Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Integrierte Systeme

Redundanzfreie Fehlerbehandlung in echtzeitfähigen FPGA Schaltungen

Dipl.-Ing. Michael Frischke

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines Doktor-Ingenieurs genehmigten Dissertation.

Vorsitzende:		Univ.-Prof. Dr. rer. nat. Doris Schmitt-Landsiedel
Prüfer der Dissertation:	1.	apl. Prof. Dr.-Ing. habil. Walter Stechele
	2.	Univ.-Prof. Dr.-Ing. Georg Sigl

Die Dissertation wurde am 24.04.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 13.10.2015 angenommen.

Kurzdarstellung

In sicherheitsrelevanten oder hochverfügbaren Anwendungen kann je nach Funktionsanforderung flexible Hardware eingesetzt werden. Traditionell werden dazu zusätzliche Redundanzmechanismen zur Fehlerdetektion und ein Reset zur Fehlerkorrektur flüchtiger Fehler verwendet. Für den wettbewerbsfähigen Einsatz flexibler Hardware bei hohem Produktionsvolumen sind jedoch kostengünstige Mechanismen zur Fehlerbehandlung im Bereich von wenigen Millisekunden notwendig. Speziell in echtzeitfähigen Kraftfahrzeug Steuergeräten mit programmierbarer Hardware in Form von *Field Programmable Gate Arrays (FPGAs)* können diese Mechanismen zu einer reduzierten *Mittleren Fehlerdetektions- und Reparaturzeit (MTTR)* und somit höheren Verfügbarkeit beitragen. Hierzu wird die redundanzfreie Fehlerdetektion, Fehlerkorrektur und Echtzeit-Synchronisation innerhalb des FPGAs ohne große Einflüsse auf das Gesamtsystem verbessert und optimiert.

Kurze Synchronisationszeiten mit dem Gesamtsystem sind eine Anforderung zum Erreichen einer hohen Verfügbarkeit der der FPGA Schaltung. Klassische Reset Konzepte erreichen diese Zeiten nicht und können durch Zustandssicherung (Checkpoint) und Zustandswiederherstellung (Rollback) teilweise ersetzt werden. Hierfür wird ein Analyseframework auf Netzlisten- und Verhaltensebene eingeführt. Checkpoint Methoden für sequentielle Schaltungen werden in Bezug auf Zeitanspruch und Mehraufwand optimiert und zyklenakkurat simuliert. Die dafür notwendige Simulationsumgebung wurde auf Basis von SystemC in der Netzlistenebene etabliert und die Simulationsgeschwindigkeit evaluiert. Nach der Bestimmung der dedizierten Checkpoint und Rollback Methode wird die Schaltung durch eine neue Analyseumgebung auf relevante Echtzeitverletzungen untersucht. Mittels diesem teilautomatisierten Framework mit Schnittstelle zur Systemsimulation wird letztlich die Strategie der Zeitpunkte von Checkpoint Erstellungen ermittelt.

Aus den Untersuchungen zu der Verfügbarkeit und der Zustandswiederherstellung von FPGA Schaltungen ergibt sich notwendigerweise eine kurze Fehlerdetektionszeit der Funktionen. Diese Zeit wird bisher durch langsame Tests des fehlersensitiven Konfigurationsspeichers dominiert. Durch den redundanzfreien Ansatz des *Priorisierten Konfigurationsspeicher Tests (PKT)* kann diese Fehlerdetektionszeit von flüchtigen und permanenten Fehlern im Konfigurationsspeicher häufig deutlich verkürzt werden. Hierfür werden auf Error-Correcting Code (ECC) basierende Testmethoden mittels zusätzlichen Datenfluss- und Platzierungsinformationen verfeinert. Das daraus folgende Scheduling von Tests in partiellen Speicherbereichen reduziert die Testzeit einzelner programmierter Hardwarefunktion.

Sowohl die optimierte Checkpoint Methodik als auch der verfeinerte Konfigurationstest werden anhand von echtzeitfähigen Schaltungen evaluiert. Die Ergebnisse zeigen eine kürzere Testzeit nach Fehlern im Konfigurationsspeicher sowie eine kürzere Korrekturzeit für flüchtige Fehler im Zustandsspeicher der Schaltung. Beide Resultate führen zu einer kürzeren MTTR und somit erhöhten Verfügbarkeit der FPGA Schaltung im Vergleich zu klassischen Testmethoden mit anschließendem Reset. Gleichzeitig wird die Verwendung der entwickelten Methoden im industriellen Umfeld durch eine kurze Anwendungszeit in der Produktentwicklung gezielt unterstützt.

Abstract

Nowadays, flexible hardware is applicable in a wide range of safety relevant and high-available applications. In order to meet the requirements in terms of reliability we often use redundancy for error detection and reset concepts for corrections of soft errors. Yet, to become more competitive on high-volume markets, the approaches for error handling at configurable hardware have to be more cost-efficient without lack in speed. Focusing on real-time electronic control units the required approaches shall reduce the Mean Time To Repair (MTTR) in commercial FPGAs. Hence, this work aims on a reduced MTTR by redundancy-free error detection, correction and synchronization within the FPGA. The methods shall comply with the real-time property of the embedded system.

Firstly, shortening the time for global synchronization after an error is a major issue because state of the art reset concepts lack in a synchronization manner. Instead, checkpoint and rollback approaches are analyzed by a novel framework dealing with netlist- and system behavior. This work does optimize and simulate checkpoint measures in sequential circuits with focus on little overhead in time and hardware. A System-C based netlist simulation is therefore introduced and evaluated. Further, a novel behavior analysis tool based on interaction monitoring reveals the missed real-time deadlines. Finally, a strategy for times gathering the checkpoints is extracted from the easy-to-use frameworks.

Upon the investigations of the increased availability via checkpoint and rollback, a short fault detection time (FDT) is mandatory. Until now, the FDT is kept high due to slow tests of the vulnerable configuration memory in the FPGA. By introducing the fast prioritized configuration memory test the FDT is reduced significantly. Together with data flow and layout information mature error correcting codes are used. Memory areas are scheduled into a test sequence in order to reduce the test time of dedicated hardware functions.

Either the optimized checkpoint approach or the refined test of the configuration memory are evaluated with real-time circuits. Almost all results show a shorter FDT for the memory as well as a very short error correction time after soft errors. The results propose a short a MTTR and, in fact, they increase the availability compared to state of the art redundancy-free fault detection and reset methods. Finally, all presented tools are intended to provide a fast product development phase by easy-to-use handling.

Danksagung

Diese Arbeit entstand während meiner Tätigkeit bei der Robert Bosch GmbH in Zusammenarbeit mit dem Lehrstuhl für Integrierte Systeme der Technischen Universität München. Somit geht mein besonderer Dank an Prof. Dr.-Ing. Walter Stechele, der diese Arbeit ermöglichte und mich in vielen fachlichen Diskussionen inspiriert und gefordert hat. Ein weiterer Dank gilt Prof. Dr.-Ing. Sigl für sein wertvolles Feedback.

Seitens der Robert Bosch GmbH möchte ich mich zuerst bei Andreas Jürgen Rohatschek für seine Unterstützung, Ratschläge und auch seine Geduld bedanken. Ferner danke ich allen Mitarbeitern und Freunden der Forschungsgruppe für digitale Hardware, insbesondere auch Jürgen Schirmer für seine Unterstützung. Für die besondere und kreative Stimmung im Team danke ich zudem René Guillaume, Simon Roth, Dr. Simon Hufnagel, Dr. Hendrik Post, Dr. Tobias Kirchner, Dr. Nico Bannow, Dieter Thoss und Stoyan Todorov.

Ein großer Dank geht an meine Eltern und an meine Schwester, die mich großartig unterstützt haben.

Zuletzt schulde ich vor allem meiner Frau Kerstin größten Respekt und Dank für ihr Verständnis, ihre Unterstützung und Liebe während der letzten Jahre.

Für Elisa

„Drei Dinge sind uns aus dem Paradies geblieben:
die Sterne der Nacht, die Blumen des Tages und die Augen der Kinder“
(Dante Alighieri)

Inhaltsverzeichnis

Kurzdarstellung **iii**

Abstract **v**

Inhaltsverzeichnis **ix**

1. Einleitung und Ziel **1**

1.1. Motivation 2

1.2. Ziele und Beiträge 2

1.3. Gliederung der Arbeit 4

2. Aufbau, Fehler und Wiederherstellung von programmierbaren Logikschaltungen **5**

2.1. Grundlagen des Schaltungsdesigns mit FPGAs 6

2.1.1. Architekturen und Eigenschaften 6

2.1.2. Entwicklungsablauf 10

2.2. Überblick der Ausfallsicherheit des FPGA auf Logikebene 11

2.2.1. Defektquellen und Fehlerausbreitung 11

2.2.2. Fehlertoleranz zur Reduzierung der Fehlerauswirkung 14

2.3. Beschreibung von flüchtigen Fehlern im Konfigurationsspeicher 16

2.3.1. Fehlerwahrscheinlichkeiten für flüchtige Fehler 17

2.3.2. Funktionale Fehlermodelle 18

2.3.3. Detektions- und Korrekturmaßnahmen 19

2.4. Analyse von Methoden zur Zustandswiederherstellung 22

2.4.1. Varianten des Checkpoint und Rollback 22

2.4.2. Methoden zur Sicherung des Zustandsvektors in sequentiellen Schaltungen 24

2.4.3. Anwendungsdomänen von Methoden der Zustandswiederherstellung 25

3. Redundanzfreie Methoden für verfügbare und sicherheitsrelevante Echtzeit-Designs **27**

3.1. Abgeleitete Anforderungen an die Fehlerdetektion der Konfiguration gemäß ISO 26262 28

3.1.1. Kurzdarstellung Standard ISO 26262 mit FPGA 28

3.1.2. Beurteilung der CRAM-Detektionsmethoden unter ISO 26262 29

3.2. Beschreibung von Verfügbarkeit in FPGA Systemen . . . 32
3.2.1. Metrik und Bestimmungen der Verfügbarkeit . . . 32
3.2.2. Beitrag der Fehlerdetektion und Zustandswiederherstellung auf die Verfügbarkeit . . . 34
3.3. Beschränkungen durch validierten Entwicklungsfluss . . . 34

4. Echtzeitfähige Zustandswiederherstellung in sequentiellen Schaltungen 37
4.1. Grundlegende Auslegung der Checkpoint Virtualisierung . . . 38
4.1.1. Größe des Zustandsvektors . . . 38
4.1.2. Schaltungs- und Betrachtungsgrenzen . . . 39
4.2. Checkpoint-Analyse auf der Netzliste . . . 41
4.2.1. Modellierung von Methoden und Schaltungen . . . 41
4.2.2. Simulation der applizierten Methoden . . . 42
4.2.3. Analyse der applizierten Methoden . . . 44
4.2.4. Minimierung der Methoden-Auswirkungen . . . 46
4.3. Dynamische Analyse der Schaltung auf der Ebene von Register Transfer Level . . . 48
4.3.1. Checkpoint Gültigkeiten . . . 48
4.3.2. SystemC basierte Evaluation der Interaktionen . . . 51
4.4. Analysetool und Anwendungsergebnisse für ein Echtzeitsystem . . . 54
4.4.1. Anwendung 1: Microcontroller System . . . 54
4.4.2. Anwendung 2: Echtzeit Timermodul . . . 57
4.5. Zusammenfassung und Erweiterungen . . . 64

5. Priorisierte und datenflussorientierte Fehlerdetektion im Konfigurationsspeicher 67
5.1. Grundlegende Auslegung . . . 68
5.1.1. Priorisierter Konfigurationstest . . . 68
5.1.2. Datenfluss-Varianten . . . 69
5.1.3. Partitionierung von Testbereichen . . . 70
5.2. Datenflussorientierte Schaltungsanalyse und Erstellung der Testbereiche . . . 70
5.2.1. Korrelation der Funktionen zum Layout und zu Testregionen . . . 71
5.2.2. Abdeckung der CRAM Fehlermodelle . . . 75
5.3. Zeitgesteuerte Testverteilung . . . 77
5.3.1. Gruppierung der Testanforderung und Verteilung auf Testslots . . . 79
5.3.2. EDF Scheduling . . . 79
5.3.3. Scheduling mit SAT . . . 80
5.4. Homogene Testverteilung . . . 82
5.4.1. Verteilungsalgorithmus . . . 83
5.4.2. Optimierung der Testzeit . . . 85

5.4.3. Abhängige Testzeit bei gemeinsamen Ressourcen . . . 88
5.5. Realisierung des Testsequenzers im FPGA . . . 89
5.5.1. Detektions- und Korrektureinheit . . . 90
5.5.2. Speicherbedarf . . . 91
5.5.3. Empirischer Nachweis der PKT Fehlerabdeckung . . . 91
5.6. Anwendungsergebnisse für verteilte und einzelne FPGA Funktionen . . . 93
5.6.1. Dual Microcontroller Schaltung . . . 94
5.6.2. Redundanzschaltung mit Überprüfer . . . 99
5.6.3. Echtzeit Timermodul . . . 100
5.7. Zusammenfassung und Erweiterungen . . . 104

6. Zusammenfassung und Ausblick **107**
6.1. Zusammenfassung und Diskussion . . . 107
6.2. Ausblick . . . 108

A. Anhang **111**
A.1. Code Templates . . . 112
A.2. Design Dateien . . . 113
A.2.1. EDIF Netzlistenformat . . . 113
A.2.2. XDL Platzierungsformat . . . 114
A.3. Checkpoint und Rollback . . . 115
A.3.1. Netzlistenrepräsentation . . . 115
A.3.2. Ergebnisse Fallbeispiel MicroBlaze MCS . . . 116
A.4. Priorisierter Konfigurationstest . . . 117
A.4.1. Kombinatorische Testbereiche . . . 117
A.4.2. Herleitung der mittleren Testdauer eines Major Frames der Xilinx 7 Serie . . . 117
A.4.3. Aufbau der Frameadressen der Xilinx 7er Serie . . . 118
A.4.4. Frame auslesen mittels ICAP . . . 119
A.4.5. Frame schreiben mittels ICAP . . . 120
A.4.6. Detaillierte Ergebnisse der homogenen Testverteilung für Dual Core Microcontroller . 123
A.4.7. Detaillierte Ergebnisse der homogenen Testverteilung für den Echtzeit Timer . . . 124

Abbildungsverzeichnis **125**

Literaturverzeichnis **127**

Abkürzungsverzeichnis **139**

1. Einleitung und Ziel

Im Bereich der digitalen Elektronik stellte die Erfindung des Field Programmable Gate Array (FPGA) in den 1980er einen enormen Fortschritt dar. Erstmals wurden programmierbare Speicherzellen Programmable Read Only Memory (PROM) mit veränderbaren Verdrahtungen (engl. Routing) verknüpft [Fre89] und mit dem ersten FPGA XC2064 kommerziell vertrieben. Die Firma Xilinx startete damit 1985 einen Markt, der heute einen Umsatz von über 4 Milliarden Dollar hat [Sou13]. Die technische Entwicklung schritt rasant voran, getrieben von den schrumpfenden Strukturbreiten in Halbleitern. Ende der 1980er und Anfang der 1990er Jahren führten nach und nach mehrere Firmen Integrated Circuit (IC)s mit programmierbarer Logik ein, darunter Actel und Altera. Im Jahre 2012 teilen sich Xilinx und Altera einen Marktanteil von ca. 90 % [Sou13]. Während FPGAs im Bereich der Telekommunikation und Netzwerktechnik reiften, sind heute weitere Domänen wie Luft- und Raumfahrt, Automotive, Medizintechnik, Audio- und Videoverarbeitung abgedeckt. In vielen Applikationen kann ein FPGA als technische Alternative zu einer Graphics Processing Unit (GPU), Central Processing Unit (CPU) [CWFH12] oder Application Specific Integrated Circuit (ASIC) [Kot06] eingesetzt werden.

Im Automotive Umfeld hat der FPGA erst in den 2000er Einzug gehalten [GNW+06]. Bereits im Jahr 2007 sind 18 Xilinx FPGAs in einer vollausgestatteten S-Klasse enthalten [Mar07]. Die Applikationen sind zu dieser Zeit auf Komfortfunktionen wie Fensterheber bis zu Fahrerassistenz Systemen [Hau08] beschränkt. Zugleich steigen der Anteil und die Kosten der Elektronik im Fahrzeug immer weiter an [Kru08]. Es ist absehbar, dass durch zukünftige Systemanforderungen in einem Kfz Steuergerät der Anteil der FPGAs zunehmen wird, auch für sicherheitsrelevante Bereiche. Zum einen muss die Zeit bis zur Markteinführung eines Steuergerätes (*time-to-market*) von ein bis drei Jahren reduziert werden, um wettbewerbsfähig zu bleiben. Zum anderen sind leistungsfähige Verarbeitungseinheiten notwendig, sowohl für teil-automatisiertes Fahren als auch für voll-automatisiertes Fahren. Gepaart mit Flexibilität in der Entwicklung und durch Reduktion der Variantenvielfalt ergibt sich ein breites Anwendungsfeld für aktuelle und zukünftige FPGA Generationen.

In der Zukunft ist absehbar, dass auf einem FPGA implementierte Funktionen teilweise vital für eine Applikation werden. Beispielsweise können Funktionen wie Sensor-Daten-Fusion oder Gateway Funktionalitäten sicherheitsrelevante Informationen in Echtzeit bereitstellen. Dabei müssen sowohl der enthaltene FPGA als auch weitere Microprocessor Control Unit (MCU) die funktionalen Sicherheitsanforderungen und die von der Qualitätssicherung geforderte Verfügbarkeit erfüllen. In ersten Systemansätzen wird der FPGA als Überwacher-IC oder physikalisch doppelt ausgelegt, um die entsprechende Sicherheitseinstufung zu erhalten [BBD+08, Chu02]. In dieser Arbeit sollen Methoden erarbeitet werden, welche die Systemrelevanz der FPGA-Schaltung durch erhöhte Verfügbarkeit mittels redundanzfreier Lösungen erhöhen können.

1.1. Motivation

Für den Einsatz von FPGAs in automobilen Echtzeitanwendungen sprechen mehrere Gründe. Während der Entwicklung profitieren die Anwender von schnellen Prototypen und kürzerer Zeit bis zur Markteinführung. Zudem bietet die Eigenschaft der Parallelisierung von Hardware (HW) Aufgaben eine reduzierte Latenzzeit [CWFH12]. Infolge der Reprogrammierbarkeit lassen sich Fehler im Feld gut korrigieren. Aus betriebswirtschaftlichen Gründen werden FPGAs zur Reduzierung von Aufbauvarianten eines Produktes eingesetzt. Ferner ergibt sich eine Unabhängigkeit von Spezialhardware durch die Möglichkeit, dedizierte Intellectual Property (IP)s oder HW Beschleuniger zu implementieren. Im Jahr 2012 sind für Strukturgrößen von 28 nm Maskenkosten von zwei bis drei Millionen US-Dollar [Cad12] einzuplanen. Diese Einmalkosten (Non Recurring Engineering (NRE)) müssen für dedizierte Hardware eingeplant werden, sofern nicht frei programmierbar Hardware eingesetzt wird. Somit ist der FPGA unter Randbedingungen wie Stückkosten bei mittleren Volumen von 1.000 bis 100.000 Stück und kleinen NRE [Kot06]eine sinnvolle Alternative.

