

Both approaches discussed above [179, 232] use isolated module-level fault injection results to decide which combinations of modules to replicate. However, the effects of replicating individual modules on the overall design reliability on the bitstream level are hard to predict: Replication of selected submodules of a design using TMR introduces additional logic cells into the design, necessarily leading to a placement and routing that differ between the original and partially redundant design variants. Even subtle changes in the routing of a design can impact the vulnerability of a circuit due to technology-dependent non-uniform SEU susceptibility of interconnect resources and dependence of fault consequences on the logic levels transported by the affected signals [3]. This difficulty is exacerbated by the complexity of the FPGA placement and routing problem in general. It is addressed using a variety of algorithms for placement (e.g., probabilistic placement using simulated annealing) and routing (e.g., sequential routing with rip-up and re-routing for each net) [199]. This makes the outcome of this process and its impact on reliability difficult to predict. Furthermore, the (optimal) selection of hierarchical submodules of a design such that the design's cost in terms of logic resources is minimized while maximizing the number of mitigated configuration bit errors can be seen as a type of Knapsack problem, famously known to be NP-complete [233].

To address these issues, we propose to use a genetic algorithm to automatically generate a set of

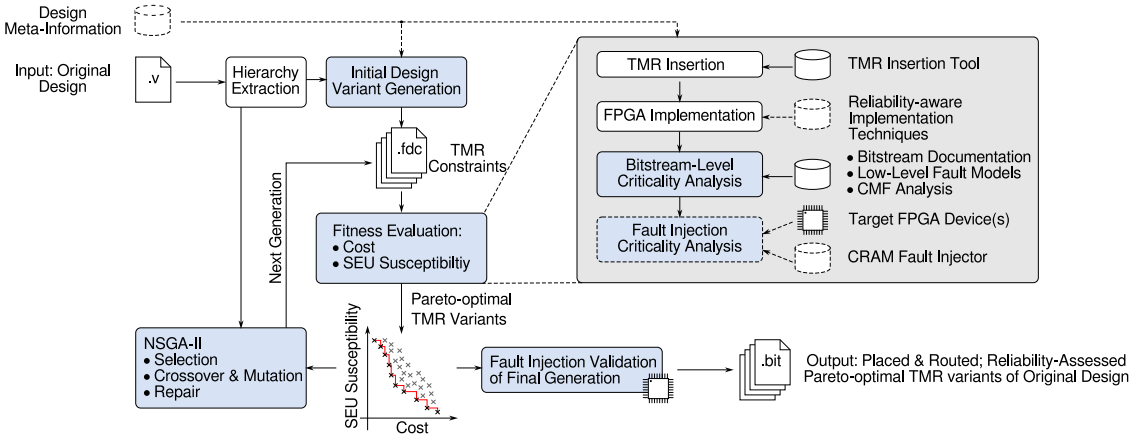


Figure 39: The Proposed DSE Toolflow

solutions to the TMR selection problem where each solution is Pareto-optimal with respect to (1) the cost of the solution in terms of FPGA resources (logic and routing) and (2) the vulnerability to bit flips in configuration memory in terms of critical configuration bits. In particular, the NSGA-II algorithm [173] has proven to be well-suited for this kind of problem in previous approaches in this field of application [172, 175].

Figure 39 provides an overview of the proposed TMR DSE toolflow. It aims to generate a set of selections of a design's hierarchical submodules. The design is assumed to be present as a hierarchical netlist (i.e., a directed tree of modules expressing hierarchical relations), either preserved from a previous synthesis step or pre-partitioned by a preceding tool. The optimization toolflow receives this netlist as its input, subsequently extracts the design's hierarchy from the netlist, and generates a set of starting solutions. This generation of the starting set may be based on information about the design that is available at this point. This may, for example, be fault simulation results, previous fault injection results, data flow analysis to identify feedback loops, or analyses carried out at a higher level of abstraction. A random starting set of solutions may be generated if no such information is available. Fixed points to be evaluated may be added to this set, e.g., a variant replicating the topmost module (i.e., full TMR). Starting from this initial set of solutions, the conventional flow of a genetic algorithm follows. After finishing the optimization process, the Pareto front of the final generation is validated using fault injection.

The following sections propose (1) an encoding of the partial TMR selection problem into chromosomes (see Section 5.1.1), (2) parameters and operations for the implementation of the genetic optimization process, and (3) a way of evaluating individual solutions' fitness (see Section 5.1.3).

5.1.1 Genetic Encoding of the Partial TMR Selection Problem

The modular hierarchy of a design can be represented as a tree, as illustrated in the example shown in Figure 40. To encode which of the design's hierarchical modules of the design shall be replicated using TMR into a chromosome, either a tree encoding or a flattened (i.e., serialized) tree representation can be used. In the former case, a tree T replicating the design's hierarchical structure is generated, with weights (i.e., genes $g_0, ..., g_{n-1}$) for each vertex encoding the decision to replicate the given node. In the latter case, the hierarchy tree is serialized into a vector V, where each element (i.e., gene) indicates the replication status for each element. This vector is free of structural information, which is retained in a separately stored hierarchy tree of the design. Figure 40 shows an exemplary design hierarchy tree, as well as options encoding replication

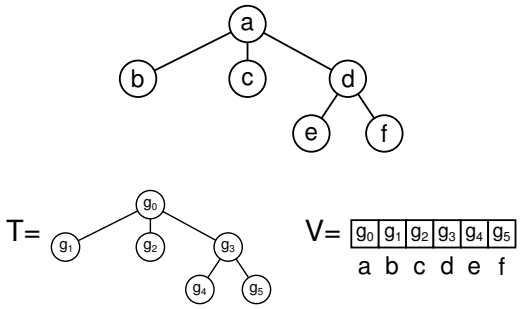


Figure 40: Design Hierarchy and corresponding Tree- and Vector-based Chromosome Encodings

decisions in a tree-based and a vector-based form.

In its most basic form, the genes $g_0, ..., g_{n-1}$ of a tree-encoded chromosome or the chromosome vector are boolean elements indicating whether the module should be replicated using TMR or not. If multiple forms of replication (e.g., optional special treatment of feedback loops [161] or NMR) are to be supported additionally, a vector *enumeration*-type elements can be used as a chromosome. To insert TMR, the chromosome is combined with the stored hierarchy of the design to generate module-level TMR constraints for a TMR insertion tool.

However, only certain combinations of modules may be feasible for replication. While *recursive* TMR (i.e., triplicated a module along with some of its already triplicated hierarchical constituents) has been proposed for intrinsically unreliable cells in IC fabrication processes [171], it may only offer limited additional reliability to protect against SEUs and is not generally supported by TMR insertion tools. Thus, the proposed toolflow eliminates these configurations from the evaluated solutions. As either the root of a subtree or a subset of its descendants can be replicated (that need to conform to this rule themselves recursively), the number of solutions for a given design hierarchy can be expressed as shown in Equation (5.1). Here, $S()$ is the recursive function to calculate the number of valid replication selections, g is the graph describing the design's hierarchy, n_i is the root node of the hierarchy subtree, and function $D()$ returns the root node n_i's set of descendant nodes. The function recurses until reaching a leaf of the tree, for which always both 0 and 1 are valid options (given that no ancestor is replicated).

$$S(g, n_i) := \begin{cases} 1 + \Pi_{n_j \in D(g,n_i)} S(g, n_j) & |D(g, n_i)| > 0 \\ 2 & \text{otherwise} \end{cases} \tag{5.1}$$

The number of valid solutions for the design hierarchy shown in Figure 40 can be calculated using Equation (5.1) as 21 out of 64 possible combinations of the six modules (including simplex as a valid option).

A chromosome violating the requirement described above may result from randomly generating initial chromosomes or from crossover and mutation. In this case, *repairing* the invalid chromosome is an option [234, 235]. For each conflict between a root and a submodule, two candidates for *more* correct chromosomes are generated (i.e., one with only the root replicated and one with the descendant replicated), which are themselves recursively corrected in case additional conflicts exist. Figure 41 shows an example of an invalid chromosome and a recursive repair process to arrive at valid chromosomes with respect to the hierarchy shown in Figure 40.

The effects of a valid chromosome on the TMR insertion process w.r.t. a circuit whose hierarchy is represented by the graph in Figure 40 can be seen on the left side of Figure 42. The right side shows the result of an invalid chromosome requiring that both the ancestor module d and its descendant e be replicated (i.e., recursive TMR). This would result in nine copies of e and nine

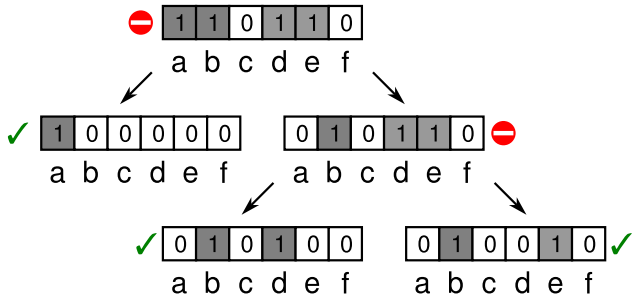


Figure 41: Recursive Repair of an *Invalid* Serialized Chromosome

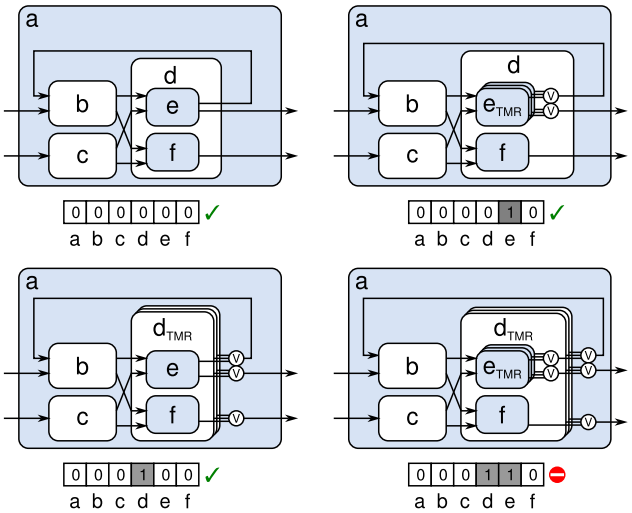


Figure 42: Effects of Chromosomes on TMR Insertion

voters for each of e's outputs within d as well as another set of voters external to d.

5.1.2 Genetic Algorithm Parameters and Operations

The performance of any optimization approach based on genetic algorithms depends on the selections made for the algorithm's parameters and operations. An implementation has to choose values for the applied population size and mutation rate. As [236] states, tuning these parameters to the problem at hand is necessary as there exists no parameter set that is optimal for all problems[115]. In this dissertation, a set of parameters is chosen in Section 5.2 to allow initial validation of the approach; it is not claimed that these parameters are optimal for the problem at hand. Furthermore, the crossover and mutation operations have to be defined within the context of the chosen chromosomal encoding of the candidate solutions. Finally, conditions for halting the optimization process have to be defined. This may be done once convergence is reached (e.g., judged by a stopping criterion such as hypervolume growth [237]), or after a set number of generations [238].

If a graph-encoded chromosome is chosen, crossover of two design variants amounts to swapping parts of the two tree's weights according to the chosen crossover strategy. For example, one or more subtrees may be randomly chosen among the nodes of the hierarchy tree, and swapped between the two parent chromosomes. The mutation operation may then be implemented by

[115] Selecting optimal genetic algorithm parameters is a meta-optimization problem that may be solved by a genetic algorithm in itself [236]

iterating over the offspring trees' vertices and flipping the associated weight from 0 to 1 or vice-versa with a selected mutation probability. An advantage of this encoding is that the crossover operation operates entirely within the semantics of the (replication) hierarchy tree: Crossover between two graph-encoded chromosomes replaces the decision how to mitigate soft errors within a well-defined hierarchical submodule of the design hierarchy. On the other hand, crossover between serialized trees potentially affects multiple hierarchical subtrees at once. These effects may impact diversity and overall optimization performance.

An example for crossover between two graph-encoded chromosomes is shown in Figure 43. The node "c" is chosen as a crossover point, and is highlighted in red (ancestor 1) and blue (ancestor 2). The binary weight of the graph vertices is indicated by them being drawn filled-in (when the weight is 1) or empty (when the weight is 0). The vertex color indicates if the node's weight assignment stems from ancestor 1 (red) or ancestor 2 (blue). It can be seen that after performing crossover by swapping subgraphs between the ancestors, both offspring individuals' weight assignments are a mixture of the original assignments in both ancestors.

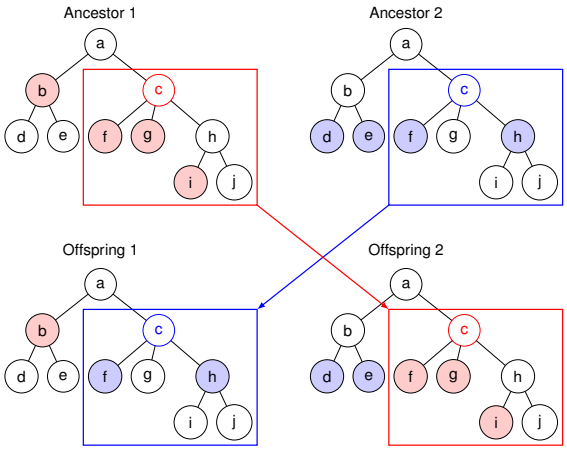


Figure 43: Crossover Operation for Graph-Encoded Chromosomes

Various crossover and mutation operations are defined for problems with chromosomes encoded as binary vectors, having both the advantage of being straightforward to implement and being widely-used. Conventional crossover operations for vector-encoded chromosomes are single- and two-point crossover, which cut the ancestor chromosomes at one or two randomly selected points, and generate the offspring chromosomes by piecing the resulting chromosome segments together by alternating between segments from each ancestor. For this purpose, the hierarchy tree must first be flattened into a vector, which may be done breadth-first or depth-first. The serialization order affects the meaning of a single-point crossover operation: If breadth-first serialization is used, all hierarchical levels beyond the selected point are exchanged between the two ancestor chromosomes. If depth-first serialization is used, this tends to exchange entire subtrees between the ancestors. After crossover, conventional bit-wise mutation can be used as is done in [173]: The mutation operation iterates over the binary vector, and flips vector elements with a selected mutation probability. Examples for crossover depth-first and breadth-first chromosomes are shown in Figures 44 and 45, respectively.

Finally, the crossover process may produce offspring chromosomes that violate the non-nesting requirement described above. Applying recursive repair, potentially, leads to multiple correct chromosomes per offspring individual resulting from a crossover and mutation operation. One possibility is to add all of the resulting chromosomes to the population. This, however, increases

the influence of a pair of ancestor chromosomes that results in a large number of repaired chromosomes after crossover. To limit the influence of a given crossover operation, and to be consistent with the performance of conventional crossover operations that generate two offspring individuals for each pair of ancestor chromosomes, one of the repaired chromosomes resulting from each pre-repair offspring chromosome can be randomly selected.

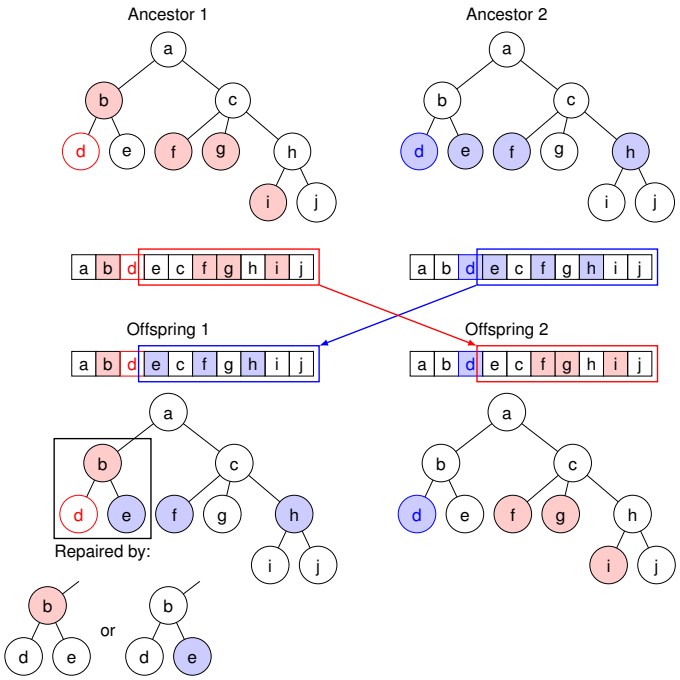


Figure 44: Crossover Operation for Depth-First Serialized Chromosome

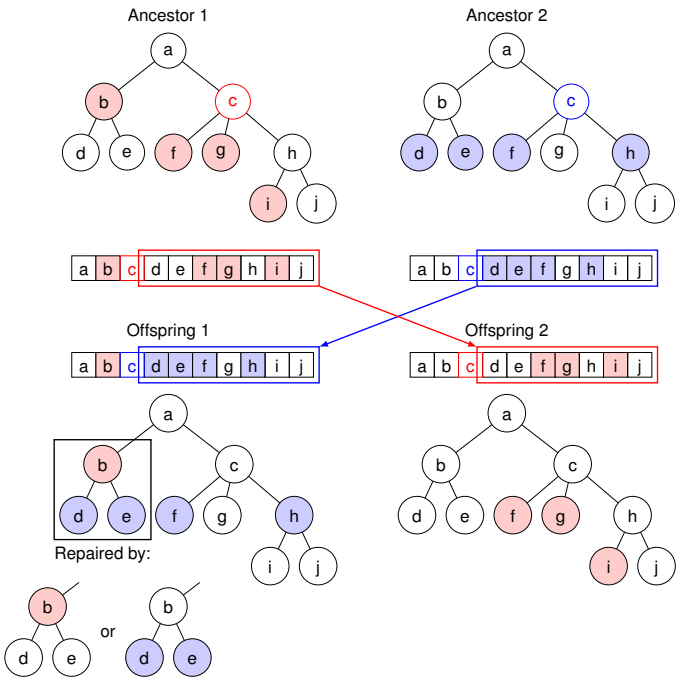


Figure 45: Crossover Operation for Breadth-First Serialized Chromosome

As discussed above, the module-based partial TMR selection problem can be encoded into a graph-based or serialized (breadth- or depth-first) chromosome. To guide further implementation decisions regarding this encoding, simulations on 8000 randomly generated hierarchy trees of sizes between 20 and 60 nodes (200 trees per size increment) were conducted using Python's *igraph* module[116] as follows:

1. A random tree is generated using `igraph.Graph.Tree_Game()`[117]
2. A set of random initial solutions is generated (recursively repairing the random solutions in the process) until at least 128 combinations of solutions exist. The solutions are represented by binary annotations of the original tree's vertices.
3. Each solution is encoded in three ways:
   (a) An unchanged representation of the original tree object (i.e., a graph-encoded chromosome).
   (b) Serialized to a vector using breadth-first traversal order.
   (c) Serialized to a vector using depth-first traversal order.
4. 128 random pairs of solutions are selected. Crossover and mutation operations are applied to these pairs as follows:
   - Select a random point in the chromosome. This represents the crossover root node (graph-encoded chromosome) or crossover point (serialized chromosome)
   - Perform crossover at the selected crossover point, resulting in two new individuals. Graph-encoded crossover is performed by swapping the replication annotations in the subtree identified by the selected vertex (see Figure 43). Single-point crossover for serialized chromosomes is performed by selecting a pivot within the chromosome and subsequently splicing the chromosomes, as shown in the example in Figures 44 and 45.
   - The offspring solutions generated by the crossover operation are mutated by flipping random replication properties with a probability of $p = 1/n$. For the sake of simplicity, mutation is omitted in Figures 43 to 45.
   - The mutated solutions are recursively repaired. As neither of the two variants generated in the tree encoding example are invalid, but one each of the ones generated in the serialized examples, repair is shown in Figures 44 and 45. One repaired variant is randomly selected for each of the two original offspring solutions and added to the set of solutions.
5. The number of solutions generated by each crossover method is counted.

The result of this simulation experiment is shown in Figure 46. The top part of the figure shows the average number of solutions generated by each of the chromosome encoding and crossover methods discussed above (serialized tree representation using breadth-first or depth-first traversal and tree-encoding) over the evaluated range of graph sizes. The error bars in the top part of the figure represent the standard deviation of the average number of generated solutions. The bottom part of Figure 46 shows the shares of "winning" encoding and crossover methods that generate the most unique chromosomes.

[116] See `https://web.archive.org/web/20250425144528/https://python.igraph.org/en/stable/`, accessed 2026-01-05

[117] See `https://web.archive.org/web/20250426071817/https://python.igraph.org/en/stable/api/igraph.GraphBase.html#Tree_Game`, accessed 2026-01-05

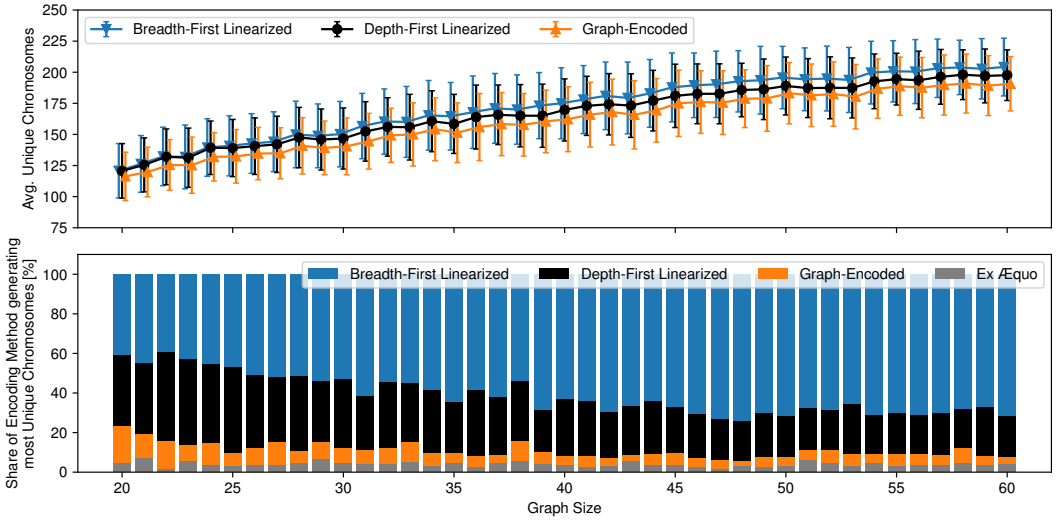


Figure 46: Simulation Results for Effect of Chromosome Encoding on Offspring Diversity

The top plot in Figure 46 shows that, on average, breadth-first serialized chromosomes produce the most diverse offspring solutions after crossover, mutation, and repair. This is closely followed by the depth-first serialized chromosomes, while graph-encoded chromosomes lag behind, especially with larger trees. The bottom plot in Figure 46 demonstrates that serialized chromosomes produce more offspring than graph-encoded chromosomes in the vast majority of cases. Especially at larger tree sizes, breadth-first serialized chromosomes generate the most diverse offspring in most cases. This simulation thus indicates that choosing breadth-first serialized chromosomes offers a higher chance of producing diverse offspring than the other encoding methods. It should be noted, however, that this simulation does not consider the quality of the generated solutions or a similarity metric to the previous generation. Nevertheless, it offers a first indication of choosing a vector-based encoding method for the proposed concept's initial implementation.

5.1.3 Fitness Evaluation

Once a candidate population of chromosomes has been generated, the fitness of each TMR solution concerning the approach's objectives needs to be evaluated for the following selection step. As described earlier, the proposed approach aims at minimizing (1) the resource usage of a solution that correlates with FPGA utilization, power dissipation, and timing performance and (2) SEU susceptibility, both of which need to be quantified by the fitness function.

A TMR variant's resource usage includes both logic and interconnect resources. The overhead in terms of logic resources results directly from replicating cells according to a given TMR strategy (e.g., Block TMR or feedback-focused TMR [161]). At minimum, all cells below the hierarchical netlist node to be replicated are tripled; additional LUTs are instantiated to implement majority voters. Interconnect resource usage is more challenging to estimate as it strongly depends on the placement of the added cells. Furthermore, modern FPGAs provide a variety of interconnect wire lengths that may cross a significant portion of the device while traversing few switch boxes. This reduces the correlation between wire length and the number of sensitive interconnect bits, making widely-used estimation models such as Half-Perimeter Wire Length (HPWL) [199] less accurate for this task.

Precisely estimating the configuration-level reliability of a design from the netlist level alone is an

even more difficult task: The individual paths implemented by the router for each net influence which subsets of the cells driven by the net are downstream of the routing resources affected by an individual bit flip and thus are affected by its consequences. Furthermore, the criticality of faults that short-circuit two or more signals of the design may depend on (1) the behavior of the interconnect technology (e.g., wired-OR, wired-AND) and (2) the "compatibility" of the affected signals in this context.

Thus, we propose performing placement and routing as part of fitness evaluation, as is done in DSE approaches in related work [174, 175]. Furthermore, we propose to perform static bitstream criticality analysis (as, e.g., described in [99]) in the fitness function. Due to the dependence on the FPGA technology, this approach must be adapted for each target technology. Performing placement and routing and bitstream generation in the fitness function allows the use of accurate data for the *resource usage* dimension of the fitness value. This also provides detailed information for subsequent estimation of *SEU* susceptibility. Finally, having a fully placed and routed design allows fault injection experiments to be performed that more accurately determine the candidate designs' reliability.

As shown in Figure 39, the proposed fitness function performs the following steps:

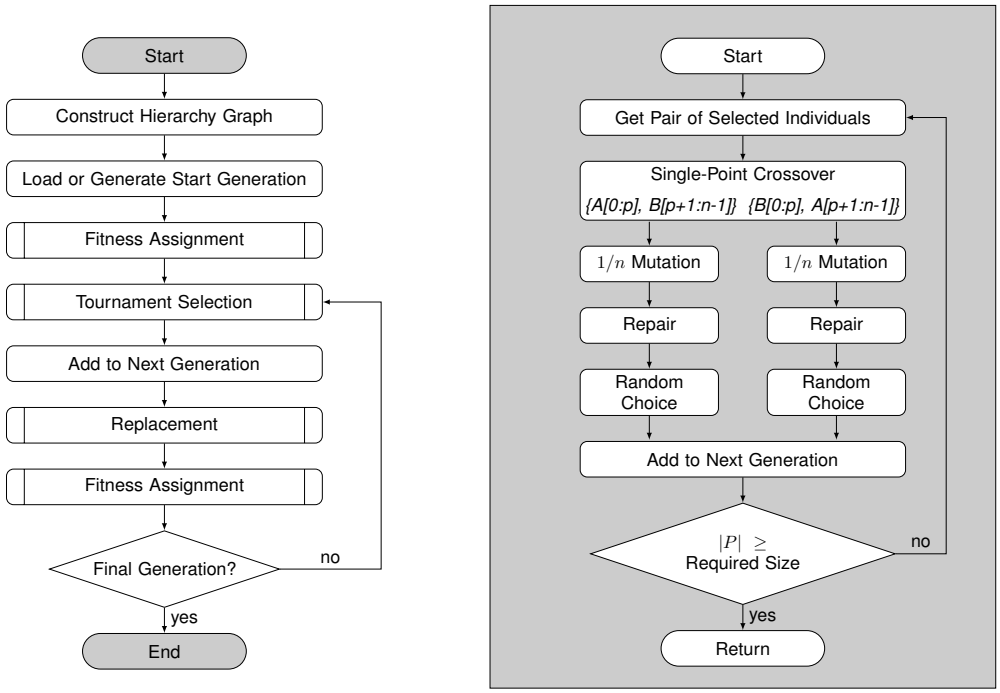
1. Insert TMR into the DUT circuit according to the chromosome of the solution using a given TMR insertion tool (e.g., AMD's TMRTool or Synopsys's Synplify Premier DP, among others). This can be seen as a type of *"genotype-to-phenotype mapping"* [238] as it instantiates the circuit resulting from the candidate solution's chromosome.

2. Place and route the DUT circuit for a target device. This step may make use of reliability-aware placement and routing techniques, such as physically separating TMR replicas using reliability-aware placement & routing algorithms to prevent CMFs [99, 226, 134, 117]. However, also non-separated TMR offers reliability improvements [153]. This step continues the *"genotype–phenotype mapping"*, implementing an FPGA bitstream based on the candidate solution's circuit.

3. Analyze the SEU susceptibility of the circuit according to static fault models for the target FPGA technology. These models allow ruling out certain configuration bits from being critical to the design's functionality due to being related to unused resources or modeled as benign faults. This prediction process results in a list of potentially SEU-susceptible configuration bits, with their total number allowing the prediction of the design's (un)reliability.

4. As an optional final step to increase reliability assessment accuracy, fault injection experiments on a subset of the potentially SEU-sensitive bits on one or more FPGAs can be conducted. By comparing expected responses to those delivered by the DUT under fault injection, SEU-sensitive bits can be separated from bits that do not affect the DUT's correct functionality when flipped.

While fault injection delivers more accurate results regarding bit-level design SEU susceptibility, it places more constraints on the designer. In particular, a set of FPGAs and hardware/software infrastructure to conduct fault injection experiments are required. Furthermore, test vectors for fault injection experiments representative of the design's workload in the field are needed to test SEU susceptibility under realistic conditions. If on-chip fault injection is to be used (e.g., using partial reconfiguration), this needs to be addressed in the FPGA floorplan. Finally, fault injection experiments may increase the execution time of the DSE process.



(a) Main Optimization Loop (b) *Replacement* Procedure

Figure 47: Implemented Optimization Procedure

5.2 Implementation

This section describes an implementation of the module-based TMR approach proposed in Section 5.1 targeting AMD's Artix 7 xc7a35t FPGA that is part of AMD's widely-used 7 Series FPGA family. This implementation enables the evaluation of the effectiveness of the proposed approach.

Its optimization procedure is described in Section 5.2.1. The implemented fitness function follows the flow described in Section 5.1 and is described in detail in Section 5.2.2. Both fitness assignment and validation experiments use fault injection as described in Section 5.2.2. Furthermore, the fitness assignment uses Synopsys Synplify to insert partial TMR according to the individual's chromosome. This process is discussed in Section 5.2.2, where possible extensions to the TMR insertion process and alternatives to Synopsys Synplify are outlined as well.

5.2.1 Genetic Optimization Procedure

The genetic optimization procedure is implemented as a collection of Python scripts using functionality from DEAP [239]. It extends DEAP's NSGA-II [173] procedure[118] by implementing a custom *Replacement* and *Fitness Assignment* procedure.

An overview of the implemented optimization procedure is shown in Figure 47. Here, Figure 47a shows the overall optimization loop, and Figure 47b shows the custom replacement procedure implemented for the problem at hand.

The main optimization loop starts with the extraction of the original design's netlist hierarchy, referring to the hierarchy of netlist modules whose root is the top-level module, and whose leaves are the individual FPGA technology cell instances in the form of a tree as shown in Section 5.1.1.

[118] See `https://web.archive.org/web/20250426072607/https://github.com/DEAP/deap/blob/master/examples/ga/nsga2.py`, accessed 2026-01-05

In the presented implementation, this hierarchy is extracted from the hierarchical names of leaf cells, which is a side-product of the RTL-to-netlist synthesis of the original RTL design. During the initial synthesis of the original design, the original RTL hierarchy must be preserved in order to provide the DSE flow with an appropriate design space. If gate-level TMR is supported by the employed TMR insertion approach, individual technology cell instances may be kept as part of the hierarchy tree. It is to be noted that the search space using gate-level TMR is considerably larger than when using the lowest-level modules – those exclusively instantiating technology cells – as the leaves of the hierarchy tree. As the TMR insertion tool employed in the presented implementation does not support gate-level TMR, the leaves of this hierarchy tree are the lowest-level modules. This hierarchy tree is then flattened into a binary vector as discussed in Section 5.1.1 to form the design's chromosome. In the presented implementation – backed by the simulation results discussed in Section 5.1.1 – the chromosome vector is generated by breadth-first traversal of the hierarchy tree. The hierarchy tree is kept as a reference for later checking of the non-nesting requirement discussed in Section 5.1.1 and the generation of module-level TMR constraints for the TMR insertion tool (see Section 5.2.2).

Starting Generation

Furthermore, a starting set of solutions must be constructed to begin the optimization process. A set of binary vectors (i.e., chromosomes) is randomly generated in the presented implementation. As these vectors contain invalid solutions violating the non-nesting property described in Section 5.1.1, they are recursively repaired as a second step. From these vectors, a set equal to the required starting set size is selected.

If preexisting information on the relative criticality of the individual modules exists, it may help generate the starting generation in a more "informed" way. This may include fault simulation results from various levels of abstraction and variants deemed potentially beneficial by the designer. Adding these variants to the starting set provides the advantage that the genetic optimization process does not need to "discover" these variants and can use them as a starting point for further optimization. In the presented implementation, two chromosomes are added to the starting set in this way: A chromosome requiring no replication (i.e., a vector set to all-0), and a chromosome requiring replication with the toplevel node replicated (i.e., full TMR).

Recursive Repair

The procedure applied for recursive repair is shown in Algorithms 1 and 2. Algorithm 1 describes the recursive utility function GETSUBTREEREPLICATIONS(T, c, v) that traverses a subtree of the design's hierarchy tree T, identifying the vertices (i.e., modules) that are to be replicated according to the chromosome c, a binary vector whose indices correspond to the vertex indexes within G. This utility function is used in the recursive repair procedure REPAIRANDFORK(T, c, v, C, r) shown in Algorithm 2. This procedure checks and, if necessary, recursively repairs variants of the initially given chromosome c concerning the design's hierarchy tree T.

For a given chromosome c, starting from a given vertex v within the hierarchy graph T, the REPAIRANDFORK() first checks if it fulfills the non-nesting property stated in Section 5.1 for vertex v. If it does, the chromosome is recursively checked starting from each successor vertex of v. Given that this check succeeds for the entire chromosome, this results in an empty $errlist$ after traversal, allowing the chromosome to be added to the set of repaired chromosomes C. If this check fails at any point, this means that both the root node of a subtree as well as a subset of its subtree nodes are set to $true$ in the chromosome. Thus, (1) the error is added to $errlist$, preventing the

Algorithm 1 GETSUBTREEREPLICATIONS(): Recursively Extract Replication Properties

Require: Hierarchy Tree T, Chromosome c, Subgraph root r
Ensure: Set of replicated vertices R in subtree

```
1: function GETSUBTREEREPLICATIONS(T, c, v)
2:     R ← ∅
3:     function _TRAVERSE(vertex)                    ▷ Nested traversal helper function
4:         for all successor ∈ T.successors(vertex) do
5:             if c[successor] then
6:                 R ← R ∪ {successor}
7:             end if
8:             _TRAVERSE(successor)
9:         end for
10:    end function
11:    _TRAVERSE(r)                                  ▷ Nested traversal helper function invocation
12:    return R
13: end function
```

addition of c to C, and (2) two copies of the chromosome are generated: A copy c_1 where the root node's replication requirement is lifted, and a copy c_2 where all subtree replication requirements are lifted. While not necessarily entirely correct, they are *more* correct than the original chromosome. The partially corrected chromosomes are thus subsequently subjected to the same recursive repair procedure to repair errors at deeper levels of recursion or in parallel hierarchy subtrees. These procedures are used in generating the random starting population and in the *replacement* procedure to repair errors occurring due to crossover and mutation.

Fitness Assignment

Referring back to Figure 47, after generation or loading the starting generation, the genetic optimization process requires to assign each chromosome with a fitness value – in this case, a two-dimensional fitness value consisting of a measure for a chromosome's replication cost and soft-error vulnerability as described in Section 5.1. Implementation details of the fitness assignment procedure for this dissertation are described in Section 5.2.2.

Selection & Replacement

After the fitness assignment, the next generation population is generated. The first half of the next generation is selected from the current population based on domination relationships between individuals as a primary factor and crowding distance as a secondary factor. This is accomplished using DEAP's `selTournamentDCD()`[119] function.

The other half is replaced by the procedure shown in Figure 47b, which combines individuals from this selected subset to generate replacement individuals. This is done by first selecting pairs of subsequent individuals from the vector of selected individuals as done in DEAP's NSGA-II example[120]. Each pair of individuals (A_i, B_i) is then subjected to single-point crossover as mentioned in Section 5.1.2. To that end, a random point within the chromosome is selected. Two offspring chromosomes are generated by combining the first part from A_i with the second part from B_i and vice-versa. These offspring chromosomes are subsequently subjected to bit-wise mutation:

[119] See `https://web.archive.org/web/20250426073109/https://deap.readthedocs.io/en/stable/api/tools.html#deap.tools.selTournamentDCD`, accessed 2026-01-05

[120] See `https://web.archive.org/web/20250426072607/https://github.com/DEAP/deap/blob/master/examples/ga/nsga2.py`, accessed 2026-01-05

Algorithm 2 REPAIRANDFORK(): Recursive Chromosome Repair

Require: Hierarchy Tree T, Chromosome c, Vertex v, Correct chromosomes set C, Root r
Ensure: Updated correct chromosomes C

```
1: procedure REPAIRANDFORK(T, c, v, C, r)
2:     errlist ← []                                          ▷ Initialize error list
3:     function _TRAVERSE(G, c, v, errlist, r, C)            ▷ Nested traversal helper function
4:         R ← GETSUBTREEREPLICATIONS(G, c, v)
5:         if R ≠ ∅ and c[v] then                            ▷ Non-nesting requirement violated
6:             errlist.append(v)                             ▷ Add vertex to error list
7:             c_1 ← copy(c)                                 ▷ Variant 1: Reset root only
8:             c_1[v] ← 0
9:             REPAIRANDFORK(T, c_1, r, C, r)
10:            c_2 ← copy(c)                                 ▷ Variant 2: Reset entire subtree
11:            for each u ∈ R do
12:                c_2[u] ← 0
13:            end for
14:            REPAIRANDFORK(T, c_2, r, C, r)
15:        else
16:            for each s ∈ T.successors(v) do               ▷ Variant 3: Recursively check
17:                _TRAVERSE(T, c, s, errlist, r, C)
18:            end for
19:        end if
20:    end function
21:    _TRAVERSE(T, c, v, errlist, r, C)                     ▷ Nested traversal helper function invocation
22:    if errlist = ∅ then
23:        C ← C ∪ {c}
24:    end if
25: end procedure
```

The mutation procedure iterates over both offspring chromosomes and randomly flips each bit in the chromosomes with a probability of $1/n$. As both crossover and mutation may lead to chromosomes violating the non-nesting requirement formulated in Section 5.1, both offspring chromosomes are subjected to the correction procedure described above. This may generate more than one variant of both offspring chromosomes if correction is necessary. Per offspring chromosome, one corrected variant is randomly selected and added to the next population generation. This procedure is repeated until the next generation population reaches the required population size.

After replacement, the next generation population is ready for another iteration of fitness evaluation. Optimization is then carried out by repeating the implemented optimization loop until a fixed number of 16 generations is reached. This fixed number of generations was chosen to evaluate and compare the convergence performance of the genetic algorithm across multiple designs: A suspected design-dependent component of convergence may be visible even when evaluating a fixed number of generations. The resulting data may deliver further insights into better choices of this optimization parameter.

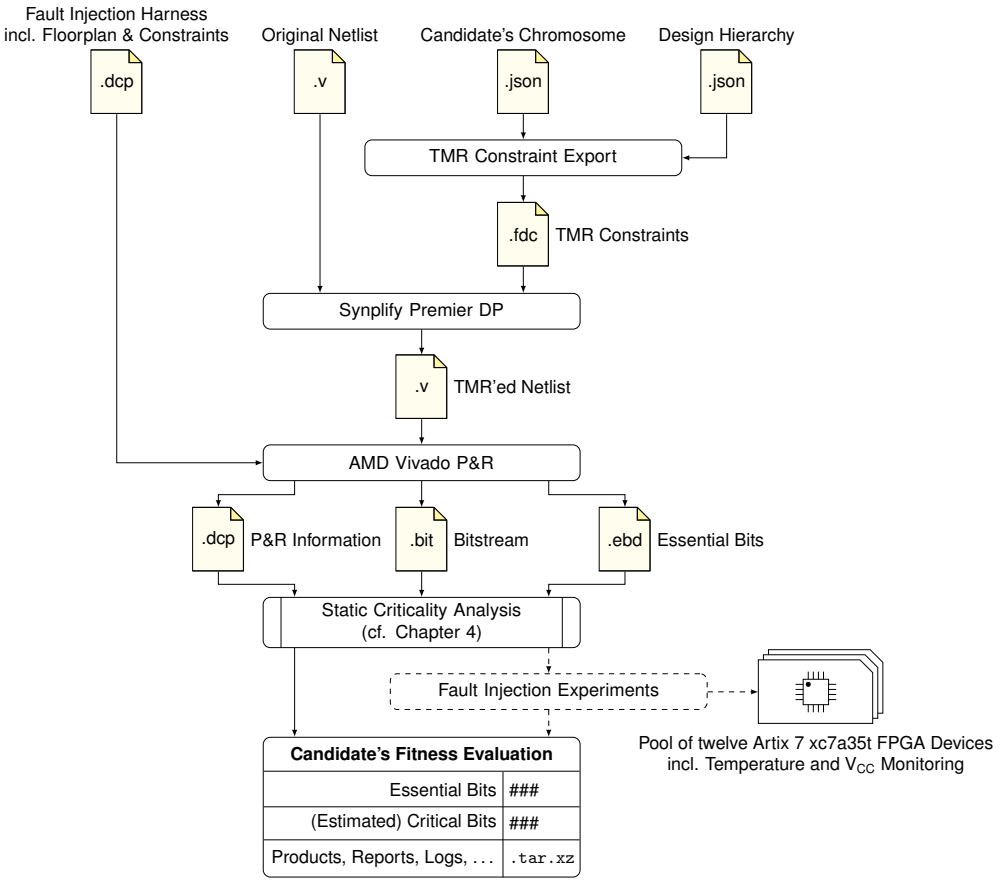


Figure 48: General Structure of Implemented Fitness Function(s)

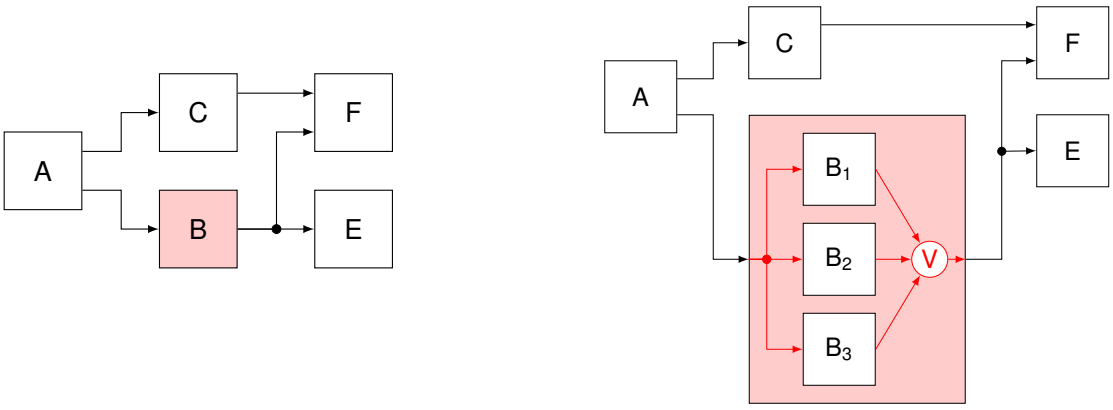
5.2.2 Fitness Assignment

The implemented fitness function follows the concepts proposed in Section 5.1.3, and thus involves the following steps:

1. Insert redundancy into the original design according to the individuals' chromosomes.
2. Implement the design variant for the target FPGA device (AMD Artix 7 xc7a35t).
3. Analyze the resulting bitstream to statically estimate the cost and reliability of the design variant under consideration.
4. Optionally, supplement the reliability analysis by performing fault injection experiments on one or more physical FPGA devices.

