Cross-Layer Dependability for Runtime Reconfigurable Architectures

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Hongyan Zhang
aus Shanghai/China

Tag der mündlichen Prüfung:	11. Mai 2017
Erster Referent:	Prof. Dr.-Ing. Jörg Henkel
Zweiter Referent:	Prof. Dr. rer. nat. habil. Hans-Joachim Wunderlich

Hongyan Zhang
Steinenbergstr. 40
72764 Reutlingen

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit — einschließlich Tabellen, Karten und Abbildungen — die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Hongyan Zhang

Contents

Acknowledgments v

List of Own Publications vii

List of Figures ix

List of Tables xi

Acronyms xv

Kurzfassung xvii

Abstract xxi

1 Introduction and Motivation 1
- 1.1 Dependability Challenges in the Nano-CMOS Era 2
  - 1.1.1 Challenges from Up-Scaling 3
  - 1.1.2 Challenges from Down-Scaling 4
  - 1.1.3 Addressing the Challenges 6
- 1.2 Thesis Contributions 7
- 1.3 DFG Research Program SPP-1500 and InvasIC 8
- 1.4 Thesis Outline 9

2 Backgrounds 11
- 2.1 Field-Programmable Gate Arrays 11
  - 2.1.1 The Reconfigurable Fabric 11
  - 2.1.2 Configurable Logic Blocks 12
  - 2.1.3 Transistor-Level LUT Model 14
  - 2.1.4 Programmable Switching Matrices 14
  - 2.1.5 Configuration Memory 15
  - 2.1.6 Partial Reconfiguration 16
- 2.2 Fine-Grained Reconfigurable Architectures 17
  - 2.2.1 Filling the Gap between GPP and ASIC 17
  - 2.2.2 Coupling of a Reconfigurable Fabric to a GPP 18
  - 2.2.3 Hardware Acceleration of Application Kernels 20
- 2.3 Dependability Issues in CMOS Circuits 21
  - 2.3.1 Basic Operation Principles of MOSFET 23
  - 2.3.2 Biased Temperature Instability 24
  - 2.3.3 Hot Carrier Injection 25
  - 2.3.4 Single Event Upset 26
  - 2.3.5 Recent Advancement in Aging 27

2.4 Fault, Stress and Aging Models Used in the Thesis 28
2.4.1 Stress Model for Aging Effects 28
2.4.2 Stress Properties 30
2.4.3 Aging Models 30
2.4.4 Fault Model for Soft Errors 31
2.5 Basic Dependability Techniques 32
2.5.1 FPGA Test and Diagnosis 33
2.5.2 Concurrent Error Detection in FPGAs 36
2.5.3 Scrubbing of Configuration Memory 39
2.6 Related Work 40
2.6.1 FPGA-Based Reconfigurable Architectures 40
2.6.2 Online Test and Diagnosis of Reconfigurable Systems 41
2.6.3 Fault Tolerance in Reconfigurable Systems 42
2.6.4 Aging Mitigation in Reconfigurable Systems 43
2.6.5 Handling Soft-Errors in the Configuration Memory 44

3 System Overview and Cross-Layer Dependability 47
3.1 Application Model 47
3.2 Target Architecture 48
3.2.1 Base Architecture 48
3.2.2 Architectural Extension 49
3.3 Architectural Assumptions 51
3.4 Cross-Layer Dependability 51
3.4.1 Lifetime Increase 53
3.4.2 Fault Discovery 54
3.4.3 Self-Repair 54
3.4.4 Reliability Guarantee 55
3.4.5 Runtime Orchestration 55
3.5 Evaluation Platform 56

4 Fault Discovery through Strategic Online Testing 59
4.1 Overview of Online Test Strategies 60
4.2 Integration of Online Tests 61
4.3 Scheduling of Online Tests 63
4.3.1 PRET Scheduling 63
4.3.2 PORT Scheduling 63
4.4 Experimental Evaluation 65
4.4.1 Fault Models of Tests 65
4.4.2 Test Configurations for PRET 66
4.4.3 PRET Scheduling 67
4.4.4 PORT Scheduling 69
4.4.5 Combined PRET and PORT Scheduling 70

5 Self-Repair by Module Diversification 73
5.1 Overview of the Module Diversification Method 73
5.2 Diversified Configurations 74
5.2.1 Matrix Representation of Configurations 74
5.2.2 Properties of Diversified Configurations 74

5.3 Generation Algorithm 75
5.4 Reliability Analysis 77
5.5 Diversification for Interconnect Resources 78
5.6 Implementation Flow 78
5.7 Experimental Evaluation 79
5.7.1 Timing Overhead 79
5.7.2 Reliability Improvement 80

6 Prolonging Lifetime via Stress Balancing 83
6.1 Overview of the Stress-Aware Placement Method 83
6.2 Representation of Stress 85
6.2.1 Stress Granularity 85
6.2.2 Stress Accumulation 86
6.2.3 Stress Estimation Flow 86
6.3 Runtime Accelerator Placement 88
6.3.1 Placement Profit 89
6.3.2 Placement Algorithm 89
6.3.3 Intermediate Results 90
6.4 Synthesis Time Logic Placement 90
6.4.1 Placement Algorithm 91
6.4.2 Stress Distribution Results 93
6.5 Extended Runtime Accelerator Placement with Module Diversification 93
6.6 Experimental Evaluation 94
6.6.1 Evaluation Flow 94
6.6.2 Timing Overhead 96
6.6.3 Stress Reduction and MTTF Improvement 96

7 Reliability Guarantee with Adaptive Modular Redundancy 99
7.1 Overview of Adaptive Modular Redundancy 100
7.2 Reliability of Accelerated Functions 102
7.3 Reliability Guarantee of Accelerated Functions 103
7.3.1 Maximum Resident Time 104
7.3.2 Acceleration Variants Selection 105
7.3.3 Non-uniform Accelerator Scrubbing 107
7.4 Reliability Guarantee of Applications 108
7.4.1 Effective Critical Bits of Accelerators 109
7.4.2 Reliability of Accelerated Kernels 109
7.4.3 Effective Critical Bits of Accelerated Kernels and Applications 111
7.4.4 Budgeting of Effective Critical Bits 112
7.5 Experimental Evaluation 114
7.5.1 Performance Improvement 115
7.5.2 Runtime Overhead 116

8 Overall Evaluation and Comparison 117
8.1 Structural Integrity 117
8.1.1 Accelerator Diversification 119
8.1.2 Aging Resilience and Fault Tolerance 119
8.2 Functional Correctness 124

9 Conclusion and Future Work **127**
9.1 Thesis Conclusion 127
9.2 Future Work 129

A Proof of the Minimal Set Generation in Module Diversification . . . **131**

B Terrestrial Soft Error Rates in a Virtex-5 FPGA **137**

Bibliography **139**

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my Ph.D. advisor Prof. Dr.-Ing. Jörg Henkel who not only guided and supported me in the research work but also has had a significant impact on my personal development. By asking the right questions, casting insightful doubts and encouraging me to formulate clear and precise answers, he showed me the proper way of thinking for scientific problems, which was absolutely essential to the accomplishment of this work and any other work in my future carrier path.

My sincere thanks also goes to Prof. Dr. rer. nat. habil. Hans-Joachim Wunderlich from the University of Stuttgart for agreeing to be my second reviewer and for his support of the OTERA project through discussion of proposals and reviewing our joint papers.

Next, I would like to thank Dr.-Ing. Lars Bauer who provided precious support and lead at key times. He helped me in thinking through some of the difficult ideas. This work was greatly influenced by the intriguing discussions with him and his suggestions in writing styles. The evaluation platform used in this work is based on his great achievements during his Ph.D.

My special thanks go to the project partners from the University of Stuttgart: Dr. rer. nat. Michael Kochte for his fruitful discussions and guiding suggestions during the writing of OTERA papers, Eric Schneider for his implementation of stress models for lookup tables, Dr. rer. nat. Claus Braun and Michael Imhof for their excellent collaboration in our first OTERA papers. Their influence is of great importance to the success of this work and it has always been a great pleasure to work with them.

I would also like to thank my colleagues at CES who provided me a comfortable and trustful work environment: Artjom Grudnitsky for the collaboration on the demonstrator and help on the the setup of the RotMan simulator; Dr.-Ing. Hussam Amrouch and Victor van Santen for their support on the aging models; Martin Buchty for his technical support and help on the lab supervision; Usman Karim and Anuj Pathania for their fun talks in the office; Semeen Rehman and Florian Kriebel for their company during the trip to SPP meetings; Marvin Damschen, Thomas Ebi, Fazal Hameed, Chih-Ming Hsieh, Anton Ivanov, Heba Khdr, Sebastian Kobbe, Farzad Samie, Sammer Srouji, Manyi Wang and Volker Wenzel for their support, friendship and advice during my stay at CES.

This work was supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500), where I was given the opportunity to examine my work in a wider scope.

Finally, I would like to express my deepest gratitude to my parents for their constant support, encouragement and understanding, and to my wife Xin Liu for sustaining me through challenging times and enriching my life with light and meaning. I could have never have done this work without you.

List of Own Publications

Publications Providing Major Contributions to This Thesis

[ZBK+17] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, H. J. Wunderlich, and J. Henkel, "Aging Resilience and Fault Tolerance in Runtime Reconfigurable Architectures," *IEEE Transactions on Computers, Special Section on Innovation in Reconfigurable Computing Fabrics: from Devices to Architectures*, 2017 (to appear).

[ZBH16] H. Zhang, L. Bauer, and J. Henkel, "Resource budgeting for reliability in reconfigurable architectures," in *Proc. 53rd Annual Design Automation Conference (DAC)*, 2016, pp. 111:1–111:6.

[ZKS+15] H. Zhang, M. A. Kochte, E. Schneider, L. Bauer, H.-J. Wunderlich, and J. Henkel, "STRAP: Stress-aware placement for aging mitigation in runtime reconfigurable architectures," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 38–45.

[ZKI+14] H. Zhang, M. A. Kochte, M. E. Imhof, L. Bauer, H.-J. Wunderlich, and J. Henkel, "GUARD: GUAranteed reliability in dynamically reconfigurable systems," in *Proc. 51st Annual Design Automation Conference (DAC)*, 2014, pp. 32:1–32:6.

[ZBK+13] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, C. Braun, M. E. Imhof, H.-J. Wunderlich, and J. Henkel, "Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures," in *Proc. IEEE International Test Conference (ITC)*, 2013, pp. 1–10.

[BBI+13] L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, E. Schneider, H. Zhang, J. Henkel, and H.-J. Wunderlich, "Test Strategies for Reliable Runtime Reconfigurable Architectures," *IEEE Transactions on Computers*, vol. 62, no. 8, pp. 1494–1507, 2013.

Publications Providing Minor Contributions to This Thesis

[BHH+15] L. Bauer, J. Henkel, A. Herkersdorf, M. A. Kochte, J. M. Kühn, W. Rosenstiel, T. Schweizer, S. Wallentowitz, V. Wenzel, T. Wild, H.-J. Wunderlich, and H. Zhang, "Adaptive multi-layer techniques for increased system dependability," *it - Information Technology*, vol. 57, no. 3, 2015.

[HBGZ14] J. Henkel, L. Bauer, A. Grudnitsky, and H. Zhang, "Adaptive embedded computing with i-Core," *ACM SIGBED Review*, vol. 11, no. 3, pp. 20–21, 2014.

[HBZ+14] J. Henkel, L. Bauer, H. Zhang, S. Rehman, and M. Shafique, "Multi-layer dependability: From microarchitecture to application level," in *Proc. 51st Annual Design Automation Conference (DAC)*, 2014, pp. 1–6.

[ABB+12] M. S. Abdelfattah, L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, H. Zhang, J. Henkel, and H.-J. Wunderlich, "Transparent structural online test for reconfigurable systems," in *Proc. IEEE 18th International On-Line Testing Symposium (IOLTS)*, 2012, pp. 37–42.

[BBI+12] L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, H. Zhang, H.-J. Wunderlich, and J. Henkel, "OTERA: Online test strategies for reliable reconfigurable architectures — Invited paper for the AHS-2012 special session "Dependability by reconfigurable hardware"," in *Proc. NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 38–45.

List of Figures

1.1 Illustrative example of a reconfigurable architecture 1

2.1 Island-style FPGA architecture 12
2.2 Xilinx-style CLB structure [Xil12d] 13
2.3 Reconfigurable resources in a slice 13
2.4 Internal structure of a 2-input LUT 14
2.5 Internal structure of a PSM and a PIP [HCJ+90] 15
2.6 Configuration frames in a Xilinx Virtex-5 FPGA 16
2.7 Partial reconfiguration with partial bitstreams 17
2.8 Three degrees of coupling between FPGA and GPP: (a) loosely coupled, (b) co-processor and (c) tightly coupled 19
2.9 Basic structure of a NMOS transistor 23
2.10 Physical mechanism of BTI in NMOS 24
2.11 Physical mechanism of HCI in NMOS 25
2.12 Physical mechanism of SEU in MOSFET [Bau05] 26
2.13 Threshold voltage increases over time due to HCI under different toggling rates. 29
2.14 Threshold voltage increases over time due to BTI under different stress duty cycles. 29
2.15 A general test structure for digital circuits 33
2.16 Array-based test structure for FPGAs 35
2.17 Array-based test process for FPGAs 36
2.18 Triple modular redundancy 37
2.19 Duplication with comparison 38

3.1 The application model used in this thesis 47
3.2 Target reconfigurable architecture 48
3.3 System layers and their interaction with the dependability approaches proposed in this thesis 52
3.4 Runtime orchestration of dependability techniques 55

4.1 Test flow with PRET and PORT 60
4.2 Test manager integration with TPG and ORA 61
4.3 Performance loss of the video encoder application under different on-demand PRET frequencies and number of regions 67
4.4 Average test latency under different PRET frequencies and number of regions 68
4.5 Comparison of the number of on-demand and periodic tests for different on-demand PRET frequencies and number of regions 68
4.6 Performance loss when both PRET and PORT are applied for a reconfigurable system with 5 regions 70

5.1 Generation of diversified configurations using the module diversification method 78
5.2 Module reliability of `apex4` for different ratios of CLB redundancy and number of configurations with CLB reliability 0.999 81
5.3 Module reliability *with* and *without* module diversification for different CLB reliabilities. Reliabilities of `des_perf` and `aes_core` are not shown in the figure for clarity, but discussed in the text. 81
5.4 Reliability improvement factor for the modules when module diversification is applied 82

6.1 Transistor stress distribution in a reconfigurable fabric with 8 regions; each region consists of 4×20 CLBs with 8 LUTs each (same setup as for evaluation); the color of a CLB corresponds to the highest toggle rate of any of its transistors; the symbol ► on the right scale denotes the maximum stress over all regions 84
6.2 Overview of the stress-aware placement method 85
6.3 Stress estimation flow 87
6.4 Toggle propagation (a) and generation (b) in multiplexers 87
6.5 Transistor stress distribution using stress-aware runtime accelerator placement 91
6.6 Transistor stress distribution using both stress-aware runtime accelerator placement and synthesis time stress diversification 93
6.7 Experimental flow to evaluate the transistor stress and threshold voltage shift 95
6.8 Comparison to related work for dynamic stress in systems with different number of reconfigurable regions 97
6.9 Comparison to related work for static stress in systems with different number of reconfigurable regions 97
6.10 Transistor stress for different STRAP optimization goals 98

7.1 Different hardware implementation variants of an Accelerated Function (AF) 100
7.2 Overview of proposed adaptive modular redundancy 101
7.3 Variants selection space for an error rate of 10 errors $\text{Mb}^{-1}\text{month}^{-1}$. 105
7.4 Execution of kernels with different degrees of redundancy 108
7.5 Illustrative execution series of an accelerated function 110
7.6 Ratios of error probability and performance improvement under different numbers of regions and reliability requirements 115
7.7 Ratios of error probability and performance improvement under different soft-error rates and reliability requirements 116

8.1 Application performance in the presence of faults under different strategies. Left Y-axis (box plots): performance degradation w.r.t. a fault-free baseline system. Right Y-axis (line plots): performance gain w.r.t. to the faulty baseline system 121

8.2 Peak stress and utilization in the reconfigurable fabric in the presence of faults. Left Y-axis (box plots): maximum transistor toggle rate. Right Y-axis (line plots): utilization of the reconfigurable fabric for acceleration w.r.t. a fault-free baseline system 123
8.3 Performance under varying soft error rate 125
8.4 Average AF error probability for different fault tolerance methods under varying soft error rate 125
8.5 Performance degradation over a wide range of soft error rates and reliability constraints 126

B.1 This figure shows how the soft error rate varys depending on the altitude. The neutron flux is lower at low altitude regions due to atmospheric shielding. It is almost 30 times higher at mountain peaks in the US than it is at sea level. 137
B.2 The neutron flux also varys in the geomagnetic field. This figure shows the resulted variation of soft error rate around the globe at 10 km altitude, where commercial flights typically cruise. In the equator regions, the soft error rate is roughly 6 times lower than it is in other regions. 138

List of Tables

3.1 Short description of accelerators implemented for H.264 57

4.1 Test configurations for CLBs and interconnects: Overhead, size, frequency and length 66
4.2 PORT performance loss and worst case test latency under different PORT frequencies 69

5.1 Configurations for different region sizes and maximal frequency of original (Orig.) and diversified (Div.) modules 80

6.1 Change in maximum frequency of accelerators 96
6.2 Reduction of avg./max. stress and MTTF increase of STRAP and state-of-the-art [AZGT11, ZBK+13] compared to the baseline; averaged over all numbers of reconfigurable regions 98

8.1 Properties of reconfigurable accelerators and their change in maximum frequency of diversified configurations 118
8.2 Compared strategies in the experiments 118
8.3 Reduction of maximum transistor toggle rate w.r.t. the baseline system [%] 119
8.4 MTTF improvement w.r.t. the baseline system [×] (e.g. 2× improvement means the MTTF is doubled) 120
8.5 Performance and error probability results 126

Acronyms

AC	Accelerator Configuration.
AF	Accelerated Function.
ASIP	Application-Specific Instruction set Processor.
BIST	Built-In-Self-Test.
BTI	Bias Temperature Instability.
CED	Concurrent Error Detection.
CGRA	Coarse-Grained Reconfigurable Architectures/Arrays.
CLB	Configurable Logic Block.
CPU	Central Processing Unit.
CUT	Circuit Under Test.
DPPM	Defective Parts Per Million.
DSP	Digital Signal Processor.
DWC	Duplication with Comparison.
ECC	Error Correcting Code.
EM	Electro-Migration.
FPGA	Field-Programmable Gate Array.
FSM	Finite State Machine.
GPU	Graphic Processing Unit.
HCI	Hot Carrier Injection.
ICAP	Internal Configuration Access Port.
ILA	Iterative Logic Array.
LUT	Look-Up Table.
MBE	Multi-Bit Error.
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor.
MTTF	Mean Time to Failure.
NMOS	n-type MOSFET.
ORA	Output Response Analyzer.
OTERA	Online Test Strategies for Reliable Reconfigurable Architectures.
PIP	Programmable Interconnection Point.
PMOS	p-type MOSFET.

PORT Post-configuration Test.
PRET Pre-configuration Test.
PSM Programmable Switching Matrix.

RAM Random Access Memory.
RDP Random Dopant Fluctuation.
RIF Reliability Improvement Factor.

SBE Single-Bit Error.
SER Soft Error Rate.
SEU Single Event Upset.
SoC System-on-Chip.
STRAP STRess-Aware Placement.

TC Test Configuration.
TDDB Time-Dependent Dielectric Breakdown.
TMR Triple Modular Redundancy.
TPG Test Pattern Generator.

Kurzfassung

Rekonfigurierbare Rechensysteme kombinieren die Effizienz der Hardware mit der Flexibilität der Software und werden konkurrenzfähig gegen herkömmliche Rechnerarchitekturen wie Allzweckprozessoren (CPUs) und Grafikprozessoren (GPUs). Sie bieten eine Hardwareorganisation, die noch nach der Herstellung durch Benutzer für verschiedene Anwendungen dynamisch angepasst werden kann. Rekonfigurierbare Architekturen, die auf Feld-Programmierbaren Gatter-Anordnungen (FPGAs) basieren, treten in den letzten Jahren dank stets wachsender Kapazität der FPGAs und einem vereinfachten FPGA-Entwurfsprozesses als eine vielversprechende Technologie für rekonfigurierbare Rechensysteme auf.

Eine typische rekonfigurierbare Architektur besteht aus einem Allzweckprozessor, einer rekonfigurierbaren Struktur und einer Kommunikationsinfrastruktur, die beide miteinander verbindet. Die rekonfigurierbare Struktur kann sich auf einem oder mehreren FPGAs befinden. Hardware-Beschleuniger, die rechenintensive Funktionen realisieren, können zur Laufzeit in der rekonfigurierbaren Struktur instanziiert werden. Die Arbeitslast auf dem Prozessor kann dann auf den FPGA ausgelagert werden, der das Rechnen in Hardware mit hoher Leistung und Energieeffizienz durchführt. Die stetige Weiterentwicklung in der Halbleiterindustrie ermöglicht die nachhaltige Verkleinerung der Transistorgröße in den Nanobereich, was zu höherer Transistordichte, schnellerer Schaltgeschwindigkeit und niedrigerem Energieverbrauch führt. Um diese Vorteile der Skalierung der Transistorgröße auszunutzen, werden moderne FPGAs in neusten Technologien hergestellt. Zum Beispiel wird der jüngste MPSoC von Xilinx mit vier ARM-Kernen und einer rekonfigurierbaren Struktur auf einem einzigen Chip mit 16 nm FinFET Technologie hergestellt.

Allerdings kommt die Skalierung mit Herausforderungen, die den zuverlässigen Betrieb von FPGAs bedrohen. Fehler können beim Herstellungstest unbemerkt bleiben und in ausgelieferten Geräten latent vorhanden sein, da die wachsende Schaltungskomplexität und neue Defektmechanismen die Wirksamkeit der Tests begrenzen. Während des Betriebs der Schaltungen verschlechtern verschiedene mikroskopische Phänomene die physikalischen und elektrischen Eigenschaften der Materialien, aus denen die Transistoren bestehen. Solche Effekte werden durch weitere Skalierung verschlimmert. Die Transistoren funktionieren nicht für ewig sondern altern. Die Schwellwertspannung der Transistoren verschiebt sich mit der Zeit, was die Stromstärke durch die Transistoren verringert und schließlich zu einem kompletten Ausfall führen kann. Darüber hinaus wechselwirkt die Hintergrundstrahlung aus der Umgebung mit den Materialien des Chips. Die Konfigurationsbits in einem FPGA können dadurch verändert werden, was im Wesentlichen die Funktionsdefinition der auf dem FPGA implementierten Schaltungen modifiziert. Die Rechenergebnisse vom FPGA sind dann fehlerhaft.

In einem zuverlässigen System, insbesondere bei sicherheits- und missionskritischen Anwendungen, sollen Fehler in kürzester Zeit erkannt, lokalisiert und vermieden werden, um ihre Einflüsse auf das Gesamtsystem zu minimieren. Außerdem sollen Gegenmaßnahmen proaktiv eingeleitet werden, um das Eintreten von Fehler überhaupt zu vermeiden. Folgende Schlüsseltechniken sind im Rahmen dieser Arbeit entwickelt worden, um eine hochzuverlässige rekonfigurierbare Architektur zu verwirklichen:

- Für das Erkennen von Fehler in der rekonfigurierbaren Struktur werden bedarfsgesteuerte und periodische Tests während des funktionalen Systembetriebs eingesetzt. Vor der Instanziierung von Beschleunigern werden die grundlegenden rekonfigurierbaren Ressourcen gründlich durch strukturelle Tests geprüft. Nach der Konfiguration der Beschleuniger wird ihre Funktionalität periodisch durch funktionelle Tests geprüft. Die Kombination von beiden Teststrategien erzielt eine hohe Fehlerbedeckung und niedrige Testlatenz bei minimalem Leistungsaufwand. Experimentelle Ergebnisse zeigen, dass die rekonfigurierbare Struktur alle 4 Sekunde bei weniger als 4,4% Leistungsaufwand gründlich getestet werden kann.
- Wenn Teile der rekonfigurierbaren Struktur als fehlerhaft erkannt werden, können die fehlerhaften Ressourcen für die Berechnung vermieden werden ohne die Systemleistung zu verändern. Dies wird durch eine neuartige Entwurfsmethode namens Modul-Diversifikation erreicht. Für jeden Beschleuniger oder Modul wird eine Menge an Konfigurationen generiert, die bezüglich der Ressourcenbenutzung diversifiziert sind. Alternative Konfigurationen, die die fehlerhaften Ressourcen nicht benötigen, können benutzt werden, um den normalen Systembetrieb sogar in Gegenwart von Fehlern aufrechtzuhalten. Zuverlässigkeitsverbesserungsfaktoren zwischen 19 und 330 wurden in den Experimenten erreicht.
- Die Verwendung der rekonfigurierbaren Struktur erzeugt elektrischen Stress für die einzelnen Transistoren, der zur Alterung der Transistoren führt. In dieser Arbeit wurde eine neuartige Technik zur Stressesbalancierung entwickelt. Sie ist in der Lage, den durch Arbeitslast erzeugten Stress gleichmäßig auf alle rekonfigurierbaren Ressourcen zu verteilen, so dass der Stress auf einzelne Transistoren minimiert wird. Die Kombination aus Beschleuniger-Platzierung zur Laufzeit und Logik-Platzierung zur Synthese-Zeit bringt eine gleichmäßige Stressverteilung, was die Lebensdauer des Systems bei geringfügigen Laufzeitkosten erheblich verlängert. Im Vergleich zum Stand-der-Technik werden die dynamischen und statischen Stresse jeweils um bis zu 64% und 35% reduziert, was einer Verländerung der Lebensdauer jeweils um bis zu 177% und 15% entspricht.
- Maßnahmen wie modulare Redundanz sind unerlässlich, um die korrekte Berechnung von Beschleunigern vor zufälligen Bitfehlern im Konfigurationsspeicher (aufgrund der Umgebungsstrahlung) zu schützen. Eine Lösung, die auf den schlimmsten Fall ausgerichtet ist, hat hohe Flächen- und Energiekosten durch den übertriebenen Schutzes gegen eine sich verändernde Umgebung. Diese Arbeit stellt ein neuartiges Laufzeitsystem vor, das dynamisch den effi-

zientesten Schutzmechanismus für unterschiedliche Beschleuniger bestimmen kann, abhängig von der Anfälligkeit der Beschleuniger, Zuverlässigkeitsrandbedingung der Anwendung und der Strahlungsstärke in der Umgebung. Im Vergleich zu einer verwandten Arbeit, die die Schutzmechanismen statisch festlegt, bietet die vorgestellte Methode bis zu 68% Leistungssteigerung bei derselben Zielzuverlässigkeit an.

Durch die Ausnutzung der inhärenten Flexibilität rekonfigurierbarer Architekturen erarbeitet diese Arbeit eine umfassende Lösung für Fehlerentdeckung, Fehlertoleranz und Verlangsamung der Alterung, die die Zuverlässigkeit des Systems gegen permanente und transiente Fehler verteidigt.

Abstract

Reconfigurable computing combines the efficiency of hardware with the flexibility of software and is becoming competitive against conventional processor architectures like Central Processing Units (CPUs) and Graphic Processing Units (GPUs). It features a hardware organization that can be dynamically customized after manufacturing by the user for different application requirements. Reconfigurable architectures based on Field-Programmable Gate Arrays (FPGAs) are emerging over the recent years as a promising technology for reconfigurable computing, thanks to the growing logic capacity of FPGAs and the ease of FPGA design process driven by FPGA vendors.

A typical reconfigurable architecture consists of a general purpose processor, a reconfigurable fabric and a communication infrastructure interconnecting them. The reconfigurable fabric can reside on one or multiple FPGAs. Hardware accelerators, which implement the compute-intensive functions, can be instantiated in the reconfigurable fabric during runtime. The workload on the processor can then be offloaded to the FPGA which performs high performance and high energy efficient computation in hardware.

Continued advancement in the semiconductor industry enables sustained down-scaling of transistor size into the nano-scale regime, which translates to higher transistor density, faster switching speed and lower energy consumption. To take these advantages of technology scaling, modern FPGA devices are manufactured in latest technology nodes. For instance, the newest MPSoC from Xilinx with four ARM cores and a reconfigurable fabric on a single die is built on 16 nm FinFET process technology.

However, the technology scaling is accompanied by challenges that threaten the dependable operation of FPGAs. Faults may escape the manufacturing test and remain latent in the shipped devices, as the growing complexity of circuits and new defect mechanisms limit the test effectiveness. During circuit operation, various microscopic phenomena degrade the physical and electrical properties of the materials that compose the transistors, which are further aggravated with down-scaling. The transistors do not operate forever, but they ages. The threshold voltage of transistors drifts over time, which weakens their current drive capability and may eventually cause them to fail entirely. In addition, the environmental background radiation interacts with the silicon and may induce bit-flip in the configuration memory of FPGAs. The resulted erroneous configuration bits essentially alter the functional definition of the circuits implemented on the FPGA, which renders the computation results from the FPGA corrupted.

In a dependable system, particularly for safety- and mission-critical applications, faults shall be discovered with small detection latency, be located and avoided to

block them from affecting the whole system. Furthermore, proactive countermeasures shall be taken to prevent the emerging of faults in the first place. To realize a dependable reconfigurable architecture, the following key dependability techniques are developed in this thesis:

- To detect faults in the reconfigurable fabric, on-demand and periodic testing are scheduled along with the functional workload. Before the instantiation of accelerators, the underlying reconfigurable resources in the fabric are exercised by exhaustive structural test. After the accelerators are configured into the fabric, their correct functionality is periodically checked by functional test. The combination of both test schemes achieves a high fault coverage and low test latency at minimal performance overhead. Experimental results show that the reconfigurable fabric can be exhaustively tested every 4 seconds at a performance cost of less than 4.4%.
- When part of the reconfigurable fabric is detected to be faulty, it can be avoided, i.e. do not participate in computation, while the system performance is not adversely affected. This is achieved by a novel design method Module Diversification. For each accelerator/module, it generates a set of configurations that are diversified in terms of resource usage. Alternative configurations that do not require the faulty resources can be used to maintain the system operation, even in the presence of faults. Reliability improvement factors between 19 and 330 were achieved in the experiments.
- Computation using the reconfigurable fabric induces electrical stress on individual transistors, which leads to the aging of transistors. A novel stress balance technique is developed that is able to distribute the stress induced by workload uniformly over all resources in the fabric, such that the stress on individual transistors is minimized. A combined approach with runtime accelerator placement and synthesis-time logic placement delivers uniform stress distribution that significantly prolongs the system lifetime at negligible runtime cost. Compared to state-of-the-art methods, this work reduces dynamic and static stress by up to 64% and 35%, which translates to a lifetime improvement up to 177% and 14%, respectively.
- In order to guard the correct functionality of accelerators against random bitflip in the configuration memory due to environmental radiation, countermeasures such as modular redundancy are indispensable. A worst-case solution incurs high area and energy overhead due to over-protection under a varying environment. This work proposes a novel runtime system that dynamically chooses the most efficient protection mechanisms for different accelerators, depending on the vulnerability of individual accelerators, reliability constraints of the application and environmental radiation level. Compared to related work with statically optimized redundancy techniques, the proposed method provides up to 68% higher performance at the same target reliability.

As a result, by exploiting the flexibility that is inherent in reconfigurable architectures, this work presents a comprehensive solution for fault discovery, fault tolerance and aging mitigation, defending against both permanent and transient faults.

1 Introduction and Motivation

Runtime reconfigurable architectures based on Field-Programmable Gate Arrays (FPGAs) are emerging over the recent years as a promising augment to conventional processor architectures such as Central Processing Units (CPUs) and Graphic Processing Units (GPUs). Their essential feature, runtime reconfiguration, enables dynamic customization of the hardware organization for changing application requirements. A typical structure of reconfigurable architectures is shown in Fig. 1.1. It consists of a general purpose processor and a reconfigurable fabric, which are interconnected over a communication infrastructure. The reconfigurable fabric is partitioned into multiple reconfigurable regions, which can reside on multiple FPGAs or in one FPGA. During runtime, these reconfigurable regions can be reconfigured to implement accelerators that perform compute-intensive functions to speedup the execution of applications using hardware. Such an approach takes the advantages of high performance and low energy consumption only achievable in hardware, while providing the flexibility of software to customize the hardware function at runtime.

With the aggressive advancement of semiconductor industry, feature size of modern microelectronic devices continues shrinking in the nano-scale regime. While higher transistor density, higher performance and lower energy consumption are the major benefits and the driving force of ever shrinking nano-CMOS devices, dependability poses as a serious challenge lying ahead of continued scaling. A paradigm shift is happening that a dependable system has to build upon components of undependable natures. Manufactured in latest technology nodes, modern FPGAs are increasingly prone to various dependability issues, which threaten the dependable acceleration in the reconfigurable fabric.

Latent defects not discovered during manufacturing, soft errors in the sequential elements caused by single event upsets and transistor degradation caused by various aging effects are the major dependability concerns in safety and mission critical

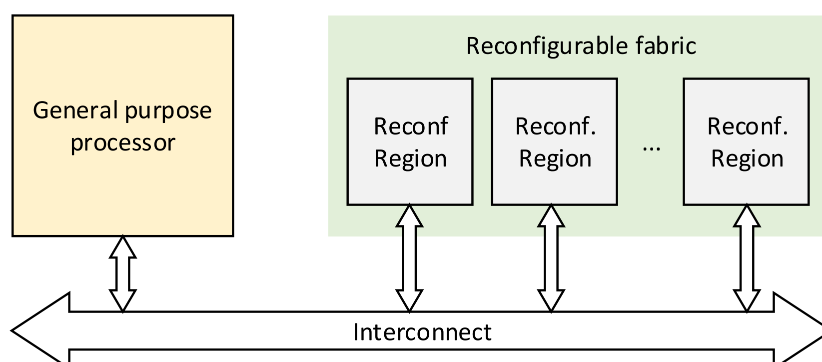

Figure 1.1: Illustrative example of a reconfigurable architecture

applications. A dependable system shall be able to discover a fault with small detection latency, to tolerate the discovered fault with minimal performance degradation and even to prevent the fault in the first place.

This thesis addresses these requirements by answering the following questions: How can we test the system when its hardware organization changes during runtime? If part of the FPGA is tested to be faulty, how can we exploit runtime reconfiguration to isolate those faulty resources such that the system continues operation with minimal performance degradation? How can we further prolong the system lifetime by delaying the failure time of the FPGA? When high reliability of correct computation is required, how can we tailor the accelerator organization to defend the soft errors caused by single-event-upset, even when the environmental condition is changing? And how all of these can be accomplished with minimal hardware and runtime overhead?

First of all, let's discuss which dependability challenges we are facing and where they come from.

1.1 Dependability Challenges in the Nano-CMOS Era

The continued down-scaling of feature sizes of digital integrated circuits still follows the trends envisioned more than 40 years ago by Gordon Moore [Moo75]. The initial motivation of scaling was to reduce the cost per electronic function, or rather the cost per transistor. As more transistors can be integrated in one chip and more chips can be fabricated with one wafer, the cost of manufacturing a wafer is now shared among hundreds of billions of transistors[1]. Although the cost per wafer rose exponentially with technology node scaling due to the ever more sophisticated manufacturing process towards the physical limit, in fact, the decreasing trend of cost per transistor has been sustained thanks to the accelerated trend in the increase of transistor density [Hol16]. In addition to the economical benefits, transistor scaling also helps the circuit to operate at higher frequencies and with lower energy consumption as transistor switching activities require less charge transport at smaller geometric scales. The soaring transistor density allows the high integration of rich functionalities at affordable price, which leads to the deep penetration of nanoelectronic devices in every corner of our daily life. High demand from traditional markets such as data-centers and mobile devices and from emerging innovations such as internet of things and autonomous vehicles are further driving the down-scaling in favor of lower cost, higher performance and lower energy consumption.

On the other hand, the down-scaling of the geometry of transistors is accompanied by a two-fold up-scaling of the products composed of nanoelectronic devices: 1) the complexity of the products grows exponentially with deep functional integration; and 2) the volume of shipped products grows with increasing demand and new markets enabled by nano-device innovations. For instance, around 100 electronic control units with 7000 semiconductor components are concurrently operating in a

[1] More than 400 billion transistors on a 300 mm wafer for the Intel Xeon E5-2600 v3 18-core processor [BSN+15].

modern passenger car [ALHS12] for the powertrain, safety, comfort and infotainment functions under a wide range of environmental conditions. And the global passenger car sale number has reached 70 million in 2015 [Sta16]. However, the up-scaling of complexity and volume poses a great challenge to the dependability and quality of the products, particularly for safety-critical systems like vehicles where a single functional failure may be life-threatening.

1.1.1 Challenges from Up-Scaling

As the number of components or integrated functions (that could be implemented with multiple components) of a system increases, the requirements to the dependability of individual components or functions is becoming increasingly stringent as well. The dependable operation of a system relies on the correct collaborative functioning of all sub-components and the failure of any one of them may manifest as a critical malfunction of the whole system. The more functions reside in a system and the complexer their interactions are, the more sources of failure exist and the higher the probability that the system fails. Because it is commonly assumed that technology advances would always benefit all aspects of a system including dependability, the dependability specification (e.g. mean time to failure) of a system desired by customers would at least remain the same or become even higher, regardless of the increased internal complexities due to functional integration. Therefore, the dependability of individual functions needs to be accordingly improved to meet the system-level dependability specification.

Extensive test during production and thorough verification during design are indispensable to guarantee the dependability of complex systems and devices. The up-scaling of functional complexity also has made a dramatic impact on the test of nanoelectronic devices containing billions of transistors. Given limited monetary and time budget for test, new defect mechanisms emerged from advanced nano-manufacturing technologies and ever sophisticated circuit designs, e.g. System-on-Chip (SoC), may render certain defects escaping from manufacturing tests and remaining in the shipped products [Zor13].

The new defect mechanisms are mainly resulted from the process variation at nano-scale (e.g. random dopant fluctuation) and from the introduction of new materials (e.g. cobalt for copper encapsulation), new structures (e.g. 3D, FinFET) and new power management techniques (e.g. voltage scaling). These defects could yet be captured by the fault models in current automatic test equipments.

An SoC is composed of a set of distinct Intellectual Property (IP) cores, e.g. multi-core processors, graphic processing unit, embedded memories, radio frequency circuits for wireless connectivity, etc. Each type of IP cores needs to be tested in individual sessions with dedicated test methods, which leads to extended test time. Moreover, on account of the intellectual protection, IP vendors provide only limited IP structural information and impose additional test constraints, e.g. specific rules for applying test patterns [WST08]. These complications lower the test efficiency and increase the test time further, which translates to the climbing cost of test.

As the volume of shipped products scales up, if the dependability of one product, typically measured in Defective Parts Per Million (DPPM), would linger on the same level as the time of the small-scale production, the cost of rejection, repair and the loss of customer satisfaction would ultimately damage the profitability and the reputation of a company. A zero DPPM target shall be met if the product is to be deployed in high-volume and safety-critical systems, such as in automotive systems [Con15].

To summarize, the up-scaling of the internal complexity and the shipping volume urges the semiconductor industry to address the aforementioned dependability challenges before continuing the down-scaling of the transistor size. Unfortunately, these challenges are being further aggravated as the manufacturing process approaches the physical limits.

1.1.2 Challenges from Down-Scaling

Nowadays, the smallest feature size (e.g. 10 nm) manufactured on a chip is well below the wavelength of the light source (193 nm) used in the most advanced available photolithography technique, i.e. immersion lithography [GBF+04]. The pattern transfering process from the mask[2] to the wafer experiences significant fidelity degradation due to the limited resolution in the optical systems which have received no resolution improvement since 2007 [STY+15]. The desired precise geometries on the wafer can thus only be approximated by auxiliary methods such as multiple patterning, optical proximity correction and phase-shift masks. These approximations rely on the interference and diffraction of light to create sub-wavelength features, which exhibit wide shape variations both locally in the feature itself and globally across the wafer [DLW09]. For example, a square shape definition on the mask will be transferred into an irregular rounded rectangle with curvy edges. The roundness and curviness will variate depending on the surrounding shapes and the surface properties of the photoresist and wafer in the shape definition region. This geometry variability caused by the inaccurate shape definition is one of the major sources of variability that impact the deterministic nature of transistor characteristics; e.g. the threshold voltage of transistors at different locations on a wafer may vary randomly in a wide range [ALWA+14].

Another major source of variability stems from the atomistic level fluctuation of dopants in the channel region of transistors, so-called Random Dopant Fluctuation (RDP). During the fabrication of transistors, impurity atoms (dopants) from group III/V elements are intentionally introduced (doped) into the intrinsic silicon substrate to modulate it into the p/n-type semiconductor. In addition, doping into the channel region of transistors can also be employed to adjust the threshold voltage or control short-channel effects [Shi16]. As the dimension of transistors scales down, the total number of dopants in the channel region decreases to an extent that a change of just only a few atoms would have significant impact on the transistor characteristics. In 45 nm CMOS technology, the average number of dopant atoms

[2] Masks are fabricated using electron beam lithography, which offers very high resolution (below 5 nm) but very low throughput and thus is not suitable for mass chip production.

in the channel region decreases below 100, where RDP contributes to around 60% of the total variation in the PMOS threashold voltage [KKK+08]. Moreover, not only the number but the spatial variation in the position of dopants also affects the transistor performance [Shi16]. State-of-the-art FinFET technologies exhibit around 20 mV standard deviation in the threshold voltage [BDB+13, ALWA+14].

To cope with the variability inherent in the manufacturing process, a large timing margin is applied during design time such that the circuit could work at a specified frequency under the most unfavorable variability conditions. Unfortunately, another variability mechanism that requires a comparable amount of margin is becoming crucial for the transistors to survive through time. It is aging, a kind of temporal variability that changes the transistor characteristics over time.

Aging effects originate from the degradation in the electrical or physical properties in the material resulting from continued operation of the device. For instance, traps which restrict the free movement of electrons or holes may form at the interface between the gate oxide and the channel of a transistor as a result of the undesired tunneling of carriers from the channel into the gate oxide when the transistor is operating. These traps are positively or negatively charged in PMOS or NMOS, respectively and shift the threshold voltage of transistors in a harmful way, i.e. increase in NMOS and decrease in PMOS. This lowers the current drive capability of the transistor under a fixed supply voltage and may eventually lead to the timing violation in sequential circuits. Major aging effects in nano-CMOS transistors include Bias Temperature Instability (BTI), Hot Carrier Injection (HCI) and Time-Dependent Dielectric Breakdown (TDDB) [RBC+13, CVS+14], while Electro-Migration (EM) is considered as a major aging issue affecting the interconnects with high current densities, e.g. power delivery networks [Lie13].

With the technology down-scaling, these aging effects are expected to persist as a major dependability challenge. 10 nm FinFETs shall continue to suffer from aging and the device susceptibility to different aging mechanisms shall also change with the introduction of new device structures and materials [SWS+14]. There is also evidence that nano-FinFETs experience degradation more severely than planer devices [LKC+13, LWL+14]. A continuous decreasing trend in transistor lifetime across advancing technology nodes [HCF+15] calls for efficient countermeasures against aging. Typical guardband techniques lower the target frequency at design time such that the system can still operate at the end of lifetime, although at the beginning of lifetime the system is capable of operating at a higher frequency. For the current technology nodes, aging, in particular NBTI, leads to an even higher shift (50 mV in average) in the threshold voltage [CKR+15] in the worst case than the variation caused by the manufacturing process. Therefore, a larger timing margin would be required on top of the margin tailored for the process variation. This pessimistic over-design may eventually offset the performance benefit enabled by the technology scaling.

1.1.3 Addressing the Challenges

The challenges from up-scaling lie in the conflict between the growing demand of high dependability product and the increasing difficulty in manufacturing a defect-free chip. Down-scaling enables higher performance which is however diminished by the introduction of large guardbands against the spatial and temporal variability in the transistor characteristics.

Besides Moore's Law in the visionary paper [Moo75], it was also recognized that system reliability was improved dramatically along with the increasing functional integration. In the 90's, Texas Instruments and Intel are striving to reach the reliability goal of 0.1 FIT[3] [Gha91] and 10 FIT [STW98] by the year 2000, respectively. However, the efforts to improve the product reliability still fall behind the imminent challenges in the nano-CMOS technology. Following the current down-scaling trend, the complications in the manufacturing process are envisioned to become more challenging in the future [Kuh12, SAB+13]. Merely relying on the improvement in the manufacturing process is therefore determined not to be able to meet the dependability goals desired by the upper tier vendors such as automotive suppliers [vT08] which expect zero defect in product and zero failure in time. New solutions in addition to the extensive testing during production and verification during design are necessary to address these challenges.

Runtime dependability management opens a new level of freedom to address the above nano-era dependability issues. A runtime strategy has the following advantages: 1) Latent faults not detected during manufacturing and permanent faults caused by aging can be detected using online tests; 2) The system is able to adapt itself to changing environment conditions in a way that optimized decisions, trading-off performance and dependability, can be made during runtime, instead of a static pessimistic design decision that targets the worst environment condition; and 3) A further optimization potential can be exploited that takes into account the different dependability requirements of workloads and the impact of them on the system states.

In this thesis, runtime self-defense mechanisms including online monitoring, runtime dependability modeling and the orchestration of dependability countermeasures are developed to increase the system reliability and availability. Escaped faults from manufacturing tests are detected and localized by online testing. Concurrent error detection captures random hardware failure and prevent them from propagating into other functional units in the system. Fault-tolerance techniques are developed to isolate permanent or transient faults such that the system continues delivering service even in an impaired state. Transistor aging effects are mitigated to allow for an extended system lifetime or a tighter guardband at the beginning of the system lifetime.

[3]Failure-In-Time: 1 FIT = 1 failure in 1 billion hours

1.2 Thesis Contributions

The contributions of this work are summarized as follows:

Pre- and Post-configuration Test: The correct operation of runtime reconfigurable architectures essentially relies on the dependability of the reconfigurable fabric on the FPGA. This requires that the underlying hardware structures are fault-free and the accelerator reconfiguration process completes without errors. In this work, these are ensured by on-demand and periodic testing: Pre-configuration test (PRET) checks the structural integrity of the underlying hardware before the instantiation of accelerators and Post-configuration test (PORT) periodically checks the correct functionality of configured accelerators after they are instantiated. The strategic scheduling of PRET and PORT delivers high fault coverage and low test latency at marginal performance cost.

Module Diversification: If part of the reconfigurable fabric is detected to be faulty, system breakdown is avoided by employing the novel design method called module diversification. Alternative configurations of accelerators that have diversified resource usage are generated during design time. Self-repairing is achieved by circumventing the faulty resources with accelerator configurations that do not require the faulty resources. The module diversification process is able to generate minimal number of diversified configurations of each accelerator in order to minimize the storage overhead of configuration bitstreams.

Stress-aware Placement: System degradation threatened by aging effects is addressed by stress-aware runtime accelerator placement and design time logic placement. During runtime, the stress induced by accelerators is uniformly distributed over all available reconfigurable resources. At design time, the overlapping of high stress regions among individual accelerators is minimized to further improve the runtime stress distribution. Both together balance the intra- and inter-region stress induced by the application workload at negligible performance cost, which leads to significant maximum stress reduction and prolonged system lifetime.

Adaptive Modular Redundancy: For applications with strict reliability constraints, highly reliable operation of accelerators is required, which is however threatened by soft-errors in the configuration memory. A single bit-flip in the configuration memory may impair the functionality of the accelerators. Concurrent error detection techniques such as modular redundancy are obligatory, which may incur high resource usage and performance loss due to over-protection. To guarantee the required reliability at minimal performance cost, the runtime system dynamically selects appropriate redundancy degree of accelerators, depending on the vulnerability of individual accelerators, reliability constraints of the application and environmental radiation level.

By exploiting the inherent flexibility provided by runtime reconfigurable architectures, this thesis addresses the nano-era dependability challenges in a cross-layer fashion, from transistor layer over circuit layer till accelerator layer, from design time to runtime. With strategic online testing, self-repairing, stress balancing and adaptive modular redundancy, a resilient and highly dependable runtime reconfigurable

architecture is accomplished at minimal hardware and runtime overhead.

1.3 DFG Research Program SPP-1500 and InvasIC

This thesis is accomplished in the scope of the the DFG Priority Program SPP 1500 "Dependable Embedded Systems"[4] [HHH+11]. This program focuses on the various reliability concerns in the nano-era, including manufacturing variability, aging, the impact of temperature and soft errors and addresses these from a wide range of perspectives including operating systems, compilers, micro-architectures and applications themselves [HBD+13].

Research groups from ten German universities proposed twelve research projects to approach these reliability issues by their own competences and expertise. These projects emphasize on solutions that leverage cross-layer methodologies, employ runtime adaptation and exploit application resilience. Cross-layer methodologies are able to capture the propagation of faults from lower to upper layers and allow the combined optimization on multiple layers. Runtime adaptation utilizes various online monitoring facilities to observe the system behavior and environmental parameters, e.g. error counters, radiation sensors, online self-test, etc. They provide the opportunities to dynamically optimize the system performance and power in a changing environment under changing system states and thus to avoid the pessimistic margining techniques at design time. Similarly, by considering the inherent reliability characteristics of applications, cost-efficient reliability methods can be developed, which prevent high performance/energy cost due to over-protection.

This thesis is funded by one of the twelve projects, called OTERA (Online Test Strategies for Reliable Reconfigurable Architectures), which targets the reliability improvement in runtime reconfigurable architectures and closely follows the preceding strategies for cost-efficient runtime reliability management.

Another related research program, Invasive computing[5] (InvasIC) [HHB+12], proposes a new programming paradigm for heterogeneous many-core architectures, where applications are delegated the ability to request (invade) and free (retreat from) compute tiles, so-called resource-aware programming. A tile-based many-core platform is developed to demonstrate the value of invasive computing. The compute tiles provide a set of rich heterogeneous computational resources, e.g. RISC cores, reconfigurable processors and special purpose hardware accelerators. An agent-based distributed system manages the efficient resource distribution depending on the application requirements. The target architecture of this thesis is based on one type of the compute tiles, *i*-Core [HBGZ14], that extends the instruction set of a general purpose processor with special instructions to allow the hardware acceleration in the reconfigurable fabric.

[4] http://spp1500.itec.kit.edu
[5] http://invasic.de

1.4 Thesis Outline

The rest of this thesis is structured as follows:

Chapter 2 provides the background knowledge of FPGA-based reconfigurable architectures and dependability issues in CMOS circuits. After introducing the structures of an FPGA and the principles of reconfiguration, the reconfigurable architectures that employ FPGAs as the reconfigurable fabric and the acceleration of applications on them is discussed. Next, the dependability issues that cause permanent and transient faults in CMOS circuits, i.e. aging and single event upset, are presented in details, along with the mathematical models used in this thesis for these degradation mechanisms. Afterwards, an overview of classical techniques for increasing the dependability of FPGAs including test, concurrent error detection and scrubbing are provided. Finally, related work on dependability improvement in FPGA-based reconfigurable systems is presented.

Chapter 3 provides a high-level overview of the contributions of this thesis and the target architecture. It shows how the dependability issues are addressed from two perspectives: structural integrity and functional correctness, along with the cause-effect analysis of the proposed methods across multiple abstraction layers. The architectural assumptions and the evaluation platform used in this thesis are introduced as well.

Chapter 4 presents the online test strategies for reconfigurable architectures, consisting of pre-configuration structural test and post-configuration functional test. The hardware integration of the proposed test schemes into the reconfigurable fabric and their runtime scheduling are discussed. The performance overhead and test effectiveness are evaluated in the experiments.

Once permanent faults are detected and localized, they need to be avoided during the computation. Chapter 5 presents a design method called Module Diversification that is able to isolate the faulty resources from the computation, without affecting the system performance. The conditions for successful fault isolation is discussed and an algorithm for generating the minimum set of diversified configurations is developed. The reliability improvement and timing cost are evaluated.

To avoid the emerging of faults due to aging effects, Chapter 6 provides a solution by distributing the stress induced by workload uniformly over all reconfigurable resources. An algorithm for runtime accelerator placement and an algorithm for synthesis-time logic placement are developed. The resulting stress reduction and lifetime improvement, together with performance overhead, are evaluated.

Targeting soft errors, Chapter 7 presents a method which allows for autonomous runtime reliability management in reconfigurable architectures. The concept of implementation variants trading-off performance and reliability is introduced. A mathematical framework is provided to determine the error probability of the computations in the reconfigurable fabric. A runtime system is developed that dynamically chooses the appropriate implementation variants to guarantees a target reliability while optimizing performance. Experimental evaluation shows the performance im-

provement compared to a static approach.

Chapter 8 evaluates the resulting system from the perspectives of structural integrity and functional correctness. It investigates the system behavior including performance and stress state in the presence of permanent faults. It shows the dynamic adaptation of the system to guarantee the computation correctness in a varying environment. The results are compared with state-of-the-art methods.

Chapter 9 concludes this thesis with a guide to future research directions.

2 Backgrounds

In this chapter, the fundamental structure of field-programmable gate arrays and the basic concepts of reconfiguration are presented in Section 2.1. An introduction to the emerging architectures based on general purpose processors and reconfigurable fabrics is given in Section 2.2. Two major dependability issues in the reconfigurable fabric, aging effects and single event upset, are discussed in Section 2.3. The employed fault models for soft-errors, stress models for aging effects and aging models for calculating threshold voltage shift are presented in Section 2.4. Section 2.5 provides an overview of the classical techniques for improving system dependability. Related work and state-of-the-art dependability techniques for reconfigurable systems are discussed in Section 2.6.