In dieser Arbeit soll durch kostengünstige Lösungen zur Fehlerdetektion und Fehlerkorrektur der Eintritt in hochvolumige automobile Anwendung (*Automotive* Anwendungen) erleichtert werden. Eine technische Voraussetzung für den Einsatz eines FPGAs in einem *echtzeitfähigen* und *sicherheitsrelevanten* Bereich der Fahrzeugelektronik ist die funktionale Sicherheit. Der für die Automobilindustrie geltende Standard im Sinne der funktionalen Sicherheit in Kraftfahrzeugen ist die Norm ISO 26262 der International Organization for Standardization [ISO11a]. In dieser Norm stellt die Fehlertoleranz einen Aspekt dar, die sich aus Fehlerdetektion und Erreichen des sicheren Zustandes zusammensetzt. Die Vorgaben für die Fehlerdetektion zielen darauf ab, die Fehlerausbreitung im System lokal begrenzt zu halten und somit die für den Fahrer spürbaren Auswirkungen zu unterbinden. Diese Zeit der Fehlerbehandlung kann von mehreren 100 ms bei einer Motorsteuerung bis zu 1 ms für das Antiblockiersystem (ABS) oder für die elektrische Lenkung reichen [Ren12]. Somit ist für den FPGA die schnelle Fehlerdetektion unter akzeptabler Fehlerabdeckung eine Bedingung für den Einsatz in heutigen echtzeitfähigen und sicherheitsrelevanten Bereichen des Automobils.

Ein weiteres Qualitätsmerkmal eines (FPGA-basierten) Steuergerätes ist eine hohe Verfügbarkeit trotz flüchtiger und nichtflüchtiger Speicherfehler. Eine hohe Verfügbarkeit meidet Rückläufer aus dem Feld und verhindert unangenehme oder spürbare Leerlaufzeiten des Systems. Nur wenn Fehler im System toleriert oder schnell behoben werden, ist eine hohe Verfügbarkeit denkbar. Da Toleranz in den meisten Fällen eine Kostenerhöhung in Form von Redundanzen bedeutet, ist eine schnelle Fehlerreaktion und Behebung unterhalb der Wahrnehmschwelle oder innerhalb der Fehlertoleranzzeit erstrebenswert. Zunehmend komplexe Systeme und Schaltungen verursachen nach einer Fehlerdetektion eine nicht zu unterschätzende Ausfallzeit durch einen Reset. Daher sind Konzepte notwendig, welche eine FPGA Schaltung sowohl während des Hochlaufs als auch nach einer Detektion von Zustands- und Konfigurationsspeicherfehlern innerhalb kürzester Zeit wieder bereitstellen und mit der Umgebung synchronisieren.

1.2. Ziele und Beiträge

Mit den in dieser Arbeit vorgestellten Methoden soll die mittlere Fehlerdetektions- und Reparaturzeit (Mean Time To Repair (MTTR)) einer echtzeitfähigen eingebetteten FPGA Schaltung merklich reduziert werden. Durch eine reduzierte MTTR wird die Verfügbarkeit im Sinne der Produktqualität erhöht. Zudem können bei

verkürzter MTTR komplexere Anwendungen die Anforderungen an die *Fehlertoleranzzeit* nach International Organization for Standardization (ISO) 26262 [ISO11a] erfüllen.

Als konkrete Maßnahmen zur Reduzierung der MTTR wird zum einen die kostengünstige und schnelle Fehlerdetektion im Konfigurationsspeicher untersucht und implementiert. Zum anderen wird eine redundanzfreie Methode zur Zustandswiederherstellung und somit zur Fehlerkorrektur auf Basis von Checkpoint und Rollback präsentiert. Mit diesen zwei Ansätzen lassen sich bestehende Konzepte für „fail safe“ FPGA Steuergeräte von Chujo et al. (2002) [Chu02] oder Beckschulze et al. (2008) [BBD+08] auf komplexe FPGA Anwendungen erweitern. Ferner müssen die Methoden sowohl industriell und kostengünstig nutzbar als auch normkonform einsetzbar sein.

Für die Fehlerdetektion im Konfigurationsspeicher werden folgende Beiträge präsentiert.

1. Verknüpfung von Teilbereichen des Konfigurationsspeichers mit dedizierten Funktionen der Anwendung

2. Gewichtung und Priorisierung von Teilbereichen des Konfigurationsspeichers nach der Anzahl der verwendeten Bits. Mit dieser Gewichtung lassen sich Testsequenzen homogener erstellen.

3. Anwendung von Scheduling Verfahren auf den Testablauf im Konfigurationsspeicher innerhalb von FPGAs.

4. Anpassung der Testdauer einer Funktionalität an deren Kritikalität. Dazu sind Testanforderungen und Speicherregionen zu einem Testablauf adaptiert.

5. Nachweis der Fehlerabdeckung des partiellen Konfigurationstests.

Nach der Fehlerdetektion soll die Fehlerbehebung von flüchtigen Fehlern ohne die Verwendung eines klassischen Schaltungsreset oder der Verwendung von redundanten Modulen durchgeführt werden. Es werden Konzepte für Zustandssicherung (Checkpoint) und Zustandswiederherstellung (Rollback) untersucht und an der Echtzeitfähigkeit gemessen. Gegebene Schaltungen sollen auf diese Konzepte untersuchbar sein und der Entwickler soll die Auswirkungen auf die Schaltung und die Applikation durch folgende Beiträge einschätzen können.

1. Schnelle und teil-automatisierte Beurteilung von Methoden der Zustandswiederherstellung durch hybride Schaltungsanalyse und Verhaltenssimulation.

2. Anwendung und Optimierung von Methoden zur Zustandswiederherstellung verschiedener Abstraktionsebenen auf ein Netzlistenmodell.

3. Erstellung von Anwendungskriterien und die Etablierung von interaktionsabhängigen Gültigskeitsmetriken für unterschiedliche Szenarien mit Zustandswiederherstellung in echtzeitfähigen Schaltungen.

4. SystemC-basierter Nachweis der Checkpoint Strategie mit anschließenden Deadline Analysen und Anwendungs-Performance Einschätzungen.

Mit Hilfe dieser Maßnahmen ist der Schaltungsentwickler in der Lage, effiziente Methoden und Strategien zur Zustandswiederherstellung durch Checkpoint und Rollback zu finden. Es werden die dafür notwendigen Schritte und Anwendungskriterien abgeleitet.

1.3. Gliederung der Arbeit

Im folgenden Kapitel wird ein Überblick über den generellen Aufbau eines FPGA und dessen Fehlerbilder gegeben. Darauf aufbauend zeigt dieses Kapitel dedizierte Testmethoden für den Konfigurationsspeicher sowie Möglichkeiten zur Fehlerkorrektur flüchtiger Fehler auf. Kapitel 3 zeigt nun die Herausforderungen eines FPGA Designs mit Hinblick auf die in der Automobilbranche angewendete ISO 26262. Ferner wird die Verfügbarkeit von FPGA Systemen eingeführt und die Einflüsse einer schnellen Fehlerdetektion und Fehlerkorrektur herausgestellt. Ein häufig unterschätzter Aspekt des normgerechten und validierten Entwicklungsflusses wird hier ebenfalls erläutert.

Das für industrielle Zwecke einsetzbare **C**heckpoint **a**nd **R**ollback Eva**l**uation, **O**ptimization, **S**imulation (CaRLOS) Framework zur Beurteilung von Checkpoint und Rollback Methoden in echtzeitfähigen FPGA Schaltungen wird in Kapitel 4 vorgestellt. Hier werden die drängenden Fragen bei einer rückwärts gerichteten Fehlerkorrektur herausgestellt und aggregiert durch Analysen und Simulationen beantwortet. Das Kapitel wird mit zwei Beispielanwendungen abgeschlossen.

Anschließend wird der grundsätzlich neue Ansatz des Priorisierten Konfigurationsspeicher Tests vorgestellt (PKT). Anstelle von linearen Tests wird nun eine zeitliche Abfolge in den ECC-basierten Test von Bereichen des Konfigurationsspeichers gebracht. Es wird der theoretische Ansatz, die Umsetzung und die Anwendung präsentiert.

Die Arbeit schließt in Kapitel 6 mit einer Zusammenfassung des CaRLOS Framework und des PKT inklusive einem kurzen Ausblick.

2. Aufbau, Fehler und Wiederherstellung von programmierbaren Logikschaltungen

Dieses Kapitel befasst sich mit der Einführung der programmierbaren Logik, speziell des FPGAs. Wie bereits dargestellt, eignet sich ein FPGA vor allem für parallele Datenverarbeitung [CWFH12]. Zudem sind schnelle HW Prototypen (*rapid prototyping*) und Serienprodukte mit volatilen Anforderungen während und nach der Entwicklungszeit möglich. Ein Beispiel für diese Kleinserien mit wechselnden Kundenanforderungen ist die Motorsteuerung für Rennfahrzeuge [Bos13]. Hier werden FPGAs seit 2008 verwendet, um die Leistungsfähigkeit und Flexibilität zu erhalten, die ein einzelner Microcontroller nicht bietet.

Im Gegensatz zu grob granularen programmierbaren Strukturen (engl. coarse grain) nach Hartenstein (2001) [Har01] muss bei der FPGA Anwendung der Datenfluss nicht bereits mit Bitbreite und Rechenoperationen spezifiziert sein. Daher liegt, trotz des größeren Flächen- und Verdrahtungsbedarfs, der Fokus dieser Arbeit auf programmierbaren fein granularen (engl. fine grain) Logikstrukturen.

Aktuelle Static Random Access Memory (SRAM) basierende FPGA Generationen werden mit dem 28 nm Halbleiter Technologieknoten in Serie gefertigt. Der damit verbundene Vorsprung in der Leistungsfähigkeit gegenüber einem vor mehreren Jahren entwickelten ASIC führt zu einem Preisverfall pro Gatter, der den FPGA auch für zeitnahe Serienprodukte mittleren oder hohen Volumens attraktiv machen kann. Durch die Flexibilität der Schaltung mittels Programmieren des Bausteins eröffnen sich auch für sicherheitsrelevante Anwendungen vielfältige Optionen. Der Ausbau und die Erweiterung von Sicherheitskonzepten für eine Schaltung ist ebenso schnell wie die Integration dedizierter IPs möglich.

Die Flexibilität wird mit einem Überschuss an verwendeter Si-Fläche gewonnen. Die regulären Strukturen und konfigurierbaren Verbindungspunkte (*routing*) verursachen eine ca. 20 bis 30 fach größere IC Fläche gegenüber einem kundenspezifischen ASIC [WBR11]. Durch die schaltbaren Verbindungsstrukturen vergrößern sich ferner die Signallaufzeiten (*delay*) um das etwa 10 bis 20 fache [KR09]. Zudem steigt die dynamische Leistungsaufnahme des FPGAs auf das 5 bis 10 fache im Vergleich zu einem ASIC. Diese Daten sind für Technologieknoten von 60 nm mit Xilinx ICs und für 90 nm mit Altera ICs erhoben.

Das Kapitel erläutert die Grundlagen der FPGA Technologie und des Schaltungsdesigns. Danach werden die wichtigsten Fehlermodelle und deren Einfluss auf die Verfügbarkeit angeführt. Neben Fehlerquellen und Fehlerarten werden auch Gegenmaßnahmen erläutert, die eine funktionale Sicherheit gewährleisten und die Verfügbarkeit erhöhen. Hierbei liegt der Fokus auf dem Konfigurationsspeicher und der Zustandswiederherstellung.

2.1. Grundlagen des Schaltungsdesigns mit FPGAs

Moderne programmierbare Logik kann mit drei Technologien konfiguriert werden: *SRAM*, *Flash* und *Antifuse* [KTR08]. Eine Konfiguration mittels Antifuse ist einmal programmierbar durch das Durchbrennen einer Isolationsschicht. Sie bietet zwar eine große Packungsdichte gegenüber SRAM Technologien gleicher Strukturbreite, wird aber aufgrund der Strukturbreiten über 100 nm und der fehlenden Flexibilität in dieser Arbeit nicht weiter betrachtet. Im Vergleich zur SRAM Technologie ist die Konfiguration mittels Flash Zellen bei einem Verlust der Versorgungsspannung nicht flüchtig und hat eine geringere statische Leistungsaufnahme [Dri11]. Dagegen steht der um mehrere Generationen zurückliegende Technologieknoten von aktuell 65 nm für Flash basierende FPGAs [Mic13] im Gegensatz zu 28 nm bei SRAM basierenden FPGAs [Xil14], [Alt14]. Dadurch verringert sich die Leistungsfähigkeit der Flash FPGAs, was sich in reduzierter Taktfrequenz und reduzierter Anzahl von Konfigurationsblöcken zeigt. Ferner haben diese IC höhere Stückkosten. Der Fokus dieser Arbeit liegt somit auf SRAM basierten FPGAs.

2.1.1. Architekturen und Eigenschaften

Am Grundaufbau von FPGAs hat sich von den 1990er bis heute wenig geändert. Für eine beliebige sequentielle Schaltung werden kombinatorische Logikelemente, Zustandsspeicher und Verbindungen zwischen den Elementen benötigt. Um diese Komponenten flexibel nutzen zu können, gibt es drei Standard-Aufbauvarianten: *island-style*, *row-based* und *hierarchical* [BRM99] [FMM12]. Bei allen Aufbauvarianten bilden konfigurierbare Eingangs- / Ausgangsblöcke (Input / Output Block, IOB) den Randbereich. Intern liegt ein regelmäßiges Gitter der Leitungs- und Verdrahtungsebene zwischen den eigentlichen Logikblöcken, genannt *Routing* oder *Interconnect*. Je nach Aufbauvariante unterscheidet sich der Aufwand für die Verdrahtungsebene. Frühere Bausteine der Firma Altera waren mit hierarchischer Architektur verfügbar. Diese setzte sich aber aufgrund des erhöhten Leitungs- und Verbindungsbedarfs, bestehend aus globalen Verbindungen und abgeschotteten lokalen Verbindungen, nicht durch. Bei der *row-based* Architektur lassen sich Logikblöcke innerhalb einer Reihe bzw. Spalte verbinden. Die Verdrahtungsebene wird reduziert, bietet jedoch nicht die volle Flexibilität bei großen Schaltungen. Diese Architektur findet sich bei FPGAs mittlerer Leistungsfähigkeit und Größe wieder [Mic13].

2.1.1.1. Aktuelle Island-Based Architektur

Aktuell verfügbare FPGAs der Marktführer Altera und Xilinx verfolgen die *island* Architektur [Xil13a] [Alt14] nach Abbildung 2.1. Die Logikblöcke lassen sich sehr regulär anordnen und die Bausteinfamilien sind somit skalierbar. Zudem vereint das Verbindungsgitter lokale, lange und globale Leitungen. Mit lokalen Leitungen lassen sich Logikblöcke verknüpfen, die nah beieinander liegen. Weiter entfernte Blöcke über mehrere Gitterabstände hinweg werden über lange Leitungen verbunden. Je nach FPGA Hersteller variiert die Länge und Anzahl der lokalen und langen Leitungen. Taktsignale werden über globale Leitungen verteilt, getrieben von speziellen Treibern mit hohem Lastfaktor (*Fan-Out*). Die Flexibilität in den Verbindungen beruht seit den 1980er auf dem Prinzip der konfigurierbaren Verbindungspunkte [Fre89]. Die Art der Verbindungspunkte werden *Programmable Interconnect Point (PIP)* genannt und sind sowohl bei FPGAs von Altera als auch von Xilinx zu finden, bei denen mittels Pass-Transistoren Leitungen verknüpft werden. Die Bit-Information

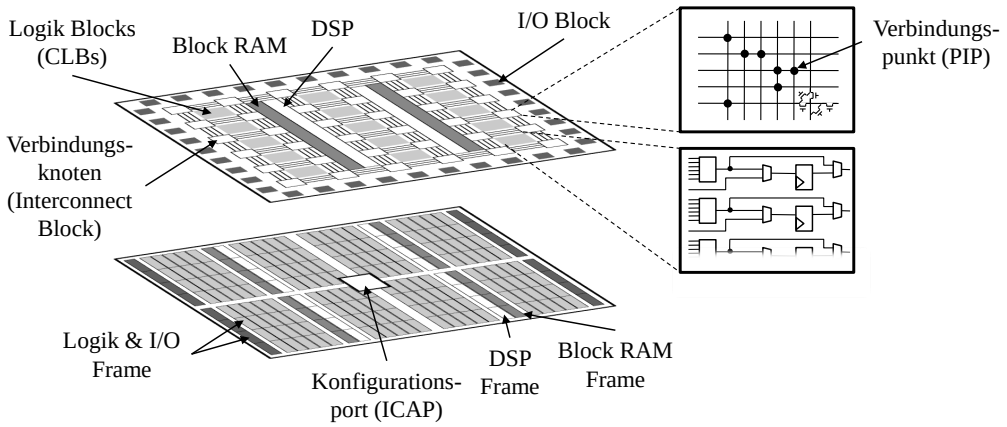


Abbildung 2.1.: Grundstruktur der Island-Based Architektur mit Konfigurationsspeicher

über einen hochohmigen (offen) oder niederohmigen (geschlossen) PIP liegt im Konfigurationsspeicher. Die PIPs werden in dem Gitter an den Verbindungsknoten gebündelt und stellen jeweils einen programmierbaren Interconnect Block dar. Die Leitungen und PIPs beanspruchen etwa 80 % bis 90 % der FPGA Si-Fläche [FMM12].

Das Verbindungsgitter besteht aus multiplen Leitungsarten. Zunächst werden Leitungen für Logik, Takt und statische Signale wie VCC und Masse (GND) unterschieden. Gemäß Abbildung 2.2 sind alle Leitungen von Logiksignalen Punkt-zu-Punkt Verbindungen zwischen Verbindungsknoten oder Logikblöcken, jedoch mit unterschiedlicher Länge. Dagegen verbreiten sich die Taktleitungen als verteiltes Netz mit wenigen globalen Treibern und mehreren Senken. Die Anbindung an eine Instanz erfolgt hier über die Verbindungsknoten. Die statischen Signale VCC und GND sind ebenfalls verteilte Netze, jedoch weitaus lokaler begrenzt. Die lokalen Grenzen resultieren aus den verteilten Quellen für die statischen Signale, wohingegen das Clock Netz nur wenige Quellen aber viele Treiber hat.

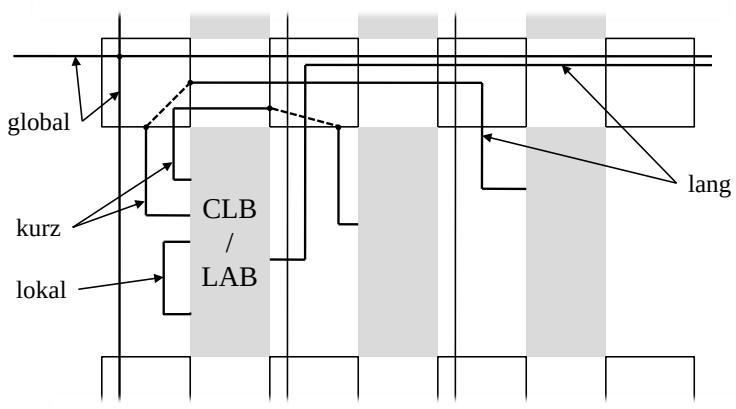


Abbildung 2.2.: Typische Leitungsarten zur Verbindung von Logikblöcken

Das Zentrum in den Zellen des Verbindungsgitters bilden die Logikblöcke. Je nach FPGA Anbieter werden diese Logikblöcke Configurable Logic Block (CLB) [Xil13a] oder Adaptive Logic Module (ALM) [Alt14] bezeichnet und basieren auf einer Grundkomposition aus dem Jahr 1984 [Car89]. Innerhalb der Logikblöcke decken die Lookup Table (LUT) beinahe vollständig alle kombinatorischen Anteile einer Schaltung ab. Eine LUT spiegelt die logische Wahrheitstabelle als Multiplexerstruktur wider, wobei die Konfiguration die Wahrheitswerte jeder möglichen Eingangskombination enthält. Aktuelle FPGA Generationen halten mehrere LUTs innerhalb der Logikblöcke vor, wobei jede LUT bis zu 6 Eingänge verarbeiten kann. Ferner

sind 4 bis 8 Zustandsspeicher Flip-Flop (FF) und dedizierte Elemente für mathematische Operationen, z.B. Übertragsketten (Carry Chain) oder Volladdierer, pro Logikblock enthalten. Der Konfigurationsspeicher einer LUT kann vielmals auch als Arbeitsspeicher konfiguriert und verwendet werden und wird in diesem Fall als LUT-Random Access Memory (RAM) oder Distributed RAM bezeichnet. Optional kann anstelle eines LUT-RAM auch ein Schieberegister in einem Logikblock instanziiert werden, bestehend aus den 32 Bit einer 6-Eingänge-LUT und dem Übertrag zur nächsten LUT .