Fitness assignment is conducted for each generation after the generation's population has been initialized. Thus, as the evaluation of the fitness of each individual is independent of the evaluation of the others, it can be parallelized given that sufficient computing (i.e., CPU and RAM for performing implementation and analysis) and hardware resources (i.e., FPGA boards for performing fault injection experiment) are available.

The details of this implementation are illustrated in Figure 48. The results of these steps are used to derive a two-dimensional fitness value consisting of the number of essential bits as a measure for design size and the (estimated) number of critical bits as a measure for the design's soft-error susceptibility.



(a) Original Circuit (b) TMR inserted in Subcircuit *B*

Figure 49: Module-Level TMR Insertion Process

Table 75: `syn_radhardlevel` attribute values in Synopsys Synplify 2019 Premier DP [240]

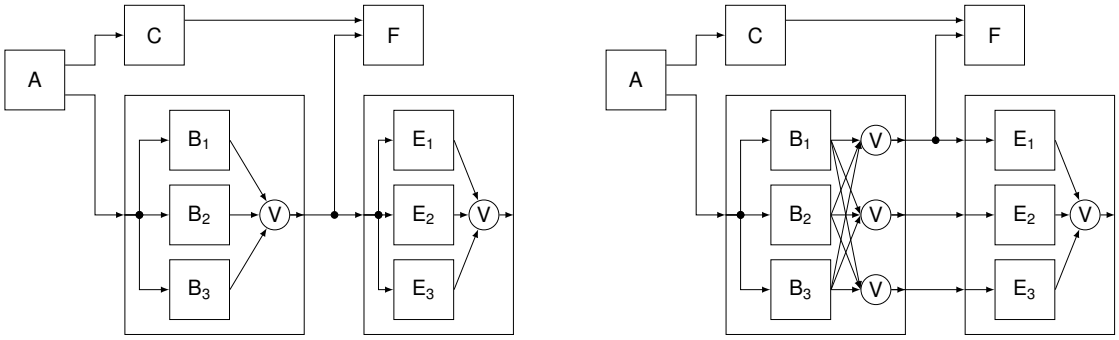
Attribute Value	Effect on Circuit
`none`	Default; No Replication.
`tmr`	Triplicates & votes registers (flip-flops, latches) only.
`block_tmr`	Triplicates & votes modules at their interface.
`distributed_tmr`	Triplicates & votes modules at their interface; (optionally) triplicates logic loops and voters.
`duplicate_with_compare`	Duplicates logic for fault detection purposes.
`cc`	Microchip-specific; affects register implementation only.
`tmr_cc`	Microchip-specific; affects register implementation only.

TMR Insertion

A TMR insertion tool suitable for use with the proposed reliability optimization flow must be able to (1) parse an input netlist, (2) parse a list of hierarchical modules to be triplicated, and (3) triplicate the internals of these modules and connect the module's inputs to the replicas' inputs as well as insert voters between the replicas' outputs and the module's outputs. The effects of the latter step are illustrated in Figure 49 for the subcircuit *B*.

As discussed in Section 5.2.2, Synopsys Synplify 2019 Premier DP is used to insert partial TMR in the DUT's netlist. Synplify Premier 2019 DP supports module-based reliability constraints that affect how the tool transforms the resulting netlist: The `syn_radhardlevel` attribute can be set to the values shown in Table 75.

Thus, to introduce module-level TMR into the circuit, `syn_radhardlevel` may be set to `block_tmr` or `distributed_tmr`. The implementation of distributed TMR can be modified using the attribute `syn_vote_loops`: Per default, distributed TMR inserts voters in feedback paths from a register back to its input when such paths are detected during synthesis [161], allowing self-correcting behavior of SEUs in register cells. However, this can be disabled by setting the attribute `syn_vote_loops` to *false*, reducing the number of introduced voters and improving the timing properties of the resulting design. As a trade-off, an SEU occurring in a register cell is no longer corrected during operation and thus requires a reset of the design. Setting `syn_vote_loops` to *false* configures the implementation of distributed TMR to behave similarly to a block TMR implementation. However,



(a) Block TMR (b) Distributed TMR

Figure 50: Comparison of Synopsys Synplify Premier TMR Insertion Strategies for Chained TMR Modules B and E [241]

a difference between distributed TMR and block TMR exists when considering a situation where one triplicated module's outputs drive the inputs of another triplicated module [241]: When using distributed TMR, Synplify triplicates the voters at the first module's outputs to minimize the potential for single points of failure, which is not done when using block TMR. This difference is illustrated in Figure 50, where both subcircuits *E* and *B* of the original circuit shown in Figure 49 are to be replicated. In the presented implementation of the proposed reliability optimization approach, distributed TMR is employed on the module level. However, as the proposed optimization toolflow primarily targets SEU affecting the configuration memory, feedback voters are omitted by setting `syn_vote_loops` to *False*.

For TMR insertion, the binary chromosome vector of the solution is transformed back into a list of hierarchical modules selected for replication by cross-referencing the corresponding hierarchy tree. This list of modules is then used to generate the TMR constraints described above for Synplify, which are written to a constraint file to be used by the re-synthesis run carried out by Synplify. Furthermore, to ensure a fair comparison between candidates – e.g., by alterations to the design's hierarchy during TMR insertion – constraints are set to limit further changes or optimization on the netlist during this TMR insertion step.

It can be seen above that DWC is another option for introducing redundancy into the circuit. While faults cannot be masked using DWC, they can be detected by comparing the outputs of two copies of the original modules. This may be an option for less critical components whose calculations can be repeated after scrubbing the respective FPGA configuration. If fault detection is required *in addition* to fault masking, Synplify allows the generation of fault monitors that compare the outputs of each triplicated module, generating an error indication if they mismatch. This signal may trigger a scrubbing cycle as is done in DWC. However, introducing fault monitors introduces further logic into the design: For each (triplet of) the module's N outputs, $e_i = o_{0,i} \oplus o_{1,i} \vee o_{1,i} \oplus o_{2,i}$ must be calculated. A single error indication signal for all N outputs of the module can be generated by OR-reducing the e_i mismatch signals, i.e., $e = \bigvee_{i=0}^{N} e_i$. While the former operation may be implemented in a single three-input LUT, the latter may result in a larger tree of LUTs depending on N. Furthermore, the logic and interconnect necessary for this calculation is itself SEU-sensitive. While a stuck-at-1 fault of the fault detection logic may be self-correcting (as this may trigger a scrubbing operation), a stuck-at-0 fault may prevent scrubbing from being triggered in this way. Thus, additional periodic scrubbing is required to address these faults.

Possible alternatives to using Synopsys Synplify Premier DP as a TMR insertion tool are AMD's TMRTool (Virtex 4 and 5 device families only) [242] and Siemens EDA's Precision Hi-Rel [243].

Potential candidates for open-source solutions are BYU's EDIF tools[121] as well as SpyDrNet-TMR[122].

FPGA Implementation & Static Criticality Analysis

The resulting netlist is placed and routed, and the resulting logic is subsequently inserted into a pre-generated fault injection harness design using AMD Vivado 2019.1. The standard implementation strategy provided by AMD Vivado is used; the implementation tool is supplied with clock constraints, pin constraints, and a floorplan physically separating the DUT from the fault injection harness. However, cross-hierarchy optimization and flattening are disabled to ensure a fair comparison between design variants and to preserve the circuit copies instantiated by the TMR insertion tool.

No spatial separation constraints for the instantiated TMR replica are supplied in the implementation for the evaluation experiments conducted in this dissertation, resulting in non-separated TMR. While this TMR variant is more prone to CMF, it nevertheless significantly reduces soft-error sensitivity [153]. To prevent CMFs, reliability-aware placement and routing algorithms, such as those discussed in Section 3.3.1, can be employed. Alternatively, floorplan regions can be defined in the Vivado implementation tool using nested *Pblocks* (i.e., physical design partitions). To automate this task, AMD Vivado 2019.1 provides the `place_pblocks` command, which calculates the required size of the rectangular physical design partition and places it in the design's floorplan [244]. However, Vivado-internal bugs prevent this functionality from being used effectively for physically separating TMR replica[123] and Vivado 2020.1 removed the `place_pblocks` command [177].

Thus, physical separation is a possible improvement in future work. Nevertheless, the presented approach is agnostic of the characteristics of the implementation tool employed, as the design is treated as a "black box" for criticality analysis. Furthermore, physically separating TMR replica leads to a less dense design, increasing both the number of interconnect resources necessary to link the design's cells and the number of slices (partially) utilized by the design.

The result of the implementation step is a placed and routed design that AMD Vivado exports as a *design checkpoint* (.dcp) file (see Section 4.1.2). Furthermore, AMD Vivado generates the design's bitstream and a map of its essential configuration bits (.ebd). While the bitstream contains the individual settings for each configuration bit, the .ebd file includes a binary mask of the bitstream that indicates which bits AMD Vivado deems relevant to the design (see Section 2.3).

The placement and routing information present in the .dcp file is parsed and further processed using AMD's RapidWright[124] API.

The bitstream and .ebd files are parsed into configuration bit addresses using tools provided by the third-party open-source bitstream documentation project for AMD 7 Series FPGAs *Project XRAY*[125]. This data forms the starting point for static criticality estimation as presented in Section 4.5, and, optionally, fault injection (see Section 5.2.2) for use in the proposed approach's fitness function. As shown in Section 4.5, the static criticality analysis approach presented in this work effectively eliminates non-critical bits from the essential bits while causing minimal false negatives. Despite

[121] See `https://web.archive.org/web/20250426072631/https://github.com/eddiehung/byuediftools`, accessed 2026-01-05

[122] See `https://web.archive.org/web/20250314162649/https://byuccl.github.io/spydrnet-tmr/`, accessed 2026-01-05

[123] See `https://web.archive.org/web/20250315142434/http://web.archive.org/screenshot/https://adaptivesupport.amd.com/s/question/0D52E00006hpKwnSAE/placepblocks-generates-cryptic-debug-generatelocation-mxradius2-myradius2-message`, accessed 2026-01-05

[124] See `https://web.archive.org/web/20250815131553/http://www.rapidwright.io/`, accessed 2026-01-05

[125] See `https://web.archive.org/web/20250630071412/https://github.com/f4pga/prjxray`, accessed 2026-01-05

this, it is still pessimistic, leading to a considerable amount of false positives. If the statically predicted number of critical bits is used as a measure of the reliability-related fitness component, these false positives may negatively impact the quality of the optimization process's results. Thus, static criticality analysis may be supplemented by fault injection experiments on (parts of) the DUT portion of the bitstream. This selection may be done based on the category of configuration bits (e.g., subjecting configuration bit categories with a comparatively large amount of false-positives to additional fault injection), using statistical methods as done, e.g., in [175], or a combination of both approaches. The following subsection describes the general property of fault injection experiments conducted within the presented implementation of the proposed optimization approach.

Configuration Memory Fault Injection

Wherever fault injection is applied in this work, both for fitness evaluation and validation, a bitstream containing the DUT and a monitoring and fault injection harness is loaded into the configuration Flash of a Digilent BASYS 3 FPGA board. This approach is based on the fault injection procedure described in Chapter 4. Single-bit errors are then injected via partial reconfiguration – a state-of-the-art approach described, e.g., in [143]. The following procedure was applied, similar to the procedure described in Section 4.5.1:

1. The FPGA loads the fault-free bitstream from configuration Flash.
2. The configuration bit to be targeted by fault injection is communicated to the FPGA via UART from a control PC.
3. The correct operation of the DUT is verified by performing an iteration of its functionality and checking the results.
4. The DUT is reset and its clock is stopped.
5. A bit flip is injected using the FPGA's internal configuration interface via partial reconfiguration: The configuration frame containing the currently targeted bit is read into a buffer, the targeted bit is inverted, and the frame is written back.
6. The DUT and its clock are enabled and the DUT performs an iteration of its functionality under the injected fault.
7. Execution results are reported back to the control PC.
8. The fault-free configuration is re-loaded from Flash to ensure a clean state for the next experiment.

The designs targeted by the presented evaluation (see Section 5.3) possess a self-test function with a pre-determined latency and signature value. A fault is considered critical if the design's self-test delivers an incorrect signature or its latency mismatches a simulation-determined value. This allows the designs-under-test to be treated as "black boxes" without detailed knowledge of bit-flips' effect on the circuit's behavioral conditions. Both fault injection and DUT monitoring are accomplished by an on-chip control system based on an AMD MicroBlaze soft CPU floorplanned into a device region not targeted by fault injection.

Generalizing this approach to a broader range of designs requires the designer to supply a representative set of test vectors that the fault injection harness may apply for each injected fault to determine criticality or a BIST test vector generator and signature analyzer hardware unit to be integrated into the harness. Finally, for the deployed version of the design, the fault

injection harness may either remain in the bitstream to be used to carry out bitstream readback and ECC checking as well as periodic or event-triggered scrubbing, or replaced by an empty partial bitstream that ties off the harness's outputs driving the DUT. The latter possibility relies on the clean separation between the harness and the DUT in the floorplan.

As shown in Section 4.4, PVT effects cause the number of critical bits determined in fault injection experiments to vary with the chip temperature and supply voltage during fault injection. Thus, as part of the evaluations carried out in this work, the individual device used for fault injection is recorded, and the environment temperature is controlled. To achieve reliable results, fault injection experiments should thus be conducted under the conditions to which the bitstream will be deployed. If a range of environment temperatures should be supported, testing AMD 7 Series devices at lower temperatures and lower voltages appear to cause worse reliability, as shown in Sections 4.4.1 and 4.4.3, respectively. Thus, testing at low temperatures or lower voltages may yield more conservative results.

Furthermore, these PVT effects may impact the fitness assigned by fault-injection-based fitness functions: A particular bitstream might yield different numbers of critical bits when evaluated using fault injection on one device versus another, as shown in Tables 67 and 68. The impact of these effects is evaluated in Section 5.3 by comparing results achieved by fitness assignment using fault injection results from a single device versus fitness assignment that re-evaluates each variant on multiple devices. Finally, a "control group" of devices is used in a subset of validation experiments in Section 5.3 to analyze the generalizability of the achieved optimization results.

Fitness Evaluation

The fitness function outputs the following values after the assessment of a particular candidate circuit: The number of *essential bits* offers a convenient numerical estimate for FPGA utilization, unifying the amount of logic and interconnect resources used by the design, and is thus used as a measure of the cost of this candidate. The number of statically predicted or empirically determined *critical bits* – or a combination of both – is used as a measure of the SEU susceptibility of this candidate. To reduce the effort in determining the fitness value for subsequent generations, fitness caching [245] is applied in "caching" fitness function variants (see Section 5.3). This ensures that the fitness of a given chromosome that appears in multiple subsequent generations only needs to be placed, routed, analyzed, and – optionally – subjected to fault injection once. In fitness function variants that apply fault injection, this implies the generalization of the fitness value determined on one device to instantiations on all target devices – thereby assuming that PVT effects do not significantly contribute to overall fitness evaluation or selection. Nevertheless, even in fitness function variants assuming that PVT effects *do* affect fitness evaluation and thereby selection results, all synthesis and P&R products, reports, and logs are archived in the form of a `xz`-compressed `tar` file per candidate solution in each generation. This allows faster re-evaluation of each implemented solution on other devices and in validation experiments after the conclusion of the optimization phase. Section 5.3 provides details on the individual sub-types of fitness evaluation investigated in this work.

5.3 Evaluation Strategy

The evaluation of the DSE approach proposed in Section 5.1 with the help of the implementation described in Section 5.2 aims at answering the following questions:

Q1: Criticality-related fitness can be assigned (1) statically by predicting fault effects from the design's bitstream, (2) empirically by conducting (statistical) fault injection experiments on one or more devices, or (3) a combination of these methods. How well do the corresponding fitness values correlate with criticality observed when evaluating the resulting designs on a set of identical FPGAs?

Q2: How effective is the design-space exploration process for each fitness assignment method mentioned above, i.e., how much can cost or criticality be optimized in a set amount of generations compared to a randomly generated set of starting solutions?

Q3: How severely is the quality of optimization results impacted by inter-device variation?

Q4: How time-consuming are the different fitness assignment methods mentioned above compared to each other?

Q5: How does design size affect scalability? Are (1) reduced generation size and (2) statistical fault injection effective at improving scalability?

Q6: How does the presented approach fare compared to related work?

To answer these questions, the implementation described in Section 5.2 was used to co-optimize cost and reliability of four case-study designs, targeting AMD's Artix 7 xc7a35t FPGA as targeted by the static criticality prediction described in Section 4.5. These four designs – *B13m*, *Add32*, *Adler32*, and *SERV* – stem from the evaluation this approach.

Table 76 shows the designs' hierarchical levels with the modules and logic resources at each level when implemented for the target device. The designs allow to evaluate the proposed approach and its implementation on a variety of hierarchical structures, with the designs furthermore representing a microcosm of commonly used FPGA resources. Due to their size, *B13m*, *Add32*, *Adler32*, and *SERV* allow demonstration and evaluation of the proposed approach in terms of larger population size and exhaustive fault injection runs. Thus, these designs are employed to analyze the accuracy of the proposed static criticality prediction under exhaustive fault injection. Due to the wrapping of *B13m*'s gate-level cells, it provides a high number of hierarchical modules suitable for replication by Synplify. While *Add32* and *Adler32* feature a relatively flat and regular hierarchical structure, *SERV* has a more *organic* hierarchical structure due to it being based on hand-written well-modularized RTL. All mentioned designs are self-contained, requiring only an external *start* pulse to start a self-test, which outputs a signature value after a certain number of clock cycles and asserts a *done* pulse. This is derived from the interface generated by PandA Bambu [227] for the CHStone [229] benchmark suite.

The non-nesting requirement described in Section 5.1.1 reduces the design space (i.e., the number of possible redundancy configurations) between one and three orders of magnitude by applying Equation (5.1): e.g., from 2^{114} to 8.12×10^{31} for *B13m* and from 2^{19} to 1.65×10^{4} for *SERV*.

For each combination of design and fitness assignment method, one optimization run of 16 generations was conducted. For each design, all runs started with the same set of 131 randomly generated starting individuals, including two static solutions: no TMR, TMR of the topmost module. Per generation, 64 of 128 individuals are selected; bit-wise mutation at a rate of $1/n$ as used in [173] is applied. These parameters were chosen arbitrarily for demonstration and evaluation. They may need to be adapted to a particular circuit. The following methods of assessing the criticality-related fitness dimension are proposed and evaluated in this work:

1. *Static Criticality Prediction (S)*: The number of statically predicted critical bits (see Section 4.5) is used for the criticality-related fitness component.

Table 76: Hierarchical Structure and FPGA Resources of Case-Study Designs

(a) *B13m*

Level	Modules	Resources					
		LUT	FF	Carry	BRAM	DSP	Wide MUX
1	1	16	–	–	–	–	–
2	1	19	1	–	–	–	–
3	3	17	27	–	–	–	–
4	5	–	–	–	–	–	–
5	104	49	55	–	–	–	–
Sum	114	101	83	–	–	–	–

(b) *Add32*

Level	Modules	Resources					
		LUT	FF	Carry	BRAM	DSP	Wide MUX
1	1	–	–	–	–	–	–
2	1	–	–	–	–	–	–
3	3	13	7	–	–	–	–
4	11	85	49	7	–	–	–
5	2	34	70	–	–	–	–
Sum	18	132	126	7	–	–	–

(c) *Adler32*

Level	Modules	Resources					
		LUT	FF	Carry	BRAM	DSP	Wide MUX
1	1	–	–	–	–	–	–
2	1	–	–	–	–	–	–
3	3	12	4	–	–	–	–
4	45	396	211	70	–	2	–
5	1	–	–	–	1	–	–
Sum	51	408	215	70	1	2	–

(d) *SERV*

Level	Modules	Resources					
		LUT	FF	Carry	BRAM	DSP	Wide MUX
1	1	128	–	–	–	–	–
2	1	4	3	–	–	–	–
3	1	–	–	–	–	–	–
4	5	82	34	–	1	–	–
5	3	28	13	–	1	–	–
6	8	163	158	–	–	–	1
Sum	19	405	208	–	2	–	1

2. *Cached Fault Injection (FI_C)*: The static criticality prediction (S) mentioned above uses only structural information, not design-specific functional information, to predict criticality, which can result in overly pessimistic predictions since the design's logic may mask additional faults. The static criticality prediction (S) mentioned above may be overly pessimistic since the design's logic can mask certain faults. To integrate this information into the criticality prediction, fault injection experiments are performed on all bits predicted to be critical. The results are then cached for each evaluated variant and thus generalized from one evaluated FPGA to the other devices.

3. *Combining Static Prediction with Cached Fault Injection ($S+FI_C$)*: Criticality variation between identical FPGAs was predominantly observed in interconnect-related configuration bits (see Sections 4.4 and 4.5) The $S+FI_C$ fitness evaluation variant assumes that all of these bits are critical according to static criticality prediction. Less device-to-device variation is assumed for the remaining bits, making these bits suitable for further reduction via fault injection. Results for these bits are collected on one device and generalized to the others as in the FI_C variant.

4. *Resampling Fault Injection (FI_R)*: Due to possible inter-chip variations in fault injection results due to PVT effects, the number of critical bits determined on two device instances might not necessarily be the same. Thus, a TMR design variant's chance to propagate through the selection process of the genetic algorithm may depend on the evaluation device. Here, all interconnect-related bits are re-evaluated on a randomly chosen device in each generation, allowing variants that are especially sub-optimal on certain devices to be removed from the population. The number of logic-related critical bits is cached as in the $S{+}FI_C$ variant; fault injection is performed under the same conditions.

The S, $S{+}FI_C$, FI_C and FI_R fitness functions were evaluated using the *B13m*, *Add32*, *Adler32*, and *SERV* designs. The S fitness function and statistical variants of the fitness functions above employing fault injection were applied to *NEORV32* and *LEON3* as a demonstration of the toolflow on a larger design.

The fault injection experiments followed the procedure described in Section 5.2.2. To minimize the influence of PVT effects on fault-injection-derived fitness results in the form of variation of individual bit criticality (see Section 4.4), all fault injection experiments were conducted under a controlled environment temperature of 35 °C. The inter-device variability of criticality results was addressed by conducting identical validation experiments on the twelve devices used in optimization (device set 1; Section 4.5.1). Across all designs and devices, the average chip temperature during all fault injection validation runs was measured at a stable 51.1 °C ± 1.0 °C by on-chip temperature sensors (approx. half of the device's extended temperature range [246]), while the supply voltage was measured at 1 V ± 0.05 V. These stable conditions limit the potential influence of PVT effects on fault injection results.

Based on the assigned fitness values, the Pareto fronts of the initial, randomly populated generation (0) and the final 16$^{\text{th}}$ generation were then subjected to another round of fault injection. To address the potential influence of PVT effects on prediction accuracy, each design was originally evaluated on the twelve FPGAs boards used in optimization (device set 1; Section 4.5.1). Faults were injected into the bits reported by static criticality analysis, allowing comparisons between prediction and fault injection results and between the optimization results achieved by each fitness variant. To further investigate the effects of PVT variations on optimization results quality, a second round of validation experiments was conducted on the *B13m*, *Add32*, *Adler32*, and *SERV* designs on a second set of devices (device set 2; see Section 4.5.1). This allows to investigate if the Pareto-optimality of the identified solutions still holds when applied to a set of devices that did not contribute to optimization when using fitness function variants based on fault injection (FI_C, $S{+}FI_C$, FI_R).

While designs comparable to *B13m* are used in related works in the field of partial TMR [165, 142] and reliability-related DSE [179], investigating the scalability of the proposed approach and its implementation to larger designs is important to demonstrate scalability-improving adaptations and highlight potential areas of future improvement. Thus, the described implementation was subjected to two significantly larger designs based on *NEORV32*, a 32-bit RISC-V CPU[126] (evaluated, e.g., in [247]) and *LEON3*, a 32-bit SPARCv8 CPU[127] (evaluated, e.g., in [248]). Both CPU systems run a self-test program consisting of an Adler32 checksum, floating-point vector dot product, and a CRC32 of the results. These designs' resources and hierarchical levels are shown in Table 77. It can be seen that these designs significantly exceed the size of the smaller designs, containing more than 10× more LUTs and 8× more FFs. The significantly larger number of wide-function multiplexers (i.e., cells that allow to combine two six-input LUTs into a seven-input LUT, or four

[126] See `https://web.archive.org/web/20250128134944/https://github.com/stnolting/neorv32`, accessed 2026-01-05

[127] See `https://web.archive.org/web/20250123185022/https://www.gaisler.com/products/leon3`, accessed 2026-01-05

Table 77: 32-Bit CPU Case-Study Designs: Structure and FPGA Resources

(a) NEORV32

Level	Modules	Resources					
		LUT	FF	Carry	BRAM	DSP	Wide MUX
1	1	–	–	–	–	–	–
2	1	–	–	–	–	–	–
3	2	3	1	–	–	–	–
4	12	607	413	–	5	–	1
5	13	698	567	6	1	–	–
6	9	1214	644	48	–	–	32
7	4	1311	575	64	–	2	–
8	2	522	272	11	–	–	–
Sum	44	4355	2472	129	6	2	33

(b) LEON3

Level	Modules	Resources					
		LUT	FF	Carry	BRAM	DSP	Wide MUX
1	1	–	–	–	–	–	–
2	1	4	6	–	–	–	–
3	7	573	249	3	–	–	59
4	3	63	87	2	–	–	–
5	4	27	–	–	2	–	–
6	7	3079	1070	49	–	–	–
7	8	680	244	4	2	1	–
8	4	74	94	–	–	–	–
9	2	–	–	–	2	–	–
Sum	37	4500	1750	58	6	1	59

six-input LUTs into an eight-input LUT in AMD's 7 series technology) indicates a significantly more complex combinational logic. Furthermore, both designs contain more hierarchical levels than the "small" benchmark designs.

To address **Q4** from Section 5.3, runtime was tracked during the optimization runs performed for the six designs. This was achieved by taking time stamps before and after each execution step. This allows the temporal evaluation of the implementation of the DSE approach proposed in this dissertation.

Due to the significantly larger size of these designs, the fitness assignment variants introduced above were extended with statistical fault injection [249]. These variants are denoted $FI_{C,Stat}$, $S{+}FI_{C,Stat}$, and $FI_{R,Stat}$. The required sample size can be estimated using Cochran's sample size formula for finite populations [250] as also done, e.g., in [175]. As parameters for statistical fault injection, a 99% confidence level, 1% margin of error, and no further knowledge of the critical bit proportion (i.e., p=0.5) are used. The upper limits of the confidence intervals for the estimated number of critical bits are used as criticality-related fitness values. Fault injection is performed under the same conditions as in the FI_C variant. Also in the validation experiments carried out for these designs, a statistical sample of configuration bits was chosen. Furthermore, the set of starting solutions was reduced from 131 to 32 and the selection operation selects 16 of 32 individuals per generation when compared to the smaller designs discussed above. The mutation rate remained unchanged at $1/n$.

5.4 Results

The corresponding optimization runs and subsequent evaluation experiments discussed above were conducted to answer the questions posed in Section 5.3 for the implemented cost-reliability optimization toolflow. Validation experiments were conducted on twelve devices that took part in optimization runs (i.e., *Device Group 1*; see Section 4.5 for all evaluated designs. For the "small" benchmark designs *B13m*, *Add32*, *Adler32*, and *SERV*, these experiments were repeated using

twelve devices that were not used in the optimization runs (i.e., *Device Group 2*). The former evaluation runs allow to judge how well results obtained on one device during optimization may be generalized to all devices within that group. The latter evaluation, on the other hand, allows to judge the influence of PVT effects on the validity of the obtained optimization results. These evaluations were repeated under the same conditions on a second set of devices that did not take part in optimization. The remainder of this section presents the obtained results in the order lined out by the evaluation questions posed in Section 5.3.

5.4.1 Precision of Criticality Prediction

To assess the correlation of prediction and fault injection results and quality of optimization results, all Pareto-optimal designs from the initial generation and the final 16th generation were evaluated using fault injection on all tested devices. Figure 51 illustrates the correlation between the criticality predictions made by the fitness assignment approaches and the average number of critical bits resulting from fault injection on all FPGAs tested in this study.

This data is organized by fitness assignment approaches (rows) and designs (columns).

Figure 51 allows to judge (1) to what extent false positives in static criticality estimation (*S*, $S{+}FI_C/FI_{C,Stat}$) are expected to negatively impact optimization performance due to limited correlation with the number of critical bits observed in fault injection on real hardware, and (2) how well fault injection results ($S{+}FI_C/FI_{C,Stat}$, $FI_C/FI_{C,Stat}$, FI_R/FI_R) obtained on one device correlate with the average observed on all devices to investigate the impact of PVT effects on optimization result quality. The virtual metric *Ess.* (i.e., “Essential”) was added as a baseline to compare against the accuracy of static criticality prediction. This metric estimates the number of critical bits expected for a design variant as the difference between the essential bits of a full-TMR variant and the individual variant’s essential bits without performing criticality analysis. It shows the innate correlation of the average number of critical bits of a TMR variant and its essential bits, which varies from design to design. The *Ess.* metric was calculated for the solutions resulting from optimization using static criticality prediction. That is, the horizontal axes of rows *Ess.* and *S* show different criticality predictions for the same set of TMR variants of the designs, while the other rows show the TMR variants generated in the respective optimization runs. Correlation results in Figure 51 are summarized by a linear regression through the solutions. Solutions with high TMR coverage (shown as triangles) were removed for its calculation.

The quality of an optimization process can be measured, for example, by the number of generations needed to converge to a solution as well as the fitness improvement made towards the optimization objectives.

The DSE approach presented in this work minimizes both *cost* (i.e., Essential Bits) and *criticality* (i.e., Critical Bits). The non-dominated solutions in each generation form a Pareto front that converges to an unknown *true* Pareto front. Progress between different generations can be compared by their Pareto front’s hypervolume. In a two-dimensional optimization problem, this is equivalent to the area between the Pareto front and a reference point representing an estimation of the worst possible result, which may be set to the worst result of the observed Pareto fronts or a slightly worse reference point [251]. A reference point at 103% of the maximum observed cost and criticality was chosen to calculate hypervolumes in this work.

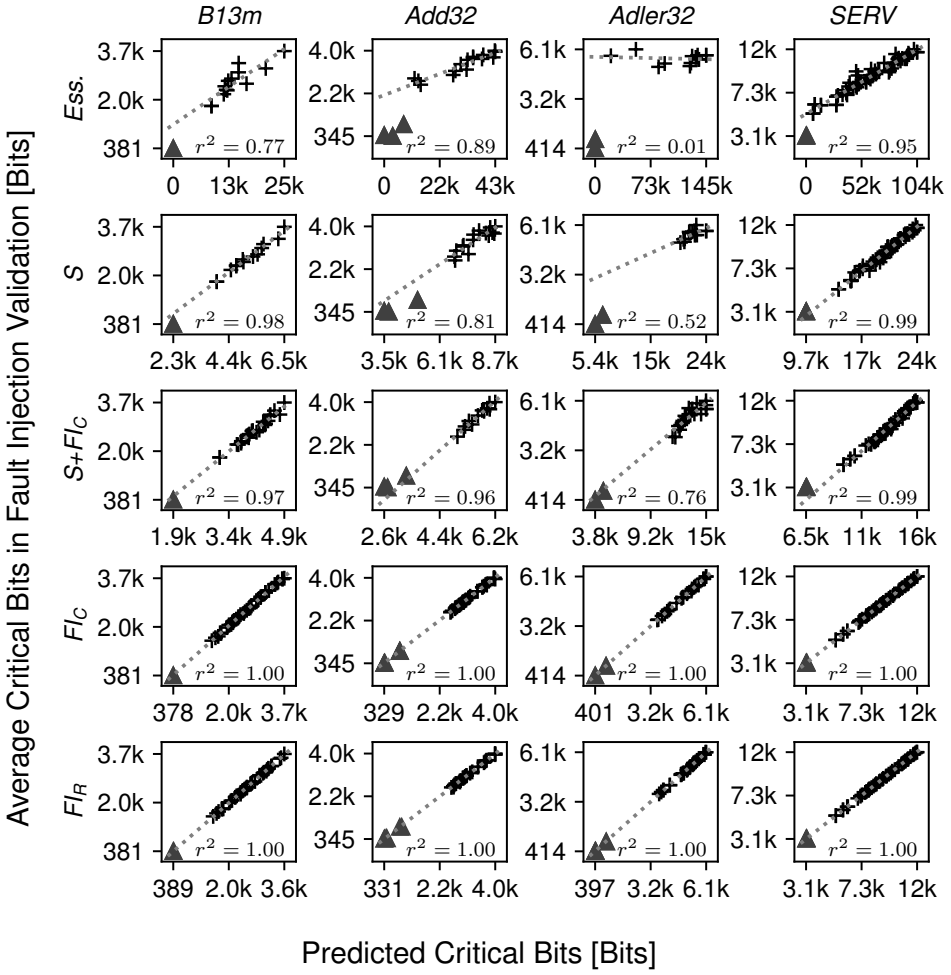


Figure 51: Correlation of Criticality Prediction and Fault Injection Results per Design and Fitness Variant

5.4.2 Quality of Optimization Results

Figure 52 shows the progress made by the performed optimization runs throughout 16 generations, measured in the percentage of the hypervolume of the Pareto front of the final 16th generation achieved by the respective preceding generation. This metric, also referred to as the Hyper-area Ratio (HR), allows to assess the fitness of the selected individuals of the initial randomly generated individuals and to track the contribution made by the following generations. Tracking the continual improvements made by each generation allows to validate the encoding, parameters, and operations chosen for the genetic optimization process. Furthermore, it delivers hints towards the development of a stopping criterion.

The results shown in Figure 52, however, only allow to compare the optimization strategies according to their convergence profile, not on the quality of the actual results generated in terms of cost and criticality. As it is known from preliminary analysis of the performance of the static criticality prediction, this prediction is still pessimistic (see *False Positives* reported in Section 4.5.1. Thus, to allow comparison between the optimization variants (partly) utilizing static prediction and those directly using fault injection results, all Pareto-optimal solutions selected from the first and last generations were subjected to fault injection on all twelve FPGAs used in optimization (Device Group 1) and a supplementary group of devices to judge the influence of PVT effects on the validity

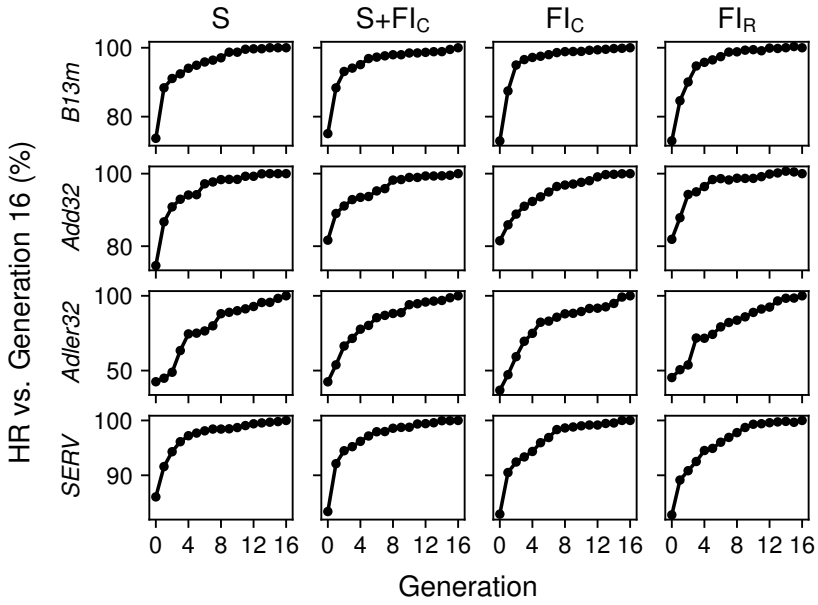
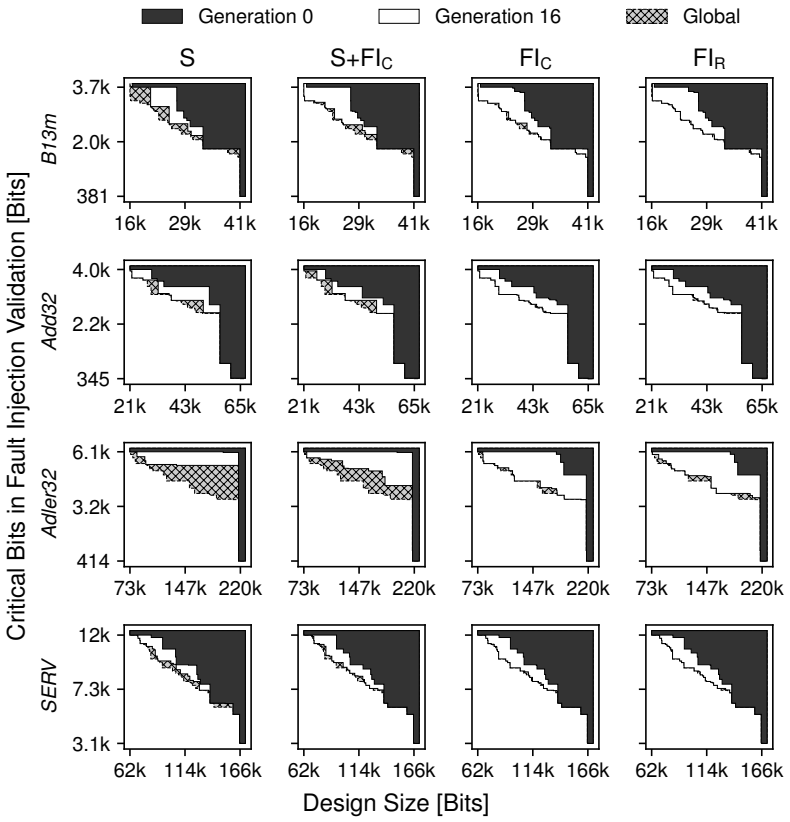


Figure 52: Convergence measured using Hyper-area Ratio (HR) per Generation

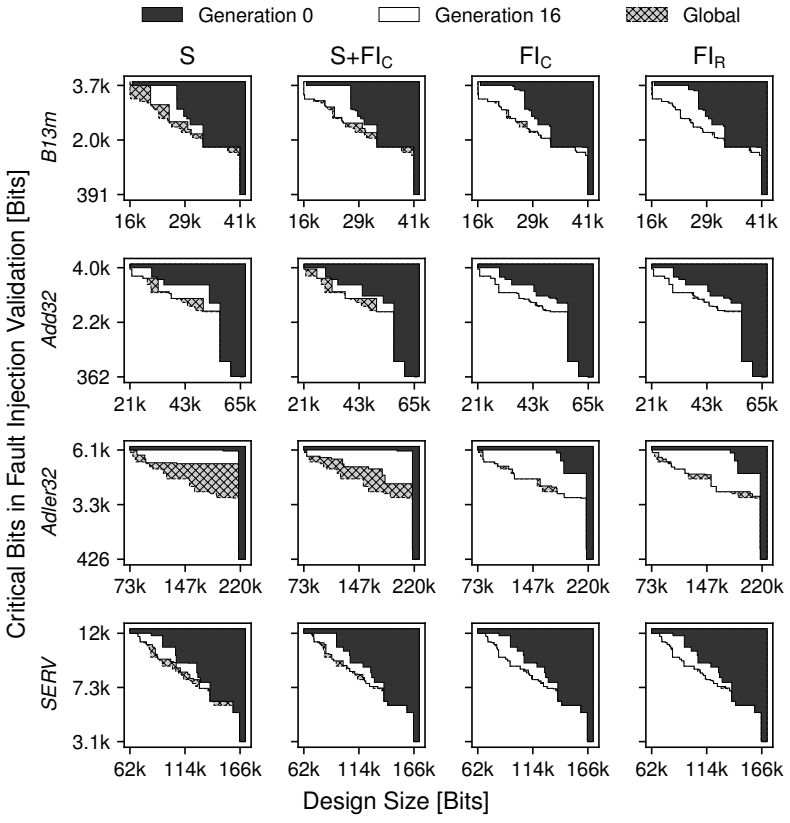
of the obtained optimization results (*Device Group 2*). The results in this section are subdivided into results obtained on each device group to allow these comparisons.

Figure 53 allows to compare the quality of the optimization results achieved in all considered designs with the evaluated fitness assignment approaches. The criticality-related fitness value assigned during optimization is replaced with the average number of critical bits determined in the subsequent fault injection experiments on all evaluated FPGAs. Thus, the improvement of the Pareto fronts and hypervolumes of the final generation over the initial randomly generated population can be seen here. As the *true* Pareto front is unknown, new virtual "global Pareto fronts" are constructed from the pool of solutions generated by all optimization runs per design. These represent the best solutions of all combined optimization runs and are plotted as hatched areas below the other hypervolumes in Figure 53. On top of the hatched areas, the hypervolumes of the final generations are plotted in white. Finally, in dark gray, the hypervolumes of the first generations are plotted on top. Thus, visible white areas represent gains made during optimization, while hatched areas represent potential gains not made by the respective variant.