2.1 Field-Programmable Gate Arrays

Field-Programmable Gate Arrays (FPGAs) are semiconductor devices whose functionalities are defined by users after manufacturing. In contrary to a *General Purpose Processor (GPP)* that delivers the desired functionalities by the execution of sequential instructions stored in a memory, an FPGA implements the functionalities directly using its hardware structure. Complex functions such as a video encoder or even a processor can be mapped to the abundant logic gates in an FPGA, while in a GPP they can only be performed using few Arithmetic Logic Units (ALUs) which provide a very limited set of primitive arithmetic and logic functions. In comparison to an Application-Specific Integrated Circuit (ASIC) that are a full-customized hardware structure for a dedicated application, an FPGA offers the flexibility that its hardware organization can be freely reconfigured to implement user-defined applications.

In this thesis, the target FPGA used for experiments and prototyping is a Xilinx Virtex-5 XC5VLX110T FPGA [Xil12d].

2.1.1 The Reconfigurable Fabric

The fundamental idea behind the hardware reconfigurability in an FPGA is to provide a large amount of reconfigurable primitive logic elements and routing structures such that arbitrary logic functions can be mapped to these primitives. After 30 years of exploration and evolution, modern commercial FPGAs have converged to the island-style architecture [Tri15] shown in Fig. 2.1. *Configurable Logic Blocks* (CLBs) and *Programmable Switching Matrices* (PSMs) are the two essential components of an FPGA. They are arranged in a two-dimensional array and repeat

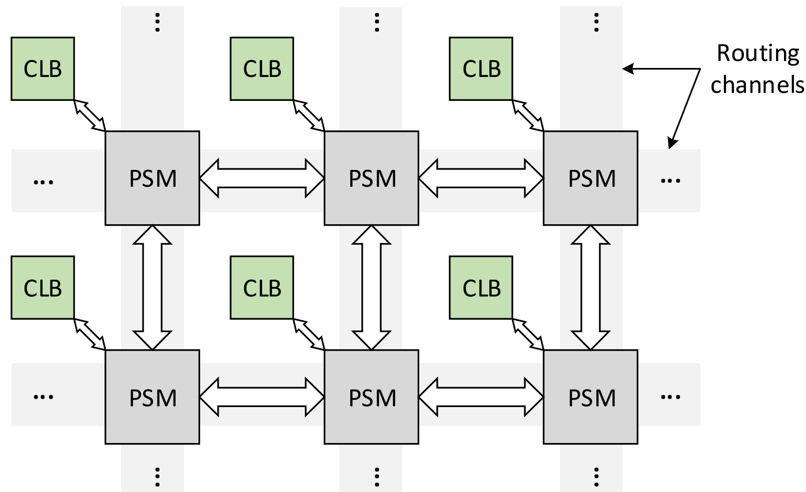

Figure 2.1: Island-style FPGA architecture

themselves across the whole FPGA. Special purpose components such as on-chip memories and Digital Signal Processors (DSPs) can be embedded in the array or in place of CLBs to provide efficient implementation of storage and arithmetic circuits. CLBs are the basic reconfigurable logic elements in an FPGA.

CLBs are surrounded by routing channels that are segmented by PSMs, like CLB islands in a sea of wires. So comes the name "island-style". The communication of electric signals among CLBs are configured using the PSMs. Each CLB has an attached PSM that manages the signals coming out of and going into the CLB. The PSMs are also responsible for the establishment of the connections between the horizontal and vertical wires in the routing channels such that any two CLBs in the fabric can communicate with each other.

2.1.2 Configurable Logic Blocks

The CLBs are the basic reconfigurable resources for implementing combinatorial and sequential logic functions. In this thesis, a Xilinx-style CLB structure [Xil12d] is targeted. Figure 2.2 shows a 2×2 arrangement of four CLBs, where PSMs and routing channels are omitted for illustrative purposes. Each CLBs consists of two *slices* that are aligned in columns. Neighboring slices in one column are directly connected by carry signals for the compact implementation of carry chains of adders so that the timing critical carry signals need not to be routed through the global routing matrix.

Every slice contains four identical set of reconfigurable resources including *Look-Up Tables* (LUTs), configurable registers and configurable internal routing. Figure 2.3 shows a simplified view of one set of the reconfigurable resources. Detailed structural view of a slice can be found in [Xil12d]. The main components in a slice are

- a 6-input LUT that is used to implement an arbitrary boolean function with maximum 6 variables,

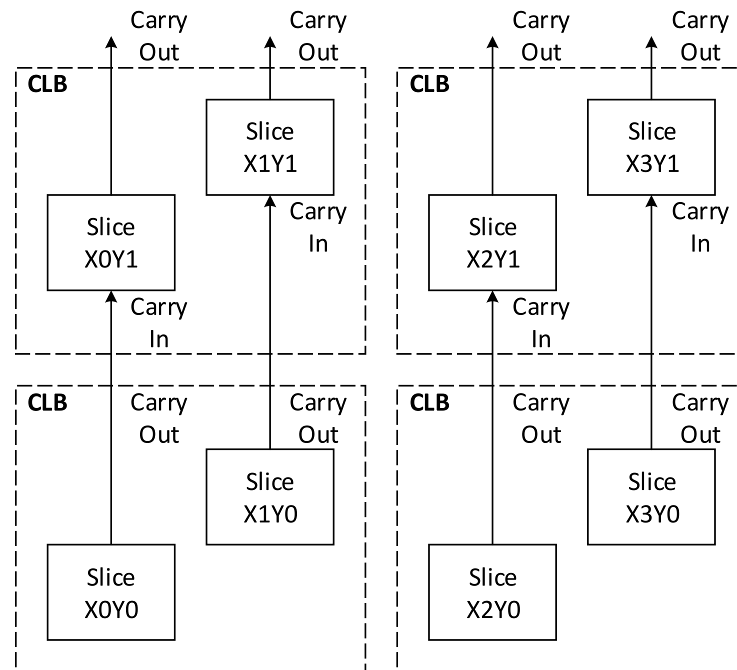

Figure 2.2: Xilinx-style CLB structure [Xil12d]

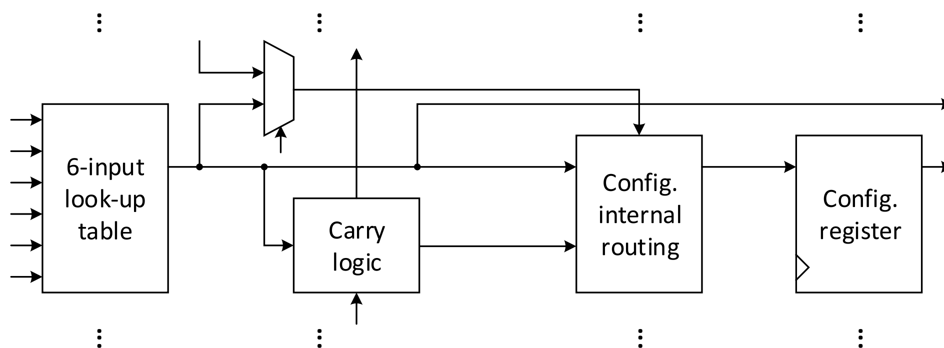

Figure 2.3: Reconfigurable resources in a slice

- a multiplexer that can be cascaded with other multiplexers within the same slice to provide a wide multiplexing,
- a carry logic that enables carry propagation through slices in one column,
- a multiplexer-based configurable internal routing to select which signal in the slice should be a registered output, and
- a configurable register that can be configured into an edge-sensitive flip-flop or a level-senstive latch and holds a user-defined initial state.

The LUTs are the core components in an FPGA for the implementation of combinatorial logic functions. The 6-input LUT is basically a truth table based on a read-only memory with 64 (2^6) 1-bit cells. The inputs of the boolean function act as the address bits of the memory and the stored single bit at that address as the result of the boolean function. Multiple LUTs in one slice can be combined to operate as a *Distributed RAM* to store up to 256 bits of data [Xil12d].

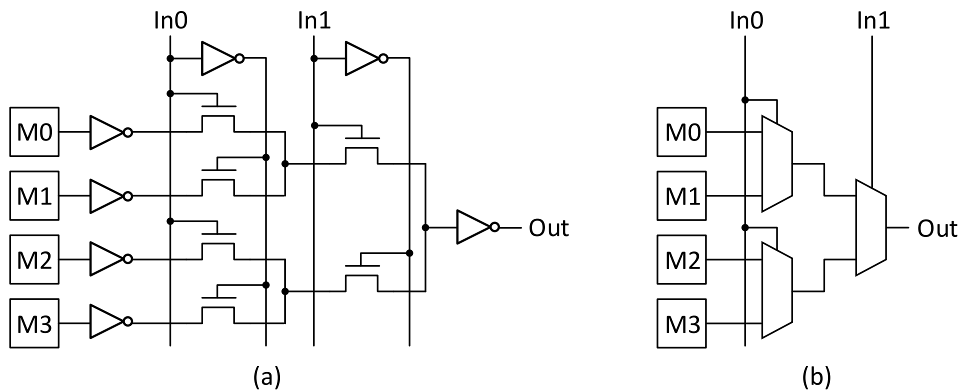

Figure 2.4: Internal structure of a 2-input LUT

2.1.3 Transistor-Level LUT Model

In this thesis, a transistor-level LUT model based on pass transistors [PC03, SSC09, KAT11] is employed, as shown in Fig. 2.4(a) for a 2-input LUT. M0–M3 are four 1-bit memory cells which store the four possible outputs of the LUT. The two inputs of the LUT, In0 and In1, determine the on/off states of the pass transistors and create a path from one of the memory cells (M0–M3) to the LUT output, i.e. forcing the value of the LUT output to be set to the value of the selected memory cell. The inverters behind the memory cells and at the output act as signal buffers for driving long paths in the LUT and large fan-outs outside the LUT, respectively. A 6-input LUT can be constructed in a similar way, where 64 memory cells are being selected by 6 stages of totally 126 $(2^6 + 2^5 + \cdots + 2)$ pass transistors.

Alternatively, the LUT structure can be modeled using 2:1-multiplexers, as shown in Fig. 2.4(b). A pair of pass transistors that take opposite control signals and share one output signal line can be wrapped as a 2:1-multiplexer. The select signals of the multiplexers are connected to the LUT inputs. From this point of view, a LUT is a wide multiplexer consists of multiple stages of cascaded 2:1-multiplexers.

2.1.4 Programmable Switching Matrices

The PSMs are the core components of the reconfigurable routing resources, which establish the interconnections among CLBs in the routing channels. The routing channels are arranged in a grid fashion, i.e. wires going either vertically or horizontally, as shown in Fig. 2.1. At the intersection of these channels, the PSMs offer the possibilities to connect the wires in orthogonal directions.

As shown in Fig. 2.5, at the cross points of wires in a PSM, *Programmable Interconnection Points (PIP)* provide the configurability for routing. Each PIP contains six pass transistors whose control terminals are connected with 1-bit memory cells. These pass transistors enable six routing options for the two crossing wires: 1) top to bottom, 2) left to right, 3) top to right, 4) bottom to right, 5) top to left and 6) bottom to left, the connectability of which is controlled by the bit stored in the corresponding memory cells. Depending on the physical implementation of the FPGA,

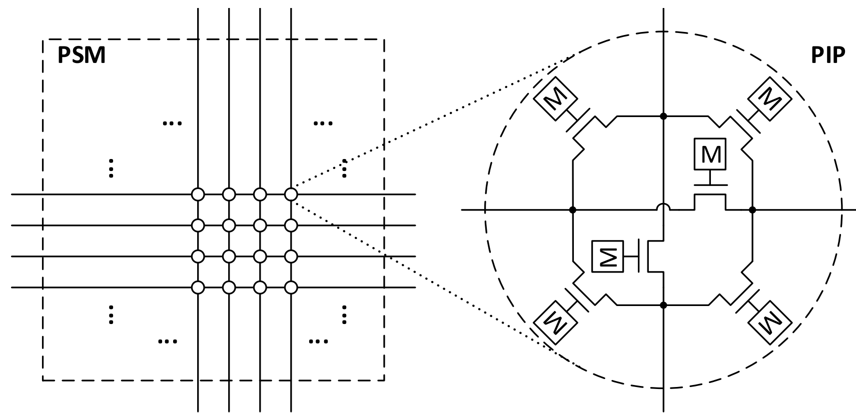

Figure 2.5: Internal structure of a PSM and a PIP [HCJ+90]

these connections may be bidirectional or unidirectional. Some constraints may apply to which routing options can be enabled simultaneously in order to prevent illegal routing scenarios or race conditions, such as turning on all pass transistors at the same time in a extreme case.

2.1.5 Configuration Memory

As presented in previous sections, the logic functions within CLBs and the interconnections among CLBs are controlled by configuration bits stored in a memory, which is called *configuration memory*. To configure an FPGA to implement the desired functionality, the configuration bits of all used reconfigurable resources need to be set to appropriate values so that CLBs are configured to perform the desired logic functions and the PIPs in PSMs are configured to establish the necessary interconnections among used CLBs.

The exact values of configuration bits are prepared by tools from the FPGA vendors. The circuit description, usually written in hardware description languages (VHDL or Verilog) are first synthesized to a gate-level netlist which describes which logic gates to use and how they are connected to implement the circuit. The logic gates and connection wires are then mapped to the logic resources (e.g. LUTs and registers) in the CLBs and wires in the routing channels, respectively. In the last step, the mapping is translated to the values of the configuration bits of CLBs and PIPs. The prepared configuration bits during design time are stored in a binary file, so-called *bitstream* file, which can then be written into the configuration memory inside the FPGA by means of different configuration modes, e.g. by using boundary scan or the Internal Configuration Access Port (ICAP) [Xil12c].

In Xilinx FPGAs, the configuration bits are arranged in *frames* in the configuration memory [Xil12c]. A frame is the smallest addressable unit in the configuration memory and contains the configuration bits of reconfigurable resources spanning the height of one clock region[1]. As shown in Fig. 2.6 for a Xilinx Virtex-5 FPGA, the FPGA is partitioned into 4 clock regions, each of which spans across the whole

[1]A clock region is a branch of the clock tree which allows zero-skew clock distribution in an FPGA. In Virtex-5 FPGAs a clock region is 20 CLBs in height [Xil12d].

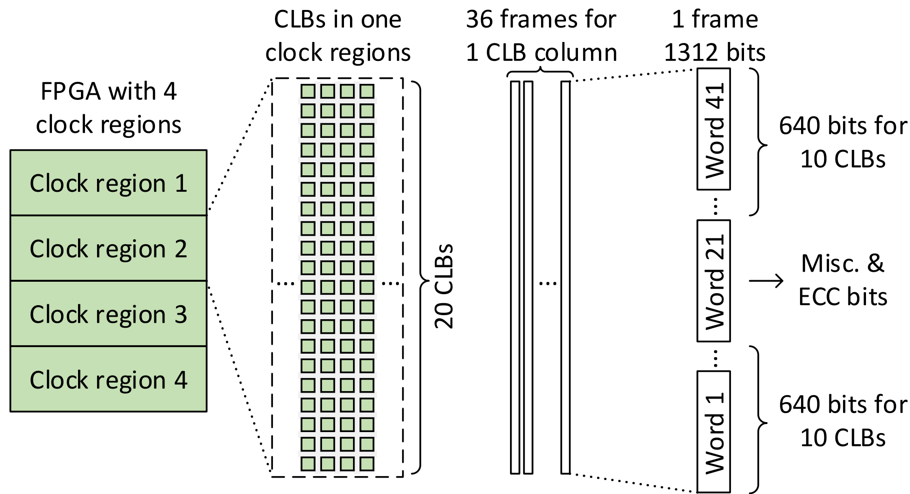

Figure 2.6: Configuration frames in a Xilinx Virtex-5 FPGA

FPGA in width and is of 20-CLB height. The configuration bits for one column of CLBs (including the attached PSMs) in one clock region are grouped in 36 frames, each of which contains 1312 bits (41 32-bit words). The bits in one frame are organized in such a way that the word 1..20 are for the 10 lower CLBs and word 22..41 are for the upper 10 CLBs. Word 21 in the middle of the frame is a special word that contains miscellaneous configuration bits and a 12-bit Error Correction Code (ECC) for the whole frame. These ECC bits allow any single-bit errors in a frame to be corrected and any double-bit errors to be detected, where the errors may be caused by single event upset (see Section 2.3.4).

In modern FPGAs, the configuration memory is typically implemented using SRAM or flash technologies [KTR08]. SRAM-based FPGAs are the dominating technology mostly because of 1) unlimited reconfigurability and 2) compatible with standard CMOS manufacturing technology. Unlike flash cells, SRAM cells do not suffer from wear-out during the write process and therefore has indefinite number of reconfiguration (write) cycles. Being manufactured using standard CMOS transistors, SRAM-based FPGAs also immediately benefit from the latest advances in the semiconductor technology and thus provide higher logic density, faster switching speed and lower energy consumption. Thanks to these advantages, SRAM-based FPGAs are considered to be the most appropriate technology for runtime reconfigurable architectures.

2.1.6 Partial Reconfiguration

Modern FPGAs allow the runtime reconfiguration of a part of the FPGA without interrupting the operation in the rest part [Xil12b]. This flexible feature is called *partial reconfiguration*. In Xilinx FPGAs, the basic unit of partial reconfiguration is a rectangular partition in the FPGA. In this thesis, this is called a *reconfigurable region* or in short a *region*.

Figure 2.7 illustrates the concept of partial reconfiguration of regions. A region consists of a rectangular array of CLBs and corresponding PSMs. The dimensions

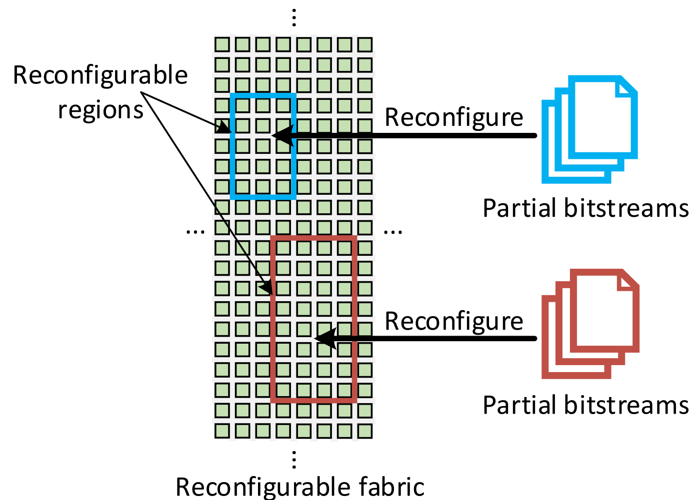

Figure 2.7: Partial reconfiguration with partial bitstreams

of the region can be arbitrarily chosen with minimum allowed height or width of one CLB. The reconfigurable resources outside the regions are defined as the *static* part in the FPGA. Every region can be reconfigured independently of each other and without compromising the operation states in other regions or in the static part. One region can be reconfigured to implement different functional modules by reconfiguration using different *partial bitstreams*, which contains only the configuration bits for the reconfigurable resources in that region. Xilinx recommends that the region boundary in height be aligned with one or multiple clock regions whenever possible [Xil12b], because one configuration frame spans a height of exactly one clock region. Otherwise, the partial bitstreams would contain unused configuration bits for resources lying outside the reconfigurable region, which leads to the unnecessary storage overhead for the partial bitstreams.

The flexibility provided by partial reconfiguration enables efficient FPGA area utilization by multiplexing the hardware in time and space. This is of great importance for runtime reconfigurable architectures, where different accelerators are prepared as partial bitstreams so that they can be configured into the FPGA on-demand depending on the application needs.

2.2 Fine-Grained Reconfigurable Architectures

The persistent demands for computing power fueled by complex applications such as big data analytics and artificial intelligence are encouraging the innovations on high performance and energy-efficient processor architectures.

2.2.1 Filling the Gap between GPP and ASIC

Multi-core general purpose architectures, including CPUs and modern GPUs, are energy inefficient compared to ASICs, since the majority of the energy consumption is for supporting the general purpose instruction-based execution model, e.g. instruc-

tion fetching/decoding, while not for performing actual computations [CGG+14]. To increase the performance for ever more complex applications, a straightforward approach of simply piling more cores will inevitably face the wall of dark silicon [EBS+11]. The power consumption and heat dissipation will eventually limit the maximum number of concurrently operating cores, which defeats the purpose of core scaling, i.e. using more cores to handle more work.

ASICs and Application-Specific Instruction set Processors (ASIPs) are much more superior than multi-core architectures in terms of performance and energy efficiency. ASICs, e.g. video processors, are fully customized and highly optimized circuits for a particular application, while ASIPs, e.g. Digital Signal Processors (DSPs), extend the instruction set of GPPs with dedicated computation units for a selected set of operations. However, both approaches are optimized or even only usable for a very narrow set of applications. For instance, running a music player on a video processor ASIC or a web server on a DSP would be very inefficient or not possible at all. In order to support a wide range of different applications while preserving the advantages, the application-specific architectures would need to integrate a large set of dedicated hardware modules for each type of applications, which would incur expensive area cost and long design time. On the other hand, not all applications would run simultaneously and therefore not all dedicated hardware modules are required at the same time. This inefficient area usage gives rise to the ideas of reconfigurable architectures: changing the hardware organization on-demand based on the application requirements.

Reconfigurable architectures strike a balance between the utmost flexibility provided by general purpose architectures and the high efficiency of application-specific hardwares. They support the execution of arbitrary applications while also providing the acceleration capabilities using reconfigurable hardwares. Meanwhile, they avoid the inefficient area utilization by time or space multiplexing the hardware area for different applications.

Two types of reconfigurable architectures exist: fine-grained reconfigurable architectures or FPGA-based reconfigurable architectures, and Coarse-Grained Reconfigurable Architectures/Arrays (CGRAs). This thesis focuses on the fine-grained reconfigurable architectures, since they are based on off-the-shelf FPGAs that are easily accessible. In contrast, CGRAs require customized hardware implementation and compiler support, and thus are not chosen as the target platform.

Figure 1.1 shows a typical structure of FPGA-based reconfigurable architectures, where an FPGA-based reconfigurable fabric partitioned into multiple regions is coupled to a GPP. At runtime, depending on the application requirements, accelerators can be configured into the regions to speedup the application execution.

2.2.2 Coupling of a Reconfigurable Fabric to a GPP

Three GPP-FPGA communication channels are involved in the FPGA-based acceleration: 1) control signals from GPP to FPGA and status information from FPGA to GPP, 2) input data from GPP to FPGA and processing results from FPGA to

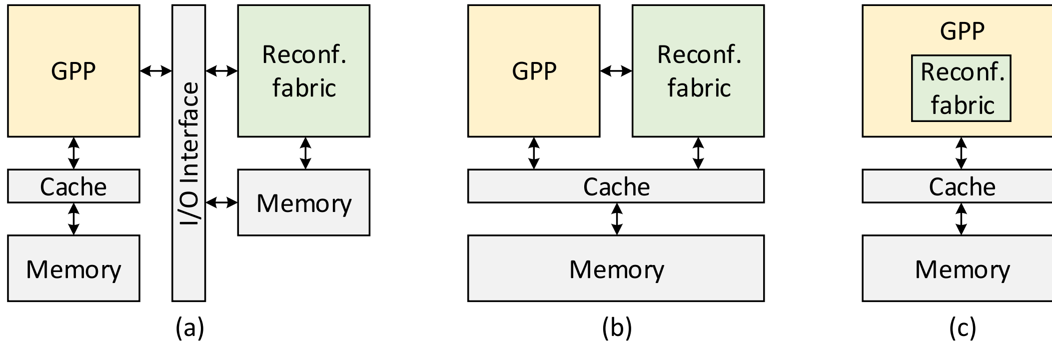

Figure 2.8: Three degrees of coupling between FPGA and GPP: (a) loosely coupled, (b) co-processor and (c) tightly coupled

GPP and 3) read and write memory access directly from FPGA. The underlying communication mechanisms depend on the degree of coupling between FPGA and GPP [TCW+05], as shown in Fig. 2.8.

In loosely coupled reconfigurable architectures (Fig. 2.8(a)), the sole communication path between GPP and FPGA is through the I/O interface of the GPP, e.g. PCI Express and the memory spaces of them are strictly separated. This sort of coupling supports relatively slow data transfer between these two devices and is only suitable for applications where minimum processor-fabric interaction is required. Before the execution of the accelerator in the fabric, the input data are transferred from the processor memory to the memory dedicated to the fabric. The accelerator then operates on the data in its local memory and transfers the results back to the processor after the accelerated computation is done. The computation in the fabric only brings speed-up when the time spent in the fabric is significantly greater than the time spent in the inter-device data transfer. A major advantage of the loose coupling is that no modification to the processor design is necessary. The fabric is just another peripheral device from the processor's point of view. The Catapult data-center accelerator from Microsoft [PCC+14] and Zynq UltraScale+ MPSoC from Xilinx [Han16] employ this sort of coupling.

The fabric can be moved more closer to the processor as a co-processor (Fig. 2.8(b)). This enables the memory sharing between the two devices and even cache coherency, as in the IBM's Coherent Accelerator Processor Interface (CAPI) [IBM15]. Therefore, no inter-device data transfer is required, which dramatically reduces the memory access overhead. The control/status channel can be established over traditional I/O or co-processor interfaces of the GPP. The shared memory model however requires modification in the processor design, particularly in the memory management unit and in the extra silicon area for implementing the cache coherency protocol [IBM15].

In the tightly coupled architectures (Fig. 2.8(c)), the fabric is embedded into the processor and becomes part of it. Essentially, the fabric plays the role as a special functional unit like floating point units and additional custom instructions together with support circuitry are created such that they can be decoded and executed in the fabric. Once the accelerator is configured into the fabric, the initiation of an accelerated execution takes very few time (in the order of cycles) and the compu-

tation results can be directly written to the processor register files. In comparison to other coupling mechanisms, the tight coupling offers the lowest performance and communication overhead for an accelerated execution in the fabric, while requiring the most design effort for integrating the fabric into the processor pipeline.

In this thesis, the proposed methods are applicable to any of the above coupling schemes, although the target platform (see Section 3.2) is a tightly coupled reconfigurable architecture.

2.2.3 Hardware Acceleration of Application Kernels

To accelerate the execution of an application in the reconfigurable fabric, various design steps are necessary during the application lifetime, from compile time to runtime.

Kernel Identification and Accelerator Development

First, the segments of codes in the application that need to be accelerated, i.e. the compute-intensive parts (so-called *kernels*), are identified. These codes are typically arithmetic operations on large arrays of data and are composed of loops iterating over the arrays. Each iteration in the loops is rather independent of another and thus the loops can be unrolled to achieve high data parallelism in hardware. In addition, the arithmetic operations might have more efficient hardware implementations than those in the ALUs of GPPs, e.g. Multiply-Accumulate (MAC) or bit-level manipulations. Next, these operations are mapped to one or more accelerators. The mapping process can be performed automatically or manually [GB11]. Depending on the I/O constraints imposed by the fabric-processor and fabric-memory communication channels, a combination of the operations or a kernel as a whole will be identified as an accelerator to be implemented in the fabric [GMP15].

After the functionalities of the mapped accelerators are determined, these hardware accelerators are synthesized by using C-to-RTL tools such as Xilinx Vivado HLS [Xil16c] that directly transforms C codes to hardware descriptions, or by manually designing the hardware according to the functions specified in the original codes such that a more optimized design can be produced. Auxiliary logic required for managing the communication with the GPP or memory such as in IBM CAPI [IBM15] may need to be inserted into the accelerator. The bitstream files for every accelerators are then generated. Meanwhile, a runtime system, usually running on the host GPP, is notified about the availability of the implemented accelerators and the storage location of the bitstreams. If multiple accelerators are cooperating with each other to perform the kernel functionality, their operation relations are described by a control-data-flow graph. It specifies in which control step an accelerator shall operate and from which of the other accelerators it receives the input data. This graph will also be made available to the runtime system for steering the cooperated accelerator execution.

In this thesis, a generic model of applications consisting of multiple accelerated kernels is employed, as present in Section 3.1. It is independent of the processor-fabric coupling and inter-accelerator communication mechanisms.

Runtime Accelerator Management

Before the execution of an application kernel, the required accelerators need to be configured into the fabric. The following problems are solved by the runtime system before starting the actual configuration.

Selection determines which accelerators shall be configured into the fabric, given the limited number of reconfigurable regions. In this work, a selection algorithm is proposed in Chapter 7 to trade-off accelerator parallelism with redundancy.

Scheduling arranges the sequence of configuring multiple accelerators, which benefits those kernels that can start computing immediately when only a subset of accelerators is available [BSKH08].

Placement decides which accelerator shall be placed into which region. It optimizes the communication overhead among accelerators [GBH12] or the stress distribution to increase lifetime as presented in Chapter 6.

The configuration of the accelerator bitstreams takes a certain amount of time. For instance, the maximum configuration bandwidth through ICAP in a Xilinx Virtex-5 FPGA is 400 Mbps and the number of configuration bits of one CLB column of one clock region is 47,232 bits (see Section 2.1.5). Configuring a small accelerator of four CLB columns would take at least 0.48 ms, which amounts to almost 5×10^4 cycles in a system running at 100 MHz. To avoid the kernel waiting for the completion of accelerator configurations, it is a better practice to perform the accelerator configuration before encountering the kernel, a technique called *prefetching* [CH11]. With prefetching, the configuration of the accelerators of the next kernel occurs while the system is still busy computing the current kernel or non-accelerated instructions on the GPP. For this purpose, runtime prediction techniques are required to guess which kernel and accelerators will be most probably executed next such that when the kernel is about to execute, the required accelerators are already configured in the fabric. In this thesis, a prediction technique based on the combination of offline profiling and online monitoring [BSKH07] is employed in the base architecture.

2.3 Dependability Issues in CMOS Circuits

Over time, microscopic phenomena change the material properties of a chip, in a negative fashion. Depending on the usage conditions, e.g. circuit activities, temperature and voltage, the behavior of the transistors and wires varies since the start of their operation, which degrades their electrical characteristics. This time-dependent degradation is called *aging*. In nano-CMOS circuits, the major aging effects are the following.

Bias Temperature Instability (BTI) is mainly caused by the breaking of the Si–H bonds at the interface between the gate dielectric and the silicon substrate under elevated temperature and electric field across the gate dielectric [JMGM12]. The dangling bonds of the interface silicon atoms constitute the interface defects or traps that restrict the movement of charges. These charged traps in the gate dielectric shield the electric field applied at the gate terminal of the transistor. They also reduce the amount of carriers induced in the conducting channel, which leads to the degradation of the current drive capability of transistors.

Hot Carrier Injection (HCI) also leads to the build-up of traps within the gate dielectric. After the transistor channel is formed, the carriers are flowing through it driven by the electric field exerted between source and drain. Part of the carriers may obtain enough energy (become "hot") to overcome the energy barrier at the interface, penetrate into the gate dielectric and eventually generate new defects/traps by kinetic energy transfer at the interface and in the gate dielectric [CVS+14].

Time-Dependent Dielectric Breakdown (TDDB) creates a undesired conducting path through the gate dielectric from the gate to the substrate and thus disables the ability of the gate to control the current between source and drain. Multiple microscopic mechanisms collectively induce the breakdown [WSV09]. Holes injected from gate electrode [HL99] and silicon substrate [TUKT97] may get trapped in the gate dielectric and when the hole density exceeds a critical value, the dielectric becomes conducting through the path formed by the holes. Thermochemical processes break the molecular bonds in the dielectric and create traps which may also form conductive paths [MKSM03]. The hot-carriers injected into the dielectric due to HCI effect generate traps in the dielectric and thus contribute to the breakdown process [CKGdK03] as well.

Electro-Migration (EM) is a well-known aging effect affecting metal wires [Bla69]. At high current density, metal ions are transported away from their original locations due to the momentum transfer from the conducting electrons. The transport of metal materials leads to void or thinning in the wires, which translates to a dramatically increased resistance or ultimately an open circuit. The metal ions that are carried away can also pile up at another location on the wire and eventually form a hillock that touches a neighboring wire, causing a short circuit. In digital circuits, power supply lines are most susceptible to EM because they have to sustain the current in a single direction for a long period [Lie13]. In contrast, signal or clock networks bear current in alternating directions and thus undergo self-healing effect where materials are transported back when current changes direction.

This thesis focuses on the mitigation of BTI and HCI effects since they are the dominating aging mechanisms in nano-CMOS technologies [CVS+14].

In addition to aging, *Single Event Upsets* (SEUs) further threatens the dependability of reconfigurable architectures with SRAM-based FPGAs. SEU originates from the ionizing particles in the nano-electronic devices, which induces transient current pulse in the circuit that may alter the values of logic signals, leading to *soft errors*. The errors are "soft" because they can be corrected by recomputation. SEU affects both combinational and sequential logic. At low operation frequency, the ma-

jority of the soft errors are from sequential logic, whereas at high frequency errors from combinatorial logic become dominant [FCMG13]. Because the functionality of a circuit implemented on an FPGA is defined by the configuration bits stored in the configuration memory, a soft error altering a configuration bit essentially modifies the circuit implementation and this malfunction will persist until the corrupted configuration bit is corrected. A “soft” error in the configuration bits can thus be considered as a “hard” fault in the circuit.

The following subsections provide the necessary physics background for BTI, HCI and SEU effects.

2.3.1 Basic Operation Principles of MOSFET

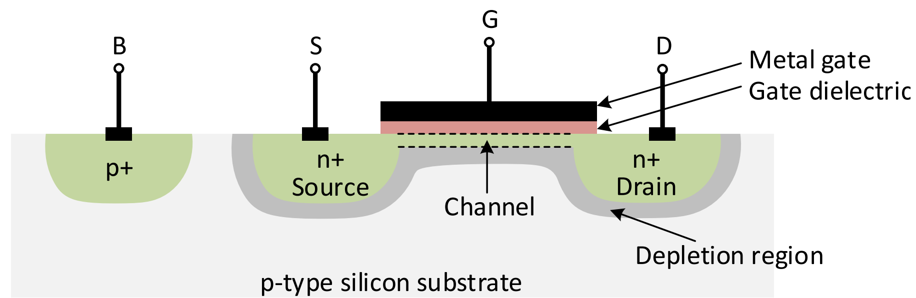

Figure 2.9: Basic structure of a NMOS transistor

The metal-oxide-semiconductor field-effect transistor (MOSFET) is a type of transistor whose conductivity is controlled by the electric field exerted across a insulating oxide between a metal and a semiconductor. Figure 2.9 shows the basic structure of a planar n-type MOSFET (NMOS) in which the current during conduction are carried by electrons, while in p-type MOSFETs positively charged holes are the carriers. A MOSFET has four terminals which are called *body* (B), *source* (S), *gate* (G) and *drain* (D). The source and drain are highly doped n-type regions embedded in the p-type substrate. The depletion region are formed at the junction of the n- and p-type semiconductors. An ultrathin stack of dielectric materials [And12] together with a metal layer is deposited above the channel region between source and drain, forming the controlling gate of the transistor. When the voltage drop between the gate and the source (V_{GS}) exceeds a positive threshold value (V_{th}), a conducting channel is induced in the substrate between source and drain. With another voltage between source and drain (V_{DS}), current flows between them through the channel. The body terminal controls the voltage potential of the silicon substrate. For NMOS, it is generally connected to the ground.

The amount of current I_{DS} flowing through the channel during conduction ($V_{GS} \geq V_{th}$), depending on the voltages at transistor terminals, can be captured as follows [RCN03]:

$$I_{DS} = \frac{\mu_n \epsilon_{ox}}{t_{ox}} \frac{W}{L} \left(V_{GT} V_{min} - V_{min}^2/2 \right) (1 + \lambda V_{DS})$$
$$\text{with } V_{GT} = V_{GS} - V_{th} \text{ and } V_{min} = \min(V_{GT}, V_{DS}, V_{DSAT}) \tag{2.1}$$

where μ_n denotes the electron mobility, ϵ_{ox} the permittivity of the gate dielectric, t_{ox} the thickness of the gate dielectric, W/L the width/height of the channel and λ the parameter for the channel-length modulation which is proportional to the inverse of the channel length. V_{DSAT} represents the drain-source voltage at which the channel electrons reach their maximum velocity, i.e. further increasing V_{DS} will not proportionally increase I_{DS}.

As shown in Eq. (2.1), the amount of current that a transistor is able to drive greatly depends on the difference of V_{GS} and V_{th}. An increase of V_{th} will reduce I_{DS} for the given terminal voltages and slow down the charging/discharging of the node capacitance (i.e. the signal transition rate) in the circuits. The threshold voltage V_{th} is determined by the material properties of the gate dielectric and the silicon substrate and the body bias voltage V_{SB}. As the aging effects slowly deteriorate the constitution of the gate dielectric during the normal operation, V_{th} *shifts over time*, which results in the weakened current drive capability of transistors and degraded circuit performance.

In p-type MOSFETs (PMOS), the type of semiconductor materials is exactly the opposite to that in NMOS, i.e. replacing n-type with p-type and vice versa. The body terminal of PMOS is generally connected to the supply voltage. The threshold voltage of PMOS is negative and it requires $V_{GS} \leq V_{th}$ to become conducting.

2.3.2 Biased Temperature Instability

BTI effect can be attributed to charge trapping at the dielectric/substrate interface and in the gate dielectric. Figure 2.10(a) shows a Transmission Electron Microscopy (TEM) image of the material stack at the gate [AFC+09]. Above the silicon substrate, a two-layer dielectric consisting of HfO_2 and SiO_2 is deposited and TiN constitutes the gate electrode at the top. During manufacturing, the surface of the Si substrate is oxidized to create a thin layer of SiO_2. However, not all surface silicon atoms will establish bonds to oxygen atoms. Some silicon atoms at Si/SiO_2 interface still have one dangling bond, which is a kind of surface defects that will impair the performance of transistors. To reduce the interface defects, the Si/SiO_2 interface is further processed by annealing with *forming gas* [RP88], a mixture of

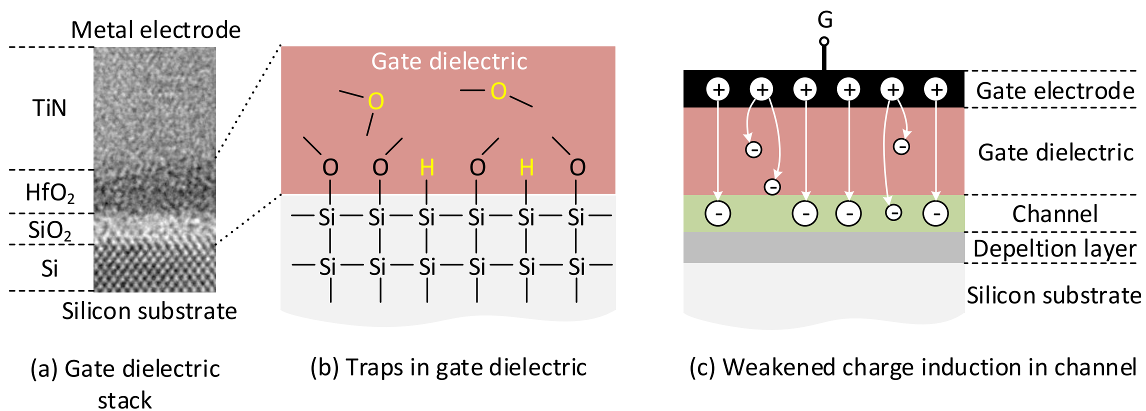

Figure 2.10: Physical mechanism of BTI in NMOS

hydrogen and nitrogen. The dangling silicon bonds are then passivated by forming Si–H bonds [Ent07].

During the operation of the transistor, the Si–H bonds at the Si/SiO_2 interface and the bonds of oxygen atoms within the gate dielectric may get broken, which generates new traps at the original locations of hydrogen and oxygen (see Fig. 2.10(b)). These are thought to occur under the combined act of the electric field across the dielectric, temperature and the interaction with electron/holes tunneled from the channel [Mah15]. These newly generated traps together with the traps pre-existing [JMGM12] in the oxides immobilize the electrons/holes in NMOS/PMOS in the gate dielectric, as illustrated in Fig. 2.10(c) for an NMOS. Intuitively, for a given gate voltage, these trapped charges shield part of the electric field applied at the gate (the field lines ending at the trapped charges instead of in the substrate), which reduces the amount of charges can be induced in the transistor channel and thus lowers the channel current during conduction. This weakening of current drive capability can be captured by the increase/decrease of threshold voltage (ΔV_{th}) for NMOS/PMOS.

Although the underlying mechanisms of BTI are still under active research [Mah15], state-of-the-art physics-based model of BTI is used in this thesis and is presented in Section 2.4.3.

2.3.3 Hot Carrier Injection

Very similar to BTI, HCI is also ascribed to the trap generation at the Si/SiO_2 interface, but due to different root causes and occurring only near the drain region. When a transistor is dynamic switching, current is flowing through the channel, as shown in Fig. 2.11 for an NMOS, where the source is connected to ground (GND) and the gate and the drain are connected to the power supply (V_{DD}). The lateral electric field between source and drain accelerates the electrons along the channel, which can gain sufficiently high energy near the drain to leave the substrate and tunnel into the gate dielectric. These high energetic ("hot") electrons can interact with the interface Si–H bonds and disassociate them, generating new interface traps [CVS+14]. In PMOS, hot holes are responsible for the trap generation. The HCI aging model used in the thesis is provided in Section 2.4.3.

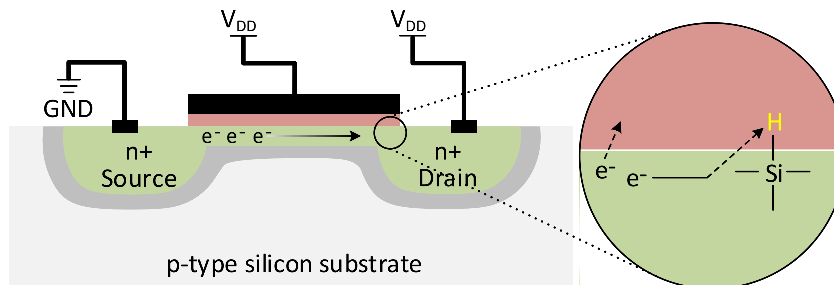

Figure 2.11: Physical mechanism of HCI in NMOS

2.3.4 Single Event Upset

Radiation emitted from the environment in which nano-electronic devices is operating may disturb the electrical states of transistors and generate a transient current in the circuit. In terrestrial environment, three types of radiation sources are responsible for this disturbance [Bau05]: 1) alpha particles emitted by trace-level radioactive impurities, e.g. uranium and thorium, in the packaging materials of chips, 2) secondary neutron flux resulted from the impact of high-energy cosmic rays with earth's atmosphere and 3) low-energy neutrons from the cosmic background.

These three types of radiation create similar ionizing events in a transistor, as shown in Fig. 2.12 (a). When a ionizing particle passes through a transistor, a high concentration of electron-hole pairs is formed along the path of the particle. The ionizing particle can be 1) an alpha particle, 2) a reaction product from the interaction of the neutron flux and the chip materials, or 3) the secondary radiation due to the reaction between low-energy neutrons and doped boron, for the three above radiation sources, respectively. When the ionizing path is close to a reverse-biased junction (Fig. 2.12(b)), e.g. the depletion region at the drain with $V_{DB} = V_{DD}$, the carriers are promptly collected by the electric field at the junction, creating a large current pulse at the circuit node that corresponds to the drain. The depletion region is distorted in the charge collection process and forms a funnel shape which penetrates deep into the substrate and dramatically enhances the collection efficiency [HMO81]. After this phase, the charges are slowly collected by carrier diffusion (Fig. 2.12(c)), until all remaining excess charges have been collected or recombined. Figure 2.12(d) shows the resulting current pulse of these three phases.

The amount of radiation, in particular of the neutrons provoked by cosmic rays, is a strong function of altitude, geomagnetic location and solar activity [JED06]. The thickness of the atmosphere decreases the intensity of cosmic rays reaching the sea level. For example, going from sea level to the cruising altitude of commercial flights (around 10 km), the neutron flux increases 500× (Figure B.1 shows the soft error rate at different terrestrial altitudes). The geomagnetic field of the earth deflects charged cosmic particles back into space, reducing the impact chance of the cosmic rays with the atmosphere. The neutron flux is about 6× lower near the geomagnetic equator than locations near the geomagnetic pole (see Fig. B.2). The solar activity

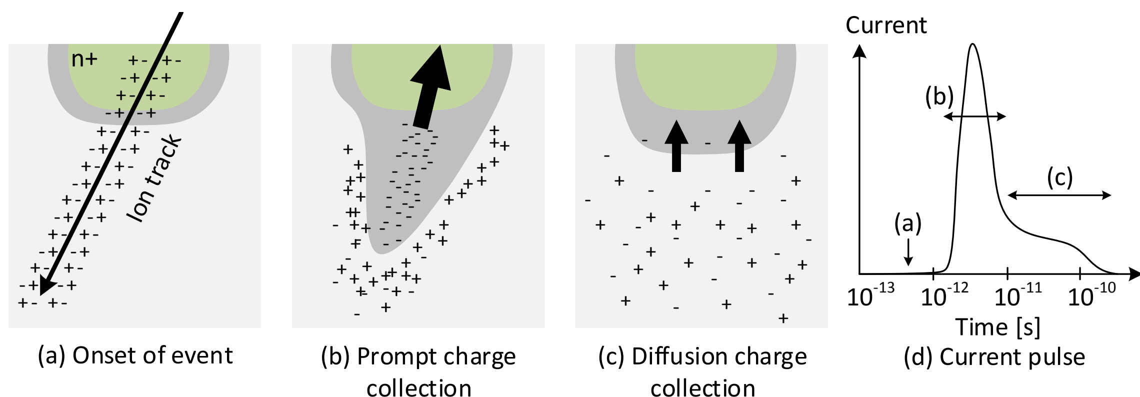

Figure 2.12: Physical mechanism of SEU in MOSFET [Bau05]

modulates the magnetic field in the near-earth space. The neutron flux is lowest when the sun is most active, e.g. with sunspots and strong solar wind plasma.

Depending on the affected site of particle striking in the circuit, the current pulse has different impact on the circuit operation [FCMG13]. If the current pulse is generated at a node in combinational logic, it will cause a transient voltage change, i.e. a change of logic value, which may propagate along the gate chain, depending on whether this logic change is masked by the logic values at other nodes in the circuit. For instance, if a transient change of logic value occurs at one input of a AND-gate while the other input of the gate holds the value 0, this transient event will have no influence in the fan-out cone of this AND-gate because its output produces a 0 regardless of the change at the affected input. On the other hand, if a register is affected, its stored logic value may flip, leading to a transient erroneous logic state, so-called *soft error*. This soft error in the register will not persist since its value will be set correctly during the computation in the next clock cycle. However, the error may propagate through the rest of the circuit, producing wrong data in the datapath and wrong states in the Finite State Machines (FSMs).

Similarly, the value stored in a SRAM cell may also be altered upon particle striking, which is particularly critical for SRAM-based FPGAs. If the affected SRAM cell stores a configuration bit for a functional module implemented on the FPGA, the soft error essentially modifies the circuit structure of the module (see Section 2.1.5). This error will persist until the module is reconfigured, i.e. rewriting the corresponding configuration memory with correct data. Therefore, a *soft error* in the configuration memory can be considered as a *permanent fault* in the mapped circuit.

2.3.5 Recent Advancement in Aging

In contrast to the common sense that aging is a long-term process that takes place over a timespan of years, BTI effects could also degrade the transistors in sub-microsecond timescale, so-called *instantaneous aging effects*, as studied by van Santen *et al.* [vAMM+16]. As a result, the guardband introduced to protect against the long-term aging effects may be violated frequently due to instantaneous aging effects in short-term. In contrast to long-term effect, short-term aging component is dependent on the transistor switching frequency. A lower switching frequency causes a higher instantaneous increase in the threshold voltage. A novel aging model incorporating instantaneous and long-term BTI effects is proposed in [vAMM+16] as well, which enables circuit designs with reduced guardband. The proposed model also allows the computation of maximum degradation solely based on duty cycle and switching frequency instead of input waveforms, the efficiency of which enables the physics-based stress estimation of large circuits.

Cross-layer approaches for circuit design have shown promising results, further reducing the pessimistic guardband. Amrouch *et al.* [AKGH16] proposes a reliability-aware design method, where circuit-level signal characteristics and physical-level degradation effects are combined. The standard cell library is extensively expanded to include stress-aware cells, i.e. cells with different delay and aging characteristics under different stress conditions. Given the input signal properties like signal prob-

abilities and slew rates, the circuit synthesis tool is able to optimize the delay of circuits under consideration of aging effects.

Under near-threshold and super-threshold computing conditions, random telegraph noise (RTN), process variation (PV) and BTI play major roles in the reliability of transistors. By capturing the underlying physical origin of threshold voltage degradation, i.e. charging/discharging defects, the authors of [vMMA+17] propose a unified aging model for combined RTN and BTI effects. In addition, the combination of probabilistic shift of transistor geometries due to PV allows a probabilistic reliability estimation for near-threshold and super-threshold circuits.

Besides the degradation of threshold voltage, BTI also impact other physical and electrical characteristics in a transistor such as charge mobility, gate-drain capacitance and sub-threshold slope, as shown in [AMv+17]. It reveals that solely considering the threshold voltage degradation alone leads to an exaggerated power estimation of an aged circuit. A joint analysis of other degradation components is necessary for more accurate power estimation.

2.4 Fault, Stress and Aging Models Used in the Thesis

To be able to algorithmically evaluate the dependability states in the system, e.g. degree of aging, correctness of circuit operation and lifetime, numerical models for the dependability issues discussed in Section 2.3 are required. This section provides such models used in this thesis to evaluate when a circuit is considered to be faulty; how the degree of aging is measured and how the lifetime of a circuit is determined.

2.4.1 Stress Model for Aging Effects

Aside from material constants, aging mainly depends on three non-material factors: supply voltage, temperature, and transistor activities. For the algorithms in our proposed method we use a simplification that focuses on the aging effects induced by transistor activities. That is a reasonable approximation as reconfigurable accelerators are typically operated in a static voltage domain (i.e. no dynamic voltage scaling) and do not show high temperature variation (they are often optimized for data-level parallelism and thus run at rather low frequency [KR07] with correspondingly low power density [NR11, VHL+05]).

The term *stress* is defined as the condition under which a transistor is experiencing electrical and physical degradations. An example for such a degradation is the threshold voltage shift ΔV_{th}, which may eventually cause a failure of the circuit. Different aging mechanisms exist that lead to threshold voltage shift. For instance, Fig. 2.13 shows the increase of threshold voltage due to HCI that depends on the transistor toggling rate. In this work, the *Mean Time to Failure* (MTTF) of a transistor is defined as the time until its threshold voltage exceeds a certain critical value. Reducing the threshold voltage shift by reducing the stress increases the MTTF.

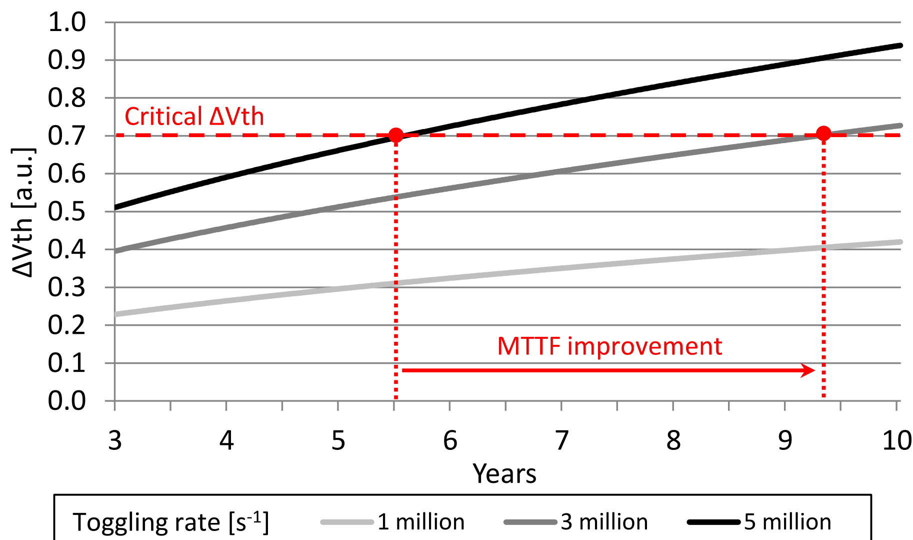

Figure 2.13: Threshold voltage increases over time due to HCI under different toggling rates.

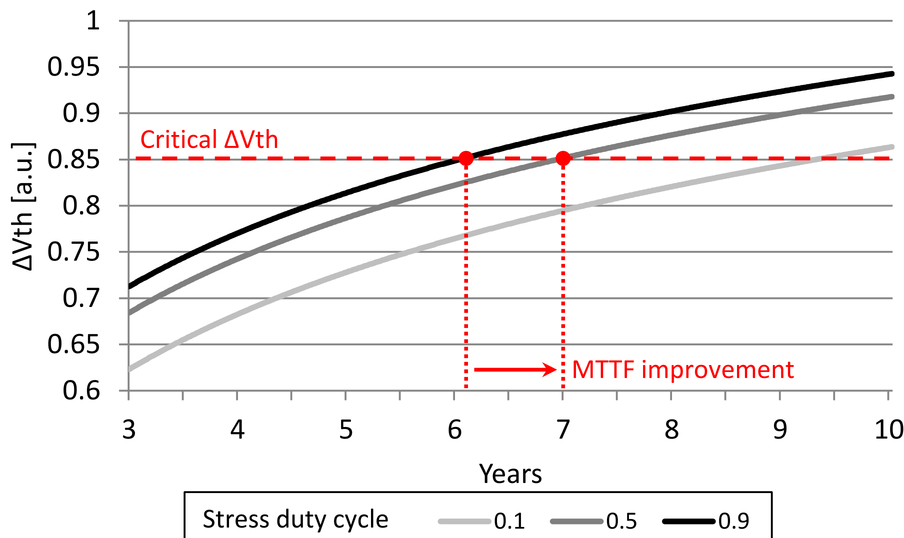

Figure 2.14: Threshold voltage increases over time due to BTI under different stress duty cycles.