Im Zuge der kleiner werdenden Transistorstrukturen und der daraus resultierenden Verfügbarkeit von Siliziumfläche im Die wurden fest implementierte IP Blöcke Bestandteil in FPGAs. Diese fest strukturierten und platzierten Blöcke werden als *Hard-IP* oder *Hard-Macro* bezeichnet. Standardmäßig werden Digital Signal Processor (DSP) Einheiten und Block Random Access Memory (BRAM) Blöcke in aktuellen FPGAs implementiert. Zurzeit stehen pro BRAM Block 20 Kb [Alt14] oder 36 Kb [Xil14] zur Verfügung, wobei die Anzahl von der FPGA Familie und Größe abhängt. Diese Blöcke lassen sich als Single-port, Simple dual-port, True dual-port, Schieberegister, Read Only Memory (ROM) oder First In - First Out (FIFO) konfigurieren und nutzen. Neben 48x48 DSP Einheiten stehen zum Teil noch einige weitere Hard-Macros zur Verfügung, wie Ethernet-IP, PCIe, Serializer-Deserializer, High-Speed Transceiver, Konfigurationsports, ECC Macros, AES-256 Verschlüsselungsmacros und Weitere.

2.1.1.2. Konfiguration und Schnittstellen zur Technologie

Wie in Abschnitt 2.1.1.1 gezeigt, beruht der Unterschied zu Standardzellen ICs in der Konfigurationsebene. Nach Abbildung 2.1 besteht der Configuration Random Access Memory (CRAM) aus Frames unterschiedlicher Art. Ein Frame ist ein Teilbereich des CRAM mit eigener Adressierung und lokalen Einstellungen. Jeder Frame speichert die Konfigurationsbits eines Logiktypes: CLB, DSP, BRAM, I/O, globaler Clock (GCLK) oder weiterer FPGA Einstellungen. Je nach FPGA Typ hat der Konfigurationsspeicher eine Größe von ca. 2 MB bis 37 MB mit mehreren tausend Frames [Xil13b]. Bei aktuellen Xilinx FPGAs werden die Frames als “Minor Frame” bezeichnet und haben eine Größe 101 Wörter x 32 bit. Je nach Bedeutung werden 30 bis 42 Minor Frames zu einer Spalte zusammengefasst. In der Literatur wird diese Spalte als *Major Frame (MF)* bezeichnet.

Bei flüchtigen SRAM-basierten Speichern muss der Inhalt während der Startsequenz geladen werden. Der CRAM wird nach dem Einschalten der Betriebsspannung (Power Up) und anschließenden Power-On Reset beschrieben. Bei nichtflüchtigen FPGAs entfällt dieser Konfigurationsschritt, der je nach Datenrate und Größe des FPGA länger als eine Sekunde werden kann. Gleichung 2.1 zeigt die Abhängigkeit der Ladezeit von der Datenrate und der Größe des Konfigurationsspeichers. Aktuelle Bausteine lassen bei serieller oder 32 Bit paralleler Datenübertragung Konfigurationstakte von 100 MHz [Xil13f] bis 133 MHz [Alt14] zu. Diese Übertragungsraten sind durch die kurzen Leitungswege vom externen Speicher bis zum FPGA erreichbar. Bei längeren Distanzen oder Busstrukturen sinkt auch die Übertragungsgeschwindigkeit.

$$t_{Konfiguration} = \frac{Bitzahl_{Konfiguration}}{Datenbreite_{Interface} \times f_{Konfiguration}} \tag{2.1}$$

Aufgrund der nicht beschränkten Schreibzyklen von SRAM Zellen lassen sich auch Schreibverfahren zur Laufzeit einsetzen. Üblicherweise wird diese Methode als Dynamische Rekonfiguration bezeichnet. Eine

Tabelle 2.1.: Rekonfigurationsarten des FPGAs

	Vollständiges FPGA	Untermenge des FPGAs
Im Betrieb	Dynamische Rekonfiguration	Partielle Dynamische Rekonfiguration (PDR)
Außerhalb des funktionalen Betrieb	Statische Konfiguration	Partielle Rekonfiguration (PR)

Einordnung der Arten von Rekonfigurationen ist in Tabelle 2.1 dargestellt. Hier ist besonders die Partielle Dynamische Rekonfiguration (PDR) interessant, da sich zur Laufzeit Teilfunktionalitäten austauschen lassen. Das Prinzip von PDR ist in Abbildung 2.3 dargestellt. Ursprünglich ist diese Funktionalität im Jahr 2004 mit der Intention des zeitlichen Multiplex von Teilfunktionen in Xilinx FPGAs integriert worden [Xil09]. PDR hat sich nun auch in aktuellen Altera Bausteinen verbreitet [Alt14] und eine Vielzahl weiterer Forschungsaktivitäten inspiriert. Ein Schwerpunkt ist die Verkürzung der Konfigurationszeit durch sequentielles Laden von Teilschaltungen [HMS+10] mittels einer verkleinerten statischen Partition. Nach Gleichung 2.1 reduziert sich die statische Konfigurationszeit zum Starten des Systems linear mit der Größe der statischen Region. Somit lassen sich zeitkritische Spezifikationen wie die PCIe®Transfer Anforderung mit 100 ms nach Anlegen der Versorgungsspannung [TK10] erfüllen. Der Aufwand und Nutzen der PDR muss in der Applikation über Modellbildung erörtert werden [PDH11], jedoch ist hier die Speicheranbindung des externen Speichers der limitierende Faktor für die Konfigurationsgeschwindigkeit. Die interne Datenrate der Konfiguration liest mit maximal 400 MB / s (100 MHz bei 32 Bit Datenbreite) [Xil13b] über der möglichen Geschwindigkeit durch den kommerziell vertretbaren Quad-Serial Peripheral Interface (SPI) Bus von 50 MB/s (100 MHz bei 4 Datenbit).

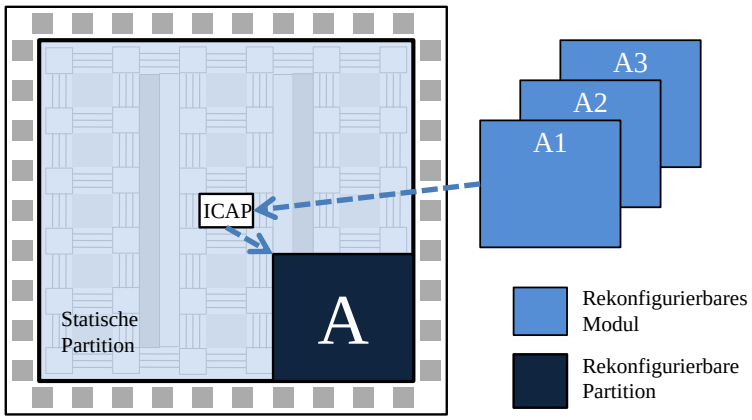


Abbildung 2.3.: Grundprinzip Partielle Rekonfiguration

2.1.1.3. Automotive Eigenschaften

Die zunehmende Datenverarbeitung im Kraftfahrzeug (Kfz) bringt auch die FPGAs in die hochvolumigen und niedrigpreisigen Steuergeräte. Vor allem Fahrerassistenzsysteme wie Rückfahrkamera, Frontkamera, Radar- und Ultraschallauswertung sowie zunehmend Sensor-Daten-Fusion benötigen die Leistungsfähigkeit und Time-To-Market von FPGAs. Daher sind bei allen großen Herstellern von FPGAs auch Produktlinien verfügbar, die für den Automobilmarkt durch den Standard AEC-Q100 [AEC07] qualifiziert sind. Hier werden Stichproben eines Quartals gegen Elektrostatische Entladung (engl. Electrostatic Discharge) (ESD), Elektromagnetische Verträglichkeit (EMV) und Halbleiterstress geprüft. Interessant für FPGAs in sicherheits-

relevante Echtzeitsystemen ist die Temperaturverträglichkeit, d.h. die Zuverlässigkeit bei hohen und niedrigen Temperaturen. Aktuell verfügbare Bausteine der Xilinx XA Familie, Altera 5er Serie und Lattice FPGAs sind bis zur Kerntemperatur T_j = -40°C bis +125°C qualifiziert. Das entspricht etwa dem Grad 2 des AEC-Q100 Temperaturgrad mit einer Umgebungstemperatur T_a = -40°C bis +105°C. Flash-basierte FPGA der Firma Microsemi sind aufgrund der ausgereiften und älteren Prozesstechnologie bis T_j = 135°C qualifiziert und auch bei höheren Temperaturen erprobt [BMVBK10]. Somit sind frei programmierbare ICs technologisch für den Automobilmarkt geeignet, wenn auch nicht in besonders rauen Umgebungen im Motorraum.

2.1.2. Entwicklungsablauf

Die Entwicklung einer FPGA Schaltung läuft in ähnlichen Schritten wie die Entwicklung eines ASIC ab [Sau10], jedoch unterscheidet sich das Resultat der Abläufe. Nach der Konzeptphase beginnen die Entwicklungsumgebungen der Hersteller bei der Beschreibung der Funktion mittels einer Hardwarebeschreibungssprache wie Very High Speed Integrated Circuit Hardware Description Language, VHSIC Hardware Description Language (VHDL) [IEE09] oder Verilog Hardware Description Language (HDL) [IEE06a]. Danach wird die Beschreibung durch die *Synthese* in eine elektrische Netzliste umgewandelt. Hier werden die in der HDL beschriebenen Architekturen und Makros interpretiert und in eine Abbildung aus FPGA spezifischen Instanzen wie LUT, FD FF oder RAM Blöcke umgewandelt. Anschließend wird im *Map* Prozess die erzeugt Netzliste zusammen mit weiteren IP Schaltungen auf die verfügbaren Ressourcen des spezifischen FPGA abgebildet und vorläufig platziert. Im anschließenden Schritt des *Place and Route* (P&R) erfolgt die finale Belegung der Ressourcen sowie die Zuweisung der Leitungen auf die konfigurierbaren Verbindungspunkte. Das Resultat nach dem nächsten Schritt ist ein Bitstrom (engl. *Bitstream*), mit dem die Konfigurationsbits nach dem Laden gesetzt werden. Während die Synthese noch mit Programmen von Drittanbietern durchgeführt werden kann, z.B. Mentor Graphics® Precision oder Synopsys® Synplify, muss der Map Prozess und nachfolgende Prozesse mit den Programmen der FPGA Hersteller ausgeführt werden.

In Industrieanwendungen muss dieser Ablauf ebenfalls eingehalten werden, jedoch um Verifikationsschritte erweitert. In Abbildung 2.4 wird der Entwicklungsfluss gemäß des V-Modells aus der ISO 26262 [ISO11b] dargestellt. Aktuelle Entwicklungen beginnen mit einem abstrakten Verhaltensmodell, erzeugt in SystemC [IEE06b], Matlab®[Mat12] oder OpenCL [Khr14]. Diese abstrakten Modelle dienen der Systementwicklung und Gesamtsystem-Simulation und werden daher mit Anwendungsfällen simuliert (Use-Case Simulation). Aus dem abstrakten Modell wird mittels High Level Synthese (HLS) oder händisch eine HDL Beschreibung erzeugt, die danach den bekannten Entwicklungsablauf fortführt. Die Verifikation der HDL Verhaltensbeschreibung wird mit Einzelfunktionstests (engl. Unit Tests) durchgeführt, da eine Simulation von Anwendungsfällen auf dieser Abstraktionsebene mehrere Tage in Anspruch nehmen würde. Nach der Synthese und dem P&R wird eine statische Timing Analyse durchgeführt, bei der unter anderem Signallaufzeiten, Taktausbreitung und Sample and Hold Zeiten von FFs unter definierten Bedingungen (*Timing Constraints*) überprüft werden. Allenfalls in seltenen Fällen wird hier noch eine Einzelfunktion mit Timing Informationen simuliert. Die endgültige funktionale Verifikation findet aufgrund der Reprogrammierbarkeit an realer FPGA Hardware statt.

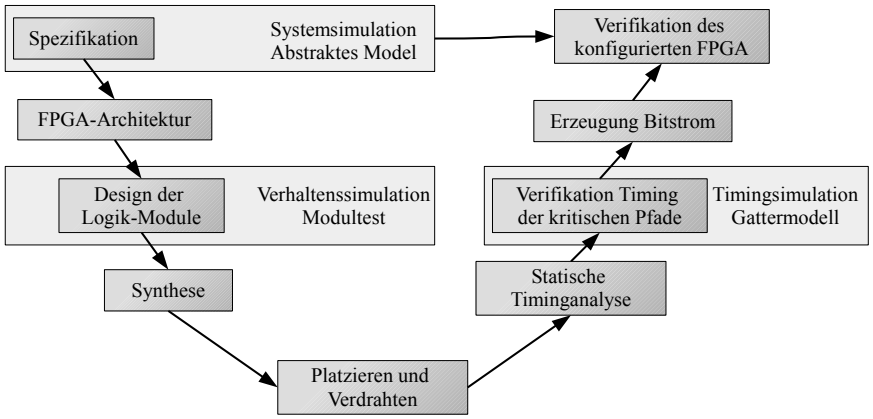


Abbildung 2.4.: FPGA spezifisches V-Modell in Industrieapplikationen

2.2. Überblick der Ausfallsicherheit des FPGA auf Logikebene

Die in Abschnitt 2.1 dargelegte FPGA Technologie muss für sicherheitsrelevante Applikationen einer ausführlichen Fehleranalyse standhalten. In diesem Abschnitt sind grundlegende Effekte und Toleranzmechanismen beschrieben, die auch im Rahmen einer für den Automobilbereich wichtigen ISO 26262 Qualifizierung in den Fokus rücken. Jedoch sind auch Applikationsbereiche wie Luftfahrt und Raumfahrt, Kraftwerktechnik und Eisenbahntechnik von Fehlern mit kritischen Ausmaßen betroffen. Bei den Gegenmaßnahmen muss unterschieden werden, ob der FPGA Anwender bzw. Programmierer oder der FPGA Hersteller wirksam einen Fehler unterdrücken oder die Fehlerwahrscheinlichkeit reduzieren kann [Sut13] [SSC08]. Hierbei wird zwischen *Detektion* und *Toleranz* von Fehlern unterschieden. Wobei für die Gewährleistung der funktionalen Sicherheit nicht beide Methoden gleichzeitig notwendig sind.

2.2.1. Defektquellen und Fehlerausbreitung

Wie bei jedem Very Large Scale Integration (VLSI) IC treten auch beim FPGA zur Fertigungszeit und zur Laufzeit *Defekte* (engl. *fault*) auf [SSC08]. Ein Defekt bezeichnet eine veränderte Bedingung oder Situation, die zu einem Fehler führen kann. Beispiele für einen Defekt in einem IC kann ein Oxiddurchbruch, ein ESD Impuls auf einer leitenden Struktur oder ein Strahlungseffekt sein. Ein *Fehler* (engl. *error*) resultiert aus einem Defekt oder aus einer systematischen Ursache und bezeichnet die Abweichung von einem Soll-Zustand oder einer Soll-Bedingung. Dies kann zum Beispiel ein falscher Ausgang einer Arithmetikeinheit oder ein falscher Zustandswert in einem FF sein. Letztendlich mündet der Fehler wiederum in einem Schadensereignis (engl. failure) des Gesamtsystems. Die Begriffe Defekt, Fehler und Schadensereignis müssen unterschieden und teilweise getrennt betrachtet werden. Für eine Quantisierung der Ausfallrate λ durch Defekte wird die irreführende Einheit Failure In Time (FIT) verwendet mit 1 FIT = 1 Ausfall pro 10^9 Stunden. Dabei gilt für die mittlere Zeit bis zu einem Defekt oder Fehler die *Mean Time Between Failures* MTBF = $\frac{1}{\lambda}$.

2.2.1.1. Permanente und flüchtige Fehler

Fehler innerhalb des FPGAs lassen sich in die Kategorien *permanent* und nicht-permanent, genannt *flüchtig*, einordnen. Neben mechanischen Fehlern resultieren permanente Fehler hauptsächlich aus der Degradation

Tabelle 2.2.: Fehlerarten aus Neutronen- und Alpha Strahlungsdefekten [NYKB08] [KPCC97]

Einteilung	Defekt	Beschreibung
Flüchtig (Soft Error)	Single Event Upset (SEU)	Eine Speicherzelle kippt ungewollt von 0 auf 1 oder von 1 auf 0.
	Multi-Bit Upset (MBU)	Mehrere Speicherzellen kippen ungewollt aufgrund des gleichen Strahlungsdefektes.
	Single Event Transient (SET)	Transienter Ladungsüberschuss in der Logik.
Pseudo-Permanent	Soft Error Functional Interrupt (SEFI)	Ein Fehler in der Logik, z.B. undefinierter Zustand oder falscher Zustand. Kann meist nur durch Reset behoben werden.
	Single Event Latch-up (SEL)	Durchschalten eines parasitären PNPN-Thyristors in CMOS Strukturen.
Permanent	Single Event Burnout (SEB)	Durchschalten einer parasitären Transistorstruktur in Leistungs-MOSFET.
	Single-Event Gate Rupture (SEGR)	Durchbruch des Gate-Oxids durch hohe Spannungen am Gate in Leistungs-MOSFET oder Flash-Zellen.

der Halbleiter Prozessstrukturen wie etwa durch freie Ladungsträger in der Gate-isolierenden Schicht des Metall-Oxid-Halbleiter Feldeffekttransistor (MOSFET)s oder durch Elektromigration in den Metalllagen [SKM+08]. Die degradierenden Defekte wie Hot Carrier (HC), Negative Bias Thermal Instability (NBTI) und Time-Dependent Dielectric Breakdown (TDDB) reduzieren maßgeblich die Lebensdauer, lassen sich aber auf Zuverlässigkeiten von größer 10 Jahre abmildern [SWC10] [HAP+07]. Für eine Zuverlässigkeitsanalyse im Zuge der Automobilqualifikation werden diese Defekte mit FIT Werten versehen [AEC07]. Ferner resultieren diese Defekte in einer Sicherheitsanalyse zu Fehlermodellen wie Haftfehlern (*Stuck-At-0* und *Stuck-At-1*), Kurzschlüssen (*Bridging* oder *Short*) und Fehlern im Signallaufzeiten (*Path-Delay*). Von diesen Fehlern sind FPGAs ebenso betroffen wie ASICs im gleichen Technologieknoten.