Figure 53a shows data for the device group used for fault injection experiments during optimization, whose PVT variations thus may have affected the outcome of the optimization process. On the other hand, Figure 53b shows data obtained on the second device set, which was not part of the optimization process. Thus, significant differences between both device groups indicate that optimization results obtained on one device group are invalid on another device group, while closely matching results may indicate that optimization results can be generalized from the original device set.



(a) Device Set 1



(b) Device Set 2

Figure 53: Comparison of Size-Criticality Pareto Fronts after Validation

While the graphical representation shown in Figure 53 allows a quick judgment of the progress made during optimization, it does not allow for a precise comparison between the optimization variants. Multiple performance metrics exist to numerically compare Pareto fronts: The data graphically shown in Figure 53 is summarized in Table 78 as the HR performance metric. This metric represents the percentage of the global hypervolume covered by the Pareto fronts achieved by the different optimization runs when the resulting designs are subjected to validation experiments on both device sets. In addition to the HR performance indicator, the number of non-dominated solutions generated by the optimization variants (*Overall Non-Dominated Vector Generation (ONVG)*) as well as distance measures between the evaluated Pareto front and the global Pareto front (*Generational Distance (GD)* and *Inverse Generational Distance (IGD)*) [252] are commonly-used performance indicators.

Table 78: HR of Pareto Fronts of First and Last Generations

		HR of Optimization Variant (%)							
		Device Set 1				Device Set 2			
Design	Gen.	S	S+FI_C	FI_C	FI_R	S	S+FI_C	FI_C	FI_R
B13m	0	72	72	73	73	72	72	73	73
	16	86	95	98	100	86	95	98	100
Add32	0	76	80	81	81	76	80	81	81
	16	94	93	100	99	94	93	100	99
Adler32	0	26	26	38	38	26	26	38	38
	16	54	69	97	94	54	69	97	94
SERV	0	80	82	82	82	80	82	82	82
	16	94	97	100	100	94	97	100	100

Table 79: Comparison of Final Pareto Fronts using ONVG, GD, and IGD

(a) Device Set 1

	ONVG				*Relative GD (%)*				*Relative IGD (%)*			
Design	S	S+FI_C	FI_C	FI_R	S	S+FI_C	FI_C	FI_R	S	S+FI_C	FI_C	FI_R
B13m	4	8	18	23	100	73	50	34	100	43	12	4
Add32	7	6	19	14	85	100	14	19	100	99	4	8
Adler32	4	4	11	13	91	100	35	71	100	36	11	21
SERV	11	15	26	30	100	53	15	10	100	61	19	18

(b) Device Set 2

	ONVG				*Relative GD (%)*				*Relative IGD (%)*			
Design	S	S+FI_C	FI_C	FI_R	S	S+FI_C	FI_C	FI_R	S	S+FI_C	FI_C	FI_R
B13m	3	9	19	25	100	68	46	7	100	46	12	4
Add32	7	6	19	14	85	100	14	19	100	99	4	8
Adler32	4	4	12	13	95	100	30	74	100	35	10	21
SERV	11	15	26	30	100	53	15	10	100	61	19	18

Table 79 shows the absolute values for the ONVG indicator, as well as GD and IGD relative to the worst front (i.e., the distance to the global Pareto front of all generated solutions in generation 16 relative to the front farthest away from it). Data is shown for both device sets on which the designs were evaluated.

5.4.3 Execution Time

Both the total execution time (i.e., the total duration spent generating and evaluating design variants) as well as the execution time of the individual fitness function invocation can be derived this way. The resulting data is shown in Table 80 for all evaluated designs and optimization variants. The average runtime per fitness evaluation run is shown alongside with its variance.

Table 80: Execution Time Results

Total Execution Time (Gen. 0 – 16) per Design (hh:mm)				
	Optimization Variant			
Design	S	S+FI_C	FI_C	FI_R
B13m	10:06	34:59	62:50	88:29
Add32	10:46	45:20	74:33	113:05
Adler32	50:08	156:02	274:08	358:06
SERV	17:07	124:16	181:46	278:04

Average Fitness Function Runtime per Candidate (hh:mm ±:mm)				
	Optimization Variant			
Design	S	S+FI_C	FI_C	FI_R
B13m	00:04 ±:00	00:19 ±:02	00:36 ±:03	00:36 ±:04
Add32	00:04 ±:00	00:27 ±:17	00:49 ±:05	00:47 ±:07
Adler32	00:19 ±:06	01:27 ±:07	02:33 ±:16	02:31 ±:15
SERV	00:08 ±:01	01:16 ±:18	02:05 ±:16	02:04 ±:19

The composition of the fitness function runtimes depending on design and fitness function variant are shown in Figure 54. The bar charts show the average shares of total fitness function runtime spent in different phases of the respective fitness functions:

- *Impl.*: Implementing the designs – including TMR insertion and P&R.
- *Analysis*: Statically analyzing the designs' essential bits.
- *Injection*: Performing fault injection experiments.
- *Other*: Performing other tasks such as loading, generating, or compressing files.

The error bars show the standard deviation of the temporal shares, allowing to judge the variance of the fitness function runtime composition. For this plot, outlier invocations of the fitness function that took excessively long, e.g., due to temporary floating license inavailability, were selectively removed.

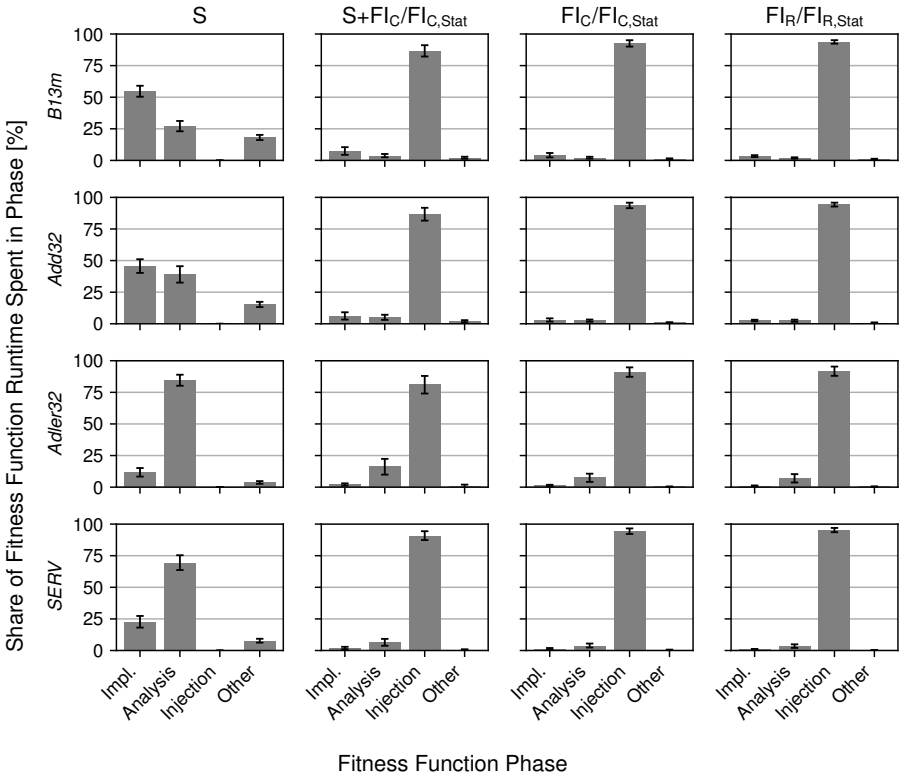


Figure 54: Composition of Fitness Function Runtime per Fitness Function Phase

5.4.4 Evaluation of 32-Bit CPU Designs

This final section provides results obtained from the application of the implemented optimization toolflow to the two substantial 32-bit CPU designs *NEORV32* and *LEON3*. Comparing Table 76 and Table 77, these designs are approximately an order of magnitude larger compared to the "small" designs for which results were described above. As noted in Section 5.3, the generation size was reduced to 32 individuals, and statistical sampling was used instead of exhaustive fault injection in both optimization (fitness functions $FI_{C,Stat}$, $S{+}FI_{C,Stat}$, and $FI_{R,Stat}$) and in validation experiments. Contrary to the "small" designs, validation fault injection experiments were carried out only on *Device Group 1*.

Results for the two larger designs are provided in the same format as is done in Sections 5.4.1 to 5.4.3 for the "small" designs: Figure 55a shows the correlation between the number of critical bits predicted by the respective fitness evaluation variant and the number of critical bits determined in statistical fault injection validation experiments. The convergence during the optimization process is depicted as a fraction of the final generation hypervolume in Figure 55b. The quality of results achieved by the optimization process for *NEORV32* and *LEON3* is represented graphically in Figure 55c and summarized by the HR, ONVG, GD and IGD metrics in Tables 81a and 81b. Table 81c shows both the total execution time and the average fitness function runtime that was necessary to evaluate the fitness of these larger designs. In a similar way as done for the smaller designs, Figure 55d shows the average composition of the fitness function runtime per design variant for the *NEORV32* and *LEON3* designs. Outlier fitness function execution times were removed as was done for Figure 54.

Table 81: Evaluation of 32-Bit CPU Designs: Summarized Results

(a) HR of Pareto Fronts of First and Last Generations

		HR of Optimization Variant (%)			
Design	Gen.	S	S+$FI_{C,Stat}$	$FI_{C,Stat}$	$FI_{R,Stat}$
NEORV32	0	65	65	65	65
	16	84	94	97	94
LEON3	0	46	46	46	46
	16	93	95	97	95

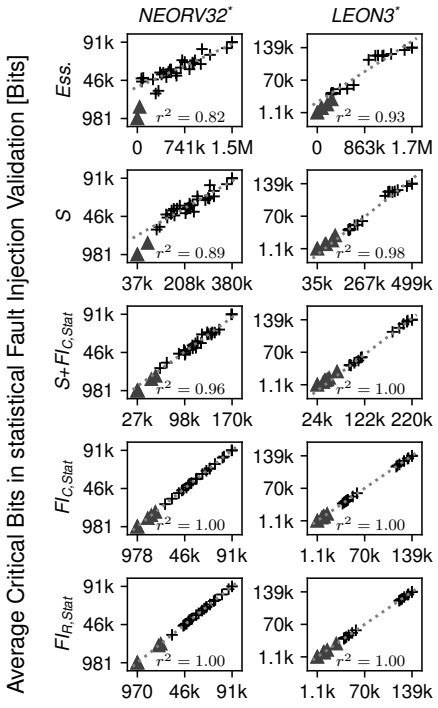
(b) Comparison of Final Pareto Fronts using ONVG, GD, and IGD

	ONVG				*Relative GD (%)*				*Relative IGD (%)*			
Design	S	S+FI_C	FI_C	FI_R	S	S+FI_C	FI_C	FI_R	S	S+FI_C	FI_C	FI_R
NEORV32	4	10	13	8	82	80	17	100	100	67	37	56
LEON3	7	9	13	14	90	100	61	8	66	77	79	100

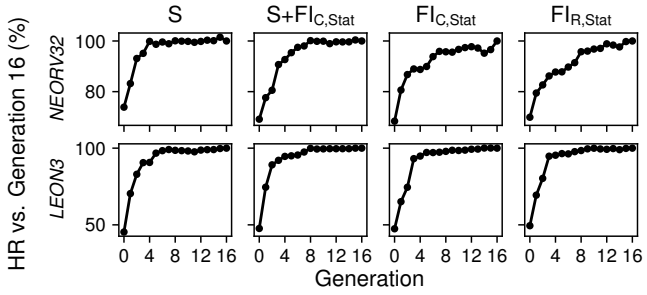
(c) Execution Time Results

Total Execution Time (Gen. 0 – 16) per Design (hh:mm)				
	Optimization Variant			
Design	S	S+$FI_{C,Stat}$	$FI_{C,Stat}$	$FI_{R,Stat}$
NEORV32	68:01	100:57	103:02	120:18
LEON3	61:10	109:55	103:52	117:46

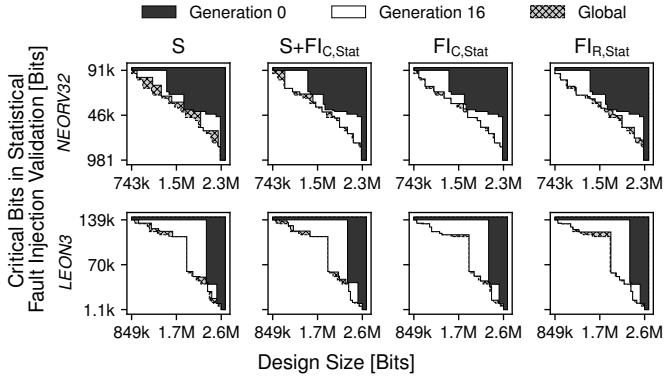
Average Fitness Function Runtime per Candidate (hh:mm ±:mm)				
	Optimization Variant			
Design	S	S+$FI_{C,Stat}$	$FI_{C,Stat}$	$FI_{R,Stat}$
NEORV32	01:24 ±:39	02:55 ±:39	03:01 ±:39	02:55 ±:45
LEON3	01:17 ±:33	03:05 ±1:00	02:58 ±:41	02:59 ±:50



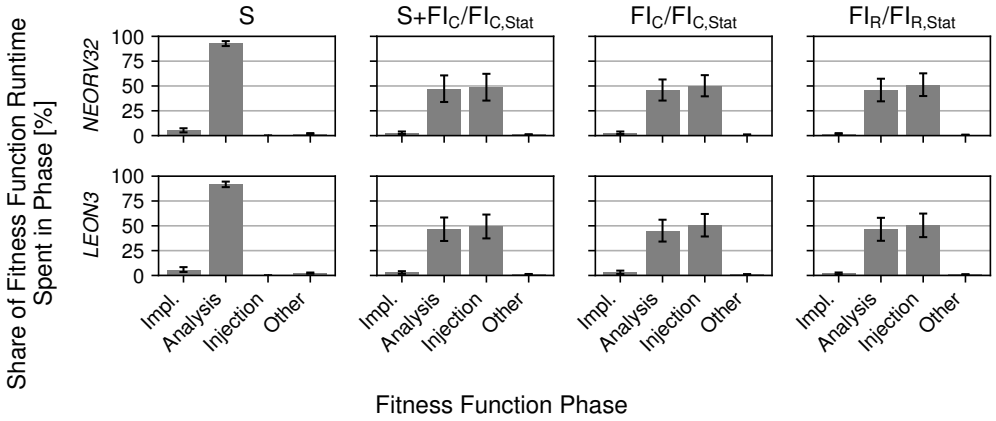
(a) Correlation of Criticality Prediction and Fault Injection Results per Design and Fitness Variant



(b) Convergence measured using Hyper-area Ratio (HR) per Generation



(c) Comparison of Size-Criticality Pareto Fronts after Validation



(d) Composition of Fitness Function Runtime per Fitness Function Phase

Figure 55: Evaluation of 32-Bit CPU Designs: Summarized Results

5.5 Discussion

Referring to **Q1** posed in Section 5.3, criticality evaluation approaches including fault injection information ($S{+}FI_C$, FI_C, and FI_R in Figure 51 and $S{+}FI_{C,Stat}$, $FI_{C,Stat}$, and $FI_{R,Stat}$ in Figure 55a) correlate reasonably well with the actual criticality observed on all devices used for fault injection in all case-study designs. It is expected that using these fitness assignment methods in optimization runs results in the selection of variants that offer higher improvements in criticality minimization. The accuracy of static criticality prediction (S) is design-dependent, correlating reasonably well with the number of critical bits in all designs except *Adler32*. Nevertheless, it reduces the number of fault injection experiments in all designs. It can be inferred from Figure 51 that the more fault injection information is added to the fitness assessment, the better the correlation between fitness value and the average of the criticality results after fault injection on all devices. Except with *Adler32*, also static criticality prediction results (S), and, to a limited degree, the prediction based on essential bits (*Ess.*), correlates reasonably well with the actual criticality. In the *Adler32* case, non-uniformly distributed false-positive bits (i.e., bits predicted to be critical, but non-critical in fault injection evaluation on a real device) presumably distort the prediction results. The pessimism brought on by these false-positive bits is in itself not a safety concern: no critical bits are being missed due to false positives. However, it may influence the quality of optimization results: When directly using static analysis results as fitness values, the number of critical bits in the design running on actual hardware differs from the assigned fitness. In such cases, the Pareto-optimal solutions of the final optimization round can be subjected to validation fault injection experiments, as shown in Figure 39. On the other hand, when employing fault injection for fitness evaluation, false positives lead to longer fault injection campaigns. As false positives stem from faults masked by circuit conditions, they can be addressed by incorporating functional knowledge about the design. For example, [124] combines simulation traces with technology-specific fault models to identify masked faults.

Rows FI_C and FI_R of Figure 51 as well as rows $FI_{C,Stat}$ and $FI_{R,Stat}$ of Figure 55a show that the average numbers of critical bits resulting from a fault injection evaluation on *all* evaluated devices correlate well with the number of critical bits predicted from fault injection results obtained on *one* device. Comparing the results on the device used for fault injection during optimization and the average criticality on the other devices during evaluation, all evaluation results for *B13m*, *Add32*, *Adler32*, and *SERV* were within a range of ±100 critical bits of the original estimate, with no apparent difference between the results obtained from the cached (FI_C) and resampling-based (FI_R) solutions. Thus, the fault-injection-based fitness variants are able to distinguish between solutions that differ by more than 200 critical bits – between 1.7% and 5.8% of the average critical bits of the smallest variants of the four designs (see Table 65). In the considerably larger *NEORV32* and *LEON3* designs that were evaluated using statistical fault injection, the device-to-device differences in critical bit estimates increased to (-800; +480) bits on *NEORV32* and (-1000;+1000) bits on *LEON3*. These results indicate the existence of PVT-induced inter-chip variation in fault injection results, as observed previously (see Section 4.4).

While results mostly stabilize after at most eight generations (see Figures 52 and 55b), *Adler32* (all runs) and *NEORV32* (runs $FI_{C,Stat}$, $FI_{R,Stat}$) take longer to converge. These designs, presumably, might have benefited from additional generations. On the other hand, the initial generation of *SERV* performed already well, offering less room for improvement.

Answering **Q2** from Section 5.3, all fitness assessment approaches converged to a set of solutions within the randomly chosen 16 generations for four designs (*B13m*, *Add32*, *SERV*, and *LEON3*), while *Adler32* and *NEORV32* might have benefited from additional generations. The selected size of the randomly generated starting population, as well as the generational population size, seem appropriate for the problem at hand. However, since comparatively few improvements over the

first generation could be made in the *Add32* and *SERV* cases, fewer generations or a smaller population could have reduced runtime with comparable results. All optimization runs improved on the randomly generated set of solutions and generated a set of diverse solutions well distributed across the cost-reliability range. In cases where static prediction correlated reasonably well with fault injection results according to Figures 51 and 55a (*B13m*, *Add32*, *SERV*, *NEORV32*, and *LEON3*), the coverage of the virtual global Pareto front by the *S* is better than in the case of *Adler32*. Results can be improved by adding fault injection information as done in the $S{+}FI_C$ variant. This shows that gains can be made even if not all bits are tested in fault injection. Using fault injection results (FI_C/$FI_{C,Stat}$ or FI_R/$FI_{R,Stat}$) usually yields the best Pareto fronts. There is little difference between the quality of optimization results obtained by runs using cached and resampled fault injection results in most cases, indicating that fault injection results from one FPGA can be generalized reasonably well to the other evaluated devices. These results are also reflected in the total share of the global Pareto front's hypervolume covered by the fronts resulting from the individual optimization strategies as shown in Tables 78 and 81a. Considering the non-dominated (i.e., Pareto-optimal) solutions generated by the individual optimization strategies as shown in the ONVG metric reported in Tables 79 and 81b, it can further be seen that the fault-injection-based optimization variants tend to generate solutions part of the global Pareto front when compared to the other optimization variants. This indicates that while static fault analysis effectively reduces fault injection workload, its precision must be improved before it can be used effectively as the sole source for criticality-related fitness evaluation.

The results obtained from validation experiments on two sets of FPGA devices (Device Groups 1 and 2, see Table 64) can be compared in Figure 53 as well as Tables 78 and 79. While no significant differences in the generated Pareto fronts can be visually spotted in Figure 53. This is also underlined by the similar HR performance indicators calculated from the post-validation Pareto fronts as reported in Table 78. However, there appear to be some differences between the two device sets in the *B13m* and *Adler32* designs when further scrutinizing the generated Pareto fronts: Evaluating these designs on the second device set results in fitness variant *S* generating one less non-dominated solution in the *B13m* design, while the other fitness variants generate one to two additional non-dominated solutions. In the *Adler32* design, this is only the case for the FI_C variant. Thus **Q3** from Section 5.3 may be answered as follows: Inter-device variations appear to have only a small effect on the validity of optimization results obtained on an initial set of devices when generalizing to a larger set of devices. It is thus reasonable to optimize a design for an initial device group and then deploy it to a larger set of AMD 7 Series devices.

Addressing **Q4** from Section 5.3, it can be seen in Tables 80 and 81c that $S{+}FI_C$, which only injects faults into selected configuration bits, performs faster than FI_C/$FI_{C,Stat}$ which (statistically) injects faults in all predicted-critical bits. Resampling (FI_R/$FI_{R,Stat}$) increases the runtime, while statistical sampling decreases it. The short runtime of the static-only optimization run (*S*) illustrates the temporal cost of running fault injection experiments. While static criticality prediction is the most temporally scalable fitness assignment variant, it relies on the number of predicted critical bits correlating well with the actually observed critical bits during fault injection. This requires a precise model for the impact of each configuration bit flip on the design. The fitness function runtime in variants relying exclusively on static analysis is dominated by the time it takes to iterate over the designs' essential bits and identify their consequences when flipped except for the smallest designs *B13m* and *Add32* as can be seen in Figures 54 and 55d. In the two smallest designs, both the execution time of the TMR insertion and P&R tools as well as *other* operations in the fitness function such as reading and writing files occupy a significant share of the total runtime. In designs subject to exhaustive fault injection (*B13m*, *Add32*, *Adler32*, *SERV*), in fault-injection-based fitness functions, the time needed to perform the fault injection experiments dominates the fitness

function's runtime. Finally, in the two largest designs – which were evaluated using statistical fault injection – due to the chosen sample size – static bitstream analysis and fault injection contribute to fitness function runtimes in roughly equal proportions.

Q5 from Section 5.3 can be addressed by assessing the nature of the increase in fitness function runtime with design size. Considering the scatter plots showing the relationship between runtime roughly estimated by the number of statically analyzed essential bits, and the number of bits evaluated using fault injection as shown in Figure 56. To a certain degree, these figures confirm a linear relationship between these numbers of bits and fitness function execution time, as is to be expected from the phase-based analysis of the fitness function runtime discussed above. However, especially with smaller designs, there is a component of static fault analysis runtimes that appears to be not linearly correlated with design size. This may be the effect of design structure, i.e., how complicated it is to determine the consequences of any individual configuration memory bit flip. For example, in designs with a large number of high-fanout nets, it appears likely that enumerating the fan-out cells for each fault in these nets may increase runtime even when compared to a design of similar size but smaller average fanout. If there are differences in complexity of design structure on a module-by-module basis, furthermore, replicating a module with a more complex design structure may significantly increase static analysis runtime when compared to a design variant that replicates large, but less complex module.

To judge the overall impact of bitstream analysis and fault injection on fitness function runtime, Figure 56 shows the individual fitness function runtime roughly estimated from the number of essential bits (i.e., bits subjected to static fault analysis) and bits tested in fault injection on the horizontal axis, and the recorded fitness function runtime on the vertical axis. These bits have been associated with a runtime the average prediction speed of 400 bit/s derived in Section 4.5 and an estimated fault injection rate of 3 bit/s. In a similar way as above, outliers were removed for this analysis.

In summary, fitness function runtime (and thus optimization time) in the presented evaluation is linearly related to design size in terms of bitstream size, which is more noticeable at larger design sizes. Scalability may be improved significantly by reducing fault injection runtime in fitness function variants based on fault injection. This may be done – as demonstrated in this work with *NEORV32* and *LEON3* – by adapting population size and employing statistical fault injection [175]. Furthermore, reducing the number of false positives in static criticality prediction (as observed in Table 65) can significantly reduce fault injection effort: fewer bits needing to be subjected to fault injection directly reduces runtime. This can be done by incorporating design-specific information, as shown in [124] where simulation traces are used to identify masked faults.

Alternatively, also speeding up the fault injection procedure itself reduces overall fitness function runtime. This can be done by implementing faster partial reconfiguration controllers, that, e.g., perform fault injection and experiment sequencing not using a soft CPU but in hardware using a state machine. The time $t_{experiment}$ needed to inject a fault using the procedure discussed in Section 5.2.2 is roughly characterized by (1) n_{cwords} the number of words that must be read and written back via the configuration interface, (2) f_{config}, the speed of the configuration interface, (3) t_{test} the time needed to perform a test run of the design (doubled, to perform a check iteration to ensure correct operation), (4) t_{comm}, the time needed to communicate the bit to be injected to the FPGA, and for it to communicate back the experiment's results, and t_{boot}, the time required to reboot the FPGA to restore a clean configuration. Finally, t_{ovhd} is a catch-all term that sums up the overhead introduced by inefficiencies in the implementation. This is shown in Equation (5.2). The time to access n_{words} of the configuration is doubled as the configuration needs to be read and written to inject a fault. t_{test} is doubled because before each fault injection experiment, according to Section 5.2.2, the correct operation of the design is verified.

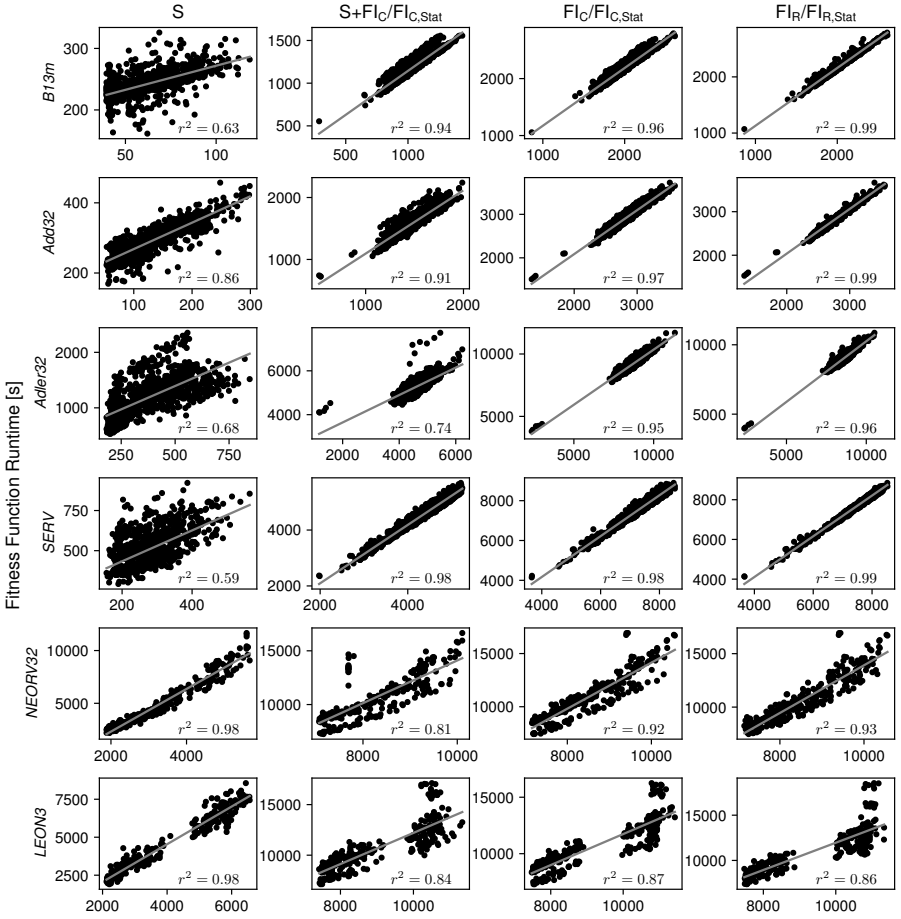


Figure 56: All Designs: Correlation of Essential and Tested Bits and Fitness Function Runtime

$$t_{experiment} = 2 \cdot \frac{n_{words}}{f_{config}} + 2 \cdot t_{test} + t_{comm} + t_{boot} + t_{ovhd} \tag{5.2}$$

In AMD 7 Series FPGAs, the smallest bitstream unit accessible using the partial-reconfiguration fault injection approach via ICAP as done in this work is a 101-word configuration frame. For each bit to be injected, two 101-word frames must be written or read via ICAP: a dummy frame, and a frame containing configuration data. Additionally, approx. 64 words of protocol overhead for communication with the device's configuration logic (e.g., communicating the frame's address, unlocking and applying the configuration) is necessary, arriving at $n_{words} = 266$. Using an ICAP clock frequency of 50 MHz as has been applied in the evaluations in this chapter, the time necessary for configuration access can be estimated to be $2 \cdot \frac{266}{50\ MHz} = 10.64\ \mu s$. This time can be halved by applying the maximum ICAP clock frequency of 100 MHz.

A design requiring 10^4 clock cycles to be tested requires a time of $2 \cdot t_{test} = 400\ \mu s$ at 50 MHz. Assuming a UART communication at 115.2 kBd as used in this evaluation and a communication volume of 20 bytes[128] requires $t_{comm} = 1736.1\ \mu s$. Finally, the `calc_config_time` provided by Vivado 2019.1 reports a minimal, typical, and maximum configuration time of 93.6 ms, 137.8 ms,

[128] Injection Request Message (14 bytes): 2 bytes header, 10 bytes frame address, 2 bytes CRC16
Results Report Message (6 bytes): 2 bytes header, 2 bytes injection result, 2 bytes CRC16

and 270.7 ms, respectively for an AMD Artix 7 35T FPGA. It shall be noted that this time may be reduced by bitstream compression that uses the `MFWR` register in the configuration logic to write identical frames to multiple locations in the FPGA configuration [30]. This time strongly depends on how much of the FPGA is utilized by the design. The typical uncompressed configuration time of 137.8 ms is thus assumed as t_{boot}.

Comparing the total required test time of 140 ms according to Equation (5.2) with the 333.3 ms fault injection time observed in the evaluation in this work (see above) leaves a t_{ovhd} of approx. 193 ms to be optimized using a hardware implementation, allowing for a speed-up of up to 2.37×. The biggest factor, as shown above, is FPGA reboot time. Other approaches [140, 141] do not reboot the FPGA but correct the fault within the bitstream and reset the design. However, this does not account for potential side-effects of an injected fault, e.g. data stored in BRAM or distributed RAM that is modified due to the consequences of a fault. This data can only be restored by reloading the configuration of these resources, which is done by a reboot of the FPGA. Additionally, [140] proposes parallelization of the fault injection problem to multiple identical devices. While this spreads the fault injection effort across multiple devices, and thus shortens overall fault injection runtime, potential PVT effects need to be considered when merging the criticality results obtained on these devices (see Section 4.4).

Results achieved by the proposed approach are compared against the most closely related works of partial TMR analysis [142] and DSE approaches [175, 179] to address **Q6** from Section 5.3 For these comparisons, we employ a Benefit-Cost-Ratio (BCR) that relates reported design size increase and reliability improvement.

The authors of [175] report that their multi-objective optimization reduced criticality in the range of 16%–52% compared to an initial design, while not reporting a design-size-related cost component. Excluding full TMR and solutions offering less than 5% reliability improvement, our approach identified a wider variety of solutions in four of the five designs: $\mathrm{FI}_{\mathrm{C(,Stat)}}$ identified solutions reducing criticality by 13%–90% (*B13m*), 7%–91% (*Add32*), 6%–72% (*SERV*), and 6%–84% (*NEORV32*) compared to a simplex design. Solutions in the range of 8%–41% were identified for *Adler32*.

In [142], cost is reported as the share of netlist cells of the ITC'99 B13 circuit [164] covered by TMR and the number of inserted voters – excluding interconnect cost. Reliability improvements are reported as the reduction in critical bits compared to a simplex design observed during fault injection. The best reported design variant (*"TF 2nd largest"*) achieves a 22% reduction of critical bits by triplicating 12% of the cells and inserting five voters. This yields an approximate overhead of 29% (100 original cells, 24 cell replicas, five voters), and thus an estimated BCR of 0.76 (22% vs. 29%).

In [179], partial TMR candidate cost is reported as the increase in bitstream size vs. a simplex design, and reliability improvement as an increase in MTTF under fault injection. Under the assumption of single-bit flips and scrubbing to prevent accumulation, MTTF is inversely proportional to the number of critical bits [253], allowing a comparison. For the "best" variant of four designs, [179] reports (i) 17% MTTF increase vs. 82% overhead (BCR=0.21), (ii) 4.5% vs. 71.5% (0.06), (iii) 4.4% vs. 23.7% (0.18), and (iv) 11.7% vs. 20.1% (0.58) compared to a simplex design (avg. BCR=0.26).

For a direct comparison with [142] and [179], we extracted the solutions with the best BCR in a similar cost and reliability increase region (overhead $\leq$ 80%, criticality reduction $\geq$ 20%) generated by our approach. The solutions fulfilling these criteria with the highest BCR are shown in Table 82a along with the optimization variant that identified them. It can be seen that the average BCR achieved by our approach in the specified region (0.52) clearly outperforms the average (0.26) reported by [179]. Except for design (iv) in [179] (BCR=0.58), this holds for each individual design

Table 82: BCR of Best Identified Solutions

(a) Comparison Range: overhead $\leq$ 80%, criticality reduction $\geq$ 20%

	Opt. Variant	BCR	Overhead	Crit. Reduction
B13m	FI_R	0.71	38%	27%
Add32	FI_C	0.55	38%	21%
Adler32	FI_C	0.37	67%	25%
SERV	FI_C	0.45	49%	22%
NEORV32	$S+FI_{C,Stat}$	0.77	26%	20%
LEON3	–	–	–	–
Average		0.52	44%	23%

(b) Comparison Range: overhead $\leq$ 80%, criticality reduction $\geq$ 15%

	Opt. Variant	BCR	Overhead	Crit. Reduction
B13m	FI_R	0.71	38%	27%
Add32	FI_C	0.55	38%	21%
Adler32	FI_C	0.41	37%	15%
SERV	FI_C	0.50	32%	16%
NEORV32	$S+FI_{C,Stat}$	0.77	26%	20%
LEON3	$FI_{C,Stat}$	0.26	58%	15%
Average		0.53	38%	19%

evaluated in this work. It can be seen that no such solution is available for *LEON3*. Relaxing the criteria to a minimum criticality reduction of 15% results in the solutions and BCRs shown in Table 82b. This yields a BCR of 0.26 (the average BCR in [179]) for *LEON3*, achieved by $FI_{C,Stat}$. Relaxing these criteria also yields better BCRs for *Adler32* (0.41 vs. 0.37) and *SERV* (0.50 vs. 0.45).

A BCR of 0.77, slightly exceeding the one calculated from the data reported in [142] (0.76) is achieved in the comparison range by $S+FI_{C,Stat}$ for NEORV32. When relaxing these criteria to a min. 10% criticality reduction, a design candidate with a BCR of 1.3 was identified for the *B13m* design by both FI_C and FI_R: Critical bits could be reduced by 13% with an overhead of just 10%, thus outperforming the 0.76 BCR observed in [142] in the low-overhead region. In the same region, a candidate of the *Adler32* design with a BCR of 0.91 was identified by FI_C (10% criticality reduction), and one in the *NEORV32* design with a BCR of 0.75 (18% criticality reduction). In contrast to [142], however, this is achieved by a black-box approach. As [142] concerns a way of *implementing* TMR insertion, it may be combined with our optimization approach.

5.6 Summary

This chapter proposes an approach for addressing the selection problem faced when introducing partial redundancy into an FPGA design in order to cost-efficiently improve its resilience against configuration memory bit flips: As shown by recent works such as [179, 232], module-based partial TMR is one technique effective at doing so. Applying this technique, requires selecting concrete module(s) to replicate, which – while improving the design's reliability – also increases its size.

The bit-level effects of these changes are difficult to predict from an RTL or netlist representation of the design alone, as placement and routing significantly affect the design's reliability. The outcome of placement and routing cannot be predicted to a sufficient degree to generate design variants effectively protected against single-event upsets in their configuration memory. This is mainly due to the complexity of the routing problem, which is solved via various heuristics in contemporary FPGA implementation tools. Thus, the approach presented in this chapter views the problem of cost-effectively selecting modules to be protected via a reliability-improvement technique such

as TMR as a design-space exploration problem that it addresses via a genetic algorithm. The selection of submodules to be replicated is encoded as a binary vector that forms the chromosome of a solution. Cross-overs and mutations of these chromosomes are applied in order to generate variants of the preceding solutions in order to generate module selections with Pareto-optimal cost-reliability properties.

The fitness of these solutions is measured in terms of area cost (i.e., the bitstream size) and design (un)reliability (i.e., the remaining number of critical bits). Due to the complexity of predicting placement and routing (and thus post-routing sensitivity to configuration memory bit flips), the fitness function performs TMR insertion, placement, and routing of each design variant to instantiate the chromosome (i.e., *"genotype-to-phenotype mapping"* [238]). Afterwards, the bitstream is analyzed statically to identify potentially critical bits using the approach proposed in Section 4.5. Ideally, this modeling is precise enough to lead to cost-reliability-efficient results on its own. Alternatively, it may be supplemented by fault injection experiments. While temporally expensive, this allows to accurately determine a solution's configuration-level soft-error sensitivity.

The end result of this optimization process is a set of Pareto-optimal solutions in terms of implementation cost and soft-error sensitivity, from which one or more implementations may be readily chosen depending on application or mission requirements. As redundant logic may also increase power consumption, this choice could take mission-phase requirements into consideration: A less redundant, but more power-saving implementation from this Pareto front may be chosen for a less critical mission phase, while a power-hungry but highly reliable variant may be chosen for a critical part of a mission. The respective variant may be loaded using partial reconfiguration depending on a pre-defined mission profile.

This chapter furthermore evaluates the proposed approach on an implementation for AMD Artix 7 FPGAs, with the static criticality prediction based on fault modeling for these devices discussed in Section 4.5. The evaluation succeeded in generating optimized variants for six FPGA designs of different sizes, functionality, and design styles, while evaluating different approaches for fitness evaluation: purely static bitstream analysis, fault-injection-based criticality analysis, and an approach combining these two approaches. Furthermore, a variant that re-evaluates each design variant on multiple devices in order to address potential PVT effects influencing fitness assignment results was evaluated. To improve the scalability of fault-injection-based fitness assignment variants for larger designs, statistical fault injection was evaluated for two designs as a basis for fitness assignment. All fitness assignment variants after 16 generations produced design variants significantly improving upon sets of randomly generated initial solutions via a genetic optimization process following the NSGA-II algorithm [173]. The validity of these results was investigated by comparing the results of validation fault injection experiments of the Pareto-optimal results of the first and last generations. To further investigate the influence of PVT effects on the validity of the optimization results, a subset of four of the six designs was validated using fault injection on two sets of FPGAs: The designs were evaluated on the twelve FPGA devices used during fault-injection-based optimization, and on a second, set of otherwise identical FPGA devices not used in optimization. From the results of this PVT validation, it can be concluded that generalization from results obtained on one device may be – within reason – generalized to other devices of the same type. This is possible in the case of the presented optimization approach, as this approach's fitness evaluation uses the *number* of critical bits regardless of their individual function. While the latter is significantly impacted by inter-device variation, the former is much less so affected as discussed in Section 4.4.4. Finally, the presented implementation of the proposed approach is compared to the performance of related work, indicating that under certain conditions the presented DSE approach is able to outperform these implementations.

Table 83 evaluates the presented approach and its implementation against the tool-level re-

quirements formulated in Section 3.1.1. It can be seen that the proposed solution fulfill these requirements well, providing a way to generate area- and reliability-optimized design variants that incorporate fault-masking mitigation techniques. Furthermore, it has been shown how statistical fault injection enables the approach to be used on designs not suitable for exhaustive fault injection experiments due to runtime constraints.

Table 83: Evaluation of the Proposed Reliability DSE Approach against Tool-Level Requirements

Requirement	Evaluation	Explanation
REQ:TOOL:MASKING	●	The approach is based on TMR and thus is able to mask configuration bit errors until scrubbed. It may also be extended with fault detection logic as supported, for example, by Synopsys Synplify Premier. In principle, the approach is also applicable to other masking (e.g., NMR) redundancy-insertion approaches.
REQ:TOOL:DSE	●	The approach explores the design space provided by the module selection problem inherent to module-based partial TMR.
REQ:TOOL:AREA_RELIABILITY	●	The presented design-space exploration approach considers both a candidate's area (via the number of essential bits) as well as its (un)reliability (via the number of unprotected predicted-critical bits).
REQ:TOOL:PARETO	●	The approach is based on NSGA-II [173]. Thus, it generates Pareto-fronts from which the solution(s) with desired properties can be selected by the designer.
REQ:TOOL:PNR	●	Placement and routing are performed as part of the instantiation of each chromosome, allowing to consider the resulting effects on the design's reliability.
REQ:TOOL:BIT	●	The presented implementation of the approach performs both bit-level fault modeling following the findings discussed in Chapter 4 for the AMD Artix 7 FPGAs.
REQ:TOOL:PVT	●	PVT influence on the optimization process is addressed by optional re-evaluation of the fitness of solutions on different device instances (FI_R/$FI_{R,stat}$). The evaluation of the approach thoroughly addresses possible influences of PVT by performing validation experiments for each Pareto-optimal solution in the first and last generations on all twelve FPGA devices part of the optimization process for all designs. Additional fault injection validation was performed on a second set of twelve FPGA devices not used in the optimization process.