In the following, two types of stress in nano-scale CMOS circuits are distinguished: static stress and dynamic stress. A transistor is under *static stress* when an electric field is exerted across its gate oxide to induce a conducting channel. Static stress is characterized by the *stress duty cycle*, i.e. the fraction of operation time that a transistor is conducting. Reducing the stress time $stress_{stat}$ also reduces the stress duty cycle, which leads to an increase of the transistor's MTTF, as shown in Fig. 2.14. Instead, a transistor is under *dynamic stress* when current flows between its source and drain. Dynamic stress is characterized by the toggling rate, i.e. the ratio of number of toggles and total operating time. Reducing the number of toggles $stress_{dyn}$ also reduces the toggling rate, which leads to an increase of the transistor's MTTF, as shown in Fig. 2.13. Static stress leads to aging effects like BTI, while dynamic stress leads to aging effects like HCI.

2.4.2 Stress Properties

There are different models for these aging effects, e.g. [SKM+08, LQB08, SWSC10, CVS+14, AvE+14], and they all indicate that in the long term the transistor degradation *monotonically* increases with $stress_{stat}$ or $stress_{dyn}$ for static or dynamic stress, respectively. For instance, $\Delta V_{th}(stress_{stat1}) > \Delta V_{th}(stress_{stat2})$ when $stress_{stat1} > stress_{stat2}$ under the same supply voltage and temperature [CVS+14, AvE+14]. In other words, the aging effects are reduced when $stress_{stat}$ or $stress_{dyn}$ is reduced.

In addition, $stress_{dyn}$ is generally considered as *additive*. For instance, the dynamic stress of two different workloads corresponds to the number of toggles that these workloads impose on a transistor. Intuitively, the combined dynamic stress is the sum of these toggles, which is proportional to the amount of charge transported between drain and source [MMM+13, AvE+14]. In general, the total stress experienced by a transistor under different workloads ($stress_{dyn}(work_1 + work_2)$) is the sum of stress experienced under the individual workloads ($stress_{dyn}(work_1) + stress_{dyn}(work_2)$). In the long term, this argument also holds for $stress_{stat}$. Actually, BTI aging may experience a *recovery effect*, but that requires complex conditions or long relaxation periods [GBS14] and will thus hardly affect the additive property. The monotonic and additive properties of $stress_{stat}$ and $stress_{dyn}$ allow a simplified consideration of CLB stress during decision making (see Chapter 6) rather than evaluating complex aging models at runtime. In the rest of the thesis, simply "*stress*" is referred to when there is no need to explicitly differentiate between static and dynamic stress.

2.4.3 Aging Models

In this thesis, state-of-the-art physics-based aging models for BTI and HCI effects are employed to evaluate the threshold voltage shift of transistors depending on stress, temperature and time. The BTI aging model is adopted from [JMGM12, AvE+14]:

$$\Delta V_{th} = \frac{q}{C_{ox}} (\Delta N_{IT} + \Delta N_{ET} + \Delta N_{OT}) \tag{2.2}$$

$$\text{where } \Delta N_{IT} = A(V_{GS} - V_{th0} - \Delta V_{th})^{\Gamma_{IT}} e^{-\frac{E_{AIT}}{kT}} d^{\frac{1}{6}} t^{\frac{1}{6}} \tag{2.3}$$

$$\text{with } d = \frac{\Lambda}{1 + \sqrt{\frac{1-\Lambda}{2}}}, \tag{2.4}$$

$$\Delta N_{ET} = B(V_{GS} - V_{th0} - \Delta V_{th})^{\Gamma_{ET}} e^{-\frac{E_{AET}}{kT}} \Lambda \tag{2.5}$$

$$\text{and } \Delta N_{OT} = C(1 - e^{-(\frac{t}{n})^{\beta_{OT}}})\Lambda \tag{2.6}$$

$$\text{with } n = \eta(V_{GS} - V_{th0} - \Delta V_{th})^{\frac{\Gamma_{OT}}{\beta_{OT}}} e^{\frac{E_{AOT}}{kT\beta_{OT}}}, \tag{2.7}$$

where T denotes temperature, t operating time and Λ stress duty cycle. A, B, C are fitting parameters depending on the manufacturing process [JMGM12]. ΔN_{IT}, ΔN_{ET} and ΔN_{OT} represents the number of interface traps, pre-existing oxide traps and generated oxide traps, respectively (see Section 2.3.2). Device dependent pa-

rameters such as C_{ox} and V_{th0} are extracted from the PTM 22-nm HKMG model [PTM]. The supply voltage is 1.0 V. All other model parameters are as in [JMGM12].

The HCI aging model is adopted from [MMM+13, AvE+14] and [HTH+85]:

$$\Delta V_{th} = \frac{q}{C_{ox}} D \left(t \frac{r Q_{DS}}{W} e^{-\frac{\phi_{it}}{q \mu E_m}} \right)^{\frac{1}{2}}, \tag{2.8}$$

where t denotes operating time, r the toggling rate, and Q_{DS} the amount of charges flowing through the channel during one toggle calculated using PTM 22-nm HKMG model [PTM]. ϕ_{it} and E_m are extracted from [HTH+85]. μ denotes the mean free path for optical-phonon–electron collision. It is temperature dependent and is calculated based on [NOY77]. W is assumed to be the same as the transistor channel length. D is a fitting parameter depending on the transistor geometry and manufacturing process [AvE+14].

2.4.4 Fault Model for Soft Errors

Soft errors in the configuration memory caused by SEUs are stochastic processes. The location (i.e. which bits), and the occurrence time of a soft error are non-deterministic. Two types of soft errors are distinguished: *Single-Bit Error* (SBE) and *Multi-Bit Error* (MBE). An SBE occurs when a single event upset corrupts the data of only one configuration bit. A single event upset may also affect multiple neighbouring memory cells and cause multiple erroneous configuration bits, which corresponds to an MBE. Experiments in the flight tests [QGM+13] and in the radiation chambers [WTH14] showed that the dominating soft errors were SBEs.

The susceptibility of the configuration memory of an FPGA to single event upsets is typically reported by the device vendor as *Soft Error Rate* (SER) [Xil16b]. It is measured in FIT/Mb (Failure In Time per Megabit) and provides the expected number of erroneous configuration bits, including both SBE and MBE, in an FPGA operating for 10^9 hours. Under the assumption that the error probabilities of individual configuration bits are equal and constant in time, the SER λ of one configuration bit can be derived. For instance,

$$\lambda \text{ in number of errors per second} = \frac{\text{SER in FIT/Mb}}{3600 \cdot 10^9 \cdot 10^6}. \tag{2.9}$$

A continuous-time Markov model can be used to describe the stochastic behaviour of the soft error of a configuration bit [SKM78]. A configuration bit is defined to have two distinct states: error-free and erroneous. Without recovery mechanisms (e.g. scrubbing, see Section 2.5.3), once a configuration bit is erroneous, it stays erroneous. The transition rate from error-free state to erroneous state is the SER of one configuration bit λ. The probability that an error-free configuration bit at time t_0 remains error-free till time $t_0 + t$ then follows the exponential distribution:

$$P(\text{error-free at } t + t_0 | \text{error-free at } t_0) = e^{-\lambda t}. \tag{2.10}$$

The *critical bits* of an accelerator are those configuration bits that define its functionality [Le12]. A functional failure of the accelerator may occur if those bits are altered due to SEU. In this thesis, it is conservatively assumed that an accelerator operates correctly only when all of its critical bits are error-free. When any critical bits of an accelerator are erroneous, the outputs of the accelerator are not trusted anymore and the accelerator is determined to be faulty. If an accelerator A is reconfigured at t_0, or scrubbed at t_0 without errors, all of its critical bits are verified to be correct at t_0. Then the reliability $R(A,t)$ of the accelerator at $t > t_0$, i.e. the probability that it produces correct outputs throughout the time period from t_0 to $t_0 + t$, is equal to the probability that none of its critical bits is affected by soft errors from t_0 to $t_0 + t$:

$$R(A,t) = \prod^{n} e^{-\lambda t} = e^{-\lambda n t}, \quad (2.11)$$

where n the number of critical bits of the accelerator and t is called the *resident time* of the accelerator. It is assumed here that the soft errors of individual configuration bits are independent of each other, regardless of whether they are SBEs or MBEs. Therefore, Eq. (7.1) overestimates the errors due to MBEs, since one MBE is counted as multiple independent SBEs in multiple configuration bits. However, this overestimation is marginal because SBE is the dominating error type over MBE.

Functionally used memory elements in FPGAs, i.e. block RAMs and flip-flops, are not protected by scrubbing, but are implicitly protected if modular or temporal redundancy is employed (see Section 2.5.2). Furthermore, block RAMs are readily protected by ECC [Xil16a] combined with physical interleaving in the recent FPGA generations, which essentially reduces their soft error rate well below their permanent failure rate [HS15].

Compared to the number of flip-flops contained in an FPGA, the amount of configuration bits is higher by two to three orders of magnitude. For instance, a configurable logic block in a Xilinx UltraScale FPGA has 8 flip-flops [Xil15a] and multiple 3936-bit configuration frames [Xil15b]. Also, flip-flops are not susceptible to upsets throughout the entire clock cycle, and not every upset leads to an error observed by the system or user. The time during which data in memory are vulnerable is bound by the duration of the accelerator execution (in the order of cycles), which is much smaller than the resident time of configuration bits of accelerators (in the order of million cycles). Therefore, soft errors in block RAM and flipflops are not considered in this thesis.

2.5 Basic Dependability Techniques

The dependability of FPGA-based reconfigurable architectures are threatened by latent faults not detected during manufacturing, emergent faults during system operation due to aging effects and transient faults due to single event upsets. This section provides an overview of basic dependability techniques for the detection, isolation and recovery from these faults.

2.5.1 FPGA Test and Diagnosis

To test a digital circuit whether it operates as expected or rather if any faults reside in the circuit, a set of input data (*test vectors* or *test patterns*) is fed to the input port of the circuit, while the operation results (*output response*) at the output port are observed and checked against the expected or specified values. Figure 2.15 shows a general test structure, where a *Test Pattern Generator* (TPG) generates the test vectors to exercise the *Circuit Under Test* (CUT) and an *Output Response Analyzer* (ORA) collects the output data from the CUT to find any anomalous behaviors in the CUT. The test control and configuration logic manages the whole test sessions, i.e. start/stop/reset of the test sessions, test pattern selection for TPG, configuration of operation modes for CUT and the selection of appropriate analyze method for ORA.

Built-in-Self-Test

In conventional manufacturing test, the test facilities (TPG, ORA and test control) are provided by external test equipments. During test, probes of the test equipment are attached to the pins of the CUT to exercise the circuit and collect the responses. This method has several drawbacks:

- The test speed is slow. Since the clock of the circuit is externally driven by the test equipment, it has to drive the large capacitance of the pin and thus cannot achieve high frequency. In addition, the circuit may be designed to run at a higher frequency than that achievable during the test, which reduces to the chance to detect delay faults.
- The test equipment can only access the circuit by its pins. Information of internal nodes can only be derived from the observation at external pins. This limits the controllability and observability of the circuit and compromises the test quality.
- Test after deployment of the circuit is very difficult, as the circuit needs to be dismounted from the system and sent back to the manufacturer, which causes high monetary and temporal costs.

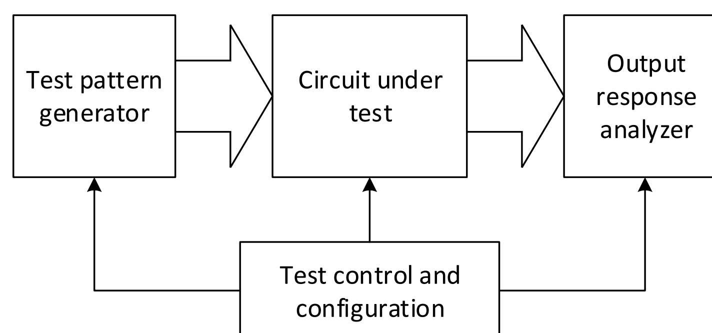

Figure 2.15: A general test structure for digital circuits

Thanks to the technology scaling, more logic functions, even including the test facilities, can be integrated into one chip, while incurring only a negligible area cost [Ste00]. *Built-In-Self-Test* (BIST) is a design-for-testability technique in which testing, including test generation, test application, test results analysis and test control, is accomplished through built-in hardware features [AKS93]. BIST addresses all the above drawbacks of manufacturing test and has since long become a standard feature in digital circuits [McC85], i.e. self-test is one of the functions of a chip.

With BIST, the circuit can be tested *at-speed*, i.e. at the same frequency as the nominal operation frequency of the circuit, which shortens the test time and improve the test coverage of delay faults. Additional control and observation points can be added to the internal node of the CUT during the design of the BIST circuitry, which significantly improves the testability of a design [Ste00]. As no external equipment is required for testing, the circuit can be easily put into test mode *in the field* to detect the faults that may occur during the system operation.

The reconfigurability of an FPGA allows to implement BIST in an FPGA even with zero area overhead [SKCA96]. To cover all reconfigurable resources in an FPGA, part of the FPGA is used to implement the BIST circuitry as the rest part of the FPGA is being tested. In a second test session, the resources that was used for BIST is tested. In this way, the whole FPGA is tested without additional BIST circuitry.

Array-Based FPGA Test

In FPGA-based reconfigurable architectures, the functionality implemented on the FPGA changes during runtime and may be unknown during the design of test facilities. Therefore, structural test of the reconfigurable fabric is typically employed for testing FPGAs [AS01], where the underlying reconfigurable resources, i.e. CLBs and PIPs, are tested independent of the functionalities that will be implemented on them. However, each CLB contains distinct types of primitive logic elements, e.g. LUTs, multiplexers and registers, each of which requires a different test approach.

Meanwhile, the reconfigurable resources are arranged regularly in a two-dimensional array, where each entry in the array is identical across the whole FPGA (see Section 2.1). Exploiting this regularity, an array-based test approach can greatly simplify the test procedure, as proposed in [Ren98]. The test procedure to test an array of CLBs is divided into multiple test sessions, where in one session only a single type of logic primitives in CLBs is tested. The logic primitives are chained into the form of an *Iterative Logic Array* (ILA) [Fri73, HMCL98], where each primitive receives the output from the previous primitive as one of its inputs and delivers its output to the next primitive in the chain. Besides, primitive also receives a set of common global inputs for additional controllability. Figure 2.16 shows three such ILAs for the testing of a 3×5 CLB array.

By deliberately configuring the logic function of the primitives, a single fault in any of the primitives in the array can be detected, i.e. the resulted erroneous output of any single faulty primitive in the array is guaranteed to be propagated to the ORA.

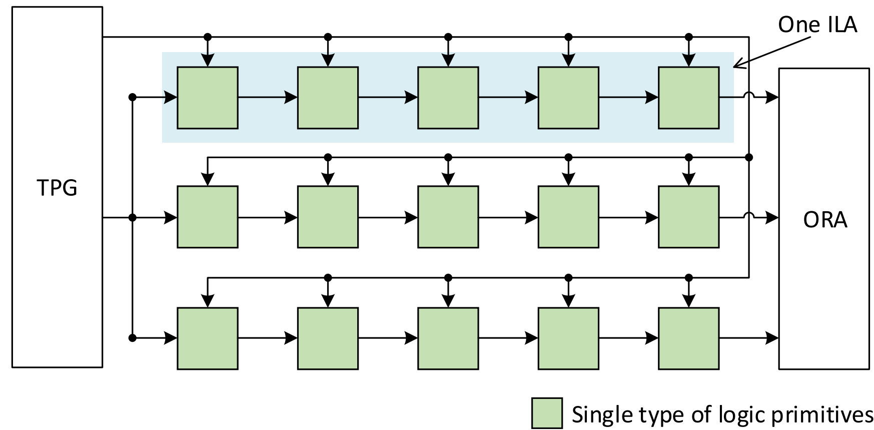

Figure 2.16: Array-based test structure for FPGAs

This kind of configuration of the FPGA that enables fault detection is called a *Test Configuration* (TC). And such an ILA is called *C-testable* [Fri73, HMCL98], where the number of required test vectors to perform an exhaustive test is independent of the number of primitives in the array. For instance, to exhaustively test the LUTs with n inputs, 2^n test vectors are to be applied during each of the two test configurations, in which the LUTs are configured to implement XOR and XNOR logic functions [ABB+12].

In general, the array-based structural test approach is performed as illustrated in Fig. 2.17. Each type of the reconfigurable resources in the CLBs is tested separately. After selecting the type of resource to be tested, a set of test configuration is applied, which configures the logic primitives to be tested into a C-testable ILA. For each test configuration, exhaustive test patterns are applied. After the logic primitives are tested in all test configurations, the test process for the next type of resource starts till all resource types are tested.

While the configurable interconnect structures are to some extent already exercised during the test of other logic resources, dedicated interconnect test based on multiple test configurations can be applied in order to achieve a high test coverage [SWHA98, SXCT00, TM05]. To tackle the complexity of the reconfigurable interconnect circuitry, a large number of test configurations is required, which can be reduced by testing only the interconnect resources used by the configured functional module [Tah03, AKL12].

Fault Diagnosis in FPGAs

To localize the faults upon detection during test, i.e. to determine in which CLB the faults are present, test data analysis and dedicated test configurations are necessary. Typical fault diagnosis techniques perform multiple overlapping column- and row-based tests to find the coordinates (column and row indices) of faulty CLBs [AS01, MDS06, ESA07]. For example, in the first test session the CLBs are tested column-wisely, e.g. by configuring CLB columns into ILAs, so that it can be found in which column the faults exist. In the second test session, the CLBs are tested row-wisely

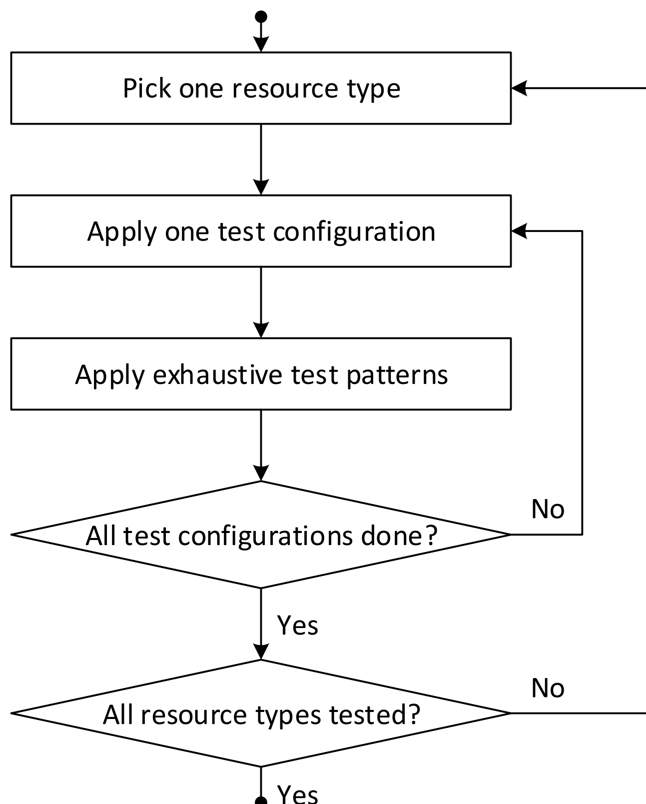

Figure 2.17: Array-based test process for FPGAs

and the faulty CLB rows are determined. Combining the results from the two test sessions, the coordinates of the faulty CLBs can then be obtained.

A recent approach [AD16] generates a set of alternative configurations for the functional module to be implemented on the FPGA. Different configurations use logic resources at different locations but all deliver the same functionality. By applying functional test to these configurations, the locations of faulty resources can be derived by conjugating the set of used resources of the configurations that do not pass the test.

2.5.2 Concurrent Error Detection in FPGAs

In safety- and mission-critical system, one computation error may lead to life-threatening accident or heavy financial loss, in particular in the case of soft errors caused by SEUs. Such errors should be immediately detected, isolated or corrected on-site to prevent it from propagating to other parts in the system. In other words, the error detection takes place concurrently with the functional operation, which is called *Concurrent Error Detection* (CED) [Muk08].

Typical CED techniques introduce certain degree of redundancy for the comparison of the output from the main circuit with the expected output from a predictor which can be identical as the main circuit. A mismatch in the comparison results indicates that computation process was compromised and the produced output is in-

correct. The redundancy for CED can be contributed in space (spatial redundancy) where the outputs from replicated circuits performing the identical computation are compared, or in time (temporal redundancy) where the results from the same computation performed consecutively by the same circuit are compared. Information redundancy embeds additional bits describing data characteristics (e.g. parity bits) into the data being processed to identify any anomalies during computation.

Since SEUs mainly affect the configuration memory of FPGAs and the corrupted configuration bits persist before a reconfiguration, as discussed in Section 2.3.4, a recomputation (temporal redundancy) will not detect the output mismatch except in the rare case when SEU occurs exactly between these two computations. Therefore, to concurrently detect soft errors in FPGAs, spatial and information redundancy are typically adopted [SSC08].

Spatial Redundancy

The most widely used redundancy technique for error detection *and* correction in FPGAs is *Triple Modular Redundancy* (TMR) [Car06]. As shown in Fig. 2.18, three replicas of the same circuit are fed with a common input and their outputs are compared by a majority voter. If any two or all of the three outputs are equal, the majority output is forwarded as the final result. For example, when Output A = Output B ≠ Output C, Output A or Output B is taken as the output of the voter. The replica producing the minority output which disagrees with the other two is identified as corrupted.

In the presence of a single corrupted replica, the TMR can continue producing correct outputs, and thus error detection and correction are simultaneously achieved. The corrupted replica can be repaired by partial reconfiguration with correct bitstream, without interrupting the operation of the other two replicas and the voter. Only when all of the three outputs disagree (i.e. two of the replicas malfunction), error correction is not possible as no correct output can be derived by the voter. In this case, all three replica need to be reconfigured to recover from error and the system operation has to be stalled. Such as case can occur when a corrupted replica is not promptly repaired after error detection and more soft errors accumulate in the configuration memory until another replica become broken as well. To prevent the

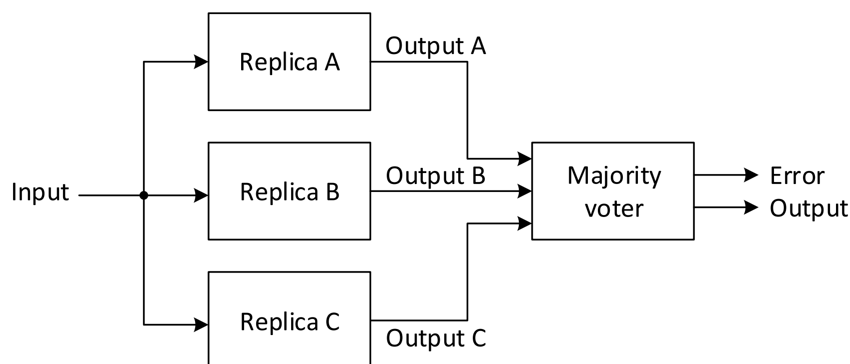

Figure 2.18: Triple modular redundancy

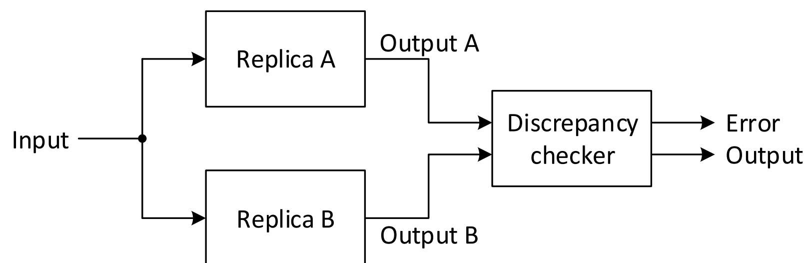

Figure 2.19: Duplication with comparison

error accumulation, a complementary technique called *scubbing* (see Section 2.5.3), which periodically checks the correctness of the configuration data, is often employed in addition to TMR.

Another common design of spatial redundancy for error detection is *Duplication With Comparison* (DWC), as shwon in Fig. 2.19. It consists of two replicates of the same circuit, which are fed with the same input. The outputs of both replicas are compared by the discrepancy checker that assert the error signal upon output mismatching. If no error is detected, one of the two identical outputs is forwarded as the final result. If error is affirmed by the discrepancy checker, it implies that one of the two replicas or both malfunction. It is not possible to derive which replica is corrupted solely based on the comparison of the two outputs. To recover from the error, both replicas need to be reconfigured to correct their configuration bits and the failed computation has to be re-executed. [IPD10] proposes to use a SEU-resistive hardcore processor executing a software version of the replicas to diagnose which replica needs to be repaired.

An important consideration when applying spatial redundancy is the high area overhead and power consumption resulted from the replicated functionality. TMR and DWC require at least 3× and 2× more area and power than an unprotected circuit, respectively. An over-protected system provides high reliability beyond the application requirements, whereas the contributed area and power cost could have been saved or diverted to increasing performance. The trade-off between spatial redundancy and performance in high-reliable reconfigurable architectures is addressed in Chapter 7.

Information Redundancy

While spatial redundancy protects the system during computation against circuit malfunction due to soft errors in the configuration memory, information redundancy efficiently protects the user data during transmission or storage in the memory which is susceptible to SEUs. For each block of user data, a small number of bits called *Error Correcting Code* (ECC) is calculated based on the individual bits in the user data. The ECC is then attached to the corresponding data block and transmitted or stored jointly with the user data. On data retrieval, the ECC bits are obtained together with the user data bits and they are verified against each other to check if any bits are modified during the transmission or storage.

A parity bit is the simplest from of ECC [Muk08]. It indicates whether the number of 1's in a data block (typically 1-byte long) is even or odd. If one of the bits in the data block including the parity bit is flipped, e.g. due to SEU, the parity bit will not match the parity of the data block, which indicate an error in the transmitted or stored data. However, with parity bits alone, errors can only be detected but not corrected.

To enable error correction, more complex coding techniques are required [Muk08], such as widely used *Reed-Solomon codes* for the error correction in Bluray discs and *Hamming codes* in server-grade memories. In Virtex-5 FPGAs, each configuration frame is protected by a Hamming code [Xil12c] which allows the detection of any two-bit errors and the correction any single-bit errors in one frame. In addition, interleaving of memory cells makes physically neighboring configuration memory cells belong to logically separate configuration frames [HS15]. This prevents more than two bits that may be upset by a single particle strike from residing in one frame, as more than two erroneous bit may not be detected by the Hamming codes.

2.5.3 Scrubbing of Configuration Memory

Scrubbing is another technique for SEU mitigation in FPGAs [BPP+08]. It periodically scrutinizes the configuration memory to ensure that any erroneous configuration bits will be corrected as soon as possible. It can be categorized into two types: blind scrubbing and readback scrubbing. Blind scrubbing uses a "golden" bitstream that are secured in an SEU-protected (e.g. with ECC) external memory to constantly overwrite the configuration memory of the FPGA via an external configuration port. Readback scrubbing utilizes the internal configuration port to read the configuration data frame by frame and checks for error with the help of the contained ECC bits (see Section 2.1.5). If it is a single-bit error, the frame will be immediately repaired with ECC and written back to the configuration memory. If multiple bits are corrupted in one frame, the ECC may not be able to repair the errors or detect them at all in the first place. The ability to detect error allows to record statistical data about soft errors, which can be used to estimate the soft error rate, e.g. number of flipped bits per month, experienced by the FPGA (see Section 3.2.2).

Scrubbing poses as a periodical non-functional workload to the configuration port of the FPGA besides the partial configuration of accelerators in reconfigurable architectures. Both share the access to the configuration port. Conflicts may occur when scrubbing and accelerator configuration are to be executed at the same time or when blind scrubbing overwrites the frames that are just being reconfigured to implement a different accelerator. Therefore, the write-access to the configuration data should be carefully scheduled to prevent the conflict between scrubbing and partial reconfiguration [HSWK09].

For each frame, there is a finite period of time between two consecutive scrubbing cycles. Configuration bits corrupted during this time period may lead to circuit malfunction. Concurrent error detection mechanisms such as TMR are thus typically coupled with scrubbing to achieve seamless protection for highest reliability.

2.6 Related Work

After an overview of state-of-the-art reconfigurable architectures is given, this section discusses related work for dependability improvement in FPGA-based runtime reconfigurable architectures. If excessive stress is imposed on the resources in the reconfigurable fabric, permanent faults may emerge. Such faults can be detected and localized by online test for FPGAs. Once located, fault tolerance and recovery methods can be applied to avoid using faulty resources. These methods are typically based on remapping to spare resources by partial reconfiguration. Soft errors in the configurable memory are typically defended by modular redundancy and scrubbing. The induced overhead for error detection and correction is reduced by identifying different vulnerability in different parts in the circuit and then selectively applying fault tolerance techniques.

2.6.1 FPGA-Based Reconfigurable Architectures

Reconfigurable computing [VS07, CH11] aims to deliver the hardware-level efficiency for computation with the software-level flexibility for programming. Customized reconfigurable structures, such as [WA10, LSV06], provide optimized organization of reconfigurable resources for the deep coupling with processors and for supporting different computation/communication patterns for the hardware acceleration. However, off-the-shelf FPGAs have attracted large interest for building the reconfigurable fabric, because they as commercialized/verified products are easily accessible from the market and are available in different product models targeting different user groups. Besides, technical and tools support from FPGA vendors also eases the development effort.

Dennl *et al.* [DZT12] propose an FPGA-based acceleration method for SQL queries. An SQL query is mapped to a processing pipeline consisting of hardware modules that corresponds to the basic operators in the SQL statement. By partial reconfiguration, the operator modules can be reconfigured to implement different SQL operators during runtime.

Oetken *et al.* [OWTK10] propose an FPGA-based SoC with a tightly coupled reconfigurable fabric which is partitioned into multiple slots. Accelerators of different sizes can be composed using different numbers of neighboring slots. In this way, the area utilization rate of the reconfigurable fabric is increased. The communication channels among accelerators and with the processor are established with the ReCoBus [KHT08] that uses interleaved multiplexer chains to reduce resource overhead for the interconnection of multiple reconfigurable accelerators. In [YWW+14], a similar approach is proposed to merge physically neighboring regions to accommodate large accelerators that cannot be fit into one region. A limitation of these architectures is that they rely on low-level manipulation of FPGA routing structures, which is only effective for one specific FPGA model.

A reconfigurable architecture for accelerating signal processing algorithms is proposed in [DKS+10]. Its FPGA-based configurable fabric is in the form of a systolic

array that is suitable for accelerating linear algebra operations. The processing elements in the systolic array reside in reconfigurable regions and can be dynamically reconfigured into two types of accelerators, one for the computation of Kalman filter and the other for discrete wavelet transform. The signal processing algorithm represented in two-dimensional dataflow is mapped to the systolic array where the input data (matrices or arrays of floating numbers) are consumed in parallel by the processing elements.

The RISPP architecture [BSH11] extends the instruction set of a general purpose processor by special instructions which indicate that they can be accelerated in hardware. When corresponding accelerators are available in the reconfigurable fabric, the special instructions will be executed using the accelerators otherwise they will be emulated in software on the processor. The FPGA-based reconfigurable fabric is partitioned into multiple regions and they are interconnected via a segmented bus [BSH08]. A runtime system predicts which accelerators will be needed soon and configures them into the regions before the actual execution of special instructions so that accelerators are available when required. Each special instruction has different implementation variants that differ in the used accelerators. The runtime system chooses the implementation variants that deliver most speedup per region to optimize the performance of the whole application.

Recently, authors of [BSB+14, CSZ+14] explore the possibility of providing FPGA-based reconfigurable fabric as a new type of computing resources in the cloud. The FPGA is partitioned into multiple partially reconfigurable regions which are managed by a hypervisor. Users can request to configure accelerators that are uploaded by themselves or prepared by the cloud vendor. Multiple users can share the usage of one FPGA by accessing separate regions.

Current reconfigurable architectures from major FPGA vendors are Zynq UltraScale+ MPSoC from Xilinx [BAG+15] and Xeon+FPGA platform from Intel [Gup15]. Zynq UltraScale+ MPSoC, targeting embedded systems, features two dual-core ARM processors and a Xilinx reconfigurable fabric. They are interconnected through ARM Advanced Microprocessor Bus Architecture (AMBA). Intel integrates a Xeon E5 server processor and an FPGA from Altera into a single package, and interconnects them using own proprietary QuickPath Interconnect (QPI) to maintain cache coherency.

2.6.2 Online Test and Diagnosis of Reconfigurable Systems

Online test and diagnosis methods are a prerequisite for handling faults in reconfigurable architectures. It can be distinguished between application-dependent and -independent test approaches for the reconfigurable fabric. Application-independent testing targets the whole fault universe of the fabric and is not limited to a specific use of the fabric. It typically employs multiple special test configurations and corresponding test stimuli [RPFZ97]. In contrast, application-dependent testing targets only a subset of the reconfigurable resources in the fabric relevant for a particular target application [Tah06]. These test approaches rely on built-in-self-test (BIST) principles where the test pattern generation and output response analysis

are implemented with the reconfigurable resources in the FPGA under test, i.e. no external test equipment or circuitry is required. In FPGAs with partial dynamic reconfiguration, the reconfigurations of the circuit under test using dedicated test configurations can be performed by an external or embedded processor at runtime [ASH+99, MDS06, ESA07]. The Roving STARs (Self Testing AReas) method [ESA07] for online test partitions the FPGA into rows and columns. While a region consisting of one row and one column (STARs) is tested by an online BIST approach, the rest resources on the FPGA is used for functional operation. The STARs sweep through the whole FPGA so that all reconfigurable resources are completely tested.

In addition to testing, the homogeneous structure of an FPGA allows the efficient diagnosis of faulty components. High resolution is achieved by failure data analysis and additional dedicated configurations to distinguish and localize faults [ASH+99, IMF98, ASE04]. In [AD16], multiple faults are diagnosed and can be tolerated using multiple diversified configurations with disjunct resource usage. The number of required configurations quickly rises with the number of faults to be detected and localized.

However, these approaches do not consider reconfigurable architectures that use runtime reconfiguration as part of their normal operation. Instead, they limit their use of runtime reconfiguration to generate test facilities on the FPGA. Targeting inherent runtime reconfigurable architectures (like the base system of this work) requires complex runtime decisions to minimize the interference between test applications and functional workloads.

2.6.3 Fault Tolerance in Reconfigurable Systems

Once a fault is detected and localized, different methods can be applied to ensure continued system operation despite of the fault. Tile-based fault tolerance techniques partition the reconfigurable fabric into a two-dimensional array of rectangular regions (tiles) [LMSP98, KKYY09]. In [LMSP98], a tile consists of multiple CLBs with one spare CLB. If a CLB in a tile is detected to be faulty, an alternative configuration for that tile is loaded to implement the same logic function but using the spare rather than the faulty CLB. In [KKYY09], the circuit in the faulty tile is entirely remapped to a spare tile. Column-based approaches apply similar concepts to CLB columns [HM01, MHS+04], where the fabric is partitioned into a one-dimensional array of CLB columns. Each column can implement an accelerator. In order to provide fault tolerance, intentionally unused columns are introduced as spares. In response to a fault, a precompiled configuration is loaded where the accelerator that resides in the faulty column has been remapped by shifting the accelerators starting from the faulty column towards the next unused spare. Both tile- and column-based approaches need complex customized routing techniques. Tile-based approaches require fixed interfaces between adjacent tiles so that each tile can be reconfigured independently of others. Column-based approaches require online routing after module remapping as the location of the accelerators change and the communication inbetween has to be re-established. They also do not maximize the inherent diversity in alternative configurations or exploit it to balance the stress in

the reconfigurable fabric.

The Roving STARs method [ESA07] combines distributed CLB spares and online compilation of configurations to replace faulty CLBs with spares. For complex designs, this online compilation or synthesis may cause unpredictable timing behavior. Instead, the fault tolerance methods proposed in this thesis work on CLB-granularity and does not need explicit tile/column-wise partitioning or online synthesis. The CLB placement and routing of the alternative configurations are prepared offline by vendor place-and-route tools.

Psarakis *et al.* [PA12] propose to use alternative configurations for accelerators, each of which uses different CLBs such that any single faulty CLB can be tolerated. However, they do not provide a method to automatically generate these configurations. They neither investigate the possibility of tolerating multiple CLB faults in general nor do they consider mitigation of aging effects within the reconfigurable fabric. The approach in [AD16] generates diversified configurations that are mainly used for diagnosing faulty CLBs. They also allow to tolerate all multi-CLB faults where up to k CLBs in a region can be faulty. For $k \geq 2$, the number of configurations quickly increases. None of these approaches use their diversified configurations to distribute the stress in the reconfigurable fabric, as proposed in this work. Pereira *et al.* [PBH$^+$11] employ online place-and-route to generate alternative configurations that map functional modules to non-faulty resources at runtime.

2.6.4 Aging Mitigation in Reconfigurable Systems

Aging mitigation by wear-leveling in runtime reconfigurable architectures can be achieved by using alternative logic mappings in CLBs, using spare resources in the fabric, or changing placements of accelerators. The coarse-grained approach in [SKM$^+$08] uses only two different configurations, which are swapped only once after a half-life period of the first failing component. A similar idea is used in [SC11], where three strategies are discussed for FPGA wear-leveling based on signal state inversion, use of spare resources for timing critical functions, and alternative placement. Since only two different configurations are used, the effectiveness is limited. Besides, high temperature accelerates the aging process and shall be avoided. Thermal aware placement algorithms such as [BB07, SS07] optimize the placement of CLBs at synthesis time to reduce the temperature difference between resources by estimating the thermal distribution among utilized CLBs based on usage and heat dissipation. They utilize simulated annealing as well as other optimization methods to determine an optimal placement. However, they do not consider runtime reconfiguration for their optimization, i.e. they assume that the placement never changes. [BMS10] considers the combination of process variation and NBTI aging and proposes a placement algorithm to reduce the delay degradation due to NBTI. While the authors suggest that the logic placement and configuration bitstream generation could be recomputed during runtime, for most embedded systems such a computation would cause too much overhead. In [RAKT13], aging in LUTs is mitigated by manipulating the configuration bits of LUTs. This method targets static systems in which the logic function of LUTs does not change during runtime.

Since typically not all CLBs in a region are actively used by an accelerator configuration [SKM+08], it is possible to prepare alternative placements and to reconfigure between them to distribute stress. This is achieved by periodically swapping of configurations that alter between used and unused resources, thus increasing the lifetime of the FPGA. The CLBs that are unused in a particular configuration can be utilized by another configuration to minimize stress [SC11]. This reduces the maximum stress in the resources and increases the system's Mean Time to Failure (MTTF), as demonstrated in [SKM+08, SWC10]. However, these techniques target the stress distribution of only *one* accelerator over the whole FPGA, whereas in runtime reconfigurable architectures the regions are typically used by different accelerators. They create alternative configurations for the entire FPGA, i.e. placing *one* complete design anywhere on the FPGA, targeting systems without runtime reconfiguration. In [ZBK+13, GB16], runtime reconfiguration with multiple regions and accelerators is considered. However, they only distribute the stress within one region, i.e. intra-region stress distribution. Ignoring the inter-region stress distribution may lead to the accumulation of high stress in individual regions, whereas this work proposes a stress distribution method that performs both intra- and inter-region stress distribution. The target system in [GB16] implements the entire application in one region and the region is exclusively used by a single application. In contrast, this thesis targets a more general architecture where any region can be used by any accelerator. The online placement of accelerators in [AZGT11] extends the KAMER placement algorithm [BKS00] by considering the maximum stress in the regions at runtime. The accumulated stress values of the resources in the candidate region are stored in a degradation table and their algorithm performs a local optimization that considers one accelerator after the other. With regard to aging mitigation, the method in [AZGT11] represents a current state-of-the-art approach and a comparison partner in the evaluation sections (Section 6.6 and 8.1).

2.6.5 Handling Soft-Errors in the Configuration Memory

Modular redundancy as a conventional approach for detecting and correcting transient errors incurs high area and power overhead. By exploiting the different vulnerability in different parts of a circuit, cost-effective protection mechanisms are selectively applied.

Prat *et al.* [PCG+06] identify the most vulnerable configuration bits by fault injection into the configuration memory. Some configuration bits may cause persistent errors in the circuit even if the corrupted bits are repaired by scrubbing. These bits are typically related to feedback structures in the control circuitry. Once corrupted, they remain malfunctioning until its state is reinitialized with a global reset. Given area constraints, these most vulnerable structures are protected by triplicating with high priority. A similar approach is proposed in [LBW12], targeting the protection of data paths. By fault injection, the vulnerability of each components on the data path is analyzed. The corrupted components that lead to larger deviation in the output numerical values are considered to be more vulnerable. Implementing these components in TMR is then given higher priority under the limited hardware budget. In [SRK04], the vulnerability of a gate to SEU is obtained from the probability

that a corrupted gate produces an erroneous result at the primary output of the circuit. It is analytically determined by the signal probabilities in the circuit. These selective modular redundancy techniques all target non-reconfigurable systems.

Selective scrubbing schemes adapt the location and period of scrubbing to the application needs. Nazar *et al.* [NSC13] propose to shift the start location of scrubbing for different designs implemented on an FPGA. The location of critical bits in the configuration memory is profiled to obtain the distribution of the number of critical bits in the linear address space of frames. As different frames carry different concentration of critical bits, it is possible to find the optimum starting frame for scrubbing that statistically minimizes the required time duration to locate and repair the corrupted frame. Santos *et al.* [SVDK14] propose a scheduling method for scrubbing depending on the criticality of hardware tasks. For a set of periodic tasks running on the FPGA, it schedules the scrubbing as close as possible to the start time of tasks to reduce the probability that the execution of a task is affected by SEUs. The scrubbing is also tasks-specific and it only checks the configuration bits used by the task following the scrubbing.

Runtime reconfiguration allows to change the redundancy mode at runtime to provide different reliability and performance levels under different environmental radiation levels and reliability requirements. The reconfigurable systems presented in [YJGR11, JCG+12] can adapt to a predefined set of modes (low, medium and high reliability, etc.) that trade off reliability and performance. Each mode corresponds to a certain degree of redundancy of partially reconfigured modules, i.e. no redundancy, DWC or TMR. For example, three identical functional modules can be instantiated in reconfigurable regions to process data in parallel for maximizing throughput or in TMR for highest reliability. An adaptive fault tolerant controller selects the appropriate mode based on the comparison between measured soft error rate and a set of predefined threshold values. The system in [AHOAD11] dynamically switch between redundancy modes to save power from replicated computation, where only one functional module is in operation when TMR is not required. Instead of making decisions directly based on the measured soft error rate, [GSR+14] first determines the error probability of a functional module due to soft error and then compares the error probability with a set of *Safety Integrity Levels (SILs)* [IEC10]. It chooses the lowest redundancy level that satisfies the reliability requirement.

These adaptive redundancy techniques target applications that only use a single hardware accelerator for which the reliability-performance trade-off has to be determined. It has not been shown that their approach could be extended to support complex applications consisting of multiple hardware accelerators.

3 System Overview and Cross-Layer Dependability

This chapter provides an overview of the proposed cross-layer dependability techniques for runtime reconfigurable architectures. Section 3.1 presents a generic model for applications running on reconfigurable architectures. The base architecture and its architectural extension to support dependable operation are given in Section 3.2. In Section 3.3, the fundamental assumptions for the target architecture are explained. Section 3.4 discusses the abstraction layers considered in this work and their relationship to the proposed dependability techniques. The basic platform and architectural simulator used for the experimental evaluation in this thesis are introduced in Section 3.5.

3.1 Application Model

In this thesis, a general application model that is not restricted to specific application domains is considered, as shown in Fig. 3.1. An application (Fig. 3.1(a)) consists of a mixture of normal operations, e.g. memory allocation and data preparation, and one or multiple computationally intensive parts, so-called *kernels*. Kernels are expected to be accelerated using dedicated hardware accelerators in the reconfigurable fabric, where runtime reconfiguration allows the optimal adaptation to different performance requirements of different applications. A kernel (Fig. 3.1(b)) corresponds to an outer loop that iterates through the whole dataset and that contains one or

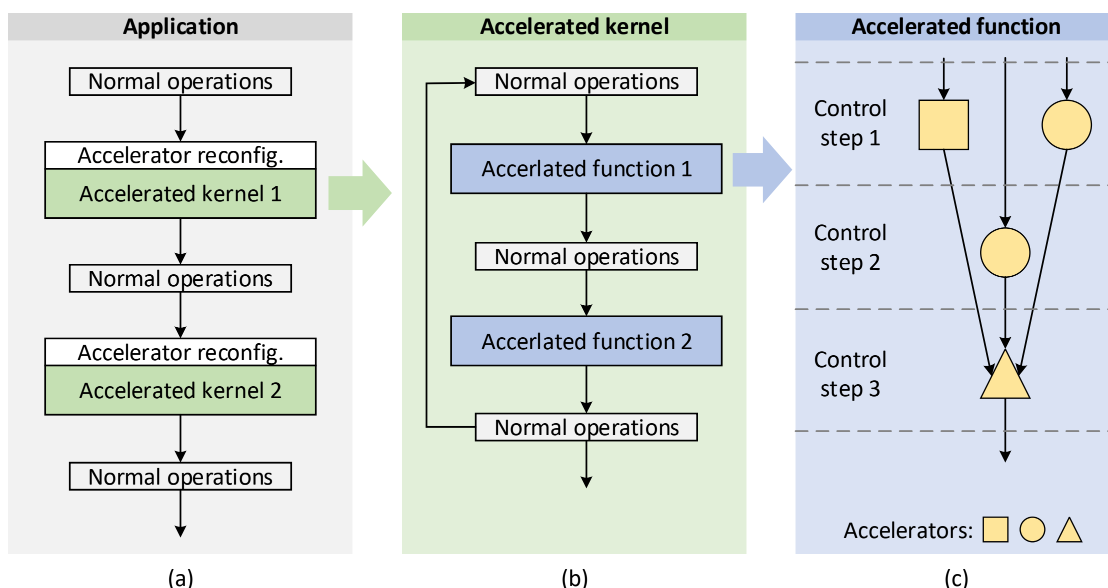

Figure 3.1: The application model used in this thesis

multiple inner loops that work on small data parts, specified by the current iteration of the outer loop. For example, in a stencil operation of an image, the outer loop iterates over each output pixel and the inner loop computes the output value based on multiple neighboring input pixel values. Such an inner loop is a good candidate to be implemented as an *accelerated function (AF)* that is composed of one or multiple accelerators of different types [CGG+14]. An AF is represented by a data-flow graph where each node corresponds to an accelerator and the edges correspond to data-flow between the accelerators. Figure 3.1(c) shows an example AF that consists of three accelerators of different types, represented by square, circle and triangle respectively. This DFG is scheduled in three control steps, where the "circle" accelerator is reused in the 1st and 2nd control steps. Before the execution of a kernel, all required accelerators need to be configured into the reconfigurable fabric, or otherwise the accelerated functions have to be emulated in software on the GPP.

3.2 Target Architecture

3.2.1 Base Architecture

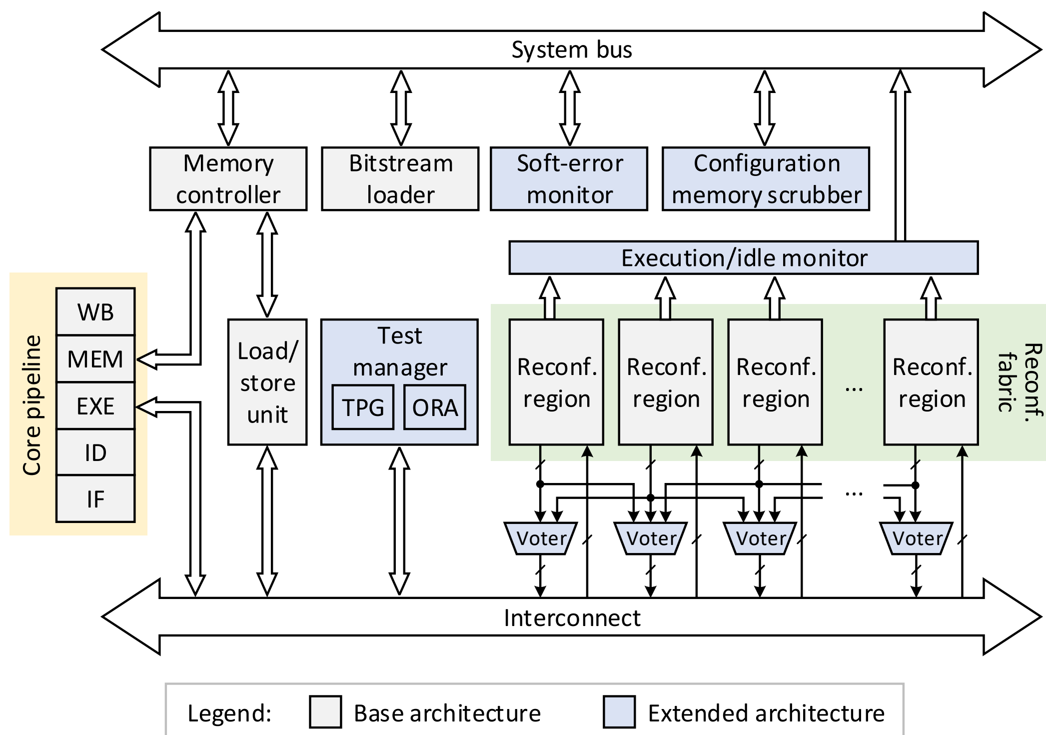

Figure 3.2: Target reconfigurable architecture

The target reconfigurable architecture is shown in Fig. 3.2, which is extended based on [BSH11]. The base architecture is composed of

a general purpose core pipeline that executes the instructions that are not accelerated in the reconfigurable fabric and a runtime system that performs acceleration and dependability management,

a reconfigurable fabric consisting of multiple *reconfigurable regions* which could be different FPGAs or different partitions within one FPGA, into which different accelerators can be reconfigured,

a memory controller handling memory accesses from the core pipeline and the reconfigurable fabric, and

a bitstream loader that processes the reconfiguration request from the runtime system and transfers the bitstreams of accelerators from the system memory to the FPGA configuration memory to reconfigure the fabric.

The core pipeline is extended to support AFs which are identified as special assembler instructions that perform application-specific computations such as transformations, filters, and encryptions. If accelerators required by an AF are available in the reconfigurable fabric, the AF is executed using the hardware accelerators. If required accelerators are not available in the reconfigurable fabric, e.g. because its reconfiguration did not finish yet or available regions are insufficient, the AF can alternatively be emulated in software on the core pipeline by issuing an “unimplemented instruction” trap [BSH08]. This ensures that the application can be executed as long as the core pipeline is functional.

One or multiple accelerators are required to be reconfigured into regions to implement an AF in hardware. A runtime system decides, into which regions accelerators shall be reconfigured as well as the reconfiguration sequence.

A state-of-the-art FPGA architecture is employed as reconfigurable fabric to provide high performance for the compute-intensive parts of the application that are offloaded to it. SRAM-based FPGAs are compatible with the standard CMOS manufacturing technology [KTR08] and therefore are able to share the benefits of performance, power and area improvement brought by aggressive technology scaling. Furthermore, SRAM-based FPGAs provide faster runtime reconfiguration than flash-based FPGAs, which is essential to reconfigurable architectures. Hence, SRAM-based FPGAs are used in the base architecture for the reconfigurable fabric.

3.2.2 Architectural Extension

In this thesis, this base architecture is extended to support dependable operation of the reconfigurable architecture. The additional components are

a test manager equipped with test pattern generator (TPG) and output response analyzer (ORA) that performs structural tests on the reconfigurable fabric and functional tests on the reconfigured accelerators,

an execution/idle monitor tracking when a region is recently reconfigured and how often the currently-configured accelerator is executed,

a soft-error monitor that estimates the soft error rate currently experienced by the reconfigurable fabric which changes with the environment and the system operation conditions,

a configuration memory scrubber periodically reading back the configuration data of currently configured accelerators to detect and correct errors in the configuration memory, and

voters supporting disparity checking or majority voting of the accelerator outputs when neighbouring accelerators are paired to compose DWC or TMR, respectively.

The extended architecture enables online testing, resource usage monitor, accelerator redundancy and configuration memory scrubbing, which are indispensable for the dependable reconfigurable architecture threatened by aging and SEU.

Online testing is triggered by special AFs from the core pipeline. The AF execution infrastructure in the base architecture is reused to transfer the test request and parameters to the test manager. Deterministic test stimuli generated by the TPG are fed to the inputs of the reconfigurable regions to exercise the configured accelerator or test configurations. The responses from the outputs of the regions are collected and evaluated by the ORA. Test results are transfered back to processor core as the outcome of the AF execution.

The execution/idle monitor keeps track of the accelerator usage profile in each region. It stores the timestamp (in cycles) when an accelerator is reconfigured into a region and counts the number of cycles while the accelerator is being executed. Whenever the region is reconfigured again, the execution counter and reconfiguration timestamp are read and reset. The idle cycles of an accelerator are calculated by subtracting the execution cycles from the total cycles during which the accelerator is configured in the region.

Since SER experienced by the reconfigurable fabric changes with its environment and depends for instance on the radiation level, temperature or voltage [FCMG13], the soft-error monitor estimates the current SER per bit λ by computing the maximum of two indicators available in the system:

1. The SER per bit λ_{scrub} in the configuration bits obtained from periodic scrubbing,
2. The SER per bit λ_{bram} in the block RAM arrays in the FPGA. This error rate can be obtained since block RAMs are protected by ECC (see Section 2.4.4).