Als flüchtige Fehler bezeichnet die Literatur diejenigen Fehler, deren ursächliche Defekte nicht dauerhaft anhalten und somit nur temporär vorhanden sind. Neben EMV sind Strahlungsdefekte eine Hauptursache für flüchtige Fehler in VLSI Schaltungen [NYKB08]. Die Strahlung wird vornehmlich durch schwere Ionen (α-Teilchen) aus der Mouldmasse sowie durch Neutronenstrahlung aus der Atmosphäre erzeugt. Hierzu zeigt Tabelle 2.2 eine Übersicht der aus Strahlungseffekten resultierenden Fehler, die unter dem Oberbegriff Single Event Effect (SEE) einzuordnen sind. Neben den flüchtigen Fehlern reduzieren auch permanente Strahlungsfehler die Zuverlässigkeit von FPGAs. Die Metrik für die Strahlungsdefekte SEE wird als Soft Error Rate (SER) bezeichnet. Bei SEEs liegt dabei ein physikalischer Vorgang der Ladungserzeugung durch hochenergetische Strahlung zugrunde, dargestellt in Abbildung 2.5. Hier erzeugt ein energiereiches Neutron oder ein α-Teilchen sekundäre Teilchen und Elektron-Loch Paare nahe der Raumladungszone einer Complementary Metal Oxide Semiconductor (CMOS) Struktur. Diese Ladungsträgern tunneln in die Strukturknoten und erzeugen eine kritische Ladung Q_c. Abbildung 2.6 zeigt eine einfache SRAM Zelle ohne Härtungsmaßnahmen, wobei Q_c den nMOS Transistor der negierten Bitline BL kippt. Tritt das SEE an anderer Stelle im Si auf, kann Q_c transient weiter in der Logik propagieren (SET). In den in dieser Arbeit betrachteten echtzeitfähigen Automobilanwendungen mit Taktraten von maximal 300 MHz überwiegt die Wahrscheinlichkeit für SEU P_{SEU} der Wahrscheinlichkeit für SET P_{SET} und weiteren SEE Effekten [GEBT05]. Für einen 180 nm Technologieknoten liegt die Schwelle für P_{SEU} größer als P_{SET} bei circa 900 MHz aufgrund der erhöhten zeitlichen Maskierung. Zudem weisen FPGAs eine um Faktor 10 höhere Leitungskapazität gegenüber technologisch gleichen ASIC auf und sind demnach um ein vielfaches unempfindlicher gegenüber SET [LDF+05].

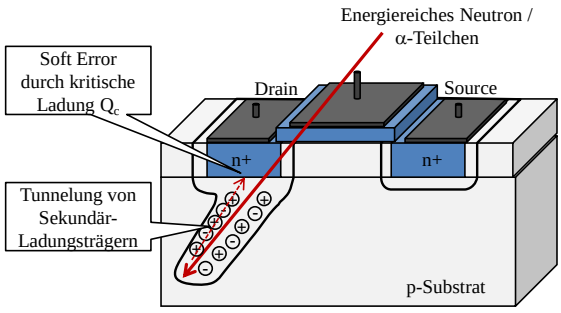


Abbildung 2.5.: Strahlungsdefekt durch Single Event Upset nach [NYKB08]

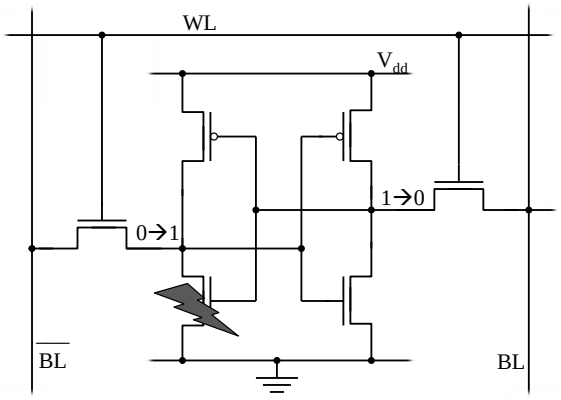


Abbildung 2.6.: SEU in SRAM Zelle

2.2.1.2. Fehlerpropagation

Einmal aufgetretene flüchtige SEE verteilen sich als Fehler über die elektrischen Netze und kombinatorischen Elemente in die Zustände der Schaltung [NYKB08]. Dieser übertragene Fehler im Zustandsspeicher wird *persistenter* Fehler genannt [MCG+05]. Persistente Fehler können nicht durch eine erneute Konfiguration des FPGA behoben werden, da bereits der Zustandsspeicher der Schaltung korrumpiert wurde. Der Übergang in die Persistenz wird jedoch durch drei wichtige Arten der Fehlermaskierung abgeschwächt. Die erste Hürde für die Fehlerpropagation wird als *Elektrische Maskierung* bezeichnet. Hierbei wird die erzeugte Ladung Q durch Leitungs- und Gatterkapazitäten unter das Niveau von Q_c verringert und nicht mehr als falscher boolescher Wert interpretiert 2.7. Bei SETs liegt der falsche Wert nur für einige Picosekunden an und zeigt somit eine kleine Wahrscheinlichkeit, an einem Flankenwechsel des Taktes in einen Zustandsspeicher übernommen zu werden. Diese zeitliche Beschränkung der Auswirkungswahrscheinlichkeit wird als *Zeitliche Maskierung* (engl. Timing Derating) bezeichnet und ist umgekehrt proportional zur Taktfrequenz. Ein weiterer limitierender Faktor für die Fehlerausbreitung liegt in der Beschränkung durch die Logikfunktion selber und gilt sowohl für SET als auch für SEU. Diese *Logische Maskierung* wird als auch als Logic Derating bezeichnet und verhindert aufgrund der booleschen Funktion des Gatters die Propagation eines fehlerhaften Signals an einem kombinatorischen Ausgang. Beispielsweise wird ein fehlerhafte “0” an einem Eingang eines UND-Gatters nicht propagiert, wenn an weiteren Eingängen Nullen anliegen. Die benannten Maskierungen gelten sowohl für ASIC Bausteine als auch für FPGAs. Da jedoch in FPGAs der Konfigurationsspeicher die Netzliste abbildet, wird bei einem SEU in diesem Speicher die logische Maskierung in der Netzliste nicht mehr vorhersagbar. Somit kann eine von der logischen Maskierung unabhängige Fehlerpropagation stattfinden.

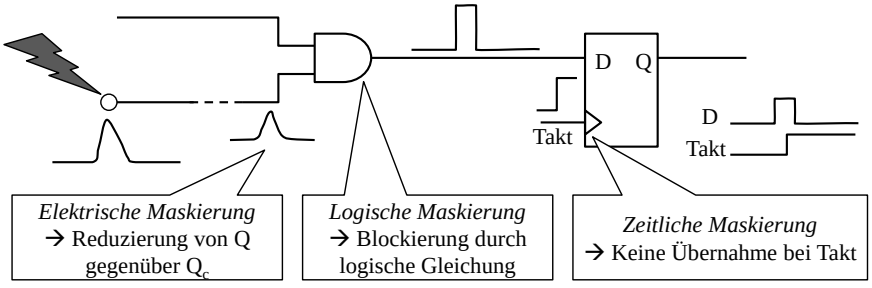


Abbildung 2.7.: Fehlerausbreitung und Fehlermaskierungen nach SEE in der Logik

Tabelle 2.3.: Schwachstellen (SEFI) Analyse

Physikalische Veränderung	Simulation	Emulation
• Definierte Neutronenstrahlung ([FPV10]) • Energieeintrag mittels Laser ([PMN+09]) • Elektromagnetische Interferenz ([VCG+05])	• Vorwärtsgerichtete Fehlerinjektion in der Netzliste ([KSL+06], [MWE+03]) ([FPV10]) • Rückwärtsanalyse der Fehlerausbreitung ([HRSH11])	• Initiale oder laufzeitfähige Bitstrom-Manipulation ([LBN10], [DASS09], [SKK11], [MCG+05]) • Statische Zusatzelemente in der Schaltung zur Fehlereinspeisung zur Laufzeit ([LOGVPGEA05],[MS13])

2.2.2. Fehlertoleranz zur Reduzierung der Fehlerauswirkung

Die Propagation der in Abschnitt 2.2.1.1 erläuterten Fehler auf Systemebene müssen so weit verhindert werden, dass kein Schadensereignis (engl. *failure*) auftritt. In diesem Abschnitt werden ausschließlich Toleranzmechanismen innerhalb des FPGAs angeführt. In einem Steuergerät mit weiteren ICs können neben den FPGA-internen Maßnahmen auch systemweite Fehlertoleranzen implementiert werden. Diese sind jedoch nicht im Fokus dieser Arbeit. Für fehlertolerante FPGA Schaltungen werden drei Schritte erläutert. Zudem ist es für einen Schaltungsentwickler entscheidend, ob eine Maßnahme durch den FPGA Hersteller oder durch den Anwender implementiert werden kann. Mechanische und degradierende Defekte der CMOS Strukturen müssen ebenso wie im ASIC adressiert werden [SKM+08].

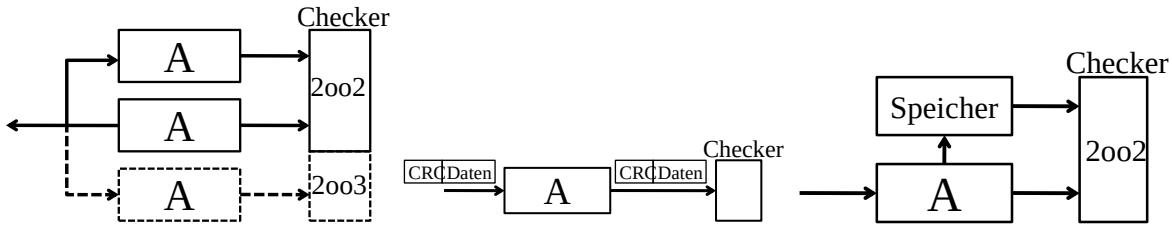
2.2.2.1. Schwachstellenanalyse

Neben der Reduzierung der Defektwahrscheinlichkeit durch Technologie- und Prozessmethoden muss die Schwere der Auswirkungen der Fehler beurteilt werden. In einer Digitalschaltung auf dem FPGA hat nicht jedes transient fehlerhaftes Konfigurationsbit oder jedes transient fehlerhaftes Gatter ein Schadensereignis zur Folge [MCG+05] [MWE+03]. Permanente Defekte und somit statische Fehler wirken sich über die Lebensdauer der Schaltung immer auf den Ausgang aus, insofern keine redundanten oder schlafenden Schaltungsteile existieren. Somit zielen die Schwachstellenanalysen aus Tabelle 2.3 in erster Linie auf flüchtige Fehler oder das Finden von kritischen Speicherelementen ab. Genauer gesagt werden dabei die Propagation von flüchtigen Fehlern und die Laufzeit von permanenten Fehlern untersucht und Gegenmaßnahmen evaluiert.

Die Methoden für die SEFI Analyse lassen sich in drei Prinzipien gliedern. Durch direkte physikalische Veränderungen in der Si-Struktur lassen sich Defekte meist ungezielt aber realistisch nachstellen. Zudem kann die Fehlerpropagation zur Laufzeit untersucht werden. Im Gegensatz dazu können mittels simulativer Methoden dedizierte Defektstellen nachgestellt und der Propagationsweg genauestens beobachtet werden. Nachteilig wirken sich hier die lange Rechenzeit, die Abhängigkeit von Eingangspattern und die Annahme einer korrekten Netzliste im CRAM aus. Durch Emulation von Fehlern kann zur Laufzeit eines FPGA Systems ein Fehler injiziert werden. Hier sind die Fehlermodelle jedoch sehr stark auf SEUs eingeschränkt. In Tabelle 2.3 werden mit den physikalischen Veränderungen des ICs und der Manipulation des Bitstromes zudem bereits Methoden der Fehlerinjektion in den Konfigurationsspeicher aufgezeigt. Diese Fehler im CRAM werden im späteren Abschnitt 2.3 eingeführt und genauer untersucht.

2.2.2.2. Detektion und Korrektur von Fehlern in der Logik

Die Defektwahrscheinlichkeit bleibt bei jedem VLSI IC immer größer als Null. Die Annahme der *zufälligen Hardwarefehler* in dem Standard ISO 26262 ist dieser Tatsache geschuldet [ISO11a]. Somit müssen gegen permanente Fehler und auch gegen die Fehlerausbreitung aus den ermittelten Schaltungsteilen der Schwachstellenanalyse Toleranzmechanismen eingefügt werden. In der Praxis haben sich physikalische, logische und zeitliche Redundanzen als wirksam erwiesen [Sut13]. Physikalische Redundanzen nach Abbildung 2.8a werden als Double Checker Redundancy (DCR) doppelt mit einer Zwei-aus-Zwei Prüfinstanz (`2oo2`) oder als Triple Modular Redundancy (TMR) dreifach mit einem Zwei-aus-Drei Entscheider (`2oo3`) eingesetzt. Diese Art der Redundanz lässt sich auf vielen Abstraktionsebenen einfügen. Auf der CMOS Strukturebene sichern doppelte p- und n-Kanal MOSFETs einen Ausfall ab [NYKB08]. Eine Abstraktionsebene höher verwenden die Hersteller verschiedene Aufbauarten für FF, LUT und SRAM Zellen mit zusätzlichen Transistoren und Leitungen. Bis zu dieser Ebene hat der Endanwender keine Möglichkeit zur Optimierung der Fehlertoleranz. Ab der Implementierungsebene mit der Netzliste können feingranulare Redundanzen eingefügt werden [NSB12], bei der einzelne Gatter verdoppelt oder verdreifacht werden. Diese Art der physikalischen Redundanz in der Schaltungsentwicklung ist skalierbar und vielseitig [CB09]. Daher bieten Hersteller wie Xilinx mit dem TMRTOOL [Xil12b] oder Mentor Graphics mit Precision Hi-Rel [PM11] vollwertige Lösungen für VHDL Modul- und Netzlistenredundanzen. Zudem gibt es weitere Optimierungen für das automatisierte Einfügen von TMR und DCR mit Hinblick auf Flächenverbrauch, Leitungsredundanzen, Timing und Anwendung der PDR zum Modultausch [BMS11] [KFC06] [JW10] [RMR09] [Ste09]. Neben der Logik lassen sich auch die RAM Blöcke redundant auslegen. Nachteilig bei allen physikalischen Redundanzstrukturen ist der Entscheider (engl. Voter) oder Überwacher (engl. Checker), der ebenfalls redundant oder extern ausgelegt werden muss [FPV10]. Ferner widerspricht der erhöhte Flächenbedarf einer kostenoptimalen Lösung.



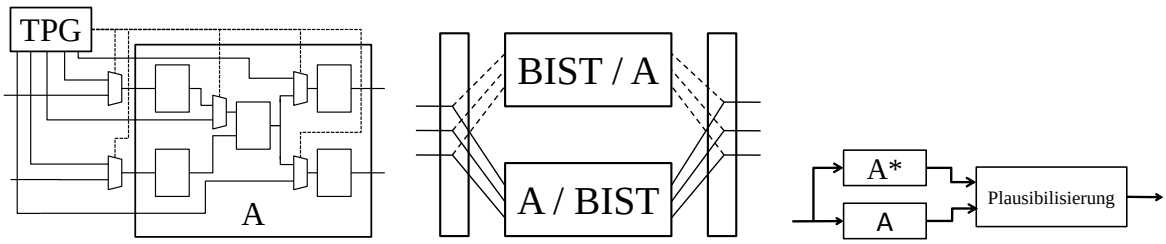
(a) Physikalische Redundanz **(b)** Logische Redundanz **(c)** Zeitliche Redundanz

Abbildung 2.8.: Redundanzen zur Fehlerdetektion und Fehlertoleranz

Bei logischen Redundanzen werden interne Zustände oder Kommunikationen gesichert ausgelegt, beispielsweise über zusätzliche Cyclic Redundancy Check (CRC) Prüfsummen im Datenstrom aus Abbildung 2.8b. Zustandsautomaten werden mittels *Hot-One* Verfahren oder durch Definition von Alternativzuständen gesichert [PM11]. Ferner kann der interne Datenfluss logisch redundant ausgelegt werden, wenn die CRC Prüfsumme für die Daten eines Busses oder eines Signalbündels mit übertragen wird [LBN11]. Interne RAM Blöcke können über ECC Paritätsbits abgesichert werden [RFW10]. Hierzu werden zu je 64 Datenworten weitere 8 Paritätsbits hinzugenommen. Mit einer zeitlichen Redundanz kann der Zeitpunkt eines Fehlerauftritts bestimmt werden, da hier derselbe Algorithmen zeitlich nacheinander zweimal durchlaufen wird [YM01] (Abbildung 2.8c).

Fehlertolerante Systeme im Bereich des automatisierten Fahrens oder elektrische Lenkung müssen auch im Fehlerfall weiter operieren (engl. fail operational). In vielen Fällen genügt nach einem Fehler jedoch die korrekturunabhängige Erreichung eines sicheren Zustandes mit anschließender Korrektur (engl. fail safe).

Hierfür müssen permanente und flüchtige Fehler detektiert werden. Mittels eines Applikationsspezifischen Selbsttest (engl. application specific Build-In Self-Test (BIST)) können Haftfehler und Kurzschlüsse auf den verwendeten FPGA Ressourcen erkannt werden [Tah06]. Durch Anpassung der Schaltung mit Zusatzgattern wie ODER und AND-Verknüpfungen können dedizierte Pfade getestet werden und die Testabdeckung steigt (Abbildung 2.9a). Ferner können FPGA Ressourcen durch einen BIST belegt werden während die funktionale Schaltung durch PDR umplatziert wird (engl. roving) [SSC08] (Abbildung 2.9b). Im Falle von detektierten permanenten Fehlern werden Alternativplatzierung aus der Designphase verwendet [WJW+13]. Eine weitere Möglichkeit zur Fehlerdetektion zeigt die Plausibilisierungsprüfung (Abbildung 2.9c). Hier wird der Algorithmus A formalisiert und es werden kritische Parameter alternativ berechnet (engl. Concurrent Error Detection (CED))[LCR03]. Die Plausibilisierung deckt latente Fehler nicht ab, aber es werden alle sicherheitskritische Parameter zur Laufzeit evaluiert. Dabei kann der ursprüngliche Datenfluss durch zeitliche Redundanz nochmal codiert durchlaufen werden oder durch physikalische Redundanz jeweils codiert und uncodiert berechnet werden. Alternativ kann durch codiertes Prozessieren (engl. encoded processing) ein Berechnungsfehler entdeckt werden [SFRB05].



(a) Applikationsspezifischer BIST **(b)** BIST mit rotierenden Modulen **(c)** Plausibilisierungstest

Abbildung 2.9.: Fehlerdetektion in der Logik einer FPGA Schaltung

2.3. Beschreibung von flüchtigen Fehlern im Konfigurationsspeicher

Wie bereits in den Ausführungen über Fehlerarten erwähnt, sind SRAM Zellen besonders anfällig für SEUs. Die Gründe für die Fokussierung auf SEUs anstelle von SET sind in Abschnitt 2.2.1 erklärt. Ein Großteil dieser Arbeit zielt aufgrund der hohen Dichte von SRAM-Zellen auf den Konfigurationsspeicher kommerzieller FPGAs, exemplarisch der Firma Xilinx. Da die FPGAs der Firma Altera technologisch ähnlich aufgebaut sind und auch bei der Taiwan Semiconductor Manufacturing Company (TSMC) gefertigt werden, treffen folgende Aussagen auch für diese Bausteine zu. Im Weiteren wird der SRAM des Konfigurationsspeichers als CRAM bezeichnet. Moderne große Bausteine haben ein CRAM von ca. 300 Mbit [Xil13b, Alt13b]. Es zeigt sich zudem in den folgenden Ausführungen, dass der Konfigurationsspeicher eine bis zu vierfach höhere Wahrscheinlichkeit für flüchtige Fehler gegenüber Logik-FF innerhalb eines FPGA hat. Flüchtige Fehler können als Einzelbitfehler gewertet werden. In diesem Abschnitt werden nichtflüchtige Fehler nicht separat behandelt, da diese zum großen Teil in der Fehlerdetektion enthalten sind.