While the presented approach fulfills the requirements formulated in Section 3.1.1, its implementation discussed in Section 5.2 and its evaluation shown in Section 5.3 are limited in the following ways:

- *Scalability*: The proposed toolflow was evaluated on case-study designs ranging from 184 to 6997 cells. In these designs, total execution time was up to 358 hours for 16 generations. Scalability can be improved by reducing fault injection effort – e.g., by improved static criticality prediction, parallelization [140], or statistical fault injection (as demonstrated on *NEORV32*). The reduced runtime of the $S{+}FI_C$ optimization variant suggests interleaving optimization using static prediction with rounds based on fault injection as a compromise. Static criticality prediction can be improved by using additional information about the design. An example is shown in [124], where simulation traces are employed to identify which faults are masked by circuit conditions. Furthermore, functional design information may also be used to improve the quality of the initial population, thereby reducing the number of generations necessary to arrive at satisfactory results. Additionally, the time necessary to place & route the TMR'ed design variants must be factored into scalability considerations.

- *Hierarchical Design Structure*: A hierarchically structured design is required to apply the proposed optimization approach. In line with state-of-the-art approaches such as [179, 232], it is assumed that the "natural" hierarchical structure of the design captures an appropriate design space for the DSE approach to be applied effectively. This assumption likely does not hold for all designs equally: On the one hand, designs with a shallow hierarchy of highly granular modules may result in a large number of voters, limiting the resource-efficiency of the resulting TMR candidates. On the other hand, designs with a small number of very large modules and designs with severely unbalanced hierarchies may limit the design space within which the DSE approach attempts to find cost (resource usage) vs. reliability trade-offs. While [179, 232] as well as the approach proposed in this dissertation generate effective results based on that assumption, future work should perform a systematic investigation into the effects of design hierarchy granularity on the effectiveness of hierarchy-based selective TMR DSE approaches.

  Furthermore, focusing on the design's RTL-derived module hierarchy implies the need to preserve the original design hierarchy as far as possible. This includes disabling cross-hierarchy optimization and flattening steps, which may negatively affect logic density. To extend the approach to highly optimized flattened designs with no modular hierarchy, clustering approaches are needed to (re-)generate groups of cells whose resource and soft-error-sensitivity impacts when replicated can be weighed against each other by the DSE approach. Options include bus- and name-based clustering [254] as well as approaches that identify feedback structures to cluster logic elements such as [161, 165, 142].

- *Design Testability*: The feasibility of fault-injection-based fitness evaluation also depends on the effort needed to determine bit flip criticality. A self-checking design (e.g., using BIST) that quickly and without extensive external interaction confirms its correct functionality is preferred over one needing extensive amounts of time and large test sets to assess bit flip effects.

- *FPGA Technology Scope*: The presented implementation targets solely AMD's Artix 7 FPGA family. However, the main concepts of the proposed DSE toolflow and fitness evaluation apply to SRAM-based FPGAs regardless of the specific device type. Furthermore, the presented implementation only *implicitly* protects data storage elements (FFs and BRAMs) in the targeted FPGA design as far as they are part of modules to be replicated. At first glance, it appears likely that a module that is highly sensitivity to configuration bit flips will also be

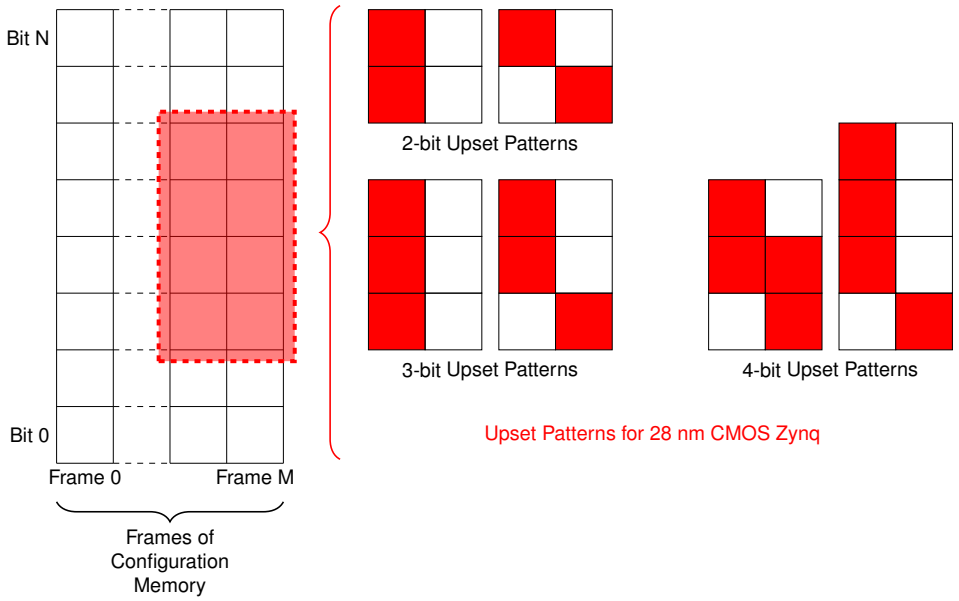


Figure 57: Most Common MBU Upset Patterns for 28 nm CMOS Zynq under Radiation Exposure as reported in [255]

highly sensitivity to bit flips in its data storage elements. This is due to the assumption that these faults experience similar propagation conditions in the downstream logic. However, this is not a given since these faults may affect the design in different ways. The effect of bit flips in storage elements may be more straightforward to predict from analyzing the original design once, as they are less likely to be impacted by the specific placement and routing. Thus, ideally, fitness evaluation may be informed by a storage-element vulnerability analysis in the form of “design meta-information” as shown in Figure 39. This analysis may also be used to generate a more “informed” initial generation.

- *Fault model*: Determining the effect of an SEU is relatively straightforward compared to MBUs. MBUs drastically increase the size of the fault space by virtue of the larger number of possible combinations of single-bit errors. Figure 57 shows the most common multi-bit upset patterns observed in [255] during radiation experiments with energies of 16 to 151.18 MeV for an AMD 7-series Zynq FPGA. This device is from the same device family as the AMD Artix 7 FPGA used in this chapter. MBUs involving two, three, and four flipped bits accounted for the overwhelming majority of observed MBUs (responsible for 81%, 12%, and 4% of the MBU-related radiation cross-section, respectively). Faults involving five and more flipped bits accounted for the remaining 3% of the MBU-related cross-section. When considering multi-bit upsets in addition to single-bit upsets, the consequences of these fault patterns must be predicted or evaluated using fault injection, thereby significantly increasing fault prediction or injection effort. However, as [206] shows for the same technology, SEU account for the vast majority (89.7%) radiation effects. In general, the SEU model applied in this work is widely used and accepted for FPGA configuration errors.

- *PVT effects*: Evaluations were conducted on two sets of twelve FPGA devices each under controlled environmental conditions to limit PVT influence on evaluation results and to quantify the effect of these variations. Their generalizability depends on the severity of the influence of PVT effects on fault injection results in the used device type.

Chapter 6

Selective SEU Mitigation for Storage Elements in High-Level Synthesis

The fault modeling approach described in Chapter 4 and the design-space exploration approach proposed and evaluated in Chapter 5 focus on the resilience of a design for an SRAM-based FPGA against upsets in the device's configuration memory. However, not only the configuration is susceptible to bit flips in FPGA designs, but also the intermediate storage elements employed by the design (i.e., flip-flops or latches and block RAM, distributed RAM).

The contribution of bit flips to the design's failure rate can be modeled as shown in Equation (2.1). As discussed in Section 2.3, not all storage elements are equally critical to the design's correct functionality: not all *essential* configuration bits are *critical* bits, which thus form a subset of the essential bits. This property is captured for the configuration by the average propagation probability $p_{\mathrm{prop,CRAM}}$. This concept can be applied to the design's user-defined storage elements as well, as not all storage elements may be equally critical to the design's correct functionality, and is captured by $p_{\mathrm{prop,FF}}$ and $p_{\mathrm{prop,BRAM}}$ in Equation (2.1).

In FPGA technologies that are immune (e.g., (Anti)fuse-based such as Microchip's RTSX and RTAX devices) or less susceptible (e.g., Flash-based such as Microchip's RT-PolarFire and RT4G devices)[129]) to bit flips in the configuration memory, the configuration does not contribute significantly to the design's bit-flip-related Failures in Time (FIT) rate (by providing a λ_{CRAM} of zero or negligible value), while the influence of the user-defined storage elements remains. In SRAM-based devices, the impact of configuration memory faults is reduced by the approach described in Chapter 5 by reducing the number of total critical bits (N_{CRAM}). This is achieved by introducing redundancy into the design to mask certain faults.

Protecting blocks of logic against SEUs in configuration memory implicitly also replicates flip-flop and BRAM cells, and thus reduces the probability of critical block memory and flip-flop bit flips ($p_{\mathrm{prop,BRAM}}$ and $p_{\mathrm{prop,FF}}$). However, single-bit flips in memories and register banks may be mitigated more efficiently by applying ECC schemes instead of triplication. Modern FPGA technologies such as AMD's 7 Series support this by providing built-in ECC encoder and decoder functionality in their memory blocks [100].

Protecting storage elements against bit flips, be it via triplication or ECC, comes at the cost of additional resource utilization, as well as reduced clock frequency due to longer critical paths, and increased energy usage due to a larger number of (potentially toggling) storage elements.

[129] See `https://web.archive.org/web/20240225005020/https://www.microchip.com/en-us/products/fpgas-and-plds/radiation-tolerant-fpgas`, accessed 2026-01-05

This motivates, in a similar way as discussed for the configuration memory in Chapter 5, the development of analysis tools to separate the design's user-defined storage elements into more and less critical ones. These tools thereby provide data that guides efficient (partial) soft-error protection. To provide this data, a way of analyzing the criticality of a design's storage elements is required. While bit flips in configuration memory potentially change the circuit structure, and thus may be masked due to technological or PVT-related effects (see Chapter 4), the effects of a bit flip in a user-defined storage element solely depend on the design's structure and its state. Thus, analysis at a higher abstraction level may deliver satisfying results at faster analysis runtimes than hardware fault injection experiments.

As the fault effects primarily depend on the design's structure, these analysis results may also serve as a proxy for design-related configuration memory criticality information. A future application of this approach may thus be the integration into the DSE approach described in Chapter 5, by (1) informing the generation of a starting set of replication solutions by preferring replications that protect larger numbers of especially critical storage elements, or (2) providing design-related fault propagation information that may be used to improve static criticality prediction.

High-Level Synthesis (HLS), that is the generation of digital logic designs from a high-level language such as C, C++, or a domain-specific modeling language such as MATLAB instead of from RTL, promises higher productivity as high-level algorithmic models of the desired system computations do not have to be manually translated into a description that is suitable for logic synthesis. In [256], 51 studies of HLS applications from 2017 – 2024 are reviewed among others for the Quality of Results (QoR) (i.e., utilized FPGA resources, latency, and clock frequency) as well as the development effort necessary to implement the same application using HLS and conventional manual RTL coding. While the solutions implemented by HLS are frequently outperformed by manual RTL implementations with respect to QoR, [256] found significant productivity improvements are possible when using HLS. Future generations of HLS tools may even utilize Large Language Models (LLMs) to further improve efficiency [257, 258]. Motivated by the potential advantages of HLS toolflows reported by [256] – and thus the potential future proliferation of the HLS approach – the criticality analysis and selective soft-error mitigation concepts discussed in this chapter focus on HLS-generated designs.

In particular, Section 6.1 describes a model for single-bit errors in user-defined storage elements, as well as an approach for analyzing a design's storage elements concerning their criticality. Section 6.2 proposes an implementation of this concept for using the open-source C/C++-to-RTL HLS tool PandA Bambu [227] to demonstrate this approach. This implementation is evaluated according to the strategy described in Section 6.3. The results of this evaluation are provided and discussed in Section 6.4. Finally, Section 6.5 summarizes this chapter and its contributions.

6.1 A Profiling-Based Approach

In Chapters 4 and 5, configuration faults in SRAM-based faults are modeled as permanent (until scrubbed) single-bit flips – with associated technology- and circuit-level consequences. The rationale for applying this model is that the configuration memory is generally not (over)written during the normal operation of the design. Essential and critical configuration bits are thus only differentiated by the masking capabilities of the design, technology, or device.

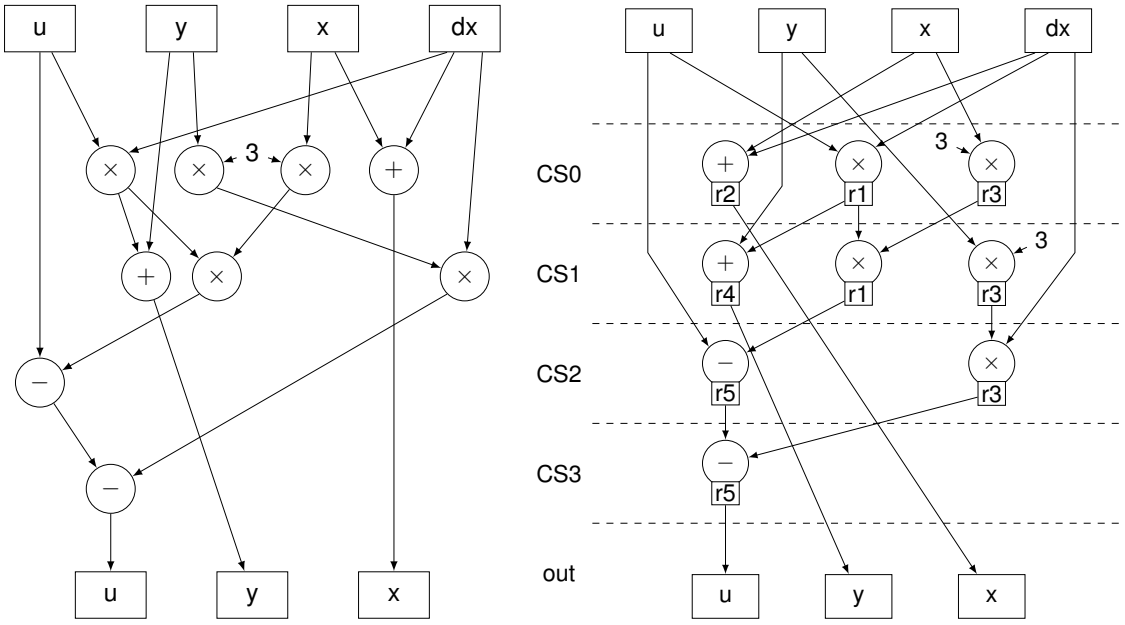
On the other hand, values stored in flip-flops, register banks, or block memory are often constantly overwritten within the normal operation of the design's circuit implementation. The design thus may "self-correct" a bit flip in a storage element if the bit flip occurs at a point in time where its flipped value has no chance of propagating further into the design before being overwritten with

the next (correct) value. In [259], the *RVF* is introduced to capture this notion for values stored in CPU registers. It is defined as the fraction of a register value's lifetime during which the value is vulnerable against bit flips due to being used in one or more upcoming instructions. It thus describes the probability for a fault that randomly occurs during the lifetime of this register value to propagate further through the execution of the program in question. The *susceptible lifetime* of a register value is described as the interval between the write access that sets the register to the value under consideration, and the last read access that transfers this register value to another location in the executing processor. This interval is defined with respect to register access patterns, where the total *susceptible lifetime* of a register value is the sum of all intervals between a write access and a read access ("W–R") and between two read accesses ("R–R"). A "W–W" interval indicates that the register value has no *susceptible lifetime* at all (as it is never used), while a "R–W" interval counts towards the register value's overall lifetime, but not its *susceptible lifetime*, as any bit flip after the read access is overwritten – and, thus, corrected – by the following write access.

In HLS, a design description in a High-Level Language (HLL) (e.g., C or C++) is first parsed by the HLS tool's front end. After common optimizations such as eliminating dead code, this step results in a Control Flow Graph (CFG) of the function to be synthesized, which is composed of *basic blocks* (i.e., sequences of operations without branches or jumps) as its vertices. The edges represent the control flow of the function [260]. To synthesize logic that performs the operations described by the basic blocks, each basic block's DFGs is analyzed. The DFG contains the input, intermediate, and output values needed for the computation of the basic block's functionality as its edges, and the necessary operations as its vertices. During synthesis, operations are allocated and bound to adequate hardware resources, and a schedule is computed to efficiently share hardware resources between like operations according to the constraints provided to the HLS tool. As the basic block's functionality usually cannot be calculated within a single clock cycle, the schedule mandates that certain values must be stored in registers, which are allocated during the synthesis process [260] as well. While there is a flexible number of these registers – their number is scaled by the HLS tool to implement the given functionality while fulfilling resource sharing and scheduling constraints – they are, in their purpose, comparable to registers within a CPU for which [259] defines the RVF. For the efficient introduction of fault tolerance measures in partial mitigation scenarios, this dissertation thus proposes to apply the concept of the RVF to HLS-generated designs. Accurately deriving the RVF for each generated storage element allows to efficiently guide the insertion of potentially costly fault mitigation measures. The remainder of this section illustrates this concept using the `diffeq` function that is used as an example to illustrate the HLS process in [260]. Furthermore, it discusses ways of deriving the *susceptible lifetime* per storage element instantiated by the design.

The `diffeq` function described in [260] solves the differential equation $y'' + 3xy' + 3y = 0$. It consists of two basic blocks with basic block #1 forming a loop condition, and basic block #2 containing the calculations within the loop body. Figure 58a shows the DFG for basic block #2 of the `diffeq` function mentioned above. Figure 58b illustrates a possible schedule and register allocation for this basic block, with *CS0* – *CS3* and *out* denoting control steps (i.e., clock cycles). It can be seen that five registers (*r1* – *r5*) have been allocated to store intermediate values necessary in the computation of one loop iteration (i.e, basic block #2), of which *r1*, *r3* and *r5* hold multiple values.

The temporal patterns with which these five registers are accessed within the given schedule and register allocation are shown in Figure 58b. In total, nine intermediate values need to be stored for the execution of basic block #2 of `diffeq`, which are labeled *val0* – *val8*. Under the definition derived from [259], these values experience different *vulnerable lifetimes*: *val0*, *val2* – *val4*, *val7*, and *val8* are each vulnerable for one control step, *val6* for two, *val5* for three, and *val1* for all control



(a) Data Flow Graph (DFG) (b) Schedule & Register Allocation

Figure 58: Basic Block #2 of `diffeq` (adapted from [260])

steps.

Assuming that each of these values is equally important for the basic block's execution to deliver correct results, the sum of the vulnerable lifetimes of these values per register constitutes the contribution of each register to the basic block's susceptibility to user memory bit flips. Thus, register *r2* contributes the most susceptibility as its values are vulnerable for four cycles. *r3 – r5* equally contribute three vulnerable control steps, while *r1* only contributes two vulnerable steps. Thus, a partial fault mitigation approach may prioritize *r2* over the other registers when selectively introducing redundancy (e.g., register-level ECC or replication). However, this only considers one execution of the considered basic block.

The total number of executions of this basic block may depend on the results calculated by this and other basic blocks from the input values to the top-level function. This influences the overall *susceptible lifetime* of the basic block's register over the total execution time of the implemented functionality: If this basic block is executed only once, but another block with a register that contains a register with a *susceptible lifetime* of one control step per iteration is executed ten times, the register within this second basic block accumulates more total *susceptible* time than *r2* in basic block #2.

Thus, while the *susceptible lifetimes* within a single basic block could be easily derived statically from the schedule and the register allocation table, the interdependence of the basic blocks, as well as cross-basic-block optimization carried out by the HLS tool may complicate the calculation of a global *susceptible lifetime* per instantiated register. For this purpose, the register access patterns of actual design executions may provide a better basis.

Register access patterns can be derived by profiling the design at multiple levels:

1. Under the assumption that there exists a correspondence between access patterns regarding variables in the original HLL source code, the RVF regarding RTL storage elements instantiated during HLS of the same source code, and, finally, the criticality of netlist-level storage cells (flip-flops and memory blocks), one way to identify vulnerable storage elements

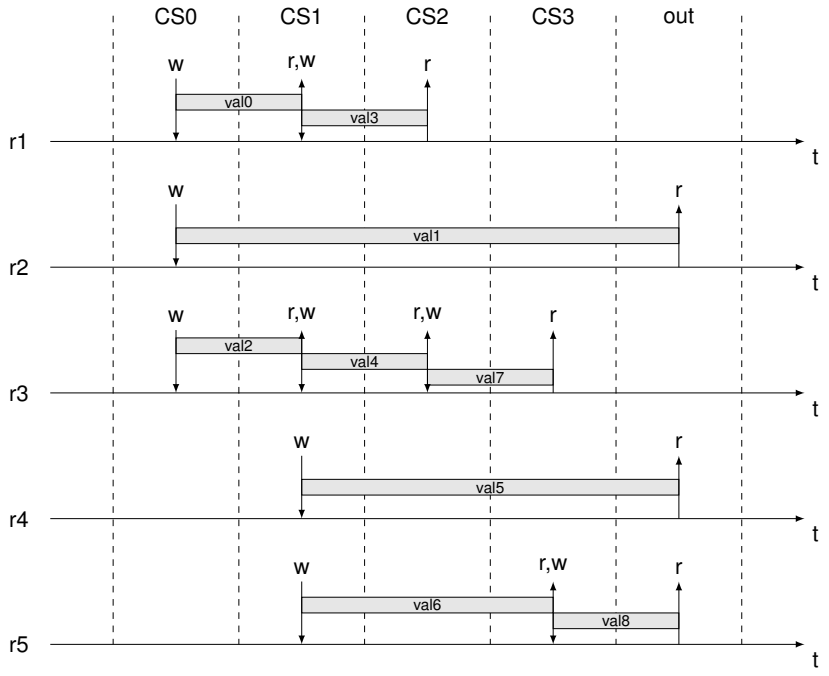


Figure 59: Register Access Patterns for Schedule & Allocation for Basic Block #2 of `diffeq` from Figure 58b for the Calculation of *Vulnerable Lifetime*

is to profile an executable representation of the original HLL source. This includes tracking accesses to the HLL variables during a profiling run, and subsequently informing the HLS tool to protect all instantiated storage element FUs derived from variables deemed to be especially critical as indicated by the profiling results. This additionally allows the HLS tool to pack especially critical, but temporally compatible, variable values into the same resilient storage element, and other optimizations of this kind that improve the cost-efficient implementation of fault mitigation measures. As execution of the original HLL source runs at software execution speeds, the runtime of this level of analysis may be attractive as well. However, analyzing variable access patterns within the original HLL source is unable to consider intermediate storage elements instantiated due to the schedule and binding found by the HLS process. Thus, it is also unable to consider the precise final temporal behavior of the implemented HLS schedule.

2. Moving one step closer to the hardware, an IR of the design after passing the HLS tool's frontend, can be profiled. Instead of the algorithmic expression of functionality found in the design's HLL representation, this representation contains the individual operations necessary to implement this functionality, e.g. in the form of basic blocks. In a Single Static Assignment (SSA) form, where all intermediate variables are assigned only once and thereby highlight data dependencies, this intermediate representation contains individual variables for all storage elements instantiated between operations on the basic blocks, which are assumed to be a proxy for registers and memory cores instantiated during the HLS process. While it may not capture all aspects of the design's behavior on the hardware level, simulations at this level may be carried out as part of the HLS process, allowing the HLS tool to consider the derived vulnerable lifetimes, for example, in a design-space exploration phase before committing the design to RTL.

3. The RTL generated by the HLS process may be simulated in a logic simulator. Register accesses must be derived by combining simulator traces with a parsed representation of the RTL design, which allows to pin-point at which time registers are written and read. While write accesses can be determined in a straightforward manner – being those clock cycles where all conditions that allow a certain register-mapped signal to be written, read accesses are those cycles where the value carried by a register-mapped signal determines the value that is latched into one or more subsequent registers. Thus, a representation of the RTL design's data flow must be consulted. Concepts such as the Waveform Analysis Language (WAL) proposed in [261] may be useful in automating this process. As this analysis requires the design to be present in RTL, a HLS run cannot directly profit from its results. Furthermore, execution of an RTL simulation may be slower than the simulated execution of the HLS design's IR.
4. Finally, a similar analysis may be carried out on the netlist level after performing logic synthesis. As the necessary gate-level simulation is even slower than RTL simulation, this approach offers the worst analysis runtime of the three levels, while delivering the most accurate results, as optimizations introduced by the synthesis tool (and, possibly, physical synthesis) can be incorporated into the analysis. On this level, accesses to storage elements must be differentiated into accesses to flip-flops, latches, block memory, and other technology-dependent memory elements. The conditions for write and read accesses for these elements must be modeled, and the corresponding access patterns determined from a gate-level simulation trace. Similar to RTL simulation analysis, approaches such as [261] may be used in parsing the large amount of data produced by this process.

Both RTL and gate-level storage-element access profiling are applicable to both manually implemented RTL designs and RTL designs generated by HLS. However, only the HLS-based analysis is able to contribute its results back to the synthesis process to generate inherently improved designs. This, along with the general motivations for employing HLS lined out in the beginning of this chapter, motivate the decision to perform this analysis based on a high-level intermediate representation of the design.

To that end, the following sections of this dissertation investigate the suitability of the two high-level analysis concepts (HLL analysis and IR analysis) to predict a design's susceptibility to bit flips in user-defined storage elements and to suggest subsets for cost-efficient partial protection against these bit flips.

6.2 Implementation

To investigate their suitability for efficient partial SEU mitigation, the two high-level SEU susceptibility concepts discussed above were implemented based on the open-source HLS tool PandA Bambu [227]. PandA Bambu allows the generation of synthesizable RTL designs from C/C++ source code. To parse the HLL source code, Bambu makes use of the GNU Compiler Collection (GCC)[130]. The frontend compiles the C/C++ code into the GIMPLE IR, which is used as a basis for the subsequent synthesis steps. The SSA form of GIMPLE is employed by Bambu [262] – meaning that each variable is only assigned once during execution of a considered function – making all potential steps where storage elements are inserted during HLS explicit. Newer versions of Bambu also support the LLVM IR, and thus also may employ *clang*[131] instead of GCC.

[130] See `https://web.archive.org/web/20250603083611/https://gcc.gnu.org/`, accessed 2026-01-05

[131] See `https://web.archive.org/web/20250531125315/https://clang.llvm.org/`, accessed 2026-01-05

Bambu was chosen for this evaluation because it provides the unique possibility to export its IR as low-level C code. As this export is based on SSA GIMPLE when using GCC as a frontend, the C export explicitly declares variables for all storage elements that are introduced by the GIMPLE frontend, and contains all the operations also found in the HDL export. Bambu uses this feature to co-simulate the IR with the generated HDL design to verify that the generated logic produces the same output values.

In the following, Section 6.2.1 describes the implementation of the HLL-based profiling concept described in Section 6.1, which is performed as a precursor step to the HLS process. Section 6.2.2 describes the implementation of the concept of profiling an IR during the HLS process. Furthermore, a concept for directing the HLS tool either from the HLL or via separate constraint files to implement selected storage elements in an ECC-protected variant is explored in Section 6.2.3.

6.2.1 High-Level Representation Profiling

The fault hypothesis used in this profiling concept is as follows: Variables within the HLL source hold intermediate values for the calculation of the described algorithm. During HLS, these variables (and derived intermediate variables) are mapped to storage elements such as a block RAM or register in the generated HDL. Variables that are accessed very often are assumed to result in storage elements with a higher total *vulnerable lifetime*, and generally to be more critical to the design's functionality. This is because the chance of an erroneous value being used is higher.

Thus, these variables (and their resulting storage elements) should be preferred during the generation of a design that is partially protected against bit flips in its storage elements. This can be done automatically, e.g., by applying a certain budget of soft-error-protected storage elements. However, a designer may also want to manually specify, based on the profiling results of the HLL source, which variables to treat preferably during redundancy insertion using constraints directing the HLS tool.

Figure 60 shows the toolflow that was implemented in the course of this dissertation for the evaluation of the proposed HLL-based profiling concept. The two key parts contributed in the context of this section are (1) a *HLL reliability profiling tool* that deduces the access frequency of variables in the HLL source code, and (2) a way of introducing user-defined protection constraints into the HLL source code that are transparently implemented by a *reliability-enhanced* version of the PandA Bambu HLS tool as storage elements (register or memory cores) protected by ECC.

The developed *HLL reliability profiling tool* shown in Figure 60, named *Vargrind*, performs the following steps when profiling the HLL source:

1. The source code is compiled for a host machine, e.g., an x86 machine running Linux. This is done as the employed version 0.9.4 of PandA Bambu[132] uses GCC's C-compiler targeting x86 (i.e., 32-bit) in its implementation. The source code is compiled without optimization (i.e., `-O0`); the compiler is instructed to add debug information to the generated binary (i.e., `-g`). For this purpose, a testbench for the HLL function to be implemented must exist, which may, for example, be derived from unit tests for this function.

2. Next, memory traces of executions of the compiled HLL are generated. This part of *Vargrind*'s functionality was implemented as a plug-in for the memory debugging framework Valgrind [263]. Valgrind re-compiles and instruments the binary produced by a compiler into an intermediate machine language and executes it on a virtual machine. This makes it possible

[132] Available via `https://web.archive.org/web/20250518085428/https://panda.deib.polimi.it/?page_id=81`, accessed 2026-01-05

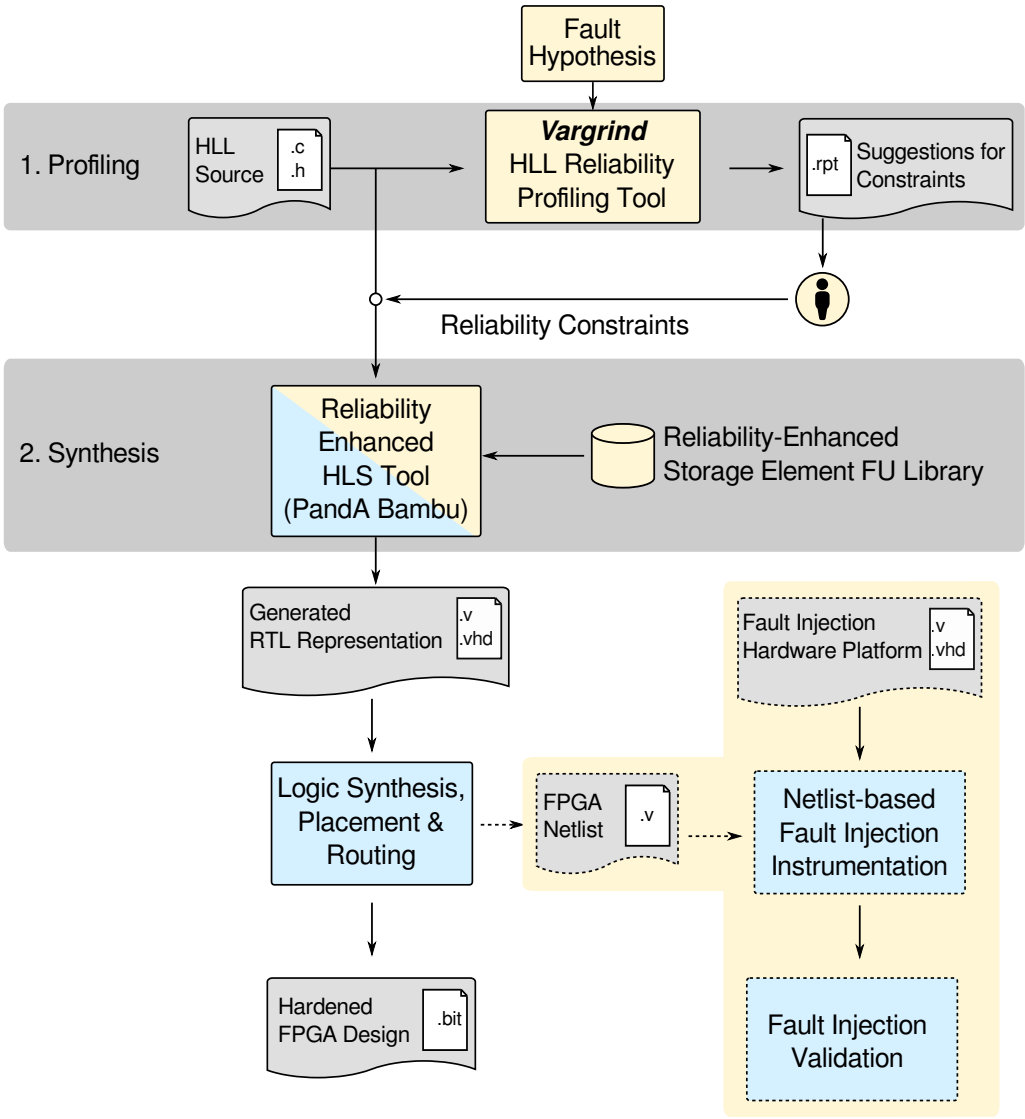


Figure 60: Implemented HLL-based Reliability Enhancement Flow

to track all memory accesses of the compiled program. Translating between the accessed addresses and variable names is possible by adding debugging symbols during compilation. The resulting memory trace contains the target address for every memory access, the variable name and location in the source code, the scope of the variable, and whether it was a load, store, or modify-type access.

3. Once the memory traces have been generated, *Vargrind* analyzes them to predict the individual variables' SEU susceptibility. The fault hypothesis for HLL analysis is based on the assumption that the number of accesses to each unique variable correlates with the corresponding storage elements' RVF. The output of the memory trace analysis, thus, is a list of all traced variables within the HLL source code sorted by their access frequency.

Following this profiling step, the user then chooses a cut-off criticality which of these variables to protect using ECC using annotations in the HLL or resource binding constraints (see Section 6.2.3).

6.2.2 Intermediate Representation Profiling

While profiling the original HLL source – as described in Section 6.2.1 – is both easy to implement and promises fast profiling runs, it cannot capture the intermediate storage elements introduced during the frontend of the HLS process. Although intermediate variables are also instantiated in the compilation process for this initial profiling run, this is done implicitly. Thus, accesses to these variables cannot be attributed by *Vargrind* to storage elements in the representation generated during HLS.

To counter these effects, as discussed earlier, an IR of the design, later used by the HLS tool as input of its subsequent synthesis steps, could be profiled. This concept was implemented in the course of this dissertation as follows:

1. Parse the HLL source code using the frontend of the HLS tool, including device-independent optimizations of the original code.
2. After building an IR of the design from the optimized source code, export the IR as an executable program at a lower level of abstraction. This representation fulfills the same functionality as the original program but contains and makes explicit all the operation and storage elements also present in the final design.
3. Execute the internal representation to generate a trace of all accesses to storage elements. Analyze this memory trace to deduce the criticality of each storage element. The quality of the output data generated by this step strongly depends on the coverage of the algorithm to be synthesized by the supplied input data for this test. Ideally, a wide range of expected input data is used to perform this execution of the internal representation. This work proposes multiple methods of differing complexity for analyzing these memory traces. These methods are described in the remainder of this section.
4. Sort the storage elements according to their criticality, and generate binding constraints that bind a given storage element to (1) an expensive, but reliable implementation (e.g., an ECC-protected memory block or a triplicated register core) or (2) an unprotected implementation according to their criticality score. This may be done until a budget of protected variables is exhausted. The final register binding step is then performed using these constraints.

To derive each register's vulnerability to single-bit flips according to the fault hypotheses formulated above, the IR export to C is re-purposed for profiling the temporal behavior of the design regarding accesses to storage elements.

While the goal of the proposed implementation is to estimate the *vulnerable lifetime* of all register and memory blocks within the FPGA netlist generated from the input HLL representation of the design, possible discrepancies may arise between the *lifetimes* derived from the analyzed IR and those in the final netlist due to optimizations during logic synthesis. Thus, using the proposed implementation, the following different methods to estimate this lifetime – and, thus, individual register criticality – from the memory trace of the IR are proposed and evaluated:

1. Metric *"Load"*: The more often a stored value is read during one execution period of the IR, the higher the probability that a flipped bit value may propagate through the design. Thus, the metric *Load* considers the total number of read accesses to every storage element for ranking.
2. Metric *"LPS"*: A flipped bit value is corrected as soon as the storage element is written the next time. Thus, if a variable is written more often throughout one execution of the IR, there

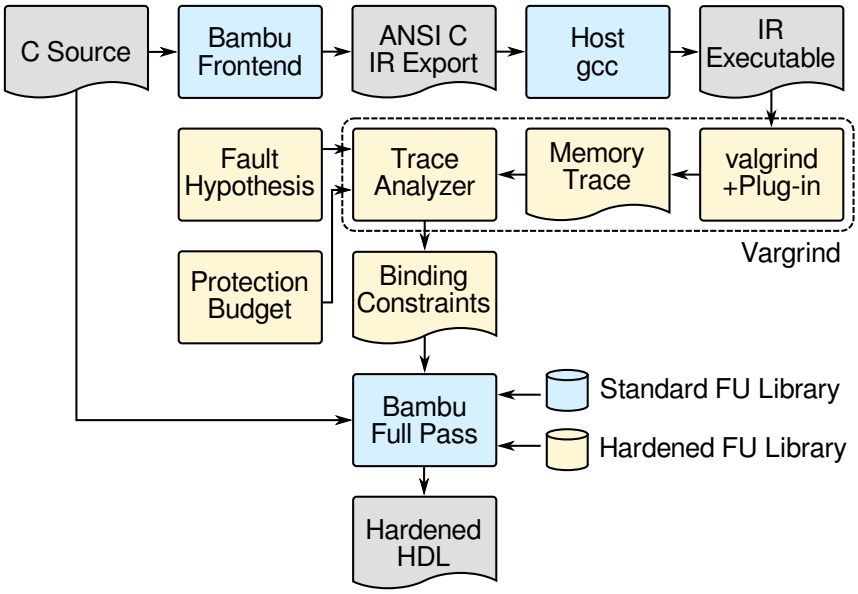


Figure 61: Implemented IR-based Reliability Enhancement Flow

is a higher chance of correction. Thus, for metric *LPS* ("loads-per-store") the average number of read accesses for every write access is used as a vulnerability factor.

3. Metric *"Temp"*: This metric considers the temporal behavior of the IR: Here, the time elapsed between every read access and the last write access to the respective storage element is summed up. This assigns high criticality to storage elements that are written once and left vulnerable until they are used much later in the execution. This metric requires a close match between the temporal behavior of the executed IR and the netlist-level implementation of the design, while metrics *Load*, and *LPS* merely require that the number of the respective storage element accesses matches.

4. Metric *"Temp2"*: Finally, as a variant of the *"Temp"* metric, the *"Temp2"* sums up the time between the first write access to a value and the last read access using that value. Thus, in its semantics, it is the closest formulation to the *susceptible lifetime* discussed in the context of the RVF in Section 6.1. It does, however, not include the number of read accesses of a value: a value written once and read once n clock cycles later produces the same *Temp2* result as a value written once and read several times, given that the last read access also happens n clock cycles after the write access.

To illustrate these metrics, consider again the register access patterns shown in Figure 59. Applying the metrics described above, the vulnerability scores shown in Table 84 can be derived. "Modify" accesses – i.e., a read and write carried out within the same control step – are considered to be two accesses for the *Acc* metric, while being counted as load accesses in the *Load* metric, and a read access immediately followed by a write access in the *LPS* and *Temp* metrics. It can be seen in Table 84 that the metrics described above assign quite different criticality scores to the individual storage registers. Which of these metrics finally corresponds with the best outcome in selectively protecting storage elements within a netlist synthesized from an HLS-generated RTL design is the goal of the evaluation(s) described in Section 6.3. It is to be noted that, in this example, *Temp* and *Temp2* produce the same results as every value is only read once. If, however, *val1* (stored in *r2*) would be read an additional time in *CS2*, *Temp2* would still evaluate to 4, while *Temp* would evaluate to 6.

Table 84: Demonstration of Proposed Criticality Metrics w.r.t. Registers & Schedule from Figure 59

Register	Criticality Metric			
	Load	*LPS*	*Temp*	*Temp2*
r1	2	1	2	2
r2	1	1	4	4
r3	3	1	3	3
r4	1	1	3	3
r5	2	1	3	3

6.2.3 Reliability-Enhanced High-Level Synthesis with PandA Bambu

As PandA Bambu's frontend uses GCC's C compiler for parsing the original HLL code, GCC's attribute syntax has been selected for implementing these constraints. GCC's plugin system allows the user to define a plugin that is executed when certain attributes are found while parsing the C code. The callback function defined by the plugin is supplied with information about the objects for which attributes are specified, making it well-suited for the application of constraining variables and function parameters. A plugin has been implemented that parses `__attribute__((ECC))` directives. In line with PandA Bambu's way of implementing other constraint files (e.g., resource binding constraints), the plugin generates an eXtensible Markup Language (XML) constraint file for each translation unit that is later used to constrain the instantiation and parametrization of registers and memories. For every variable the user chooses to protect using the implemented ECC approach, an `__attribute__((ECC))` directive may be placed at the variable's declaration, as shown in Listing 2. However, it is also possible to provide the XML file directly if desired, forgoing the need to modify the original HLL source. Globally protecting all variables using ECC for testing purposes can be specified using a command-line switch passed to PandA Bambu.

```
int __attribute__((ECC)) g;            /* global variable */
int func (int __attribute__((ECC)) a, /* function parameter
                                        * (local variable) */
          int b) {                     /* unprotected variable */
char __attribute__((ECC)) arr[40];     /* array (local variable) */
...
}
```

Listing 2: ECC Attribute Examples

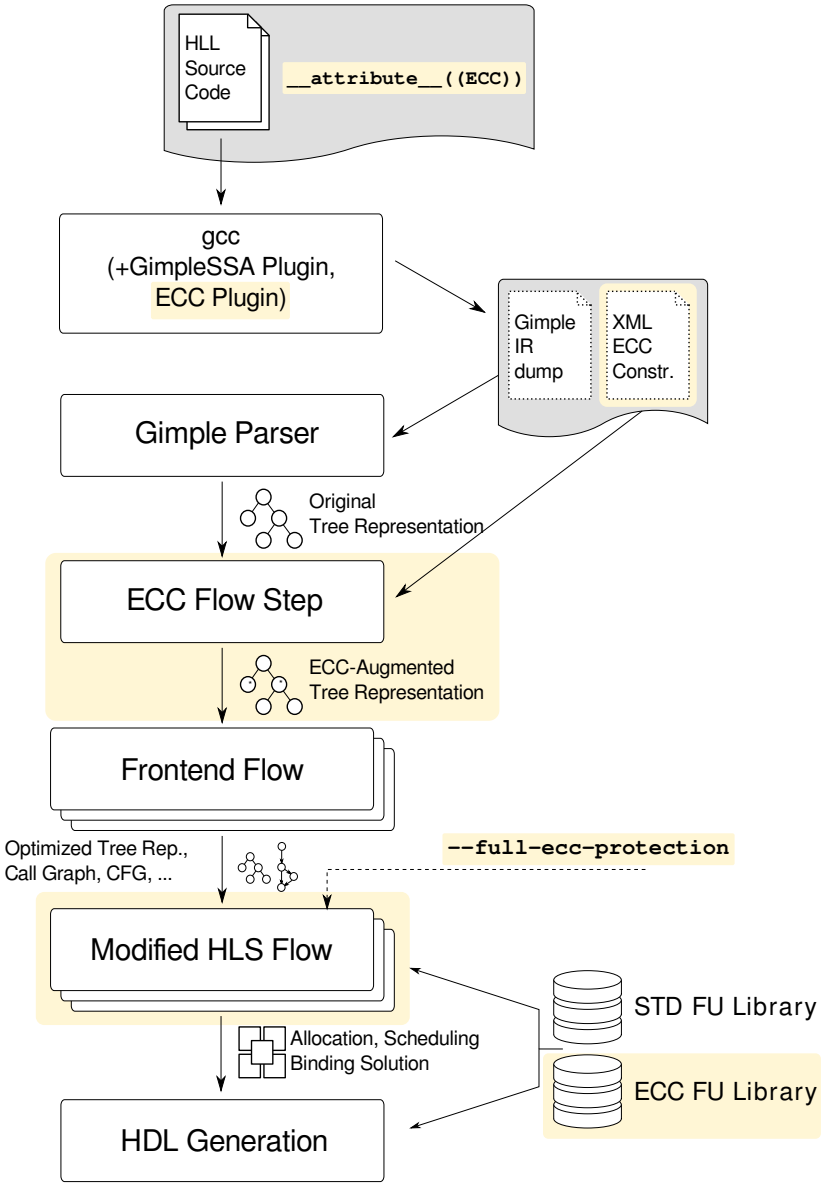


Figure 62: Modified PandA Bambu Flow

To implement the functionality discussed above, PandA Bambu was modified to include the following features as illustrated in Figure 62, where the changes made are highlighted.