The current SER per bit in the reconfigurable fabric is then computed conservatively and concurrently to the system operation as their maximum: $\lambda = \max(\lambda_{scrub}, \lambda_{bram})$. In the terrestrial environment, if the neutron flux per second at the operation location of the device is known [JED06], the SER per bit can be alternatively estimated by using the neutron cross-section per bit [Xil16b]:

$$\lambda = \text{neutron flux per second} \times \text{bit area} \times \text{neutron cross-section per bit} \tag{3.1}$$

Soft-errors in the configuration memory are defended by the combination of modular redundancy and periodic scrubbing. Modular redundancy, i.e. DWC or TMR, is achieved by configuring neighboring regions with identical accelerators and using voters to detect (DWC) or correct (TMR) errors in the accelerator outputs. When no redundancy is required, the voter simply passes the input coming from its belonging region to the output. The configuration memory scrubber periodically readbacks the configuration frames of each configured accelerator and calculates the syndrome bits for every frame [Xil12c]. All-zero syndrome bits indicate an error-free frame. Non-zero syndrome bits indicate the error type (SBE or MBE) and the location of the erroneous bit in the case of SBE. An SBE will trigger the correction of the bit in error using the syndrome information and the corrected frame will be written back to the configuration memory. An MBE cannot be corrected using the syndrome information and will trigger the reconfiguration of the affected accelerator using the bitstream stored in the system memory.

3.3 Architectural Assumptions

In this thesis, the following basic assumptions about the hardening of the non-reconfigurable components and the partitioning of the reconfigurable fabric are applied.

Statically hardening the reconfigurable fabric (i.e. FPGA) would further worsen its silicon area efficiency, which is known to be significantly lower than an ASIC implementation for a given circuit [KR07, WBR11], due to the provided capability of customization of the hardware organization. The core pipeline and voters are instead assumed to be hardened by manufacturing processes [HB03, DSSF10]. And the communication infrastructure is assumed to be protected by ECC. Therefore, they are much less susceptible to soft-errors than the reconfigurable fabric and thus this thesis focuses on the dependability of the reconfigurable fabric. The runtime system is assumed to be running on the hardened processor.

All reconfigurable regions are assumed to be of identical size, shape and composition of reconfigurable resources, which allows any accelerator to be configured into any region. Accelerator relocation techniques allow to use only one implementation (i.e. partial bitstream) per accelerator, regardless of the region into which the accelerator shall be reconfigured at runtime [BKT14, BHW+14]. While it would be possible to use the reconfigurable fabric in a more flexible manner (e.g. variable-sized regions), it would come with significant drawbacks such as the demand to create different implementations per accelerator (optimized for different region types), complex management of available resources [Ahm07] and the difficult aspect of establishing communication between variable-sized regions [PAS+09].

3.4 Cross-Layer Dependability

In reconfigurable architectures, most of the computations are offloaded to the reconfigurable fabric for hardware acceleration. The importance of the core pipeline

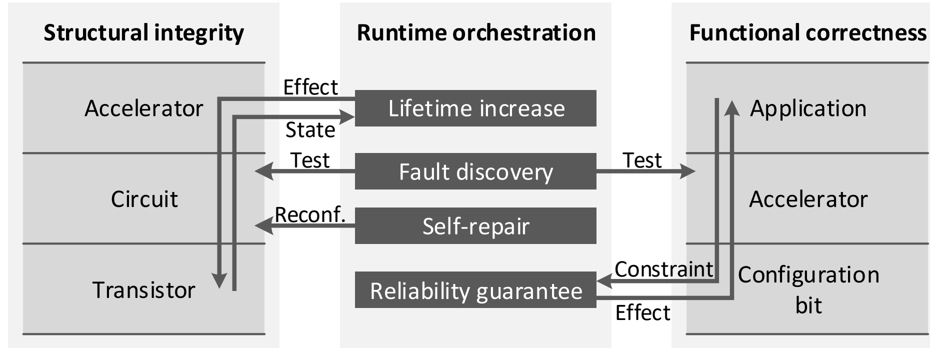

Figure 3.3: System layers and their interaction with the dependability approaches proposed in this thesis

is diminished as it does not perform the actual computation but plays the role as a logistics manager that take care of the tasks around the computation, e.g. configuration of accelerators and initiating acceleration. The “management" core merely need to execute light-weight tasks and thus do not require the performance benefit of cutting-edge process technologies. Therefore, the core pipeline can be built with less complexity using dependable technologies from previous generations while this thesis focuses on the dependability of the reconfigurable fabric on FPGAs that are manufactured in newest technology nodes. The dependable operation of a hardware accelerator implemented on FPGAs relies on the following two aspects.

- **Structural integrity**: the underlying reconfigurable hardware which is employed to implement the accelerator shall be fault-free.
- **Functional correctness**: the configuration data for the accelerator, which is stored in the configuration memory of the FPGA and tailor the underlying hardware to perform the desired functionality, shall be error-free. Functional memory elements such as block RAMs and flip-flops are dominated by configuration bits in terms of error susceptibility and thus are not considered in this work (see Section 2.4.4).

Latent defects not detected during manufacturing and electrical and physical degradation induced by aging processes threaten the structural integrity of the reconfigurable fabric. Structural integrity is the prerequisite of functional correctness, while structural integrity alone is not sufficient for functional correctness. The reason behind it is twofold. Firstly, the delay on the critical path of an accelerator is determined by the delay of all logic and routing components on the path, which experiences the manufacture variation and aging process. A structural fault-free fabric may still cause delay faults of configured accelerators. Secondly, the configuration data for an accelerator define the logic function of the configurable logic blocks and signal routings among logic blocks through the configurable switching matrices. The configuration data precisely define the functionality of an accelerator in terms of how reconfigurable resources shall be customized and how they shall be interconnected. Corrupted configuration data, e.g. due to SEU, lead to incorrect functionality definition and operation of the accelerator.

In this thesis, the dependability of the reconfigurable fabric is viewed from two

perspectives (structural integrity and functional correctness) and each of which comprises three abstraction layers, as shown in Fig. 3.3. The layers from the perspective of structural integrity are as follows.

- *Transistor layer*: The basic operation unit at this layer is a transistor, i.e. NMOS or PMOS. The computation is carried out by the transport of charges among transistors.
- *Circuit layer*: The basic operation unit at this layer is a reconfigurable primitive in an FPGA, e.g. LUTs, flipflops and multiplexers, which are composed of transistors. The computation is performed in the form of basic logic operations.
- *Accelerator layer*: An accelerator is the basic operation unit at this layer. It is a composite of multiple reconfigurable primitives. It performs complex functions such as mathematical transformations.

From the perspective of functional correctness, the layers are:

- *Application layer*: At this layer, an application is viewed as the sole operation unit. An application operates correctly when it produces error-free results which rely on the correct outputs from used accelerators.
- *Accelerator layer*: An accelerator is the basic operation unit at this layer. It delivers correct outputs when its configuration bits are not corrupted. It can be replicated to implement modular redundancy.
- *Configuration bit layer*: At this layer, individual configuration bits are the basic information fragment that defines the functionality of reconfigurable resources. Scrubbing detects and repairs individual erroneous configuration bits.

The propagation and transformation of parameters across multiple layers allow a precise cause-effect analysis, e.g. how the usage of accelerators affects the lifetime of transistors, which leads to the delivery of runtime decisions directly on the concerned dependability objectives. The dependability techniques proposed in this thesis and their relationship to the abstraction layers are introduced in the following sections.

3.4.1 Lifetime Increase

Electrical and physical stress induced in transistors by acceleration workload is balanced among all available transistors in the reconfigurable fabric such that the stress is not accumulated in individual transistors and causes them to fail much earlier than others. In this way, the lifetime of the reconfigurable fabric can be prolonged as the failure time of the transistors is delayed. The stress balance is achieved by smartly arranging the placement of accelerators, i.e. into which region an accelerator shall be placed.

The stress states at the transistor layer is propagated through circuit layer to the decision making at the accelerator layer. The decision making at accelerator layer aims at a uniform distribution of stress over all configurable logic blocks at circuit

layer, which is automatically translated to the stress balancing at transistor layer. The stress states of individual transistors (transistor layer) is aggregated and represented as the stress states of individual configurable logic blocks (circuit layer), which form the basis of the decision making of accelerator placement (accelerator layer). The goal of the stress-aware accelerator placement is to distribute the stress uniformly among all available reconfigurable resources depending on the their current stress states and the stress patterns of the accelerators to be placed. Thorough discussions of the stress balancing technique are presented in Chapter 6.

3.4.2 Fault Discovery

To detect structural faults and configuration errors in the reconfigurable fabric, online test must be an integral part of the system operation. Two different types of test procedures are integrated: pre-configuration test (PRET) and post-configuration test (PORT). PRET exercises the underlying reconfigurable hardware structures in a region, e.g. LUTs, flipflops, multiplexers, etc. before the actual configuration of an accelerator. Permanent or intermittent structural faults that may impair the operation of configured accelerators can be detected at this phase. After the accelerator configuration, PORT is performed to examine whether the configured accelerator delivers the desired functionality. During PORT, accelerator-dependent test stimuli are fed to the inputs of the accelerator at the operation frequency of the accelerator. The outputs of the accelerator are aggregated and verified against a pre-computed signature. A mismatch signalizes that an erroneous operation of the accelerator is detected. It may originate from incorrect configuration data or delay faults. The integration of PRET and PORT into the target reconfigurable architecture is discussed in Chapter 4.

3.4.3 Self-Repair

If a certain resource in a region is detected to contain structural faults, any accelerator that requires the resource cannot be placed into that region, as otherwise the accelerator may produce incorrect outputs. This limits the placement freedom of accelerators and may eventually lead to unplaceable accelerators and thus performance degradation.

Due to the regular and homogeneous resource structure in FPGAs, an accelerator can have multiple physical implementations in a region that are diversified in terms of resource usage. An accelerator diversification design method is developed at the circuit layer. A minimal set of accelerator configurations is generated for tolerating any single-CLB fault in one region. Additional configurations enable tolerance of multi-CLB faults (i.e. multiple faulty CLBs). After the faults in a region are detected and localized, any accelerator can be configured into that region, as long as one of its diversified configurations does not require the faulty resources. And therefore the placement freedom and performance can be maintained at the level of a faulty-free system. The accelerator diversification design method is presented in Chapter 5.

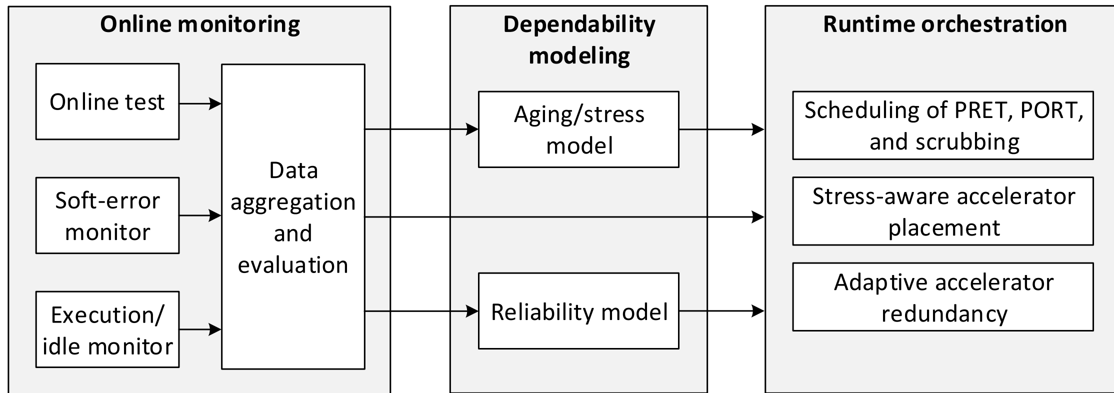

Figure 3.4: Runtime orchestration of dependability techniques

3.4.4 Reliability Guarantee

In mission-critical applications, demanding reliability requirements such as IEC61508 [IEC10] and ISO 26262 [ISO11] and need to be satisfied under changing operation and environmental conditions. Such reliability requirements are typically given at the application layer since it is the closest layer to the products or end-user experience. As most of the computations in reconfigurable architectures are offloaded to the reconfigurable fabric, the reliable computation of accelerators is essential to the reliable operation of the whole system. Since the functionality of accelerators is defined by their configuration bits, the reliable operation of accelerators is determined by whether their configuration bits are error-free.

Outputs of accelerators can be protected by modular redundancy and the configuration bits of accelerators can be protected by periodic scrubbing. Blindly applying modular redundancy to all accelerators regardless of the application requirements and environmental conditions (e.g. radiation level) incurs high cost in error detection and correction. The system may be overprotected under the excessive usage of reconfigurable resources for redundancy, which could have been used for acceleration. Optimized adaptation of redundancy to different application reliability constraints and environmental conditions requires the propagation of reliability constraints from application layer through accelerator layer down to configuration bit layer. The reliability constraint of the application determines which accelerators shall be implemented with redundancy for a given soft error rate and scrubbing frequency of the configuration bits. In the other direction, the scrubbing of the configuration bits and redundant accelerators determine the perceived reliability of the application by the user. Chapter 7 proposes a runtime adaptation method for optimizing the system performance under a given reliability constraints of the application.

3.4.5 Runtime Orchestration

All the above dependability techniques are managed by the runtime system during system operation, as shown in Fig. 3.4.

Online monitoring uses different techniques to obtain important physical and log-

ical information that characterizes the current system states. They range from monitors of the environment (e.g. soft error rate) to performance monitors (e.g. resource utilization). The monitored data are collected and aggregated to provide the input metrics for the dependability models or for the decision making during the application of dependability techniques. The aging and stress models derive the stress states of the reconfigurable resources based on the monitored resource usage information. Based on the stress states, the runtime system optimizes the stress distribution to increase the system lifetime. The reliability model evaluates the error probability of the execution of accelerators based on the monitored soft error rate and resident time of accelerators. Appropriate redundancy modes for different accelerators are then selected by the runtime system to satisfy the application reliability requirements. These models are evaluated at runtime on the target architecture where the available computational and memory resources for model evaluation are limited. Thus, model abstraction and simplification is exploited such that evaluation at runtime imposes only negligible performance overhead while still offering an effective assessment of the dependability states.

The dependability techniques (non-functional workload), particularly PRET, PORT and scrubbing, have to operate concurrently together with the application execution (functional workload) in the reconfigurable fabric. These two functionalities compete with each other for shared resources, e.g. the reconfiguration port of the FPGA for PRET and scrubbing, and thus the application execution may be affected by performing dependability techniques. At runtime, the functional and non-functional workload are orchestrated in such a way that the dependability techniques introduce minimum disturbance to the application execution and are fully transparent to the user.

3.5 Evaluation Platform

The target reconfigurable architecture forms the platform for the experimental evaluation in this thesis. A LEON processor [Aer] is used as the core pipeline with a reconfigurable fabric with configurable number of regions (see Fig. 3.2). The actual hardware prototyping is performed on an XUPV5 FPGA board with a Xilinx Virtex-5 LX110T. A SystemC-based cycle-accurate architectural simulator (parameterized by the hardware prototype) [BSH09] is used for evaluating different system parameters like the number of regions and different runtime strategies. It accurately models the hardware implementation of the reconfigurable architecture including the bus arbitration in the reconfigurable fabric, the duration of reconfiguration, and the application behavior including request arbitration for accelerator configuration and software-emulation of unavailable accelerators. The system operates at a clock frequency of 100 MHz and a reconfiguration bandwidth of 50 MB/s (limited by off-chip system memory that is also used to store partial bitstreams).

A sophisticated H.264 video encoder is the main application used in evaluation since video and image processing are typical applications for reconfigurable architectures [GCS+06]. The encoder is also a challenging application since it frequently reconfigures accelerators in the regions. The H.264 encoder consists of three differ-

Table 3.1: Short description of accelerators implemented for H.264

Accelerator	Description
Clip3	clipping to a configurable min/max interval
CollapseAdd	summing up the 4 bytes inside a 32-bit word
LF_BS4	4-pixel edge filter for in-loop deblocking
LF_Cond	filtering condition for in-loop deblocking
PointFilter	six-tap filter for sub-pixel motion estimation and compensation
QuadSub	4 parallel byte subtractions
SADrow_4	sum of absolute differences of two 4-pixel rows
SAV	sum of absolutes of four 16-bit values
Transform	(inverse) discrete cosine transform or (inverse) Hadamard transform

ent kernels that are executed in sequence for each video frame: Motion Estimation (ME), Encoding Engine (EE), and in-loop deblocking filter. Each kernel requires different accelerated function as well as different accelerators that are reconfigured before the execution of the kernel. For instance, when EE processes a frame then the accelerators for EE are reconfigured which replaces the accelerators for ME that finished processing this frame before EE starts. The accelerator requirements for a particular kernel may vary over time. For instance, EE uses different encoding techniques (accelerated by different accelerated functions) depending on the input data, e.g. slow moving objects vs. hectically changing structures. In total, nine accelerated functions are implemented for the H.264 encoder by using combinations of 9 different types of accelerators, as listed in Table 3.1).

4 Fault Discovery through Strategic Online Testing

As latent faults and aging effects threaten the structural integrity of nano-CMOS devices, conventional manufacturing and burn-in tests are no longer sufficient to guarantee dependable reconfigurable architectures throughout the whole product lifecycle. The reconfigurable fabric on the FPGA needs to be constantly monitored by thorough *online testing* to check its correct operation over time. In contrast to offline testing, online test can be performed concurrently to the system operation and has the potential to minimize the interference to the functional workload and thus the performance overhead. This task is particularly challenging for runtime reconfigurable architectures where the hardware organization changes during runtime as an integral part of the normal operation.

This work proposes the integration of two types of online test that complement with each other: *pre-configuration online Tests (PRET)* and *post-configuration online tests (PORT)*. PRET is an application-independent *structural test* while PORT performs application-dependent *functional test*. PRET and PORT differ in their target fault models, test intervals, and test application time. PRET is designed to exhaustively test the underlying hardware structure, e.g. logic resources in CLBs, in the reconfigurable fabric periodically or on-demand. However, PRET alone does not guarantee the correct functionality of a configured accelerator. Errors may occur during the loading of bitstreams, e.g. due to faults in the configuration logic or transient events like SEU or crosstalk. As a consequence, the configured function of the targeted region may be wrong or the configuration in other parts of the reconfigurable fabric may be adversely altered. PORT aims to perform at-speed functional tests on accelerators after they are instantiated in the reconfigurable regions. PORT checks whether they have been configured correctly and deliver the desired functionality. At mission time, PORT also periodically checks the accelerators for malfunction due to emergent permanent faults or soft errors in the configuration memory. The test application interrupts the system operation only for a minimal amount of time in the order of a few microseconds.

The development of the test infrastructures for PRET and PORT, including test configurations, test pattern generators and output response analyzers, is accomplished by the project partners at the University of Stuttgart and is not part of the contributions in this thesis. This work seamlessly integrates the infrastructures for PRET and PORT into the target reconfigurable architecture such that they are transparent to applications and users, i.e. they don't have to be modified. Both test scheme are scheduled concurrently to functional workload by the runtime system with minimum impact on application and system performance.

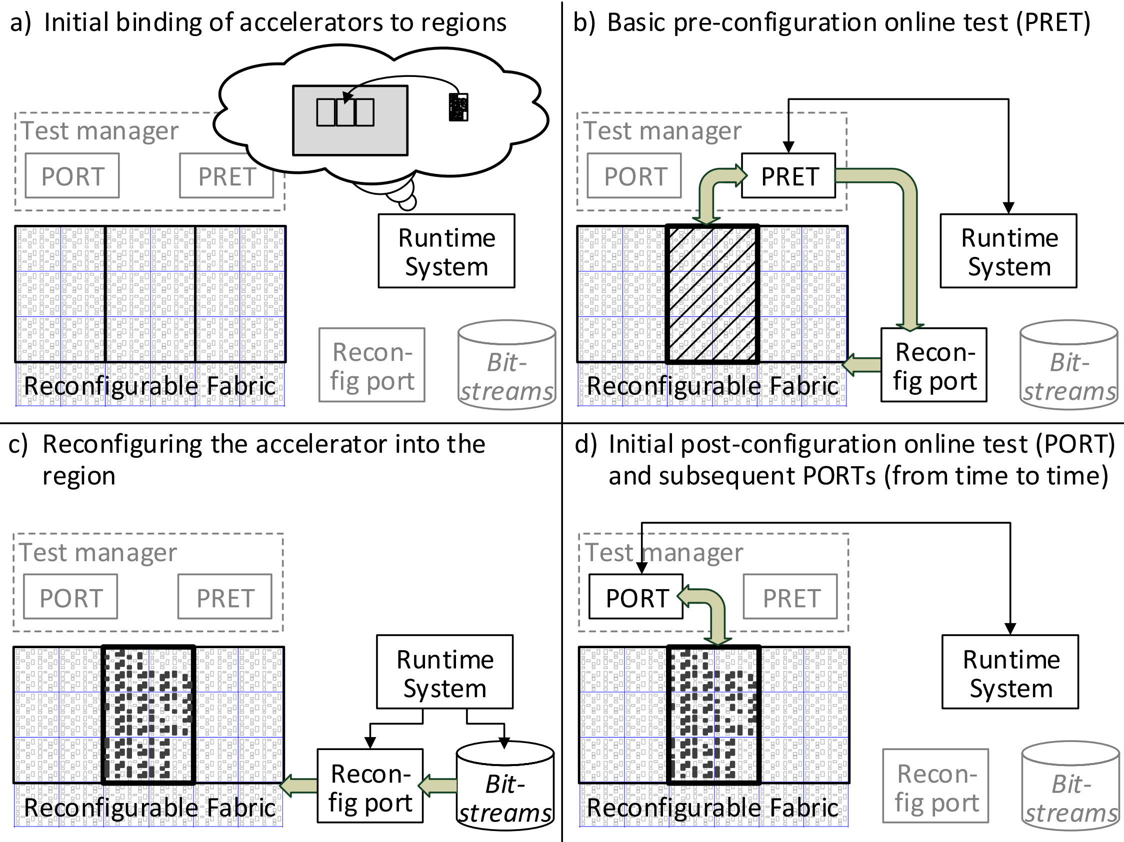

Figure 4.1: Test flow with PRET and PORT

4.1 Overview of Online Test Strategies

For PRET, the array-based structural test approach as proposed in [ABB+12, BBI+12] is used to generate the test configurations for the exhaustive test of all logic resources in a reconfigurable region. Additional PRET test configurations for the application-dependent interconnect test are generated as proposed in [BBI+13], where all logic functions in the CLBs used by an accelerator are replaced by the XOR function. This allows a high controllability and observability of the used interconnects. For PORT, commercial Automatic Test Pattern Generation (ATPG) tools such as Synopsys TetraMAX are used to generate the test patterns and the corresponding response signatures for different accelerators.

Figure 4.1 shows the proposed online test flow for a reconfigurable fabric with three regions. In the first step (Fig. 4.1a), the runtime system decides that an accelerator shall be reconfigured into a particular region, which triggers the demand to test the hardware structures in that region first before the actual configuration of accelerators (so-called *on-demand PRET*). To exhaustively test all reconfigurable resources in the region, multiple test configurations (TCs) are required (see Section 2.5.1) and they would delay the accelerator reconfiguration significantly. The runtime system can choose to execute PRET incrementally to reduce the delay. This means that not all TCs are applied to the region at once, but only some of them. The runtime system decides how many and which TCs should be applied (potentially none) before reconfiguring the accelerator, based on the test history of the region. The runtime

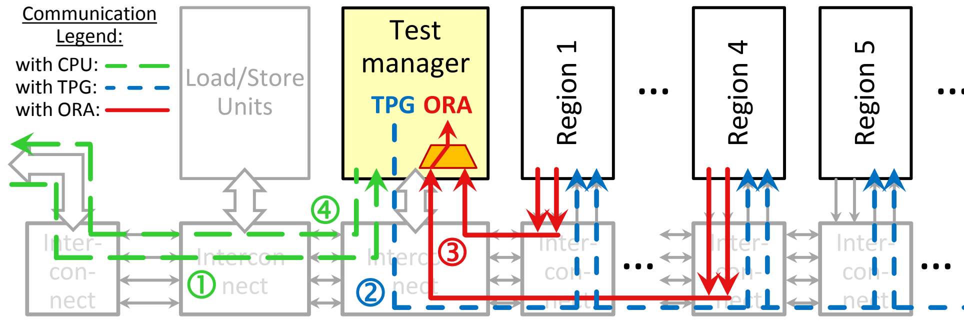

Figure 4.2: Test manager integration with TPG and ORA

system keeps track of which TCs were applied to a region in the past and how much time passed since the last exhaustive PRET. Depending on this test history, it activates PRET, reconfigures the selected TCs into the region, and uses TPG and ORA of the Test Manager to exercise the reconfigurable fabric (Fig. 4.1b). In addition to the on-demand PRETs, the runtime system also schedules *periodic PRETs* to ensure that also seldomly reconfigured regions are tested.

If no structural fault is found by PRET, the runtime system reconfigures the desired accelerator into the region (Fig. 4.1c). Before the accelerator is used by the application, the runtime system triggers an *on-demand PORT* (Fig. 4.1d) to test whether the reconfiguration process has completed without error, i.e. whether the accelerator delivers the desired functionality at the specified clock frequency. Additionally, accelerators instantiated in other regions are tested as well to check that they were not affected by the reconfiguration. As PORT does not require any reconfiguration of TCs, it operates significantly faster than PRET and can also be applied during normal operation, which is scheduled by the runtime system as subsequent *periodic PORTs*.

4.2 Integration of Online Tests

The test manager including TPG and ORA for the online testing is integrated into the base reconfigurable architecture (see Section 3.2.1) and coupled to the interconnect for the reconfigurable fabric such that communication channels between reconfigurable regions and the test manager can be established. The test manager reuses the same interconnect infrastructure that is already available for the inter-region communication of accelerators. Figure 4.2 shows a detailed view of the interconnection between the test manger and the regions.

In order to utilize this infrastructure, both PRET and PORT are implemented as special accelerated functions (test-AFs). In the base architecture, all AFs contain an implicit control word which is stored in an on-chip memory for each executable AF. This control word configures the interconnect infrastructure according to the dataflow graph of the AF to control the data flow among accelerators and between the reconfigurable fabric and the CPU. For the special test-AFs, the same mechanism is used to establish the connection between the test manager and the region under test.

For PRET, the test configuration needs to be reconfigured into the region under test before the test-AF can be executed. For PRET and PORT, the procedures of sending the test patterns to the region under test and analyzing the responses are similar and are both performed by test-AFs. When the processor executes a test-AF, the test parameters for the AF are sent as the AF input data from the register file of the processor to the test manager, as shown in Fig. 4.2 step ①. These parameters determine which region shall be tested and which test patterns are to be applied. The test manager then sends the test patterns to all regions (Fig. 4.2 step ②). The test patterns can be directly generated online by the TPG in the test manager, or can be prepared offline and stored along with the expected output signatures in an on-chip memory. The memory is attached as a slave to the system bus and initialized when the system starts.

For PRET targeting the logic primitives within CLBs, a test pattern and its corresponding response can be packed into one 32-bit word each. For each test pattern, the responses from 4 regions are sent back to the test manager (Fig. 4.2 step ③). The limitation to 4 responses per cycle is due to the availability of 4 buses for the interconnect infrastructure. The example in Fig. 4.2 step ③ shows how the responses of regions 1–4 are sent back via the buses. The test manager then selects the response of the region under test with an internal multiplexer. To perform PRET on regions 5–8, another test-AF needs to be triggered such that the interconnect infrastructure is reconfigured for sending the responses of these regions back.

During PORT, the responses of the accelerators are compacted in space and time using a 32-bit multiple input signature register [WWW06]. After the application of the test patterns, a single 32-bit signature per region has been computed and is stored locally. The hardware that computes the signature is integrated into the interconnect infrastructure such that the outputs and the bus interface of a region are tested simultaneously as well. PORT only needs one test-AF that tests all configured accelerators consecutively. After applying all test patterns, the locally stored signatures are transferred to the test manager in multiple cycles (four signatures per cycle). The ORA of the test manager then compares the response signatures with the expected signatures that are specific for each accelerator. The information which accelerator is reconfigured to which region is available in the hardware architecture and is updated before and after each reconfiguration.

For PRET targeting interconnects, an interconnect test configuration is reconfigured into the region under test similar to PRET for the logic primitives. Then, stored test patterns are applied similar to PORT, and a test signature is computed and transferred back to the test manager.

At the end of the tests (PRET or PORT), the final test result (passed or failed) is written back to the register file of the processor (Fig. 4.2 step ④). As the PORT test-AF tests all configured accelerators consecutively, the result contains a 1-bit information (passed or failed) for each region.

4.3 Scheduling of Online Tests

Both PRET and PORT are scheduled by the runtime system periodically and on-demand to detect faults as early as possible.

4.3.1 PRET Scheduling

During PRET, multiple test configurations need to be configured into the region under test, which blocks the configuration port of FPGA and delays the configuration of accelerators. Thus, in order to reduce the impact on the application performance due to the unavailability of accelerators, PRET is only executed at times when the system needs to be reconfigured anyway. An on-demand PRET *Test Configuration* (TC) is scheduled after a certain number of *Accelerator Configurations* (ACs). For instance, one TC is scheduled before every AC or before every 2nd AC, 3rd AC, etc. This allows to distribute the tests over space (in different regions) and time (in one region). The tests are initiated by the runtime system that is responsible for scheduling the accelerator configurations. After a reconfiguration completes, the runtime system is informed by an interrupt. If that reconfiguration was a test configuration, the runtime system executes a PRET test-AF for the corresponding region and evaluates the test result before triggering the next reconfiguration.

In addition to on-demand PRETs before accelerator configurations, a timer-based periodic PRET is integrated as well. This limits the time period during which a region remain untested, i.e. test latency, when that region is only rarely reconfigured by the application. In extreme cases, an application could reconfigure a particular region a single time at start and then use the accelerator in that region without ever reconfiguring the region again. In such a case, no PRET would be performed on this region except at the start, which can be prevented by the periodic PRET.

The periodic PRET is implemented by a handler (see Alg. 1) consisting of two phases: i) triggering the reconfiguration of a TC for a particular region (lines 3–14) and ii) executing the PRET test-AF after the TC is reconfigured (lines 15–29).

The first phase of the handler scans all regions for their last test time (maintained by a data structure of the runtime system) and identifies the least recently tested region (lines 4–6 in 1). If the time since the last test is larger than a configurable threshold (500 ms is used for the evaluation in Section 4.4.3), a periodic PRET is triggered for this region. The second phase of the handler is activated when the reconfiguration of the TC triggered by the first phase is completed. It then executes the PRET test-AF, informs the runtime system about the health state of the region under test and updates the data structures for the next PRET.

4.3.2 PORT Scheduling

The on-demand PORT is conducted directly after accelerator configuration to assure that the reconfiguration process has correctly completed without error and that

Algorithm 1 Interrupt Handler for periodic PRET using the 'Least Recently Tested Strategy'.

```
Input: Trigger by timer_event and reconf_complete_event
Input: reg[i]: runtime system information about regions
 1: static pret_reg := NULL // which region
 2: static pret_tc := NULL // which test configuration
 3: if (triggered by timer_event) then
 4:    // determine least recently tested region
 5:    lrt_reg := min∀Region i {reg[i].last_test_time}
 6:    lrt_time := cont[lrt_cont].last_test_time
 7:    if (current_time - lrt_time > Threshold) then
 8:       pret_reg = lrt_reg
 9:       pret_tc = reg[lrt_reg].next_tc
10:       // Trigger the reconfiguration of the test config
11:       reconfiguration_queue.push(pret_reg, pret_tc)
12:       return
13:    end if
14: end if
15: if (triggered by reconf_complete_event and pret_reg ≠ NULL
        and reg[pret_reg].accelerator = pret_tc) then
16:    switch (pret_reg)
17:    case 0–3:
18:       result = pret_si_reg0_3(pret_reg)
          // this calls the test-FA for regions 0–3
          // parameter: which of these 4 regions to test
19:       break
20:    case 4–7:
21:       result = pret_si_reg4_7(pret_reg-4);
22:       break
23:    end switch
24:    reg[pret_reg].health_state := result
25:    reg[pret_reg].last_test_time := current_time
26:    reg[pret_reg].next_tc := (reg[pret_reg].next_tc+1) mod number_of_tcs
27:    pret_reg := NULL
28:    pret_tc := NULL
29: end if
```

the configured accelerator delivers the expected functionality. As PORT tests all configured accelerators consecutively in one test session (see Section 4.2), errors in the other accelerators, e.g. due to address decoder faults in the configuration logic or errors in the configuration address, are detected as well.

In addition, periodic PORTs are also scheduled at runtime to check the accelerators for malfunctions, caused e.g. by emergent faults in CLBs due to aging effects or soft errors in the configuration memory. Periodic PORT is realized by an interrupt handler similar to the periodic PRET handler (Alg. 1), but without the need to trigger reconfigurations.

4.4 Experimental Evaluation

The evaluation platform introduced in Section 3.5 is used to investigate the overhead and test effectiveness of PRET and PORT, integrated as described in Section 4.2 into the target architecture.

4.4.1 Fault Models of Tests

The *stuck-at fault model* is widely adopted in the literature for FPGA testing [Ren98]. For complex FPGA components such as lookup tables (LUTs) and flip-flops, typically only a functional description is available from the vendor and structural implementation details are missing. This results in a weak modeling of defects. Here, the stuck-at fault model is used for components in which the structural information is sufficient for fault derivation and for the interconnects. For the remaining components, structural and cell faults are targeted during test generation resulting in a hybrid fault model. The tests are generated under the single fault assumption.

- *LUT as truth table:* The LUT is treated as a combinational function of n inputs and m outputs, and the cell fault model [Kau67, PGP98] is applied. Cell faults describe any mismatch at the outputs of a unit under test for the possible inputs. The number of cell faults equals the number of possible inputs multiplied by the number of faulty outputs $2^m(2^n-1)$.
- *LUT as RAM:* If the LUT is operated in RAM mode, the following memory faults [van93] are targeted: stuck-at faults, address decoder faults, transition faults, coupling faults, and data retention faults.
- *Sequential elements:* CLBs contain sequential elements such as flip-flops, latches, or LUTs in shift register mode. For these elements, the commonly used fault models in hardware testing are considered [WWW06], i.e. stuck-at and transition faults.
- *Interconnects:* The stuck-at faults assigned to all input and output ports of the logic primitives are targeted such that all faults at signal fanout stems and their branches are tested.

4.4.2 Test Configurations for PRET

A full test session consists of multiple test configurations (TCs), each of which tests a subset of the logic primitives in the CLBs of a region or a set of interconnects used by the accelerator to be configured. Altogether nine TCs are required to test all logic primitives in the CLBs [ABB+12], and another nine TCs are required to test the interconnects of the nine accelerators of the H.264 application [BBI+13] (see Table 3.1). Partial bitstreams for these TCs are stored in the system memory.

Table 4.1 provides an overview of the 18 TCs for PRET. 9 TCs for testing CLBs are labeled TC 1-9 and the 9 TCs for the interconnects are labeled 10-18. Column one shows the configuration number. Column two shortly describes the parts of fabric under tested. Columns three and four list the area overhead of PRET in CLBs used for the TPG and ORA and the size of the generated partial bitstreams. The total area overhead introduced by PRET for all TCs is 17 CLBs. The total test time for a region consists of two parts: the configuration time of the TC and the test pattern application time (see 'Test length' in Table 4.1). Typically, the latter ranges from a few cycles up to a few hundred cycles. For instance, applying all test patterns for TC 9 (the TC with the largest number of patterns) lasts 3.2 µs at 100 MHz system frequency. The configuration time of TCs dominates the test

Table 4.1: Test configurations for CLBs and interconnects: Overhead, size, frequency and length

TC	Tested primitives	PRET overh. *[CLBs]*	Bitstr. size *[KB]*	Freq. *[MHz]*	Test length *[Patterns]*
1	LUT conf. as XOR, connected to FF	2	24.0	207	64
2	LUT conf. as XNOR, connected to FF	2	24.0	207	64
3	Carry MUX, interleaved with MUX and latch	1	28.6	168	6
4	Carry MUX, interleaved with MUX and latch	1	26.1	154	6
5	Carry XOR, interleaved with MUX and FF	1	28.0	168	6
6	Carry XOR, interleaved with MUX and FF	1	28.2	154	6
7	Carry-in/-out with multiplexed scan chain	1	27.1	183	6
8	LUT conf. as SR with slice MUX	1	22.9	157	6
9	LUT conf. as RAM with slice output	7	22.3	225	320
10-18	Interconnect and PIPs of the nine accelerators	n.a.	29.6	78.8-191.9	13-123

time with tens of thousands of cycles and is directly proportional to the size of the configuration data (partial bitstreams) and the reconfiguration bandwidth. As shown in Table 4.1, the bitstream size of each TC varies from 22.3 KB to 29.6 KB, which corresponds to a configuration time between 0.45 ms and 0.59 ms at 50 MB/s configuration bandwidth, i.e. between 45 and 59 thousand cycles at 100 MHz system frequency.

The PRET overhead for the interconnect TCs is not applicable as the deterministic patterns are not generated by a TPG but stored similar to PORT patterns. For response compaction, the same compaction unit for PORT is reused. In total 3780 bytes are required to store the test patterns of all interconnect TCs together with their signatures. One of the nine accelerators (and its corresponding TC) requires two clock cycles for execution (78.8 MHz). All others require only a single clock cycle and have a frequency higher than 100 MHz. All interconnect TCs reach a fault coverage of 100%, except for SADrow_4 with a fault coverage of 98.28%.

4.4.3 PRET Scheduling

Figure 4.3 shows the simulation results for the performance loss under different test frequencies, depending on the number of reconfigurable regions. The test frequencies vary from one test configuration before every accelerator configuration (1 TC/AC) to one test configuration before every 4th accelerator configuration (1 TC/4 ACs). Using a lower test frequency (e.g. 1 TC/4 ACs) reduces the overhead. The PRET handler is triggered every 1 ms and performs PRET if a region has not been tested for 500 ms. For reference, in a system with 10 regions and without PRET/PORT, the time between two consecutive accelerator configurations in a region ranges from 13.2 ms to 1240 ms (average: 200 ms).

Systems with more regions have a lower runtime overhead as more regions are still operational during the test application. For instance, in a system with 5 regions, only 4 regions can be used for acceleration during the PRET reconfiguration and pattern application period, whereas in a system with 14 regions, still 13 regions can be used for acceleration.

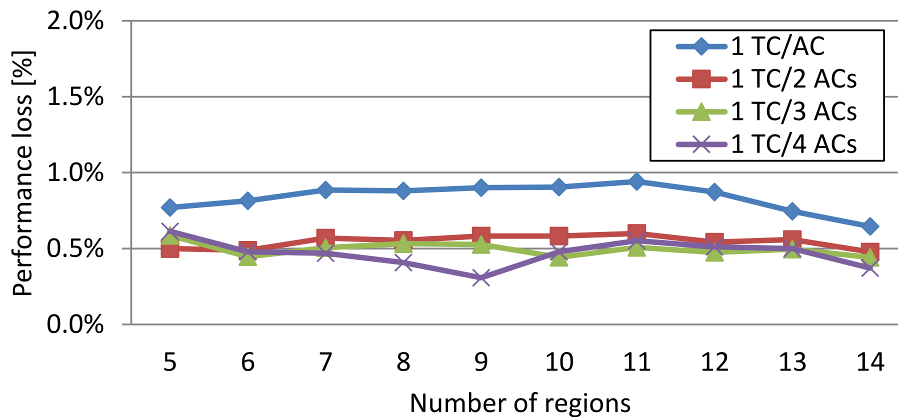

Figure 4.3: Performance loss of the video encoder application under different on-demand PRET frequencies and number of regions

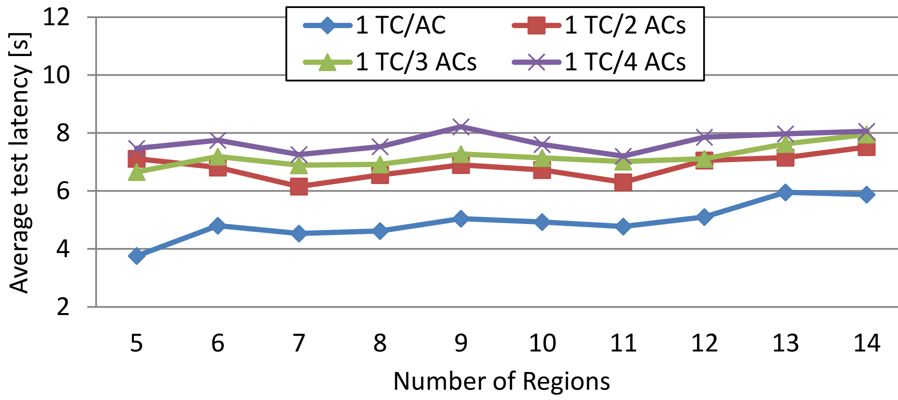

Figure 4.4: Average test latency under different PRET frequencies and number of regions

Figure 4.4 shows the average test latency. For example, for a system with 10 regions and a test frequency of 1 TC/3 ACs, each region is completely tested (by all TCs) every 7.1 seconds while the performance loss introduced by PRET is only 0.5%. The observed test latencies (3.8 s to 8.1 s) show that emergent faults do not remain undetected in the system for longer than 1.9 s to 4.05 s in average.

As shown in Fig. 4.5, with decreasing on-demand PRET frequencies, the number of on-demand tests (solid lines) decreases while the number of periodic tests (dashed lines) increases. A low on-demand PRET frequency increases the chance that a periodic test will be triggered because the possibility that a region remains untested for a time that exceeds the threshold (in our experiment 500 ms) is higher. For systems with a large number of regions, all accelerators can be fit into the regions and available for the application at the same time. Hence, less reconfigurations are required leading to fewer on-demand tests and a higher number of periodic tests.

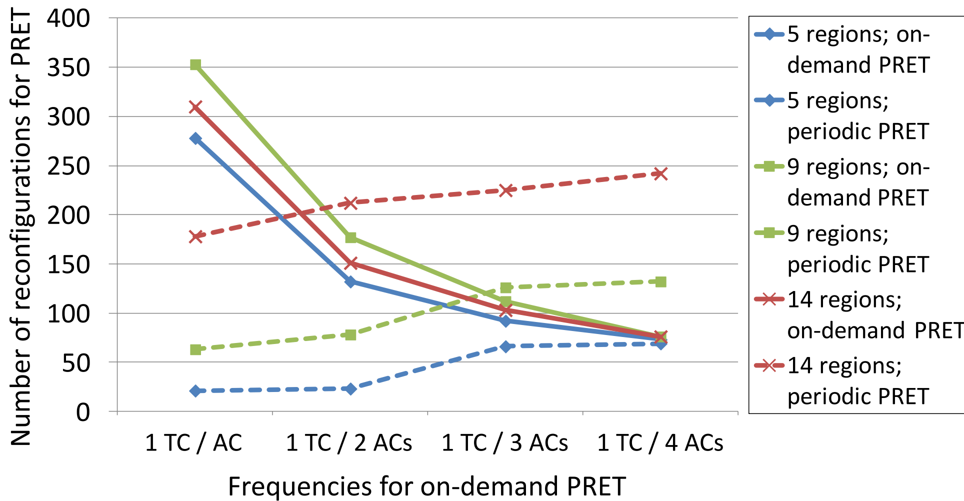

Figure 4.5: Comparison of the number of on-demand and periodic tests for different on-demand PRET frequencies and number of regions

4.4.4 PORT Scheduling

one PORT is scheduled right after each accelerator configuration and periodically over runtime to test all configured accelerators in the fabric for functional integrity. Since PORT is implemented as a dedicated test-AF (see Section 4.2), the application execution time is affected by PORT. When the total execution time without PORT is t_{base}, then the total execution time with activated PORT t_{PORT} can be expressed as

$$t_{PORT} = t_{base} + t_{PORT} \cdot f_P \cdot d + n_C \cdot d, \quad (4.1)$$

where f_P is the frequency of periodic PORT executions, d is the duration in cycles of one PORT execution, and n_C is the number of accelerator configurations. Therefore, the performance loss due to PORT is

$$\frac{t_{PORT} - t_{base}}{t_{base}} = \frac{1 + n_C \cdot d / t_{base}}{1 - f_P \cdot d} - 1. \quad (4.2)$$

Since for the H.264 video encoder the term $n_C \cdot d / t_{base}$ is significantly smaller than 1, the performance loss is dominated by the periodic PORT. The upper part of Table 4.2 shows the performance loss due to PORT for different PORT frequencies from 143 Hz to 1,000 Hz, i.e. test periods from 1 ms to 7 ms. For each PORT frequency, the table shows the minimum and maximum performance loss of 10 reconfigurable systems with different number of regions (5 to 14). The small difference between the minimum and the maximum values shows that PORT is basically unaffected by an increasing number of regions. This is because one execution of a PORT test-AF tests all configured accelerators at the same time (see Section 4.2). Altogether, the performance overhead due to PORT is very low (between 0.51% and 3.73%) and scales well with higher PORT frequencies.

Table 4.2: PORT performance loss and worst case test latency under different PORT frequencies

	PORT frequency *[Hz]*						
Performance loss	143	167	200	250	333	500	1000
min.* *[%]*	0.51	0.59	0.72	0.89	1.20	1.81	3.68
max.* *[%]*	0.56	0.63	0.75	0.92	1.23	1.85	3.73

	PORT frequency *[Hz]*						
Worst case** test latency	143	167	200	250	333	500	1000
min.* *[ms]*	7.0	6.0	5.0	4.1	3.3	2.3	1.7
max.* *[ms]*	7.8	6.8	5.8	4.8	3.8	2.8	1.8

* Summarizing 10 reconfigurable systems with 5 to 14 regions.
** Corresponds to the longest time period in the whole runtime in which a configured accelerator remains untested.

In addition to the configured test frequency, the following further situations affect the actual test latency of a region:

(1) If PORT is scheduled while an AF is being executed in the reconfigurable fabric, then PORT must be delayed until the AF execution finishes.

(2) If PORT is scheduled right before or after an accelerator configuration, then all configured accelerators (in all regions) are tested twice in a very short period.

(3) During the reconfiguration of a region, no PORT can be executed for that region, as no accelerator is available in that region during reconfiguration.

In situations (1) and (3), the test latency of a region is prolonged while in (2) it is shortened. The observed worst case test latency, which corresponds to the longest untested time period of a region is shown in the lower part of Table 4.2.

4.4.5 Combined PRET and PORT Scheduling

With PRET and PORT both enabled, the system is able to defend the configured accelerators against structural faults induced by aging effects or latent faults and transient events such as crosstalk or radiation. Since both test schemes differ in their test intervals and test methods, they do not interfere with each other. Fig. 4.6 shows the simulation results for the performance loss of a system with 5 regions when both PRET and PORT are enabled. All combinations of PRET and PORT frequencies used in previous sections are applied.

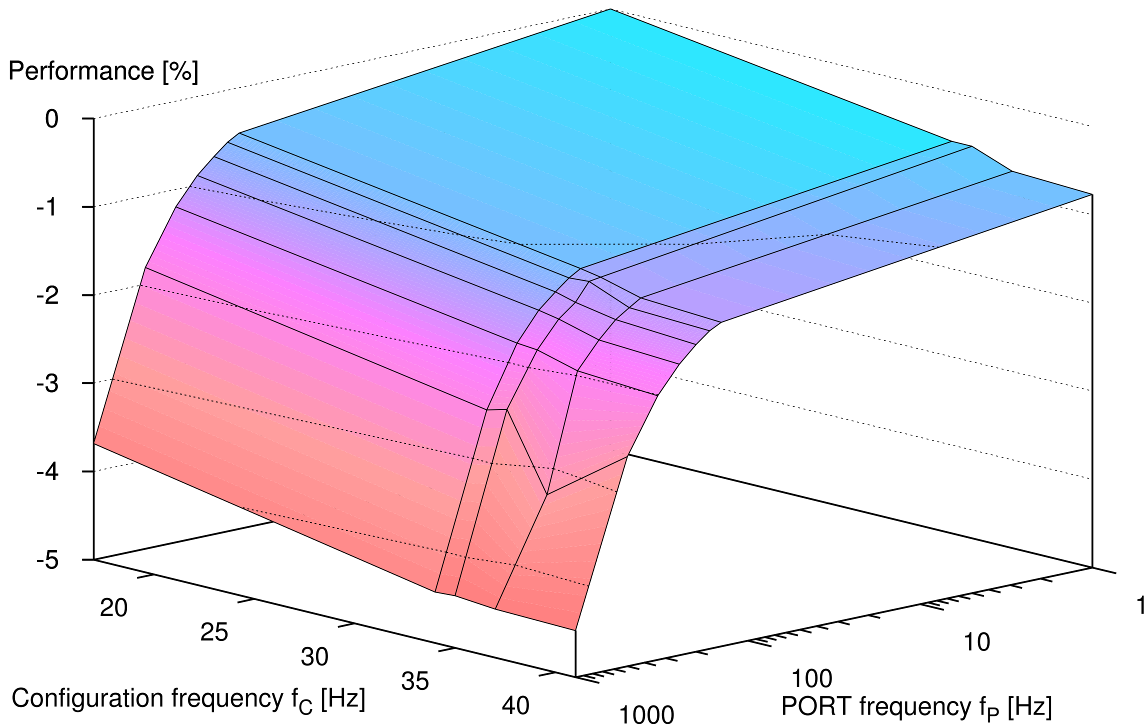

Figure 4.6: Performance loss when both PRET and PORT are applied for a reconfigurable system with 5 regions

The average configuration frequency f_C is determined by considering all reconfigurations, i.e. accelerator configurations (AC) and test configurations (TC). The lowest configuration frequency of 17 Hz corresponds to the case where on-demand and periodic PRET is disabled, i.e. only accelerator configurations are performed. When enabling PRET, the configuration frequency doubles to 34–41 Hz, but due to the PRET scheduling that distributes the TCs over time, the performance loss remains limited.

The highest configuration frequency of 41 Hz in Fig. 4.6 is obtained for the highest on-demand PRET frequency of 1 TC/AC. For lower PRET frequencies (1 TC/2 ACs and 1 TC/3 ACs), the configuration frequencies reduce correspondingly (35 Hz and 34 Hz). For an on-demand PRET frequency of 1 TC/4 ACs, the configuration frequency increases again (37 Hz), because more periodic PRETs are executed due to the reduced number of on-demand PRETs. That explains the bend that is visible in Fig. 4.6 for $f_C = 37\,\text{Hz}$.

For a PORT frequency f_P of less than 100 Hz the performance loss is dominated by the configuration frequency f_C. After that point, the PORT frequency dominates the performance loss. The highest performance loss of 4.4% occurs for a PORT frequency of 1,000 Hz and a configuration frequency of 41 Hz.

5 Self-Repair by Module Diversification

If a CLB in the reconfigurable fabric is detected to be faulty, the faulty CLBs shall be isolated from the system, i.e. not usable anymore for accelerators, as they may lead to erroneous computations. This isolation can be performed in two granularities, either disabling the usage of the regions that contains the faulty CLBs or forbidding configuring any accelerators that require the faulty CLB. Both isolation strategies limits the number of usable regions for accelerators. Certain accelerators required by the application may failed to be configured due to the fault isolation and thus the application performance is degraded due to unavailable accelerators.

5.1 Overview of the Module Diversification Method

This thesis proposes a novel design method called *module diversification* to tolerate permanent or intermittent faults in the reconfigurable regions. For each module/accelerator, a set of configurations is generated that is diversified in terms of their CLB usages, such that for every CLB in a region, at least one configuration of a module does not require that CLB.

A CLB is considered as faulty if it is affected by one or multiple permanent or intermittent faults. The type of faults within a CLB is not distinguished. The proposed module diversification method enables the system to tolerate at least any single-CLB faults and part of multi-CLB faults. If a fault is localized in a region, a diversified configuration of a module can be reconfigured into the region at runtime that does not use the faulty CLB. Self-repairing is achieved from the application perspective since in the present of faults the application experiences no performance degradation.

A generic algorithm is developed to generate the minimal set of configurations to tolerate arbitrary single-CLB faults and to generate additional configurations to tolerate multi-CLB faults in a reconfigurable region. The relationship between the required number of configurations, amount of spare resources, and reliability is investigated. In addition, since the number of these configurations shall be as small as possible to reduce storage overhead, these alternative configurations are inherently highly diversified, i.e. the number of common CLBs of two different configurations is as small as possible.

5.2 Diversified Configurations

An module defines the logic functions to be implemented in a region which consists of CLBs that are arranged regularly in a 2-dimensional array in the FPGA. The *configuration* of the module determines which CLBs in the region are used to implement the functionality.

5.2.1 Matrix Representation of Configurations

A natural way to describe the CLB usage of a configuration is to use a Boolean matrix. The size of the matrix matches the size of the region: A rectangular region with X CLBs in width and Y CLBs in height requires an $X \times Y$ matrix. If a CLB is used, the corresponding matrix element is 1, otherwise 0. This Boolean matrix is called a *configuration matrix.* For example, a module configuration using 5 CLBs implemented in a 3×3 region can be represented in a configuration matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.1}$$

This module uses 5 out of 9 CLB resources. This matrix represents one possible configuration of the module.