2.3.1. Fehlerwahrscheinlichkeiten für flüchtige Fehler

Das CRAM wird größtenteils statisch zur Ablage der Netzlisteninformation genutzt. Daher können flüchtige Fehler nur als SEUs zuverlässig detektiert und ausgewertet werden. In Abbildung 2.10 wird quantitativ die SER für Xilinx FPGAs dargestellt. Die aus dem Xilinx Zuverlässigkeitsreport [Xil13c] entnommenen Werte entsprechen in der Größenordnung auch Werten weiterer Untersuchungen [Les12, DLGO04]. Die Defektwahrscheinlichkeit für SEUs im CRAM hängt unter anderem von der kritischen Ladung Q_c, der kritischen Fläche A_c und dem Strahlungsfluss im Substrat ab [NYKB08]. Durch zusätzliche Kapazitäten und Widerstände innerhalb der SRAM Zelle wird Q_c auf Kosten zusätzlicher Fläche erhöht. Bei kleiner werdenden Technologieknoten verringert sich Q_c jedoch und muss durch die reduzierte kritische Fläche A_c kompensiert werden. Zudem wird in aktuellen 28 nm TSMC Technologien der α-Strahlungsfluss durch bis zu 12 Metalllagen reduziert [DW12]. Diese Abschirmung bewirkt zusammen mit der Verwendung von *Ultra Low-Alpha* Mouldmasse und Materialien mit kleiner Wechselwirkungswahrscheinlichkeit eine reduzierte SER gegenüber Vorgänger-Technologien.

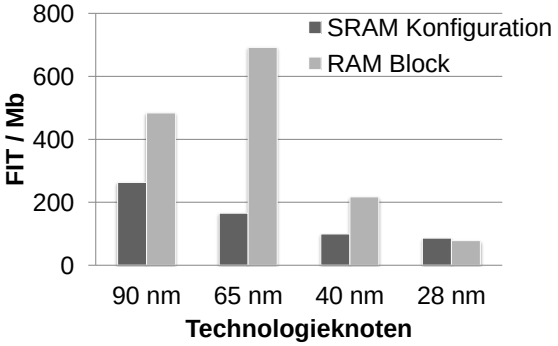


Abbildung 2.10.: Mittelwert SER für Xilinx FPGAs [Xil13c]

Tritt ein SEU im Konfigurationsspeicher auf, trifft er mit einer Wahrscheinlichkeit von maximal 10 % ein *essentielles Bit* [Xil13c]. Nur essentielle Bits im CRAM haben Einfluss auf die Konfiguration und die Netzliste der Schaltung. Somit werden etwa 90 % des CRAM aufgrund der feingranularen Struktur nicht verwendet. Für eine weitere Fehlerpropagation in der Logik gemäß Abschnitt 2.2.1 muss demnach ein essentielles Bit betroffen sein.

Der Vergleich der Fehlerwahrscheinlichkeit von FF und CRAM soll exemplarisch anhand des Xilinx Kintex 7 XC7K325T aufgezeigt werden. Tabelle 2.4 zeigt den quantitativen Vergleich, wobei die SER Metriken für den 28nm Technologieknoten von TSMC gelten. Es ergibt sich eine bis zu vierfach höhere SER für das CRAM gegenüber den enthaltenen FF, trotz der durchschnittlich nur 10 % verwendeten Konfigurationsbits. Mit diesen Zahlen liegt auf der Absicherung des Konfigurationsspeichers eine sehr hohe Priorität.

Die Fehlerwahrscheinlichkeit für einen SEU hängt davon ab, wie viele dieser flüchtigen Fehler zu Einzelbitfehlern führen. Um Multibit-Fehler (MBU) zu vermeiden, verwenden die Hersteller Bitverschiebungen zwischen den CRAM Bereichen (engl. bit scrambling) in einer verschachtelten Anordnung (engl. interleaved bits) [BSV11] [WTH13]. Hierdurch werden logisch zusammenhängende Speicherzellen aus einem Wort oder Bereich auf dem Die physikalisch voneinander getrennt (Abbildung 2.11 b). Dadurch kann ein MBU zu SEUs in mehreren Speicherwörtern oder Speicherframes werden [ICW+06]. Die Wahrscheinlichkeit, dass ein MBU sich in allen beteiligten Frames als SEU auswirkt liegt bei über 90 % [WTH13]. Ein Zweibitfehler in einem

Frame hat noch eine Wahrscheinlichkeit von 7,5 %.

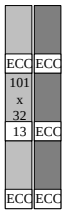
Tabelle 2.4.: Fehlerwahrscheinlichkeiten für flüchtige Fehler im CRAM und in FF eines Xilinx Kintex 7 XC7K325T

	CRAM	Flip-Flop
SER [1]	86 FIT / Mbit [Xil13c]	≈ 500 - 1000 FIT / Mbit [GBN+11, LBW+13]
Anzahl Bits	91,5 Mbit [Xil13b]	0,41 Mbit [Xil14]
SER pro Device [1]	≈ 790 FIT [2]	≈ 200 - 410 FIT

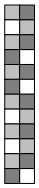
[1] Höhenniveau: New York

[2] Max. 10 % Essentielle Bits [Xil13c]

Auch Flash-basierte Speicherzellen sind nicht immun gegen SEU. Sie zeigen jedoch eine um 10^3 reduzierte Wahrscheinlichkeit für Datenerhaltungsprobleme durch SEU gegenüber SRAM-basierten FPGA [Sla11, CPL+02] . Im Zuge der kleiner werdenden Technologieknoten für Flash Zellen sinkt die Ladung im *Floating Gate* und erhöht in der Zukunft die SEU Rate [Sla11] [KFV+11]. Somit sind Detektionsmechanismen für Fehler im Konfigurationsspeicher auch für diese Technologie nötig.



(a) Logische Anordnung mit SECDED ECC Paritätsbits



(b) Physikalische verschachtelte Anordnung (interleaving bits)

Abbildung 2.11.: Logische und physikalische Anordnung von CRAM Frames

2.3.2. Funktionale Fehlermodelle

Im Falle eines SEUs im CRAM sind die Verbindungsknoten der flexiblen Verdrahtung aus Abbildung 2.2 in Abschnitt 2.1.1 zu ca. 90 % betroffen. Insofern stellen die Verbindungsknoten und deren Test die besondere Herausforderung für eine effiziente Fehlerdetektion und -Korrektur dar. Die genaue Analyse der potentiellen Fehlermodelle in den Verbindungen zeigt auf, dass auch diese CRAM Anteile deterministisch und lokal begrenzt getestet werden können. Im Folgenden werden die Fehlermodelle und notwendigen Überwachungen für die logischen Verbindungen, die statischen Signale und die verteilen Netze aufgezeigt. Als verteiltes Netz ist standardmäßig der Taktbaum anzusehen oder in Einzelfällen eine Enable- oder Reset-Struktur.

- *Logische Verbindungen*
  Die Logiknetze sind Punkt-zu-Punkt Verbindungen von einem Verbindungsknoten zum anderen. Jegliche Signalverteilung erfolgt über die Konfiguration in diesen Knoten, wobei die Verbindungen selbst dedizierte Signalwege ohne CRAM Anteil sind. Daher entstehen Fehler für die logischen Signale in den Verbindungsknoten, die sich nach dem P&R Vorgang für die elektrischen Netze lokalisieren lassen. Ein Fehler in einem anderen Knoten als den zum Netz gehörigen wirkt sich nicht auf die logischen

Tabelle 2.5.: Korrelation der Fehlermodelle im Verbindungsgitter zu CRAM Ressourcen

Verbindungsart	Logische Verbindung	Statisches Signal	Verteiltes Netz
Ausbreitung	Punkt-zu-Punkt	Ein Major Frame	Multiple Major Frames
Lage der PIP	Verbindungsknoten	Verbindungsknoten Quelle: Instanz `TIEOFF`	global: Takttreiber lokal: Verbindungsknoten
Fehlermodell	Falsche Verbindung	Kurzschluss	Erhöhter Jitter, kein Takt
Auswirkung	Fehlerhafter Pegel	Erhöhte Stromaufnahme	Funktionale Fehler

Verbindungen einer Schaltung aus.

- *Statische Signale*
  Die statischen Signale werden an multiplen Quellen der Logik über die `TIEOFF`-Platzhalter mittels ungenutzter LUTs zugänglich gemacht. Die Verteilung erfolgt lokal begrenzt über die Verbindungsknoten an mehrere Senken. Das Fehlermodell ist hier, dass in einem an der Quelle angeschlossenen Knoten ein Fehler auftritt und VCC zu GND oder zur Logik kurzschließt. Der Kurzschluss auf Logikleitungen hat keinen funktionalen Einfluss wenn die Logikleitung kein elektrisches Netz der Schaltung ist. Der Kurzschluss von VCC zu GND kann zum Ausfall des gesamten FPGA führen.

- *Verteilte Netze*
  Die verteilten Netze sind als Taktsignale global oder in Regionen hierarchisch verteilt, haben jeweils eine Quelle pro Taktregion und werden durch Treiber propagiert. Die globale Verteilung der Taktsignale erfolgt über dedizierte Takttreiber wie `HCLK` oder `CLK_BUFG`. Die lokale Verteilung zur Logik oder zu dedizierten Blöcken wird mittels Verbindungsknoten hergestellt. Ein Ausfall oder Jitter im Taktnetz wird durch einen fehlerhaften Takttreiber hervorgerufen. Ein Kurzschluss zu VCC oder GND ist nur in den Verbindungsknoten möglich.

Ein Überblick über die genannten Fehlerbilder stellt Tabelle 2.5 dar. Für die Fehlerdetektion ist vor allem die Ausbreitung der verteilten Netze relevant, da dadurch der maximale Testbereich für diese Verbindungen definiert wird.

2.3.3. Detektions- und Korrekturmaßnahmen

Zur Fehlerdetektion des gesamten Konfigurationsspeichers stellen die FPGA Hersteller eine periodische Signaturprüfung mit einer 32 Bit CRC Prüfsumme [Xil13b, Alt13a, Lat11] bereit. Die CRC Prüfsumme wird über den Inhalt des gesamten Konfigurationsspeichers gebildet und deckt SEUs und permanente Fehler ab, die den Zustand der Speicherzelle, die Adressierung oder die Datenleitung zum Auslesen (engl. read back) korrumpiert. Die längstmögliche Detektionszeit ist abhängig von der FPGA Größe und liegt bei heutigen SRAM-basierten FPGAs im Bereich von 8 ms bis 130 ms. Dabei werden bis zu 350 Mbit große Speicher mit 100 MHz und 32 bit Datenbreite intern gelesen. Nachteilig bei CRC Prüfungen wirkt sich die schlechte Fehlerdetektionszeit aus, die im Schlimmstfall das Doppelte der CRC-Zykluszeit betragen kann. Zudem können Fehler nicht lokalisiert werden.

Seit dem Jahr 2006 lassen sich Fehler im FPGA auch über Hamming Codes detektieren, lokalisieren und anschließend korrigieren [Xil09]. Die Fehlerdetektionszeit ist nach Gleichung 2.3 kürzer als bei einem

CRC Tests. Die Paritätsbits (engl. parity bits) des verwendeten ECCs werden für CRAM Frames des Konfigurationsspeichers berechnet und gegen die im Bitstrom übertragenen Paritätsbits verglichen [Xil13b] [Alt12]. Abbildung 2.11a zeigt die Bereiche des Konfigurationsspeichers aktueller Xilinx FPGAs. Diese haben die Größe von 101 Wörtern x 32 Bit und enthalten 13 ECC Paritätsbits [Xil13b]. Altera erzeugt für jeden Bereich eine 32 Bit Prüfsumme und zusätzliche Syndrome Bits [Alt14]. Die Hamming Distanz erlaubt bei Xilinx und Altera eine Korrektur von Einzelbitfehlern und eine Erkennung von Zweibitfehlern (engl. Single Error Correction Double Error Detection (SECDED)). Durch die in Abschnitt 2.3.1 gezeigten Maßnahmen liegt die Wahrscheinlichkeit für ein MBU mit mehr als 2 betroffenen Zellen bei ca. 2 %.

Die ECC Prüfung bei Xilinx FPGAs arbeitet mit einer Testabdeckung für SEU von 99 % unter der Voraussetzung, dass der ECC Controller fehlertolerant implementiert ist [DS09b]. Zudem zeigt die Fehlerabdeckung mittels ECC in einem Standard-RAM nach ISO 26262 ebenfalls 99 % der Fehler an [ISO11b]. Durch die hohe Fehlerabdeckung und dem Zugang zum CRAM über die interne Konfigurationsschnittstelle Internal Configuration Access Port (ICAP) verwenden viele zyklische Tests des Speichers die verfügbaren Paritätsbits. Tabelle 2.6 zeigt aktuelle Methoden der Fehlerdetektion im CRAM auf.

Tabelle 2.6.: Stand der Technik für Fehlerdetektion und Korrektur im CRAM

		Ablage Checksumme Paritätsbits	Fehlerabdeckung pro Frame	Datenwort pro Frame in bit	Lesezyklus	# Paritätsbits	Testbereich	Struktur der Detektions-IPs
LEFK	Lanuzza et al. (2009) [LZF+09]	RAM	MBU (14 bit)	256	zyklisch linear	9	Essentielle Frames	TMR Logik
LVKT	Hjortland und Chen (2006) [HC06]	RAM	SECDED	32	zyklisch linear	6	Alle Frames	Logik
	Yang et al. (2010) [YWZ10]	RAM	SECDED	11	zyklisch linear	4	Alle Frames	FrameECC & Logik
	Heiner et al. (2008) [HCW08]	Frame	SECDED	101 x 32	zyklisch linear	13	Alle Frames	FrameECC & TMR Logik
	Dutton und Stroud (2009) [DS09b]							FrameECC & Logik
	Legat et al. (2011) [LBN11]							FrameECC & Watchdog Logik
	Xilinx SEM IP (2013) [Xil13e]							FrameECC & Logik

Die Fehlerkorrektur verläuft bei allen Methoden ähnlich. Dabei wird aus den Paritätsbits die bitgenaue Position des Fehlers im Datenwort ermittelt, korrigiert und mittels ICAP zurück in den CRAM geschrieben. Die Methoden unterscheiden sich jedoch in der Größe der Datenwörter, die jeweils eine Checksumme oder eine festgelegte Anzahl von Paritätsbits überwacht. Zudem können die Paritätsbits entweder direkt im CRAM-Frame oder in nutzererzeugten RAM Strukturen abgelegt sein. Bei Lanuzza et al. (2009) [LZF+09] werden

selektiv verwendete Frames ausgelesen, sogenannte *Essentielle* Frames. Diese Eigenschaft wird zusammen mit den im Frame abgelegten Paritätsbits in dieser Arbeit weiter verfolgt und verfeinert.

Im Folgenden werden die zyklisch linearen Testvarianten als Linearer Konfigurationsspeicher Test (LKT) zusammengefasst. Aktuelle FPGAs mit über 300 MBit Konfigurationsspeicher benötigen für einen LKT mit Frame-internen Paritätsbits bis zu 70 ms. Die Zeit für die Fehlerdetektion mittels CRC und ECC sind mit den Gleichungen 2.2 und 2.3 bestimmbar. CRC-basierte Prüfungen benötigen einen vollständigen Durchlauf des kompletten CRAM zur Berechnung einer Prüfsumme. Somit werden bei einem Bitfehler unmittelbar nach dessen fehlerfreiem Auslesen noch beinahe zwei vollständige Durchläufe zur Detektion über die Prüfsumme benötigt. Folglich werden aufgrund der um die Hälfte reduzierten Testzeit nur ECC basierte Test weiter betrachtet. Hier ist t_{LKT} auch von dem Testbereich abhängig, der entweder essentielle Frames oder alle Frames beinhalten kann. Bei maximaler Testzeit werden alle internen Frames ausgelesen, unabhängig von der Ausnutzung und Beitrag zur überwachten Funktion. Ferner sind auch in Tests mit essentiellen Frames keinerlei feingranulare Priorisierungen für den Test vorgesehen. Die in Abschnitt 5 beschriebene Methode schließt die Lücke zwischen schnellen Testanforderungen und großen FPGA CRAM. Für eine bessere Anwendbarkeit in der späteren Ergebnisauswertung und durch ihre Gemeinsamkeit in der Fehlerdetektionszeit (FDZ) werden die Methoden von Hjortland und Chen (2006) [HC06], Yang et al. (2010) [YWZ10], Heiner et al. (2008) [HCW08], Dutton und Stroud (2009) [DS09b], Legat et al. (2011) [LBN11] und Xilinx SEM IP (2013) [Xil13e] als *Linearer Vollständiger Konfigurationstest (LVKT)* zusammengefasst. Die Methodik von Lanuzza et al. (2009) [LZF+09] wird unter *Linearer Essentieller Frame Konfigurationstest (LEFK)* zusammengefasst.

$$t_{CRC,Det,max} = 2 \cdot \frac{N_{SB}}{b \cdot f_{config}} \tag{2.2}$$

$$t_{ECC,Det,max} = \frac{N_{SB}}{b \cdot f_{config}} \tag{2.3}$$

$$t_{LKT} = t_{ECC,Det,max} \cdot E_{LKT} \tag{2.4}$$

$$E_{LKT} = \begin{cases} 1 & \text{für Alle Frames} \\ \frac{N_{EF}}{N_F} & \text{für Essentielle Frames} \end{cases}$$

N_{SB}	=	Anzahl statischer Konfigurationsbits (Konfigurationsspeicher - RAM)
b	=	Datenbreite für den Zugriff auf den Konfigurationsspeicher. $b_{typ} = 32$ bit.
f_{config}	=	Taktfrequenz für den Zugriff den Konfigurationsspeicher. $f_{config,typ} = 100$ MHz.
t_{LKT}	=	Testzeit für lineare zyklische Konfigurationstests
E_{LKT}	=	Faktor für essentielle Bits bei linearen zyklischen Tests
N_{EF}	=	Anzahl essentieller Frames der Schaltung
N_F	=	Anzahl aller Frames im FPGA

Neben der ECC Prüfung kann der Konfigurationsspeicher des FPGAs mit Hilfe der PDR permanent partiell überschrieben werden (engl. scrubbing) [BPP+07]. Vor allem strahlungsexponierte Applikationen wie im CERN oder in der Raumfahrt profitieren aufgrund einer hohen SER vom zyklischen Überschreiben. Zudem ist der Zugang zum Xilinx Konfigurationsspeicher (engl. ICAP) unter Kontrolle des Anwenders und wird nicht von der periodischen ECC Prüfung beansprucht. Nachteilig wirkt sich die langsame Performance von maximal 100 MB/s aus, wenn der Bitstrom im externen nichtflüchtigen Speicher verblieben ist [PDH11]. Für eine interne Speicherung des Bitstroms mit 11,5 MB genügt bei dem betrachteten Kintex 7 XC7K325T der interne Speicher nicht. Zudem kann ein zyklisches Schreiben des Konfigurationsspeichers keine Fehler detektieren und somit keine Korrektur potentieller persistenter Fehler veranlassen.

2.4. Analyse von Methoden zur Zustandswiederherstellung

Nach einer Fehlerdetektion erfolgt im Falle eines flüchtigen Fehlers in sequentiellen Schaltungen die Fehlerkorrektur. Flüchtige Fehler lassen sich flüchtige Fehler ohne Redundanzen (redundanzfrei) korrigieren. Im Gegensatz dazu benötigt ein System zur Kompensation von permanenten Fehlern eine fehlerfreie Redundanz. Die Literatur unterscheidet drei grundlegende redundanzfreie Korrekturmethoden: *Reset*, vorwärts gerichtete Korrektur (*Roll-Forward*) und rückwärts gerichtete Korrektur (*Checkpoint* und *Rollback*).

Ein Reset versetzt die Zustandsspeicher der Schaltung auf einen vordefinierten Wert, standardmäßig der Initialwert der HDL Beschreibung. Dies kann unmittelbar innerhalb eines oder mehrerer Taktzyklen durchgeführt werden. Diese einfache Methode entzieht der Schaltung jedoch jeden zeitlichen Bezug zum Gesamtsystem und kann demnach nur auf das gesamte eingebettete System angewendet werden, z.B. auf das gesamte Steuergerät eines Fahrzeugs. Für Echtzeitschaltungen bedeutet ein Reset den Verlust der Echtzeitfähigkeit bis zur erneuten Synchronisation mit der Umgebung.