- The GCC plugin for parsing the attribute directives was added to the call to GCC from PandA Bambu.
- The XML files generated by the GCC plugin are parsed after compilation.
- While building an internal tree structure from the GIMPLE IR, Bambu uses the information from the XML file mentioned above. It attempts to match variable and parameter declaration nodes to entries in the XML file using the declaration information supplied by GIMPLE. In particular, the path to the source file, the variable or parameter name, and the declaration line number are used. An additional flag is stored for each variable declaration node, stating if the variable is to be protected using ECC.

- When mapping desired functionality to modules from the tool's FU library, this information is used to select ECC-protected variants of the respective modules.
- The generation of memory initialization files has been modified: When generating such files for ECC-protected memories, the initialization values are ECC-encoded.

In addition to the changes made to the tool itself, ECC-protected variants of register and memory cores have been implemented and added to the tool's FU library. As an encoding scheme Hamming Codes have been selected, providing the possibility to completely mask any single bit error occurring during operation [96].

While changes to registers as well as memories that are only accessed using a single data width are trivial (i.e., adding ECC encoders at the input ports and ECC decoders at the output ports), more thought has to be given to byte-enabled memories: Byte-enabled memories are updated byte-wise during operation. If ECC parity is calculated over the entire word as stored in the memory, each write that does not update the entire word would require to decode the stored word, update some bytes of this word, re-encode, and finally store it. This would change the temporal behavior of the memory core, making a "drop-in" replacement impossible. In byte-enabled memories, each byte is protected by its own ECC parity, thereby enabling a single-cycle update of the stored word. As an additional benefit, one bit flip per byte instead of one bit flip per word may be tolerated. This comes, however, at increased area costs: A 64-bit data word that is accessed without byte enables may be protected using 7 parity bits, while a 64-bit wide byte-enabled memory needs 32 parity bits[133].

6.3 Evaluation

The approaches regarding vulnerability profiling of an HLL design or its IR described in Section 6.1, their implementations discussed in Section 6.2, and the subsequent partial protection of the resulting design promise cost-effective soft-error mitigation in the resulting FPGA design's user storage elements. Thereby, it is assumed that the predictions made from profiling the HLL representation of the design or its IR hold in the FPGA. The purpose of this section is to evaluate the quality with which they hold, and their suitability for providing cost-effective protection of HLS-generated designs against bit flips in user-defined storage elements.

To that end, the implemented profiling approaches discussed in Sections 6.2.1 and 6.2.2 are applied to sets of case-study designs sourced from several HLL benchmark suites [229, 264] and supplemented by HLL designs implemented specifically for this evaluation. The unchanged HLL sources are compiled into RTL designs by the PandA Bambu HLS tool and subsequently implemented for an AMD Zynq 7000 FPGA evaluation platform. These designs are subjected to fault injection experiments, in the course of which faults are injected into the design's register cores and memory blocks at runtime. The effects of these faults on the original design and design variants protecting different subsets of storage elements according to the HLL-based profiling approach are compared. Evaluation details, procedure, and employed case studies are described for HLL-profiling in Section 6.3.1, and for IR-profiling in Section 6.3.2. Furthermore, in the course of the evaluation of the HLL-based profiling approach, the ECC insertion implementation described in Section 6.2.3 was evaluated.

While the fault injection approaches applied in Chapters 4 and 5 target the FPGA's configuration memory, the experiments described in this chapter target flip-flops and BRAM cells. Configuration

[133] 4 parity bits are needed to protect an 8-bit byte; 8 bytes in a 64-bit word

memory faults in Chapters 4 and 5 are modeled as permanent faults, which allows either using the target device's partial reconfiguration capabilities to flip the targeted configuration bit at run-time or pre-generating faulty bitstreams offline and configuring the device with them for each experiment. As in conventional FPGA designs without reconfiguration or scrubbing capabilities no mechanism exists for configuration memory faults to "resolve themselves" during normal operation, this model is well-suited to configuration memory bit flips. Conversely, user-defined storage elements are continuously overwritten by the design during normal operation – which is precisely the reason for the existence of a *vulnerable lifetime* as discussed in Section 6.1. Thus, modeling these faults as permanent stuck-at faults, as is done for configuration memory, does not apply to the evaluations carried out in this chapter.

The goal of the fault injections carried out in this chapter is to replicate randomly occurring bit flips at run-time. Two ways to produce this behavior in the running FPGA design are (1) netlist instrumentation and (2) technology-dependent fault injection:

1. Netlist instrumentation introduces fault injection modules, also referred to as *"saboteurs"* [265], into the design's netlist. These modules extend the fault-free design with the capability to exhibit faulty behavior on demand. Bit flips within storage elements can be implemented in several ways: a simple way may be to insert an inverter and a multiplexer at the output of the targeted storage element, allowing the injection of transient pulses into the output. While this can be done for both memory and flip-flops/latches, it only partly replicates the behavior of SEUs in these storage elements: It essentially replicates the behavior of SEUs in storage elements that are overwritten with new data at every clock cycle. To correctly emulate SEUs for storage elements with a *clock enable* signal and single-bit SEUs in memory, additional care has to be taken in saboteur development and instrumentation to capture the semantics of these element's control signals.

2. On the other hand, the FPGA's technology may be exploited to emulate the effects of bit flips in certain storage elements: [266] injects bit flips into flip-flops and latches in AMD Virtex 4 FPGAs by (1) capturing the stored values into configuration memory, (2) copying these values into the set/reset value configuration cells and inverting the value for the targeted storage element in the process, and (3) pulsing the set/reset inverter configuration bit of the targeted storage element to load the changed value. [267] implement a similar approach for flip-flops and latches in AMD 7 Series FPGAs. As these devices lack the per-slice set/reset inverter present in Virtex 4 devices, the FPGA's global set/reset signal – that is maskable for partial reconfiguration purposes – is pulsed. However, the mechanisms described above do not apply to distributed memory and BRAM, for which another method must be implemented.

While netlist instrumentation significantly increases the DUT's resource consumption, it is convenient to implement, applicable to distributed memory and BRAM, and does not require reverse engineering as is necessary to derive and implement approaches such as the ones described in [266, 267]. Thus, the evaluations described in this chapter employ the netlist instrumentation and fault injection tool FIJI [268, 269] — co-written by the author of this dissertation – to implement fault injection via netlist implementation.

6.3.1 High-Level Representation Profiling

The HLL-based profiling approach was evaluated on three case-study designs, which are all implemented in ISO C: As a lightweight case-study design for initial tests and demonstrations, (1) a *bubble sort*[270, pg. 106] implementation over a 256-element array of 32-bit integers has

been selected. Furthermore, the significantly larger (2) MIPS and (3) Motion benchmarks from the CHStone benchmark suite [229] were selected as case studies. All benchmark programs are self-contained and self-checking, i.e., they contain all the input data to the implemented algorithms as well as expected output data, and report via their return value if the algorithm produced the expected results.

Multiple different variations of each of the use cases were generated for comparison:

- As a baseline for comparison, a completely unprotected version was created.
- Using the implemented HLL profiling approach (see Section 6.2.1), usage statistics about the variables present in the evaluation programs were derived. For this proof of concept evaluation, one execution using the input data provided by the benchmark programs was considered during evaluation. Based on this information, two additional hardware designs were obtained using the implemented profiling approach: One only protecting the Least-used (LU) variables, and another one protecting only the Most-used (MU) variables. Attention has been paid that the sets of protected variables in both designs do not overlap. Only variables that are effectively mapped to registers or memories by the HLS process have been annotated for ECC protection.
- Variations of all hardware designs have been created where all variables present in the source code have been annotated with ECC attributes.
- Finally, for the purpose of comparing resource savings, a variation of all hardware designs, where all registers and memory cores have been unconditionally replaced with their newly added ECC-capable counterparts, has been created.

All of the variations were obtained by the author's modified version of PandA Bambu with compiler optimizations turned off (`-O0` option passed to GCC).

For each use case, the number of variables mapped to storage elements, as well as the number of non-protected and protected storage element FUs as obtained from Bambu's synthesis report can be observed in Table 85. It can be seen that the fully-protected version and the version where all variables have been constrained by attributes are not equivalent in terms of the number of instantiated ECC-protected storage elements. This is a result of the translation to GIMPLE IR applied by Bambu's GCC front-end, which introduces temporary variables that are not accessible to the designer via the original source code, and thus cannot be constrained for ECC protection with this proof of concept implementation.

To be able to evaluate the effects of bit flips in registers and memories with pin-point accuracy, netlist-level fault injection has been selected as an evaluation method. To that end, the RTL source generated by PandA Bambu was first synthesized into a hierarchical Verilog netlist and placed&routed for an AMD Zynq 7000 FPGAs using AMD Vivado 2016.4 with a 50 MHz clock constraint. The resulting resource usage and timing data (in the form of t_{cpd}, the critical path delay) can be seen in Table 86 both in absolute numbers, and in terms of the overhead compared to the original design variant.

The netlist was then instrumented by using FIJI [268, 269]. One bit from each register FU and each port of each memory FU instantiated by the netlist was selected for instrumentation to emulate the effects of single-bit errors in these storage elements. To maximize the effects of the bit flips while keeping fault injection effort reasonable, the most significant bit in the output of each register and memory core was instrumented. Each of the generated design variations for each design was instrumented in this way, which were subsequently instantiated in the fault injection platform shown in Figure 63.

Table 85: Profiling-Derived ECC variable mapping

Design	Constrainable Variables	Variation	Registers		Memories	
			Plain	ECC	Plain	ECC
Bubblesort	5	Original	22	0	3	0
		3 LU	21	1	1	2
		2 MU	21	1	2	1
		All Vars	20	2	0	3
		Full ECC	0	22	0	3
MIPS [229]	14	Original	63	0	9	0
		7 LU	60	3	2	7
		7 MU	57	10	7	2
		All Vars	52	15	0	9
		Full ECC	0	67	0	9
Motion [229]	24	Original	103	0	18	0
		10 LU	95	8	12	6
		10 MU	99	4	11	7
		All Vars	90	13	1	17
		Full ECC	0	103	0	18

Finally, fault injection experiments were performed in the following way for each instrumented net in each design and its variants as shown in Table 85[134]:

1. Configure the *Timer+Result Classifier* hardware module (see Figure 63) unit with the DUT's expected result and the latency value derived from simulation as a timeout.
2. Start back-to-back execution of the DUT controlled by the *DUT Control* hardware module.
3. Select an instrumented net for fault injection.
4. Perform N_{inj} = 256 injections on the selected net. As a fault model, the *SEU* fault supported by FIJI was selected. This fault model causes the respective net to be inverted for one clock cycle for each activation of the fault. A fault was configured to occur roughly every 5000 clock cycles with a random offset between 0 and 255 cycles. During the back-to-back executions of the DUT, the *Timer+Result Classifier* accumulates the number of executions either conforming to the simulation results (i.e., error-free executions) or violating them either in timing (i.e., the DUT signals completion too early, too late, or not at all) or in value (i.e., the returned data mismatches the data expected from simulation).
5. After the N_{inj} injections have been performed, the back-to-back execution of the DUT is stopped, and the results from the classification hardware are logged.
6. The FPGA is re-configured from the SD card to ensure a clean starting point for the next injection run.

As the consequence of a bit flip in a storage element may be masked due to the element not being used at the time of injection, N_{rep} = 128 injection runs as described above have been carried out

[134] Only the full-ECC variant of the *MIPS* design was tested under fault injection, however: Once the functionality of full ECC protection is confirmed, no further insight w.r.t. partial redundancy can be gained from repeating this experiment on other designs.

Table 86: Resource Usage & Timing of Case-Study Designs

Design	Variation	Absolute				Overhead (%)			
		LUTs	FFs	BRAM Tiles	t_{cpd} (ns)	LUTs	FFs	BRAM Tiles	t_{cpd}
Bubblesort	Original	299	347	3.0	9.3	0.0	0.0	0.0	0.0
	3 LU	634	423	4.0	13.6	112.0	21.9	33.3	46.2
	2 MU	530	384	4.0	12.4	77.3	10.7	33.3	33.3
	All Vars	869	460	5.0	13.5	190.6	32.6	66.7	45.2
	Full ECC	1946	682	5.0	15.5	550.8	96.5	66.7	66.7
MIPS	Original	2141	737	2.0	17.2	0.0	0.0	0.0	0.0
	7 LU	3226	870	2.5	17.4	50.7	18.0	25.0	1.2
	7 MU	3179	981	2.5	19.3	48.5	33.1	25.0	12.2
	All Vars	4466	1189	3.0	19.0	108.6	61.3	50.0	10.5
	Full ECC	6027	1541	3.0	19.0	181.5	109.1	50.0	10.5
Motion	Original	2675	1959	1.0	14.5	0.0	0.0	0.0	0.0
	10 LU	3260	2208	1.0	14.6	21.9	12.7	0.0	0.7
	10 MU	4192	2172	2.0	19.1	56.7	10.9	100.0	31.7
	All Vars	5905	2544	2.0	19.5	120.7	29.9	100.0	34.5
	Full ECC	10715	3592	2.0	19.7	300.6	83.4	100.0	35.9

for each instrumented net to ensure a reliable result, totaling in $N_{inj} \times N_{rep}$ = 32768 individual fault injection operations for each instrumented net. Each fault injection run was only categorized as error-free if all executions within that run were error-free; otherwise the run was categorized as erroneous. By comparing the percentage of error-free runs obtained from the different variants of each case-study design, the effectiveness of the different selections of variables (and their resulting storage elements after HLS and RTL synthesis) for ECC protection can be compared. In particular, comparing the effectiveness of the variants protecting the least-used and most-used variables (LU and MU) variants provides hints towards the efficacy of the HLL profiling approach with respect to cost-effective protection. This data is provided and interpreted in Section 6.4.

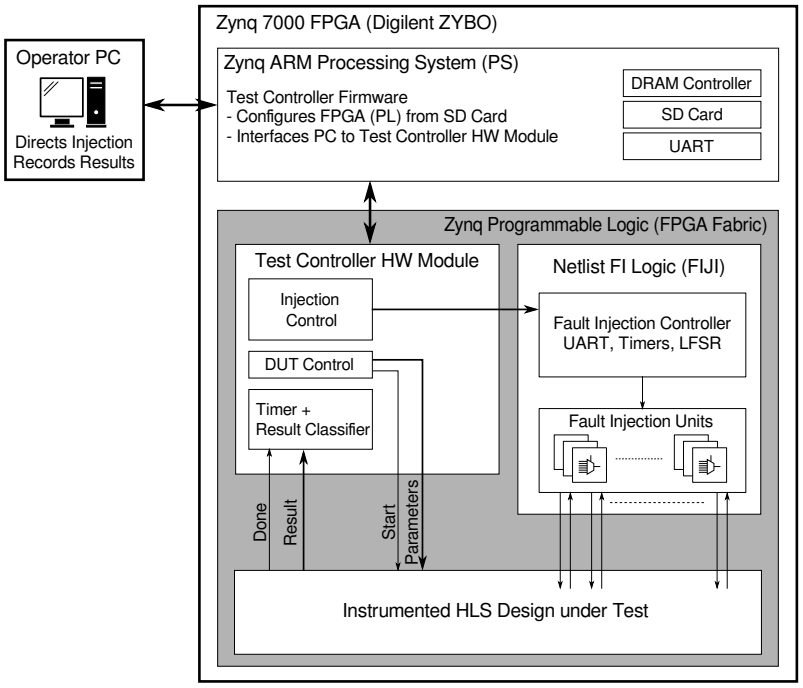


Figure 63: Evaluation Platform

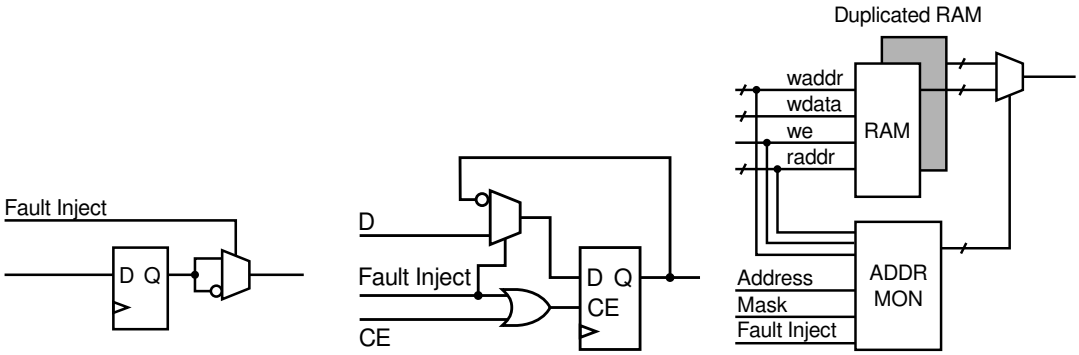
6.3.2 Intermediate Representation Profiling

The evaluation of the IR-based profiling approach described in Section 6.2.2 follows a similar pattern as the evaluation of the HLL-based approach described above. Its starting points are, in a similar way, the high-level (i.e., ISO C) descriptions of the designs to be protected. This description is synthesized into RTL using PandA Bambu [227], which yields both a C export of the design's intermediate representation after Bambu's front-end process and a Verilog representation after completing the high-level synthesis process. The former is compiled using the x86 host's *gcc* and subsequently executed, traced, and analyzed using the proposed *Vargrind* tool, resulting in *Loads*, *LPS*, and *Temp* metrics per variable as described in Section 6.2.2. The RTL design is synthesized for an AMD Zynq 7000 FPGA, generating a technology-dependent netlist that is subsequently instrumented for fault injection.

To analyze the predictive power of the analysis approach the per-storage-element criticality estimate derived from fault injection results is, on the one hand, compared to the respective IR-level variable's assessment according to the three metrics mentioned above. On the other hand, averaging over all storage elements, a design criticality value can be deduced as is done in Section 6.3.1. Assuming half of the design's storage elements to be protected perfectly against SEUs according to the respective metric (e.g., using ECC), and excluding these elements from the calculation of the above design criticality value, results in a projected design criticality value after protection according to each metric. This allows to compare the overall effectiveness of the criticality metrics in selecting the most critical storage elements for protection without requiring repeated fault injection experiments.

The applied instrumentation flow considers D-type flip-flops and memory blocks separately in three different fault modes:

- Bit flips in D-type flip-flops that have their Clock Enable (CE) signal tied to VCC manifest themselves in a flipped output value persisting for – at most – one clock cycle. This value is overwritten immediately in the next clock cycle. In order to emulate this fault mode, each of



(a) Flip-flops without Clock Enable (b) Flip-flops with Clock Enable (c) Memory Blocks

Figure 64: Netlist-level Instrumentation Circuits

these flip-flops is instrumented by inserting a multiplexer that selects between the original and inverted output value of the register. The inverted value is selected for one clock cycle by fault control logic for each injected fault. This is shown in Figure 64a.

- D-type flip-flops whose CE pin is driven by logic store the written value for more than one clock cycle. The stored value is overwritten in the next clock cycle in which the signal driving the clock enable pin is asserted. Here, the currently stored value can be inverted independently of the design's clock enable signal using the instrumentation logic shown in Figure 64b.
- Memory bits behave essentially in the same way as registers with a clock enable signal driven by logic. A flipped bit in a memory block retains its flipped value until the next time the respective word is addressed and the write enable signal is asserted. While in principle this fault injection behavior for BRAMs could be implemented in logic, all possible configurations of the block memory would need to be faithfully replicated in logic (e.g., registered address inputs/outputs, port configuration, read-during-write behavior). This additional complexity is avoided by duplicating each memory block, with multiplexers inserted between the data output signals of each port. The data input pins of the duplicate RAM as well as the initialization data are inverted. An address monitor is placed on the address buses for the RAM ports. This monitor is configured with the address of the word and the offset of the bit that is targeted by fault injection. When activated by external fault control logic, each time the address monitor detects the configured address it selects the data bit from the inverted duplicate RAM instance. This condition persists until the next write access to the configured address. Figure 64c provides a schematic of the described instrumentation.

Applying a random fault injection process to each storage element in a case-study design, the share of fault injection runs per storage element that resulted in at least one erroneous execution can be determined. This can, in turn, be used to calculate an average criticality of a particular storage element FU. The resulting per-storage-element criticality can then be contrasted with the score that this storage element received from the three IR-level profiling metrics *Load*, *LPS*, and *Temp* (see Section 6.2.2). The strength of the correlation of the respective metric and the criticality observed in fault injection indicates the relative quality of the metric in predicting in-hardware criticality of the storage element FUs. Furthermore, comparing the shares of faults avoided by an (assumed-to-be-perfect) ECC protection allows to judge the suitability of when assuming protection of the top half of the storage elements indicated by those metrics.

To that end, a set of case-study designs were subjected to (1) IR-level profiling as described in Section 6.2.2, (2) subsequent synthesis, instrumentation, and implementation for the test device,

Table 87: IR Profiling Case Study Designs

Design	Source	CRAM Bits	FF Bits	BRAM Bits
Blowfish	CHStone [229]	856442	2004	40536
SHA	CHStone [229]	861934	2835	152640
GSM	CHStone [229]	2031314	5922	10152
JPEG	CHStone [229]	518400	17163	518400
CoreMark	CoreMark	4553468	13381	21888
Black-Scholes	Parsec [264]	1773772	6318	64512
Swaptions	Parsec [264]	6343860	21791	147456

and, finally (3) fault injection experiments. The used case-study designs are listed in Table 87. Various applications, taken from the three CHStone [229], CoreMark[135], and Parsec [264] benchmark suites from the fields of cryptography (Blowfish, SHA), finance (Black-Scholes, Swaptions), as well as telecommunication and image processing have been used as test cases.

The benchmark applications in the CHStone [229] suite are intended for end-to-end validation of an HLS tool. They are self-contained, meaning that they contain all necessary input data for their core algorithms. Additionally, these benchmarks are designed to be self-checking: they include golden output data which is used to validate the generated output data during execution. On the top level, these benchmarks provide a pass/fail output that indicates if the data generated by the core algorithm in the current iteration was valid, as well as a *start* input and a *done* strobe output indicating the completion of the application's execution. The correct latency of these benchmarks can be determined, e.g., using a functional RTL simulation. The benchmark designs derived from the CoreMark and Parsec [264] benchmark suites were adapted to behave similarly.

The designs were implemented using AMD Vivado 2016.4. Basic data regarding the number of CRAM (i.e., configuration) bits as reported to be *essential* by Xilinx Vivado [88], as well as the number of bits stored in register banks and BRAM are provided in Table 87.

The designs are instantiated as part of a fault injection platform as shown in Figure 63. However, in this case, faults are injected by the controlling CPU not via FIJI's fault injection logic, but by triggering a pulse on the desired *Fault Injection* line of the respective instrumentation circuit (see Figure 64) and, in the case of BRAMs, configuring the address monitor (see Figure 64c) beforehand.

The process of identifying storage elements in the design, duplicating block memories, and routing the nets to be instrumented to the netlist's toplevel is automatically performed with the help of Verilog-Perl[136]. Parameterizable fault injection logic for flip-flops as well as block memories are implemented in a Verilog wrapper that instantiates the DUT and exposes its original input and output ports.

At run-time, the instrumentation logic added to the original netlists is controlled by fault injection control logic implemented as a peripheral unit of a hard CPU core. This control logic allows the configuration of a specific bit for fault injection and enables the respective fault injection control signal for a single clock cycle at a time. A timer allows to defer the configured fault injection operation until a certain, configurable number of clock cycles has elapsed after the DUT has been started.

[135] Available from `https://web.archive.org/web/20250601160016/https://www.eembc.org/coremark/`, accessed 2026-01-05

[136] Available from `https://web.archive.org/web/20230121165546/https://github.com/veripool/verilog-perl`, accessed 2026-01-05

As in the evaluation of the HLL profiling approach, the result classification hardware unit is configured with the parameters that describe the correct functionality of the DUT: The expected output data and the expected temporal behavior (i.e., latency). In parallel to the fault injection process, the results generated by the DUT, as well as the DUT's timing are monitored by this classification unit. Only executions of the DUT that return the respective *pass* result within the correct time frame are accepted as fault-free, every other case is seen as the effect of an injected error.

The fault injection sequence is controlled using the on-chip ARM CPU in combination with control software running on a host PC. For each of the conducted random fault injections, a random bit out of all sequential elements in the design is chosen along with a random point in time (as a random offset in clock cycles after starting the DUT). The number of injections may be selected depending on the number of sequential elements present in the DUT. The DUT is reset before each injection operation to ensure a clean working state for the next fault injection operation. This clean working state is then verified by observing the design's behavior for a few execution cycles of the DUT without fault injection. This is done to detect persistent errors that cannot be corrected by a system reset. In this case, the FPGA is reconfigured, thereby restoring, e.g., also BRAM initialization data.

6.4 Results and Discussion

This section analyzes and discusses the results of the evaluation experiments described in Section 6.3 with the purpose of determining if the proposed and implemented profiling approach(es) deliver desired results in terms of cost-effective protection against single bit flips in an HLS generated design's storage elements.

Table 88 presents the results of the fault-injection-based evaluation of the HLL profiling approach using the case-study designs *Bubblesort*, *MIPS*, and *Motion* as described in Section 6.3.1. It shows the absolute share of fault-free fault injection runs per design variant as the variant's basic reliability assessment. Furthermore, the relative improvement compared to the original variant is shown in the variants augmented with ECC-capable storage element FUs – i.e., the *benefit* component of partial soft-error mitigation. The related *cost* component has been shown in the relative increase in resource usage of the different variants in Table 86. From these components, the Benefit-Cost-Ratio (BCR) can be deduced as a measure of the relative effectiveness of the protection strategies with respect to LUTs, FFs, and BRAMs. It must be noted that a fully ECC-protected variant was only evaluated for the *MIPS* design as stated in Section 6.3.1.

From the data shown in Table 88, it can be seen that protecting both the least-used and most-used, as well as protecting all variables using ECC by inserting the appropriate `__attribute__` statements into the original HLL source, leads to an improvement in the resilience of the resulting designs against the injected single bit flips in their storage elements. However, protecting the most-used variables, in all three cases, corresponds with a larger improvement in fault-free fault injection runs than protecting the least-used ones. This may serve as an indication that considering HLL profiling results are useful when deciding which subset of HLL variables to protect.

However, this does not consider the resource and timing overhead (i.e., cost) incurred when implementing the respective mitigation measures. To take this into consideration, Table 88 also shows a Benefit-Cost-Ratio (BCR), contrasting the benefit (relative increase in fault-free fault injection runs) with the cost (the relative overhead in terms of LUTs, FFs, and BRAMs, and the relative increase in critical path delay as shown in Table 85). In terms of the BCR, the difference between least-used and most-used variables is not as clearly distinguished as when solely comparing fault-free fault injection runs: While protecting the most-used variables delivers

Table 88: Fault-Free Fault Injection Runs & Benefit-Cost-Ratio (BCR) by HLL-profiling-based ECC protection strategy

Design	Variation	Fault-Free FI Runs		BCR			
		Absolute	Relative	LUTs	FFs	BRAM	t_{cpd}
Bubblesort	Original	38.4%	N/A	N/A			
	3 LU	40.9%	6.5%	0.06	0.30	0.20	0.14
	2 MU	47.8%	**24.5%**	**0.32**	**2.3**	**0.73**	**0.73**
	All Vars	50.1%	30.5%	0.16	0.94	0.46	0.67
MIPS	Original	50.1%	0.0%	N/A			
	7 LU	58.1%	16.0%	0.32	**0.88**	0.64	**13.73**
	7 MU	63.1%	**25.9%**	**0.54**	0.78	**1.04**	2.13
	All Vars	71.0%	41.7%	0.38	0.68	0.83	3.99
	Full ECC	100.0%	99.6%	0.55	0.91	1.99	9.52
Motion	Original	85.3%	0.0%	N/A			
	10 LU	86.4%	1.3%	**0.06**	0.10	N/A	**1.87**
	10 MU	88.3%	**3.5%**	**0.06**	**0.32**	**0.04**	0.11
	All Vars	90.2%	5.7%	0.05	0.19	0.06	0.17

the best BCR in all four kinds of analyzed overheads, in the *MIPS* case protecting the most-used variables only provides a better BCR in terms of LUTs and BRAMs, but not FFs and timing. In the *Motion* design, on the other hand, protecting the most-used variables provides a better BCR in terms of FFs, but a worse one when considering LUTs and timing. As the variant of *Motion* protecting the least-used variables does not lead to an overhead in BRAMs, no comparison can be made here. Thus, the better protection offered by the variants protecting the most-used variables may, on the one hand, be due to a correlation between variable accesses in HLL profiling and storage element susceptibility in hardware. On the other hand, however – especially in the *MIPS* and *Motion* case – these increases may also be influenced by the size (i.e., bit width) of the protected variables. It can further be seen that protecting all storage elements using ECC can be prohibitively expensive when FPGA resources must be expended for implementing ECC encoders and decoders: In the “Full ECC” case, the insertion of ECC logic leads to an overhead that is significantly larger than the combinational resources making up the original design.

Finally, the *MIPS* case shows a significant gap in reliability between a variant that constrains all variables in the HLL to be protected using ECC (“All Vars”), and the variant that uses lower-level constraints to replace all storage-element FUs implementations with ECC-capable ones (“Full ECC”). As discussed in Section 6.3.1, this is caused by additional storage elements introduced in the front-end phase, in which the HLS tool PandA Bambu transforms the HLL source code into a SSA GIMPLE representation, which later result in storage elements holding intermediate values for accomplishing the schedule developed during HLS. Thus, the results can be seen as a first indication that profiling variable (or register) accesses may lead to beneficial selective mitigation decisions. However, the HLL representation of the design (and its transformation into a CPU-executable program, that is executed on a virtual machine) is too far removed from the hardware implementation resulting from HLS and RTL synthesis to capture these effects.

In contrast to the evaluation of the HLL-based profiling approach, where only Most Significant Bits (MSBs) of storage element FUs were instrumented, in the evaluation of the IR-based profiling approach, all storage elements were instrumented and evaluated according to the procedure described in Section 6.3.2, allowing for more accurate criticality evaluation. Applying this procedure

Table 89: Criticality of Original Designs derived from Fault Injection Experiments

Design	Criticality (%)			FIT	
	FF	BRAM	Total	M2S050	Zynq 7000
Blowfish	10.37	64.10	61.57	8.574	1.729
SHA	19.10	86.71	85.48	43.548	8.771
GSM	4.17	40.06	26.84	1.393	0.285
JPEG	3.69	8.06	0.04	13.853	2.799
CoreMark	4.20	23.29	16.05	1.806	0.374
Black-Scholes	5.54	32.87	30.43	7.039	1.423
Swaptions	23.01	0.94	8.95	1.638	0.422

to the case-study designs shown in Table 87, the fault-sensitive flip-flop and BRAM bits in these designs were determined experimentally. The ratio of these sensitive bits to the total bits can be seen as the average fault propagation probabilities for flip-flops and BRAM bits. These values are shown in the columns labeled “criticality” in Table 89, a combined storage-element criticality can be calculated as the average of these numbers weighted by the number of bits stored in each category of storage elements (see Table 87). This results in the “Total” subcolumn of the reported criticality data.

Furthermore, employing Equation (2.1), the contribution of FFs and BRAMs to the design’s failure rate can be determined for a particular device. In Table 89, this is done for two devices:

- As a representative for Flash-based devices, Microsemi’s SmartFusion2 M2S050 FPGA, for which reliability data is available in [79], was selected: for BRAM (assumed to be LSRAM in the selected device), the FIT rate is 344 FITs per 2^{20} bits. For flip-flops, and 248 FITs per 2^{20} bits are reported.
- For comparison with SRAM-based devices, also data for a Xilinx Zynq 7000 FPGA are provided. AMD publishes results from radiation experiments in their *Reliability Report*. The FIT rate for CRAM is stated to be 73 FITs per 10^6 bits, and the rate for BRAM is 66 FITs per 10^6 bits in the 2017 iteration [271]. No FIT rate for flip-flops is published. For this work, a flip-flop FIT rate equal to that of BRAM bits is assumed as a conservative estimate.

As part of this evaluation, the per-variable predictions of criticality are contrasted with the empirically determined criticality in the evaluated designs in the scatter plots shown in Figure 65. Outlier points more than three standard deviations from the mean estimated criticality value were omitted. As a basic measure of correlation Figure 65 also shows the results of a linear regression through the data points along with their coefficients of determination (r^2). This data shows how well the different metrics defined in Section 6.2.2 on the memory traces obtained from executions of the IR of an HLS-generated design predict the criticality in actual hardware for individual variables and their derived storage elements. It can be seen that while neither metric strongly predicts the in-hardware criticality ($r^2 < 0.72$ for all designs), the *Temp2* metric stands out as the metric delivering the best results in all evaluated designs.

This evaluation is extended towards the evaluation of the effect of the different profiling strategies on design reliability by applying assumed SEU protection to the top 50% as indicated by the three metrics would – perfect protection assumed – yield the criticality improvements shown in Table 90. The first columns of the table show the average storage element criticality (divided into the categories FF and BRAM, and the average of all bits), which is derived from the average share

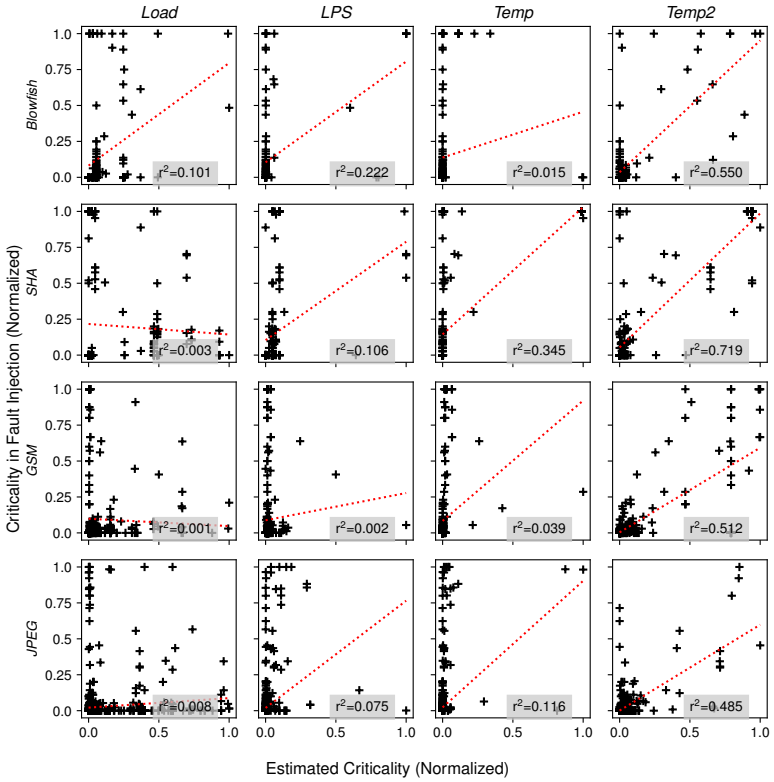
of fault injection runs delivering erroneous results when factoring in the protection as suggested by the different metrics. This table also shows the share of bits suggested to be protected by the respective metric, as well as the overhead necessary to protect these storage elements in terms of both FFs and BRAM bits. It is to be noted that the combinational logic to implement the necessary encoders and decoders, as well as the impact on circuit timing cannot be derived in this way as is done for HLL-based profiling. The register and memory overheads were calculated per storage element, assuming protection by ECC (use of a Hamming code [272] is assumed) and thus adding the required number of parity bits depending on the register or memory core's width[137]. For RAM blocks, this overhead applies to each word stored in the memory. From the improvement in reliability (i.e., the decrease in criticality) and the overhead, a BCR is calculated for bits stored in flip-flops and BRAM. The overall design BCR is calculated using a weighted average of these values. Finally, Table 90 shows the respective design variants' FIT on the two representative devices discussed above.

Thus, the four metrics can be compared – in addition to the correlation discussed above – by their impact on the design variants' reliability and resulting failure rate related to user-defined memory. To facilitate this comparison, the cells shaded in a color indicate those metrics that lead to the "best" value in the respective category (e.g., the lowest criticality or the highest BCR). The color hues visually differentiate the designs and bear no other meaning regarding the comparison. Finally, cells where all four metrics lead to similar results are shaded in gray.

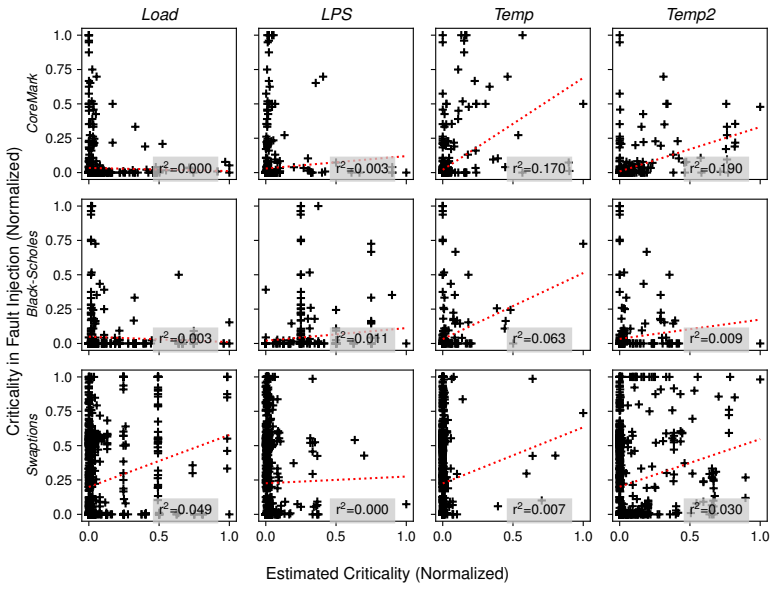
In all case-study designs except *Swaptions*, it can be seen that employing the *Temp* and *Temp2* metrics leads to a significant reduction in FF-related criticality compared not only to the original criticality (see Table 88) but also to the results achieved by the other metrics. While in most of these designs, *Temp* and *Temp2* perform similarly well, the *Temp2* metric performs significantly better in the FF category in the *JPEG* design. This not only applies to the absolute FF-related criticality but also to the BCR related to these elements, indicating that temporal analysis leads to a higher protection efficiency. *Swaptions* is the only design where *Load* as a simpler metric performs better with respect to FFs.

Due to the low granularity of the memory blocks included in the case-study designs (i.e., a memory block can only be protected as a whole), all metrics lead to a high percentage of BRAM bits suggested for protection by either metric. An exception is, once again, the *JPEG* design, where only between 41% and 63% of BRAM bits are suggested for protection. While *Temp2* reaches higher efficiency in BRAM protection, in absolute numbers it reaches the worst BRAM protection in this design. In the other designs, higher BRAM protection efficiency leans towards the *Temp*/*Temp2* metrics as well (with the exception of *Swaptions*), while in terms of absolute protection, this holds for all designs except for *Blowfish* and *Swaptions*.