5.2.2 Properties of Diversified Configurations

The module diversification method generates a set of configurations, each of which implements the same module, but uses different CLB resources, such that any single-CLB fault in a region can be tolerated by one of the diversified configurations. Formally, we search a set of configurations C for a module implemented in an $X \times Y$ region.

$$C = \{\mathbf{A_1}, \cdots, \mathbf{A_w}\},\ \mathbf{A_i} : X \times Y \text{ Boolean matrix} \tag{5.2}$$

Assume that all of these configurations utilize the same amount of CLBs U and there is at least one free CLB, i.e.

$$\forall \mathbf{A_i} \in C : \sum_{x,y} [\mathbf{A_i}]_{x,y} = U < XY \tag{5.3}$$

To be able to tolerate any single-CLB fault, this set of configurations must satisfy the *completeness condition*:

$$\begin{aligned} &\forall x, y, 1 \leq x \leq X, 1 \leq y \leq Y : \\ &\exists \mathbf{A_i} \in C \text{ such that } [\mathbf{A_i}]_{x,y} = 0 \end{aligned} \tag{5.4}$$

The completeness condition guarantees that if any CLB is detected to contain faults, there always exists a diversified configuration $\mathbf{A_i}$ that does not require the faulty

CLB. For a module requiring U CLBs to be implemented in an $X \times Y$ region, at least w_{min} configurations are required for the completeness condition:

$$w_{min} = \lceil \frac{XY}{XY-U} \rceil \tag{5.5}$$

In each configuration, exactly $XY - U$ CLBs are spare. For a configuration $\mathbf{A_i}$, at most $XY - U$ CLBs that were not spare in any of the configurations $\mathbf{A_j}, j < i$ can be spare in $\mathbf{A_i}$, which directly results in this lower bound.

In order to minimize the number of diversified configurations for satisfying the completeness condition and to improve the effect of aging mitigation, it is required that the generated set of configurations also satisfies the *max diversification condition*:

$$\begin{gathered} \forall i, 1 \le i \le w : \exists \mathbf{A_j} \in C, j \neq i \text{ such that} \\ \sum_{x,y} \Big([\mathbf{A_i}]_{xy} \cdot [\mathbf{A_j}]_{xy} \Big) = \begin{cases} 2U - XY & \text{if } U > \frac{1}{2} XY \\ 0 & \text{else} \end{cases} \end{gathered} \tag{5.6}$$

Two configurations are maximally diversified if the difference between them is maximized. The minimum number of common CLBs between two configurations is either 0, if the module requires at most half of the available CLB resources, or $2U - XY$, whenever all $XY - U$ unused CLBs in one configuration are used in the other configuration. In the latter case, the number of common CLBs is $U - (XY - U)$. The max diversification condition states that for every configuration $\mathbf{A_i} \in C$ there is at least one other configuration $\mathbf{A_j}$ which differs from $\mathbf{A_i}$ as much as possible w.r.t. the used CLB resources.

For example, consider a module requiring 5 CLBs to be implemented in a 3×3 region. The following set of configurations satisfies the completeness condition but does not satisfy the max diversification condition:

$$\mathbf{A_1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \mathbf{A_2} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{A_3} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{5.7}$$

When 5 out of 3×3 CLBs are used, the minimal possible number of common CLBs between two configurations is 1 CLB (see Eq. (5.6)). Yet, in the above 3 configurations, all pairs of configurations have at least 2 CLBs in common. One possible set of configurations that satisfies both conditions is as follows:

$$\mathbf{A_1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \mathbf{A_2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \mathbf{A_3} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.8}$$

5.3 Generation Algorithm

Enumerating all possible configurations to find a maximally diversified set of configurations is computationally intractable. For instance, if a module requires 50 CLBs

in a region with 80 CLBs, then there are $\binom{80}{50} \approx 9 \times 10^{21}$ possible configurations. Alg. 2 presents the generation of a given number of configurations that satisfy the completeness condition and maximizes their diversity. It incrementally generates diversified configurations from an initial configuration $\mathbf{A_1}$.

Algorithm 2 Generation of diversified configurations C

1. $C := \{\mathbf{A_1}\}$ // $\mathbf{A_1}$ is the initial configuration
2. $\mathbf{G} := \mathbf{A_1}$ // Score matrix $\mathbf{G}$ stores swapping priority of CLBs
3. $\mathbf{A_{new}} := \mathbf{A_1}$
4. **loop**
5. zero_elem_list $:= \{(x,y) \mid [\mathbf{A_{new}}]_{xy} = 0\}$ // unused CLBs
6. candidate_list $:= \{(x,y) \mid [\mathbf{A_{new}}]_{xy} = 1\}$
7. sort candidate_list according to the value of $\mathbf{G}_{xy}$ in descending order // first element has the highest score
8. **for all** (x,y) in zero_elem_list **do**
9. swap_candidates $:= \{(p,q) \mid (p,q) \in$ candidate_list and $\mathbf{G}_{pq} = \mathbf{G}_{\texttt{candidate_list[0]}}\}$ // all CLBs with the highest score
10. farthest_swap_candidate $:= (p,q) \in$ swap_candidates with max. Manhattan distance between (x,y) and (p,q) // farthest elements are swapped first so that CLBs are located near each other and better timing is achieved
11. swap($[\mathbf{A_{new}}]_{xy}$, $[\mathbf{A_{new}}]_{\texttt{farthest_swap_candidate}}$)
12. candidate_list.pop(farthest_swap_candidate)
13. **if** candidate_list $= \emptyset$ **then**
14. **break**
15. **end if**
16. **end for**
17. **while** $\mathbf{A_{new}} \in C$ **do**
18. swap a random zero- with random one-element in $\mathbf{A_{new}}$
19. **end while**
20. $\mathbf{G} := \mathbf{G} + \mathbf{A_{new}}$ // update CLB score
21. $C := C \cup \{\mathbf{A_{new}}\}$
22. **if** $|C| = \text{desired number of config.} \vee |C| = \binom{XY}{U}$ **then**
23. **break**
24. **end if**
25. **end loop**

In Line 30, the set of diversified configurations C is initialized with the initial configuration. The score matrix $\mathbf{G}$, which has the same dimension as the configuration matrix, stores for each CLB the number of diversified configurations which use that CLB. The score matrix is simply the sum of all configuration matrices in C. In Line 31, $\mathbf{G}$ is initialized to $\mathbf{A_1}$, the only element in C at the moment. In Line 32, the next new configuration matrix $\mathbf{A_{new}}$ is initialized to the initial configuration matrix. In the inner loop (Lines 37 to 45), it is further modified by swapping zero- and one-elements. The inner loop iterates through all zero-elements in $\mathbf{A_{new}}$ and swaps zero-elements with one-elements in $\mathbf{A_{new}}$ in an order determined by the score matrix (Line 36). If a CLB has a higher score (i.e. it is used more often in the diversified configurations), its corresponding one-element in $\mathbf{A_{new}}$ will be first swapped. If there are several CLBs with the same score, the farthest one from the current zero-element is swapped first (Lines 38 to 40) so that in the resulting con-

figuration, the used CLBs are located near each other. The first generated $\lceil \frac{XY}{XY-U} \rceil$ configurations correspond to the *minimal* set of configurations that satisfies both the completeness and max diversification condition (see proof in Appendix A). It is guaranteed that the random swapping (Line 47) does not occur while generating the minimal set.

If the user requires more configurations for higher reliability (i.e. tolerate more multi-CLB faults) or to have more alternatives for aging mitigation by stress balancing, further possible configurations can be generated (this might use the random swapping in Line 47 at some time). The algorithm terminates when either the desired number of configurations or all possible configurations have been generated. In both cases, the generated set of configurations always satisfies the completeness condition but may violate the max diversification condition due to the while loop from Lines 46 to 48, where random changes are made to $\mathbf{A_{new}}$ to generate a new unique configuration matrix.

5.4 Reliability Analysis

The reliability of an entity is the probability that it operates without failure for at least the specified time period t. Let $R_{CLB}(t)$ be the reliability of a CLB at time t. Without any fault-tolerance techniques applied, the reliability of a module using U out of $X \times Y$ CLBs is

$$R_{\text{No FT}}(t) = R_{CLB}(t)^{U}, \tag{5.9}$$

i.e. all U CLBs are required to be operational to allow the module to operate without failure. Using the module diversification method, the reliability of the module can be increased: In case of CLB failures, the module can be reconfigured with a diversified configuration such that only operational CLBs are used by the configured module. In this case, the reliability of the module becomes:

$$R_{Div}(t) = R_{CLB}(t)^{XY} + \sum_{f=1}^{XY} \underbrace{C_f \alpha_f \binom{XY}{f} (1 - R_{CLB}(t))^f R_{CLB}(t)^{(XY-f)}}_{\text{Probability that } f\text{-fold CLB failures can be tolerated}} \tag{5.10}$$

The first term states the probability that all CLBs are fault free. The second term aggregates all the scenarios where only a single CLB is faulty, two CLBs are faulty, three CLBs are faulty, ..., all CLBs are faulty.

Fault coverage C_f, $0 \le C_f \le 1$, is the fraction of f-CLB faults which are detected by an online test or concurrent error detection scheme (see Section 2.5) such that reconfiguration with a diversified configuration allows to continue the module operation.

The number α_f, $0 \le \alpha_f \le 1$ denotes the fraction of f-CLB faults which can be tolerated with the set of configurations generated by the module diversification method. For example, $\alpha_2 = 0.5$ means 50% of 2-CLB faults can be tolerated by loading a diversified configuration. The completeness condition of Eq. (5.4) guarantees that

any single-CLB fault can be tolerated. Therefore, for every generated set of configurations we have $\alpha_1 = 1$. The values of α_f for $f \geq 2$ depend on the placement details of each configuration and need to be calculated from the generated set of configurations.

5.5 Diversification for Interconnect Resources

The proposed module diversification design method is in principle applicable for all regularly distributed resources of the fabric. In Xilinx FPGAs, the routing resources are regularly distributed: One programmable switching matrix is attached to each CLB. Thus, the resource usage patterns for target configurations computed by the proposed method can also diversify the use of programmable routing resources.

5.6 Implementation Flow

This section explains the overall flow of the generation of diversified configurations and tool integration using the Xilinx tool flow. The Xilinx place-and-route tools support the `PROHIBIT` placement constraint [Xil12a], which prevents the place-and-route tool to use specific resources such as CLBs or Block RAMs at specified locations[1]. In the following, this constraint is employed to implement diversified configurations in term of CLB usages.

As shown in Fig. 5.1, an initial configuration is generated for the module by synthesis and place-and-route of the original design files. From this configuration, the used CLBs are extracted and stored in the matrix $\mathbf{A_1}$.

Using Alg. 2, the diversified configuration matrices $\mathbf{A_i}$ which specify the diversified CLB usage is computed. They are exported as PROHIBIT placement constraints and then provided to the Xilinx place-and-route tools which produce the final set of

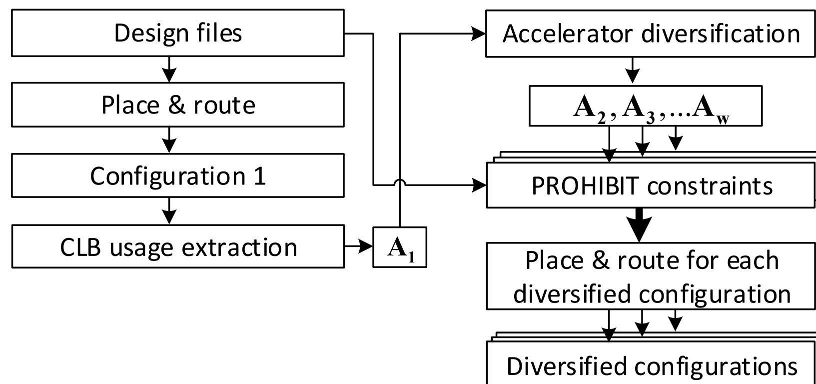

Figure 5.1: Generation of diversified configurations using the module diversification method

[1]Currently the PROHIBIT constraint is not effective/supported for routing resources.

diversified configurations. An example constraint file for a diversified configuration of module `alu4` implemented in a 4×20 region is as follows:

```
INST "alu4_inst" AREA_GROUP = "pblock_alu4_inst";
AREA_GROUP "pblock_alu4_inst" RANGE=SLICE_X48Y80:SLICE_X57Y99;
PIN "alu4_inst.i_0" LOC = SLICE_X56Y80;
PIN "alu4_inst.i_1" LOC = SLICE_X56Y80;
⋮       ⋮       ⋮
PIN "alu4_inst.o_0" LOC = SLICE_X57Y80;
PIN "alu4_inst.o_1" LOC = SLICE_X57Y80;
⋮       ⋮       ⋮
CONFIG PROHIBIT=SLICE_X56Y84;
CONFIG PROHIBIT=SLICE_X56Y85;
⋮       ⋮       ⋮
CONFIG PROHIBIT=SLICE_X53Y80;
CONFIG PROHIBIT=SLICE_X54Y80;
```

5.7 Experimental Evaluation

The module diversification method was applied to a set of functional modules from MCNC benchmark suite [Yan91] and OpenCores[2]. The reliability improvement and timing cost were evaluated under different parameters such as the amount of redundant CLBs, reliability of CLB and number of configurations.

The target platform was a Xilinx Virtex-5 FPGA with reconfigurable regions that are 20 CLBs in height, or 80 CLBs for large modules from OpenCores, as recommended by Xilinx to align to the clock region boundary. The region width is varied from 3 up to 13 CLB columns to investigate different degrees of redundancy.

5.7.1 Timing Overhead

For each module/region-size combination, the minimal set of configurations is generated using the proposed module diversification method (tool-flow overview in Section 5.6). Table 5.1 summarizes the region setup and reports the minimal number of configurations and the timing costs of diversified configurations for every module.

Column 1 lists the implemented modules and column 2 shows the minimal (W_{min}) and maximal (W_{max}) used region width. The 3rd column shows the degree of CLB redundancy for different region sizes. For example, `apex4` uses 98 CLBs in the 20×6 region (i.e. W_{min}), which corresponds to $(20 \times 6 - 98)/98 \approx 22.4\%$ redundancy. Column 4 lists the minimum number of configurations tolerating all single-CLB faults in W_{min} and W_{max}. For larger regions with higher redundancy, fewer configurations are required.

Since the module diversification design method applies additional constraints to prohibit certain CLB placements, the maximally achievable frequency of a module may be affected. The last column in Table 5.1 reports the maximal frequency (over all region widths) of the original unconstrained module (Orig.) in comparison to

[2] http://www.opencores.org

Table 5.1: Configurations for different region sizes and maximal frequency of original (Orig.) and diversified (Div.) modules

Module	Region Width* [CLB]		CLB Redundancy [%]		Minimal #Config.		Frequency		
	W_{min}	W_{max}	W_{min}	W_{max}	W_{min}	W_{max}	Orig. [MHz]	Div. [MHz]	Δ[%]
pdc	3	5	9.1	64.0	12	3	150.8	145.2	3.7
misex3	4	7	11.1	81.8	10	3	136.5	123.3	9.7
alu4	4	7	3.9	81.8	27	3	130.4	127.5	2.3
apex4	6	9	22.4	111.8	6	2	126.2	114.5	9.3
apex2	6	11	14.3	117.8	8	2	122.4	115.3	6.2
des_perf	7	13	4.9	117.1	22	2	135.3	127.3	6.3
aes_core	3	5	27.7	127.3	5	3	124.7	124.7	0.04

* Region height for large OpenCore modules `des_perf` and `aes_core` is 80 CLBs. Region height for other modules is 20 CLBs.

the diversified modules (Div.). For the diversified modules, the reported maximal frequency always corresponds to the slowest configuration of the module. The timing cost is under 9.7%, which is a promising result for an approach that obtains fault tolerance at no additional area overhead. Note that the original module implementation is one of the diversified configurations and thus can be used when full performance is required. If the system frequency is lower than the maximal frequency of the diversified modules, there are no timing costs at all.

5.7.2 Reliability Improvement

In this section, the reliability improvement of the module diversification method is evaluated for different degrees of CLB redundancy, CLB reliabilities, and number of configurations.

Figure 5.2 shows the module reliability according to Eq. (5.10) of Section 5.4 of module `apex4` for a CLB reliability $R_{CLB}(t) = 0.999$, i.e. the probability of any single CLB to be operational throughout a given time period t is 0.999. We assume $C_f = 1.0$. The figure displays the reliability increase for different numbers of diversified configurations (from the minimum number of configurations up to 20) and for region sizes from 20×6 to 20×9 CLBs, which corresponds to CLB redundancies from 22.4% to 111.8%. Without any diversified configurations, the module reliability is very low at approximately 0.91. Using diversified configurations, the module reliability increases dramatically due to higher single- and multi-CLB fault tolerance from extra configurations. For example, three configurations are sufficient to tolerate all single-CLB faults for `apex4` implemented in a 20×8 region. In addition, 47% of double and 22% of triple-CLB faults can be tolerated with these three configurations as well. An extra set of 17 diversified configurations increases 2-CLB and 3-CLB fault tolerability further to 88% and 57%, respectively.

Larger region sizes imply higher CLB redundancy which reduces the probability

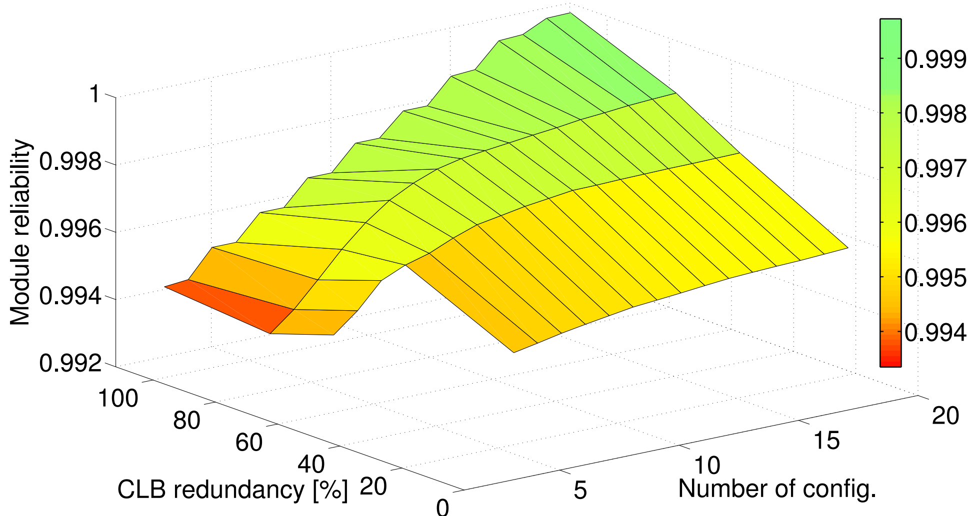

Figure 5.2: Module reliability of `apex4` for different ratios of CLB redundancy and number of configurations with CLB reliability 0.999

$R_{CLB}(t)^{XY}$ that *all* CLBs in the region are fault free (see Eq. (5.10)). This may reduce the overall module reliability as seen on the left in Figure 5.2. With increasing number of configurations, the tolerance of f-CLB faults rises and very high module reliability can be achieved.

Figure 5.3 shows the reliability of modules using module diversification versus a single configuration without fault tolerance measure for different CLB reliabilities, computed according to Eq. 5.10. For all modules, the minimal set of diversified configurations implemented in the smallest regions is evaluated. It is clear that with diversified configurations the module reliability is substantially higher than without any fault tolerance approaches. The module reliability of `des_perf` and `aes_core` ranges from 0.59 to 0.95, respectively 0.83 to 0.98, when module diversification is not applied. With module diversification, the reliability increases from 0.895 to 0.999 and from 0.980 to 0.9998, respectively.

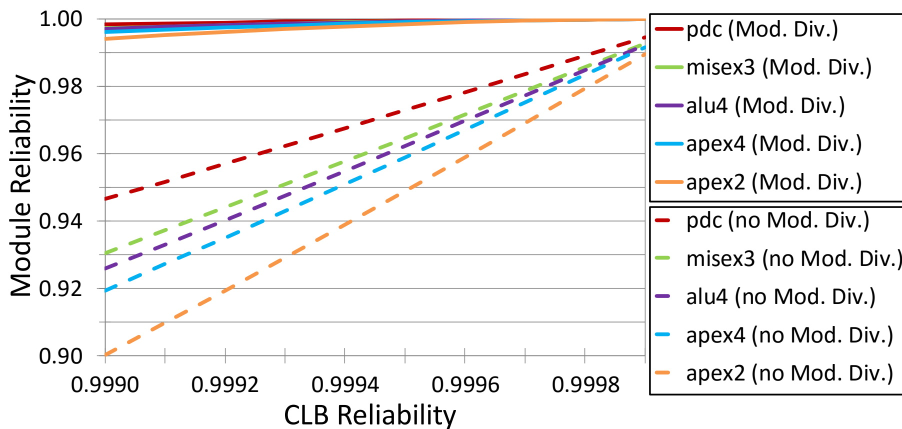

Figure 5.3: Module reliability *with* and *without* module diversification for different CLB reliabilities. Reliabilities of `des_perf` and `aes_core` are not shown in the figure for clarity, but discussed in the text.

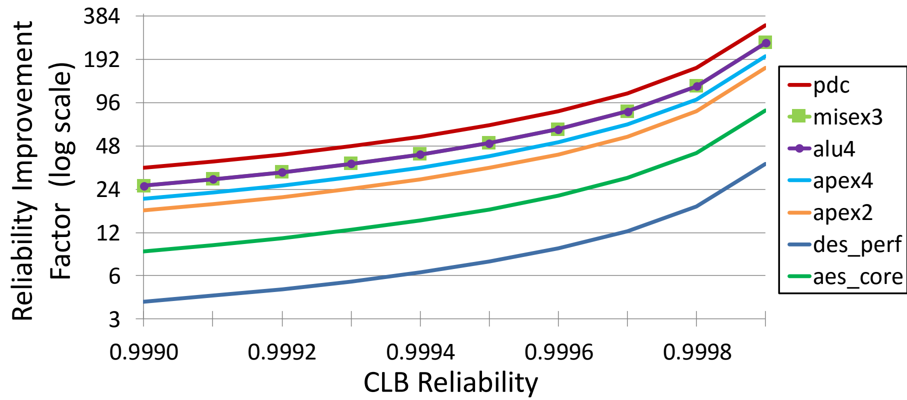

Figure 5.4: Reliability improvement factor for the modules when module diversification is applied

The reliability improvement factor (RIF) is a metric to estimate the effectiveness of fault tolerance schemes [Lal01]. The RIF is the ratio of the failure probability of the original system and the failure probability of the fault tolerant system, i.e. the system using diversified module configurations:

$$\text{RIF} = \frac{1 - R_{\text{No FT}}}{1 - R_{Div}} \tag{5.11}$$

Figure 5.4 plots the RIF for the five investigated modules and CLB reliabilities ranging from 0.9990 to 0.9999. With the proposed module diversification design method, reliability improvement factors of up to 330 are achieved.

6 Prolonging Lifetime via Stress Balancing

The FPGA-based reconfigurable fabric, manufactured in latest technology nodes (e.g. 20 nm/16 nm for Xilinx' UltraScale/UltraScale+ family), suffers from degradation due to aging [SWSC10, GBS14]. The resilience of the reconfigurable fabric is essential to the dependability of reconfigurable architectures, as most of the application's computations are offloaded to the reconfigurable fabric. The manifestations of aging can range from increased transistor switching delay up to permanent faults that cause a transistor or interconnect wire to fail entirely. Different aging mechanisms have been reported for the current generation of CMOS designs, as discussed in Section 2.3, e.g. Biased Temperature Instability (BTI), Time-Dependent Dielectric Breakdown (TDDB), Hot Carrier Injection (HCI), or Electro-Migration (EM).

The main causes of these effects are environmental and electrical *stress*. Stress can be induced in different ways, e.g. through the presence of strong electrical fields or high current density [SKM+08, SWSC10]. Due to the increasing susceptibility of ever-shrinking nano-CMOS devices, these effects cannot be ignored anymore and their consideration has become essential for dependable reconfigurable architectures [HBD+13] .

This work proposes a novel STRess-Aware Placement (STRAP) method to reduce the maximum stress by aging mitigation. For the first time, it combines complex offline optimizations at synthesis time with situation-dependent adaptation at runtime to optimize the intra- and inter-region stress distribution simultaneously. At the runtime, STRAP proposes an algorithm that places accelerators to different reconfigurable regions (i.e. it decides to which region they shall be reconfigured) while considering the induced intra- and inter-region stress distribution simultaneously. At the synthesis time, STRAP proposes an algorithm that diversifies stress during place-and-route by preventing overlapping of high stress CLBs from different accelerators, which further improves the intra-region stress distribution at runtime. For prototyping purposes, we have integrated STRAP into the Xilinx tool-chain and the runtime system of the target reconfigurable architecture.

6.1 Overview of the Stress-Aware Placement Method

The MTTF of a system is constrained by the component with the highest stress [SKM+08]. In order to prolong the MTTF of a reconfigurable fabric, stress accumulation on individual resources need to be avoid to reduce the peak stress. Fig. 6.1 shows a typical reconfigurable fabric with 8 reconfigurable regions and 4×20 CLBs per region. The figure visualizes the distribution of HCI stress after running an H.264 video encoder. Higher HCI stress corresponds to more toggles per second of a transistor (see Section 2.4.1). For each CLB, the highest toggle rate of any transistor

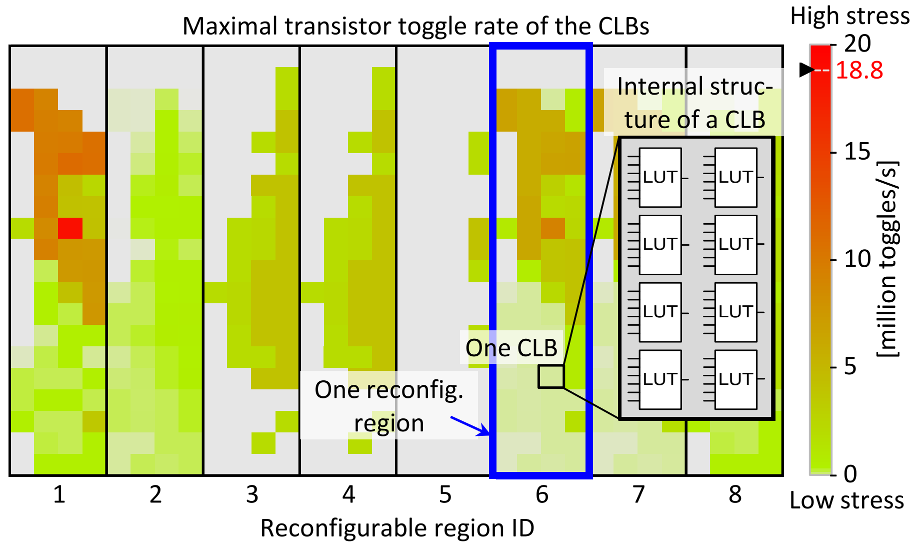

Figure 6.1: Transistor stress distribution in a reconfigurable fabric with 8 regions; each region consists of 4×20 CLBs with 8 LUTs each (same setup as for evaluation); the color of a CLB corresponds to the highest toggle rate of any of its transistors; the symbol ▶ on the right scale denotes the maximum stress over all regions

is identified and plotted in a color-scale from 0 (*low stress*, bright gray) to 20 million toggles per second (*high stress*, dark red). It is noticeable that several CLBs are not used, e.g. most parts of region 5 and some parts of regions 3 and 4, whereas some CLBs in region 1 contain transistors that are highly stressed. The latter represent *stress hotspots* where high stress accumulates in some of the components in the fabric which have a higher chance to fail much earlier than others, hence reducing the MTTF of the system.

The basic idea of STRAP is to place accelerators such that the maximal stress is minimized. The method considers stress at the granularity of CLBs, whereas the evaluation in Section 6.6 considers stress at transistor granularity. If the stress from a stress hotspot can be distribute to less stressed CLBs like in regions 3–5 in Fig. 6.1, then the maximum stress in the reconfigurable regions is reduced, leading to increased MTTF.

Figure 6.2 provides an overview of the stress-aware placement method STRAP, showing the synthesis time techniques, the runtime techniques, and how they interact with the hardware architecture of a reconfigurable system. For logic placement at synthesis time, the challenge is to place-and-route accelerators in a way that supports stress balancing at runtime, but without having runtime information. STRAP first performs an offline application profiling of each application kernel to obtain estimates on (i) how often accelerators will be executed relative to each other and (ii) how long each accelerator executes to finish its task. This information is used to steer runtime accelerator placement (Section 6.3) and synthesis time logic placement (Section 6.4).

Based on the accelerator configuration after place-and-route, the stress estimation process in Fig. 6.2 analyzes the signal activities in all CLBs used by the accelerator

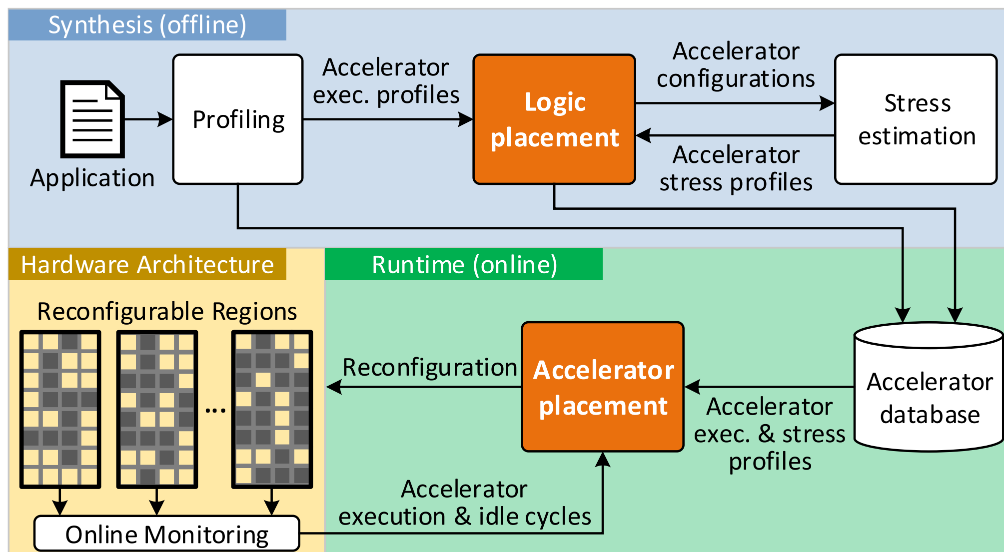

Figure 6.2: Overview of the stress-aware placement method

to obtain the information how much stress it induces to a reconfigurable region. Accelerator execution and stress profiles are stored together with the accelerator bitstreams in main memory for runtime decision making.

At runtime, STRAP decides into which reconfigurable region an accelerator shall be reconfigured, whenever the application demands different accelerators. It performs online monitoring of each region to track when the region was reconfigured last and how often the currently-reconfigured accelerator was executed. Whenever a region is reconfigured, the execution counter and reconfiguration timestamp is read and reset. Together with the accelerator stress profile created at synthesis time, STRAP then calculates the exact *stress state* for all CLBs of the region. This information is used to decide the runtime accelerator placement. Note that with the feature of partial reconfiguration provided by FPGA vendors, reconfigurable regions are spatially isolated from each other, i.e. the reconfiguration of one region does not affect the resource usage (and thus stress) of any other region.

6.2 Representation of Stress

In order to handle the transistor stress in an algorithmic way, it needs to be represented compactly to allow an efficient runtime computation for the stress states of regions and the placement decision making.

6.2.1 Stress Granularity

The transistors of a reconfigurable region are stressed by the reconfigured accelerator in a way that is determined by its logic functionality and input signal patterns. As the number of transistors in a region may be huge, the stress experienced by individual transistors is lumped to CLB granularity for the stress-aware placement

method. *CLB stress* is defined as the sum of the stress experienced by all transistors in a CLB. With this definition, CLB stress preserves the additive property of transistor stress, i.e. the total stress a CLB experienced from different accelerators is the sum of the induced stress from individual accelerators.

Runtime reconfigurable architectures can execute different applications. Each of them can use different accelerators that use different CLBs to implement their timing-critical path. As any reconfigurable region can be reconfigured to implement any of these accelerators, it is not possible to identify upfront which CLBs are more important than others w.r.t. protection against aging. Therefore, all CLBs in the reconfigurable fabric are treated equally important.

6.2.2 Stress Accumulation

With the established stress properties (see Section 2.4.2), the stress in the reconfigurable fabric can be described in a formal way. The stress state of a reconfigurable region (as it is visualized in Fig. 6.1) is denoted as matrix **S**, where each entry represents the stress experienced by the corresponding CLB in the region. The stress that a particular accelerator induces per clock cycle is obtained from offline stress estimation and called *unit stress*, denoted by a matrix of the same size as **S**. In general, the stress increase due to the work done by an accelerator is shown in Eq. (6.1), where scalars τ_{exec} and τ_{idle} denote the number of clock cycles when the accelerator is in execution or idle, while matrices $\mathbf{s}^{\mathbf{unit}}_{\mathbf{exec}}$ and $\mathbf{s}^{\mathbf{unit}}_{\mathbf{idle}}$ denote the unit stress induced by the accelerator during execution or idle time:

$$\mathbf{s} := \tau_{exec}\mathbf{s}^{\mathbf{unit}}_{\mathbf{exec}} + \tau_{idle}\mathbf{s}^{\mathbf{unit}}_{\mathbf{idle}} \tag{6.1}$$

During idle, we assume all inputs to the accelerator are hold at constant values, e.g. all zeros. In this case, the accelerator exhibits a different stress pattern from when it is being executed.

During synthesis time, the values for τ_{exec} and τ_{idle} are obtained from application profiling to construct the stress matrices (Eq. (6.1)) for every accelerator. They are used by the stress-diversifying logic placement and the runtime system. The runtime system uses them to determine how much stress an accelerator would induce to a region *before* actually placing it. It also uses online monitoring information (see Section 6.1) that provides the actual number of accelerator executions and idle times for each region *after* a computational kernel finished execution. This allows to keep track of the actual stress that a region experienced, which is the starting point for the next placement decision.

6.2.3 Stress Estimation Flow

Figure 6.3 shows the stress estimation flow for an accelerator. To obtain the unit stress of it, the placed-and-routed configuration and its input signal activities (toggle rate and average duty cycle) are fed to Xilinx XPower that computes the signal activity of every wire in the accelerator. The wires are then matched to the CLB

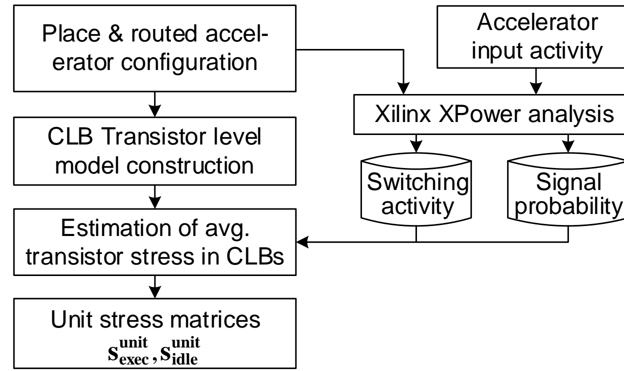

Figure 6.3: Stress estimation flow

inputs to obtain the input signal activities of every LUT in the CLBs used by the accelerator configuration. Based on the signal activity propagation through a transistor-level LUT model (see Section 2.1.3), the toggle rate and stress duty cycle of the LUT transistors are calculated.

As discussed in Section 2.1.3, a LUT can also be viewed as a tree of 2:1-multiplexers. All SRAM configuration cells of a LUT are connected to the data inputs of the first-level multiplexers and the LUT inputs are connected to the select signals of multiplexers in their respective level of the tree. The configuration SRAM cells are not on the critical path of accelerators during logic operations, because logic transitions in SRAM cells only happen when they are reconfigured. Therefore, stress in configuration SRAM cells is not explicitly targeted here.

For the calculation of the internal signal probabilities, the signal values at the multiplexer data inputs are weighted according to the duty time of the corresponding select input. Hence, for a multiplexer with input values v_0, v_1 and select signal sel, the output value is calculated by: $v_{out} := v_0 \cdot P[\overline{sel}] + v_1 \cdot P[sel]$, where $P[sel]$ is the probability that $sel = 1$. Once the output values of all multiplexers (and hence the inputs of each succeeding multiplexer) are determined, the calculation of the toggle propagation is performed.

In the toggle analysis, two types of switching sources are distinguished as shown in Fig. 6.4: (a) *propagated* toggles that are fed in through the multiplexer data inputs, and (b) *generated* toggles that spawn by changing the select signal. In the LUT model, the data inputs of the first-level multiplexers in the tree are connected to the configuration bits. Thus, upon a select signal switch, toggles can only be generated (if the two configuration bits have different values), but not propagated.

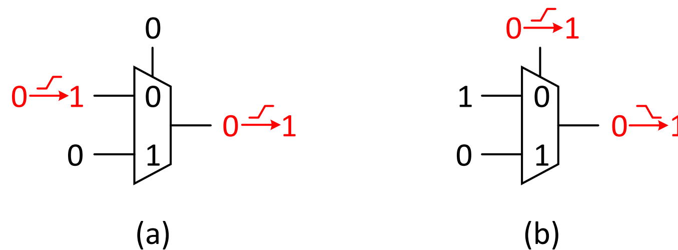

Figure 6.4: Toggle propagation (a) and generation (b) in multiplexers

On succeeding stages, the propagation of generated toggles then takes into account the switching activity at all of the input signals: Again, all sources of the toggles to be propagated from data inputs are weighted according to the signal probability at the multiplexer select input: $t_{prop} := t_0 \cdot P[\overline{sel}] + t_1 \cdot P[sel]$, where t_0 and t_1 are the toggle counts of the data inputs. As for the generated toggles, the likelihood of spawning a toggle after a select input switch is the XOR of the two data inputs multiplied by the toggle frequency f_{sel} as $t_{gen} := f_{sel} \cdot (v_0 \oplus v_1)$. The total toggle count at each multiplexer output is the sum of the propagated toggles and the toggles generated in its instance: $t_{tot} := t_{prop} + t_{gen}$.

Each multiplexer is composed of a pair of pass transistors, and thus the calculated signal probabilities at the data input and select signals of a multiplexer are directly mapped to the respective transistor terminals to obtain the stress duty cycles for the static stress. Similarly, for the dynamic stress, the number of toggles experienced by each transistor is derived from the toggle activities of the data inputs and the signal probability of the select signal.

6.3 Runtime Accelerator Placement

The reconfigurable fabric consists of N equally sized rectangular regions. During runtime, the application requests to configure M ($M \leq N$) accelerators to speed up its computational kernels. The runtime system has to decide to which regions the M accelerators shall be configured. For $M < N$ application-requested accelerators, the runtime system first decides which $N - M$ regions shall *not* be reconfigured, e.g. by using a least recently used replacement policy. The decision to which of the remaining regions an accelerator is placed does not affect the application performance.

Each region contains $X \times Y$ CLBs with an (x, y) coordinate relative to the top-leftmost CLB in the region. The stress experienced so far by the CLBs in region k is denoted as $[\mathbf{S_k}]_{xy}$ (with $1 \leq k \leq N$, $1 \leq x \leq X$, $1 \leq y \leq Y$). Similarly, the stress that will be induced by an accelerator j ($1 \leq j \leq M$) is denoted as $[\mathbf{s_j}]_{xy}$ (see Eq. (6.1)). It depends on how often the accelerator will be executed, as determined by offline profiling (see Section 6.1). If an accelerator j is placed into region k, then the accelerator executions increase the stress state of the region to $\mathbf{S'_k} = \mathbf{S_k} + \mathbf{s_j}$.

The problem is to place each accelerator to a region, such that upon completion of the application kernel the maximum CLB stress over the N regions is minimized, i.e. $\max_{k,x,y}[\mathbf{S'_k}]_{xy}$ is minimized. It can be easily seen that the strict lower bound of the maximum CLB stress is

$$\frac{1}{NXY}\left(\sum_{k}^{N}\sum_{x,y}[\mathbf{S_k}]_{xy} + \sum_{j}^{M}\sum_{x,y}[\mathbf{s_j}]_{xy}\right) \tag{6.2}$$

which is reached if and only if the stress is uniformly distributed over all CLBs. Therefore, to minimize the maximum CLB stress in the reconfigurable fabric, the CLB stress from the accelerators that are to be placed needs to be distributed evenly. To achieve this at runtime, this thesis proposes a heuristic that follows

these two rules: 1) maximal utilization of under-stressed CLBs within one region, i.e. the stress shall be evenly distributed among different CLBs within the region (*intra-region* distribution), and 2) avoid placing high-stress accelerators into highly stressed regions, i.e. the stress shall be evenly distributed among different regions (*inter-region* distribution). Mathematically, the conformance to these two rules is formulated as a profit function.

6.3.1 Placement Profit

The profit function of placing accelerator j into region k is defined as

$$\text{Profit}_{jk} = \text{Profit}_{jk}^{intra} + \text{Profit}_{jk}^{inter}, \tag{6.3}$$

where $\text{Profit}_{jk}^{intra}$ and $\text{Profit}_{jk}^{inter}$ represent the profit from the stress distribution within one region and across all regions, respectively:

$$\begin{gathered}\text{Profit}_{jk}^{intra} = \sum_{x,y}\left|[\mathbf{S_k}]_{xy} - \lambda_k\right| - \sum_{x,y}\left|[\mathbf{S_k}+\mathbf{s_j}]_{xy} - \lambda'_{k,j}\right| \\ \text{with } \lambda_k = \frac{1}{XY}\sum_{x,y}[\mathbf{S_k}]_{xy} \text{ and } \lambda'_{k,j} = \frac{1}{XY}\sum_{x,y}[\mathbf{S_k}+\mathbf{s_j}]_{xy}\end{gathered} \tag{6.4}$$

$$\begin{gathered}\text{Profit}_{jk}^{inter} = \left|\sum_{x,y}[\mathbf{S_k}]_{xy} - \Lambda\right| - \left|\sum_{x,y}[\mathbf{S_k}+\mathbf{s_j}]_{xy} - \Lambda'\right| \\ \text{with } \Lambda = \frac{1}{N}\sum_{k,x,y}[\mathbf{S_k}]_{xy} \text{ and } \Lambda' = \frac{1}{N}\left(\sum_{k,x,y}[\mathbf{S_k}]_{xy} + \sum_{j,x,y}[\mathbf{s_j}]_{xy}\right)\end{gathered} \tag{6.5}$$

The two summation operations in the intra-region profit function in Eq. (6.4) express the sum of the CLB stress deviation from the average stress value before and after placing accelerator j into region k, respectively. A larger sum of deviation implies that more CLBs are *over-* or *under*-stressed. This profit function thus describes the improvement of stress distribution within region k after placing accelerator j into it. In a similar manner, the inter-region profit function in Eq. (6.5) describes the deviation from perfect even stress distribution evaluated at the level of reconfigurable regions, i.e. the deviation of the total stress in a region from the average total stress per region.

6.3.2 Placement Algorithm

The stress-aware runtime accelerator placement (Alg. 3) iterates through all required accelerators (Lines 2 to 17). In each iteration, it calculates the profits of placing the accelerator into all available regions (Lines 5 to 14) and places the accelerator into the region that provides the highest profit (Line 15). The complexity of this algorithm is $\mathcal{O}(M^2XY)$. If the application decides to keep an accelerator configuration for a longer time (i.e. to not reconfigure it), then the stress may not be distributed evenly to all regions. The region where this accelerator resides would be

Algorithm 3 Stress-aware runtime accelerator placement

Input: List of accelerators `Acc` and list of regions `Reg` that shall be reconfigured

```
 1. occupied := array of length len(Reg) initialized to zeros
 2. for j := 1 to len(Acc) do
 3.   max_profit := −∞
 4.   selected_reg := null
 5.   for k := 1 to len(Reg) do
 6.     if occupied[k] == 1 then
 7.       continue
 8.     end if
 9.     profit := CalcProfit(Acc[j], Reg[k]) // Eq. (6.3)
10.     if profit > max_profit then
11.       max_profit := profit
12.       selected_reg := k
13.     end if
14.   end for
15.   Place accelerator j into region selected_reg
16.   occupied[selected_reg] := 1
17. end for
```

constantly stressed by one accelerator without stress redistribution. This happens if an accelerator delivers high speedup and is frequently required by the application. As a solution, the runtime accelerator placement forces that region to be reconfigured after a user-defined time period. This time period should not be too short to prevent increased reconfiguration overhead, while also not too long to avoid stress accumulation. For instance, a time period of 100 million cycles (1 s at 100 MHz) is short enough to avoid aging accumulation and the induced application performance degradation is only 0.21%.

6.3.3 Intermediate Results

Figure 6.5 shows that dynamic stress is uniformly distributed over all reconfigurable regions after employing the runtime accelerator placement, compared to the stress-unaware placement in Fig. 6.1. The maximal transistor toggle rate is reduced by more than 73% from 18.8 to 5.0 million toggles/s. However, when high stress CLBs of different accelerators *overlap* at the same relative (x, y) location, the runtime accelerator placement cannot achieve intra-region stress distribution, as noticeable in the upper-middle part of all reconfigurable regions in Fig. 6.5.

6.4 Synthesis Time Logic Placement

STRAP addresses this problem by applying placement constraints at *synthesis time* to diversify the CLB usage among different accelerators, which reduces the overlapping of high stress CLBs. To minimize the timing impact on accelerators, the mapping of logic functions to CLBs is left to the vendor place-and-route algorithm.

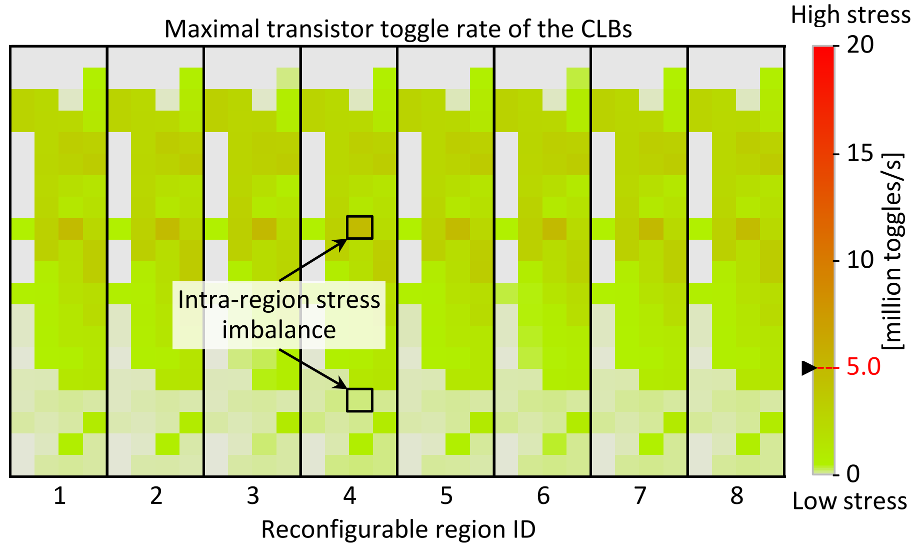

Figure 6.5: Transistor stress distribution using stress-aware runtime accelerator placement

Instead, it is only constrained *which* CLBs shall be used to place-and-route an accelerator, without additional constraints on logic mapping or routing.

6.4.1 Placement Algorithm

The logic placement algorithm (Alg. 4) diversifies the high stress CLBs of different accelerators to different (x,y) CLB locations in the reconfigurable regions. First, unconstrained configurations of all accelerators are generated (Lines 1 to 5). For each accelerator configuration the CLB stress is estimated (see Section 6.2), and the maximal achievable frequency is extracted from the place-and-route log files (Lines 3 and 4). The generated initial configurations are then sorted in ascending order of their maximal achievable frequencies (Line 6). The reconfigurable fabric typically runs at the frequency of the slowest accelerator f_{min}. In order to minimize the impact on system performance, it is placed and routed without stress-diversifying placement constraints. Its CLB stress distribution is taken as the initial reference distribution (Line 7). As long as the proposed logic placement does not reduce the frequency of an accelerator below f_{min}, there is no performance impact/penalty for the whole system. During the generation of other accelerator configurations, $\mathbf{R}$ keeps track of the sum of the stress distribution of all $j-1$ previously generated accelerators, i.e. $\mathbf{R} = \sum_{i=1}^{j-1} \mathbf{s_i}$.

The remaining accelerators will be placed-and-routed again in ascending order of their maximal frequencies (Lines 8 to 23). To avoid that high stress CLBs of the currently placed accelerator `Acc[j]` overlap with those in previously placed accelerators `Acc[1]`,...,`Acc[j-1]`, we prohibit the placement to specific CLB locations for `Acc[j]` (Lines 9 to 17), identified by their (x,y) coordinates, if the following

Algorithm 4 Stress-diversifying logic placement

```
Input: List of accelerators Acc.
 1. for j := 1 to len(Acc) do
 2.    Place-and-route Acc[j] without any placement constraints
 3.    s_j := get_stress(Acc[j])
 4.    Acc[j].max_freq := get_max_freq(Acc[j])
 5. end for
 6. Acc := sort_ascending(Acc, key=max_freq)
 7. R := s_1
 8. for j := 2 to len(Acc) do
 9.    prohibit_xy := ∅
10.    for x := 1 to Acc[j].n_cols do
11.       for y := 1 to Acc[j].n_rows do
12.          if Condition Equation (6.6) is satisfied for (x,y) then
13.             prohibit_xy.add((x,y))
14.          end if
15.       end for
16.    end for
17.    Place-and-route Acc[j] with prohibited CLB locations
       listed in prohibit_xy
18.    if Place-and-route failed then
19.       prohibit_xy.remove(argmin_{xy∈prohibit_xy}[R̂+ŝ_j]_xy)
20.       goto Line 17
21.    end if
22.    R := R + get_stress(Acc[j])
23. end for
```

condition is satisfied:

$$\left[\hat{\mathbf{R}}\right]_{xy} > \frac{1}{L_j}\sum_{uv}[\hat{\mathbf{s}}_\mathbf{j}]_{uv}$$
$$\text{with } \hat{\mathbf{R}} = \frac{\mathbf{R}}{\max_{uv}\left[\mathbf{R}\right]_{uv}} \text{ and } \hat{\mathbf{s}}_\mathbf{j} = \frac{\mathbf{s_j}}{\max_{uv}[\mathbf{s_j}]_{uv}} \tag{6.6}$$

where L_j is the number of used CLBs by the currently place-and-routed accelerator `Acc[j]`. $\hat{\mathbf{R}}$ and $\hat{\mathbf{s}}_\mathbf{j}$ are normalized stress matrices of $\mathbf{R}$ and $\mathbf{s_j}$. In earlier iterations, the reference distribution is less even, which implies that few CLB locations in the reference distribution have much higher values than the others, and therefore it is less likely that the condition (Equation (6.6)) is satisfied. In turn, fewer locations are prohibited for placement in earlier iterations, which implies less timing impact on slower accelerators. If place-and-route fails due to too many prohibited CLB locations, the locations xy where the stress overlapping $[\hat{\mathbf{R}} + \hat{\mathbf{s}}_\mathbf{j}]_{xy}$ is lowest are removed from `prohibit_xy` (Line 19), and place-and-route is re-executed with the relaxed constraints.

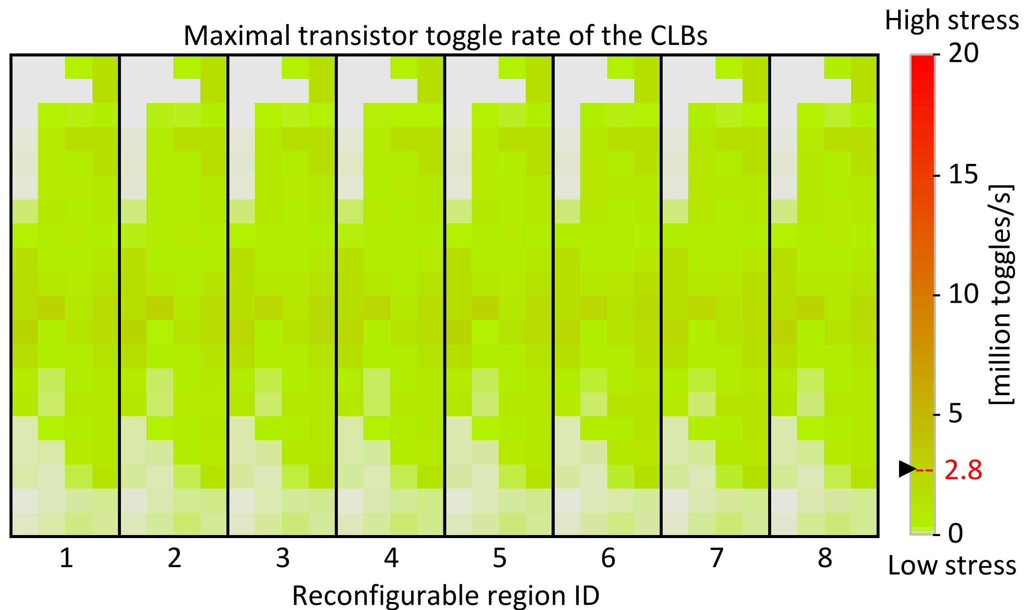

Figure 6.6: Transistor stress distribution using both stress-aware runtime accelerator placement and synthesis time stress diversification

6.4.2 Stress Distribution Results

With synthesis time stress diversification, high stress CLBs from different accelerators are placed to different CLB locations, and thus better intra-region stress distribution can be achieved during runtime placement. As shown in Fig. 6.6, after applying both stress-aware runtime placement and synthesis time stress diversification for dynamic stress, the maximal transistor toggle rate is further reduced by additional 44% from 5.0 to 2.8 million toggles/s.

6.5 Extended Runtime Accelerator Placement with Module Diversification

The module diversification method (see Chapter 5) generates a set of configurations for each accelerator that are diversified in terms of CLB usage. This not only allows to tolerate any single-CLB fault in a region but can also improves the stress distribution with the extra CLB diversity.