Bei einer vorwärts gerichteten Korrektur wird der aktuelle Zustandsvektor oder ein zukünftig gültiger Zustandsvektor in die sequentielle Schaltung geladen. Für das Laden eines aktuellen Zustandsvektors wird eine Redundanz verwendet. So kann bei Xu & Randell (1995) [XR95] in festgelegten Perioden ein Übertrag der Zustandsdaten innerhalb eines DCR Systems bei gleichzeitiger Fehlererkennung erfolgen. Für eine redundanzfreie vorwärts gerichtete Korrektur nach Kopetz (2011) [Kop11] muss eine periodisch auftretende Einsprungstelle *g-state* ermittelt werden. Der Zustandsvektor dieser Einsprungstelle im Zustandsautomaten wird im Fehlerfall geladen und die Ausführung wird fortgesetzt. Eine Erweiterung durch Echtzeit-Umgebungsmodellierung wird bei Mikolasek & Kopetz (2011) [MK11] präsentiert, um die Periodizität zu vermeiden. Dennoch wird im Folgenden die vorwärts gerichtete Fehlerkorrektur nicht betrachtet, da eine echte redundanzfreie Methode aufgrund der benötigten Umgebungsmodelle oder doppelten Referenzzustände schwer möglich ist. Zudem sind zur Laufzeit veränderte interne Zustände der Netzliste nicht im g-state enthalten.

Die rückwärts gerichtete Korrektur (*Checkpoint* und *Rollback*) kann redundanzfrei ausgeführt werden und kann zudem während der Hochlaufphase des FPGA Systems einen vollständigen Zustandsvektor wiederherstellen. Im Folgenden wird eine kurze Einführung des Prinzips für Checkpoint und Rollback gegeben. Die Voraussetzungen für den sinnvollen Einsatz liegen bei einer deutlich größeren Ausführungszeit und Periodizität des Systems gegenüber der MTBF.

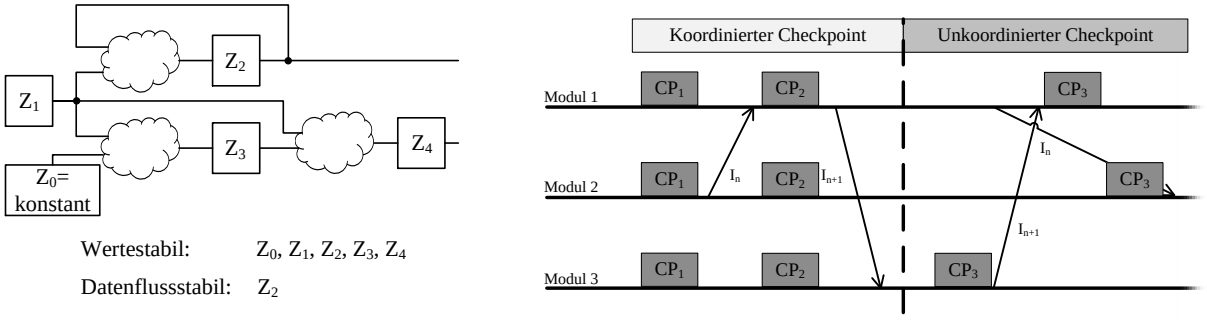
2.4.1. Varianten des Checkpoint und Rollback

Eine Übersicht der Grundbegriffe des Checkpoint (CP) und Rollback (Rb) ist bei Wolter (2008) [Wol08] und Kopetz (2011) [Kop11] gegeben. Für FPGA Schaltungen ist eine Aufteilung in *globaler* und *lokaler* Checkpoint sinnvoll. Ein globaler Checkpoint umfasst das gesamte System inklusive aller ICs oder die gesamte gekapselte Schaltung, je nach Betrachtungsgrenzen. Bei einem lokalen CP werden innerhalb der Betrachtungsgrenzen nur einige Module des Systems gesichert. In dieser Arbeit wird der gesamte FPGA als globale Betrachtungsgrenze gesehen, nicht etwa das gesamte Steuergerät. Somit entspricht ein globaler CP dem gesamten FPGA Zustandsvektor.

Wird nun während eines globalen oder lokalen CP der gesamte Zustandsvektor aller Datenpfad- und

Rückkopplungspfade erzeugt, spricht man von einem *wertestabilen* Checkpoint. Nach einem Rollback dieses Typs enthalten die Ausgangsdaten aufgrund des identischen Schaltungszustandes exakt dieselben Werte wie zum Zeitpunkt der Checkpoint Erstellung. Dagegen werden bei einem *datenflussstabilen* Checkpoint ausschließlich die Rückkopplungspfade gesichert. Hier kann es nach einem Rollback durch zeitabhängige Eingangsdaten im direkten Signalpfad zu abweichenden Ausgangsdaten gegenüber dem Checkpoint Zeitpunkt kommen. Beispielweise können in einem datenflussstabilen CP nur Filterkoeffizienten, Kalibrierdaten oder sonstige interne Pufferdaten enthalten sein. Abbildung 2.12a zeigt diese beiden Varianten auf.

In der Abbildung 2.12b wird hingegen der CP Zeitpunkt betrachtet. Im Falle eines abgestimmten Zeitpunktes der CP Erstellung mit dem Gesamtsystem wird von einem *koordinierten* Checkpoint gesprochen. Hier werden die Datenabhängigkeiten und Interaktionen `I` des Systems betrachtet, sowohl bei der Checkpoint Erstellung (Checkpointing) als auch bei einem Rollback. Anderenfalls kommt es aufgrund zeitlicher Datenabhängigkeiten zu einem Domino-Effekt bei den Zeitpunkten des Rollbacks. Bei dem Domino-Effekt werden Daten aus Interaktionen mehrfach in Checkpoints gespeichert, so etwa jeweils im Checkpoint des Senders und des Empfängers. Dies kann im Falle eines *unkoordinierten* Checkpoint vorkommen. Bei dieser Variante entscheidet jede Systemkomponente selbst über den Zeitpunkt des Checkpointing. Für Echtzeitsysteme ist diese Variante von Vorteil, da nur Teilkomponenten dem CP unterliegen können, während andere zeitkritische Module unabhängig davon agieren. So kann beispielsweise ausschließlich der FPGA Anteil eines Systems den Zustand unkoordiniert wiederherstellen.



(a) Wertestabiler und Datenflussstabiler CP Inhalt

(b) Koordinierter und Unkoordinierter CP

Abbildung 2.12.: Checkpoint Varianten für Echtzeitsysteme

Die Zeitpunkte eines unkoordinierten Checkpointing können periodisch oder interaktionsabhängig sein. Interaktionsabhängige Checkpoints bieten eine verbesserte Echtzeitfähigkeit bei größerem Hardware Mehraufwand [Wol08]. Ab dem Zeitpunkt des Checkpointing wird in vielen Anwendungen ein *Audit Trail* für eine erneute Synchronisation des CP Moduls mit der Umgebung verwendet. Dieser Audit Trail ist häufig als FIFO ausgelegt und speichert alle eingegangen Interaktionen und spielt sie nach einem Rollback ab.

Zur Behandlung der Interaktionen und Deadlines müssen für eine Fehlerkorrektur in Echtzeitsystemen die Checkpoint Zeitpunkte optimal bestimmt werden. Zusätzlich sollten der Hardware Mehraufwand, die Checkpoint Erstellzeit und die Rücksprungzeit berücksichtigt werden. Die Parameter der Checkpoint Erstellung werden bei Kwak et al. (2001) [KCK01] im Sinne der Zuverlässigkeitserhöhung betrachtet. Das mit den zusätzlichen Variablen für die Fehlerwahrscheinlichkeit aufgestellte Markov-Modell berücksichtigt jedoch nicht den Rollback und die voraussichtlichen Deadline Überschreitungen oder die Bestimmung der anwendungsspezifischen CP Strategie. Es werden jedoch im weiteren Verlauf dieser Arbeit die verwendeten CP Parameter eingeführt.

Die CP Parameter werden auch bei Koren & Krishna (2010) [KK10] angewendet, um eine geringe Deadline Überschreitung zu erreichen. Zudem werden grundlegende Zusammenhänge zwischen CP Erstellzeit und Ausführungszeit eingeführt und die Frage der optimalen Strategie in Bezug auf Periodendauer und HW Mehraufwand aufgeworfen. Es wird jedoch keine Lösungsmethode mit Einbindung der dedizierten sequentiellen Schaltungen für Echtzeitsysteme angeboten.

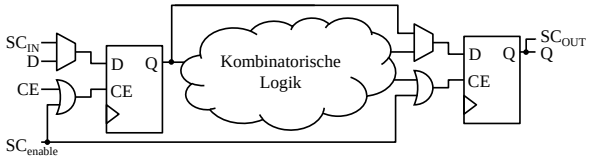
2.4.2. Methoden zur Sicherung des Zustandsvektors in sequentiellen Schaltungen

Innerhalb von eingebetteten Systemen unterscheidet die Literatur software-basierende und hardware-basierende Checkpoints. Bei Kommunikationsintensiven CPU-Anwendungen kommt CP und Rb seit den 90er Jahren zum Einsatz ([BP93], [EAWJ02]). In diesen verteilten Systemen wird vor allem der koordinierte und globale Checkpoint mit periodischen Erstellzeitpunkten auf Basis der `SafetyNet` Methode von Sorin et al. (2002) [SMHW02] eingesetzt. Zudem werden bei Nakano et al. (2006) [NMGT06] mittels der `ReViveI/O` Methode die Interaktionen der CPU an die Verzögerung durch das Checkpointing angepasst. Die vorgestellten software-basierenden CP Methoden sichern jeweils die Registersätze, Caches und Teile des Arbeitsspeichers über Befehlszugriffe.

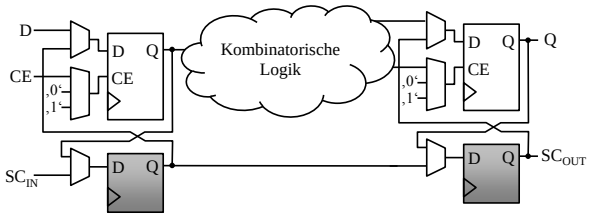
Software-basierende Methoden können bei FPGA Schaltungen nur bei Soft-Prozessoren oder Microcode Sequencern eingesetzt werden. Daher liegt der Fokus dieser Arbeit auf hardware-basierenden CP Methoden für den breiten Einsatz in sequentiellen Echtzeitschaltungen. Bei FPGA Schaltungen müssen daher FFs, BRAMs und LUT-RAMs gesichert werden. Hierfür kann bei Schmidt et al. (2011) [SHSF11] mit "Context Interface" oder bei Chan et al. (2012) [CSNSM12] mit „CpR Verilog“ der adressierbare HW Zugriff auf `signals`, `variables` oder in dedizierten Speicherinstanzen innerhalb der HDL Beschreibung implementiert werden. Diese Methoden zeigen eine hohe Nutzerfreundlichkeit bei gleichzeitigen unbestimmten CP Zeiten und ungenügender Kostenabschätzung. Alternativ zu den HDL Erweiterungen werden direkte Zugriffe auf Speicherstellen in der Netzliste platziert.

Die Netzliste weist bereits die dedizierten Speichertypen (FF, BRAM, LUT-RAM) aus und somit kann der HW Mehraufwand abgeschätzt werden. So werden etwa bei Koch et al. (2007) [KHT07] mittels des proprietären Tools „StateAccess“ die relevanten Netzlistenprimitive erkannt und durch dedizierte CP Primitiven ersetzt. Anschließend erfolgt eine erneute Synthese. Somit kann der erhöhte Hardware-Mehraufwand teilweise sehr genau bestimmt werden. Die verwendeten CP Primitive ermöglichen die Implementierung einer Scan-Chain (SC), Shadow Scan-Chain (SHC) und direkte Registeradressierung (Memory-Mapped (MM)). Diese auch für ASICs anwendbaren Methoden sind in den Abbildungen 2.13a bis 2.13c schematisch dargestellt. Aus Kostengründen wird für FFs die SC und SHC Anbindung bevorzugt, wohingegen für RAM Strukturen die direkte Speicheradressierung bevorzugt wird.

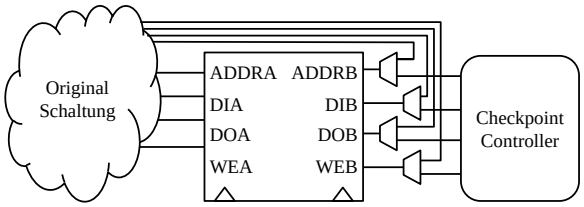
Eine Erweiterung der HW Checkpoint Methoden in FPGA zeigen Sahraoui et al. (2012) [SGBG12] mit dem Einsatz von PDR. Hierbei werden dedizierte CRAM Frames mit vorherigem Erfassen (`capture`) der FF Werte ausgelesen und im externen Speicher abgelegt, siehe Abbildung 2.13d. Diese Frames enthalten somit BRAM, LUT-RAM und FF Werte. Wobei das Erfassen der FF Werte innerhalb weniger Taktzyklen erfolgen kann und das Auslesen der RAM Blöcke sequentiell durchgeführt werden muss. Die Hardware Kosten für diese Methode sind durch einen einzelnen PDR Controller sehr gering. Jedoch kann aufgrund fehlender



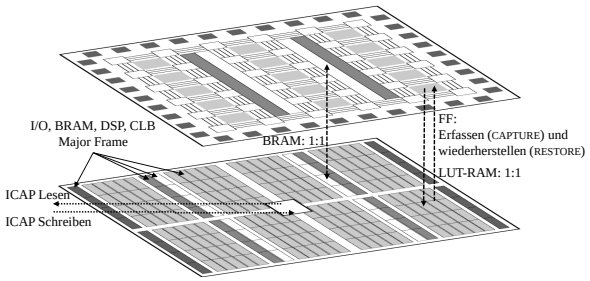
(a) Scan Chain (SC)



(b) Shadow Scan Chain (SHC) nach [KHT07] und [Lac14]



(c) Direkte Speicheradressierung



(d) Partielle Dynamische Rekonfiguration (PDR) nach [SGBG12] und [YHH+13]

Abbildung 2.13.: Schematische Darstellung der verwendeten Checkpoint Methoden

Schatten RAM Blöcke die Pausezeit der Schaltung mehrere Millisekunden betragen. Der Zustandsvektor ist in den gelesenen Frames mit einem Verhältnis von Nutzlast zu Rohdaten von etwa 0,7 % für FF und etwa 11 % für LUT-RAM enthalten (Lack (2014) [Lac14]). Dieses niedrige Verhältnis erhöht die Dauer der Checkpoint Erstellung gegenüber SC, SHC oder MM Methoden erheblich. BRAM Daten haben hingegen ein Verhältnis der Zustandsdaten zum gelesen Konfigurationsspeicher von etwa 100 %. Eine Reduzierung der nötigen Speichergröße des externen Speichers wird bei Yang et al. (2013) [YHH+13] durch bitweise Extraktion der FF Werte aus dem CRAM Datenstrom erreicht.

2.4.3. Anwendungsdomänen von Methoden der Zustandswiederherstellung

Cao & Singhal (1998) [CS98] oder auch Prakash & Singhal (1996) [PS96] zeigten die erfolgreiche Anwendung von koordinierten CP in verteilten und vernetzten Systemen in den 90er. Überdies zeigen Chen et al. in mehreren Arbeiten durch die Anwendung eines Audit Trails und adaptiver CP Frequenzen die Erhöhung der Verfügbarkeit von offenen und verteilten Systemen ([CR07], [CYR09]). Bei diesen Fällen ist die Übertragbarkeit auf eingebettete Echtzeitsysteme aufgrund des kostenintensiven Audit Trails und der hohen Latenz der Kommunikationspakete nicht gegeben.

Ein weiterer Einsatzzweck des CP und Rb ist das zeitliche Multiplexing von HW Ressourcen im Rahmen eines Kontext Wechsels. Diese Methodik zeigen Jovanovic et al. (2007) [JTW07] mit HW-basierenden Checkpoints. Im Gegensatz zur Zustandswiederherstellung nach einem Fehler erfolgt hier der Rollback zu definierten Zeitpunkten. Verletzungen der Echtzeitfähigkeit durch Deadlines der Interaktionen können bei definierten Kontext Wechseln bereits in der Entwurfsphase beachtet werden und erzeugen in der Regel keine Verletzungen zur Laufzeit. Hier sind die Übergangszeiten der Register und die Stillstandszeiten an diskreten Punkten der Laufzeit bekannt. Dies ermöglicht eine Systemauslegung in Abhängigkeit der Context-Wechsel. Im Gegensatz dazu können bei Fehlerbehandlungen keine expliziten Zeitpunkte für die Zustandswiederherstellung betrachtet werden und somit kann die Systemauslegung nur die schlechtesten zeitlichen Annahmen berücksichtigen.

3. Redundanzfreie Methoden für verfügbare und sicherheitsrelevante Echtzeit-Designs

Die in Abschnitt 2.2.1 vorgestellten Fehlermechanismen reduzieren die Verfügbarkeit und die Ausfallsicherheit des Systems und die Toleranzmaßnahmen in Abschnitt 2.2.2 wirken gegen diese Reduzierung. Im folgenden Kapitel wird analysiert, in welchem Maße die Verfügbarkeit als Qualitätsmerkmal für ein industrielles FPGA System angewendet und erhöht werden kann. Für die Anwendung in kommerziellen Produkten mit einem Volumen von 1 Mio. Stück sollen zudem nicht die bisher vorgestellten redundanz-basierten Toleranzmechanismen für die CRAM Absicherung angewendet werden.

Stattdessen werden Maßnahmen ohne zusätzliche Redundanzen (redundanzfrei) zur optimierten Fehlerdetektion im Konfigurationsspeicher und zur Zustandswiederherstellung von sequentiellen Schaltungen auf die Verfügbarkeit projiziert. Zudem wird gezeigt, wie der Ansatz der effektiven Fehlerdetektion im Konfigurationsspeicher einen Beitrag zur funktionalen Sicherheit leistet. Die gewählten Methoden sind innerhalb eines FPGAs anwendbar und müssen ja nach Sicherheitseinstufung mit architektonischen Maßnahmen im Steuergerät erweitert werden.

3.1. Abgeleitete Anforderungen an die Fehlerdetektion der Konfiguration gemäß ISO 26262

Das komplexe Thema der funktionalen Sicherheit nach ISO 26262 wird hier auf den Konfigurationsspeicher beschränkt, da fest verdrahtete Logikelemente bereits durch den Standard behandelt werden [ISO11b]. Zudem zeigen aktuelle Untersuchungen, dass auch der Entwicklungsablauf, die Logik- und Softwarepartitionierung und interne Logikredundanzen bereits der ISO 26262 genügen können [FF12]. Somit liegt der Fokus auf den Beiträgen einer effektiven Fehlerdetektion des CRAMs zur funktionalen Sicherheit. Zukünftige Systeme der Fahrerassistenz und weitere hochreaktive Systeme wie Airbag Datenauswertung sind hierfür die geeigneten Anwendungen.