[137] That is, two parity bits for a single-bit register, four for an eight-bit register, and six for a 32-bit register[272]



(a) Designs derived from CHStone [229]



(b) Designs derived from CoreMark and Parsec [264]

Figure 65: Correlation of Profiling-Derived Predictions and Fault-Injection-Derived Criticality per Profiling Metric (*Load*, *LPS*, *Temp*, *Temp2*)

Table 90: Criticality, Benefit-Cost-Ratio (BCR), and FIT Rate by IR-profiling-based ECC protection metrics

Design	Metric	Criticality (%)			Covered (%)		ECC Overhead (%)		BCR			FIT Rate	
		FF	BRAM	Total	FF	BRAM	FF	BRAM	FF	BRAM	Total	M2S050	Zynq 7000
Blowfish	*Load*	2.41	0.37	0.46	57.7	96.8	16.8	16.5	0.53	10.78	10.30	0.060	0.013
	LPS	4.29	0.74	0.91	41.0	93.8	13.6	15.6	0.50	11.29	10.78	0.119	0.025
	Temp	1.35	0.74	0.77	70.0	93.8	17.2	15.6	0.58	11.29	10.79	0.105	0.022
	Temp2	1.35	0.64	0.68	70.0	94.0	17.2	15.7	0.58	11.27	10.77	0.092	0.019
SHA	*Load*	6.95	0.00	0.13	68.1	100.0	16.9	43.5	0.89	15.00	14.74	0.047	0.013
	LPS	6.07	0.00	0.11	51.3	100.0	14.6	43.5	1.10	15.00	14.74	0.041	0.011
	Temp	2.41	0.01	0.06	73.5	99.6	16.4	43.4	1.26	15.02	14.76	0.022	0.006
	Temp2	2.41	0.01	0.06	73.5	99.6	16.4	43.4	1.26	15.02	14.76	0.022	0.006
GSM	*Load*	1.70	0.00	0.63	61.3	92.2	16.1	25.6	0.16	2.61	1.71	0.024	0.007
	LPS	1.95	0.17	0.83	56.0	90.8	14.9	25.2	0.15	2.64	1.72	0.033	0.009
	Temp	1.02	0.17	0.48	43.6	90.8	12.7	25.2	0.26	2.64	1.76	0.020	0.005
	Temp2	1.02	0.17	0.49	43.3	90.8	12.7	25.2	0.26	2.64	1.76	0.020	0.005
JPEG	*Load*	2.41	0.08	0.16	57.7	59.1	15.5	13.0	0.07	0.00	0.00	0.239	0.056
	LPS	2.19	0.06	0.13	49.6	51.6	14.0	11.8	0.12	0.00	0.00	0.197	0.047
	Temp	2.19	0.00	0.07	48.9	62.7	14.0	13.6	0.11	0.00	0.00	0.094	0.026
	Temp2	0.97	4.78	4.66	50.4	40.9	14.0	8.4	0.20	0.00	0.01	8.172	1.647
CoreMark	*Load*	2.59	2.44	2.50	52.5	90.8	13.9	24.5	0.12	1.11	0.73	0.257	0.058
	LPS	2.26	2.44	2.37	53.2	90.8	14.0	24.5	0.14	1.11	0.74	0.247	0.055
	Temp	1.95	2.44	2.25	52.7	90.8	14.1	24.5	0.17	1.11	0.75	0.237	0.052
	Temp2	1.95	2.44	2.25	53.0	90.8	14.1	24.5	0.17	1.11	0.75	0.237	0.052
Black-Scholes	*Load*	2.28	0.00	0.20	49.5	100.0	12.1	16.7	0.28	2.94	2.70	0.034	0.010
	LPS	2.14	0.00	0.19	49.8	100.0	12.2	16.7	0.29	2.94	2.70	0.032	0.009
	Temp	2.04	0.00	0.18	45.4	100.0	11.4	16.7	0.33	2.94	2.70	0.030	0.009
	Temp2	2.04	0.00	0.18	45.3	100.0	11.4	16.7	0.32	2.94	2.70	0.030	0.009
Swaptions	*Load*	10.32	0.00	1.33	46.4	100.0	9.6	27.8	2.17	0.00	0.28	0.532	0.148
	LPS	12.23	0.00	1.58	51.2	100.0	10.5	27.8	2.26	0.00	0.29	0.631	0.176
	Temp	11.62	0.00	1.50	44.3	100.0	9.5	27.8	2.18	0.00	0.28	0.599	0.167
	Temp2	11.65	0.00	1.50	43.7	100.0	9.4	27.8	2.15	0.00	0.28	0.601	0.168

Taken as a whole, the results of the evaluation of the IR-level profiling approach towards protecting user-defined storage elements suggest that the temporal analysis of the design behavior plays a role in efficiently selecting vulnerable storage elements for protection: Temporal analysis both delivered the best correlation between IR-level prediction and in-hardware criticality, as well as the overall best results when assuming metric-guided selection of storage element protection in the set of case-study designs discussed above. However, the observed correlations are marginal, suggesting that the IR-level profiling does not wholly capture the behavior of the design at runtime.

This may be the case because the execution of the IR of the design on a host CPU may still be far removed from the actual behavior in hardware, even if it is closer to the hardware than the HLL profiling discussed earlier. An important aspect that is not adequately captured by the execution on a host is the implicit concurrency of a hardware implementation. This behavior is, by nature, sequentialized when performing the same operations in a software execution on a CPU model as done with *Vargrind*. An approach that additionally uses the schedule found during HLS may deliver more accurate results.

Furthermore, to execute the IR under `valgrind` to be profiled, it is compiled as a program for a certain CPU architecture (x86 in the case of this evaluation), which may introduce additional discrepancies. Directly executing the SSA GIMPLE representation – or the LLVM IR when the `clang` frontend of an HLS tool is used – in a simulator may help to mitigate these discrepancies.

The approach's effectiveness appears to be influenced by a certain design-dependent component that is visible in the comparison of the case-study designs: In five out of the seven evaluated designs, temporal analysis delivers the best FIT rate, while simple counting of load accesses delivers the best results in two others.

In literature, [273, 254, 274] propose lower-level approaches to prioritize subsets of a design's flip-flops for selective SEU mitigation. [273] presents an approach based on circuit- and netlist-level fault simulation, that incrementally selects flip-flops for mitigation based on their contribution to the overall soft-error sensitivity. In their experimental evaluation on two multiplier circuits (an integer multiplier and a floating-point multiplier), the authors of [273] show that only a fraction of flip-flops are responsible for a large share of circuit failures. The authors summarize their findings by estimating that a 20$\times$ reduction in soft-error sensitivity can be achieved by augmenting just 40% of flip-flops in the floating-point multiplier case with fault-detection logic. Secondly, [254] proposes clustering flip-flops for soft-error mitigation using static RTL analysis based on design hierarchy and bus membership. The clusters of flip-flops are then subjected to statistical fault simulation, which enables the authors to sort them based on their predicted mean soft-error sensitivity. These clusters of flip-flops can then be targeted using selective soft-error mitigation, starting with the most sensitive ones. In their experimental evaluation of selective soft-error mitigation, the authors of [254] apply their approach to two blocks of a commercial network processor, referred to as *Epsilon* and *Omega*. By applying fault mitigation to only approximately 4.1% of flip-flops in the *Epsilon* design, [254] achieves a 7$\times$ reduction in soft-error sensitivity. On the other hand, applying fault mitigation to 38% of flip-flops in the *Omega* design achieves a 10$\times$ reduction. As a third point of comparison, [274] proposes an approach with a similar aim. It, too, aims at identifying which flip-flops in the synthesized circuit should be prioritized when introducing soft-error mitigation measures to achieve cost-efficient soft-error protection. The authors suggest instantiating radiation-hardened cell implementations or applying TMR for this purpose. To that end, the proposed approach extracts a set of features from each storage element described in RTL, including structural properties such as the flip-flop's proximity to primary outputs and its fan-out, as well as signal-activity properties derived from RTL simulation traces. These features are then used to divide the design's flip-flops into clusters according to their predicted soft-error sensitivity, with each cluster covering a fixed fraction of the design's flip-flops. Using statistical fault injection, the mean sensitivity of the

clusters is determined, and the flip-flops in the most soft-error sensitive clusters are protected. The approach is evaluated on an RTL description of a 10 GBit/s Ethernet MAC core, with the cluster-based approach compared to an ideal grouping with exhaustive fault simulation. The results of this *ideal* protection strategy as performed in [274] reflect the results obtained from the floating-point multipier evaluated in [273]: Protecting 40% of flip-flops against soft errors is enough to achieve near full error coverage.

When considering soft-error protection efficiency, the cost of protection must be defined. This can be done, for example, in terms of the introduced overhead. To fully mask a bit flip in a flip-flop, a mitigation approach such as transparent TMR or ECC must be used. When TMR protection on the individual flip-flop level is assumed, the total number of flip-flops N_{FF} in the mitigated design can be calculated from $N_{FF,0}$ (the number of flip-flops in the original design) and p_{TMR} (the proportion of flip-flops to be triplicated) as $N_{FF} = N_{FF,0} \cdot (1 + 2 \cdot p_{TMR})$. The associated overhead O_{FF} can be calculated as $O_{FF} = \frac{N_{FF} - N_{FF,0}}{N_{FF,0}}$. Thus, mitigating 40% of flip-flops using per-flip-flop TMR would incur an 80% overhead. Considering the column titled "Covered" in Table 90, it can be seen that a share of flip-flops exceeding the 40% reported by [273] is covered by fault mitigation measures using the approach presented in this chapter, while not achieving the associated high reduction in soft-error sensitivity observed in [273, 274] at this level of mitigation. However, as the flip-flops to be augmented by soft-error mitigation measures stem from FUs instantiated by HLS, which often represent banks of flip-flops with identical access patterns, the approach proposed in this dissertation allows more efficient ECC schemes to be used, leading to a reduced overhead[138]. The approaches described in [273, 254, 274], however, do not report this functional grouping, requiring the application of TMR to the individual flip-flops. Thus, for a comparison of overall protection efficiency, flip-flop-level TMR is assumed. For the sake of comparability, only the flip-flop-related results of the evaluation of the approach proposed in this dissertation (see Table 90) are considered.

It can be seen that the approach proposed in this dissertation incurs a flip-flop overhead of between 9.4% (*Swaptions/Temp2*) and 17.2% (*Blowfish/Temp(2)*). The largest reduction in flip-flop-related soft-error sensitivity achieved by the approach proposed in this dissertation is 8x at a flip-flop overhead of 16.4% in the *SHA/Temp(2)* case study[139]. When constrained to a TMR-induced flip-flop overhead of an equivalent 16.4%, i.e., the protection of 8.2% of flip-flops, the approach proposed in [273] still achieves an error coverage of 80% and a 16$\times$ reduction in error rate in the floating-point multiplier case study[140]. Finally, in the integer multiplier case study described in [273], an equivalent level of flip-flop protection achieves an error coverage of approximately 32%, and thus only a reduction of 1.5$\times$ in soft-error rate[141]. This reduction is exceeded in all case studies in this work at similar or lower flip-flop overheads[142]. Overall, the *Temp2* selection strategy leads to reductions in flip-flop related soft-error sensitivity between 1.98$\times$ and 8$\times$ (Median: 3.4$\times$) with overheads between 9.4% and 17.2% (Median: 14%).

Considering the *Epsilon* in [254], where a 7$\times$ reduction in soft-error sensitivity is achieved by protecting just 4.1% of flip-flops[143], protecting these flip-flops using TMR would incur an overhead of 8.2%. This outperforms the approach discussed in this work in the presented case studies, achieving an 8$\times$ reduction only in the *SHA/Temp(2)* case study, albeit with a flip-flop overhead of 16.4%. On the other hand, applying fault mitigation to 38% of flip-flops in the *Omega* design

[138] This assumes that single-bit errors as the predominant fault model in the design's storage elements.

[139] Flip-flop-related soft-error sensitivity is reduced from 19.1% to 2.4% in the SHA/Temp(2) case study

[140] See [273, Figure 10]

[141] See [273, Figure 11]

[142] Only Blowfish/Temp(2) incurs a higher overhead of 17.2%, achieving a 7.7$\times$ reduction in soft-error rate

[143] See [254, Figure 10]

achieves a 10× reduction. When using the protection of 8.2% of flip-flops using TMR (derived from the SHA/Temp(2) case study) as a point of comparison, the nearest data point to be found in the *Omega* design in [254] is 10% mitigation coverage[144]. Using this level of mitigation, an approximate 2.6× reduction in soft-error sensitivity is achieved in [254], which would incur a TMR overhead of 20%. This reduction is exceeded by the presented approach at lower overheads in all case studies except *CoreMark* (2.15× soft-error sensitivity reduction) and *Swaptions* (2.23×).

Finally, also for comparison with [274], a protection of 8.2% of the flip-flops using individual TMR – resulting in a flip-flop overhead of 16.4% – is assumed. According to the data published in [274][145], this would lead to a reduction of the flip-flop-related soft-error rate by approximately 3.4×. This is comparable to the median reduction in flip-flop-related soft-error sensitivity achieved by the approach proposed in this dissertation when applied to the case-study designs discussed in this chapter.

It can be seen particularly from the comparison to the floating-point multiplier case study in [273], the *Epsilon* case-study in [254], and the 10 GBit/s Ethernet MAC in [274] that impressive reductions in soft-error sensitivity can be achieved while covering only a fraction of flip-flops with fault mitigation measures. The approach presented in this chapter does not achieve this level of protection while covering a larger share of flip-flops in most cases. However, due to the register-bank grouping inherent to flip-flops in HLS-generated designs, the reductions in soft-error sensitivity at a flip-flop overhead of 16.4% chosen for comparison[146] yield comparable results. Furthermore, [273, 254, 274] achieve their respective reductions in soft-error sensitivity by conducting varying amounts of time-consuming fault simulation at RTL or netlist level, whereas the proposed approach directly analyzes a higher-level design representation. This allows these measures to be introduced earlier in the design flow and potentially enables less costly design-space exploration.

Nevertheless, in addition to improving the correlation between soft-error sensitivity predictions at the HLL or IR level and the soft-error sensitivity observed in hardware, future work should also conduct a more in-depth comparison of the soft-error coverage achieved by the approach. Particularly useful – as it is widely used in [273, 254, 274] – would be a comparison of the achieved reduction in soft-error sensitivity when incrementally covering storage elements as suggested by the presented approach and metric(s). In its current form, the data presented in Table 90 is predominantly useful for comparisons among the proposed metrics (Load, LPS, Temp, Temp2), while this comparison would improve the overall comparability with other approaches.

6.5 Summary

This chapter is concerned with the protection of FPGA designs against the consequences of bit flips affecting the storage elements defined by the user's design. In the context of this dissertation, this complements the previous Chapter 4 on bitstream-level fault modeling and Chapter 5, which focus on the protection against bit flips in configuration memory. Considering the user-defined storage elements as well is important for (1) ensuring protection of designs in FPGAs using SEU-insensitive configuration technology (e.g., Flash-based or Antifuse-based devices) and (2) for ensuring the integrity of an SRAM-based FPGA design's state even if configuration-level fault mitigation measures are added. If a selective TMR approach such as the one presented in Chapter 5 suggests certain modules for protection against configuration bit flips, some non-redundant modules remain. Even if they are not deemed critical enough to be protected using

[144] See [254, Figure 11]

[145] See [274, Figure 2]

[146] Stemming from the *SHA/Temp(2)* case study, see Table 90

full-scale logic redundancy, their storage elements may be protected at a lesser cost using, e.g., ECC, thereby making them at least resilient against bit flips in their state.

The approach presented in this chapter applies the notion of the Register Vulnerability Factor (RVF) [259] – i.e., the share of total execution time of a program during which a given register value is vulnerable to bit flips due to the later propagation of the now erroneous value – from the analysis of the register-level soft-error vulnerability of the execution of programs on a conventional CPU or the analysis of the criticality of storage elements in digital logic designs generated by HLS. This chapter proposes to profile a design for the criticality of its storage elements using memory traces recorded from executions of (1) the original HLL source code or (2) its lower-level Single Static Assignment (SSA) Intermediate Representation (IR) on a virtual machine to identify potentially vulnerable variables that are later mapped to storage elements during the HLS process. Its underlying assumption is that the behavior of these executions concerning the variables storing temporary values reflects the behavior of the resulting hardware implementation well enough to deliver insights into the relative storage elements' vulnerabilities. For the HLL-based analysis, the number of accesses to each variable is seen as a measure of the relative criticality of each HLL variable. For the IR-based analysis, four metrics are introduced – and later contrasted in evaluation – concerning the numbers of load (read) and store (write) accesses to variables in the IR, as well as the number of virtual instructions executed between accesses as a measure of the temporal behavior of the IR.

Both approaches are evaluated on case-study designs derived from popular benchmark suites such as CHStone [229] or Parsec [264]. The HLL-based approach is evaluated by contrasting the storage-element-related criticality of each original HLS-derived case-study design under fault injection with that of variants protecting the "less-important" and "more-important" half of its storage elements using ECC. Results indicated that, indeed, protecting the "more-important" half of their storage elements significantly reduced the storage-element-related criticality compared to both the original variant and the variant protecting the "less-used" variables. However, results also showed that not all storage elements in the final design can be captured in this way, as several storage elements are instantiated from intermediate values necessary to be stored to implement the basic-block-level functionality of the design as defined by the schedule found during HLS, which motivated the IR-level analysis approach.

The evaluation of the IR-based analysis approach showed that temporal analysis (i.e., assigning temporal importance to the number of virtual instructions executed between two accesses to the same variable) delivered the best correlation between profiling-based criticality prediction and actual in-hardware criticality under fault injection when compared to the other criticality metrics defined over the recorded memory traces. Furthermore, protecting the "more-important" half of the storage elements suggested by these "temporal" metrics leads, in most case-study designs, to a lower overall design criticality as well as to a better protection efficiency when comparing benefit (criticality reduction) to cost (increased number of storage elements due to ECC parity bits). However, even the best observed correlations between profiling-based prediction and in-hardware criticality were not particularly strong ($r^2 < 0.72$), and varied from design to design. While this suggests that temporal analysis of the design's register access patterns may be useful in predicting the resulting storage elements' criticality, the precision of this approach is likely impacted by differences in the linear execution of a design's HLL or IR on a (simulated) CPU and the inherently more concurrent hardware implementation generated by HLS. Furthermore, neither the design in its HLL nor IR form has access to the schedule found during HLS, thus lacking an important source for the hardware design's temporal behavior that is strongly connected to the RVF. Thus, the temporal behavior can only roughly be predicted from these high-level implementations by implicit causal relationships between blocks of code within the design's source code.

Compared to related works with a similar objective [273, 254, 274] working on RTL or netlist-level, the proposed profiling approach covers a larger share of the design's flip-flops while achieving a lower reduction in soft error sensitivity. This points at a suboptimal selection of storage elements for protection with fault mitigation measures that may be related to (1) the coarser FU-related grouping of storage elements to be protected in contrast to protection measures at individual flip-flop granularity and (2) the above-mentioned issues in predicting storage element criticality from the HLL or IR level. However, as the approach works on register-bank level (stemming from the storage FU instantiated by the HLS process), the selected groups of flip-flops have identical access patterns, enabling protection using more resource-efficient ECC instead of requiring per-flip-flop TMR. When judged at low overheads – comparable to those achieved by the approach using ECC – the presented profiling approach achieves reductions in soft error sensitivity in the case studies evaluated in this work that are comparable to related work at equivalent TMR overheads.

In future work, a more precise profiling approach may utilize the schedule exported by a preliminary HLS tool run to better interpret the memory access patterns recorded during profiling execution runs. This may improve the accuracy of criticality prediction as it includes more precise temporal information about the control steps, i.e., clock cycles between different register accesses. The memory access patterns may be generated, instead of by compiling the design's IR for execution on a particular CPU architecture, by directly executing the IR in an interpreter or simulator. This should be guided by an in-depth comparison against comparable approaches such as described in [273, 254, 274]. As a future use, analysis of a design's storage elements' criticality may be used to inform a DSE approach such as presented in Chapter 5 of certain modules with a high concentration of "important" storage elements. This may, in turn, suggest certain modules for an initial solution set, possibly improving the number of generations needed to converge to a sufficiently well-performing set of design variants.

Chapter 7

Discussion

The overarching goal of this dissertation is to improve the understanding of the factors that affect the soft-error susceptibility of digital logic designs implemented on FPGAs. A further goal of this dissertation, based on this improved understanding, is to contribute approaches to automatically and cost-effectively introduce fault mitigation measures into the respective designs. Due to their unique flexibility, density, and thus utility on the one hand, and their larger vulnerability to soft errors on the other hand, the main focus lies on SRAM-based FPGAs. Together, the contributions made by this dissertation allow a more precise assessment of a design's innate robustness against soft errors in both the configuration memory of SRAM-based FPGAs and user-defined memories. Combined with technology- and mission-specific soft error probabilities, they provide a more accurate representation of the likelihood that a given design will exhibit a potentially hazardous failure. This chapter presents the most significant findings from the previous chapters and discusses their limitations, as well as potential ways to address them in future work.

7.1 Configuration Memory Fault Modeling

The effects of soft errors affecting the design's configuration are modeled in Chapter 4. The entire modeling process, starting from the target device's bitstream and post-routing netlist, and arriving at a well-founded prediction of the effects of individual configuration bit flips on the design's functionality. While there are related developments by the scientific community concurrent to the writing of this dissertation – such as [121, 124] – this dissertation extends the relevant state of the art (see Section 3.2), especially in the following ways:

1. This dissertation proposes a systematic empirical fault characterization approach for bit flips affecting the configuration of interconnect elements in the target FPGA.

   Applied to real-world devices, this provided insights about technological masking of certain types of interconnect bit flips in several FPGA families. For example, both for AMD 7 Series FPGAs and Lattice's ECP5 family, the conditions under which *Input Antenna* faults are masked by the technology can be determined experimentally. These findings, combined with more precise configuration-related fault modeling for LUTs, have been thoroughly validated using several case studies on a set of FPGAs belonging to AMD's 7 Series device family.

   This work shows that the proposed fault modeling approach allows a more precise discrimination of likely-to-be critical and non-critical configuration bits. Compared to the number of false positives when using the vendor's *essential bits* feature, the predictions made by the

proposed implementation reduce the number of falsely predicted critical bits by between 74% and 98% across the four case-study designs. This corresponds to a reduction in fault injection effort (i.e., the number of bits requiring evaluation using fault injection) of between 60% and 98%. The process only incurs a small number of false negatives (i.e., bits predicted to be non-critical that turned out to be critical in exhaustive validation experiments) of between 0 and 6 bits (i.e., $\leq$ 0.008%). However, the assessment remains pessimistic, with false positives (i.e., bits predicted to be critical that turn out to be non-critical when tested in fault injection) still ranging between 43% and 91% of predicted-critical bits.

2. With ever-shrinking feature sizes, PVT effects in manufactured digital integrated circuits become more and more significant. This includes larger inter- and intra-chip variations in the performance of FPGA routing resources such as multiplexers and drivers. Configuration-level faults that affect routing elements are likely to create circuit configurations that violate the typically cleanly defined digital behavior of a circuit implemented in an FPGA, causing wire segments to float or resulting in short-circuits. These erroneous configurations are predestined to show PVT-related inter- and intra-chip variations in fault consequences.

   This dissertation contributes the first systematic analysis of variations in fault effects related to chip temperature, operating voltage, and device-to-device variations. It found significant variations in fault effects with all of these factors for the popular AMD 7 Series FPGA family and Lattice's iCE40 low-power FPGAs.

   This is demonstrated both in purpose-built synthetic evaluations and a case-study design comprising several widely used FPGA design components (LFSR, counter, and FSM). Experimental evaluation of device-to-device variation on a synthetic test design on a set of 16 AMD Artix 7 FPGAs revealed that while the absolute number of critical bits stays consistent, the set of bits observed to be critical on *any* device is significantly larger than the set of bits observed to be critical on *all* devices.

   Experiments conducted on AMD's 7 Series FPGA devices show that this can be exacerbated with chip temperature: The intersection-over-union (IoU) quotient (i.e., the ratio of bits critical on *all* vs. *any* device) fell from 93% observed at 50 °C environment temperature to 76% observed at 20 °C. A similar effect was observed in a systematic evaluation of *Input Antenna* faults at different operating voltages in this technology. Here, the IoU quotient of critical faults fell from 96% at 1.05 V (the device's maximum supply voltage) to 65% at 0.88 V (the device's minimum supply voltage).

3. Building on the empirically derived criticality conditions of particular interconnect faults on AMD 7 Series FPGAs, a zero-overhead bitstream patching approach is proposed.

   It is shown that by inverting the initialization of related but unused LUTs, the preconditions for the criticality of these interconnect faults can be eliminated: A bitstream patched in this way has a reduced number of critical configuration bits and thereby a smaller error cross-section without any resource usage penalty. The effectiveness of masking these faults by patching is evaluated in four case-study designs, showing that between 1.1% and 3.6% of critical bits can be masked in this manner.

   In addition to validating the effectiveness of this masking approach, this also indicates the correctness of the fault models for AMD 7 Series interconnect resources, which were initially derived solely based on their configuration layout.

In the design of future FPGA-based systems for safety-critical applications, these contributions allow, on the one hand, accelerated fault injection experiments due to a more precise discrimination between potentially critical and non-critical configuration bits. This aids in reducing pessimism in

the reliability assessment of these designs, which in turn reduces total system cost by fine-tuning the amount of fault-masking redundant logic to the application's reliability requirements. While this work, in line with a large portion of the state-of-the-art, mainly focuses on AMD 7 Series FPGAs, the concepts for developing detailed fault models are transferable to other devices and technologies. This is demonstrated by the additional consideration of Lattice's iCE40 and ECP5 devices.

On the other hand, the contributions discussed above clearly suggest that when evaluating the robustness of a particular FPGA design against configuration memory bit flips, it is insufficient to do so on one device only due to the presence of inter-device variations in fault injection results. Furthermore, the devices' operating conditions, such as supply voltage and chip temperature, must be closely monitored during such experiments to obtain reliable results.

Finally, the resulting improved understanding of the factors contributing to individual-bit criticality allows the implementation of more sophisticated and targeted fault mitigation approaches that can better exploit technological effects masking the criticality of certain configuration bit flips.

7.2 Design-Space Exploration for Selective Fault Mitigation

Chapter 5 takes up the latter suggestion and proposes an automated cost-reliability design-space exploration approach targeting configuration memory bit flips. This approach views the problem of cost-effectively generating a reliable SRAM-based FPGA design as the problem of deciding which subset of a design's submodules in a hierarchical netlist to protect against SEUs using TMR. The proposed approach involves encoding this problem as a chromosome for a genetic algorithm, along with the definition of a set of fitness functions that determine the cost (in terms of configuration bits utilized by the design) and reliability (in terms of the number of critical bits). It results in a set of Pareto-optimal designs at different points of the cost-reliability trade-off inherent to the individual design.

An approach of this kind, however, must appropriately address the complexity of the FPGA implementation process itself: Even small changes in the design's netlist before placement and routing can result in significant changes to the resulting device configuration. This can happen, e.g., due to different placement and routing necessary to accommodate additional redundant logic and voters, thereby both lengthening and shortening individual routes. Thus, the proposed approach involves performing full placement and routing runs for each design candidate. This is in contrast to existing similar approaches, such as [179], which assume that reliability evaluations for each component can be done in isolation and later linearly combined to obtain the possible design space statically. This, however, does not accurately capture the reliability-related effects of placement and routing in the resulting bitstream-level design. While performing full placement and routing runs in the fitness function increases total runtime, the proposed approach is the first of its kind to appropriately account for the effects of altered placement and routing.

Furthermore, Chapter 5 contributes an implementation of the approach discussed above for AMD 7 Series FPGAs that integrates the findings from Chapter 4. Fitness functions based on static criticality prediction, building on Chapter 4, as well as fitness functions that supplement these predictions with (statistical) fault injection, were implemented. Care has also been taken to develop a way of addressing possible inter-device variations in fault injection results for the latter fitness functions by employing multiple evaluation devices.

Based on this implementation, the proposed design-space exploration approach is evaluated using a set of six case study designs. The optimization procedure was carried out with four variants of

reliability-evaluating fitness functions for each design, utilizing an initial set of twelve AMD Artix 7 FPGA devices to perform fault injection. The resulting Pareto-optimal designs were subjected to another round of fault injection on both the initial set of twelve devices as well as a “control” set of an additional twelve FPGAs to assess the effect of PVT variations on the validity of the optimization results.

It was shown that the implementation discussed in Chapter 5 produces the best results when the reliability-related fitness function has access to fault injection results that refine the static criticality prediction, as outlined in Chapter 4. While purely static fitness assessment also results in a significant improvement in the reliability of the generated design variants compared to a randomized starting set, better results can be achieved by incorporating additional fault injection experiments. This discrepancy is caused by the pessimism in the static criticality prediction – i.e., bits falsely predicted to be critical.

Nevertheless, all variants of the approach were able to significantly improve on a random starting set of solutions over the course of 16 generations: When compared to a final “global” Pareto front across all four optimization runs for each of the six case-study designs, the presented optimization approach yielded the following growth of the Pareto front’s hypervolumes from the starting generation to the final (16^{th}) generation (in percentage points of the final hypervolume):

- Using **static criticality analysis only**: Between 14 and 47 percentage points, median 18.5.
- Combining **static criticality analysis with fault injection**: Between 13 and 49 percentage points, median 26.
- Using static criticality analysis to pre-filter bits for **fault injection on one device**: Between 18 and 59 percentage points, median 28.5.
- Using static criticality analysis to pre-filter bits for **fault injection on multiple devices**: Between 18 and 56 percentage points, median 28.

This indicates that static criticality analysis, in its implemented form, is both useful for pre-filtering bits for fault injection and, to a lesser extent, as the sole component for assessing criticality-related fitness. Due to the higher precision in pointing out critical bits, incorporating fault injection experiments into the fitness function yields better result quality at the cost of increased runtime.

Compared to other related approaches, the proposed black-box approach (i.e., no internal details of the design need to be known by the approach) yielded benefit-to-cost ratios (BCRs) comparable to [142], a white-box approach that identifies certain feedback structures in the ITC’99 B13 benchmark circuit [164] and protects them using TMR. It further mostly outperforms the BCRs provided in [179] in the chosen low-overhead comparison region.

Finally, validations with two sets of twelve FPGA devices each – as briefly touched on above – show that the proposed approach is resilient towards PVT variations: Re-evaluating the reliability of the generated Pareto-optimal designs led to only minimal shifts in the Pareto fronts and their covered hypervolumes.

The proposed DSE concept and its implementation for AMD 7 Series FPGAs enable the designer to select a design variant that provides the appropriate characteristics for the desired application, mission, or mission phase. Since all designs are functionally equivalent, and SRAM-based FPGAs are freely reconfigurable, it is thus possible to choose a less reliable design variant that utilizes fewer resources (and thus, likely, consumes less energy) for a less critical mission phase, while selecting a highly reliable variant for a more critical one. This is an advantage in contrast to optimization approaches that yield a single design variant optimized to meet a specific cost-reliability goal.

With respect to the requirements during the development of FPGA designs targeting safety-critical applications, the fitness function directly yields an evaluation of the (predicted) number of critical bits, and thus – combined with vendor data – a way to estimate the contribution of the configuration memory to the design's FIT rate.

7.3 High-Level Storage Element Reliability Profiling

Finally, while Chapter 4 and Chapter 5 focus on the improvement of resilience against configuration-memory bit flips, Chapter 6 complements this by proposing an approach aimed at automatically analyzing and predicting the vulnerability of user-defined storage elements (flip-flops and block memories) to SEUs during runtime. The approach focuses on designs generated by HLS.

Its goal is to derive the register vulnerability factors (RVFs) for each storage element instantiated during HLS by profiling a design either in its original high-level language representation or an intermediate representation emerging during the HLS process. Due to the higher abstraction level, this promises faster evaluation runs than performing fault simulation on RTL or gate level. This is deemed possible for user-defined storage elements as the effects of bit flips in these elements are assumed to be less dependent on the technology-level implementation of the circuit.

An experimental fault-injection evaluation of the two sub-concepts (profiling the HLL or IR) suggested that RVF estimations can be determined for a larger share of storage elements when profiling the design's IR during the HLS process. Furthermore, profiling metrics that relate the read and write accesses made to the instantiated storage elements with a temporal component (i.e., the duration between a write and read access to a storage element) yielded the best results compared to simpler metrics when assuming protection of the most critical storage elements according to either metric. However, in general, the correlation between the RVF derived from profiling the design's IR and the in-hardware criticality under fault injection was not observed to be particularly strong: r^2 was observed to be between 0.485 and 0.719 for four designs derived from the CHStone [229], and only between 0.009 and 0.190 for designs derived from CoreMark and Parsec [264]. This indicates that IR profiling only delivers a partial picture of storage element criticality, while still delivering some insights into their criticality, aiding in the choice of which storage elements to protect.

The findings of Chapter 6 can be integrated with those of Chapter 4 and particularly Chapter 5 as follows: Protecting storage elements against bit flips in an SRAM-based FPGA necessarily means adding logic. This includes, for example, triplicating register bits and inserting voters, or the insertion of ECC encoders and decoders at the inputs and outputs of register banks or memory blocks. This, in turn, affects the design's placement and routing, and thereby potentially affects the configuration-related reliability of the design. The DSE approach proposed in Chapter 5 may be expanded to also cover user-defined storage elements by adding a third fitness dimension – the predicted impact on storage element reliability of the found replication solution, which implicitly also protects the values stored in flip-flops and memory blocks. For this prediction, not only the number of protected storage elements, but also the criticality of the remaining unprotected storage elements is relevant. These numbers may be derived from a profiling run as discussed above for HLS-generated designs. For designs implemented on RTL, fault simulation results are necessary.

Furthermore, concentrations of highly critical storage elements in certain hierarchical submodules, as determined by profiling, may also indicate the overall criticality of that design unit to the entire design's correct functionality. As this data is available from high-level profiling before carrying out the DSE run as proposed in Chapter 5, this information can be used to generate a more informed initial population for the genetic algorithm. This may lower the time – or number of generations –

necessary to reach convergence.

7.4 Key Limitations

Finally, this section summarizes the key limitations of the concepts and evaluations discussed in the previous sections, assesses their impact, and outlines directions for addressing these in future work. A more in-depth discussion of these limitations can be found in the respective chapter.

The key limitations of the configuration bit flip fault modeling approach proposed in Chapter 4 are as follows.

LM1: The proposed fault modeling approach focuses on bit flips affecting single configuration bits. In real-world radiation experiments, most observed events are SEUs. However, a more rare but still significant portion of events flips multiple bits at once (i.e., MBUs). Extending the fault modeling approach to MBUs requires modeling the combined effects of two or more bit flips in the circuit by applying the resource-related fault models to each configuration bit part of the MBU. This opens the possibility for more complex fault conditions, e.g., an interconnect fault that short-circuits three design signals, or two design signals and an unused wire segment. Accurately predicting the criticality of these faults requires extended characterization experiments evaluating the general effect of these conditions.

LM2: While the proposed fault modeling approach and the derived static criticality prediction procedure significantly reduce fault injection effort, and do so while generating minimal false negatives (i.e., a negligible amount of bits are falsely predicted to be non-critical), the prediction is still pessimistic. This results in a considerable amount of false positives, which in turn reduces the utility of directly using the static prediction results to direct the inference of fault mitigation measures. Therefore, fault injection experiments are still required to obtain an accurate picture of the criticality of individual configuration bits. This pessimism stems, in part, from the purely static and structural nature of the implemented criticality prediction approach: No dynamic – run-time – information about the design is considered during prediction. This may be addressed by combining the proposed static criticality prediction with simulation-derived information as proposed by [124].

Addressing the following limitations of the configuration-level cost-reliability DSE approach proposed in Chapter 5 may increase its applicability.

LD1: Since the implementation of the DSE approach is based on the fault modeling approach discussed above, it inherits its limitations. The remaining pessimism in the static criticality prediction led to optimal results only being achievable by performing time-consuming fault injection experiments as part of the fitness evaluation. Especially for larger designs, this drastically increases the overall optimization runtime, requiring to resort to statistical fault injection for the two largest evaluated designs. Improving the static criticality prediction – i.e., reducing its pessimism by, for example, including design information gleaned from simulation traces – improves the performance of the faster fitness functions that forgo fault injection, and thereby enhances the approach's temporal performance. This also applies to the support for MBUs as discussed above.

LD2: The proposed DSE approach assumes – in line with related approaches in the state of the art [179, 232] – that the design's "natural" hierarchy stemming from the original RTL implementation or instantiated by an HLS tool is suitable to derive cost-reliability trade-offs in

partial TMR solutions. While the proposed DSE approach shows promising results with case studies of different hierarchical makeups, this assumption (1) limits the applicability to design flows integrating cross-hierarchy optimization and design flattening, which may negatively impact the design's logic density, and (2) may lead to sub-optimal results in designs with both highly granular and coarse hierarchies as well as severely unbalanced hierarchical makeup. Thus, the approach's effectiveness may be improved in future work by performing systematic analyses of the effects of hierarchical structure and granularity on DSE result quality. An initial analysis of the original design's hierarchy based on these results may inform the designer which changes to the hierarchy would improve DSE performance. Furthermore, to support cross-hierarchy optimizations, as well as highly granular, coarse, and flat designs, future work should combine the proposed hierarchy-based DSE approach with clustering approaches such as those proposed in [161, 165, 142, 254].

LD3: The implementation of the proposed approach does not physically separate TMR circuit copies, thereby incurring common-mode faults. This lowers the efficacy of the introduced redundancy. In literature, multiple placement and routing approaches aware of common-mode faults have been proposed (see Section 3.3.1). Integrating these approaches into the FPGA placement and routing tool used in the fitness evaluation of the proposed DSE approach would mitigate this limitation. As the fault modeling approach identifies common-mode faults in the current form and reports them as critical faults, no adaptation of the DSE tool would be necessary when integrating these capabilities. The resulting designs would "simply" contain fewer critical bits – and thus improved performance under configuration bit flips. The same applies to more sophisticated methods of redundancy or voter insertion – such as the feedback loop identification proposed by [121]: This would enhance the quality of all generated solutions while requiring no modification to the DSE approach.

LD4: The validation experiments were limited to a certain set of operating conditions (i.e., one environment temperature and one operating voltage) to keep experimental effort feasible. However, as shown in Chapter 4, both temperature and voltage variations may have a significant impact on circuit reliability under fault injection. Future evaluations should consider a broader range of operating conditions to increase confidence in the reliability of the obtained Pareto-optimal solutions. If the optimization procedure is set up to contain fault injection experiments as part of the fitness function, these experiments may be conducted under worst-case conditions (e.g., by undervolting the FPGA devices) to achieve a larger reliability margin in the field.

The key limitations of the profiling approach targeting user-defined storage element criticality described in Chapter 6 are discussed in the following.

LP1: One major limitation of the proposed profiling approach is the low correlation between the profiling-derived RVF and the criticality observed in hardware during fault injection experiments. This suggests that the profiling approach does not sufficiently capture the factors that lead to a storage element's criticality. A potential implementation issue is the way the design's intermediate representation is executed during profiling: In the proposed implementation, this is done sequentially on a simulated processor, which likely does not accurately capture the design's parallelism. By additionally consulting the schedule implemented by the HLS tool in interpreting these results, a more accurate picture of the design's temporal behavior may be derived. This could improve the assessment of the time passed between reads and writes of individual register values, thereby more accurately capturing the in-hardware RVF.

LP2: The proposed profiling implementation utilizing the design's HLL or IR form is only applicable to HLS-generated designs. However, RTL remains, as of today, still a popular design entry style. To apply the concept of the RVF to these designs, as discussed in Chapter 6, a different profiling approach more suitable to RTL designs must be found. This could, for example, be based on logic simulation on RTL or gate level. A profiling approach of this kind needs to be able to identify the conditions under which a value is written into a register (bank) or memory, and, possibly by using dataflow analysis techniques, determine if this value is used in RTL constructs or cells in subsequent clock cycles. In this way, the RVF can be determined per flip-flop or memory bit, and subsequently aggregated into register bank modules or memory blocks, also in RTL or gate-level designs.

LP3: Addressing **LP1** and **LP2** in future work should be guided by an in-depth comparison of the high-level profiling approach's performance with comparable RTL or netlist-level approaches such as [273, 254, 274]. While the data points for the case studies evaluated in this dissertation provided in Table 90 allow a basic comparison to related works, they are primarily useful for comparisons among the evaluated metrics, and indicate the potential usefulness of the *Temp/Temp2* metrics. Future work should therefore evaluate performance following the methodology of [273, 254, 274], by incrementally protecting portions of the design's flip-flops according to the metric ranking (*Load*, *LPS*, *Temp*, *Temp2*) and measuring the resulting change in fault impact.

Chapter 8

Conclusion

FPGAs are invaluable components in contemporary embedded systems due to their unique ability to flexibly implement application-specific digital circuitry. In addition to their use in pre-silicon prototyping for ASIC, they are deployed in systems that, for example, require highly customized interfaces and high parallelism while also necessitating flexibility and updateability in the field.

FPGAs achieve this by providing an array of combinational, sequential, and interconnect elements pre-fabricated in silicon. These elements are given their functionality by device configuration information. This information is stored in configuration memory cells distributed across the device, with each memory cell's output directly configuring the functionality of an associated pre-fabricated element. Current FPGA devices commonly use PROM (in the form of Antifuse-based cells), Flash EEPROM, or SRAM cells to implement these configuration memory cells.

The highest flexibility and virtually limitless reconfigurability are offered by SRAM-based devices. Here, volatile SRAM cells hold the configuration, which are loaded from an internal or external non-volatile memory at power-up. Commonly, these devices also support partially updating the device's configuration at run-time, referred to as run-time partial reconfiguration. Today's most dense, affordable, and feature-rich FPGA devices tend to use SRAM as their configuration memory technology.

Compared to Antifuse and EEPROM cells, however, volatile memory elements such as flip-flops and SRAM cells are significantly more vulnerable to environmental factors such as radiation: The value stored in a volatile memory cell can flip if it is hit by a charged particle. This is referred to as a *soft error* as its effects are removed once the memory cell is next overwritten. In FPGAs based on Antifuse and EEPROM configuration memory cells, this concerns only the circuit's state stored in user-defined flip-flops and memory blocks. In SRAM-based FPGAs, however, this additionally affects the circuit's structure via the configuration information that defines the functionality of combinational and sequential logic as well as the structure of the interconnect between these cells. Faults both in the state storage elements as well as in the configuration memory can lead to circuit malfunctions. These, in turn, can lead to severe consequences if the device is used in a safety-critical application.

To improve the applicability of SRAM-based FPGA devices in this field of applications, several fault modeling and mitigation approaches exist in today's state of the art. These approaches are explored in-depth in Chapter 3 of this work, which first motivates and formulates requirements and challenges for these approaches in Section 3.1. It further discusses current approaches towards configuration memory fault modeling (see Section 3.2) as well as cost-aware reliability improvement approaches (see Section 3.3). While today's state of the art contains multiple approaches towards modeling configuration bit flips – partly developed in parallel to the writing

of this dissertation – this analysis found identified gaps in the state of the art with respect to (1) modeling the bit-by-bit fault effects, (2) grounding in empirical fault characterization, (3) systematic bit-by-bit validation, and, especially (4) with respect to the consideration of PVT-induced inter- and intra-chip variations in fault effects. Furthermore, it identified the need for reliability improvement approaches capable of co-optimizing cost and reliability while working on the (configuration) bit level and therefore being able to take bit-level effects on reliability brought about by placement and routing changes into account. Due to the general lack of fault modeling approaches taking PVT effects into account, none of the identified state-of-the-art reliability improvement approaches consider these effects. Finally, it noted the lack of high-level analysis approaches towards the improvement of resilience against bit flips in user-defined memories that may be combined with netlist-level DSE approaches to offer a comprehensive fault mitigation solution at the configuration level, including bit flip mitigation.