When faults are detected in the reconfigurable fabric, the placement freedom of accelerators is reduced. The *placement freedom* of an accelerator corresponds to the number of regions for which the accelerator has at least one diversified configuration that can be placed into that region (i.e. that tolerates the permanent faults in that region). Such a region is called a *compatible region*. If the available regions (i.e. those into which no accelerators are placed by the placement algorithm so far) have rather many permanent faults, it can happen that no configuration of the accelerator can be placed into any of them. If an accelerator cannot be placed, then its hardware functionality has to be emulated in software on the processor pipeline. This actually reduces the stress for the regions, as they are not used to execute the

accelerator, however, it comes at the cost of significantly degraded performance (i.e. less acceleration for that kernel).

To avoid such situations, the runtime placement algorithm (Alg. 3) tries to place the accelerators one after the other in ascending order of their number of compatible regions. If it still comes to the situation that some accelerator cannot be placed into the available regions, then the algorithm re-evaluates some of its previous placement decisions (note that the actual reconfigurations are just started after all placements are finally decided). It tries whether it can *swap* one of the already placed accelerators into one of the still available regions such that accelerator can be placed into the region that became free due to swapping.

When calculating the placement profits (Line 9 in Alg. 3), the algorithm also iterates through all diversified configurations to find out which configuration of the accelerator produces the highest placement profits. Permanent faults also reduces the freedom of selecting diversified configurations for stress distribution as some configurations may require the faulty CLBs.

6.6 Experimental Evaluation

In the evaluation platform (see Section 3.5), each region consists of 4×20 CLBs with eight 6-input LUTs per CLB. STRAP performs optimizations on CLB granularity. To evaluate the actual stress for each transistor, the transistor-level model of LUTs using NMOS pass transistors for multiplexers is used (see Section 2.1.3). To evaluate the threshold voltage shift due to stress, state-of-the-art aging models are employed (detailed equations and used parameters are given in Section 2.4.3).

The resource usage of each accelerator within one region for the H.264 application ranges from 8.8% to 66.3%. The architectural simulator is used to evaluate the STRAP method for systems that differ in the number of reconfigurable regions and runtime strategies, and to compare it with related work. Algorithm 3 is integrated into the simulator and Alg. 4 is implemented as a script that generates the placement constraints and automatically calls the Xilinx place-and-route tools.

6.6.1 Evaluation Flow

The experimental evaluation flow is shown in Fig. 6.7. The placed-and-routed accelerators are fed to Xilinx XPower analyzer to obtain the signal activities and power consumption of logic elements and nets. The power consumption is then aggregated to CLB granularity by summing up the power consumed by LUTs and their fan-in nets in one CLB. The leakage power of a region is proportional to its size. Architectural simulation produces the accelerator execution trace, i.e. the complete execution and idle history of each accelerator in each region. Together with the power profile of each accelerator, we obtain the power trace of each CLB. The power trace and the fabric floorplan of the FPGA (based on a die image acquired from chipworks.com)

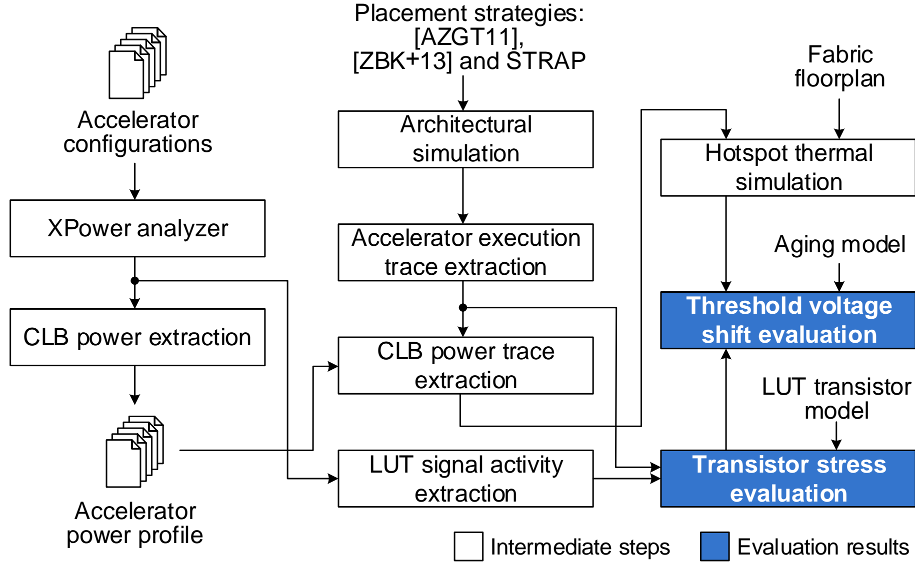

Figure 6.7: Experimental flow to evaluate the transistor stress and threshold voltage shift

is then fed into Hotspot[1] [HGV$^+$06] to obtain the temperature trace of each CLB, which will be used to evaluate the threshold voltage shift. The accelerator execution trace and the LUT signal activities of each accelerator are combined to calculate the LUT signal activities for the regions. This is then used to evaluate the stress of individual transistors by using the LUT transistor model.

The number of regions is varied from 5 to 12 and separate evaluation of the proposed method is performed for dynamic and static stress mitigation, since STRAP optimizes either for dynamic or for static stress. The baseline system does not use any stress distribution method. For comparison, two state-of-the-art stress distribution methods [AZGT11, ZBK$^+$13] was implemented. Zhang *et al.* [ZBK$^+$13] use three different configurations for each accelerator and switch between them to migrate stress, whereas Angermeier *et al.* [AZGT11] consider the peak stress of regions to place an accelerator. As proposed for STRAP, [AZGT11, ZBK$^+$13] was extended to replace an accelerator if its reconfigurable region has not been reconfigured for 100 million cycles (see Section 6.3.2). This improvement reduces the peak stress of [AZGT11, ZBK$^+$13] and thus makes the comparison with state-of-the-art more competitive. Regarding temperature variation, a conservative comparison is performed. To calculate the threshold voltage shift for [AZGT11, ZBK$^+$13], the lowest temperature that was observed for any CLB at any time in the obtained temperature trace is used as the constant temperature for all CLBs, while the highest observed temperature is applied for STRAP. Thus, the threshold voltage shift reported for [AZGT11, ZBK$^+$13] is a lower limit, whereas the one for STRAP is a conservative upper limit.

[1] smallest possible heat spreader and heat sink with 10 µm thickness, ambient temperature 50℃

Table 6.1: Change in maximum frequency of accelerators

Accelerator	Original [MHz]	STRAP [MHz]	Worstcase Δ [%]
Clip3	133	122–130	8.2
CollapseAdd	158	158–158	0.0
LF_BS4	121	115–120	5.0
LF_Cond	146	132–140	9.6
PointFilter	89	89	0.0
QuadSub	257	232–257	9.7
SADrow_4	100	96–96	4.0
SAV	139	120–138	13.7
Transform	167	145–166	13.2
System freq.	89	89	0.0

6.6.2 Timing Overhead

STRAP's stress-diversifying logic placement at synthesis time may affect the accelerator frequency. The timing impact of the placement constraints is shown in Table 6.1. The place-and-route tool is given a target frequency of 250 MHz as timing constraint to obtain the maximum operating frequency of each accelerator. On average, the maximum accelerator frequency decreases by 7%. Since accelerators with longer critical path (lower maximum frequency) are imposed with fewer constraints (see Section 6.4.1), their maximum frequencies are less affected. The maximum *system* frequency is however limited by the accelerator with the longest critical path, i.e. PointFilter, which runs at $f_{min} = 89$ MHz (see Section 6.4.1). Therefore, STRAP has no negative timing impact on the whole system.

6.6.3 Stress Reduction and MTTF Improvement

Figure 6.8 shows the maximal (lighter color) and average (darker color; arithmetic mean) dynamic transistor stress, measured in million toggles/s, in the whole reconfigurable fabric for systems with different number of regions. Similarly, Figure 6.9 shows the static transistor stress measured in normalized stress time (stress duty cycle), i.e. the fraction of operation time the transistor is under static stress. The figures show that all methods reduce the average stress compared to the baseline because they all distribute the stress to more transistors. While the reduction of the average stress is similar for all three methods, the reduction of the maximal stress (i.e. the critical part for system mean time to failure/MTTF) differs significantly and requires both runtime and synthesis time optimization. The reason is that Angermeier *et al.* [AZGT11] perform only runtime inter-region stress distribution, while Zhang *et al.* [ZBK+13] perform only synthesis time intra-region stress distribution for individual accelerators. In contrast, STRAP performs cross-layer stress-aware placement at runtime and synthesis time, which leads to the highest reduction of maximal stress in all evaluated cases. The reduction of the maximum stress by STRAP in Fig. 6.8 and 6.9 is up to 64% and 35% higher than the closest

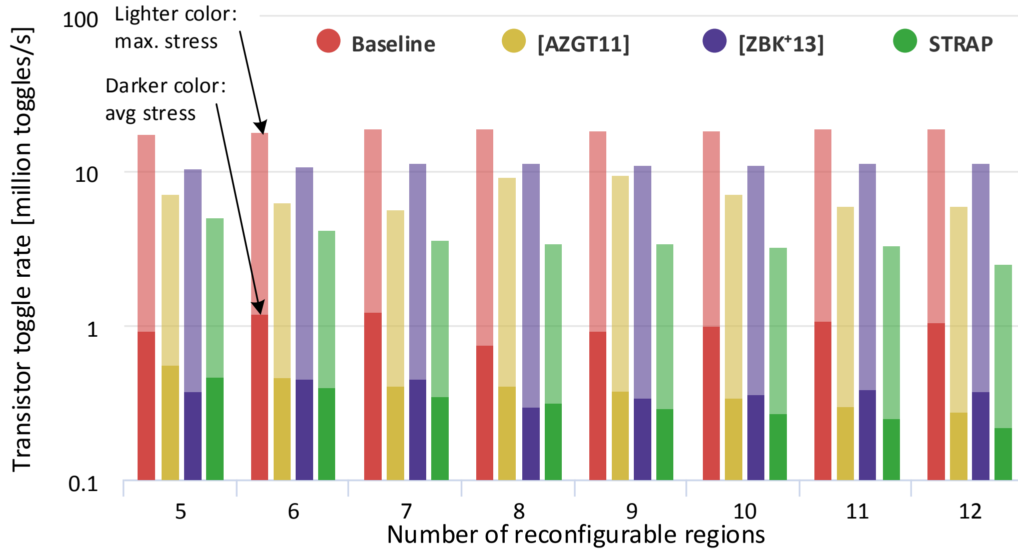

Figure 6.8: Comparison to related work for dynamic stress in systems with different number of reconfigurable regions

competitors w.r.t. dynamic and static stress, respectively. Table 6.2 summarizes the stress reduction shown in Fig. 6.8 and 6.9.

Figure 6.10 analyzes the detailed per-transistor stress (static and dynamic) of a system with 8 reconfigurable regions. It compares STRAP that optimizes for static stress or dynamic stress against the baseline. Each point represents the stress value of one transistor in the reconfigurable fabric. Points closer to the lower-left corner denote less dynamic and less static stress. The horizontal and vertical lines represent the upper boundary for dynamic stress and static stress in all three cases. Although during optimization only one type of stress is considered, actually both types of stress are reduced simultaneously. With STRAP targeting the static stress distribution, a

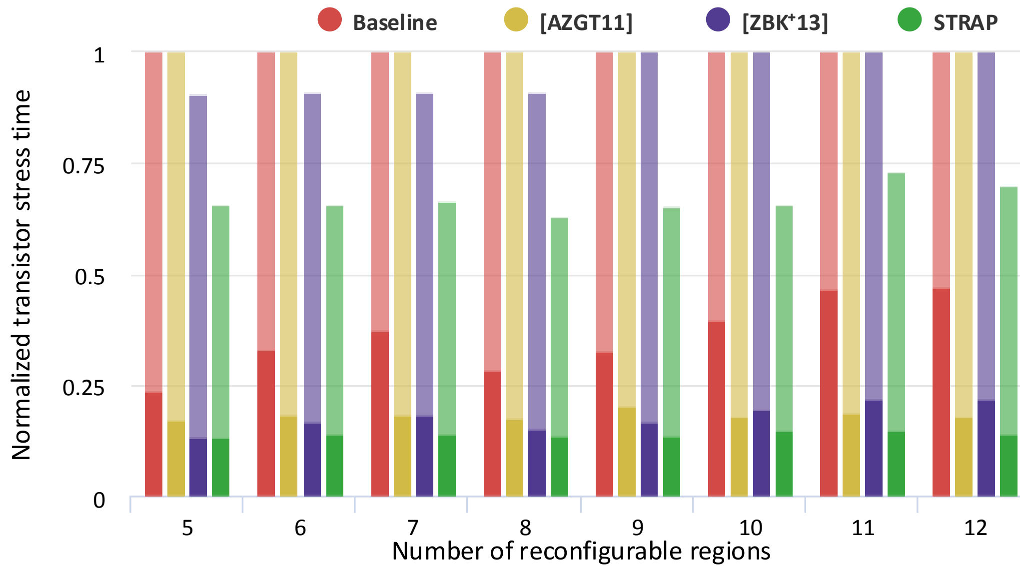

Figure 6.9: Comparison to related work for static stress in systems with different number of reconfigurable regions

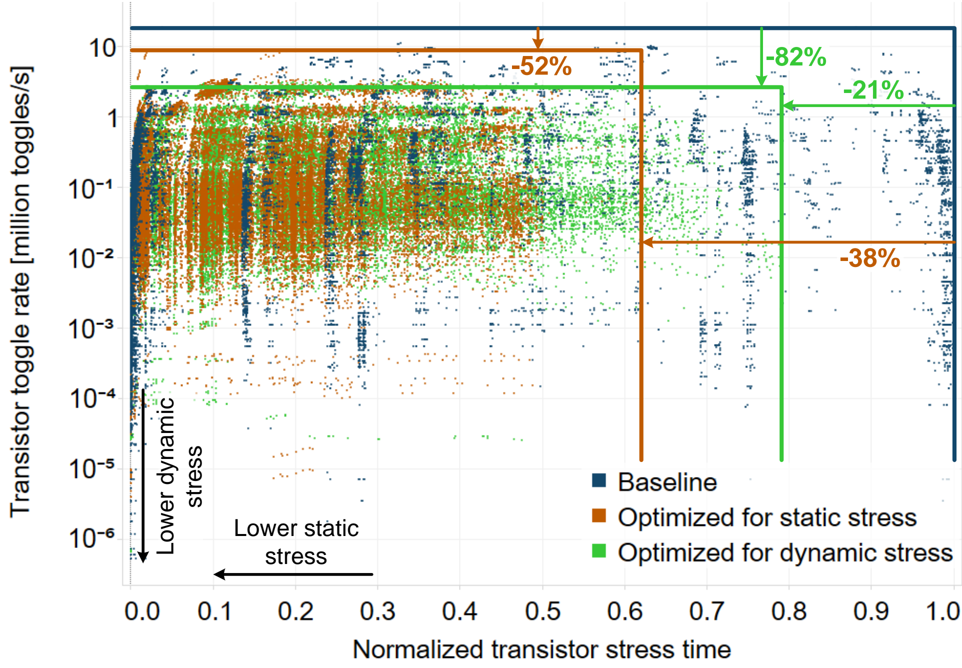

Figure 6.10: Transistor stress for different STRAP optimization goals

reduction of 52% in dynamic and 38% in static stress is observed. When targeting dynamic stress, STRAP delivers 82% reduction in dynamic stress and 21% reduction in static stress. The reason behind the reduction of both stress types is that STRAP implicitly distributes the transistor usage as well, which reduces the individual static and dynamic transistor stress.

The MTTF improvement due to the stress reduction is calculated by assuming that a device fails when ΔV_{th} of any transistor exceeds 50% of its original value (V_{th0}). The MTTF improvement due to dynamic and static stress reduction is shown in the last two columns in Table 6.2. With the STRAP method, the MTTF improvement relative to the baseline is 413% and 13% in average for HCI and BTI aging, respectively. Relative to the closest competitors, STRAP achieves up to 177% and 14% MTTF improvement w.r.t. HCI and BTI aging, respectively.

Table 6.2: Reduction of avg./max. stress and MTTF increase of STRAP and state-of-the-art [AZGT11, ZBK+13] compared to the baseline; averaged over all numbers of reconfigurable regions

Strategy	Reduction of avg. stress[%]		Reduction of max. stress[%]		MTTF improvement[%]	
	dyn.	stat.	dyn.	stat.	HCI	BTI
[AZGT11]	60.6	47.4	61.2	0.02	157.7	0.0
[ZBK$^+$13]	62.6	49.6	39.9	4.5	66.4	2.3
STRAP	**67.9**	**59.6**	**80.5**	**33.1**	**413.0**	**13.4**

7 Reliability Guarantee with Adaptive Modular Redundancy

Harsh environmental conditions (radiation, temperature, power noise) may cause transient errors and failures which are not acceptable in safety- and mission-critical applications (e.g. automotive, industrial, medical or aviation), where stringent reliability requirements such as ASIL [ISO11] have to be met under different environmental and operating conditions and changing error rates in the system. The reliable acceleration in the reconfigurable fabric in SRAM-based FPGAs are threatened by soft errors resulted from SEUs in the configuration memory and functionally used memory (e.g. block RAMs and flipflops), which may alter the functionality of hardware accelerators and lead to wrong results. To ensure reliable computation, accelerators must be protected by fault tolerance methods such as modular redundancy (e.g. duplication with comparison (DWC), triple modular redundancy (TMR)), or information redundancy (self-checking circuits, ECC of memory). These fault tolerance methods incur high error detection cost which must run concurrently to regular system operation, in terms of hardware resources, performance, and energy. Error *correction* by re-execution after an error has been detected typically incurs only a small performance cost and happens rarely.

Due to changing soft error rates (see Section 2.3.4), application requirements (data dependencies) and system states (available/used resources), it is not possible to *statically* determine appropriate error detection methods for a given target reliability at minimal cost. A static optimization is pessimistic since it must consider the worst case and when the error rate is low, the system is over-protected at additional hardware or performance cost. A static selection of fault tolerance methods during design time cannot adapt to changing soft error rates during runtime and thereby hinders trading off performance and reliability.

In contrast to the static and therefore pessimistic selection of fault-tolerance methods, this thesis presents a method of adaptive modular redundancy for reconfigurable architectures. It guarantees an application-specified minimum level of reliability of the accelerated computation at minimal performance cost. This is achieved by use of monitoring information to dynamically choose between different redundancy modes so that the error-detection overhead is minimized. At runtime, the soft error rate is monitored and the reliability of future computations is estimated. Based on statically or dynamically given target reliability constraints, runtime reliability management is performed. Based on the reliability estimation, the selection of accelerators and the application of optimal fault tolerance methods are performed at runtime. This allows for fast adaptation to changing reliability threats and guarantees the given reliability constraints while maximizing the performance.

7.1 Overview of Adaptive Modular Redundancy

The accelerated computations in the reconfigurable fabric are described by data-flow graphs where each node represents an accelerator and the edges represents the data dependencies between accelerators. These data-flow graphs correspond to accelerated functions (AFs, see Section 3.1). These AFs can be of different size, from a complex function down to a short sequence of instructions. Figure 7.1a) shows an example AF that consists of three different accelerator types (A_1, A_2, A_3) and requires at least three different reconfigurable regions (one for each accelerator type) to be implemented. The example in Fig. 7.1a) uses exactly three regions and thus the two instances of A_3 in the DFG have to be executed in different *control steps*.

An AF may have multiple hardware implementation variants that trade-off performance and resource usage (i.e. number of regions). The two variants shown in Fig. 7.1a) and Fig. 7.1b) differ in latency and resource usage per step. Variant a) uses only one instance A_3 per step and finishes in 3 steps while variant b) uses two instances of A_3 in parallel (demanding two separate regions) in step 1 and thus finishes in 2 steps. Variants that use more accelerators exploit more parallelism and can achieve higher performance. It is also possible to provide a partially or completely fault tolerant variant, e.g. by triplicating A_3, as shown in Fig. 7.1c). This variant has the same schedule as variant a) but uses A_3 in TMR mode to increase reliability at higher resource usage. Variant a) is called the *base variant* of the reliable variant c), which is derived from variant a) by duplicating or triplicating a subset of its accelerators.

The term *reliability* denotes the probability of error-free operation for a specified period of time [ALRL04]. The reliability requirement of an application specifies the upper bound of error probability of individual AFs or the whole application. The reliability of a system depends on the reliability of its components. It is assumed that the processor core is a reliable computing base and an AF is error free during its execution all of its component accelerators are not affected by soft errors (see Section 3.3). An AF that is executed as a software routine on the reliable computing base is considered as reliable. The reliability of an AF that is executed on the

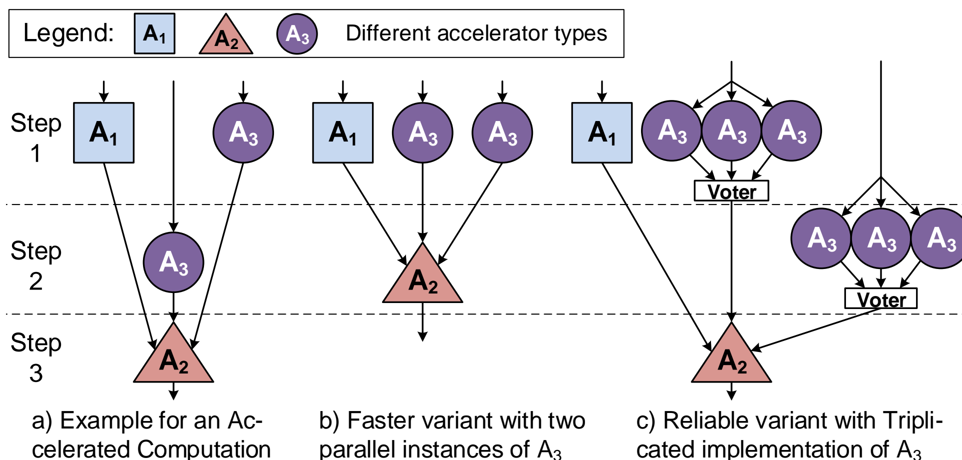

Figure 7.1: Different hardware implementation variants of an Accelerated Function (AF)

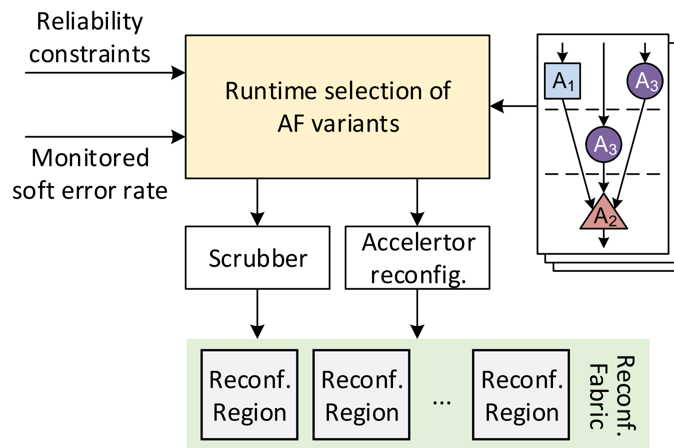

Figure 7.2: Overview of proposed adaptive modular redundancy

reconfigurable fabric depends on the following factors:

Current error rate is determined by the environment.

System state corresponds to the reliability history of the regions, i.e. the time since a region was last known to be error free because it was tested, reconfigured, or scrubbed (i.e. reconfiguring it with the configuration data of the accelerator that was already configured or reading back the configuration data and correcting possible errors by an error correction code).

Hardware usage depends on the accelerators that implement the AF. The configuration information of an accelerator is stored in the SRAM configuration memory of the FPGA, which is susceptible to soft errors. The critical bits of an accelerator are those configuration bits that define its functionality (see Section 2.4.4). Different accelerators exhibit different susceptibility to soft errors in their configuration memory depending on the number of critical bits.

The variety in soft error vulnerability of accelerators also extends to the hardware implementations of AFs: different AFs and various implementation variants of an AF differ in their soft error vulnerability. This variety can be exploited by the runtime system of the reconfigurable architecture. In order to guarantee a given target reliability while optimizing the performance, the runtime system (see Fig. 7.2) needs to address the following challenges:

1. If target reliability is specified for an AF, whenever the AF shall execute, guarantee that it meets the target reliability for the current error rate and system state. If the target reliability cannot be satisfied at the moment due to pending reconfigurations of redundant accelerators or limited hardware resources, then the AF needs to be executed on the reliable computing base.

2. If target reliability is specified for the application itself, decompose it into the target reliability of individual AFs such that appropriate implementation variants can be selected accordingly.

3. For all AFs to be executed by the application, decide which implementation

variant shall be reconfigured and find a good trade-off that ensures the target reliability while maximizing performance for the monitored error rate and system state. The runtime system chooses reliability measures accordingly to give more resources to those more vulnerable accelerators while maximize the performance of less vulnerable accelerators.

4. Decide for each region when to perform scrubbing. After scrubbing, an accelerator is known to be error-free. As no other region can be reconfigured until scrubbing completes, scrubbing also reduces performance.

7.2 Reliability of Accelerated Functions

The reliability of an accelerated function depends on the soft error rate, the type, structure and size of the used hardware accelerators, and the *resident time* the accelerators have been instantiated without errors in the reconfigurable fabric, i.e. the time elapsed since the last reconfiguration or scrubbing event of the region. As already established in Section 2.4.4, the reliability $R(A,t)$ of an accelerator A with number of critical bits n and resident time t is

$$R(A,t) = \prod^{n} e^{-\lambda t} = e^{-\lambda n t}, \tag{7.1}$$

which decreases with increasing resident time.

For accelerators without any fault-tolerance methods, the reliability of an accelerated function AF (probability that it produces the correct result) is

$$R(AF,t,\tau) = \prod_{i}^{A_i \in AF} e^{-n_i \lambda (t_i + \tau_i)} = e^{-\lambda \sum_i n_i (t_i + \tau_i)}, \tag{7.2}$$

where t_i is the resident time of accelerator A_i until the accelerated function starts to execute, and τ_i denotes the time period until accelerator A_i finishes all its executions. Since $\tau_i \ll t_i$, τ is ignored in the following calculation. It is assumed conservatively that an accelerator computes the correct results only if all its critical bits are correct, i.e. logic and data-dependent masking of errors are ignored here. Such error masking can be added to this computation by derating factors derived for instance from fault injection experiments. In a similar manner, it is assumed that an AF produces correct results only if all of its belonging accelerators compute correctly and an application operates correctly only when all its AFs are correct.

Frequent scrubbing improves the reliability of accelerators by checking and repairing errors in short periods. It is however limited by the bandwidth of the configuration port of the FPGA. When a reconfigurable fabric with N reconfigurable regions is periodically scrubbed, the minimum scrubbing period (i.e. the time between two scrubbing operations) of an accelerator is $N \cdot T_S$, where T_S denotes the time required to scrub one reconfigurable region. In other words, the correctness of the critical bits of an accelerator can only be checked in a period longer than or equal to $N \cdot T_S$. By scrubbing alone, the reliability of an accelerator A_i in a reconfigurable fabric

with N regions can be maintained at

$$R(A_i) \geq e^{-\lambda n_i N \cdot T_S} \tag{7.3}$$

Similarly, the reliability of an accelerated function AF_j (Eq. (7.2)) is maximized when all of its accelerators are scrubbed at the fastest period $N \cdot T_S$. In this case, it holds that $\forall A_i \in AF : t_i \leq N \cdot T_S$ and thus

$$R(AF_j) = e^{-\lambda \sum_i n_i t_i} \geq e^{-\lambda (\sum_i n_i) N \cdot T_S} \tag{7.4}$$

where the right-most term expresses the lower bound of the reliability of the accelerated function AF_j.

Implementation variants of accelerators may include partially or completely protected accelerators based on duplication or triplication (see Section 7.1). For accelerators in TMR mode with hardened voter, the probability that it delivers the correct output is the probability that at most one of the three replicated accelerators is affected by soft errors in their critical bits, which is

$$\begin{aligned} R(A_i^{TMR}) = & (1 - R(A_a))\, R(A_b) R(A_c) + \\ & (1 - R(A_b))\, R(A_a) R(A_c) + \\ & (1 - R(A_c))\, R(A_a) R(A_b) + \\ & R(A_a) R(A_b) R(A_c) \\ = & \, e^{-n\lambda(t_a + t_b)} + e^{-n\lambda(t_a + t_c)} + e^{-n\lambda(t_b + t_c)} \\ & - 2e^{-n\lambda(t_a + t_b + t_c)}, \end{aligned} \tag{7.5}$$

where $R(A_a)$, $R(A_b)$ and $R(A_c)$ denote the reliability of the three replicated accelerators. t_a, t_b and t_c denote the resident times of the three replicated accelerators. For accelerators in DWC mode, the accelerated function is re-executed on the hardened processor if an error is detected. Thus the probability of correct results equals to the probability that at most one of the replicated accelerators is erroneous:

$$R(A_i^{DWC}) = e^{-n\lambda t_a} + e^{-n\lambda t_b} - e^{-n\lambda(t_a + t_b)}. \tag{7.6}$$

7.3 Reliability Guarantee of Accelerated Functions

When a *reliability constraint* is specified for each AF, it requires that the error probability of every execution of the accelerated function AF_j, i.e. $1 - R(AF_j, t_j)$, is less than or equal to a statically or dynamically given threshold, usually written in powers of ten as 10^{-r_j}:

$$\forall j \ : \ 1 - R(AF_j, t_j) \leq 10^{-r_j}. \tag{7.7}$$

For instance, when $r_j = 5$, the error probability of each execution of AF_j must be less than 10^{-5}. In Eq. (7.2) and (7.7), the values of n_i and τ_i are derived from the

AF implementations at design time. λ, t_i, and the target reliability r_j are variables whose values may dynamically change during runtime.

7.3.1 Maximum Resident Time

To satisfy the reliability constraint in Eq. (7.7), the runtime system must ensure that unprotected accelerators used in the next execution of AF_j are still sufficiently reliable. This requires that the resident times of non-redundant accelerators in AF_j satisfy the inequality: $\prod_i^{A_i \in AF_j} e^{-n_i \lambda t_i} \geq 1 - 10^{-r_j}$. After applying the logarithm on both sides, we obtain

$$\sum_i^{A_i \in AF_k} n_i t_i \leq -\frac{1}{\lambda} \log\left(1 - 10^{-r_k}\right). \tag{7.8}$$

By making t_i small enough, e.g. by scrubbing accelerators more frequently, the reliability constraint can be fulfilled. However, there are many combinations of resident times t_i which satisfy Equation (7.8). To find the optimal combination which maximizes every t_i so that the scrubbing overhead is minimized, the runtime system has to solve a max-min problem involving $\|AF_j\| + 2^{\|AF_j\|}$ constraints, where $\|AF_j\|$ is the number of accelerators required by AF_j. This is too complex for the runtime system and would decrease its responsiveness to other important tasks.

To simplify the problem, let t_{max} denote the maximum resident time of all accelerators required by an accelerated function AF_j, i.e. $t_{max} = \max_i\{t_i\}$. Then,

$$\sum_i^{A_i \in AF_j} n_i t_i \leq \sum_i^{A_i \in AF_j} n_i t_{max}, \tag{7.9}$$

and Equation (7.8) is automatically satisfied when

$$t_{max} \leq \underbrace{\frac{1}{\sum_i^{A_i \in AF_k} n_i} \left(-\frac{1}{\lambda} \log\left(1 - 10^{-r_k}\right)\right)}_{T_j^{up}}. \tag{7.10}$$

We denote the right-hand side of Equation (7.10) as T_j^{up}, the upper bound of t_{max} for AF_j. With the above *tightening*, the runtime system only needs to schedule scrubbing for non-redundant accelerators such that t_{max} satisfies Equation (7.10), which is stricter than required.

For an AF_j consisting of only triplicated accelerators and applying tightening by $t_{max} = \max\{t_a, t_b, t_c\}$, the reliability constraint $1 - R(A_i^{TMR}) \leq 10^{-r_j}$ becomes $3e^{-2n\lambda t_{max}} - 2e^{-3n\lambda t_{max}} \geq 1 - 10^{-r_j}$. This can be easily solved by substitution to obtain the bound for t_{max}. But it becomes difficult when we compute t_{max} for partially fault tolerant variants as shown in Fig. 7.1c). However, we can always find a suitable q (usually < 1) such that

$$3e^{-2n\lambda t_{max}} - 2e^{-3n\lambda t_{max}} \geq e^{-qn\lambda t_{max}} \tag{7.11}$$

holds for all t_{max} where $e^{-n\lambda t_{max}}$, the reliability of a non-redundant accelerator, is assumed to be larger than a very conservative value such as 0.99. Therefore the reliability constraint for an arbitrary accelerated function combining non-redundant and triplicated accelerators is tightened to

$$\prod_{i}^{A_i \in AF_j,\text{non-red.}} e^{-n_i \lambda t_{max}} \prod_{i}^{A_i \in AF_j,\text{TMR}} e^{-q n_i \lambda t_{max}} \geq 1 - 10^{-r_j}, \tag{7.12}$$

where t_{max} is the maximum resident time of all accelerators. After taking the logarithm on both sides, we obtain

$$t_{max} \leq \underbrace{\frac{1}{\sum_i^{\text{non-red.}} n_i + \sum_i^{\text{TMR}} q n_i} \left(-\frac{1}{\lambda} \log \left(1 - 10^{-r_j} \right) \right)}_{T_j^{up}}, \tag{7.13}$$

In a similar way, tightening is also applied to accelerated functions with accelerators in duplicated mode.

7.3.2 Acceleration Variants Selection

When the application requests to execute accelerated functions in reconfigurable fabric, the runtime system has to select from a large set of acceleration variants to configure, which have distinct performance, reliability and resource usage characteristics. The variants of an AF consists of a common set of accelerators and the bitstream of these accelerators are stored in the memory for online reconfiguration. As an motivational example, Fig. 7.3 shows the selection space for a complex H.264 encoder application, in which nine AFs are implemented. Each data point in the

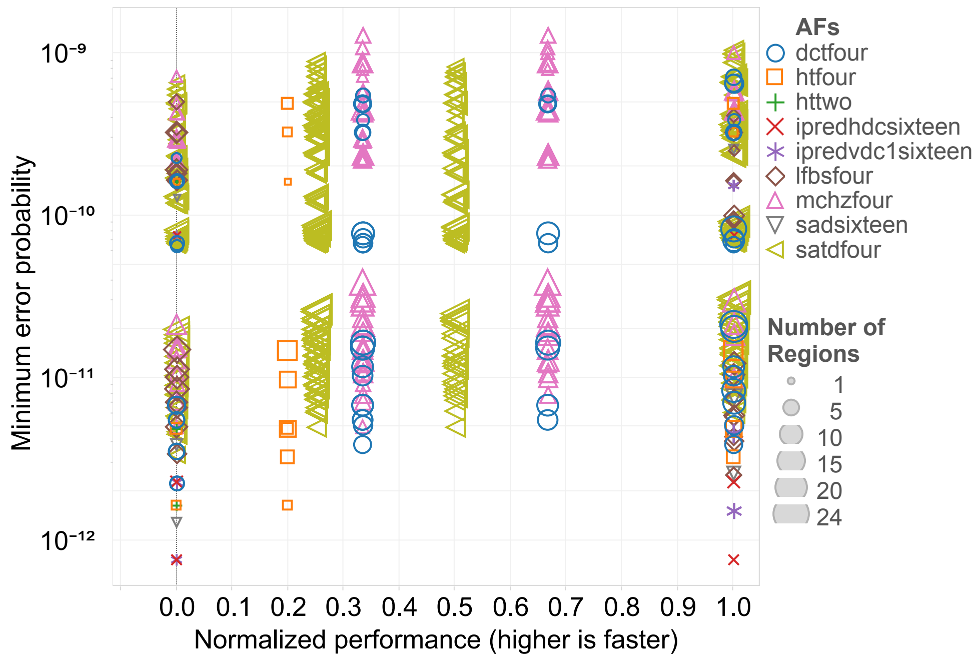

Figure 7.3: Variants selection space for an error rate of 10 errors $\text{Mb}^{-1}\text{month}^{-1}$.

figure denotes an acceleration variant of a specific AF (coded in color and shape) including partially and completely fault tolerant variants. Each variant is described by three metrics: minimum error probability (Y-axis), performance (X-axis) and number of regions (size of the data point). The *minimum* error probability of a variant is its error probability when t_{max} equals the minimum scrubbing period of the system. For the variants, the error probability differs by more than three orders of magnitude. The performance shows the speedup of each variant compared to software execution, normalized for each AF. The absolute speedup ranges from 6.3 to 70.2×. The displayed performance is normalized for every AF. Zero means least speedup and one means most speedup.

The runtime system selects the accelerator variants upon an application request. Thus, the selection must complete in a short time period despite of the large selection space. This makes it computationally inviable to obtain an exact solution to the underlying NP-complete Knapsack problem, where—in addition to satisfying the reliability constraint—the number of accelerators to implement the chosen AF variant must not exceed the number of regions (capacity of the Knapsack) and the performance of the AFs shall be maximized (optimization).

Algorithm 5 shows the proposed greedy algorithm that selects the appropriate variants for requested accelerated functions such that the target reliability and resource constraints are satisfied and the performance of the whole application is maximized. Its worst-case complexity is $O(n^2)$, where n is the number of variants to be selected.

The variant selection is guided by a *performance score* which ensures that the selection is resource efficient and the performance of the whole application increases: Line 1 collects those acceleration variants v for the requested accelerated functions ($v.\text{fct} \in \mathcal{F}$) into set $\mathcal{C}$ which are able to meet the reliability constraint, i.e. the upper bound of t_{max} for the variant is greater or equal to the minimum scrubbing period of the system. As discussed in Section 7.3.1, the upper bound of t_{max} depends on the used resources and applied fault tolerance method of the variant. Line 2 keeps the smallest derived variant per base variant (see Section 7.1) in $\mathcal{C}$, i.e. the variant using the fewest regions ($\|v\|$ denotes the number of regions required by v). The loop from Line 5 to Line 30 iteratively selects the variant with the highest *performance score* among others in $\mathcal{C}$, and which still fits into the available regions. Line 16 calculates the performance score of a variant as the weighted speedup gain compared to a previously selected variant for the same accelerated function: The weight is the history execution frequency f_{EX} of the accelerated function divided by the number of regions required by the variant. If there is no previously selected variant, the speedup gain is calculated relative to the software execution (Line 14). The variant v_{best} with highest score is added to the result set $\mathcal{R}$ if there is no faster variant (fewer execution cycles) of the same function already in $\mathcal{R}$. The main loop continues until $\mathcal{C}$ is empty, or no variant with the targeted reliability fits into the remaining regions.

Before the actual execution of an accelerated function, the runtime system checks if the hardware variant selected by Alg. 5 is already configured, and if it still satisfies the reliability constraint for the current error rate (both might have changed since

Algorithm 5 Acceleration variants selection

Input: The set of accelerated functions to be executed $\mathcal{F}$.
Output: Selected variants for each accelerated function in $\mathcal{F}$.

1. $\mathcal{C} := \{\text{all variants } v \text{ for } v.\text{fct} \in \mathcal{F} \mid T^{up}(v) \geq N \cdot T_S\}$
2. $\mathcal{C} := \{v \mid v \in \mathcal{C} \text{ and } \forall u \in C, u.\text{base} = v.\text{base} : \|u\| > \|v\|\}$
3. N := NumberOfRegions // Total number of reconfigurable regions
4. $\mathcal{R} := \emptyset$; $\mathcal{A} := \emptyset$ // Result set and set of accelerators required by the selected variants
5. **while** $\mathcal{C} \neq \emptyset$ **do**
6. $\quad\mathcal{C} := \mathcal{C} \setminus \{v \mid v \in \mathcal{C}, \|v.acc \cup \mathcal{A}\| > N\}$
7. $\quad$**if** $\mathcal{C} = \emptyset$ **then**
8. $\quad\quad$break
9. $\quad$**end if**
10. $\quad v_{best}$:= NULL ; BestScore := $-\infty$
11. $\quad$**for all** $v \in \mathcal{C}$ **do**
12. $\quad\quad v_{sel}$:= fastest variant $w \in \mathcal{R}$ with $w.\text{fct} = v.\text{fct}$
13. $\quad\quad$**if** v_{sel}=NULL **then**
14. $\quad\quad\quad$Score := $f_{\text{EX}}(v.\text{fct}) \cdot (v.\text{sw_cycles-}v.\text{hw_cycles})/\|v\|$
15. $\quad\quad$**else**
16. $\quad\quad\quad$Score := $f_{\text{EX}}(v.\text{fct}) \cdot (v_{sel}.\text{hw_cycles-}v.\text{hw_cycles})/\|v\|$
17. $\quad\quad$**end if**
18. $\quad\quad$**if** Score > BestScore **then**
19. $\quad\quad\quad v_{best} := v$
20. $\quad\quad\quad$BestScore := Score
21. $\quad\quad$**end if**
22. $\quad$**end for**
23. $\quad v_{replace} := v \in \mathcal{R} \wedge v.\text{fct} = v_{best}.\text{fct}$; $\mathcal{C} := \mathcal{C} \setminus \{v_{best}\}$
24. $\quad$**if** $v_{replace}$ = NULL **then**
25. $\quad\quad \mathcal{R} := \mathcal{R} \cup \{v_{best}\}$; $\mathcal{A} := \mathcal{A} \cup v_{best}.acc$
26. $\quad$**else if** $v_{replace}$.hw_cycles > v_{best}.hw_cycles **then**
27. $\quad\quad \mathcal{R} := (\mathcal{R} \setminus \{v_{replace}\}) \cup \{v_{best}\}$
28. $\quad\quad \mathcal{A} := (\mathcal{A} \setminus v_{replace}.acc) \cup v_{best}.acc$
29. $\quad$**end if**
30. **end while**
31. **return** $\mathcal{R}$ // Selected variants to be configured

the last execution of Alg. 5). If that is not the case, the AF is executed in software by the hardened processor.

7.3.3 Non-uniform Accelerator Scrubbing

The scrubbing rate for each region is determined by the accelerator implemented in it. If the accelerator belongs to an accelerator variant which requires a short resident time to satisfy the reliability constraint, the region must be scrubbed more frequently. More precisely, if t_{max} of a variant has to satisfy Eq. (7.13), then all the regions it uses are scrubbed as soon as the resident time exceeds $(T_j^{up} - N \cdot T_S)$. In this way, t_{max} of every implemented variant is guaranteed to satisfy the tightened reliability constraint and the scrubbing overhead is minimized.

7.4 Reliability Guarantee of Applications

When the reliability constraint is specified for an whole application, instead of individual accelerated functions, that the error probability of the outputs from the application must be lower than a given bound, kernels in the application (see Section 3.1) bring another layer of complexity to the reliability-performance trade-off.

Consider the execution of an application consisting of two kernels targeting maximum performance, as shown in Fig. 7.4a). Kernel 1 requires a large amount of resources and finishes in short time (fewer loops or lighter computation), while kernel 2 has a lower resource utilization and needs more time to finish. When the application is imposed by a reliability constraint, e.g. that the error probability of the computed results must be small, accelerators in the kernels can be duplicated or triplicated to build redundancy. This is shown in Fig. 7.4b), where some accelerators in kernel 1 and all accelerators in kernel 2 are protected by DWC. However, due to the limited amount of available resources (marked by the dashed horizontal line), resources devoted to redundancy are not available for acceleration, which leads to longer execution time of kernel 1. Longer execution time implies a higher chance of being affected by soft-errors for protected and unprotected accelerators during the execution. The protection strategy in Fig. 7.4b) does not necessarily fulfill the low error probability required by the application. The resources spent for protection in kernel 1 exhibit a lower protection efficiency than those in kernel 2. In Section 7.4.3 it will be shown that the error probability of a kernel without redundancy can be expressed as $1 - e^{-k \times \text{amount of resource} \times \text{execution time}}$. Instead, if just kernel 2 is protected with the awareness of the intrinsic low error probability of kernel 1 due to its short execution time, the reliability constraint of the application may be met without any loss of performance.

Overall, targeting the reliability of a whole application, multiple factors need to be considered simultaneously: the kernel execution time, the implementation variants of accelerated functions, the vulnerability of accelerators and their impact on each other. This work presents a novel resource budgeting method to maximize the system performance under a given application reliability constraint. This is achieved by budgeting the *effective critical bits*, which is a metric that allows to capture all reliability impacting factors as one single value. Budgeting of effective critical bits is performed by the following three steps: 1) Transform the reliability constraint of the application (i.e. its allowed error probability) to the number of allowed effective

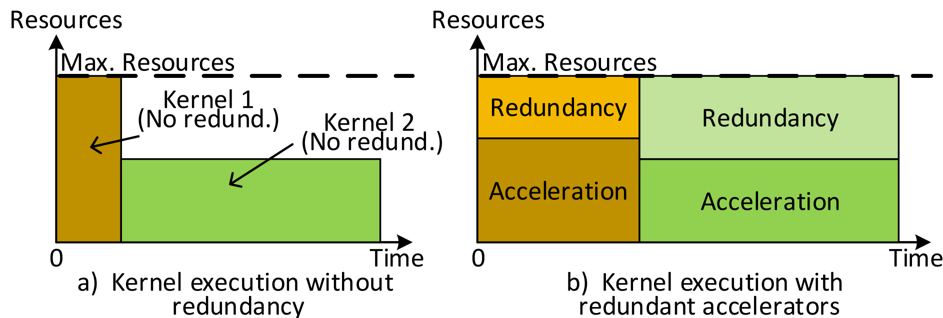

Figure 7.4: Execution of kernels with different degrees of redundancy

critical bits of the application; 2) Theses allowed effective critical bits are then assigned to the kernels based on their resource requirement and expected execution time; 3) Based on that, the runtime system selects the redundancy modes for the accelerated functions to maximize the performance within the budget.

7.4.1 Effective Critical Bits of Accelerators

By scrubbing with maximum frequency, the reliability of an accelerator pair in DWC, i.e. Eq. (7.6) with $t_a = t_b = N \cdot T_S$, can be maintained at

$$R(A^{DWC}) \geq 2e^{-\lambda n N \cdot T_S} - e^{-2\lambda n N \cdot T_S}. \tag{7.14}$$

By solving

$$2e^{-\lambda n N \cdot T_S} - e^{-2\lambda n N \cdot T_S} = e^{-\lambda \alpha n N \cdot T_S} \tag{7.15}$$

for α, we obtain

$$\alpha(n) = \log_u(2u - u^2) \text{ with } u = e^{-\lambda n N \cdot T_S} \text{ and} \tag{7.16}$$

$$R(A^{DWC}) \geq e^{-\lambda \alpha n N \cdot T_S}, \tag{7.17}$$

where α is a number that is always less than 1 for $0 < u < 1$. Equation (7.17) can be interpreted in a way that by introducing redundancy with DWC, the reliability of the accelerator pair can be maintained at a much higher level, as if the number of critical bits of the accelerator were reduced from n to αn. Here, αn is called the *effective critical bits* of the accelerator pair in DWC. By duplicating one of the accelerators for DWC, e.g. A_1, the lower bound of the reliability of an accelerated function AF_j is raised to

$$R(AF_j) \geq e^{-\lambda \left(\sum_{i \neq 1} n_i + \alpha(n_1) n_1\right) N \cdot T_S}. \tag{7.18}$$

By comparing Eq. (7.4) and Eq. (7.18), it can be observed that after introducing DWC for an accelerator with n_i critical bits, the total number of effective critical bits of AF_j is reduced by $(1 - \alpha(n_i))n_i$.

When three instances of accelerators with n critical bits each are paired to implement TMR, based on the similar derivation, the effective critical bits βn of the TMR pair can be obtained:

$$\beta(n) = \log_u(3u^2 - 2u^3) \text{ and } u = e^{-\lambda n N \cdot T_S}. \tag{7.19}$$

7.4.2 Reliability of Accelerated Kernels

The correct functionality of a kernel depends on the error-free execution of its AFs. It is assumed that a kernel delivers correct results only when all its AFs are executed

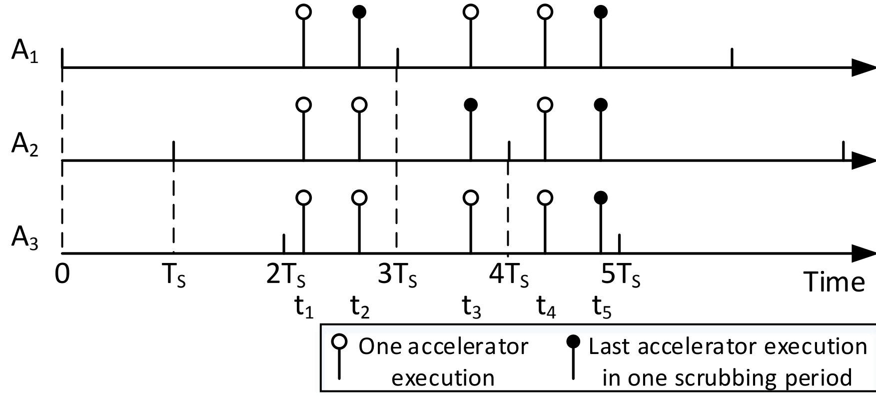

Figure 7.5: Illustrative execution series of an accelerated function

error-free. The reliability of a kernel K_k, i.e. the probability that it delivers correct results, can be formulated as

$$R(K_k) = \prod_{AF_j \in K_k} P(\text{every exec. of } AF_j \text{ is error-free}). \tag{7.20}$$

Since the configuration bits of accelerators in AF_j are independent of each other, the probability of error-free execution of an AF can be decomposed into the product of the probability of error-free execution of its accelerators:

$$\begin{aligned} &P(\text{every execution of } AF_j \text{ is error-free}) = \\ &\prod_{A_i \in AF_j} P(\text{every execution of } A_i \text{ is error-free}). \end{aligned} \tag{7.21}$$

The example shown in Fig. 7.5 is used to illustrate the calculation of the probability of error-free execution of an accelerator. It shows an execution series of an AF consisting of three accelerators. Short solid lines on the time axis represent the time points at which the scrubbing process of respective accelerators is finished. Long solid lines with circles represent the time points (t_1 to t_5) at which the AF is executed. The time difference between the consecutive execution of different accelerators (few tens of cycles) is ignored since it is negligible in comparison to the scrubbing period (thousands of cycles).

The error-free execution of A_1 requires that all 5 executions of A_1 at t_1 to t_5 are error-free. The probabilities of error-free consecutive executions of A_1, e.g. at t_1 and t_2, cannot be considered independent, as they rely on the same configuration bits. The error-free execution at t_2 implies that the underlying configuration bits are error-free throughout the time period from 0 to t_2 and therefore also implies the error-free execution of A_1 at t_1:

$$\begin{aligned} &P(\text{error-free execution of } A_1 \text{ at } t_1 \text{ and } t_2) = \\ &\qquad P(\text{error-free execution of } A_1 \text{ at } t_2) = e^{-\lambda n_1 t_2}. \end{aligned} \tag{7.22}$$

However, after scrubbing at time point $3T_S$, the underlying configuration bits are

ensured to be error-free and thus independent of those before the scrubbing. Therefore, the executions of A_1 at t_3 to t_5 are independent of those at t_1 and t_2 and their correctness is implied by the correctness at t_5. The probability of error-free execution from t_3 to t_5 is

$$\begin{aligned} &P(\text{error-free execution of } A_1 \text{ at } t_3 \text{ to } t_5) = \\ &\qquad P(\text{error-free execution of } A_1 \text{ at } t_5) = e^{-\lambda n_1 (t_5 - 3T_S)}. \end{aligned} \tag{7.23}$$

The probability of error-free executions of A_1 from time 0 throughout t_5 is then

$$P(\text{every execution of } A_1 \text{ is error-free}) = e^{-\lambda n_1 (t_2 + t_5 - 3T_S)}. \tag{7.24}$$

In general, the probability that every execution of an accelerator is error-free within one scrubbing period is equal to the error-free probability of the last execution of the accelerator within that scrubbing period (marked by the long solid lines with filled circles in Fig. 7.5). For example, the probability of error-free execution of A_2 is $e^{-\lambda n_2 (t_3 - T_S + t_5 - 4T_S)}$ and $e^{-\lambda n_3 (t_5 - 2T_S)}$ for A_3.

7.4.3 Effective Critical Bits of Accelerated Kernels and Applications

The execution of an accelerator A_i that occurs just before the next scrubbing period starts has the lowest error-free probability $e^{-\lambda n_i N \cdot T_S}$. For an arbitrary execution series of an accelerator A_i in a kernel K_k with total execution time T_k, the lower bound of the its error-free probability can be obtained:

$$P(\text{every execution of } A_i \text{ is error-free}) \geq e^{-\lambda n_i \frac{T_k}{N \cdot T_S} N \cdot T_S}. \tag{7.25}$$

where $\frac{T_k}{N \cdot T_S}$ represents the number of scrubbing periods occurred during the execution of A_i. Therefore, the lower bound of the reliability of the kernel can be derived from Eq. (7.20), (7.21) and (7.25):

$$\begin{aligned} R(K_k, T_k) &\geq e^{-\lambda n_k N \cdot T_S} \text{ with} \\ n_k(T_k) &= \frac{T_k}{N \cdot T_S} \sum_{AF_j \in K_k} \sum_{A_i \in AF_j} n_i, \end{aligned} \tag{7.26}$$

where n_k denotes the effective critical bits of kernel K_k, which is dependent on its own execution time T_k.