3.1.1. Kurzdarstellung Standard ISO 26262 mit FPGA

Der Ende des letzten Jahrzehnts entwickelte Standard ISO 26262 stellt in 9 Bänden qualitative und quantitative Anforderungen an die funktionale Sicherheit von Fahrzeugen [ISO11a]. Die Anforderungen für FPGAs in Bezug auf die Ausfallrate der Hardware werden in Band 5 der ISO 26262 dargelegt [ISO11b]. Ein Merkmal in sicherheitsrelevante Anwendungen ist die Einhaltung der Fehlertoleranzzeit (FTZ), dargestellt in Abbildung 3.1. Innerhalb dieses Intervalls nach einem Defekteintritt muss der sichere Zustand erreicht sein. Beispielsweise müssen in der Motorsteuerung die Endstufen für die Kraftstoff-Einspritzung abgeschaltet werden oder im Notbremsassistent muss die Kraft auf die Bremsen abgebaut sein. In der FTZ sind die Fehlerdetektion und die Fehlerreaktion enthalten. Die Fehlerdetektion hängt von der Architektur und der elektrischen Schaltung ab und die Fehlerreaktion ist dabei in der Regel systemspezifisch. In dieser Arbeit wird die Fehlerdetektionszeit (FDZ) betrachtet und dabei der Fokus auf den Konfigurationsspeicher gelegt.

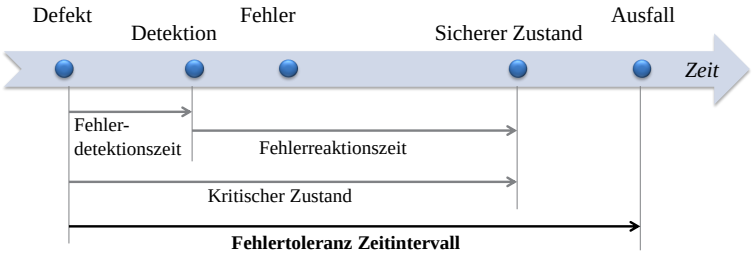


Abbildung 3.1.: Fehlertoleranzzeit nach ISO 26262

Der für die Auslegung ebenso wichtige Teil wie die FTZ ist die Unterteilung von Hardware auf die Auswirkungen im Fehlerfall. Die Auswirkungen im Fehlerfall werden durch den Automotive Safety Integrity Level (ASIL) repräsentiert. Die 4 ASIL Stufen A bis D geben sowohl die potentielle Gefahr bei einem Fehler der Hardware als auch die Güte der Anforderungen an die funktionale Sicherheit an. ASIL A repräsentiert die schwächste Anforderung an die Ausfallsicherheit und ASIL D die stärkste Anforderung. Eine Übersicht der Anforderungen in Bezug auf Erkennungswahrscheinlichkeiten von Fehlern und tolerierten Hardwarefehlern ist in Tabelle 3.1 gegeben. Die Erkennungswahrscheinlichkeit von Fehlern ist von der Maßnahme abhängig, die für bestimmte Fehlermodelle angewendet wird. Der Zusammenhang von Fehlermodell und Fehlerdetektionsmaßnahme wird mittels Failure Modes Effects and Diagnostic Analysis (FMEDA) evaluiert. In Tabelle 3.1 ist die Summe aller Fehlerabdeckungen für alle ermittelten Fehlermodelle aufgeführt. Fehlermodelle für FPGAs sind beispielsweise

Tabelle 3.1.: Anforderungen an die FPGA Hardware nach ISO 26262 Band 5

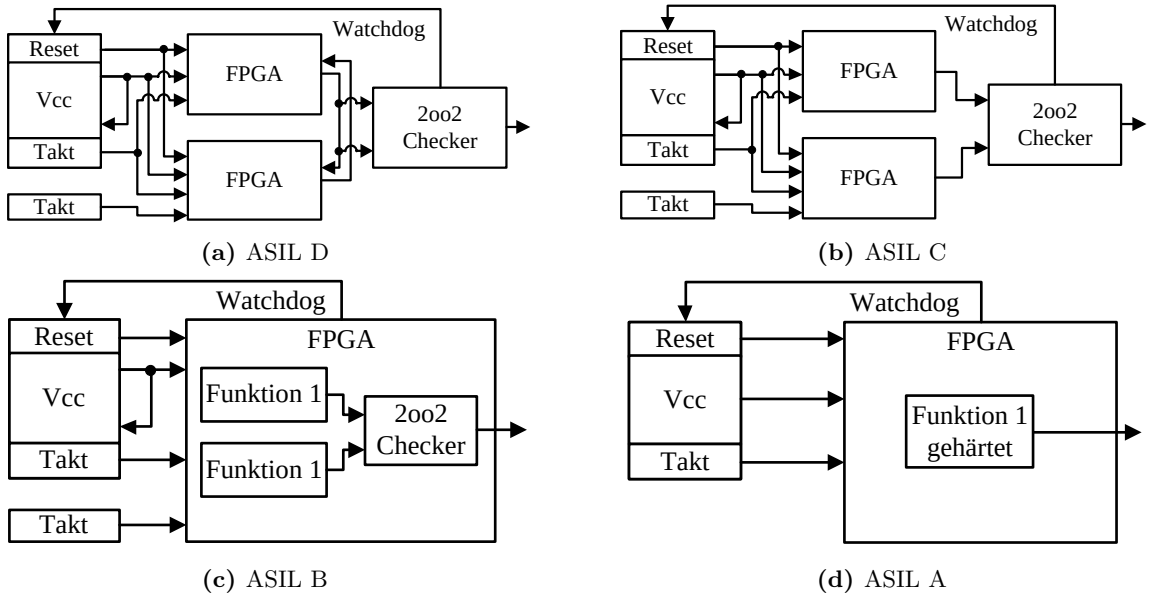
	Erkennungswahrscheinlichkeit		Zufällige Hardwarefehler
	Einzelfehler	Mehrfachfehler	in h^{-1}
ASIL D	$\geq 99\,\%$	$\geq 90\,\%$	$< 10^{-}8$
ASIL C	$\geq 97\,\%$	$\geq 80\,\%$	$< 10^{-}7$
ASIL B	$\geq 90\,\%$	$\geq 60\,\%$	$< 10^{-}7$
ASIL A	-		-

Haftfehler in der Logik, RAM Fehler und Haft- und Flüchtige Fehler im Konfigurationsspeicher.

Die Idee eines Einsatzes von FPGAs in sicherheitsrelevanten Systemen der Automobilindustrie hat seit Mitte des letzten Jahrzehnts konkrete Untersuchungen nach sich gezogen. Zum Erreichen der Einstufung nach ASIL wird in vielen Fällen der FPGA auf Board-Ebene redundant ausgelegt [BBD+08, Chu02]. Diese Art der redundanten Auslegung ist jedoch nur für ASIL C und D notwendig. In Abbildung 3.2 sind exemplarisch Architekturmöglichkeiten von FPGAs in Steuergeräten aufgezeigt. Um die Fehlerabdeckungen nach Tabelle 3.1 zu erreichen, müssen neben den eigentlichen Schaltungen noch Ein- und Ausgänge und gemeinsame Versorgungsleitungen wie Takt- und Spannungsnetz überwacht werden. Zum Erreichen der Anforderungen für latente Fehler in Form von Mehrfachfehlern müssen auch die Checkereinheiten und die Versorgungsbausteine überwacht sowie redundante Taktversorgung etabliert werden. Dennoch lassen sich bereits Funktionen mit ASIL B in einem FPGA unterbringen, insofern zusätzliche Überwachungsmaßnahmen wie Watchdog, interner Checker, externe Taktreferenz, BIST und Spannungsüberwachung einfügt werden. Die Schaltungsmodule müssen redundant und physikalisch unabhängig voneinander platziert sein. Die Unabhängigkeit kann während des Entwicklungsflusses durch Hersteller-Programme überprüft werden [Cor13] und minimiert die Wahrscheinlichkeit eines gemeinsames Fehlers für beide Module (engl. common cause). Eine weitere Abschwächung der Überwachung kann in ASIL A Systemen vorgenommen werden. Hier genügt eine Ablaufkontrolle durch einen Watchdog während die Ergebnisüberprüfung mittels Checker entfällt. Zur Absicherung der Logik kann eine logische Redundanz nach Abbildung 2.8 oder feingranulare Redundanz [NSB12] eingefügt werden. Alternativ kann ein BIST angewendet werden. In beiden Fällen genügt eine Instanz des Schaltungsmodules innerhalb des FPGAs.

3.1.2. Beurteilung der CRAM-Detektionsmethoden unter ISO 26262

In jedem System muss der Fehler vor der Korrektur detektiert werden, insofern er nicht durch das Schaltungsdesign lokal begrenzt werden kann. Da SEU und Haftfehler im Konfigurationsspeicher persistent in die Logik übergehen können, kann demnach ein zyklisches Überschreiben (Scrubbing) nicht angewendet werden. Ferner unterstützt Scrubbing kein Fehlersignal für die Logik. Somit liegt in dieser Arbeit der Fokus auf einer ECC-basierenden Fehlererkennung. Um einen ECC in sicherheitsrelevanten Anwendungen anwenden zu können, muss die Wirksamkeit seiner Fehlererkennung gegeben sein. Die zu erkennenden Fehlermodelle sind SEUs und Haftfehler in den RAM Zellen. Hierbei wird gemäß des Weißen-Rauschen Modells eine Gleichverteilung der Auftretenswahrscheinlichkeiten aller SRAM Fehlermodelle angenommen. Mit Hilfe des Rauschen Fehlermodells kann die Fehlerabdeckung nach Gleichung 3.1 berechnet werden. Dabei wird das Verhältnis von gültigen Zuständen zu allen möglichen Zuständen eines Codewortes aufgestellt. Gültige Zustände werden mittels ECC nicht als fehlerhaft erkannt und reduzieren so die Testabdeckung. Für aktuelle Xilinx FPGA liegt die Erkennungswahrscheinlichkeit bei 99,988 %. In der Wahrscheinlichkeit ist keine Gewichtung der



(a) ASIL D **(b)** ASIL C

(c) ASIL B **(d)** ASIL A

Abbildung 3.2.: Architekturvorschläge von *fail-safe* Steuergeräten mit FPGA

Einzel- und Mehrfachfehler enthalten und sie liegt im Bereich der Untersuchungen von Dutton et al. (2009) [DS09b].

Die Wahrscheinlichkeit einer erfolgreichen Fehlerkorrektur wird durch die Tatsache verkleinert, dass Mehrbitfehler fälschlicherweise als Einzelbitfehler erkannt werden und somit falsch korrigiert werden. Dadurch wird die Fehlererkennung um die Wahrscheinlichkeit für die Korrektur vermindert, die sich nach Gleichung 3.2 aus dem Verhältnis der möglichen Zustände mit Einzelbitfehler durch die Gesamtmenge der Zustände ergibt. Die Wahrscheinlichkeit für eine erfolgreiche Fehlerkorrektur mittels ECC liegt bei 60,53 % nach einem Rauschen-Fehlermodell. Es zeigt sich, dass zumindest für die Fehlererkennung ein reiner ECC ohne Zusatzmaßnahmen die Anforderungen der ISO 26262 bezüglich Fehlerabdeckung erfüllt.

$$Q_{ECC,Det} = 1 - \frac{2^k}{2^n} = 99,99\,\% \tag{3.1}$$

$$Q_{ECC,Korr} = 1 - \frac{2^k}{2^n} - \frac{2^k \cdot n}{2^n} = 60,53\,\% \tag{3.2}$$

$Q_{ECC,Det}$	=	Wahrscheinlichkeit für eine Fehlerdetektion mit ECC
$Q_{ECC,Korr}$	=	Wahrscheinlichkeit für eine Fehlerkorrektur mit ECC
n	=	Anzahl der Gesamtbits pro Wort (Nutzbits + Paritätsbits), Xilinx FPGA: 101x32 = 3232
k	=	Anzahl der Nutzbits pro Wort, Xilinx FPGA: 101 x 32 - 13 = 3219

Ein weiterer Aspekt der CRAM Fehlerdetektion ist die Detektionsgeschwindigkeit. Eine effiziente Fehlerdetektion im Konfigurationsspeicher innerhalb der gegebenen FDZ erhöht ebenso die funktionale Sicherheit wie eine hohe Testabdeckung. In allen Architekturen mit einem FPGA nach Abbildung 3.2 erlaubt eine schnelle Fehlerdetektion die Annahme einer korrekten Netzliste während der funktionalen Sicherheitsanalyse. Durch die Annahme einer korrekten Netzliste können Fehlermodelle in der FMEDA eingegrenzt werden. Ferner ist für die Berechnungen der Metriken nach ISO 26262 auch eine Architectural Vulnerability Factor (AVF) Analyse gültig.

Mit Hilfe einer schnellen Fehlerdetektion lassen sich zudem *latente* Fehler innerhalb der FDZ aufdecken.

Anderenfalls können die latenten Stellen bei einem Doppelfehler oder bei einer Übernahme in einen persistenten Zustand kritisch werden. In hoch sicherheitsrelevanten Systemen mit ASIL D sind latente Fehler zu detektieren, die bei gleichzeitigen Auftreten in den redundanten Modulen zu einer Fehlaussage des Checkers führen können. Hier unterstützt ein schneller Konfigurationstest die Detektion von latenten Fehlern.

In Systemen mit der Einstufung ASIL A und B liegt der Beitrag der CRAM Fehlerdetektion in der Einhaltung der FTZ. Die Maßnahmen aus Abschnitt 2.3 können die Anforderungen zum Teil nicht erfüllen. Vor allem die in der Fehlertoleranzzeit enthaltene FDZ stellt eine große Herausforderung für klassische linear zyklische Konfigurationstests (LKT) dar. Dennoch sind diese ECC basierten Tests kostengünstig und leicht zu implementieren. Dieser Vorteil ist bei redundanten Schaltungen nicht gegeben. Ferner sind in ASIL A Systemen Redundanzen unnötig, auch wenn eine schnelle Fehlerdetektion notwendig wird. Eine auf BIST basierende Fehlerdetektion erfüllt ebenfalls nicht die Anforderungen der FDZ in jeder potentiellen FPGA Anwendung. Notwendige Maßnahmen zur Einhaltung der FDZ des CRAM und der Sicherheitseinstufung sind in Tabelle 3.2 aufgeführt. Diese Übersicht zeigt eine mögliche Einordnung von Testmethoden für vier ASIL Stufen und für zwei Stufen von Fehlerdetektionszeiten. Es stellt sich heraus, dass in vielen Fällen ein momentan nicht verfügbarer schneller Konfigurationstest benötigt wird. Solch ein Test wird in dieser Arbeit als Priorisierter Konfigurationsspeicher Test (PKT) entwickelt und eingeführt. In Tabelle 3.2 wird die kurze Testzeit als Eigenschaft des PKT für den Vergleich herangezogen. Es wird deutlich, dass durch einen verbesserten Konfigurationstest zusätzliche Anwendungsgebiete besetzt werden können. Das Vorgehen zur Anwendung des PKT ist darüber hinaus in Abschnitt 5 erläutert.

Tabelle 3.2.: Methoden der Fehlerdetektion im Konfigurationsspeicher unter ASIL Einstufung

FDZ	ASIL A	ASIL B	ASIL C	ASIL D
$< t_{LKT}$ [1]	**PKT** für Funktion	**PKT** für Checker	2 FPGA + LKT / BIST	2 FPGA + **PKT** + BIST
$\geq t_{LKT}$ [1]	LKT	LKT / BIST	2 FPGA + LKT / BIST	2 FPGA + LKT + BIST

[1] t_{LKT} nach Gleichung 2.4

In Szenarien mit hoher Anforderung an die FDZ kann ein PKT sowohl für ASIL A und B als auch für ASIL D eingesetzt werden. Beispielsweise schreibt eine Airbag Anwendung eine FDZ von max. 500 μs vor und hat gleichzeitig eine ASIL A Einstufung. Ferner wird für eine elektronische Lenkung eine FDZ von max. 10 ms bei einer ASIL D Einstufung verlangt. In beiden Anwendungsbeispielen ermöglicht der Einsatz eines PKT die Umsetzung dieser Applikationen im FPGA mit Rücksicht auf das CRAM. Die Differenzierung der ASIL Stufen liegt in der Anwendung von Redundanzen und der zusätzliche bzw. alternative Einsatz eines BIST. Für hoch sicherheitsrelevante Systeme mit ASIL D kann ein BIST jedoch nur kostengünstig ausgeführt werden, wenn die Netzliste korrekt arbeitet. Dies wiederum kann nur durch einen PKT oder LKT geprüft werden. Für ASIL C ist ein PKT jedoch nicht nötig, da die Anforderungen an die latente Fehlerdetektion in redundanten FPGAs mit einem LKT ausreichend abgedeckt sind.

Tabelle 3.3.: Aufteilung und Ziele der Verfügbarkeit von FPGA Schaltungen

	Qualitätsmerkmal (QM)	Funktionale Sicherheit
Fehlertoleranz	nein	ja
Unabhängige Fehlererkennung	nein	ja
Schnelle Fehlererkennung	mittel	hoch
Fehlerabdeckung	mittel	hoch
Ziel	Verbesserte Nutzbarkeit im Sinne einer schnellen Fehlerkorrektur nach der Detektion	Im Fehlerfall muss das System weiter operieren (engl. fail operational).
Potential für Erhöhung der Verfügbarkeit	FPGA-Intern	FPGA-Extern

3.2. Beschreibung von Verfügbarkeit in FPGA Systemen

Nach dem Beitrag zur Fehlerdetektion im Sinne der funktionalen Sicherheit folgt in diesem Abschnitt eine Übersicht über die Erhöhung der Verfügbarkeit durch PKT und Zustandswiederherstellung. Für die hier betrachteten industriellen Anwendungen wird an dieser Stelle die Anforderung der Verfügbarkeit in zwei Kategorien unterteilt. Tabelle 3.3 zeigt die Ziele der Verfügbarkeit für die Qualität des Produkts und für die funktionale Sicherheit auf. In dieser Arbeit wird Verfügbarkeit als Qualitätsmerkmal betrachtet, da nur hier eine Einzel-FPGA Architekturlösung mit Methoden innerhalb des ICs beitragen kann. Eine hohe Verfügbarkeit im Sinne der funktionalen Sicherheit nach ISO 26262 wird in Zukunft eine redundante IC Lösung oder Lösungen mit redundanten Steuergeräten enthalten [BBD+08, Chu02].

Eine weitere Herausforderung für die Verfügbarkeit sind Methoden, die sehr kleine Kosten verursachen. In diesem Zusammenhang sind Kosten mit Mehraufwand für Hardware und höherer Latenzzeit der Echtzeitschaltung definiert. Da innerhalb von FPGAs klassische Algorithmen der Automobilbranche in Echtzeit berechnet werden können [LD10], sind diese Bausteine auch für mittlere und große Volumengrößen interessant. Daher sind vollständige Redundanzansätze zu vermeiden, insofern die Sicherheitseinstufung nach ISO 26262 dies zulässt.

3.2.1. Metrik und Bestimmungen der Verfügbarkeit

In der Literatur finden sich einige Ansätze für die Bestimmung der Verfügbarkeit [KK10]. Für die hier untersuchten FPGA Echtzeitsysteme wird die Definition nach Gleichung 3.3 angewendet. In dieser Form wird das Verhältnis von Laufzeiten und Korrekturzeiten ausschließlich auf die Basis der mittleren Zeit zwischen zwei Fehlern (MTBF) gesetzt. Diese Umformung berücksichtigt somit auch Fehler während der Korrekturzeit. Um die Einflussfaktoren auf die Verfügbarkeit detaillierter darzustellen, wird die in dieser Arbeit verwendete Definition mit Gleichung 3.4 verfeinert. Es zeigt sich deutlich der Einfluss der mittleren Fehlerdetektions- und Korrekturzeit (Mean Time To Repair MTTR). In sicherheitsrelevanten Echtzeitsystemen wird die Fehlerdetektionszeit $t_{Detektion}$ in die MTTR eingerechnet, da ab dem Zeitpunkt der Defektentstehung ein korrekter Schaltungsausgang nicht mehr vorausgesetzt werden kann. Die Zeiten t sind in Abbildung 3.3 dargestellt.