To address these gaps in the state of the art, the research goals of this dissertation, as laid out in Chapter 1, call for (**G1**) precise and effective fault modeling approaches for configuration bit flips, taking special care in considering PVT effects, (**G2**) a cost-effective and automated way of introducing fault mitigation measures into existing FPGA designs in this context, and (**G3**) a similarly cost-effective and automated way of protecting digital designs against bit flips in user-defined storage elements, especially for designs generated by highly automated HLS approaches, which may be integrated into reliability-enhancement DSE approaches.

Chapter 4 addresses the above-mentioned gaps in the state of the art relevant to goal **G1** by performing in-depth fault modeling based on a proposed empirical device characterization procedure. The validity of the resulting criticality prediction procedure is substantiated by extensive validation experiments on case-study designs.

This analysis, to the author's best knowledge, is the most in-depth analysis of FPGA interconnect behavior under configuration memory bit flips, considering not only AMD FPGA technology but also that of other FPGA vendors. It is furthermore the first systematic analysis of the impact of operating conditions on the effects of fault injection results, which uncovered significant inter- and intra-device variations in the consequences of interconnect bit flips in two FPGA technologies.

The criticality prediction approach enabled by these results allows a reduction in bits to be tested using fault injection by 60% to 98% in the validation case-study designs, with minimal false negatives incurred in AMD 7 Series FPGAs. The procedures proposed in Chapter 4 contribute to the state of the art by delivering static analysis tools that allow for a more realistic assessment of a design's upper limit of critical bits compared to current approaches, while considering the often-overlooked impact of PVT effects.

Building on these fault characterization, modeling, and prediction procedures, goal **G2** is addressed in Chapter 5 of this work by proposing and validating a novel automated redundancy-inserting DSE approach to improve a given FPGA design's resilience to soft errors affecting the configuration memory. The proposed approach, based on a genetic algorithm, generates a set of cost-reliability-optimized partially redundant design variants by exploring the design space opened up by the trade-off between system cost (i.e., FPGA resource usage) and reliability (i.e., remaining number of configuration bits able to critically affect design functionality).

In its experimental validation, an implementation of the proposed approach allowed the generation of Pareto-optimal sets of design variants of six case-study designs, including two 32-bit CPUs. In all cases, the approach improved on a randomly generated starting set for all designs: Comparing the hypervolumes of the generated Pareto fronts of the final generation and a randomly generated starting generation, the approach improved the cost-effectiveness of the generated TMR solutions by 13 to 59 percentage points, delivering an attractive benefit-cost ratio.

The presented approach effectively generates a diverse set of TMR solutions across a wide cost-reliability range, allowing the designer to choose a variant that best fulfills the application's, mission's, or mission phase's cost-reliability requirements. This supplements the wealth of fault mitigation approaches available for SRAM-based FPGAs today by an automated approach. As a beneficial side effect of its built-in fault prediction and optional fault injection, this approach also provides confidence in the effectiveness of the introduced fault mitigation measures.

Working towards goal **G3**, Chapter 6 aims at applying the concept of the Register Vulnerability Factor to registers and memory blocks instantiated by HLS-generated designs to rank these storage elements according to their assumed susceptibility to soft errors: The longer a value is stored untouched in such an element, which is subject to a constant soft error rate, the more likely it becomes that it is affected by an error. As a method for deriving the RVF for each variable, profiling-based approaches at the levels of the HLL and the IR of HLS-generated designs are proposed, implemented, and evaluated.

On the one hand, this analysis approach can be used independently, for example, when using an FPGA technology whose configuration storage is not susceptible to soft errors. On the other hand, as future work, it may be combined with the DSE approach proposed in Chapter 5. In this context, it may either supply additional design-specific meta-information to the presented configuration-related reliability improvement approach for it to generate a more informed initial generation, or be used to implement a three-dimensional DSE variant that considers the fitness dimensions (1) FPGA resource cost, (2) configuration-related SEU susceptibility, and (3) SEU susceptibility of user-defined storage elements.

In an experimental evaluation on sets of benchmark designs, in which the analysis results were used to guide which portion of the user-defined storage elements to protect against bit flips, profiling variants that most closely resembled the RVF definition by taking a notion of time into account, delivered better results than simplistic approaches that, for example, merely counted accesses to storage elements. Nevertheless, no strong correlation between the profiling-derived RVF estimate and the in-hardware soft-error susceptibility derived from fault injection experiments could be established. This indicates room for improvement in the profiling approach, as implemented and evaluated, e.g., by integrating additional temporal information from the HLS-derived schedule.

The main limitations of the approaches that address goals **G1** – **G3**, and the resulting proposed directions for future work can be summarized as follows:

- In line with academic literature, much of this work focuses on AMD FPGAs. While on the one hand relevant due to the large market share of this device vendor, this is often also helped by the comparatively "open" tool support by AMD – such as the inclusion of the *EssentialBits* feature in the publicly available version of the *Vivado* implementation tool. However, this focus on AMD devices and complementary under-analysis of devices from other manufacturers bears the risk of configuration-level fault analysis and mitigation approaches developing a "blind spot": Some concepts applicable to AMD FPGAs may not be directly transferable to devices manufactured by other vendors. While this work aims to counteract this trend by analyzing two device families by Lattice Semiconductors (iCE40 and ECP5), this analysis is not as full-featured as the one provided for AMD devices. Future work should investigate the possibility of a unified soft-error analysis and mitigation approach that is as device-independent as possible. This cause could be greatly helped by FPGA device vendors by providing public bitstream documentation, which is usually kept as a trade secret. As an example for a recent development in that regard, FPGA vendor CologneChip[147] has supplied

[147] See `https://web.archive.org/web/20250831094915/https://github.com/YosysHQ/prjpeppercorn`, accessed 2026-01-05

the open source FPGA Electronic Design Automation (EDA) community with this information.

- Static criticality prediction has to perform the balancing act of providing reliable criticality information by not missing potentially critical bits (i.e, generating no false-negatives) while still not being too pessimistic to be useful for, e.g., use in a fitness function of a DSE approach. While the provided criticality prediction approach for AMD 7 Series devices does produce only a minuscule number of false negatives, it is quite pessimistic concerning other bits, i.e., it generates considerable numbers of false positives. This hampers the performance of the proposed DSE approach, which generated significantly better results when considering fault injection information – which reduced these false positives. This stems from the purely static and structure-related nature of the prediction approach: It mainly predicts the benignity or malignancy of the changes to the structure of the circuit induced by single-bit flips. Future work should extend this approach by considering dynamic information about the design's behavior. This can be done, for example, by considering simulation traces or simulating individual faults [124]. Fault injection experiments may be accelerated by drawing inspiration from conventional ATPG, which involves collapsing faults whose consequences are assumed to be identical, thereby lowering fault simulation effort.
- The approaches proposed in this work consider only single-bit flips. This decision was made since the single-bit flip model is widely accepted and used in literature, while still keeping the complexity and analysis efforts feasible. However, experimental evaluation under radiation, as discussed in Section 5.6, has shown that a non-negligible proportion of soft errors affecting the configuration are multi-bit errors. Future work should modify the proposed approaches to appropriately model the effects of multiple bit flips, which implies the possible modeling of multiple fault-induced changes to the circuit defined by the device configuration.

Together, the proposed, implemented and evaluated concepts provide important, well-validated insights into the precise causes of configuration-related FPGA (un)reliability, and thereby provide impulses for the more accurate design of future criticality prediction, fault injection, and fault consequence mitigation approaches targeting modern SRAM-based FPGAs. Furthermore, they provide concepts and validated implementations for current and future automatic fault mitigation requirements.

The advent of FPGAs based on emerging memory technologies such as Resistive RAM (RRAM) or MRAM – proposed often in literature, but still commercially out of reach – may offer new challenges in potential fault sources, potentially shifting from radiation to other possible environmental conditions, especially if multi-level cells are used. However, the modeling concepts, as well as fault characterization, prediction, and mitigation, presented in this work are still applicable to these devices. For the foreseeable future, however, SRAM-based FPGAs remain an important device category: The life cycle for AMD's 7 Series FPGAs devices, which are well-used throughout this work, has been extended by AMD to 2040. The likewise SRAM-based UltraScale+ devices will be available at least until 2045, and Versal devices will be available beyond 2045[148].

By addressing key challenges in fault analysis and mitigation, this dissertation contributes toward improving the reliability and safety of SRAM-based FPGAs, particularly in safety-critical applications where fault tolerance is essential. These contributions advance the state of the art by providing means for the design of more robust and dependable FPGA-based systems, while also considering their cost-effectiveness.

[148] See `https://web.archive.org/web/20250723193920/https://www.amd.com/en/blogs/2024/amd-supports-new-long-lifecycle-fpga-designs-thro.html`, accessed 2026-01-05

Bibliography

[1] C. Fibich, M. Horauer, and R. Obermaisser, “Device- and Temperature Dependency of Systematic Fault Injection Results in Artix-7 and iCE40 FPGAs,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1600–1605. doi: 10.23919/DATE51398.2021.9474161 .

[2] ——, “Bitstream-Level Interconnect Fault Characterization for SRAM-based FPGAs,” in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–2. doi: 10.23919/DATE56975.2023.10136911 .

[3] ——, “Characterization of Interconnect Fault Effects in SRAM-based FPGAs,” in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023, pp. 65–68. doi: 10.1109/DDECS57882.2023.10139343 .

[4] C. Fibich and L. Lovric, “Leveraging Open-Source Bitstream Documentation for FPGA Soft-Error Robustness Analysis,” in *2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2025, pp. 1–7. doi: 10.1109/IOLTS65288.2025.11116854 .

[5] C. Fibich, M. Horauer, and R. Obermaisser, “Automated Bitstream-Level Cost-Reliability Design-Space Exploration for SRAM-Based FPGAs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 45, no. 1, pp. 218–231, 2026. doi: 10.1109/TCAD.2025.3573225 .

[6] ——, “HLshield: a reliability enhancement framework for high-level synthesis,” in *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2017, pp. 1–10. doi: 10.1109/SIES.2017.7993378 .

[7] ——, “Vulnerability analysis of storage elements in HLS-generated designs using high-level profiling,” in *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, 2017, pp. 190–194. doi: 10.1109/ICSRS.2017.8272819 .

[8] ——, “Reliability-Enhanced High-Level Synthesis using Memory Profiling and Fault Injection,” in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, 2019, pp. 1363–1370. doi: 10.1109/ISIE.2019.8781158 .

[9] M. Nicolaidis, “Soft Errors in Modern Electronic Systems,” in *Frontiers in Electronic Testing*, 1st ed. New York, NY, USA: Springer, 2010, vol. 41. doi: 10.1007/978-1-4419-6993-4 .

[10] A. K. Richter and I. Arimura, “Simulation of Heavy Charged Particle Tracks Using Focused Laser Beams,” *IEEE Transactions on Nuclear Science*, vol. 34, no. 6, pp. 1234–1239, 1987. doi: 10.1109/TNS.1987.4337458 .

[11] F. Reghenzani, D. Baroffio, T. A. Lopez, W. Fornaciari, T. Borel, and F. Krimmel, “Laser Fault Injection Methodology for Software Reliability Testing,” *IEEE Reliability Magazine*, vol. 2, no. 1, pp. 52–63, 2025. doi: 10.1109/MRL.2024.3521879 .

[12] S. Gerardin and M. Bagatin, “Introduction to the Effects of Radiation on Electronic Devices,” in *Ionizing Radiation Effects in Electronics: From Memories to Imagers*, 1st ed., M. Bagatin, S. Gerardin, and K. Iniewski, Eds. Boca Raton, FL, USA: CRC Press, Taylor & Francis Group, 2018, pp. 1–22. doi: 10.1201/b19223 .

[13] B. Bhat, N. Upadhyaya, and R. Kulkarni, “Total radiation dose at geostationary orbit,” *IEEE Transactions on Nuclear Science*, vol. 52, no. 2, pp. 530–534, 2005. doi: 10.1109/TNS.2005.846881 .

[14] G. Maral, M. Bousquet, and Z. Sun, “The Space Environment,” in *Satellite Communications Systems*, 6th ed. John Wiley & Sons, Ltd, 2020, ch. 12, pp. 721–735. doi: 10.1002/9781119673811.ch12 .

[15] L. Ciani and M. Catelani, “A fault tolerant architecture to avoid the effects of Single Event Upset (SEU) in avionics applications,” *Measurement*, vol. 54, pp. 256–263, 2014. doi: 10.1016/j.measurement.2014.02.018 .

[16] G. Korkian, D. León, F. J. Franco, J. C. Fabero, M. Letiche, Y. Morilla, P. Martín-Holgado, H. Puchner, H. Mecha, and J. A. Clemente, “Single Event Upsets Under Proton, Thermal, and Fast Neutron Irradiation in Emerging Nonvolatile Memories,” *IEEE Access*, vol. 10, pp. 114 566–114 585, 2022. doi: 10.1109/ACCESS.2022.3217527 .

[17] Australian Transport Safety Bureau (ATSB), “In-Flight Upset, 154 km West of Learmonth, WA, 7 October 2008, VH-QPA, Airbus A330-303,” *Transport Safety Report AO-2008-070*, December 2011.

[18] D. Binder, E. C. Smith, and A. B. Holman, “Satellite Anomalies from Galactic Cosmic Rays,” *IEEE Transactions on Nuclear Science*, vol. 22, no. 6, pp. 2675–2680, 1975. doi: 10.1109/TNS.1975.4328188 .

[19] T. C. May and M. H. Woods, “A New Physical Mechanism for Soft Errors in Dynamic Memories,” in *16th International Reliability Physics Symposium*, 1978, pp. 33–40. doi: 10.1109/IRPS.1978.362815 .

[20] G. Agrawal, L. Massengill, and K. Gulati, “A proposed SEU tolerant dynamic random access memory (DRAM) cell,” *IEEE Transactions on Nuclear Science*, vol. 41, no. 6, pp. 2035–2042, 1994. doi: 10.1109/23.340539 .

[21] T. O’Gorman, “The effect of cosmic rays on the soft error rate of a DRAM at ground level,” *IEEE Transactions on Electron Devices*, vol. 41, no. 4, pp. 553–557, 1994. doi: 10.1109/16.278509 .

[22] M. Patel, J. S. Kim, H. Hassan, and O. Mutlu, “Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices,” in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 13–25. doi: 10.1109/DSN.2019.00017 .

[23] S. M. Guertin, J. Yang-Scharlotta, P. Blaisdell-Pijuan, and R. Some, “Options for Radiation Tolerant High-Performance Memory,” in *2017 17th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2017, pp. 1–7. doi: 10.1109/RADECS.2017.8696197 .

[24] M. Herrmann, “Radiation Effects in DRAMs,” in *Ionizing Radiation Effects in Electronics: From Memories to Imagers*, 1st ed., M. Bagatin, S. Gerardin, and K. Iniewski, Eds. Boca Raton, FL, USA: CRC Press, Taylor & Francis Group, 2018, pp. 77–96. doi: 10.1201/b19223 .

[25] P. K. Mukku and R. Lorenzo, “Double Node Upset Immune RHBD-14T SRAM Cell for Space and Satellite Applications,” *IEEE Access*, vol. 11, pp. 96 256–96 271, 2023. doi: 10.1109/ACCESS.2023.3310570 .

[26] G. Messenger, “Radiation Tolerance,” in *Nonvolatile Semiconductor Memory Technology: A Comprehensive Guide to Understanding and Using NVSM Devices*, W. D. Brown and J. Brewer, Eds. Piscataway, NJ, USA: IEEE Press, 1998, pp. 438–465. doi: 10.1109/9780470545409.ch7 .

[27] N. Battezzati, L. Sterpone, and M. Violante, “Reconfigurable field programmable gate arrays: Basic concepts,” in *Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications*. New York, NY: Springer New York, 2011, pp. 7–35. doi: 10.1007/978-1-4419-7595-9_2 .

[28] JEDEC Solid State Technology Association, “JESD89A: Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices,” October 2006.

[29] Advanced Micro Devices, Inc, “UG116 (v10.20): Device Reliability Report,” July 2025, available: https://web.archive.org/web/20250725095038/https://docs.amd.com/api/khub/maps/66EpZI5rDuOf2Bd0djXvSg/attachments/kSBWxhjG9UFKuC0QMgZCfw-66EpZI5rDuOf2Bd0djXvSg/content, accessed 2026-01-06.

[30] Xilinx, Inc., “UG470 (v1.17): 7 Series FPGAs Configuration User Guide,” December 2023, available: https://web.archive.org/web/20250912154035/https://docs.amd.com/api/khub/documents/FOs3lXmlcWxBhTIFxVKyGA/content, accessed 2026-01-06.

[31] M. Bagatin and S. Gerardin, “Radiation Effects in Flash Memories,” in *Ionizing Radiation Effects in Electronics: From Memories to Imagers*, 1st ed., M. Bagatin, S. Gerardin, and K. Iniewski, Eds. Boca Raton, FL, USA: CRC Press, Taylor & Francis Group, 2018, pp. 97–114. doi: 10.1201/b19223 .

[32] J. Prinzie, F. M. Simanjuntak, P. Leroux, and T. Prodromakis, “Low-power electronic technologies for harsh radiation environments,” *Nature Electronics*, vol. 4, no. 4, pp. 243–253, 2021. doi: 10.1038/s41928-021-00562-4 .

[33] M. J. Marinella, “Radiation Effects in Advanced and Emerging Nonvolatile Memories,” *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 546–572, 2021. doi: 10.1109/TNS.2021.3074139 .

[34] M. J. Gadlage, M. J. Kay, D. I. Bruce, A. H. Roach, A. R. Duncan, A. M. Williams, and J. D. Ingalls, “Total Ionizing Dose Effects in Commercial Floating-Gate-Alternative Non-Volatile Memories,” in *2017 IEEE Radiation Effects Data Workshop (REDW)*, 2017, pp. 1–5. doi: 10.1109/NSREC.2017.8115457 .

[35] Microchip Technology Inc., “DS00003023D: Radiation-Tolerant FPGAs,” November 2022, available: https://web.archive.org/web/20240806110803/https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/Brochures/RT-FPGA_Space+Solution+Brochures_00003023D.pdf, accessed 2026-01-26.

[36] Advanced Micro Devices, Inc, "UG908 (v2025.1): Vivado Design Suite User Guide: Programming and Debugging," May 2025, available: https://web.archive.org/web/20250725102428/https://docs.amd.com/api/khub/maps/tiXQBlmCjwnGoYCr~8JTtA/attachments/LnKVIYfW1AOKid4zIODOdQ-tiXQBlmCjwnGoYCr~8JTtA/content, accessed 2026-01-06.

[37] Infineon Technologies AG, "002-12878 Rev.*H: 64 Mb (8 MB) FL-L flash Datasheet," April 2023, available: https://web.archive.org/web/20250725104437/https://www.infineon.com/assets/row/public/documents/10/49/infineon-s25fl064l-64-mbit-8-mbyte-3-datasheet-en.pdf, accessed 2026-01-06.

[38] Avalanche Technology, "AS3xxx204/8 Space-Grade Dual QSPI Persistent SRAM Memory Datasheet (Rev C.4)," June 2025, available: https://web.archive.org/web/20250725103226/https://www.avalanche-technology.com/wp-content/uploads/64Mb-128Mb-Quad-SPI-Space-Grade-56-Ball-10_June_2025-Rev-C_4-1.pdf, accessed 2026-01-06.

[39] RAMXEED Limited, "DS501-00071-1v1-E: Memory ReRAM 12M (1536 K x 8) Bit SPI MB85AS12MT Datasheet," August 2024, available: https://web.archive.org/web/20250725105340/https://www.ramxeed.com/assets/images/products/datasheet/ReRAM/MB85AS12MT-DS1v1-E.pdf, accessed 2026-01-06.

[40] J. Kim, Y. Song, K. Cho, H. Lee, H. Yoon, and E.-Y. Chung, "STT-MRAM-Based Multicontext FPGA for Multithreading Computing Environment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1330–1343, 2022. doi: 10.1109/TCAD.2021.3091440 .

[41] T. Rizzi, A. Baroni, A. Glukhov, D. Bertozzi, C. Wenger, D. Ielmini, and C. Zambelli, "Process-Voltage-Temperature Variations Assessment in Energy-Aware Resistive RAM-Based FPGAs," *IEEE Transactions on Device and Materials Reliability*, vol. 23, no. 3, pp. 328–336, 2023. doi: 10.1109/TDMR.2023.3259015 .

[42] Y. Gonzalez-Velo, H. J. Barnaby, and M. N. Kozicki, "Review of radiation effects on ReRAM devices and technology," *Semiconductor Science and Technology*, vol. 32, no. 8, p. 083002, jul 2017. doi: 10.1088/1361-6641/aa6124 .

[43] A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 4–29, 2021. doi: 10.1109/MCAS.2021.3071607 .

[44] Altera Corporation, "SG-PRDCT-25.1: Altera® Product Catalog," March 2025, available: https://web.archive.org/web/20250927133957/https://cdrdv2-public.intel.com/730595/product-catalog.pdf, accessed 2026-01-06.

[45] Xilinx, Inc., "XMP097 (v1.3.2): Zynq-7000 SoC Product Selection Guide," June 2019, available: https://web.archive.org/web/20250927133713/https://docs.amd.com/api/khub/documents/1L_hkh2pbc5l0Oz7tcLspA/content, accessed 2026-01-06.

[46] L. Bozzoli, C. De Sio, L. Sterpone, and C. Bernardeschi, "PyXEL: An Integrated Environment for the Analysis of Fault Effects in SRAM-Based FPGA Routing," in *2018 International Symposium on Rapid System Prototyping (RSP)*, 2018, pp. 70–75. doi: 10.1109/RSP.2018.8632000 .

[47] M. Darvishi, Y. Audet, Y. Blaquière, C. Thibeault, and S. Pichette, "On the Susceptibility of SRAM-Based FPGA Routing Network to Delay Changes Induced by Ionizing Radiation," *IEEE Transactions on Nuclear Science*, vol. 66, no. 3, pp. 643–654, 2019. doi: 10.1109/TNS.2019.2898894 .

[48] D. Tummalapalli, C. Kunapareddy, V. Akalwadi, R. Govindan, and B. G, "Novel Design partitioning technique for ASIC prototyping on multi-FPGA platforms using Graph Deep Learning," in *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2022, pp. 1–4. doi: 10.1109/ICECS202256217.2022.9970882 .

[49] U. Farooq, I. Baig, M. K. Bhatti, H. Mehrez, A. Kumar, and M. Gupta, "Prototyping using multi-FPGA platform: A novel and complete flow," *Microprocessors and Microsystems*, vol. 96, p. 104751, 2023. doi: 10.1016/j.micpro.2022.104751 .

[50] H. Rudramuniyappa, S. Soujanya, P. Singh, and E. Sharma, "Optimized Algorithms for FPGA Implementation of PDCCH Chain for 5G-NR Base Station," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 7, pp. 3474–3487, 2025. doi: 10.1109/TCSI.2024.3519752 .

[51] H. Rezaei, E. Abbasi, N. Rajatheva, and M. Latva-Aho, "Unrolled, Pipelined, and Stage-Folded Architectures for Encoding of Multi-Kernel Polar Codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 11, pp. 2107–2120, 2024. doi: 10.1109/TVLSI.2024.3436872 .

[52] W. Xiang, Y. Liu, Y. Zhou, and Y. Wu, "Directional Sound-Capture System With Acoustic Array Based on FPGA," *IEEE Transactions on Instrumentation and Measurement*, vol. 73, pp. 1–12, 2024. doi: 10.1109/TIM.2023.3334344 .

[53] D.-D. Zhao, W.-B. Mao, P. Chen, Y.-J. Dang, and R.-H. Liang, "FPGA-Based Real-Time Synchronous Parallel System for Underwater Acoustic Positioning and Navigation," *IEEE Transactions on Industrial Electronics*, vol. 71, no. 3, pp. 3199–3207, 2024. doi: 10.1109/TIE.2023.3266583 .

[54] A. Yadav, V. K. Gupta, and B. Kumar, "FPGA-based Medical Image Processing Using Hardware-Software Co-design Approach," *IEEE Transactions on Biomedical Circuits and Systems*, pp. 1–12, 2025. doi: 10.1109/TBCAS.2025.3594840 .

[55] M. Jung, S. A. McKee, C. Sudarshan, C. Dropmann, C. Weis, and N. Wehn, "Driving into the memory wall: the role of memory for advanced driver assistance systems and autonomous driving," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 377–386. doi: 10.1145/3240302.3240322 .

[56] M. Li, Y. Zhang, S. He, Q. Liu, X. Zeng, C. Chen, and H. Zhu, "RT-FLOW: FPGA Implementation of Real-Time Optical-Flow-Based SLAM for High-Speed Tracking and High-Quality Mapping," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–13, 2025. doi: 10.1109/TCSI.2025.3566090 .

[57] C. Guo and W. Luk, "FPGA-Accelerated Sim-to-Real Control Policy Learning for Robotic Arms," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 3, pp. 1690–1694, 2024. doi: 10.1109/TCSII.2024.3353690 .

[58] K. Compton, S. Dasu, A. Farmahini-Farahani, S. Fayer, R. Fobes, R. Frazier, T. Gorski, G. Hall, G. Iles, J. Jones, P. Klabbers, D. Newbold, A. Perugupalli, S. Rader, A. Rose,

D. Seemuth, W. Smith, and J. Tikalsky, “The MP7 and CTP-6: multi-hundred Gbps processing boards for calorimeter trigger upgrades at CMS,” *Journal of Instrumentation*, vol. 7, no. 12, p. C12024, dec 2012. doi: 10.1088/1748-0221/7/12/C12024 .

[59] J. Wittmann, G. Aradi, B. Arnold, H. Bergauer, M. Jeitler, T. Matsushita, and C.-E. Wulz, “Design and performance of the phase I upgrade of the CMS Global Trigger,” *Journal of Instrumentation*, vol. 12, no. 1, p. C01046, jan 2017. doi: 10.1088/1748-0221/12/01/C01046 .

[60] H. F. Grip, J. Lam, D. S. Bayard, D. T. Conway, G. Singh, R. Brockers, J. H. Delaune, L. H. Matthies, C. Malpica, T. L. Brown, A. Jain, A. M. S. Martin, and G. B. Merewether, “Flight Control System for NASA’s Mars Helicopter,” in *AIAA Scitech 2019 Forum*, January 2019. doi: 10.2514/6.2019-1289 .

[61] C. González, M. Ruiz, J. Nieto, A. Carpeño, A. Piñas, V. Costa, E. Barrera, G. Arranz, W. Lee, T. Tak, and A. Zagar, “Integration of Hardware Acceleration Techniques in a Real-Time Framework Using FPGA Devices,” *IEEE Transactions on Nuclear Science*, vol. 72, no. 3, pp. 273–279, 2025. doi: 10.1109/TNS.2025.3534905 .

[62] H. Yang, T. Xue, Q. Pan, B. Liang, Y. Liu, and J. Li, “FPGA Implementation of Digital Filters for China Dark Matter Experiment,” *IEEE Transactions on Nuclear Science*, vol. 72, no. 2, pp. 189–196, 2025. doi: 10.1109/TNS.2025.3525716 .

[63] R. A. Amin, M. Hasan, V. Wiese, and R. Obermaisser, “FPGA-Based Real-Time Object Detection and Classification System Using YOLO for Edge Computing,” *IEEE Access*, vol. 12, pp. 73 268–73 278, 2024. doi: 10.1109/ACCESS.2024.3404623 .

[64] K.-L. Chiu, G. Eichler, C.-T. Lin, G. Di Guglielmo, and L. P. Carloni, “WOLT: Transparent Deployment of ML Workloads on Lightweight Many-Accelerator Architectures,” in *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, 2024, pp. 637–644. doi: 10.1109/ICCD63220.2024.00102 .

[65] F. N. Peccia, S. Pavlitska, T. Fleck, and O. Bringmann, “Efficient Edge AI: Deploying Convolutional Neural Networks on FPGA with the Gemmini Accelerator,” in *2024 27th Euromicro Conference on Digital System Design (DSD)*, 2024, pp. 418–426. doi: 10.1109/DSD64264.2024.00062 .

[66] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang, Y. Dai, J. Li, Z. Wang, R. Zhang, K. Wen, X. Ning, and Y. Wang, “FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGAs,” in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 223–234. doi: 10.1145/3626202.3637562 .

[67] M. Huang, A. Shen, K. Li, H. Peng, B. Li, Y. Su, and H. Yu, “EdgeLLM: A Highly Efficient CPU-FPGA Heterogeneous Edge Accelerator for Large Language Models,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 7, pp. 3352–3365, 2025. doi: 10.1109/TCSI.2025.3546256 .

[68] J. Boudjadar, S. U. Islam, and R. Buyya, “Dynamic FPGA reconfiguration for scalable embedded artificial intelligence (AI): A co-design methodology for convolutional neural networks (CNN) acceleration,” *Future Generation Computer Systems*, vol. 169, p. 107777, 2025. doi: 10.1016/j.future.2025.107777 .

[69] Society of Automotive Engineers (SAE), “Surface Vehicle Recommended Practice—Taxonomy and Definitions for Terms Related to Driving Automation Systems for on-Road Motor Vehicles,” 2021, Standard SAE J3016:202104.

[70] Q. Zou, Q. Sun, L. Chen, B. Nie, and Q. Li, “A Comparative Analysis of LiDAR SLAM-Based Indoor Navigation for Autonomous Vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6907–6921, 2022. doi: 10.1109/TITS.2021.3063477 .

[71] N. H. H. Aung, P. Sangwongngam, R. Jintamethasawat, S. Shah, and L. Wuttisittikulkij, “A Review of LiDAR-Based 3D Object Detection via Deep Learning Approaches Towards Robust Connected and Autonomous Vehicles,” *IEEE Transactions on Intelligent Vehicles*, vol. 10, no. 1, pp. 526–547, 2025. doi: 10.1109/TIV.2024.3415771 .

[72] D. Froidevaux and P. Sphicas, “General-Purpose Detectors for the Large Hadron Collider,” *Annual Review of Nuclear and Particle Science*, vol. 56, no. 1, pp. 375–440, 2006. doi: 10.1146/annurev.nucl.54.070103.181209 .

[73] J. Lemaire, “Exploring FPGA hardening solutions at the detector level for the future high luminosity phase of the CMS experiment at the LHC,” 2016, Master’s Thesis (CERN-THESIS-2016-344), Vrije Universiteit Brussel, Brussels Faculty of Engineering. Available: https://web.archive.org/web/20250928093350/https://inspirehep.net/files/57e76ba7e0d5b64db657fdb1c624a567, accessed 2026-01-06.

[74] B. Balaram, T. Canham, C. Duncan, H. F. Grip, W. Johnson, J. Maki, A. Quon, R. Stern, and D. Zhu, “Mars Helicopter Technology Demonstrator,” in *2018 AIAA Atmospheric Flight Mechanics Conference*, January 2018. doi: 10.2514/6.2018-0023 .

[75] J. Fugate, G. Stitt, N. V. R. Masna, A. Dasgupta, S. Bhunia, N. Dorairaj, and D. Kehlet, “An Exploration of ATPG Methods for Redacted IP and Reconfigurable Hardware,” in *2023 IEEE 41st VLSI Test Symposium (VTS)*, 2023, pp. 1–7. doi: 10.1109/VTS56346.2023.10139957 .

[76] R. Nair, “Comments on “An Optimal Algorithm for Testing Stuck-at Faults in Random Access Memories”,” *IEEE Transactions on Computers*, vol. C-28, no. 3, pp. 258–261, 1979. doi: 10.1109/TC.1979.1675331 .

[77] H. Cook and J. Goeders, “Techniques for Exploring Fine-Grained LUT and Routing Aging on a 28nm FPGA,” in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, 2024, pp. 178–186. doi: 10.1109/FPL64840.2024.00033 .

[78] M. Tehranipoor, N. Nalla Anandakumar, and F. Farahmandi, “Recycled FPGA Detection,” in *Hardware Security Training, Hands-on!* Cham: Springer International Publishing, 2023, pp. 53–72. doi: 10.1007/978-3-031-31034-8_4 .

[79] Microsemi Corporation, “TR0020 Test Report: SmartFusion2 and IGLOO2 Neutron Single Event Effects (SEE),” 2015, available: https://web.archive.org/web/20230310003626/https://www.microsemi.com/document-portal/doc_download/135249-tr0020-smartfusion2-and-igloo2-neutron-single-event-effects-see-test-report, accessed 2026-01-06.

[80] A. M. Keller, T. A. Whiting, K. B. Sawyer, and M. J. Wirthlin, “Dynamic SEU Sensitivity of Designs on Two 28-nm SRAM-Based FPGA Architectures,” *IEEE Transactions on Nuclear Science*, vol. 65, no. 1, pp. 280–287, 2018. doi: 10.1109/TNS.2017.2772288 .

[81] D. S. Lee, M. King, W. Evans, M. Cannon, A. Pérez-Celis, J. Anderson, M. Wirthlin, and W. Rice, "Single-Event Characterization of 16 nm FinFET Xilinx UltraScale+ Devices with Heavy Ion and Neutron Irradiation," in *2018 IEEE Radiation Effects Data Workshop (REDW)*, 2018, pp. 1–8. doi: 10.1109/NSREC.2018.8584313 .

[82] M. Glorieux, A. Evans, T. Lange, A.-D. In, D. Alexandrescu, C. Boatella-Polo, R. G. Alía, M. Tali, C. U. Ortega, M. Kastriotou, P. Fernández-Martínez, and V. Ferlet-Cavrois, "Single-Event Characterization of Xilinx UltraScale+® MPSOC under Standard and Ultra-High Energy Heavy-Ion Irradiation," in *2018 IEEE Radiation Effects Data Workshop (REDW)*, 2018, pp. 1–5. doi: 10.1109/NSREC.2018.8584296 .

[83] P. Maillard, Y. P. Chen, J. Arver, V. Merugu, A. Shui, and M. Voogel, "Neutron and 64MeV Proton Characterization of Xilinx 7nm VersalTM Multicore Scalar Processing System (PS)," in *2022 IEEE Radiation Effects Data Workshop (REDW) (in conjunction with 2022 NSREC)*, 2022, pp. 1–5. doi: 10.1109/REDW56037.2022.9921497 .

[84] A. M. Keller and M. J. Wirthlin, "Single-Event Characterization of a Stratix® 10 FPGA Using Neutron Irradiation," in *2019 IEEE Radiation Effects Data Workshop*, 2019, pp. 1–6. doi: 10.1109/REDW.2019.8906534 .

[85] R. Koga, S. Davis, A. Yarbrough, J. Shanney, K. Pham, C. Cao, K. Pham, and J. Dixon, "Proton Induced Single Event Effects on the Arria 10 Commercial off-the-shelf CMOS Field Programmable Gate Array," in *2022 IEEE Radiation Effects Data Workshop (REDW) (in conjunction with 2022 NSREC)*, 2022, pp. 1–5. doi: 10.1109/REDW56037.2022.9921477 .

[86] L. Koers, R. Jung, A. Scialdone, R. Ferraro, M. Karagounis, S. Danzeca, and A. Masi, "Assessing the Suitability of a 28nm European FPGA for CERN LHC Environments: An In-depth Radiation Qualification Study," in *2024 IEEE Radiation Effects Data Workshop (REDW) (in conjunction with 2024 NSREC)*, 2024, pp. 1–5. doi: 10.1109/REDW61286.2024.10759213 .

[87] R. Erdmann, J. Gebelein, and U. Kebschull, "Beamtest results of Lattice LFE2M20E FPGA," in *2013 14th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2013, pp. 1–4. doi: 10.1109/RADECS.2013.6937405 .

[88] Xilinx, Inc., "XAPP538 (v1. 0): Soft Error Mitigation Using Prioritized Essential Bits," 2012, available: https://web.archive.org/web/20200805232308/https://www.xilinx.com/support/documentation/application_notes/xapp538-soft-error-mitigation-essential-bits.pdf, accessed 2026-01-06.

[89] G. Brunetti, G. Campiti, M. Tagliente, and C. Ciminelli, "COTS Devices for Space Missions in LEO," *IEEE Access*, vol. 12, pp. 76 478–76 514, 2024. doi: 10.1109/ACCESS.2024.3405373 .

[90] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, 2005. doi: 10.1109/TDMR.2005.853449 .

[91] Microchip Technology Inc., "DS00003831F: PolarFire® FPGA Datasheet," May 2025, available: https://web.archive.org/web/20250604213938/http://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/DataSheets/PolarFire-FPGA-Datasheet-DS00003831.pdf, accessed 2026-06-01.

[92] M. Wirthlin and A. Harding, "Hybrid Configuration Scrubbing for Xilinx 7-Series FPGAs," in *FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design*, F. Kastensmidt and P. Rech, Eds. Cham: Springer International Publishing, 2016, pp. 91–101. doi: 10.1007/978-3-319-14352-1_7 .

[93] M. R. Naeemi Khaledi, M. Ebrahimi, and Z. Navabi, “Compensating Detection Latency of FPGA Scrubbers with a Collaborative Functional Hardware Duplication,” in *2021 IEEE Microelectronics Design & Test Symposium (MDTS)*, 2021, pp. 1–6. doi: 10.1109/MDTS52103.2021.9476122 .

[94] M. Wenzl, C. Fibich, P. Rössler, H. Taucher, and M. Matschnig, “Logic synthesis of assertions for saftey-critical applications,” in *2015 IEEE International Conference on Industrial Technology (ICIT)*, 2015, pp. 1581–1586. doi: 10.1109/ICIT.2015.7125322 .

[95] U. Alsaiari and F. Gebali, “Hardware Trojan Detection Using Reconfigurable Assertion Checkers,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 7, pp. 1575–1586, 2019. doi: 10.1109/TVLSI.2019.2908964 .

[96] E. Dubrova, “Dependability Evaluation Techniques,” in *Fault-Tolerant Design*. New York, NY: Springer New York, 2013, pp. 21–54. doi: 10.1007/978-1-4614-2113-9_3 .

[97] F. L. Kastensmidt, L. Carro, and R. Reis, “High-Level SEU Mitigation Techniques,” in *Fault-Tolerance Techniques for SRAM-based FPGAs*. Boston, MA: Springer US, 2006, pp. 83–90. doi: 10.1007/978-0-387-31069-5_5 .

[98] Advanced Micro Devices, Inc., “UG474 (v1.9): 7 Series FPGAs Configurable Logic Block,” April 2025, available: https://web.archive.org/web/20250912162924/https://docs.amd.com/api/khub/maps/60VkPuiXw6aa9hhsGrWlBA/attachments/daKhsPcghrMHn0rRKLtKIQ-60VkPuiXw6aa9hhsGrWlBA/content, accessed 2026-01-05.

[99] L. Sterpone, M. Reorda, and M. Violante, “RoRA: a reliability-oriented place and route algorithm for SRAM-based FPGAs,” in *Research in Microelectronics and Electronics, 2005 PhD*, vol. 1, 2005, pp. 173–176. doi: 10.1109/RME.2005.1543031 .

[100] Advanced Micro Devices, Inc., “UG473 (v1.14): 7 Series FPGAs Memory Resources User Guide,” July 2019, available: https://web.archive.org/web/20250926144852/https://docs.amd.com/api/khub/documents/9gZGbqBxtlKXxBfkBt~lAg/content, accessed 2026-01-06.

[101] ——, “UG573 (v1.13): UltraScale Architecture Memory Resources,” September 2021, available: https://web.archive.org/web/20250926145831/https://docs.amd.com/api/khub/documents/7~v2GDujU~qYXF1VUBLMAg/content, accessed 2026-01-06.

[102] Lattice Semiconductor Corp., “FPGA-TN-02204-1.6: ECP5 and ECP5-5G Memory User Guide,” November 2023, available: https://web.archive.org/web/20250926151258/https://www.latticesemi.com/-/media/LatticeSemi/Documents/ApplicationNotes/EH/FPGA-TN-02204-1-6-ECP5-and-ECP5-5G-Memory-Usage-Guide.ashx, accessed 2026-01-06.

[103] Intel Corporation, “UG-20208 (683241): Intel® Agilex™ Embedded Memory User Guide,” April 2022, available: https://web.archive.org/web/20250926150351/https://cdrdv2-public.intel.com/666819/ug-ag-memory-683241-666819.pdf, accessed 2026-01-06.

[104] International Electrotechnical Commission (IEC), “IEC 61508:2010: Functional Safety for Electrical, Electronic and Programmable Electronic (E/E/PE) Safety Related Systems,” April 2010.

[105] International Standards Organization (ISO), “ISO 26262:2018 Road vehicles – Functional safety,” December 2018.

[106] RTCA, Inc., “Design Assurance Guidance for Airborne Electronic Hardware,” April 2000.

[107] L. Mutuel, “Single event effects mitigation techniques report,” *Federal Aviation Administration, William J. Hughes Technical Center, Aviation Research Division, Atlantic City International Airport, Final report DOT/FAA/TC-15/62*, 2016.

[108] Xilinx, Inc., “PG036: Soft Error Mitigation Controller v4.1,” May 2022, available: https://web.archive.org/web/20250927095836/https://docs.amd.com/api/khub/maps/5pnNqkAzBFLBVm7WmrCFsA/attachments/5lo9N~5VeHPYt8HUVEn1HA-5pnNqkAzBFLBVm7WmrCFsA/content, accessed 2026-01-06.

[109] M. Ceschia, M. Violante, M. Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, and A. Candelori, “Identification and classification of single-event upsets in the configuration memory of SRAM-based FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2088–2094, 2003. doi: 10.1109/TNS.2003.821411 .

[110] L. Sterpone and M. Violante, “A new analytical approach to estimate the effects of SEUs in TMR architectures implemented through SRAM-based FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2217–2223, 2005. doi: 10.1109/TNS.2005.860745 .

[111] M. S. Reorda, L. Sterpone, and M. Violante, “Efficient estimation of SEU effects in SRAM-based FPGAs,” *Proceedings - 11th IEEE International On-Line Testing Symposium, IOLTS 2005*, vol. 2005, pp. 54–59, 2005. doi: 10.1109/IOLTS.2005.26 .

[112] L. Sterpone, M. Violante, R. H. Sorensen, D. Merodio, F. Sturesson, R. Weigand, and S. Mattsson, “Experimental Validation of a Tool for Predicting the Effects of Soft Errors in SRAM-Based FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2576–2583, 2007. doi: 10.1109/TNS.2007.910122 .