An application *App* delivers error-free results when all of its accelerated kernels $\{K_k\}$ are executed error-free. With Eq. (7.26), the lower bound of the reliability of the application and its effective critical bits n_{app} is obtained:

$$\begin{aligned} R(App) &= \prod_k R(K_k) \geq e^{-\lambda n_{app} N \cdot T_S} \text{ with} \\ n_{app} &= \sum_k n_k = \frac{1}{N \cdot T_S} \sum_k T_k \sum_{AF_j \in K_k} \sum_{A_i \in AF_j} n_i. \end{aligned} \tag{7.27}$$

7.4.4 Budgeting of Effective Critical Bits

Equation (7.27) shows that the reliability of an application is able to stay above a certain lower bound depending on the application's effective critical bits. The reliability constraint r of an application denotes that the probability of obtaining an error-free result is greater than $1-10^{-r}$ and it is satisfied when its reliability lower bound satisfies

$$e^{-\lambda n_{app} N \cdot T_S} \geq 1 - 10^{-r} \tag{7.28}$$

which is equivalent to

$$n_{app} \leq -\frac{1}{\lambda N \cdot T_S} \log(1 - 10^{-r}). \tag{7.29}$$

If the number of effective critical bits of the application is lower than the value given in Eq. (7.29), then the reliability requirement is automatically satisfied. This work proposes a two-step budgeting method which assigns the maximum allowable number of effective critical bits of the application to kernels and accelerated functions. In the first step, the number of effective critical bits of each kernel is determined such that the application performance is maximized. In the second step, the number of effective critical bits of each AF in the kernels is calculated, which indirectly determines the required redundancy for each AF.

Budgeting for Kernels

To find the number of effective critical bits allowed for individual kernels, which depends on the kernel execution time, it is necessary to determine the relationship between the number of accelerators and the kernel execution time.

The amount of resources available for hardware acceleration is limited by the number of reconfigurable regions, which can be devoted to (i) accelerators of different types to accelerate different functions, (ii) accelerators of the same type for parallel execution and (iii) accelerators of the same type paired to compose DWC or TMR. All these different scenarios of resource usage lead to different reliability-performance trade-offs.

It is assumed here that the reliability-*unaware* runtime system of the reconfigurable architecture is optimized to maximize the application performance for the given reconfigurable regions. The execution time $T_k(N)$ of individual kernels k is determined offline for different numbers N of regions in the reconfigurable fabric. When more regions are available for acceleration, more independent accelerators can be configured and the execution time of the kernel is reduced. When accelerators are paired to implement modular redundancy, the number of used regions devoted to acceleration is actually reduced. The acceleration provided by two accelerators paired in DWC or three accelerators paired in TMR is as much as only one accelerator. In other word, M DWC or TMR pairs of accelerators reduce the number of regions available for acceleration by M or $2M$, respectively.

Given the maximum number of effective critical bits of the application determined by Eq. (7.29), the problem of finding the number of effective critical bits of each kernel K_k while maximizing the application performance, i.e. minimizing the execution time of all kernels $\sum_k T_k$, is formulated as follows:

$$\begin{aligned}
\text{minimize } & \sum_k T_k(N_k^{acc}) \\
\text{subject to } & \sum_k (n_k(T_k) - \Delta_k) \leq -\frac{\log(1-10^{-E})}{\lambda N \cdot T_S} \\
& \Delta_k \approx (1-\alpha(\mu_k))\,\mu_k M_k + \\
& \qquad \mu_k \max(M_k - (N - ||K_k||), 0) \\
& \mu_k = \frac{n_k(T_k)}{||K_k||} \\
& N_k^{acc} = \min(||K_k||, N - M_k) \\
& M_k \leq \min(||K_k||, \lfloor N/2 \rfloor),
\end{aligned}$$

where M_k denotes the number of DWC pairs in kernel K_k, $||K_k||$ denotes the number of regions used by the kernel without any redundancy, and μ denotes the average number of critical bits in one region before applying DWC. Δ_k estimates the reduction of the effective critical bits in kernel k after introducing M_k DWC pairs. N_k^{acc} denotes the number of regions available for acceleration. In the above formulation, it is assumed that accelerator redundancy is achieved by DWC. If TMR is chosen as the redundancy method, Δ_k, N_k^{acc} and M_k need to be changed as follows:

$$\begin{aligned}
& \Delta_k \approx (1-\beta(\mu_k))\,\mu_k M_k + \mu_k \max(2M_k - (N - ||K_k||), 0) \\
& N_k^{acc} = \min(||K_k||, N - 2M_k) \\
& M_k \leq \min(||K_k||, \lfloor N/3 \rfloor),
\end{aligned}$$

where M_k denotes the number of TMR pairs in kernel K_k.

The number of redundancy pairs M_k needed in each kernel is determined by iterating through all possible combinations of M_k. The complexity of solving the problem is thus $(N/2)^{||K||}$ for DWC or $(N/3)^{||K||}$ for TMR, where $||K||$ denotes the number of kernels. This search process is performed before the application starts and when the soft error rate changes. Since the number of kernels is typically small, in spite of the exponential time complexity, the runtime overhead is low, as evaluated in Section 7.5.

After determining M_k, the number of effective critical bits of each kernel after budgeting, i.e. $n'_k = n_k - \Delta_k$, can be calculated. It determines the maximum total number of effective critical bits of accelerated functions in each kernel: $\sum_{AF_j \in K_k} n'_j = n'_k N \cdot T_S / T_k$ (see Eq. (7.26)).

Budgeting for Accelerated Functions

During the execution of each kernel, the accelerators to implement the AFs co-exist in the reconfigurable fabric at the same time. They share the reconfigurable regions among each other. The runtime system selects the appropriate implementation variants for each AF, such that the performance of the kernel is maximized for a given number of reconfigurable regions. More effective critical bits budgeted to an AF allow it to be implemented with more accelerators for higher performance. Therefore, AFs that contribute more to the computation in the kernel or require complex accelerators (more intrinsic critical bits) shall be assigned with more effective critical bits.

Given the total number of effective critical bits $n'_k N \cdot T_S/T_k$ that are budgeted to all accelerated functions in each kernel, the budgeted critical bits of each accelerated function n'_j are calculated by solving the following equation:

$$\frac{n'_j}{n'_k N \cdot T_S/T_k} = \frac{w_j n_j}{\sum_{AF_j \in K_k} w_j n_j}, \tag{7.30}$$

where w_j and n_j denote the proportion of the execution time of AF_j in the kernel and the number of critical bits of AF_j obtained from the reliability-unaware kernel profiling, respectively. Before the start of each kernel, the reliability-unaware runtime system then selects from those AF implementations, whose number of effective critical bits are within the budget, to maximize the kernel performance.

7.5 Experimental Evaluation

The presented method is evaluated in a reconfigurable architecture implemented on a Xilinx Virtex-5 FPGA with an H.264 video encoder as the target application (see Section 3.5). The application consists of three kernels which contain multiple accelerated functions that are composed from 9 distinct accelerators as in Table 3.1. The number of critical bits of the accelerators are obtained from the Xilinx `bitgen` tool and range from 19,036 to 86,796 bits. The reliability requirement specifies the upper bound of the error probability of one encoded frame, i.e. to guarantee the reliability of the application encoding one frame. Therefore, the method of budgeting of effective critical bits (see Section 7.4.4) is applied.

The architectural simulator (see Section 3.5) is used to evaluate the method for systems with different number of reconfigurable regions. The algorithm for finding the effective critical bits for kernels and AFs is implemented in fixed-point arithmetic and integrated it into the runtime system. The reliability model presented in Section 7.2 is integrated into the simulator to evaluate the application reliability. To evaluate the response of the system to different environmental conditions, the soft error rate is changed from 0.1 to 10 errors per Mb per month to simulate realistic cases [QGM+13].

For the performance evaluation, the method of critical bits budgeting is applied

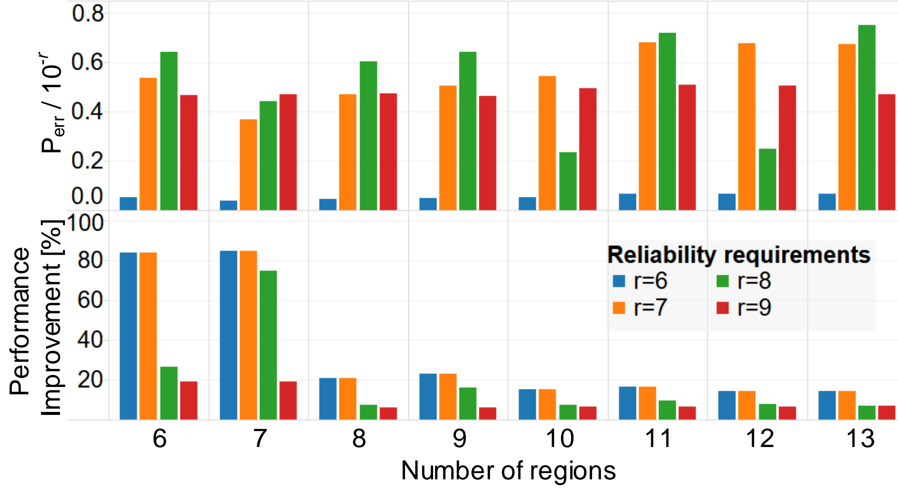

Figure 7.6: Ratios of error probability and performance improvement under different numbers of regions and reliability requirements

with reliability requirements from $r=6$ to $r=9$ (see Section 7.4.4), i.e. the error probability of each encoded frame must be less than 10^{-r}. DWC is applied as the redundancy mode and compare the execution time for encoding one frame to an approach which applies DWC to all used accelerators in the AFs (full DWC). The error probability of a full DWC system is close to zero. To evaluate the achieved application reliability, we use the ratio of the calculated application error probability (see Section 7.2) and the required error probability ($P_{err}/10^{-r}$). A value smaller than 1 implies that the reliability requirement is satisfied.

7.5.1 Performance Improvement

Figure 7.6 shows the results in systems with different number of reconfigurable regions when the soft error rate is set to $3\,\mathrm{Mb}^{-1}\mathrm{month}^{-1}$. As can be seen from the upper part of the figure, the reliability requirements are satisfied for all evaluated cases. Without any modular redundancy, the error probability is lower than 10^{-6} and closer to 10^{-7}, which leads to the smaller value of $P_{err}/10^{-r}$ in case of $r=6$. When the number of available regions is small, full DWC incurs high performance loss, because some accelerated functions have to be implemented in software due to lack of regions. In contrast, the proposed method only duplicates those accelerators that are essential to the required application reliability, which leads to 85% performance improvement.

Figure 7.7 shows the results under different soft error rates in a system with 8 regions. When the soft error rates and the reliability requirements are low, it is not necessary to duplicate all accelerators to achieve high reliability, which translates to about 20% performance improvement against full DWC while still satisfying the required reliability. When the soft-error rate raises, more accelerators need to be duplicated to compensate the increasing error rate. In worst case, almost all accelerators are duplicated and the resulting error probability is lower than 10^{-15}, as shown in the upper part of Fig. 7.7, where the red bars are too small to be visible

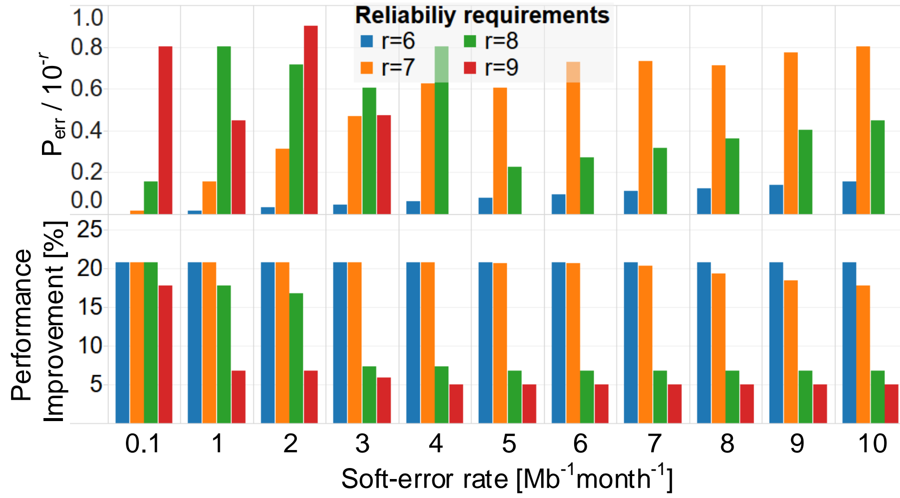

Figure 7.7: Ratios of error probability and performance improvement under different soft-error rates and reliability requirements

for soft-error rates higher than $4\,\mathrm{Mb}^{-1}\mathrm{month}^{-1}$ and r=9. The resulting reliability and performance of the system converges to a full DWC system.

7.5.2 Runtime Overhead

The budgeting of effective critical bits introduces two types of runtime overhead. They are due to the computation of budgeted critical bits for each kernel (Type 1) and each AF (Type 2). Type 1 overhead occurs before the start of an application and when the soft error rate changes. In the worst case, which corresponds to a system with 13 regions, it takes 4.1 ms on a SPARC V8 LEON3 processor running at 100 MHz. Type 2 overhead occurs before the start of each kernel by solving Eq. (7.30) and takes 0.07 ms.

8 Overall Evaluation and Comparison

This chapter presents and discusses the evaluation results of the proposed dependability approaches from the perspectives of structural integrity and functional correctness, as described in Section 3.4. They show the effectiveness of the proposed methods against permanent faults in the reconfigurable fabric resulted from aging effects and soft errors in the configuration memory due to single event upsets. The results are compared with related state-of-the-art techniques.

8.1 Structural Integrity

In the presence of permanent faults, the system behavior and the transistor stress in the fabric are investigated in a reconfigurable architecture with eight reconfigurable regions. A complex H.264 video encoder (with nine distinct accelerators), an ADPCM audio encoder (one accelerator), an AES encryption (one accelerator) and a JPEG image decoder (three accelerators) were chosen as target applications to represent different computational requirements. Table 8.1 shows all accelerators along with their logic utilizations (2nd column) and bitstream sizes (3rd column). The total bitstream size of all configurations that need to be stored in the system memory is about 1636 KB. To accommodate the accelerator resource requirement, each reconfigurable region has a size of 4×20 CLBs with eight 6-input LUTs per CLB. The only exceptions are the JPEG accelerators that use 8×20 CLBs for their reconfigurable regions. They would have fit into 4×20 CLB regions when using DSP blocks. But as the transistor-level structure of Xilinx DSP blocks is not publicly known, the stress balancing for DSP blocks cannot be evaluated and thus the JPEG accelerators are implemented using CLBs only. A minimal set of diversified configurations to tolerate any single-CLB fault is generated for each accelerator (4th column).

For the evaluation, a fault model at CLB granularity is assumed. A CLB is considered faulty if any faulty behavior in its internal structure makes it unusable for functional operation. It is not limited to a specific fault model (e.g. stuck-at faults), but it is assumed that the position of faulty CLBs is detected and diagnosed. To investigate the system behavior under the presence of faulty CLBs, 1 up to 80 CLBs (from all CLBs of all regions) are randomly selected to be faulty. To ensure correct operation, any accelerator configuration that requires at least one of the faulty CLBs is not allowed to be configured anymore. For a given number of faulty CLBs, 100 simulation runs are executed with randomly selected faulty CLBs.

The architectural simulator (see Section 3.5) is used to evaluate the system behavior in the presence of different number of detected permanent faults and using different runtime strategies. For comparison, a baseline, a state-of-the-art strategy,

Table 8.1: Properties of reconfigurable accelerators and their change in maximum frequency of diversified configurations

Accelerator	CLB utilization [%]	Bitstream [KByte]	#Divers. Config.	Original [MHz]	AccDiv [MHz]	Worstcase Δ [%]
Clip3	66	30	3	133	119–133	10.5
CollapseAdd	23	30	2	158	142–158	10.1
LF_BS4	48	30	2	121	117–121	3.3
LF_Cond	23	30	2	146	139–146	4.8
PointFilter	65	30	3	89	81–89	9.0
QuadSub	9	30	2	257	217–257	15.6
SADrow_4	38	30	4	100	99–103	1.0
SAV	33	30	2	139	122–139	12.2
Transform	45	30	2	167	160–167	4.2
AdpcmEncDec	84	30	7	67	63–67	5.6
AesLutEnc	53	30	3	269	258–269	3.7
JpegTransform	59	52	3	108	98–108	8.8
Jpegidcte	69	52	4	156	135–156	13.1
Jpegidcto	83	52	6	158	149–160	5.5
System freq.				67	63	5.6

STRAP without module diversification and STRAP with module diversification (summarized in Table 8.2) are evaluated:

- The baseline strategy does not perform fault-tolerant and stress-aware placement and each accelerator has only one configuration, i.e. without diversified configurations. When the runtime system decides to place an accelerator into a region where one of the required CLBs is faulty, then the accelerator will not be configured and its functionality will be emulated in software.
- Angermeier *et al.* [AZGT11] proposed another state-of-the-art stress distribution method which places one accelerator after the other into the region that minimizes the peak stress. It is not capable to tolerate faults.
- The 'STRAP alone' strategy distributes the stress of accelerators uniformly into all reconfigurable resources, where each accelerator has only one configuration, i.e. module diversification is not applied. Thus, the runtime placement algorithm provides limited capability of fault tolerance.

Table 8.2: Compared strategies in the experiments

Strategy	Fault-tolerance	Stress-aware	Mod. Div.
Baseline	No	No	No
[AZGT11]	No	Yes	No
STRAP alone	Yes	Yes	No
STRAP+ModDiv	Yes	Yes	Yes

- STRAP together with module diversification can maximally exploit the diversity in different accelerators for fault tolerance and stress distribution.

As proposed for STRAP, [AZGT11] is extended to replace an accelerator if its reconfigurable region has not been reconfigured for 20 million cycles to provide a fair comparison (see Section 6.3.2).

8.1.1 Accelerator Diversification

Since the module diversification method applies additional constraints to prohibit certain CLB locations during place-and-route, the maximally achievable frequency of an accelerator may be affected. Table 8.1 reports the maximum frequency of the diversified configurations of each accelerator (6th column) and of the original configuration (5th column) that is place-and-routed without prohibit constraints. The worst-case frequency impact (7th column) compares the slowest configuration of the accelerator to the original configuration. The place-and-route tool is given a target frequency of 250 MHz as timing constraint to obtain the maximum operating frequency of each accelerator. The frequency of one of the diversified configuration may actually be better than the original configuration, e.g. in the case of SADrow_4 and Jpegidcto, because the additional placement constraints may actually help the place-and-route tool to explore new placement and routing possibilities that are not discovered during the generation of the original configuration. The maximum system frequency is however limited by the accelerator with the longest critical path, i.e. AdpcmEncDec, which runs at 67.2 MHz with the original configuration and at 63.4 MHz with the slowest configuration, which leads to 5.6% decrease of the system frequency. The original configuration is one of the diversified configurations and thus can be used when full performance is required.

8.1.2 Aging Resilience and Fault Tolerance

The MTTF improvement due to the stress reduction is calculated by assuming that a device fails when ΔV_{th} of any transistor exceeds 50% of its original value (V_{th0}). Table 8.3 reports the dynamic stress reduction for different benchmark applications in a fault-free system and Table 8.4 reports the resulting MTTF increase. On average, STRAP with module diversification achieves 6.8× higher MTTF than the baseline, 1.8× higher than [AZGT11] and 1.6× higher than using STRAP alone.

Table 8.3: Reduction of maximum transistor toggle rate w.r.t. the baseline system [%]

Strategy	H.264	ADPCM	JPEG	AES	Avg.
[AZGT11]	49.5	87.3	63.9	49.7	62.6
STRAP alone	66.0	87.3	73.6	49.7	69.2
STRAP+ModDiv	78.5	90.8	87.3	73.5	82.5

Table 8.4: MTTF improvement w.r.t. the baseline system [×] (e.g. 2× improvement means the MTTF is doubled)

Strategy	H.264	ADPCM	JPEG	AES	Avg.
[AZGT11]	2.0	7.9	2.8	2.0	3.7
STRAP alone	2.9	7.9	3.8	2.0	4.1
STRAP+ModDiv	4.7	10.9	7.9	3.8	6.8

The impact of faults in the fabric on the application performance and the peak stress is investigated in detail for a H.264 video encoder as the target application.

Figure 8.1 shows the application performance in the presence of faults in the reconfigurable fabric when different strategies are applied. The box plots (whose values refer to the left Y-axis) show the statistical distribution of the performance degradation w.r.t. a fault-free baseline system, which is measured by

$$\frac{\text{Execution time in the presence of } n \text{ faulty CLBs}}{\text{Execution time in a fault-free baseline system}}. \tag{8.1}$$

It represents how many times *slower* the application runs than it would run in a fault-free system. If only one or two CLBs are detected faulty, the application performance is typically not affected, as these faulty CLBs are not required by the accelerators. When the number of faulty CLBs increases in the baseline system, fewer accelerators can be placed and computationally intensive parts of the application have to be executed in software, which significantly degrades the performance. In extreme cases, e.g. with 80 faulty CLBs, the application is completely executed in software without acceleration, which leads to more than 22× degraded performance.

With module diversification, the application experiences less than 8% performance degradation with 1 to 8 faulty CLBs, since these faulty CLBs are avoided by using diversified configurations. As faults accumulate (e.g. more than 25 faulty CLBs), they are not tolerable any more by the diversified configurations, which leads to the increased performance degradation. Without module diversification, STRAP alone delivers limited capability of fault tolerance. The application performance degrades in a similar rate to the baseline system after the number of faulty CLBs exceeds 7.

The line plots in Fig. 8.1 (with values referring to the right Y-axis) show the average performance gain of a runtime strategy w.r.t. the baseline system, given the same number of faulty CLBs. It is measured by

$$\frac{\text{Avg. Exec. time in baseline with } n \text{ faulty CLBs}}{\text{Avg. Exec. time in other strategy with } n \text{ faulty CLBs}}, \tag{8.2}$$

where the average is over all simulation runs for a given number of faulty CLBs. This metric measures the ability of a strategy to provide fault-tolerance. With the STRAP algorithm and module diversification, the application is able to deliver 1.9–3.7× the performance of a baseline system in the presence of 4 to 40 faulty CLBs. As the number of faulty CLBs increases, the system performance is gracefully degraded until it converges to the baseline system. Without module diversification, STRAP

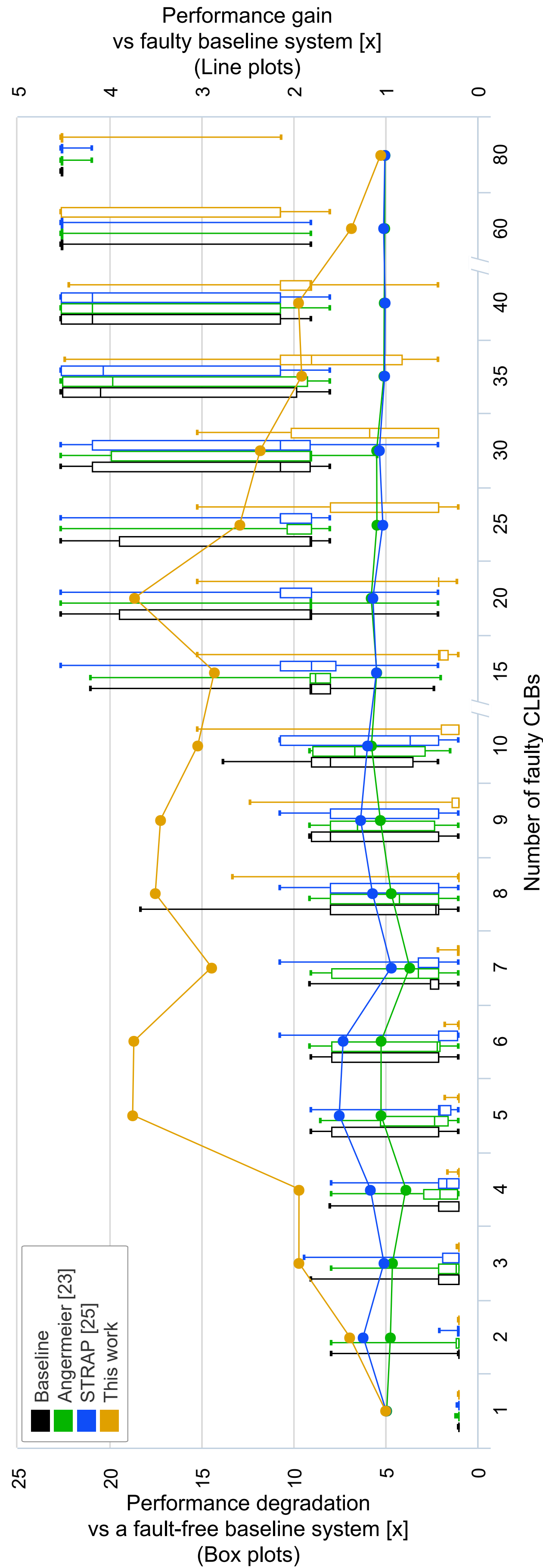

Figure 8.1: Application performance in the presence of faults under different strategies. Left Y-axis (box plots): performance degradation w.r.t. a fault-free baseline system. Right Y-axis (line plots): performance gain w.r.t. to the faulty baseline system

alone delivers up to 1.5× performance gain for fewer than 10 faulty CLBs. [AZGT11] optimizes for stress reduction and do not consider fault-tolerance during accelerator placement. It place an accelerator into a region that results in the lowest peak stress. However, this may prevent the successful placement of other accelerators, which leads to lower application performance.

The box plots (left Y-axis) in Fig. 8.2 shows the statistical distribution of peak dynamic stress in the reconfigurable fabric in the presence of different number of CLB faults. When there are only a few (e.g. fewer than 6) faulty CLBs in the reconfigurable fabric, accelerators have a higher placement freedom. The runtime system can freely choose into which region an accelerator shall be placed such that the maximum stress is minimized. All stress-aware strategies avoid the stress accumulation in individual CLBs and result in a lower peak stress than the baseline system. [AZGT11] performs only inter-region stress distribution and thus produces higher peak stress than other stress-aware strategies in systems with fewer faults (e.g. 1 to 3 faulty CLBs), i.e. high placement freedom.

Combining module diversification and intra- and inter-region stress distribution, STRAP+ModDiv produces the most uniform stress distribution in the fabric, which leads to the lowest peak stress compared to other approaches. The placement freedom diminishes as more CLBs become faulty and the stress distribution becomes ineffective. As can be seen from the resulting peak stress, STRAP alone does not have clear advantage over the baseline system when the fabric has 6 to 10 faulty CLBs. As the number of faulty CLBs increases further (e.g. 15 to 60 faulty CLBs), the resulting peak stress from STRAP+ModDiv is higher than that from other strategies. The reason behind that is, that other strategies are not capable to tolerate these faults and are not able to find feasible placements of accelerators into the faulty regions, i.e. the resources in the reconfigurable fabric are less stressed by the execution of accelerators. This can be seen by looking at the performance degradation in Fig. 8.1, where computations are more frequently emulated in software instead of being executed on the reconfigurable fabric.

The line plots (right Y-axis) in Fig. 8.2 show the utilization of the reconfigurable fabric for acceleration w.r.t. a fault-free baseline system. The *fabric utilization* is defined as follows:

$$\frac{\text{Exec. time in the reconfigurable fabric (accelerated)}}{\text{Total execution time}}. \tag{8.3}$$

The values plotted are

$$\frac{\text{Average utilization in the presence of faults}}{\text{Utilization in the fault-free baseline system}}. \tag{8.4}$$

The average is over all simulation runs for a given number of faulty CLBs. The fault-free baseline system has a fabric utilization value of 75.0%. It means that 75% of the total execution time is spent in the execution of accelerators in the reconfigurable fabric while the rest 25% is on the processor pipeline. Since the fault-free baseline system is optimized for performance which fully utilizes the reconfigurable fabric, 75% is the maximum utilization value that can be achieved. In the presence of

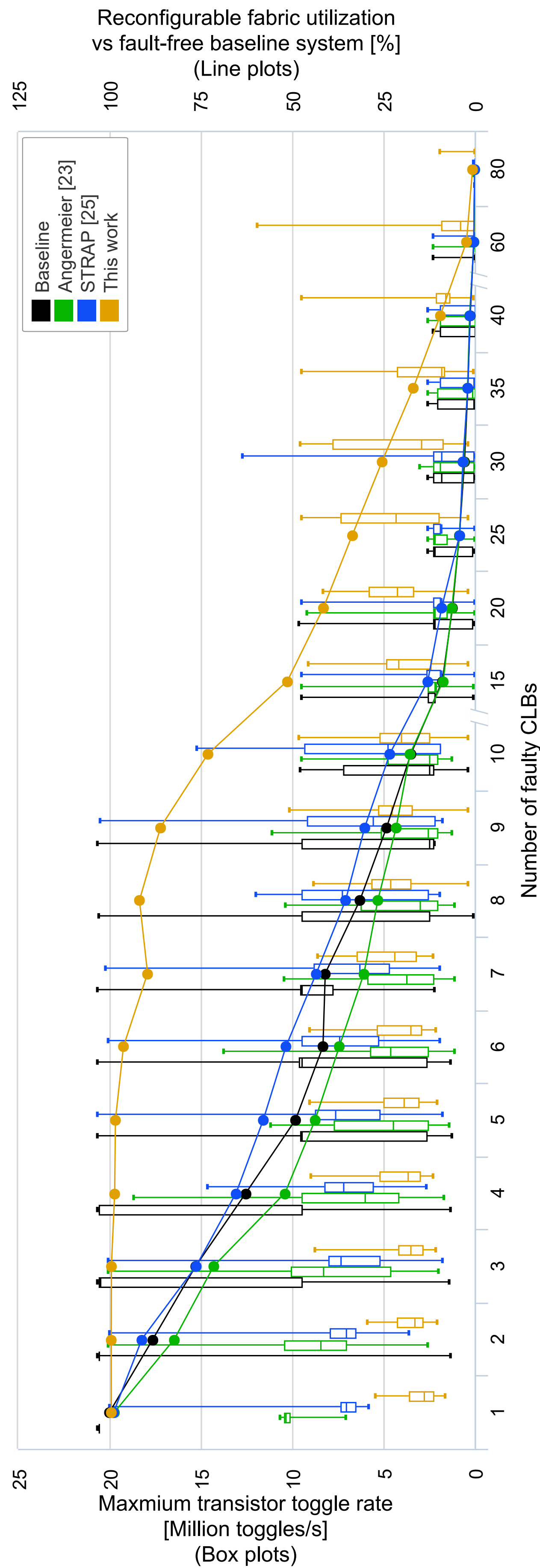

Figure 8.2: Peak stress and utilization in the reconfigurable fabric in the presence of faults. Left Y-axis (box plots): maximum transistor toggle rate. Right Y-axis (line plots): utilization of the reconfigurable fabric for acceleration w.r.t. a fault-free baseline system

faults, requested accelerators may not be placed into the reconfigurable fabric due to faults, which leads to a reduced fabric utilization and less CLB stress. In extreme case (e.g. with 60 or 80 faulty CLBs) the reconfigurable fabric is not used at all as almost all accelerator functions are emulated in software. With fault tolerance by diversified configurations, nearly full fabric utilization is achieved for less than 7 faulty CLBs. Even at a high amount of faults (e.g. 15 to 35 faulty CLBs), the reconfigurable fabric is still used for acceleration, as can be seen from the induced stress. At around 25% full fabric utilization, the application delivers more than $2\times$ higher performance than other strategies (see Fig. 8.1).

For the evaluated systems, the worst-case runtime overhead of the accelerator placement algorithm occurs when 8 accelerators need to be placed into 8 regions. These calculations only take 1.3 ms on a LEON3 processor running at 100 MHz.

8.2 Functional Correctness

To evaluate the functional correctness of the system, i.e. the capability to produce correct outputs, in response to different environmental conditions (e.g. radiation), soft error rates between 0 (no errors) and 10 errors $\mathrm{Mb}^{-1}\mathrm{month}^{-1}$ are simulated to comprise the realistic cases [QGM+13]. The variation speed is in the order of seconds to exercise the dynamic adaptation in the system. Therefore, a sinusoidal soft error rate is used as input stimuli for the runtime system. The period corresponds to 10 s in real time for a 100 MHz clock frequency.

To evaluate the performance of a H.264 encoder application under different reliability constraints and different soft error rates, the runtime system is imposed with reliability constraints from $r = 8$ to $r = 11$ (see Section 7.3), i.e. the error probability of each AF execution must be less than 10^{-r}. It is compared to a threshold-based approach similar to [JCG+12] which duplicates (DWC) or triplicates (TMR) every accelerator when the error rate exceeds 1.8 $\mathrm{Mb}^{-1}\mathrm{month}^{-1}$. This guarantees that the AF error probability is always less than 10^{-10}. In the threshold-based approach, scrubbing is performed at maximum rate.

The results are shown in Fig. 8.3. Depending on the soft error rate, the system reacts and implements modular redundancy methods adaptively. These require hardware resources which are no longer available for acceleration and thus the performance decreases. With more relaxed reliability constraints (i.e. smaller values of r), it is less probable that modular redundancy are required and therefore less performance impact is observed. When the threshold-based methods switch to duplicated or triplicated implementations, much more resources are consumed. This causes a stark performance drop. For a low soft error rate, the performance is still below the proposed approach since the high scrubbing frequency blocks the configuration port and delays the reconfiguration of accelerators.

Figure 8.4 shows the average AF error probability of the approaches. In the unprotected system, the failure probability reaches 4.4×10^{-6}. For the proposed method, the error probability is effectively bound by the given reliability constraint, even for higher soft error rates. The step-shaped change in the curve for $r = 11$ is due

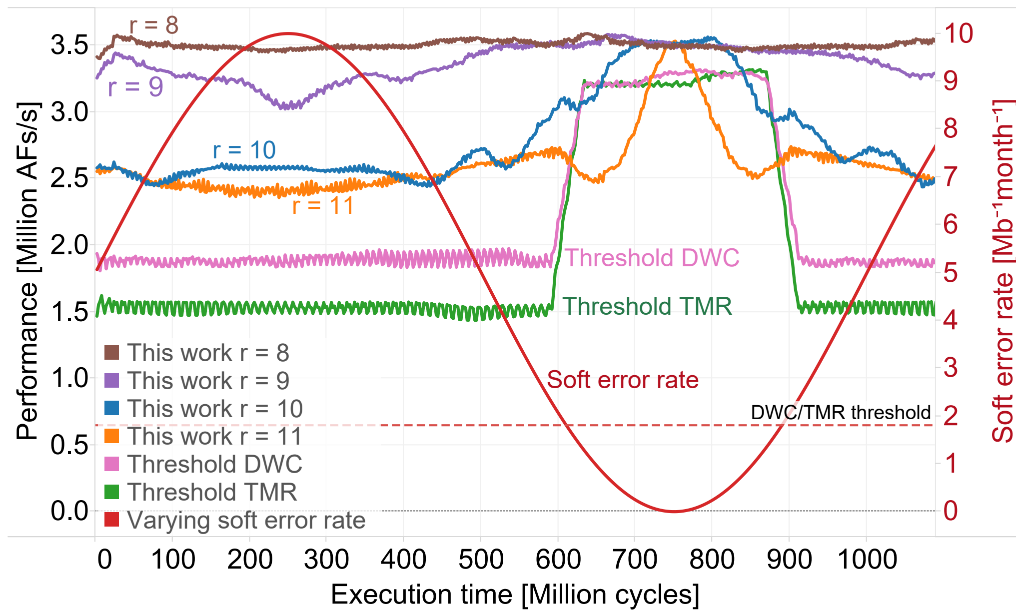

Figure 8.3: Performance under varying soft error rate

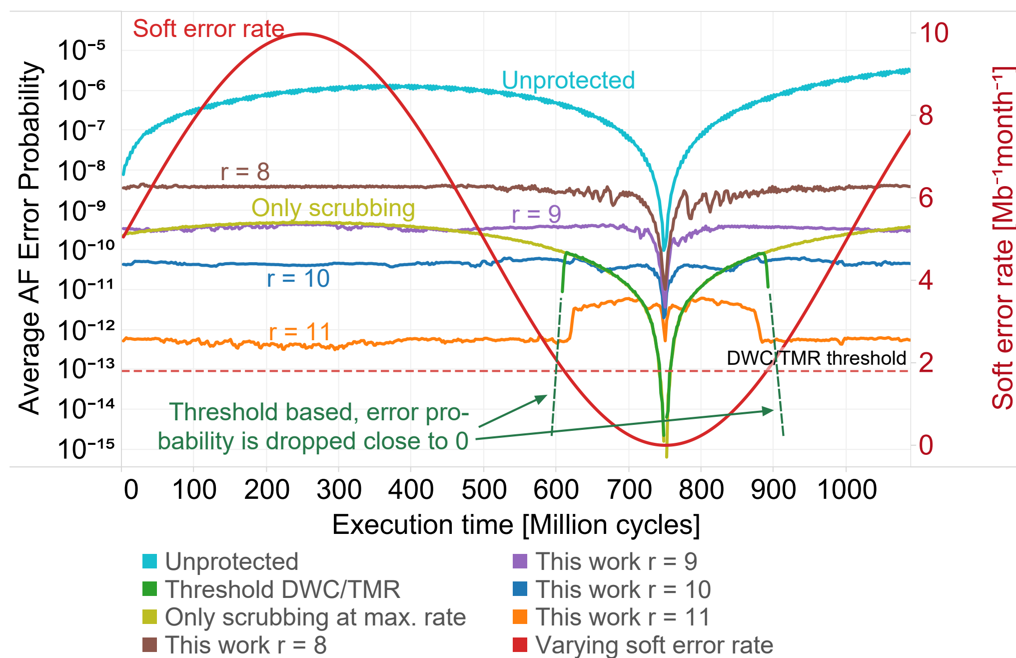

Figure 8.4: Average AF error probability for different fault tolerance methods under varying soft error rate

to the large gap in error probability in the selection space (see Fig. 7.3). A system which applies only scrubbing at maximal frequency can obtain a minimum error probability of approx. 10^{-9}. The threshold-based methods over-protect the system when scrubbing alone cannot guarantee sufficient reliability and accelerators are replicated. Then, the resource usage is excessive with adverse performance impact as shown in Fig. 8.3.

Figure 8.5 gives an overview of the system performance degradation over a wide span of soft error rate and reliability specifications, in comparison to the scenario

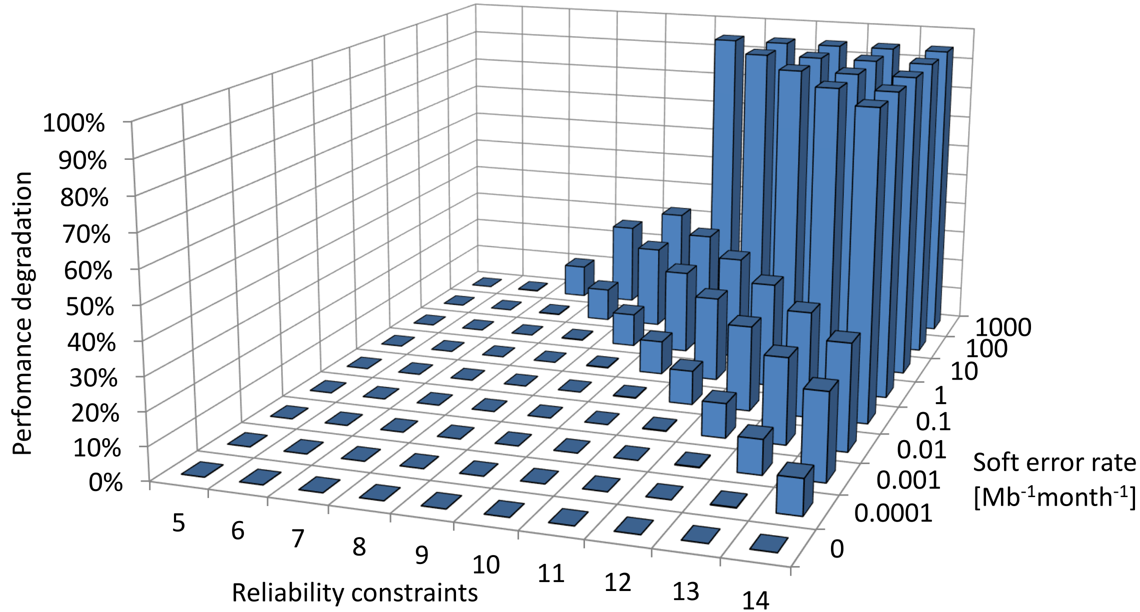

Figure 8.5: Performance degradation over a wide range of soft error rates and reliability constraints

when the system does not need any fault tolerance techniques (i.e. soft error rate is zero). Given reliability constraint $r = 10$, the system performance degradation is kept under 27% with the proposed method even when the soft error rate tends to 100 $\text{Mb}^{-1}\text{month}^{-1}$, while the threshold-based approach (see Fig. 8.3) shows 57% performance degradation due to static and pessimistic design decisions.

Table 8.5 summarizes the results of the scenarios investigated in Fig. 8.3 and Fig. 8.4. For $r = 9$, this work achieves a *reliability improvement factor* of 2384 at only 5.3% average performance reduction compared to the unprotected system. Compared to the threshold-based methods, this work guarantees the same target reliability while providing 20.0% (DWC) or 42.6% (TMR) higher performance in average. In the best case, this work is up to 34.8% (DWC) or 68.3% (TMR) faster.

Table 8.5: Performance and error probability results

Method	Perf. [Million AFs/s]			Error Probability	
	min	avg	max	avg	max
Unprotected	3.45	3.56	3.60	8.70×10^{-7}	4.40×10^{-6}
Thresh. DWC	1.81	2.34	3.30	8.28×10^{-12}	9.12×10^{-11}
Thresh. TMR	1.45	1.97	3.32	8.28×10^{-12}	9.12×10^{-11}
This work r=8	3.40	3.50	3.59	3.49×10^{-9}	5.49×10^{-9}
This work r=9	3.03	3.37	3.58	3.65×10^{-10}	5.62×10^{-10}
This work r=10	2.44	2.81	3.56	4.77×10^{-11}	7.91×10^{-11}
This work r=11	2.36	2.61	3.53	1.48×10^{-12}	6.92×10^{-12}

9 Conclusion and Future Work

9.1 Thesis Conclusion

A sustainable down-scaling in the semiconductor industry is of great importance for driving innovations in an ever more digitalized and interconnected world, which in turn serves to promote progress and prosperity of the human society. The benefits of higher logic density, faster processing speed and lower energy consumption brought by smaller transistor size can only be utilized if the nano-CMOS devices perform dependable computation, i.e. produce results that we can trust and rely on. However, Latent faults escaped from manufacturing tests, transistor degradation caused by microscopic aging effects and soft errors induced by environmental radiation threaten the dependable operation of modern semiconductor devices.

FPGA-based reconfigurable architectures provide the capability to change the hardware organization at runtime so that compute-intensive functions of different applications can be efficiently accelerated using the reconfigurable fabric on FPGAs. The structural integrity of the reconfigurable fabric and the error-free configuration data of accelerators are of crucial significance for the dependable operation of reconfigurable architectures. Targeting permanent and transient faults, this thesis provides a comprehensive solution for maintaining the structural integrity of FPGAs and guaranteeing the functional correctness of applications. These are enabled by the following proposed techniques:

- The underlying hardware structures of the reconfigurable fabric and the error-free accelerator reconfiguration process are ensured by on-demand and periodic testing: Pre-configuration test checks the structural integrity of the underlying hardware before the instantiation of accelerators and Post-configuration test periodically checks the correct functionality of configured accelerators after they are instantiated. The tight integration of the test schemes into the runtime system and the system scheduling minimize the performance overhead while achieving a high fault coverage and low test latency. The structure of the whole reconfigurable fabric with 5 regions can be exhaustively tested every 4 seconds. The time a soft-error remains undetected in the system is reduced by up to two orders of magnitude to 64.7 µs. These are achieved at a performance cost of less than 4.4 percent.
- A novel design method called module diversification was developed to tolerate permanent faults in the reconfigurable regions. For each module or accelerator, a set of configurations is generated that is diversified in terms of their CLB usages, such that for every CLB in a region, at least one configuration of a module does not require that CLB. A generic algorithm was developed to generate the minimal set of configurations to tolerate arbitrary single-CLB

faults and to generate additional configurations to tolerate multi-CLB faults in a reconfigurable region. If a fault is localized in a region, a diversified configuration of a module can be reconfigured into the region at runtime that does not use the faulty CLB. Self-repairing was achieved from the application perspective since in the present of faults the application experiences no performance degradation. For a set of benchmark modules, experimental results showed a significant improvement of module reliability due to fault tolerance, where reliability improvement factors between 19 and 330 were achieved. For an H.264 video application, the system experiences less than 8% performance degradation with 1 to 8 faulty CLBs, since these faulty CLBs are avoided by using diversified configurations. In addition, it delivers from 1.9× up to 3.7× the performance of a baseline system in presence of 4 to 40 faulty CLBs.

- System degradation threatened by aging effects was addressed by a novel stress-aware placement method to reduce the maximum stress induced from workload. It combines complex offline optimizations at synthesis time with situation-dependent adaptation at runtime to balance the intra- and inter-region stress distribution simultaneously. At the runtime, it places accelerators to different reconfigurable regions (i.e. it decides to which region they shall be reconfigured) while considering the induced intra- and inter-region stress distribution simultaneously. At the synthesis time, it diversifies stress during place-and-route by preventing overlapping of high-stress CLBs from different accelerators, which further improves the intra-region stress distribution at runtime. Compared to state-of-the-art methods, this work significantly reduces the maximum dynamic and static stress by up to 64% and 35% with negligible impact on application performance, respectively. As a result, STRAP achieves up to 177% and 14% Mean-Time-To-Failure (MTTF) improvement relative to the closest competitors w.r.t. Hot Carrier Injection (HCI) and Bias Temperature Instability (BTI) aging, respectively.
- An adaptive runtime system was developed that guaranteed a target reliability in a varying environment while optimizing the performance. To guarantee the required reliability for individual Accelerated Functions (AFs), the runtime system dynamically determines whether the AF should be executed by a hardened processor, or whether it should be accelerated by inherently less reliable reconfigurable hardware which can trade-off performance and reliability. Compared to related work with statically optimized fault tolerance techniques, the proposed method provides up to 68.3% higher performance at the same target reliability. To guarantee the required reliability for an application consisting of multiple functions, the runtime system performs budgeting of critical bits, i.e. decomposing the critical bits allowed by an application into critical bits allowed by its computational kernels and then into critical bits of their AFs. This budgeting enables the runtime system to select appropriate accelerators and fault tolerance techniques to maximize the performance under a given target reliability. Compared to a strategy that duplicates all accelerators in the system, this method achieves up to 85% higher performance for a variety of reliability targets and soft error rates.

In these techniques, the runtime system makes decisions based on parameters

propagated across multiple layers and is able to perform a precise cause-effect analysis, which leads to the delivery of optimized impact directly on the concerned dependability objectives.

9.2 Future Work

A basic architectural assumption in this thesis is that all reconfigurable regions are of the same dimension. A heterogeneous reconfigurable fabric can be considered in the future work. The regions can be of different sizes, have different types of resources such as DSPs and floating point units, which in turn exhibit different performance levels and vulnerability to aging and SEUs.

When the assumption of a hardened core pipeline is dropped, software transformation techniques such as those proposed in [RSKH11] can be employed to generate different software variants for a given function to trade-off performance and reliability. Combining software variants and hardware variants will create a larger exploration space for the runtime system. Complex decisions have to be made whether a function shall be executed in software or be accelerated in hardware and with which variants to meet the performance and reliability goals. In addition, stress distribution method can be extended to balance the stress between the processor and the reconfigurable fabric, where the different degradation rates of different types of resources need to be taken into consideration. A function executed in software induces stress in the processor but not in the fabric and vice versa.

Multi-tasking poses another challenge to the runtime system since different tasks have different performance, reliability and resource demands and they compete with each other for available resources. Hence, a resource allocation approach will be needed that considers system decisions for all tasks based on a global knowledge of the task characteristics and requirements. In addition, the resource allocation shall take short-term and long-term solutions for performance, reliability and lifetime management into account.

Short-term decisions are employed to quickly react to temporary changes in the system state and environment like exposure to high radiation. Therefore available resources will be distributed according to the required performance and reliability levels of the tasks and configuration requests for test and scrubbing from multiple tasks will be scheduled to minimize conflicts. This can be achieved by a global optimization procedure for scheduling the configurations during runtime that selects the best execution variants depending on the current environmental conditions, available reconfigurable resources (with various degradation levels) and task constraints.

A first step would be to address the following problem: given the reliability constraints of multiple tasks, decide the number of regions shall be allocated to each task while optimizing the average speedup of these tasks. Tasks assigned with fewer regions may need to offload computations to the hardened processor to satisfy the reliability constraints, resulting in less speedup. Tasks assigned with more regions generally achieve higher speedup as they have more freedom to use the region for parallelism or redundancy. Speedup or reliability increase per region may be used

as a metric of efficiency during decision making for the allocation of regions. The runtime algorithm needs to maximize the efficiency of the allocated regions for each task, maximizing the performance under the reliability constraints.

Long-term decisions keep track of the healthiness of resources with respect to aging. Aging of hardware resources is mitigated by distributing the stress between multiple resources in a periodic fashion. This results in a *homogeneous aging* of the regions and increases the availability of computing resources for as long as possible. For critical tasks with high performance demands, available resources can be grouped into subsets of resources for general and critical tasks, where *heterogeneous aging* between groups is applied. The resource utilization is managed in such a way that the critical resources are less stressed in order to ensure high performance and lifetime by pro-active resource reservation and aging mitigation. These subsets can be further subdivided into groups with different performance levels. Given a set of tasks/workload and operating conditions, offline lifetime characterization needs to be performed to aid the online decision.

A first step would be to address the following problem: given the performance constraints of multiple tasks, decide the number *and* location of regions allocated to each task while optimizing the lifetime of the reconfigurable fabric. When no full performance is required, some regions can be turned off (e.g. by power gating) to reduce aging. The stress balancing algorithm proposed in this work distributes the stress uniformly among the regions used by one task (intra-task stress distribution). If multiple tasks share the reconfigurable fabric, the stress should also be distributed among different tasks (inter-task stress distribution) by dynamically reallocating the regions to tasks. Intuitively, tasks inducing less stress should be assigned with highly stressed regions and vice versa.

A Proof of the Minimal Set Generation in Module Diversification

This section gives the proof that for a module using U CLBs and implemented in a $X \times Y$ region, the first generated $\lceil \frac{XY}{XY-U} \rceil$ configurations by Alg. 2, including the initial configuration, satisfy the completeness and max diversification condition. In addition, these configurations constitute the *minimal* set of configurations that satisfy both conditions, since at least $\lceil \frac{XY}{XY-U} \rceil$ configurations are required to satisfy the completeness condition (see Section 5.2.2). Furthermore, the random swapping in Lines 46 to 48 of Alg. 2 is not executed during the generation process of the minimal set, as no repeated configuration is generated.

First, let's define a few special sets and notations that will be used later in the proof.

Definition 1. *Let the universe set* $\mathbb{U}$ *be the set of positions of all elements in an* $X \times Y$ *matrix:*

$$\mathbb{U} = \{(x,y) | 1 \le x \le X, 1 \le y \le Y \text{ and } x,y \in \mathbb{N}\}.$$

Given a set of configuration matrices, each using U *CLBs*

$$C = \{\mathbf{A_1}, \mathbf{A_2}, \cdots\}, \mathbf{A_i} : X \times Y \text{ Boolean matrix,}$$

let $\mathbb{O}_i$ *be the set of positions of all zero-elements (i.e. unused CLBs) in* $\mathbf{A_i}$*:*

$$\mathbb{O}_i = \{(x,y) | [\mathbf{A_i}]_{x,y} = 0\} \subset \mathbb{U}$$

and similarly $\mathbb{I}_i$ *the set of positions of all one-elements (i.e. used CLBs) in* $\mathbf{A_i}$*:*

$$\mathbb{I}_i = \{(x,y) | [\mathbf{A_i}]_{x,y} = 1\} \subset \mathbb{U}.$$

Give the universe set $\mathbb{U}$*, we denote the complement of a subset* $\mathbb{X}$ *of* $\mathbb{U}$*, which is* $\mathbb{U} \setminus \mathbb{X}$*, simply as* $\mathbb{X}^{\mathsf{C}}$*. Thus,* $\mathbb{O}_i^{\mathsf{C}} = \mathbb{U} \setminus \mathbb{O}_i = \mathbb{I}_i$ *and* $\mathbb{I}_i^{\mathsf{C}} = \mathbb{U} \setminus \mathbb{I}_i = \mathbb{O}_i$*. Also note that* $|\mathbb{O}_i| = XY - U$ *and* $|\mathbb{I}_i| = U$*.*

Algorithm 2 generates configuration matrices iteratively. In each iteration, incremental changes are made on the configuration matrix generated in the last iteration ($\mathbf{A_i}$) to produce a new one ($\mathbf{A_{i+1}}$), where zero-elements in $\mathbf{A_i}$ are swapped with one-elements in $\mathbf{A_i}$ to generate $\mathbf{A_{i+1}}$. In other words, given the initial configuration matrix $\mathbf{A_1}$, $\mathbf{A_2}$ is generated by swapping zero-elements and one-elements in $\mathbf{A_1}$, $\mathbf{A_3}$ is generated by swapping zero-elements and one-elements in $\mathbf{A_2}$, $\cdots$, $\mathbf{A_{i+1}}$ is generated by swapping zero-elements and one-elements in $\mathbf{A_i}$.

There are two cases that need to be proved separately:

1. When the number of one-elements $|\mathbb{I}_i|$ is less than or equal to the number of zero-elements $|\mathbb{O}_i|$, and

2. When the number of one-elements $|\mathbb{I}_i|$ is greater than the number of zero-elements $|\mathbb{O}_i|$.

PROOF OF CASE 1) $|\mathbb{I}_1| \leq |\mathbb{O}_1|$ $(U \leq \frac{1}{2}XY)$:

In this case $\lceil \frac{XY}{XY-U} \rceil = 2$. $\mathbf{A_2}$ is generated by swapping all one-elements with a part of or all zero-elements in $\mathbf{A_1}$ depending on whether the equal sign holds, which leads to $\mathbb{I}_1 \subseteq \mathbb{O}_2$ and $\mathbb{I}_2 \subseteq \mathbb{O}_1$, or rather $\mathbb{I}_1 \subseteq \mathbb{I}_2^{\complement}$ and $\mathbb{I}_2 \subseteq \mathbb{I}_1^{\complement}$. Therefore we have

$$\mathbb{I}_1 \cap \mathbb{I}_2 = \emptyset \text{ and thus } \sum_{x,y} [\mathbf{A_1}]_{x,y} \cdot [\mathbf{A_2}]_{x,y} = 0,$$

i.e. both completeness and max diversification conditions are satisfied. □

In the second case, as there are more one-elements than zero-elements, we need to select a subset of one-elements to swap in each iteration. This selection is based on the score matrix $\mathbf{S}$.

Lemma 1. *Let the score matrix* $\mathbf{S}$ *be the sum of* N *configuration matrices:* $\mathbf{S} = \sum_{1\leq i\leq N} \mathbf{A_i}$. *If* $\forall 1 \leq i < j \leq N : \mathbb{O}_i \cap \mathbb{O}_j = \emptyset$, *then*

$$\forall (x,y) \in \bigcup_{1\leq i\leq N} \mathbb{O}_i : [\mathbf{S}]_{x,y} = N-1.$$

Proof. Because for any two matrices $\mathbf{A_i}$ and $\mathbf{A_j}$ $(i \neq j)$, they do not have common zero elements $(\mathbb{O}_i \cap \mathbb{O}_j = \emptyset)$, namely $\forall (x,y) \in \mathbb{O}_i : [\mathbf{A_i}]_{x,y} = 0, [\mathbf{A_j}]_{x,y} = 1$ and $\forall (x,y) \in \mathbb{O}_j : [\mathbf{A_j}]_{x,y} = 0, [\mathbf{A_i}]_{x,y} = 1$, so $\forall (x,y) \in \mathbb{O}_i \cup \mathbb{O}_j : [\mathbf{A_i} + \mathbf{A_j}]_{x,y} = 1$.

Therefore we have

$$\begin{aligned}\forall (x,y) \in \mathbb{O}_1 : [\mathbf{S}]_{x,y} &= \sum_{i=1}^{N} [\mathbf{A_i}]_{x,y} = [\mathbf{A_1}]_{x,y} + \sum_{i=2}^{N} [\mathbf{A_i}]_{x,y} \\ &= 0 + \sum_{i=2}^{N} ([\mathbf{A_1}]_{x,y} + [\mathbf{A_i}]_{x,y}) \\ &= \sum_{i=2}^{N} 1 = N-1.\end{aligned}$$

Similarly this holds for all other $\mathbb{O}_i$ $(2 \leq i \leq N)$. □

PROOF OF CASE 2) $|\mathbb{I}_1| > |\mathbb{O}_1|$ $(U > \frac{1}{2}XY)$:

In this case $\mathbf{A_{i+1}}$ is generated by swapping in $\mathbf{A_i}$ all zero-elements with those one-elements with higher scores.

In the first iteration, $\mathbf{S} = \mathbf{A_1}$ where

$$\forall (x,y) \in \mathbb{O}_1 : [\mathbf{S}]_{x,y} = 0 \text{ and } \forall (x,y) \in \mathbb{I}_1 : [\mathbf{S}]_{x,y} = 1.$$

After the generation of $\mathbf{A_2}$ by swapping in $\mathbf{A_1}$ all zero-elements with a part of one-elements (all one-elements have equal scores 1), which leads to $\mathbb{O}_2 \subset \mathbb{I}_1$ and $\mathbb{O}_1 \subset \mathbb{I}_2$, we have $\mathbb{O}_1 \cap \mathbb{O}_2 = \emptyset$ and from Lemma 1

$$\forall (x,y) \in \mathbb{O}_1 \cup \mathbb{O}_2 : [\mathbf{S}]_{x,y} = 1.$$

It is also easy to verify that

$$\forall (x,y) \in (\mathbb{O}_1 \cup \mathbb{O}_2)^{\complement} = \mathbb{I}_1 \cap \mathbb{I}_2 : [\mathbf{S}]_{x,y} = 2$$

In the second iteration, we assume that $|\mathbb{I}_1 \cap \mathbb{I}_2| \geq |\mathbb{O}_i|$. $\mathbf{A_3}$ is then generated by swapping in $\mathbf{A_2}$ all zero-elements with those one-elements in the set $\mathbb{I}_1 \cap \mathbb{I}_2$, because they all have a higher score 2 than the rest of one-elements. We get $\mathbb{O}_3 \subseteq \mathbb{I}_1 \cap \mathbb{I}_2$ and thus $\mathbb{O}_3 \cap \mathbb{O}_1 = \emptyset$, $\mathbb{O}_3 \cap \mathbb{O}_2 = \emptyset$. By applying Lemma 1 again we have the updated score matrix $\mathbf{S}$:

$$\forall (x,y) \in \bigcup_{1 \leq i \leq 3} \mathbb{O}_i : [\mathbf{S}]_{x,y} = 2$$

$$\forall (x,y) \in \bigcup_{1 \leq i \leq 3} \mathbb{O}_i^{\complement} = \bigcap_{1 \leq i \leq 3} \mathbb{I}_i : [\mathbf{S}]_{x,y} = 3.$$

In the Nth iteration, as long as $|\cap_{1 \leq i \leq N} \mathbb{I}_i| \geq |\mathbb{O}_i|$, $\mathbf{A_{N+1}}$ is always generated by swapping in $\mathbf{A_N}$ all zero-elements with those one-elements with score N, which results in

$$\mathbb{O}_{N+1} \subseteq \bigcap_{1 \leq i \leq N} \mathbb{I}_i \text{ and} \tag{A.1}$$

$$\forall 1 \leq i < j \leq N+1 : \mathbb{O}_i \cap \mathbb{O}_j = \emptyset. \tag{A.2}$$

and the updated score matrix $\mathbf{S}$:

$$\forall (x,y) \in \bigcup_{1 \leq i \leq N+1} \mathbb{O}_i : [\mathbf{S}]_{x,y} = N$$

$$\forall (x,y) \in \left(\bigcup_{1 \leq i \leq N+1} \mathbb{O}_i \right)^{\complement} = \bigcap_{1 \leq i \leq N+1} \mathbb{I}_i : [\mathbf{S}]_{x,y} = N+1.$$

Note that

$$\left| \bigcap_{1 \leq i \leq N} \mathbb{I}_i \right| = \left| \mathbb{U} \setminus \bigcup_{1 \leq i \leq N} \mathbb{O}_i \right| = |\mathbb{U}| - N\,|\mathbb{O}_i| = XY - N(XY - U).$$

When $|\cap_{1 \leq i \leq N} \mathbb{I}_i| = |\mathbb{O}_i|$ or $N+1 = \frac{XY}{XY-U}$, because we have Eq. (A.1), and thus $\mathbb{O}_{N+1} = \bigcap_{1 \leq i \leq N} \mathbb{I}_i = \mathbb{I}_{N+1}^{\complement}$, so

$$\bigcap_{1 \leq i \leq N+1} \mathbb{I}_i = \emptyset, \tag{A.3}$$

which is equivalent to

$$\sum_{x,y} \prod_{i=1}^{N+1} [\mathbf{A_i}]_{x,y} = 0.$$

The completeness condition is then satisfied with the first generated $N+1$ or $\frac{XY}{XY-U}$ configurations.

The condition $|\cap_{1\leq i\leq N}\mathbb{I}_i| \geq |\mathbb{O}_i|$ fails when $XY - N(XY-U) < XY - U$ or $N > \frac{XY}{XY-U} - 1$. It occurs earliest when $N = \lceil \frac{XY}{XY-U} - 1 \rceil = \lceil \frac{XY}{XY-U} \rceil - 1$. In this iteration, $\mathbf{A_{N+1}}$ (the $\lceil \frac{XY}{XY-U} \rceil$th configuration matrix) is generated by swapping in $\mathbf{A_N}$ $|\cap_{1\leq i\leq N}\mathbb{I}_i|$ zero-elements with those one-elements with score N and swapping the remaining zero-elements with other one-elements with score $N-1$.

$\mathbb{O}_{N+1}$ can then be divided into two disjoint sets $\mathbb{O}^{(1)}_{N+1}$ and $\mathbb{O}^{(2)}_{N+1}$, where

$$\mathbb{O}^{(1)}_{N+1} = \bigcap_{1\leq i\leq N} \mathbb{I}_i \text{ and } \mathbb{O}^{(2)}_{N+1} \subset \bigcup_{1\leq i\leq N} \mathbb{O}_i. \tag{A.4}$$

To check whether the completeness condition is satisfied now, note that

$$\mathbb{O}^{(1)}_{N+1} = \left(\bigcap_{1\leq i\leq N} \mathbb{I}_i\right)^{\complement\complement} = \left(\bigcup_{1\leq i\leq N} \mathbb{O}_i\right)^{\complement} = \mathbb{U} \setminus \left(\bigcup_{1\leq i\leq N} \mathbb{O}_i\right)$$

$$\begin{aligned}
\mathbb{O}^{(1)}_{N+1} \cup \left(\bigcup_{1\leq i\leq N} \mathbb{O}_i\right) &= \mathbb{U} \\
\left(\mathbb{O}^{(1)}_{N+1}\right)^{\complement} \cap \left(\bigcap_{1\leq i\leq N} \mathbb{I}_i\right) &= \emptyset \\
\left(\mathbb{I}_{N+1} \cup \mathbb{O}^{(2)}_{N+1}\right) \cap \left(\bigcap_{1\leq i\leq N} \mathbb{I}_i\right) &= \emptyset \\
\bigcap_{1\leq i\leq N+1} \mathbb{I}_i \cup \left(\mathbb{O}^{(2)}_{N+1} \cap \left(\bigcap_{1\leq i\leq N} \mathbb{I}_i\right)\right) &= \emptyset.
\end{aligned} \tag{A.5}$$

Because

$$\mathbb{O}^{(2)}_{N+1} \subset \bigcup_{1\leq i\leq N} \mathbb{O}_i = \left(\bigcap_{1\leq i\leq N} \mathbb{I}_i\right)^{\complement},$$

and therefore

$$\mathbb{O}^{(2)}_{N+1} \cap \left(\bigcap_{1\leq i\leq N} \mathbb{I}_i\right) = \emptyset.$$

Equation (A.5) remains valid if and only if

$$\bigcap_{1\leq i\leq N+1} \mathbb{I}_i = \emptyset,$$

which is exactly the same as Eq. (A.3), so the completeness condition is in this case also satisfied with the first generated $\lceil \frac{XY}{XY-U} \rceil$ configurations.

Because each configuration matrix $\mathbf{A_{i+1}}$ is generated by swapping in $\mathbf{A_i}$ all zero-

elements with certain one-elements, we have

$$\forall 1 \leq i \leq \lceil \tfrac{XY}{XY-U} \rceil : \mathbb{O}_{i+1} \subseteq \mathbb{I}_i.$$

We can therefore write

$$\begin{aligned} |\mathbb{I}_i \cap \mathbb{I}_{i+1}| &= \left| \mathbb{I}_i \cap \mathbb{O}_{i+1}^{\complement} \right| = |\mathbb{I}_i \setminus \mathbb{O}_{i+1}| = |\mathbb{I}_i| - |\mathbb{O}_{i+1}| \\ &= U - (XY - U) = 2U - XY, \end{aligned}$$

which means

$$\forall 1 \leq i \leq \lceil \tfrac{XY}{XY-U} \rceil : \sum_{x,y} [\mathbf{A_i}]_{x,y} \cdot [\mathbf{A_{i+1}}]_{x,y} = 2U - XY.$$

The max diversification condition is herewith satisfied with the first generated $\lceil \frac{XY}{XY-U} \rceil$ configurations.

It is easy to verify from Eq. (A.2) and (A.4) that

$$\forall 1 \leq i < j \leq \lceil \tfrac{XY}{XY-U} \rceil : \mathbb{O}_i \neq \mathbb{O}_j, \text{ so } \mathbf{A_i} \neq \mathbf{A_j}.$$

Therefore, the while loop in Alg. 2 from Lines 46 to 48 will not be executed for the first $\lceil \frac{XY}{XY-U} \rceil$ configurations. □

B Terrestrial Soft Error Rates in a Virtex-5 FPGA

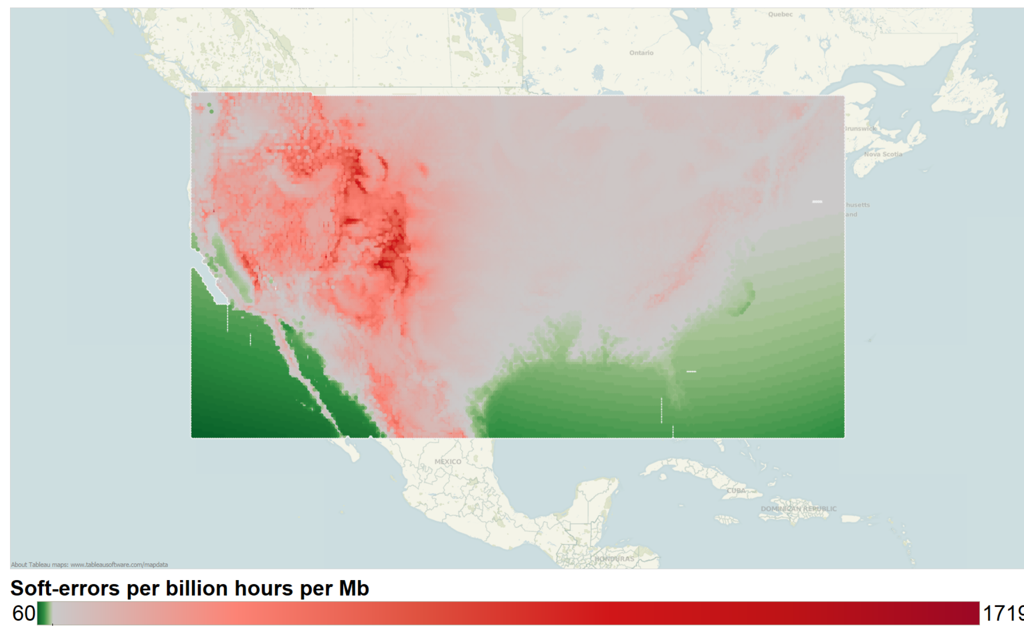

Figure B.1: This figure shows how the soft error rate varys depending on the altitude. The neutron flux is lower at low altitude regions due to atmospheric shielding. It is almost 30 times higher at mountain peaks in the US than it is at sea level.

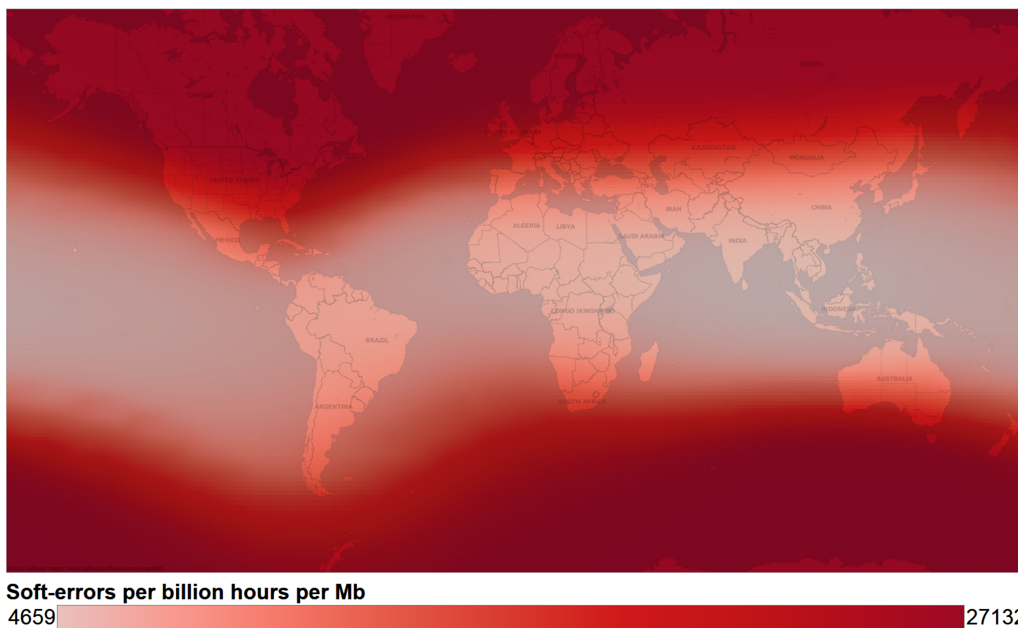

Figure B.2: The neutron flux also varys in the geomagnetic field. This figure shows the resulted variation of soft error rate around the globe at 10 km altitude, where commercial flights typically cruise. In the equator regions, the soft error rate is roughly 6 times lower than it is in other regions.

Bibliography

[ABB+12] M. S. Abdelfattah, L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, H. Zhang, J. Henkel, and H.-J. Wunderlich, "Transparent structural online test for reconfigurable systems," in *Proc. IEEE 18th International On-Line Testing Symposium (IOLTS)*, 2012, pp. 37–42.

[AD16] A. Alzahrani and R. F. DeMara, "Fast online diagnosis and recovery of reconfigurable logic fabrics using design disjunction," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 3055–3069, 2016.

[Aer] Aeroflex Gaisler, "Homepage of the Leon Processor." Online available: http://www.gaisler.com/index.php/products/processors/leon3 (Accessed on 28.10.2016).

[AFC+09] T. Ando, M. M. Frank, K. Choi, C. Choi, J. Bruley, M. Hopstaken, M. Copel, E. Cartier, A. Kerber, A. Callegari, D. Lacey, S. Brown, Q. Yang, and V. Narayanan, "Understanding mobility mechanisms in extremely scaled HfO2 (EOT 0.42 nm) using remote interfacial layer scavenging technique and Vt-tuning dipoles with gate-first process," in *Proc. IEEE International Electron Devices Meeting (IEDM)*, 2009, pp. 1–4.

[Ahm07] A. Ahmadinia, "Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices," *IEEE Transactions on Computers*, vol. 56, no. 5, pp. 673–680, 2007.

[AHOAD11] R. Al-Haddad, R. Oreifej, R. A. Ashraf, and R. F. DeMara, "Sustainable modular adaptive redundancy technique emphasizing partial reconfiguration for reduced power consumption," *International Journal of Reconfigurable Computing*, vol. 2011, no. 2, pp. 1–25, 2011.

[AKGH16] H. Amrouch, B. Khaleghi, A. Gerstlauerz, and J. Henkel, "Reliability-aware design to suppress aging," in *Proc. 53rd Annual Design Automation Conference (DAC)*, 2016, pp. 12:1–12:6.

[AKL12] H. A. Almurib, T. N. Kumar, and F. Lombardi, "A single-configuration method for application-dependent testing of SRAM-based FPGA interconnects," in *Proc. 20th Asian Test Symposium (ATS)*, 2012, pp. 444–450.

[AKS93] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A tutorial on built-in self-test. I. Principles," *IEEE Design & Test of Computers*, vol. 10, no. 1, pp. 73–82, 1993.

[ALHS12] U. Abelein, H. Lochner, D. Hahn, and S. Straube, "Complexity, quality and robustness - the challenges of tomorrow's automotive electronics," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 870–871.

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[ALWA+14] F. Adamu-Lema, X. Wang, S. M. Amoroso, C. Riddet, B. Cheng, L. Shifren, R. Aitken, S. Sinha, G. Yeric, and A. Asenov, "Performance and variability of doped multithreshold FinFETs for 10-nm CMOS," *IEEE Transactions on Electron Devices*, vol. 61, no. 10, pp. 3372–3378, 2014.

[AMv+17] H. Amrouch, S. Mishra, V. van Santen, S. Mahapatra, and J. Henkel, "Impact of BTI on dynamic and static power: From the physical to circuit level," in *Proc. IEEE International Reliability Physics Symposium (IRPS)*, 2017, pp. CR–3.1–CR–3.6.

[And12] T. Ando, "Ultimate scaling of high-k gate dielectrics: Higher-k or interfacial layer scavenging?" *Materials*, vol. 5, no. 12, pp. 478–500, 2012.

[AS01] M. Abramovici and C. E. Stroud, "BIST-based test and diagnosis of FPGA logic

blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, pp. 159–172, 2001.

[ASE04] M. Abramovici, C. E. Stroud, and J. M. Emmert, "Online BIST and BIST-based diagnosis of FPGA logic blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 12, pp. 1284–1294, 2004.

[ASH+99] M. Abramovici, C. Strond, C. Hamilton, S. Wijesuriya, and V. Verma, "Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications," in *Proc. International Test Conference (ITC)*, 1999, pp. 973–982.

[AvE+14] H. Amrouch, V. M. van Santen, T. Ebi, V. Wenzel, and J. Henkel, "Towards interdependencies of aging mechanisms," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 478–485.

[AZGT11] J. Angermeier, D. Ziener, M. Glaß, and J. Teich, "Stress-aware module placement on reconfigurable devices," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 277–281.

[BAG+15] V. Boppana, S. Ahmad, I. Ganusov, V. Kathail, V. Rajagopalan, and R. Wittig, "Xilinx 16nm UltraScale+ MPSoC and FPGA Families," in *Hot Chips 27: A Symposium on High Performance Chips*, 2015.

[Bau05] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, 2005.

[BB07] S. Bhoj and D. Bhatia, "Thermal modeling and temperature driven placement for FPGAs," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007, pp. 1053–1056.

[BBI+12] L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, H. Zhang, H.-J. Wunderlich, and J. Henkel, "OTERA: Online test strategies for reliable reconfigurable architectures — Invited paper for the AHS-2012 special session "Dependability by reconfigurable hardware"," in *Proc. NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 38–45.

[BBI+13] L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, E. Schneider, H. Zhang, J. Henkel, and H.-J. Wunderlich, "Test Strategies for Reliable Runtime Reconfigurable Architectures," *IEEE Transactions on Computers*, vol. 62, no. 8, pp. 1494–1507, 2013.

[BDB+13] A. R. Brown, N. Daval, K. K. Bourdelle, B.-Y. Nguyen, and A. Asenov, "Comparative simulation analysis of process-induced variability in nanoscale SOI and bulk trigate FinFETs," *IEEE Transactions on Electron Devices*, vol. 60, no. 11, pp. 3611–3617, 2013.

[BHW+14] R. Backasch, G. Hempel, S. Werner, S. Groppe, and T. Pionteck, "Identifying homogenous reconfigurable regions in heterogeneous FPGAs for module relocation," in *Proc. International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2014, pp. 1–6.

[BKS00] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design & Test of Computers*, vol. 17, no. 1, pp. 68–83, 2000.

[BKT14] C. Beckhoff, D. Koch, and J. Torresen, "Portable module relocation and bitstream compression for Xilinx FPGAs," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–8.

[Bla69] J. R. Black, "Electromigration—A brief survey and some recent results," *IEEE Transactions on Electron Devices*, vol. 16, no. 4, pp. 338–347, 1969.

[BMS10] A. A. M. Bsoul, N. Manjikian, and L. Shang, "Reliability- and process variation-aware placement for FPGAs," in *Proc. Design, Automation and Test in Europe (DATE)*, 2010, pp. 1809–1814.

[BPP+08] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. A. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis," *IEEE Transactions*

on Nuclear Science, vol. 55, no. 4, pp. 2259–2266, 2008.

[BSB+14] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, “FPGAs in the cloud: Booting virtualized hardware accelerators with OpenStack,” in *Proc. IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014, pp. 109–116.

[BSH08] L. Bauer, M. Shafique, and J. Henkel, “A computation- and communication- infrastructure for modular special instructions in a dynamically reconfigurable processor,” in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2008, pp. 203–208.

[BSH09] L. Bauer, M. Shafique, and J. Henkel, “Cross-architectural design space exploration tool for reconfigurable processors,” in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2009, pp. 958–963.

[BSH11] L. Bauer, M. Shafique, and J. Henkel, “Concepts, architectures, and run-time systems for efficient and adaptive reconfigurable processors,” in *Proc. NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011, pp. 80–87.

[BSKH07] L. Bauer, M. Shafique, S. Kramer, and J. Henkel, “RISPP: rotating instruction set processing platform,” in *Proc. 44th Annual Design Automation Conference (DAC)*, 2007, pp. 791–796.

[BSKH08] L. Bauer, M. Shafique, S. Kreutz, and J. Henkel, “Run-time System for an Extensible Embedded Processor with Dynamic Instruction Set,” in *Proc. Design, Automation and Test in Europe (DATE)*, 2008, pp. 752–757.

[BSN+15] B. Bowhill, B. Stackhouse, N. Nassif, Z. Yang, A. Raghavan, C. Morganti, C. Houghton, D. Krueger, O. Franza, J. Desai, J. Crop, D. Bradley, C. Bostak, S. Bhimji, and M. Becker, “The Xeon® processor E5-2600 v3: A 22nm 18-core product family,” in *Proc. IEEE International Solid-State Circuits Conference (ISSCC)*, 2015, pp. 1–3.

[Car06] C. Carmichael, “Triple Module Redundancy Design Techniques for Virtex FPGAs,” *Xilinx Application Note 197 (v1.0.1)*, 2006.

[CGG+14] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, and G. Reinman, “Accelerator-rich architectures: Opportunities and progresses,” in *Proc. 51st Annual Design Automation Conference (DAC)*, 2014, pp. 180:1–180:6.

[CH11] J. M. P. Cardoso and M. Hübner, *Reconfigurable computing: From FPGAs to hardware/software codesign.* Springer, 2011.

[CKGdK03] F. Crupi, B. Kaczer, G. Groeseneken, and A. de Keersgieter, “New insights into the relation between channel hot carrier degradation and oxide breakdown short channel nMOSFETs,” *IEEE Electron Device Letters*, vol. 24, no. 4, pp. 278–280, 2003.

[CKR+15] A. Chaudhary, B. Kaczer, P. J. Roussel, T. Chiarella, N. Horiguchi, and S. Mahapatra, “Time dependent variability in RMG-HKMG FinFETs: Impact of extraction scheme on stochastic NBTI,” in *Proc. IEEE International Reliability Physics Symposium (IRPS)*, 2015, pp. 3B.4.1–3B.4.8.

[Con15] Continental AG, “Supplier Requirements Manual,” 2015.

[CSZ+14] F. Chen, Y. Shan, Y. Zhang, Y. Wang, H. Franke, X. Chang, and K. Wang, “Enabling FPGAs in the cloud,” in *Proc. 11th ACM Conference on Computing Frontiers*, 2014, pp. 1–10.

[CVS+14] Y. Cao, J. Velamala, K. Sutaria, M. S.-W. Chen, J. Ahlbin, I. S. Esqueda, M. Bajura, and M. Fritze, “Cross-layer modeling and simulation of circuit reliability,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 1, pp. 8–23, 2014.

[DKS+10] A. Dasu, R. Kallam, A. Sudarsanam, J. Carver, and R. Barnes, “Dynamically reconfigurable systolic array accelerators: A case study with extended Kalman filter and discrete wavelet transform algorithms,” *IET Computers & Digital Techniques*, vol. 4, no. 2, pp. 126–142, 2010.

[DLW09] D. G. Drmanac, F. Liu, and L.-C. Wang, “Predicting variability in nanoscale lithog-

raphy processes," in *Proceedings of the 46th Annual Design Automation Conference (DAC)*, 2009, p. 545.

[DSSF10] P. E. Dodd, M. R. Shaneyfelt, J. R. Schwank, and J. A. Felix, "Current and Future Challenges in Radiation Effects on CMOS Electronics," *IEEE Transactions on Nuclear Science*, vol. 57, no. 4, pp. 1747–1763, 2010.

[DZT12] C. Dennl, D. Ziener, and J. Teich, "On-the-fly Composition of FPGA-Based SQL Query Accelerators Using a Partially Reconfigurable Module Library," in *Proc. IEEE 20th International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2012, pp. 45–52.

[EBS+11] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[Ent07] R. Entner, *Modeling and simulation of negative bias temperature instability*, 2007.

[ESA07] J. M. Emmert, C. E. Stroud, and M. Abramovici, "Online fault tolerance for FPGA logic blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 2, pp. 216–226, 2007.

[FCMG13] V. Ferlet-Cavrois, L. W. Massengill, and P. Gouker, "Single Event Transients in Digital CMOS—A Review," *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 1767–1790, 2013.

[Fri73] A. D. Friedman, "Easily testable iterative systems," *IEEE Transactions on Computers*, vol. C-22, no. 12, pp. 1061–1064, 1973.

[GB11] C. Galuzzi and K. Bertels, "The instruction-set extension problem: A survey," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 4, no. 2, pp. 1–28, 2011.

[GB16] Z. Ghaderi and E. Bozorgzadeh, "Aging-aware high-level physical planning for reconfigurable systems," in *Proc. 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 631–636.

[GBF+04] D. Gil, T. A. Brunner, C. Fonseca, N. Seong, B. Streefkerk, C. Wagner, and M. Stavenga, "Immersion lithography: New opportunities for semiconductor manufacturing," *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures*, vol. 22, no. 6, p. 3431, 2004.

[GBH12] A. Grudnitsky, L. Bauer, and J. Henkel, "Partial online-synthesis for mixed-grained reconfigurable architectures," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 1555–1560.

[GBS14] X. Guo, W. Burleson, and M. Stan, "Modeling and experimental demonstration of accelerated self-healing techniques," in *Proc. 51st Annual Design Automation Conference (DAC)*, 2014, pp. 1–6.

[GCS+06] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2006, no. 7, pp. 1–19, 2006.

[Gha91] P. B. Ghate, "Industrial perspective on reliability of VLSI devices," *MRS Proceedings*, vol. 225, 1991.

[GMP15] E. Giaquinta, A. Mishra, and L. Pozzi, "Maximum convex subgraphs under I/O constraint for automatic identification of custom instructions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 3, pp. 483–494, 2015.

[GSR+14] R. Glein, B. Schmidt, F. Rittner, J. Teich, and D. Ziener, "A self-adaptive SEU mitigation system for FPGAs with an internal block RAM radiation particle sensor," in *Proc. IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014, pp. 251–258.

[Gup15] P. K. Gupta, "Xeon+FPGA platform for the data center," in *Proc. Workshop on the Intersections of Computer Architecture and Reconfigurable Logic*, vol. 119, 2015.

[Han16] L. Hansen, "Unleash the Unparalleled Power and Flexibility of Zynq UltraScale+ MPSoCs," *WP470 (v1.1)*, 2016.

[HB03] H. L. Hughes and J. M. Benedetto, "Radiation effects and hardening of MOS technology: Devices and circuits," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 500–521, 2003.

[HBD+13] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *Proc. 50th Annual Design Automation Conference (DAC)*, 2013, pp. 1–10.

[HBGZ14] J. Henkel, L. Bauer, A. Grudnitsky, and H. Zhang, "Adaptive embedded computing with i-core," *ACM SIGBED Review*, vol. 11, no. 3, pp. 20–21, 2014.

[HCF+15] V. Huard, F. Cacho, X. Federspiel, W. Arfaoui, M. Saliva, and D. Angot, "Technology scaling and reliability: Challenges and opportunities," in *Proc. IEEE International Electron Devices Meeting (IEDM)*, 2015, pp. 20.5.1–20.5.6.

[HCJ+90] H.-C. Hsieh, W. S. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey, and R. Kanazawa, "Third-generation architecture boosts speed and density of field-programmable gate arrays," in *Proc. IEEE Custom Integrated Circuits Conference (CICC)*, 1990, pp. 31.2/1–31.2/7.

[HGV+06] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, 2006.

[HHB+12] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hubner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe, "Invasive manycore architectures," in *Proc. 17th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012, pp. 193–200.

[HHH+11] J. Henkel, L. Hedrich, A. Herkersdorf, R. Kapitza, D. Lohmann, P. Marwedel, M. Platzner, W. Rosenstiel, U. Schlichtmann, O. Spinczyk, M. Tahoori, L. Bauer, J. Teich, N. Wehn, H.-J. Wunderlich, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, and H. Härtig, "Design and architectures for dependable embedded systems," in *Proc. 9th International Conference on Hardware/-Software Codesign and System Synthesis (CODES+ISSS)*, 2011, pp. 69–78.

[HL99] C. Hu and Q. Lu, "A unified gate oxide reliability model," in *Proc. IEEE 37th International Reliability Physics Symposium (IRPS)*, 1999, pp. 47–51.

[HM01] W.-J. Huang and E. J. McCluskey, "Column-based precompiled configuration techniques for FPGA," in *Proc. 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2001, pp. 137–146.

[HMCL98] W. K. Huang, F. J. Meyer, X.-T. Chen, and F. Lombardi, "Testing configurable LUT-based FPGA's," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 2, pp. 276–283, 1998.

[HMO81] C. M. Hsieh, P. C. Murley, and R. R. O'Brien, "A field-funneling effect on the collection of alpha-particle-generated carriers in silicon devices," *IEEE Electron Device Letters*, vol. 2, no. 4, pp. 103–105, 1981.

[Hol16] W. M. Holt, "Moore's law: A path going forward," in *Proc. IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 8–13.

[HS15] J. Hussein and G. Swift, "Mitigating Single-Event Upsets," *Xilinx White Paper 395 (v1.1)*, 2015.

[HSWK09] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2009, pp. 99–104.

[HTH+85] C. Hu, S. C. Tam, F.-C. Hsu, P.-K. Ko, T.-Y. Chan, and K. W. Terrill, "Hot-electron-induced MOSFET degradation—Model, monitor, and improvement," *IEEE Transactions on Electron Devices*, vol. 32, no. 2, pp. 375–385, 1985.

[IBM15] IBM, "Coherent Accelerator Processor Interface User's Manual (Version 1.2)," 2015.

[IEC10] "IEC Functional Safety and IEC 61508," IEC, 2010. Online available: http://www.iec.ch/functionalsafety/ (Accessed on 21.07.2016).

[IMF98] T. Inoue, S. Miyazaki, and H. Fujiwara, "Universal fault diagnosis for lookup table FPGAs," *IEEE Design & Test of Computers*, vol. 15, no. 1, pp. 39–44, 1998.

[IPD10] A. Ilias, K. Papadimitriou, and A. Dollas, "Combining duplication, partial reconfiguration and software for on-line error diagnosis and recovery in SRAM-based FPGAs," in *Proc. 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 73–76.

[ISO11] "ISO 26262: Road vehicles – Functional safety," ISO, 2011. Online available: http://www.iso.org/iso/catalogue_detail?csnumber=43464 (Accessed on 14.11.2016).

[JCG+12] A. Jacobs, G. Cieslewski, A. D. George, A. Gordon-Ross, and H. Lam, "Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive FPGA-based space computing," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 5, no. 4, pp. 1–30, 2012.

[JED06] JEDEC Standard, "Measurement and Reporting of Alpha Particles and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices," *JEDEC Solid State Technology Association*, 2006.

[JMGM12] K. Joshi, S. Mukhopadhyay, N. Goel, and S. Mahapatra, "A consistent physical framework for N and P BTI in HKMG MOSFETs," in *Proc. IEEE International Reliability Physics Symposium (IRPS)*, 2012, pp. 5A.3.1–5A.3.10.

[KAT11] S. Kiamehr, A. Amouri, and M. B. Tahoori, "Investigation of NBTI and PBTI induced aging in different LUT implementations," in *Proc. International Conference on Field-Programmable Technology (FPT)*, 2011, pp. 1–8.

[Kau67] W. H. Kautz, "Testing for faults in combinational cellular logic arrays," in *Proc. 8th Annual Symposium on Switching and Automata Theory (SWAT)*, 1967, pp. 161–174.

[KHT08] D. Koch, C. Haubelt, and J. Teich, "Efficient reconfigurable on-chip buses for FPGAs," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, K. L. Pocek, Ed., 2008, pp. 287–290.

[KKK+08] K. Kuhn, C. Kenyon, A. Kornfeld, M. Liu, A. Maheshwari, W.-k. Shih, S. Sivakumar, G. Taylor, P. VanDerVoorn, and K. Zawadzki, "Managing process variation in Intel's 45nm CMOS technology," *Intel Technology Journal*, vol. 12, no. 2, 2008.

[KKYY09] A. Kanamaru, H. Kawai, Y. Yamaguchi, and M. Yasunaga, "Tile-based fault tolerant approach using partial reconfiguration," in *Proceedings of the 5th International Workshop on Reconfigurable Computing (ARC)*, vol. 5453, 2009, pp. 293–299.

[KR07] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[KTR08] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[Kuh12] K. J. Kuhn, "Considerations for ultimate CMOS scaling," *IEEE Transactions on Electron Devices*, vol. 59, no. 7, pp. 1813–1828, 2012.

[Lal01] P. K. Lala, *Self-checking and fault-tolerant digital design.* Morgan Kaufmann, 2001.

[LBW12] M. Lin, Y. Bai, and J. Wawrzynek, "Selectively fortifying reconfigurable computing device to achieve higher error resilience," *Journal of Electrical and Computer Engineering*, pp. 5:5–5:5, 2012.

[Le12] R. Le, "Soft Error Mitigation Using Prioritized Essential Bits," *Xilinx Application Note 538 (v1.0)*, 2012.

[Lie13] J. Lienig, "Electromigration and its impact on physical design in future technologies," in *Proc. ACM International Symposium on Physical Design (ISPD)*, 2013, p. 33.

[LKC+13] K. T. Lee, W. Kang, E.-A. Chung, G. Kim, H. Shim, H. Lee, H. Kim, M. Choe, N.-I. Lee, A. Patel, J. Park, and J. Park, "Technology scaling on High-K & Metal-Gate FinFET BTI reliability," in *Proc. IEEE International Reliability Physics Symposium (IRPS)*, 2013, pp. 2D.1.1–2D.1.4.

[LMSP98] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 2, pp. 212–221, 1998.

[LQB08] X. Li, J. Qin, and J. B. Bernstein, "Compact modeling of MOSFET wearout mechanisms for circuit-reliability simulation," *IEEE Transactions on Device and Materials Reliability*, vol. 8, no. 1, pp. 98–121, 2008.

[LSV06] R. Lysecky, G. Stitt, and F. Vahid, "Warp Processors," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 3, pp. 659–681, 2006.

[LWL+14] S. E. Liu, J. S. Wang, Y. R. Lu, D. S. Huang, C. F. Huang, W. H. Hsieh, J. H. Lee, Y. S. Tsai, J. R. Shih, Y.-H. Lee, and K. Wu, "Self-heating effect in FinFETs and its impact on devices reliability characterization," in *Proc. IEEE International Reliability Physics Symposium (IRPS)*, 2014, pp. 4A.4.1–4A.4.4.

[Mah15] S. Mahapatra, *Fundamentals of bias temperature instability in MOS transistors: Characterization methods, process and materials impact, DC and AC modeling*, ser. Springer series in advanced microelectronics, 2015, vol. 52.

[McC85] E. McCluskey, "Built-in self-test techniques," *IEEE Design & Test of Computers*, vol. 2, no. 2, pp. 21–28, 1985.

[MDS06] D. Milton, S. Dhingra, and C. E. Stroud, "Embedded processor based built-in self-test and diagnosis of logic and memory resources in FPGAs," in *Proc. International Conference on Embedded Systems and Applications (ESA)*, 2006, pp. 87–93.

[MHS+04] S. Mitra, W.-J. Huang, N. R. Saxena, S.-Y. Yu, and E. J. McCluskey, "Reconfigurable architecture for autonomous self-repair," *IEEE Design & Test of Computers*, vol. 21, no. 3, pp. 228–240, 2004.

[MKSM03] J. McPherson, J.-Y. Kim, A. Shanware, and H. Mogul, "Thermochemical description of dielectric breakdown in high dielectric constant materials," *Applied Physics Letters*, vol. 82, no. 13, p. 2121, 2003.

[MMM+13] C. Ma, H. J. Mattausch, M. Miyake, T. Iizuka, M. Miura-Mattausch, K. Matsuzawa, S. Yamaguchi, T. Hoshida, M. Imade, R. Koh, T. Arakawa, and J. He, "Compact reliability model for degradation of advanced p-MOSFETs due to NBTI and hot-carrier effects in the circuit simulation," in *Proc. IEEE International Reliability Physics Symposium (IRPS)*, 2013, pp. 2A.3.1–2A.3.6.

[Moo75] G. E. Moore, "Progress in digital integrated electronics," in *Proc. IEEE International Electron Devices Meeting (IEDM)*, 1975, pp. 11–13.

[Muk08] S. Mukherjee, *Architecture design for soft errors.* Morgan Kaufmann Publishers, 2008.

[NOY77] T. H. Ning, C. M. Osburn, and H. N. Yu, "Emission probability of hot electrons from silicon into silicon dioxide," *Journal of Applied Physics*, vol. 48, no. 1, p. 286, 1977.

[NR11] A. N. Nowroz and S. Reda, "Thermal and power characterization of field-programmable gate arrays," in *Proc. 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2011, pp. 111–114.

[NSC13] G. L. Nazar, L. P. Santos, and L. Carro, "Accelerated FPGA repair through shifted scrubbing," in *Proc. 23rd International Conference on Field Programmable Logic and Applications (FPL)*, 2013, pp. 1–6.

[OWTK10] A. Oetken, S. Wildermann, J. Teich, and D. Koch, "A bus-based SoC architecture for flexible module placement on reconfigurable FPGAs," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2010, pp. 234–239.

[PA12] M. Psarakis and A. Apostolakis, "Fault tolerant FPGA processor based on runtime reconfigurable modules," in *Proc. 17th IEEE European Test Symposium (ETS)*, 2012, pp. 1–6.

[PAS+09] C. Patterson, P. Athanas, M. Shelburne, J. Bowen, J. Surís, T. Dunham, and J. Rice, "Slotless module-based reconfiguration of embedded FPGAs," *ACM Transactions on Embedded Computing Systems*, vol. 9, no. 1, pp. 1–26, 2009.

[PBH+11] M. M. Pereira, L. Braun, M. Hubner, J. Becker, and L. Carro, "Run-time resource

instantiation for fault tolerance in FPGAs," in *Proc. NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011, pp. 88–95.

[PC03] T. Pi and P. J. Crotty, "FPGA lookup table with transmission gate structure for reliable low-voltage operation," Patent US6 667 635, 2003.

[PCC+14] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proc. ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 13–24.

[PCG+06] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," in *Proc. 44th International Reliability Physics Symposium (IRPS)*, 2006, pp. 226–232.

[PGP98] M. Psarakis, D. Gizopoulos, and A. Paschalis, "Test generation and fault simulation for cell fault model using stuck-at fault model based test tools," *Journal of Electronic Testing*, vol. 13, no. 3, pp. 315–319, 1998.

[PTM] "Predictive Technology Model." Online available: http://ptm.asu.edu/

[QGM+13] H. Quinn, P. Graham, K. Morgan, Z. Baker, M. Caffrey, D. Smith, M. Wirthlin, and R. Bell, "Flight Experience of the Xilinx Virtex-4," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2682–2690, 2013.

[RAKT13] P. M. B. Rao, A. Amouri, S. Kiamehr, and M. B. Tahoori, "Altering LUT configuration for wear-out mitigation of FPGA-mapped designs," in *Proc. 23rd International Conference on Field Programmable Logic and Applications (FPL)*, 2013, pp. 1–8.

[RBC+13] A. Rahman, P. Bai, G. Curello, J. Hicks, C.-H. Jan, M. Jamil, J. Park, K. Phoa, M. S. Rahman, C. Tsai, B. Woolery, and J.-Y. Yeh, "Reliability studies of a 22nm SoC platform technology featuring 3-D tri-gate, optimized for ultra low power, high performance and high density application," in *Proc. IEEE International Reliability Physics Symposium (IRPS)*, 2013, pp. PI.2.1–PI.2.6.

[RCN03] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits (2nd Edition)*. Pearson Education, 2003.

[Ren98] M. Renovell, "SRAM-based FPGAs: a structural test approach," in *Proc. XI Brazilian Symposium on Integrated Circuit Design*, 1998, pp. 67–72.

[RP88] M. L. Reed and J. D. Plummer, "Chemistry of Si-SiO2 interface trap annealing," *Journal of Applied Physics*, vol. 63, no. 12, p. 5776, 1988.

[RPFZ97] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian, "Test pattern and test configuration generation methodology for the logic of RAM-based FPGA," in *Proc. 6th Asian Test Symposium (ATS)*, 1997, pp. 254–259.

[RSKH11] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel, "Reliable software for unreliable hardware," in *Proc. 9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011, pp. 237–146.

[SAB+13] K. Schuegraf, M. C. Abraham, A. Brand, M. Naik, and R. Thakur, "Semiconductor logic technology innovation to achieve sub-10 nm manufacturing," *IEEE Journal of the Electron Devices Society*, vol. 1, no. 3, pp. 66–75, 2013.

[SC11] E. Stott and P. Y. K. Cheung, "Improving FPGA reliability with wear-levelling," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 323–328.

[Shi16] C. Shin, *Variation-aware advanced CMOS devices and SRAM*. Springer, 2016.

[SKCA96] C. Stroud, S. Konala, P. Chen, and M. Abramovici, "Built-in self-test of logic blocks in FPGAs (Finally, a free lunch: BIST without overhead!)," in *Prof. 14th IEEE VLSI Test Symposium (VTS)*, 1996, pp. 387–392.

[SKM78] S. SU, I. Koren, and Y. Malaiya, "A Continuous-Parameter Markov Model and Detection Procedures for Intermittent Faults," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 567–570, 1978.

[SKM+08] S. Srinivasan, R. Krishnan, P. Mangalagiri, Yuan Xie, V. Narayanan, M. J. Irwin, and K. Sarpatwari, "Toward Increasing FPGA Lifetime," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 2, pp. 115–127, 2008.

[SRK04] P. K. 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.

[SS07] K. Siozios and D. Soudris, "A novel methodology for temperature-aware placement and routing of FPGAs," in *Proc. IEEE Computer Society Annual Symposium on VLSI*, 2007, pp. 55–60.

[SSC08] E. Stott, P. Sedcole, and P. Y. K. Cheung, "Fault tolerant methods for reliability in FPGAs," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2008, pp. 415–420.

[SSC09] E. Stott, P. Sedcole, and P. Y. K. Cheung, "Modelling degradation in FPGA lookup tables," in *Proc. International Conference on Field-Programmable Technology (FPT)*, 2009, pp. 443–446.

[Sta16] Statistica, "Global car sales 1990-2016," 2016. Online available: http://www.statista.com/statistics/200002/international-car-sales-since-1990/ (Accessed on 23.08.2016).

[Ste00] A. Steininger, "Testing and built-in self-test — A survey," *Journal of Systems Architecture*, vol. 46, no. 9, pp. 721–747, 2000.

[STW98] K. Seshan, J. M. Timothy, and K. J. Wu, "The quality and reliability of Intel's quarter micron process," *Intel Technology Journal*, 1998.

[STY+15] M. C. Smayling, K. Tsujita, H. Yaegashi, V. Axelrad, R. Nakayama, K. Oyama, S. Yamauchi, H. Ishii, and K. Mikami, "7nm logic optical lithography with OPC-Lite," in *Proc. SPIE Advanced Lithography*, 2015, p. 94261U.

[SVDK14] R. Santos, S. Venkataraman, A. Das, and A. Kumar, "Criticality-aware scrubbing mechanism for SRAM-based FPGAs," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–8.

[SWC10] E. Stott, J. S. Wong, and P. Y. Cheung, "Degradation analysis and mitigation in FPGAs," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2010, pp. 428–433.

[SWHA98] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici, "Built-in self-test of FPGA interconnect," in *Proc. International Test Conference (ITC)*, 1998, pp. 404–411.

[SWS+14] J. H. Stathis, M. Wang, R. G. Southwick, E. Y. Wu, B. Linder, E. G. Liniger, G. Bonilla, and H. Kothari, "Reliability challenges for the 10nm node and beyond," in *Proc. IEEE International Electron Devices Meeting (IEDM)*, 2014, pp. 20.6.1–20.6.4.

[SWSC10] E. A. Stott, J. S. Wong, P. Sedcole, and P. Y. Cheung, "Degradation in FPGAs: Measurement and modelling," in *Proc. 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2010, pp. 229–238.

[SXCT00] X. Sun, J. Xu, B. Chan, and P. Trouborst, "Novel technique for built-in self-test of FPGA interconnects," in *Proc. International Test Conference (ITC)*, 2000, pp. 795–803.

[Tah03] M. B. Tahoori, "Using satisfiability in application-dependent testing of FPGA interconnects," in *Proc. 40th Design Automation Conference (DAC)*, 2003, pp. 678–681.

[Tah06] M. Tahoori, "Application-dependent testing of FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 9, pp. 1024–1033, 2006.

[TCW+05] T. J. Todman, G. A. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, "Reconfigurable computing: architectures and design methods," *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2005.

[TM05] M. B. Tahoori and S. Mitra, "Application-independent testing of FPGA interconnects," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1774–1783, 2005.

[Tri15] S. M. Trimberger, "Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 318–331, 2015.

[TUKT97] T. Tomita, H. Utsunomiya, Y. Kamakura, and K. Taniguchi, "Hot hole induced breakdown of thin silicon dioxide films," *Applied Physics Letters*, vol. 71, no. 25, p. 3664, 1997.

[vAMM+16] V. M. van Santen, H. Amrouch, J. Martin-Martinez, M. Nafria, and J. Henkel, "Designing guardbands for instantaneous aging effects," in *Proc. 53rd Annual Design Automation Conference (DAC)*, 2016, pp. 69:1–69:6.

[van93] A. J. van de Goor, "Using march tests to test SRAMs," *IEEE Design & Test of Computers*, vol. 10, no. 1, pp. 8–14, 1993.

[VHL+05] S. Velusamy, W. Huang, J. Lach, M. Stan, and K. Skadron, "Monitoring temperature in FPGA based SoCs," in *Proc. International Conference on Computer Design (ICCD)*, 2005, pp. 634–637.

[vMMA+17] V. M. van Santen, J. Martin-Martinez, H. Amrouch, M. M. Nafria, and J. Henkel, "Reliability in super- and near-threshold computing: A unified model of RTN, BTI, and PV," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2017.

[VS07] S. Vassiliadis and D. Soudris, Eds., *Fine- and coarse-grain reconfigurable computing.* Springer, 2007.

[vT08] V. von Tils, "Zero defects - Reliability for automotive electronics," in *Proc. International Interconnect Technology Conference (IITC)*, 2008, pp. 1–3.

[WA10] M. A. Watkins and D. H. Albonesi, "ReMAP: A reconfigurable heterogeneous multicore architecture," in *Proc. 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2010, pp. 497–508.

[WBR11] H. Wong, V. Betz, and J. Rose, "Comparing FPGA vs. custom cmos and the impact on processor microarchitecture," in *Proc. 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2011, pp. 5–14.

[WST08] L.-T. Wang, C. E. Stroud, and N. A. Touba, Eds., *System-on-chip test architectures: Nanometer design for testability.* Morgan Kaufmann, 2008.

[WSV09] E. Y. Wu, J. Sune, and R.-P. Vollertsen, "Comprehensive physics-based breakdown model for reliability assessment of oxides with thickness ranging from 1 nm up to 12 nm," in *Proc. IEEE International Reliability Physics Symposium (IRPS)*, 2009, pp. 708–717.

[WTH14] M. J. Wirthlin, H. Takai, and A. Harding, "Soft error rate estimations of the Kintex-7 FPGA within the ATLAS Liquid Argon (LAr) Calorimeter," *Journal of Instrumentation*, vol. 9, no. 01, p. C01025, 2014.

[WWW06] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon).* Morgan Kaufmann, 2006.

[Xil12a] Xilinx, "Constraints Guide," *UG625 (v13.4)*, 2012.

[Xil12b] Xilinx, "Partial Reconfiguration User Guide," *UG702 (v14.1)*, 2012.

[Xil12c] Xilinx, "Virtex-5 FPGA Configuration User Guide," *UG191 (v3.11)*, 2012.

[Xil12d] Xilinx, "Virtex-5 FPGA User Guide," *UG190 (v5.4)*, 2012.

[Xil15a] Xilinx, "UltraScale Architecture Configurable Logic Block User Guide," *UG574 (v1.4)*, 2015.

[Xil15b] Xilinx, "UltraScale Architecture Configuration User Guide," *UG570 (v1.6)*, 2015.

[Xil16a] Xilinx, "7 Series FPGAs Memory Resources User Guide," *UG473 (v1.12)*, 2016.

[Xil16b] Xilinx, "Device Reliability Report," *UG116 (v10.4)*, 2016.

[Xil16c] Xilinx, "Vivado Design Suite User Guide: High-Level Synthesis," *UG902 (v2016.2)*, 2016.

[Yan91] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," 1991.

[YJGR11] S. Yousuf, A. Jacobs, and A. Gordon-Ross, "Partially reconfigurable system-on-

chips for adaptive fault tolerance," in *Proc. International Conference on Field-Programmable Technology (FPT)*, 2011, pp. 1–8.

[YWW+14] L. Yan, B. Wu, Y. Wen, S. Zhang, and T. Chen, "A reconfigurable processor architecture combining multi-core and reconfigurable processing units," *Telecommunication Systems*, vol. 55, no. 3, pp. 333–344, 2014.

[ZBK+13] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, C. Braun, M. E. Imhof, H.-J. Wunderlich, and J. Henkel, "Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures," in *Proc. IEEE International Test Conference (ITC)*, 2013, pp. 1–10.

[Zor13] Y. Zorian, "Test & reliability challenges in advance semiconductor geometries," in *Proc. IEEE Semiconductor Wafer Test Workshop*, 2013.