[113] A. Sari, D. Agiakatsikas, and M. Psarakis, “A soft error vulnerability analysis framework for Xilinx FPGAs,” in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’14. New York, NY, USA: Association for Computing Machinery, 2014, p. 237–240. doi: 10.1145/2554688.2554767 .

[114] L. Sterpone, S. Azimi, and C. De Sio, “A Framework for Uniformly Analyze and Mitigate Radiation-effects on FPGAs for Aerospace,” in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, ser. CF ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 257–262. doi: 10.1145/3587135.3592768 .

[115] E. Vacca, S. Azimi, C. D. Sio, A. Portaluri, D. Rizzieri, L. Sterpone, D. M. Codinachs, and C. Poivey, “Soft Error Reliability Prediction of SRAM-based FPGA Designs,” in *2022 22nd European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2022, pp. 1–4. doi: 10.1109/RADECS55911.2022.10412546 .

[116] M. Desogus, L. Sterpone, and D. M. Codinachs, “Validation of a tool for estimating the effects of soft-errors on modern SRAM-based FPGAs,” in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, 2014, pp. 111–115. doi: 10.1109/IOLTS.2014.6873681 .

[117] C. De Sio, S. Azimi, L. Sterpone, D. M. Codinachs, and F. Decuzzi, “PyXEL: Exploring Bitstream Analysis to Assess and Enhance the Robustness of Designs on FPGAs,” in *2023 19th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2023, pp. 1–4. doi: 10.1109/SMACD58065.2023.10192116 .

[118] L. A. Aranda, A. Sánchez-Macián, and J. A. Maestro, "ACME: A Tool to Improve Configuration Memory Fault Injection in SRAM-Based FPGAs," *IEEE Access*, vol. 7, pp. 128 153–128 161, 2019. doi: 10.1109/ACCESS.2019.2939858 .

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

[120] I. Tuzov, D. de Andrés, J.-C. Ruiz, and C. Hernández, "BAFFI: a bit-accurate fault injector for improved dependability assessment of FPGA prototypes," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6. doi: 10.23919/DATE56975.2023.10137300 .

[121] A. E. Wilson, N. Baker, E. Campbell, J. Sahleen, and M. Wirthlin, "Post-Radiation Fault Analysis of a High Reliability FPGA Linux SoC," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 123–133. doi: 10.1145/3543622.3573191 .

[122] A. E. Wilson, N. Baker, E. Campbell, and M. Wirthlin, "Improving Fault Tolerance for FPGA SoCs through Post-Radiation Design Analysis," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 17, no. 3, Sep. 2024. doi: 10.1145/3674841 .

[123] A. E. Wilson, "Enhancing Fault Tolerance in TMR Soft RISC-V FPGA SoCs through Failure-Driven Mitigation Strategies," Ph.D. dissertation, Brigham Young University, 2025, available: https://web.archive.org/web/20250909161535/https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=11901&context=etd, accessed 2026-01-06.

[124] G. Bricas, G. Tsiligiannis, J. Boch, and S. Bricas, "A new analytical approach to evaluate the radiation sensitivity of circuits implemented on SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 71, no. 10, pp. 2230–2241, 2024. doi: 10.1109/TNS.2024.3467009 .

[125] Xilinx, Inc., "DS031 (v4.0): Virtex-II Platform FPGAs: Complete Data Sheet," April 2014, available: https://web.archive.org/web/20250912152636/https://docs.amd.com/api/khub/documents/k5Q1wPM1sl2rOU2OLtgaTg/content, accessed 2026-01-06.

[126] ——, "DS001 (v2.9): Spartan-II FPGA Family Data Sheet," March 2021, available: https://web.archive.org/web/20250912152850/https://docs.amd.com/api/khub/documents/UOAB20VKtExyleq1j4~Z_g/content, accessed 2026-01-06.

[127] ——, "XCN12026 (v1.0.2): Product Discontinuation Notice: Spartan-IIE, Virtex-E, Virtex-EM, Virtex-II, EasyPath Virtex-II and XC17V00 PROMs FPGA Products," April 2013, available: https://web.archive.org/web/20250912152513/https://docs.amd.com/api/khub/documents/Sp4erNTxXQgkdXNG9j8SwA/content, accessed 2026-01-06.

[128] C. Beckhoff, D. Koch, and J. Torresen, "The Xilinx Design Language (XDL): Tutorial and use cases," in *6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*, 2011, pp. 1–8. doi: 10.1109/ReCoSoC.2011.5981545 .

[129] T. Townsend and B. Nelson, "Vivado design interface: An export/import capability for Vivado FPGA designs," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7. doi: 10.23919/FPL.2017.8056809 .

[130] Xilinx, Inc., “XCN23005 (v1.0): Product Discontinuation Notice for Selective FPGA Products,” September 2023, available: https://web.archive.org/web/20250912153032/https://docs.amd.com/api/khub/documents/tArcU~GISAL2tktrBMMJpQ/content, accessed 2026-01-06.

[131] J. D. Swift, “Root Cause Analysis and Classification of Single Point Failures in Designs Applying Triple Modular Redundancy in SRAM FPGAs,” MSc. Thesis, Brigham Young University, 2020, available: https://web.archive.org/web/20240909120410/https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?params=/context/etd/article/9766/&path_info=6672527823127920150801_etd.pdf, accessed 2026-01-06.

[132] M. Alderighi, F. Casini, S. D’Angelo, A. Gravina, V. Liberali, M. Mancini, P. Musazzi, S. Pastore, M. Sassi, and G. Sorrenti, “A preliminary study about SEU effects on programmable interconnections of SRAM-based FPGAs,” *J. Electron. Test.*, vol. 29, pp. 341–350, June 2013. doi: 10.1007/s10836-013-5387-7 .

[133] J. S. Monson, M. Wirthlin, and B. Hutchings, “A Fault Injection Analysis of Linux Operating on an FPGA-Embedded Platform,” *International Journal of Reconfigurable Computing*, vol. 2012, pp. 1–11, 2012. doi: 10.1155/2012/850487 .

[134] M. J. Cannon, A. M. Keller, H. C. Rowberry, C. A. Thurlow, A. Pérez-Celis, and M. J. Wirthlin, “Strategies for Removing Common Mode Failures From TMR Designs Deployed on SRAM FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 207–215, 2019. doi: 10.1109/TNS.2018.2877579 .

[135] H. Wei, W. Yueke, X. Kefei, and D. Wei, “SEE vulnerability bit analysis method for switch matrix of SRAM-based FPGA circuits,” in *2016 IEEE International Conference on Mechatronics and Automation*, 2016, pp. 2355–2359. doi: 10.1109/ICMA.2016.7558933 .

[136] F. Brglez and H. Fujiwara, “A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran,” in *Proceedings of IEEE Int’l Symposium Circuits and Systems (ISCAS 85)*. IEEE Press, Piscataway, N.J., 1985, pp. 677–692.

[137] F. Brglez, D. Bryan, and K. Kozminski, “Combinational profiles of sequential benchmark circuits,” in *1989 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1989, pp. 1929–1934 vol.3. doi: 10.1109/ISCAS.1989.100747 .

[138] M. Darvishi, Y. Audet, Y. Blaquière, C. Thibeault, S. Pichette, and F. Z. Tazi, “Circuit Level Modeling of Extra Combinational Delays in SRAM-Based FPGAs Due to Transient Ionizing Radiation,” *IEEE Transactions on Nuclear Science*, vol. 61, no. 6, pp. 3535–3542, 2014. doi: 10.1109/TNS.2014.2369424 .

[139] N. A. Harward, M. R. Gardiner, L. W. Hsiao, and M. J. Wirthlin, “Estimating Soft Processor Soft Error Sensitivity through Fault Injection,” in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015, pp. 143–150. doi: 10.1109/FCCM.2015.61 .

[140] S. T. Fleming and D. Thomas, “Injecting FPGA Configuration Faults in Parallel,” in *2018 International Conference on Field-Programmable Technology (FPT)*, 2018, pp. 198–205. doi: 10.1109/FPT.2018.00037 .

[141] C. Thurlow, H. Rowberry, and M. Wirthlin, “TURTLE: A Low-Cost Fault Injection Platform for SRAM-based FPGAs,” in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2019, pp. 1–8. doi: 10.1109/ReConFig48160.2019.8994782 .

[142] A. M. Keller, “Partial Circuit Replication for Masking and Detecting Soft Errors in SRAM-Based FPGAs,” Ph.D. dissertation, Brigham Young University, 2021, available: https://web.archive.org/web/20241104184932/https://scholarsarchive.byu.edu/etd/9326/, accessed 2026-01-06.

[143] A. E. Wilson, S. Larsen, C. Wilson, C. Thurlow, and M. Wirthlin, “Neutron Radiation Testing of a TMR VexRiscv Soft Processor on SRAM-Based FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 1054–1060, 2021. doi: 10.1109/TNS.2021.3068835 .

[144] A. E. Wilson and M. Wirthlin, “Fault Injection of TMR Open Source RISC-V Processors using Dynamic Partial Reconfiguration on SRAM-based FPGAs,” in *2021 IEEE Space Computing Conference (SCC)*, 2021, pp. 1–8. doi: 10.1109/SCC49971.2021.00008 .

[145] T. Rajkumar and J. Öberg, “Guided Fault Injection Strategy for Rapid Critical Bit Detection in Radiation-Prone SRAM-FPGA,” in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6. doi: 10.23919/DATE58400.2024.10546801 .

[146] Microsemi Corporation, “51700107-6/3.20: Radiation-Tolerant ProASIC3 Low Power Spaceflight Flash FPGAs with Flash*Freeze Technology Data Sheet, Rev. 6,” March 2020, available: https://web.archive.org/web/20230315075946/https://ww1.microchip.com/downloads/aemdocuments/documents/fpga/ProductDocuments/DataSheets/rtpa3_ds.pdf, accessed 2026-01-06.

[147] Microchip Technology Inc., “DS00003669D: RTG4™ FPGA Data Sheet,” 2024, available: https://web.archive.org/web/20250511060511/https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/DataSheets/RTG4_FPGA_Datasheet.pdf, accessed 2026-01-06.

[148] NanoXplore, “NG-LARGE SPACE NX1H140ATSP Datasheet Version 1.0.13,” January 2024, available: https://web.archive.org/web/20250912153502/https://files.nanoxplore.com/seafhttp/f/c380cad67b644ffdb4ae/, accessed 2026-01-06.

[149] Xilinx, Inc., “Xilinx Redefines State of the Art With New 7 Series FPGAs,” *Xcell Journal*, vol. 72, 2010, available: https://web.archive.org/web/20250113024728/https://www.xilinx.com/publications/archives/xcell/Xcell72.pdf, accessed 2026-01-06.

[150] ——, “Xilinx Ships Industry’s First 20-nm All Programmable Devices,” *Xcell Journal*, vol. 86, 2013, available: https://web.archive.org/web/20250726062457/https://www.xilinx.com/publications/archives/xcell/Xcell86.pdf, accessed 2026-01-06.

[151] ——, “Zynq UltraScale+ Says ‘Hello World’,” *Xcell Journal*, vol. 93, 2015, available: https://web.archive.org/web/20230708061541/https://www.xilinx.com/publications/archives/xcell/Xcell93.pdf, accessed 2026-01-06.

[152] L. Sterpone and M. Violante, “A new reliability-oriented place and route algorithm for SRAM-based FPGAs,” *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 732–744, 2006. doi: 10.1109/TC.2006.82 .

[153] Y. Zhang, L. Chen, S. Wang, J. Zhou, C. Tian, H. Feng, and Y. Pang, “Novel Distributed Placement Techniques Based on Routing Network for TMR Designs,” in *2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 10, 2022, pp. 489–495. doi: 10.1109/ITAIC54216.2022.9836807 .

[154] L. Bozzoli and L. Sterpone, “An Optimized Frame-Driven Routing Algorithm for Reconfigurable SRAM-Based FPGAs,” *IEEE Access*, vol. 8, pp. 116 226–116 238, 2020. doi: 10.1109/ACCESS.2020.2978632 .

[155] J. Su, J.-Y. Lee, C. Wu, and L. He, “In-place LUT polarity inversion to mitigate soft errors for FPGAs,” in *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2016, pp. 81–86. doi: 10.1109/DFT.2016.7684074 .

[156] N. Jing, J.-Y. Lee, W. He, Z. Mao, and L. He, “Mitigating FPGA interconnect soft errors by in-place LUT inversion,” in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 582–586. doi: 10.1109/ICCAD.2011.6105389 .

[157] S. Yang, *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC) Research Triangle Park, NC, USA, 1991, available: https://web.archive.org/web/20250922201244/https://ddd.fit.cvut.cz/www/prj/Benchmarks/LGSynth91.pdf, accessed 2026-01-06.

[158] X. She and N. Li, “Reducing Critical Configuration Bits via Partial TMR for SEU Mitigation in FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 64, no. 10, pp. 2626–2632, 2017. doi: 10.1109/TNS.2017.2743198 .

[159] P. Samudrala, J. Ramos, and S. Katkoori, “Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2957–2969, 2004. doi: 10.1109/TNS.2004.834955 .

[160] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, “Improving FPGA Design Robustness with Partial TMR,” in *2006 IEEE International Reliability Physics Symposium Proceedings*, 2006, pp. 226–232. doi: 10.1109/RELPHY.2006.251221 .

[161] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan, and M. Wirthlin, “Fine-Grain SEU Mitigation for FPGAs Using Partial TMR,” *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2274–2280, 2008. doi: 10.1109/TNS.2008.2000852 .

[162] S. C. Seth and V. D. Agrawal, “A new model for computation of probabilistic testability in combinational circuits,” *Integration*, vol. 7, no. 1, pp. 49–75, 1989. doi: 10.1016/0167-9260(89)90059-X .

[163] T. Marchok, A. El-Maleh, W. Maly, and J. Rajski, “Complexity of sequential ATPG,” in *Proceedings the European Design and Test Conference. ED&TC 1995*, 1995, pp. 252–261. doi: 10.1109/EDTC.1995.470387 .

[164] F. Corno, M. Reorda, and G. Squillero, “RT-level ITC’99 benchmarks and first ATPG results,” *IEEE Design & Test of Computers*, vol. 17, no. 3, pp. 44–53, 2000. doi: 10.1109/54.867894 .

[165] A. M. Keller and M. J. Wirthlin, “Partial TMR for Improving the Soft Error Reliability of SRAM-Based FPGA Designs,” *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 1023–1031, 2021. doi: 10.1109/TNS.2021.3070856 .

[166] A. J. Sánchez-Clemente, L. Entrena, and M. García-Valderas, “Partial TMR in FPGAs Using Approximate Logic Circuits,” *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2233–2240, 2016. doi: 10.1109/TNS.2016.2541700 .

[167] B. D. Sierawski, B. L. Bhuva, and L. W. Massengill, “Reducing Soft Error Rate in Logic Circuits Through Approximate Logic Functions,” *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3417–3421, 2006. doi: 10.1109/TNS.2006.884352 .

[168] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, “Selective Hardening for Neural Networks in FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2019. doi: 10.1109/TNS.2018.2884460 .

[169] T. G. Bertoa, G. Gambardella, N. J. Fraser, M. Blott, and J. McAllister, "Fault-Tolerant Neural Network Accelerators With Selective TMR," *IEEE Design & Test*, vol. 40, no. 2, pp. 67–74, 2023. doi: 10.1109/MDAT.2022.3174181 .

[170] K. L. Woo and M. R. Guthaus, "Fault-tolerant synthesis using non-uniform redundancy," in *2009 IEEE International Conference on Computer Design*, 2009, pp. 213–218. doi: 10.1109/ICCD.2009.5413153 .

[171] D. D. Thaker, F. Impens, I. L. Chuang, R. Amirtharajah, and F. T. Chong, "Recursive TMR: scaling fault tolerance in the nanoscale era," *IEEE Design & Test of Computers*, vol. 22, no. 4, pp. 298–305, 2005. doi: 10.1109/MDT.2005.93 .

[172] C. Bolchini, P. L. Lanzi, and A. Miele, "A multi-objective genetic algorithm framework for design space exploration of reliable FPGA-based systems," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8. doi: 10.1109/CEC.2010.5586376 .

[173] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002. doi: 10.1109/4235.996017 .

[174] J. Anwer, M. Platzner, and S. Meisner, "FPGA Redundancy Configurations: An Automated Design Space Exploration," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, 2014, pp. 275–280. doi: 10.1109/IPDPSW.2014.37 .

[175] I. Tuzov, D. d. Andrés, and J.-C. Ruiz, "Improving Robustness-Aware Design Space Exploration for FPGA-Based Systems," in *2020 16th European Dependable Computing Conference (EDCC)*, 2020, pp. 1–8. doi: 10.1109/EDCC51268.2020.00011 .

[176] Xilinx, Inc., "UG901 (v2020.1): Vivado Design Suite User Guide: Synthesis," June 2020, available: https://web.archive.org/web/20250912154643/https://docs.amd.com/api/khub/documents/HXRyt0BM9maBKjNBHSbUIQ/content, accessed 2026-01-06.

[177] ——, "UG835 (v2020.1): Vivado Design Suite Tcl Command Reference Guide," June 2020, available: https://web.archive.org/web/20250912154625/https://docs.amd.com/api/khub/documents/sVk_6_gQCXtNma2FQ7FaJw/content, accessed 2026-01-06.

[178] I. Tuzov, D. de Andrés, and J.-C. Ruiz, "Accurate Robustness Assessment of HDL Models Through Iterative Statistical Fault Injection," in *2018 14th European Dependable Computing Conference (EDCC)*, 2018, pp. 1–8. doi: 10.1109/EDCC.2018.00013 .

[179] J. Lojda, R. Panek, L. Sekanina, and Z. Kotasek, "Automated design and usage of the Fault-Tolerant dynamic partial reconfiguration controller for FPGAs," *Microelectronics Reliability*, vol. 144, p. 114976, 2023. doi: 10.1016/j.microrel.2023.114976 .

[180] J. Lojda, R. Panek, and Z. Kotasek, "Automatic Design of Fault-Tolerant Systems for VHDL and SRAM-based FPGAs," in *2021 24th Euromicro Conference on Digital System Design (DSD)*, 2021, pp. 549–552. doi: 10.1109/DSD53832.2021.00088 .

[181] J. Lojda, J. Podivinsky, O. Cekan, R. Panek, and Z. Kotasek, "FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 244–251. doi: 10.1109/DSD.2018.00053 .

[182] M. Straka, J. Kastil, and Z. Kotasek, “Generic partial dynamic reconfiguration controller for fault tolerant designs based on FPGA,” in *NORCHIP 2010*, 2010, pp. 1–4. doi: 10.1109/NORCHIP.2010.5669477 .

[183] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, and Yuan Xie, “Reliability-Centric High-Level Synthesis,” in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, ser. DATE ’05. USA: IEEE Computer Society, 2005, p. 1258–1263. doi: 10.1109/DATE.2005.258 .

[184] M. Glass, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich, “Reliability-Aware System Synthesis,” in *2007 Design, Automation & Test in Europe Conference & Exhibition*, 2007, pp. 1–6. doi: 10.1109/DATE.2007.364626 .

[185] S. Safari and S. Aminzadeh, “Co-evolutionary reliability-oriented high-level synthesis,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2008, pp. 2026–2029. doi: 10.1109/ISCAS.2008.4541845 .

[186] Y. Hara-Azumi and H. Tomiyama, “Cost-efficient scheduling in high-level synthesis for Soft-Error Vulnerability Mitigation,” in *Proceedings of the International Symposium on Quality Electronic Design, ISQED*, 2013, pp. 502–507. doi: 10.1109/ISQED.2013.6523658 .

[187] L. Chen and M. Tahoori, “Reliability-aware register binding for control-flow intensive designs,” in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–6. doi: 10.1145/2593069.2593200 .

[188] L. Chen, “Soft Error Analysis and Mitigation at High Abstraction Levels,” Ph.D Thesis, Karlsruher Instituts für Technologie (KIT), 2015. doi: 10.5445/IR/1000048480 .

[189] J. Tonfat, L. Tambara, A. Santos, and F. Kastensmidt, “Method to Analyze the Susceptibility of HLS Designs in SRAM-Based FPGAs Under Soft Errors,” in *Applied Reconfigurable Computing*, V. Bonato, C. Bouganis, and M. Gorgon, Eds. Cham: Springer International Publishing, 2016, pp. 132–143. doi: 10.1007/978-3-319-30481-6_11 .

[190] S. Golshan and E. Bozorgzadeh, “SEU-aware resource binding for modular redundancy based designs on FPGAs,” in *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009, pp. 1124–1129. doi: 10.1109/DATE.2009.5090832 .

[191] S. Golshan, H. Kooti, and E. Bozorgzadeh, “SEU-Aware High-Level Data Path Synthesis and Layout Generation on SRAM-Based FPGAs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 6, pp. 829–840, 2011. doi: 10.1109/TCAD.2011.2106851 .

[192] S. T. Fleming and D. B. Thomas, “StitchUp: Automatic control flow protection for high level synthesis circuits,” in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6. doi: 10.1145/2897937.2898097 .

[193] G. Lee, D. Agiakatsikas, T. Wu, E. Cetin, and O. Diessel, “TLegUp: A TMR code generation tool for SRAM-based FPGA applications using HLS,” *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 129–132, 2017. doi: 10.1109/FCCM.2017.57 .

[194] D. Agiakatsikas, G. Lee, T. Mitchell, E. Cetin, and O. Diessel, “From C to Fault-Tolerant FPGA-Based Systems,” in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 212–212. doi: 10.1109/FCCM.2018.00046 .

[195] Z. Zhao, N. T. H. Nguyen, D. Agiakatsikas, G. Lee, E. Cetin, and O. Diessel, "Fine-Grained Module-Based Error Recovery in FPGA-Based TMR Systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 1, Jan. 2018. doi: 10.1145/3173549 .

[196] Z. Zhu and B. C. Schafer, "Light-Weight Soft-Errors Detection Mechanism in High-Level Synthesis," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5. doi: 10.1109/ISCAS45731.2020.9180591 .

[197] G. L. Nazar, P. H. Kopper, M. T. Leipnitz, and B. Juurlink, "PReCEP: Automatic insertion of Partial Redundancy based on Critical Error Probability," *Microelectronics Reliability*, 126, p. 114226, 2021. doi: 10.1016/j.microrel.2021.114226 ., Proceedings of ESREF 2021, 32nd European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

[198] T. Aarrestad, V. Loncar, N. Ghielmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, K. Pedro, N. Tran, M. Liu, E. Kreinar, Z. Wu, and D. Hoang, "Fast convolutional neural networks on FPGAs with hls4ml," *Mach. Learn. Sci. Tech.*, vol. 2, no. 4, p. 045015, 2021. doi: 10.1088/2632-2153/ac0ea1 .

[199] S.-C. Chen and Y.-W. Chang, "FPGA placement and routing," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 914–921. doi: 10.1109/ICCAD.2017.8203878 .

[200] A. Bhardwaj and A. Mishra, "Accurate Estimation of Circuit Process Variations Using an On-Chip Process Monitoring Sensor," in *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET*, 2024, pp. 1–5. doi: 10.1109/ICECET61485.2024.10698716 .

[201] N. Baker, E. Campbell, A. Wilson, and M. Wirthlin, "Post-Irradiation Fault Injection for Complex FPGA Designs," *IEEE Transactions on Nuclear Science*, vol. 72, no. 8, pp. 2919–2927, 2025. doi: 10.1109/TNS.2025.3566344 .

[202] Lattice Semiconductor Corp., "FPGA-TN-02001-3.4: iCE40 Programming and Configuration – Technical Note," 2022, available: https://web.archive.org/web/20250225211158/https://www.latticesemi.com/-/media/LatticeSemi/Documents/ApplicationNotes/IK/FPGA-TN-02001-3-4-iCE40-Programming-Configuration.ashx?document_id=46502, accessed 2026-01-06.

[203] ——, "FPGA-TN-02039-2.4: ECP5 and ECP5-5G sysCONFIG User Guide – Technical Note," 2024, available: https://web.archive.org/web/20250225210730/https://www.latticesemi.com/-/media/LatticeSemi/Documents/ApplicationNotes/E---H-Folder-2/FPGA-TN-02039-2-4-ECP5-and-ECP5-5G-sysCONFIG.ashx?document_id=50462, accessed 2026-01-06.

[204] Xilinx, Inc., "UG912 (v2019.1): Vivado Design Suite Properties Reference Guide," July 2019, available: https://web.archive.org/web/20250927133430/https://docs.amd.com/api/khub/documents/GhS1PmvnNCfa2BfwKXmrqg/content, accessed 2026-01-06.

[205] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic, "Yosys+nextpnr: An Open Source Framework from Verilog to Bitstream for Commercial FPGAs," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 1–4. doi: 10.1109/FCCM.2019.00010 .

[206] E. Trumann, G. B. Thieu, J. Schmechel, K. Weide-Zaage, D. v. Wolff, A. Bausen, A. Müller, and G. Payá-Vayá, "Characterization of Soft Errors on a 28-nm SRAM-based FPGA under Neutron Radiation Exposure," in *2024 25th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*, 2024, pp. 1–5. doi: 10.1109/EuroSimE60745.2024.10491535 .

[207] D. Lewis, B. Pedersen, S. Kaptanoglu, and A. Lee, "Fracturable Lookup Table and Logic Element," Dec 2007, US Patent No. US 7,312,632 B2, Available: https://web.archive.org/web/20250928103730/https://patentimages.storage.googleapis.com/f3/17/d6/f9750a0c4569c0/US7312632.pdf, accessed 2026-01-06.

[208] Advanced Micro Devices, Inc., "UG574 (v1.6): UltraScale Architecture Configurable Logic Block," January 2025, available: https://web.archive.org/web/20250912162623/https://docs.amd.com/api/khub/maps/PMZcVknSAGtHJm8Eo9kGcA/attachments/VdxWAILnpVckWU2ZjHF78A-PMZcVknSAGtHJm8Eo9kGcA/content, accessed 2026-01-05.

[209] Lattice Semiconductor Corp., "FPGA-DS-02008-2.3: iCE40 UltraPlus™ Family Data Sheet," October 2023, https://web.archive.org/web/20240630105521/https://www.latticesemi.com/-/media/LatticeSemi/Documents/DataSheets/FPGA-DS-02008-2-3-iCE40-UltraPlus-Family-Data-Sheet.ashx?document_id=51968, accessed 2026-01-06.

[210] ——, "FPGA-DS-02012-3.3: ECP5 and ECP5-5G Family Data Sheet," January 2024, https://web.archive.org/web/20240717093249/https://www.latticesemi.com/-/media/LatticeSemi/Documents/DataSheets/ECP5/FPGA-DS-02012-3-3-ECP5-ECP5G-Family-Data-Sheet.ashx?document_id=50461, accessed 2026-01-06.

[211] S. P. Young, "Integrated Circuit Having Fast Interconnect Paths Between Memory Elements And Carry Logic," May 2007, US Patent No. US 7,218,143 B1, Available: https://web.archive.org/web/20250928103850/https://patentimages.storage.googleapis.com/7c/7c/2d/a29712da2cc2d3/US7218143.pdf, accessed 2026-01-06.

[212] Nikolić, Stefan, "Where Did the FPGAs Come from and Where Are They Headed?" in *Modern Programmable Interconnect Design*. Cham: Springer Nature Switzerland, 2025, pp. 13–44. doi: 10.1007/978-3-031-80629-2_2 .

[213] C. Chiasson and V. Betz, "Should FPGAs abandon the pass-gate?" in *2013 23rd International Conference on Field programmable Logic and Applications*, 2013, pp. 1–8. doi: 10.1109/FPL.2013.6645511 .

[214] V. Santurkar, A. K. Kandala, S. Yachareni, S. Zhou, R. Fu, P. Costello, S. Vundavalli, S. P. Young, and B. C. Gaide, "Interconnect multiplexers and methods of reducing contention currents in an interconnect multiplexer," Nov 2016, US Patent No. US 9,509,307 B1, Available: https://web.archive.org/web/20250928104002/https://patentimages.storage.googleapis.com/b9/18/ac/4239242d0ec593/US9509307.pdf, accessed 2026-01-06.

[215] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose, "The Stratix II logic and routing architecture," in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 14–20. doi: 10.1145/1046192.1046195 .

[216] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921)*, 2004, pp. 41–48. doi: 10.1109/FPT.2004.1393249 .

[217] M. I. Masud and S. J. E. Wilton, "A New Switch Block for Segmented FPGAs," in *Field Programmable Logic and Applications*, P. Lysaght, J. Irvine, and R. Hartenstein, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 274–281. doi: 10.1007/978-3-540-48302-1_28 .

[218] M. Cannon, A. Keller, and M. Wirthlin, "Improving the Effectiveness of TMR Designs on FPGAs with SEU-Aware Incremental Placement," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 141–148. doi: 10.1109/FCCM.2018.00031 .

[219] C. Wolf and M. Lasser, "Project IceStorm," https://web.archive.org/web/20251214094302/https://prjicestorm.readthedocs.io/en/latest/overview.html#what-is-project-icestorm, accessed 2026-01-06.

[220] J. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*, 2nd edition ed., ser. Prentice Hall electronics and VLSI series. Pearson Education, 2003.

[221] H. Quinn, G. R. Allen, G. M. Swift, C. W. Tseng, P. S. Graham, K. S. Morgan, and P. Ostler, "SEU-Susceptibility of Logical Constants in Xilinx FPGA Designs," *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3527–3533, 2009. doi: 10.1109/TNS.2009.2033925 .

[222] M. Wirnshofer, *Variation-aware adaptive voltage scaling for digital CMOS circuits*. Springer Science & Business Media, 2013, vol. 41.

[223] Advanced Micro Devices, Inc., "DS181 (v1.27.1): Artix 7 FPGAs Data Sheet: DC and AC Switching Characteristics," July 2024, available: https://web.archive.org/web/20250810035522/https://docs.amd.com/api/khub/documents/iAkxxTOk96ANLJqYf2hgrQ/content, accessed 2026-01-06.

[224] J. McMaster, "AMD xc7a50t Die Microscopy "xilinx_xc7a50t_s1-2_mit20x (l1)"," 2020, Available: https://siliconprawn.org/map/xilinx/xc7a50t/single/xilinx_xc7a50t_unknown_s1-2_mit20x.jpg, accessed 2026-01-06 – archived from original URL under https://web.archive.org/web/20251008181238/https://siliconprawn.org/map/xilinx/xc7a50t/single/xilinx_xc7a50t_unknown_s1-2_mit20x.jpg, accessed 2026-01-06. Licensed under CC Attribution 4.0 International, see https://creativecommons.org/licenses/by/4.0/.

[225] P. A. Bhakta, J. Plusquellic, A. Suchanek, and T. J. Mannos, "Fail-Safe Logic Design Strategies Within Modern FPGA Architectures," *IEEE Access*, vol. 13, pp. 3434–3452, 2025. doi: 10.1109/ACCESS.2024.3520877 .

[226] K. Huang, Y. Hu, and X. Li, "Reliability-Oriented Placement and Routing Algorithm for SRAM-Based FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 2, pp. 256–269, 2014. doi: 10.1109/TVLSI.2013.2239318 .

[227] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Lattuada, M. Minutoli, C. Pilato, and A. Tumeo, "Invited: Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, Dec 2021, pp. 1327–1330. doi: 10.1109/DAC18074.2021.9586110 .

[228] L. P. Deutsch and J. loup Gailly, "ZLIB Compressed Data Format Specification version 3.3," RFC 1950, May 1996. doi: 10.17487/RFC1950 .

[229] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, Hiroaki Takada, and Katsuya Ishii, "CH-Stone: A benchmark program suite for practical C-based high-level synthesis," in *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2008, pp. 1192–1195. doi: 10.1109/ISCAS.2008.4541637 .

[230] Xilinx, Inc., "UG479 (v1.10): 7 Series DSP48E1 User Guide," March 2018, available: https://web.archive.org/web/20250702103240/https://docs.amd.com/api/khub/documents/gu4oRPFEh_Pm2uaAlfY6Kg/content, accessed 2026-01-06.

[231] IEEE Standards Association, "IEEE Standard Testability Method for Embedded Core-based Integrated Circuits," *IEEE Std 1500-2022 (Revision of IEEE Std 1500-2005)*, pp. 1–168, 2022. doi: 10.1109/IEEESTD.2022.9916221 .

[232] G. Cora, C. De Sio, S. Azimi, and L. Sterpone, "Selective hardening of RISCV soft-processors for space applications," *Microelectronics Reliability*, vol. 167, p. 115667, 2025. doi: 10.1016/j.microrel.2025.115667 .

[233] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds. Boston, MA: Springer US, 1972, pp. 85–103. doi: 10.1007/978-1-4684-2001-2_9 .

[234] D. Orvosh and L. Davis, "Using a genetic algorithm to optimize problems with feasibility constraints," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 1994, pp. 548–553 vol.2. doi: 10.1109/ICEC.1994.350001 .

[235] O. Kramer, "Chapter 5: Constraints," in *Genetic Algorithm Essentials*. Cham: Springer International Publishing, 2017, pp. 39–46. doi: 10.1007/978-3-319-52156-5_5 .

[236] ——, "Chapter 3: Parameters," in *Genetic Algorithm Essentials*. Cham: Springer International Publishing, 2017, pp. 21–28. doi: 10.1007/978-3-319-52156-5_3 .

[237] L. Martí, J. García, A. Berlanga, and J. M. Molina, "A stopping criterion for multi-objective optimization evolutionary algorithms," *Information Sciences*, vol. 367-368, pp. 700–718, 2016. doi: 10.1016/j.ins.2016.07.025 .

[238] O. Kramer, "Chapter 2: Genetic algorithms," in *Genetic Algorithm Essentials*. Cham: Springer International Publishing, 2017, pp. 11–19. doi: 10.1007/978-3-319-52156-5_2 .

[239] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, " DEAP: Evolutionary Algorithms Made Easy ," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[240] Synopsys, Inc., "Synopsys FPGA Synthesis Attribute Reference Manual," in *Synplify Documentation (P-2019.03)*, March 2019.

[241] ——, "Synopsys FPGA Synthesis User Guide," in *Synplify Documentation (P-2019.03)*, March 2019.

[242] Xilinx, Inc., "UG156 (v3.1.2): Xilinx TMRTool User Guide," June 2017, available: https://web.archive.org/web/20250927094003/https://docs.amd.com/api/khub/documents/vfWaYuqNduXfSdWmQgu5QQ/content, accessed 2026-01-06.

[243] Siemens, "82960-D1 1/24 X: Precision® Hi-Rel Advanced FPGA Synthesis Technologies," 2024, available: https://web.archive.org/web/20250314161827/https://static.sw.cdn.siemens.com/siemens-disw-assets/public/k3JFgxRC2D4AExfaV7JAI/en-US/Siemens-SW-Precision-Hi-Rel-FS-82960-D1.pdf, accessed 2026-01-06.

[244] Xilinx, Inc., "UG835 (v2019.1): Vivado Design Suite Tcl Command Reference Guide," May 2019, available: https://web.archive.org/web/20250927090951/https://docs.amd.com/api/khub/documents/iehzkkbf6tiwy5N7I2dA2g/content, accessed 2026-01-06.

[245] M. W. Przewozniczek and M. M. Komarnicki, "Fitness Caching – From a Minor Mechanism to Major Consequences in Modern Evolutionary Computation," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 1785–1791. doi: 10.1109/CEC45853.2021.9504686 .

[246] Xilinx, Inc., "DS180 (v2.6.1): 7 Series FPGAs Data Sheet: Overview," September 2020, available: https://web.archive.org/web/20250707050838/https://docs.amd.com/api/khub/documents/2LByHkO~nSZXcei2D55fTg/content, accessed 2026-01-06.

[247] E. Vacca, G. Cora, S. Azimi, and L. Sterpone, "Assessment of RISC-V Processor Suitability for Satellite Applications," in *Proceedings of the 21st ACM International Conference on Computing Frontiers: Workshops and Special Sessions*, ser. CF '24 Companion. New York, NY, USA: Association for Computing Machinery, 2024, p. 116–121. doi: 10.1145/3637543.3652978 .

[248] A. M. Keller and M. J. Wirthlin, "Benefits of Complementary SEU Mitigation for the LEON3 Soft Processor on SRAM-Based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 519–528, 2017. doi: 10.1109/TNS.2016.2635028 .

[249] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009, pp. 502–506. doi: 10.1109/DATE.2009.5090716 .

[250] W. G. Cochran, "Chapter 4: The Estimation of Sample Size," in *Sampling Techniques*, 3rd ed. New York: Wiley, 1977.

[251] Y. Cao, B. J. Smucker, and T. J. Robinson, "On using the hypervolume indicator to compare Pareto fronts: Applications to multi-criteria optimal experimental design," *Journal of Statistical Planning and Inference*, vol. 160, pp. 60–74, 2015. doi: 10.1016/j.jspi.2014.12.004 .

[252] G. G. Yen and Z. He, "Performance Metric Ensemble for Multiobjective Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 1, pp. 131–144, 2014. doi: 10.1109/TEVC.2013.2240687 .

[253] N. H. Rollins and M. J. Wirthlin, "Reliability of a softcore processor in a commercial SRAM-based FPGA," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 171–174. doi: 10.1145/2145694.2145723 .

[254] A. Evans, M. Nicolaidis, S.-J. Wen, and T. Asis, "Clustering techniques and statistical fault injection for selective mitigation of SEUs in flip-flops," in *International Symposium on Quality Electronic Design (ISQED)*, 2013, pp. 727–732. doi: 10.1109/ISQED.2013.6523691 .

[255] S. Azimi, C. De Sio, A. Portaluri, D. Rizzieri, and L. Sterpone, "A comparative radiation analysis of reconfigurable memory technologies: FinFET versus bulk CMOS," *Microelectronics Reliability*, 138, p. 114733, 2022. doi: 10.1016/j.microrel.2022.114733 ., 33rd European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

[256] S. Lahti and T. D. Hämäläinen, "High-Level Synthesis for FPGAs—A Hardware Engineer's Perspective," *IEEE Access*, vol. 13, pp. 28 574–28 593, 2025. doi: 10.1109/ACCESS.2025.3540320 .

[257] Y. Liao, T. Adegbija, and R. Lysecky, "Are LLMs Any Good for High-Level Synthesis?" in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. New York, NY, USA: Association for Computing Machinery, 2025. doi: 10.1145/3676536.3699507 .

[258] C. Xiong, C. Liu, H. Li, and X. Li, "HLSPilot: LLM-based High-Level Synthesis," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. New York, NY, USA: Association for Computing Machinery, 2025. doi: 10.1145/3676536.3676781 .

[259] J. Yan and W. Zhang, "Computing and Reducing Transient Error Propagation in Registers," *Journal of Computing Science and Engineering*, vol. 5, no. 2, pp. 121–130, 2011. doi: 10.5626/JCSE.2011.5.2.121 .

[260] J. M. P. Cardoso and M. Weinhardt, "High-Level Synthesis," in *FPGAs for Software Programmers*, D. Koch, F. Hannig, and D. Ziener, Eds. Cham: Springer International Publishing, 2016, pp. 23–47. doi: 10.1007/978-3-319-26408-0_2 .

[261] L. Klemmer and D. Große, "WAVING Goodbye to Manual Waveform Analysis in HDL Design With WAL," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 3198–3211, 2024. doi: 10.1109/TCAD.2024.3387312 .

[262] J. J. Zhang, N. Bohm Agostini, S. Song, C. Tan, A. Limaye, V. Amatya, J. Manzano, M. Minutoli, V. G. Castellana, A. Tumeo, G.-Y. Wei, and D. Brooks, "Towards Automatic and Agile AI/ML Accelerator Design with End-to-End Synthesis," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2021, pp. 218–225. doi: 10.1109/ASAP52443.2021.00040 .

[263] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," *SIGPLAN Not.*, vol. 42, no. 6, pp. 89–100, Jun. 2007. doi: 10.1145/1273442.1250746 .

[264] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 72–81. doi: 10.1145/1454115.1454128 .

[265] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool," in *Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing*, 1994, pp. 66–75. doi: 10.1109/FTCS.1994.315656 .

[266] M. Azkarate-askasua, I. Martinez, X. Iturbe, and R. Obermaisser, "Dependability assessment of the time-triggered SoC prototype using FPGA fault injection," in *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, Nov 2011, pp. 2802–2807. doi: 10.1109/IECON.2011.6119756 .

[267] A. Ullah, P. Reviriego, and J. A. Maestro, "An Efficient Methodology for On-Chip SEU Injection in Flip-Flops for Xilinx FPGAs," *IEEE Transactions on Nuclear Science*, vol. 65, no. 4, pp. 989–996, 2018. doi: 10.1109/TNS.2018.2812719 .

[268] C. Fibich, P. Rössler, S. Tauner, H. Taucher, and M. Matschnig, "A netlist-level fault-injection tool for FPGAs," *e & i Elektrotechnik und Informationstechnik*, vol. 132, no. 6, pp. 274–281, Sep 2015. doi: 10.1007/s00502-015-0315-4 .

[269] C. Fibich, S. Tauner, P. Rössler, M. Horauer, M. Matschnig, and H. Taucher, "FIJI: Fault injection instrumenter," *EURASIP Journal on Embedded Systems*, vol. 2019, pp. 1–19, 2019. doi: 10.1186/s13639-019-0088-7 .

[270] D. E. Knuth, *The art of computer programming, volume 3: sorting and searching*, 2nd ed. USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[271] Xilinx, Inc., "UG116 (v10.7.1): Device Reliability Report - First Half 2017," 2017.

[272] E. Dubrova, "Information redundancy," in *Fault-Tolerant Design*. New York, NY: Springer New York, 2013, pp. 87–136. doi: 10.1007/978-1-4614-2113-9_5 .

[273] E. L. Hill, M. H. Lipasti, and K. K. Saluja, "An accurate flip-flop selection technique for reducing logic SER," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, 2008, pp. 128–136. doi: 10.1109/DSN.2008.4630081 .

[274] T. Lange, A. Balakrishnan, M. Glorieux, D. Alexandrescu, and L. Sterpone, "Machine Learning Clustering Techniques for Selective Mitigation of Critical Design Features," in *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2020, pp. 1–7. doi: 10.1109/IOLTS50870.2020.9159751 .