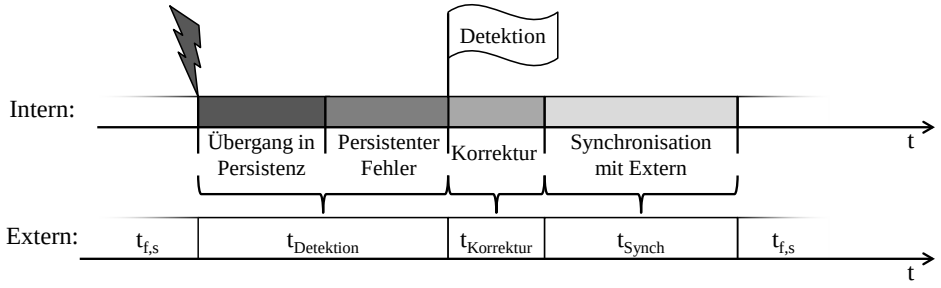


Abbildung 3.3.: Reduzierung der Verfügbarkeit im Fehlerfall

$$\text{Verfügbarkeit } A = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR} \qquad MTBF = \frac{1}{\lambda_F} \tag{3.3}$$

$$A = \frac{t_{f,s}}{t_{f,s} + N_F(\underbrace{t_{Detektion} + t_{Korrektur} + t_{Synch}}_{MTTR})} \tag{3.4}$$

MTTF	=	Mean Time To Failure - Mittlere Zeit bis zum Fehler eines funktionierenden Systems
MTBF	=	Mean Time Between Failure - Mittlere Zeit zwischen Fehlern
MTTR	=	Mean Time To Repair - Ausfallzeit über Laufzeit geteilt durch Anzahl der Fehler
$t_{f,s}$	=	Fehlerfreie und Synchrone Laufzeit
N_F	=	Anzahl der Fehler innerhalb der Laufzeit
λ_F	=	Fehlerwahrscheinlichkeit

Tabelle 3.4.: Auswirkungen der Fehlerzeiten auf die Verfügbarkeit

MTTR	1 s	100 ms	10 ms	1 ms
A[1]	99,94 %	99,994 %	99,9994 %	99,99994 %

[1] Beispiel: λ_F = 500 FIT, Stückzahl = 1 Mio. => MTBF = 0,5 h

Die Gleichungen 3.3 und 3.4 gehen dabei von der Eigenschaft aus, dass jeder Fehler zu einer Detektion, Korrektur und Synchronisation führt. Im schlechtesten Fall ist diese Annahme korrekt und muss für die automobilen eingebetteten Systeme auch angewendet werden. Für FPGAs gibt es jedoch auch modellbasierte Ansätze, die dedizierte Fehlerdetektions- und Korrekturverfahren und deren Wahrscheinlichkeiten abbilden [MG12]. Hierbei ist die Grundlage ein *Markov-Modell*, was um Umgebungsparameter, Systemparameter und Methodenparameter erweitert wird. Auch bei diesen Verfahren ist feststellbar, dass $t_{Detektion}$ und $t_{Korrektur}$ die entscheidenden Stellgrößen zur Erhöhung der Verfügbarkeit sind, da die Ausfallraten durch die Zuverlässigkeit der Halbleitertechnologie gegeben ist. Die Synchronisationszeit t_{Synch} ist in den Modellen nicht abgebildet, da dies eine Anwendungseigenschaft und weniger eine Eigenschaft des FPGAs ist. Mit den Methoden der Zustandswiederherstellung in Abschnitt 4 liegt der Fokus auf einer verringerten Zeit t_{Synch}. Der priorisierte Konfigurationsspeicher Test aus Abschnitt 5 fokussiert auf eine verringerte Zeit $t_{Detektion}$

Die Verfügbarkeit wird im Allgemeinen über die Anzahl der 9en nach dem Komma quantisiert. In Tabelle 3.4 ist für beispielhafte Zeiten der MTTR die Verfügbarkeit des Systems angegeben. Die Verfügbarkeit wird für hohe Stückzahlen mit einer CRAM typischen Fehlerrate berechnet. Es zeigt sich, dass bei sinkender MTTR mit konstanter Fehlerrate die Anzahl der 9en proportional zunimmt.

3.2.2. Beitrag der Fehlerdetektion und Zustandswiederherstellung auf die Verfügbarkeit

Beeinflussende Faktoren für die Erhöhung der Verfügbarkeit sind nach Abschnitt 3.2.1 neben der technologiebedingten Fehlerrate die Fehlerdetektion $t_{Detektion}$, Fehlerkorrektur $t_{Korrektur}$ und die Synchronisation t_{Synch} mit der Umgebung. Für die Logik gilt, dass die Fehlerkorrektur nach Stand der Technik mit einem *Reset* nur wenige Taktzyklen in Anspruch nimmt. Im Konfigurationsspeicher dauert die eigentliche Fehlerkorrektur eines aktuellen Xilinx FPGA ebenfalls nur wenige Mikrosekunden [Xil13e] unter der Annahme eines Einzelbit-Fehlers. Das Rückschreiben des korrigierten Frames nimmt etwa 1,1 μs in Anspruch und lässt sich ohne Architekturänderungen des FPGA nicht beschleunigen. Somit kann durch eine optimierte Fehlerkorrektur keine nennenswerte Verbesserung der Verfügbarkeit erreicht werden. Daher liegt der Fokus zur Erhöhung der Verfügbarkeit auf einer schnellen Fehlerdetektion $t_{Detektion}$. Zudem kann eine schnelle Wiederholung der SECDED Maßnahme dauerhaft defekte CRAM Zellen in kurzer Zeit identifizieren und anschließend Abschalt- oder Toleranzmechanismen aktivieren. Hierzu wird in dieser Arbeit ein Ansatz zum optimalen Konfigurationstest vorgestellt.

Gemäß Gleichung 3.4 trägt neben der Fehlerdetektion auch eine kurze Synchronisationszeit t_{Synch} zur hohen Verfügbarkeit bei. Die besten Ergebnisse zeigt eine vorwärts gerichtete Fehlerkorrektur aus Redundanzen heraus. Für eine redundanzfreie Fehlerkorrektur und Synchronisation ohne die Verwendung eines *g-state* wird in dieser Arbeit der Rollback verwendet. Unter Verwendung eines Rollback erhöht sich die Verfügbarkeit zum einen durch die verkürzte Hochlaufzeit der Schaltung ohne dedizierte Initialisierungsphasen. Zum anderen wird in dieser Arbeit eine CP und Rb Methodik ausschließlich in Hinblick auf eine kurze Ausführungszeit optimiert. Sie hat das Ziel, eine einzelne sequentielle und echtzeitfähige Schaltung mittels Rb unabhängig vom Gesamtsystem wiederherzustellen und somit die Synchronisationszeit auf wenige Millisekunden zu reduzieren. Im Gegensatz dazu benötigt ein Reset und die damit verbundene Einbindung aller Systemkomponenten je nach Komplexität eines Steuergerätes mehrere zehn oder hundert Millisekunden zur funktionalen Wiederherstellung.

3.3. Beschränkungen durch validierten Entwicklungsfluss

Der Fokus dieser Arbeit liegt auf Methoden, die neben kostengünstiger Anwendung auch in den industriellen Entwicklungsfluss integriert werden können. In der Entwicklungsphase für automobile Anwendungen sind dabei die ISO 26262 und die ISO 9001[1] zu beachten. Diese Normen geben vor, dass innerhalb des V-Modells der Entwicklung verifizierte und validierte Tools verwendet werden sollen, die ebenfalls einem Qualitätsmanagement unterliegen. Ist dies nicht der Fall, müssen die Resultate dieser Tools einer vollständigen Verifizierung und Validierung unterzogen werden. Für die Entwicklungsphase einer FPGA Schaltung bedeutet diese Einschränkung, dass ab der HDL Synthese ausschließlich zertifizierte Tools verwendet werden sollen. Dies stellt eine korrekte HDL Synthese sicher und die Netzliste und folgende Arbeitsschritte müssen nicht nachträglich vollständig verifiziert werden. Die Zertifizierung und Freigabe der Tools für Produkte auf Basis der angegebenen Normen wird von den FPGA Herstellern übernommen. Aus Kostengründen werden nur in Ausnahmefällen betriebsinterne Tools für den offiziellen Entwicklungsfluss zertifiziert[2]. Um diesen formalen Anforderungen zu genügen, stellen die Hersteller einige Tools für sicherheitsrelevante Schaltungen

[1] Anforderung Qualitätsmanagement. Aktuelle Ausgabe - EN ISO 9001:2008
[2] Freigabe durch zuständige Stellen, beispielsweise TÜV Rheinland

zur Verfügung. Hiermit können unter anderem isolierte Schaltungsmodule innerhalb eines FPGA erstellt [Cor13], Redundanzen hinzugefügt [Men12a] oder Lockstep Architekturen implementiert werden [Xil12a].

Für diese Arbeit folgt daraus, dass aktuell keine Veränderungen der Netzliste vorgenommen werden. Eine vollständige Verifikation nach diesem Schritt ist in den meisten Fällen nicht wirtschaftlich realisierbar. Ferner sind die Ergebnisse der Untersuchung zur Zustandswiederherstellungen unterstützende Empfehlungen, die mittels kommerzieller und zertifizierter Tools umgesetzt werden können. Auch hier sind keine Änderungen des platzierten und verdrahteten (P&R) Designs vorgesehen. Denn die bei diesem Schritt entstandenen ungewollten Fehler in Tools oder Umgebungen können nicht verifiziert werden. Dies schließt die Nutzung von Tools aus Arbeiten von Bolchini et al. [BMS11], Niknahad et al. [NSB12] oder Nazar [Naz13] nahezu aus. Diese Arbeiten basieren unter anderem auf der Veränderung des P&R Design im Xilinx Design Language (XDL) Format und verstoßen somit gegen die oben genannten kostengünstigen Entwicklungsanforderungen mit Erfüllung der ISO Normen.

Die weiteren Untersuchungen und Methoden zielen demnach darauf ab, das Schaltungsdesign in der Konzeptphase zu unterstützen. Ferner werden Sicherheitsmechanismen angepasst, die in den Grundzügen bereits durch die Hersteller qualifiziert sind. Diese Mechanismen greifen nicht aktiv in die Funktion der Schaltung ein sondern dienen der Verbesserung der Ausfallsicherheit. Solche Methoden sind nach gründlicher Verifikation zulässig. Im Falle der Partiellen Dynamischen Rekonfiguration zur Zustandswiederherstellung muss eine vollständige Verifikation und Validierung durchlaufen werden. Je nach Anwendungsfall und weiterer Nutzung von PDR kann sich hier der finanzielle Aufwand lohnen.

4. Echtzeitfähige Zustandswiederherstellung in sequentiellen Schaltungen

Im vorherigen Abschnitt wurde der Einfluss einer schnellen Zustandswiederherstellung auf die Verfügbarkeit beschrieben. In diesem Kapitel wird nun ein ganzheitliches Konzept zur Evaluierung von Methoden für die Erstellung von Zustandsabbildern (Checkpoint, CP) und Zustandswiederherstellung (Rollback, Rb) bei gegebenen (FPGA-) Schaltungen vorgestellt. Hierbei sollen die wesentlichen Fragen eines Entwicklers vor dem Einsatz von Checkpoints schnell im Sinne von kurzer Entwicklungszeit beantwortet werden. Zu diesen Fragen zählen: Wie viel Speicherplatz benötigt der Checkpoint? Wann und wie oft wird der Checkpoint erstellt? Und welche Konsequenzen hat die Einführung von Checkpoints für die zu entwickelnde Echtzeitschaltung? Die Konsequenzen können sowohl Schaltungsaufwand als auch Latenz der Ausführungszeit sein.

Zu diesem Zweck wird eine Checkpoint Virtualisierung für sequentielle Schaltungen vorgenommen. Mittels der Virtualisierung von Checkpoint Methoden sollen aufwendige Implementierungen und Tests durch simulative und analytische Verfahren ersetzt werden. Checkpoints werden virtuell in zwei verschiedene Simulationsebenen, die Netzlisten- und Anwendungssimulation, eingefügt und die Auswirkungen beobachtet. Der Fokus liegt dabei auf einer kurzen Evaluationsphase von Checkpoint Methoden, um den Entwicklungsablauf eines Produktes so wenig wie möglich zu verzögern. Zudem ist das Checkpointing in den meisten Fällen parallel oder als Konkurrenz zum Reset zu sehen und muss somit in der Evaluationszeit mit diesem schritthalten. Für die geschlossene Optimierung und Bestimmung von Checkpoint Methoden und Strategien wurde das CaRLOS Framework entwickelt und wird im Folgenden vorgestellt.

Das Kapitel beginnt mit einer Beschreibung der grundlegenden Auslegung der CP Virtualisierung mit anschließendem Überblick der Netzlisteneinbindung. Hier werden auch dedizierte Maßnahmen zur Reduzierung und Optimierung der zeitlichen Nachteile vorgestellt. Grundzüge der Netzlistenevaluation und Simulation sind bereits in [KRS13] publiziert. Anschließend werden die Auswirkungen auf die Echtzeitfähigkeit durch eine dynamische Analyse aufgezeigt. Dieses Kapitel schließt mit zwei Anwendungsergebnissen.

4.1. Grundlegende Auslegung der Checkpoint Virtualisierung

Für den erfolgreichen Einsatz von virtuellen Checkpoints in Anwendungssimulationen und Analysen werden die Suchräume und Betrachtungsgrenzen definiert. In diesem Abschnitt sind daher die Voraussetzungen für eine erfolgreiche Zustandswiederherstellung beschrieben. Zudem werden die benötigten Aspekte für eine Charakterisierung von Checkpoints in Echtzeitschaltungen eingeführt. Im Gegensatz zum Roll-Forward Verfahren mittels *g-state* nach Kopetz (2011) [Kop11] werden in dieser Arbeit Checkpoints erstellt und bei Bedarf wieder eingespielt. Dadurch lässt sich theoretisch jede sequentielle Schaltung in einen fehlerfreien Zustand versetzen und ist weitestgehend unabhängig von periodischen Zuständen. Ferner lässt sich bei komplexen Schaltungen ein regulärer g-state schwer ermitteln und im eingebetteten System hinterlegen.

4.1.1. Größe des Zustandsvektors

Unter wirtschaftlichen Aspekten werden Schaltungsredundanzen in FPGA Systemen soweit möglich vermieden. Daher wird auch in dieser Arbeit von einer Einzelinstanz der gegebenen Schaltung ausgegangen. Dies führt dazu, dass der Zustandsvektor nicht von einer benachbarten Redundanz übertragen werden kann und ein vollständiger Rollback von einer Speicherinstanz nötig wird. Ein vollständiger Rollback wird für die betrachteten komplexen Schaltungen *wertestabil* implementiert. Damit entfällt die Bestimmung der Rückkopplungspfade und es werden alle Zustandsspeicher der Schaltung berücksichtigt. Zudem kann ein wertestabiler Checkpoint für den Hochlauf eingesetzt werden.

Innerhalb des wertestabilen Checkpoints wird in jedem Fall die Bedingung einer Checkpoint-Gruppe nach Streichert et al. (2006) [SKHT06] eingehalten. Diese Bedingung besagt, dass alle Zustände und Module in einem Checkpoint enthalten sein müssen, die eine Datenabhängigkeit ausweisen. Die Gruppen selber müssen unabhängig voneinander sein. Die Unabhängigkeit der Gruppen garantiert einen interferenzfreien Rollback. Gleichzeitig beinhaltet die Checkpoint-Gruppe den gesamten Zustandsvektor eines Schaltungsmodules.

Für die betrachteten Module und Schaltungen entspricht die Checkpoint-Gruppe der gesamten Netzliste mit einem globalen Checkpoint. Die Ausweitung auf den globalen CP für FPGA Schaltungen liegt zum einen an der fehleranfälligen Netzliste im FPGA. Somit existiert keine logische Maskierung oder zuverlässige Verdrahtung für eine Abhängigkeitskette. Bei einem Fehler können demnach eigentlich unabhängige Zustandswerte korrumpiert werden. Ein weiterer Grund für den globalen Checkpoint liegt in den wenigen Signalwegen ohne Einfluss bei komplexen Schaltungen, denn die Schaltungssynthese optimiert die Netzliste in diese Richtung. Beispielsweise kann in Netzlisten von Prozessoren aufgrund der flexibel verwendbaren Architektur keine partielle Abhängigkeitskette aufgestellt werden. Somit entspricht für FPGAs mit komplexen Echtzeitschaltungen die Checkpoint-Gruppe gemäß Definition 4.1 dem kompletten Schaltungsmodul.

Definition 4.1 *Für sequentielle FPGA Schaltungen wird ein wertestabiler Checkpoint erstellt. Die Größe der Checkpoint-Gruppe entspricht einem globalen Checkpoint des gesamten Schaltungsmoduls. Somit enthält der Zustandsvektor alle Speicherelemente einer gegebenen Schaltung, insofern keine dedizierten Checkpoint-Gruppen erstellt werden.*

4.1.2. Schaltungs- und Betrachtungsgrenzen

Nach der Bestimmung der Größe des Zustandsvektors folgt eine Betrachtung der Anwendungsszenarien und Interaktionen mit dem Gesamtsystem. Wie bereits erläutert, verkürzt ein Rollback die Synchronisationszeit nach einem Fehler oder nach dem Neustart. Diese Zeit ist nur bei Schaltungen ohne kontinuierlichen Datenfluss betrachtbar. Es muss mindestens ein Rückkopplungspfad für die Bestimmung des internen Zustandes Z_{t+1} mit $Z_{t+1} = f(Z_t, E_t)$ in der Schaltung existieren (Abbildung 4.1). Eine Bestimmung des internen Schaltungszustandes direkt aus den Eingangsgrößen ohne zusätzliche Zustandsinformationen macht einen Rollback überflüssig. So sind beispielsweise Schnittstellen zu analogen Komponenten sehr eingeschränkt für einen Rollback geeignet. Die dritte Voraussetzung für die Schaltung sind eine oder mehrere komplexe Zustandsautomaten mit einer Periodendauer des gesamten Schaltungsmodules T_M deutlich über der Periodendauer des Checkpointing T_{CP}. All diese Bedingungen für einen erfolgreichen Einsatz von Checkpoint und Rollback sind in Abbildung 4.1 dargestellt. So lassen sich beispielsweise Schaltungen für Navigation mit internen Referenzpunkten, Prozessorsysteme oder Timer-Module hier wiederfinden

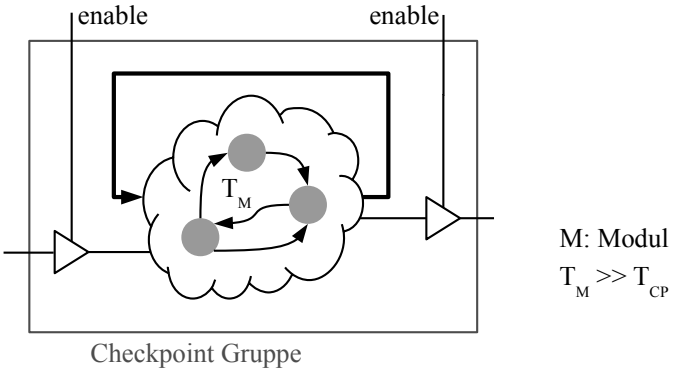


Abbildung 4.1.: Bedingungen für den Einsatz von Checkpoint und Rollback

Entspricht eine Schaltung im Echtzeitsystem den eben genannten Kriterien, dann werden die Schnittstellen der Schaltung zum System betrachtet. In dieser Arbeit werden die Interaktionen einer Schaltung mit dem System als nicht wiederholbar angenommen und entspricht damit einem Echtzeitverhalten in physikalischen Systemen. Demnach darf nach einem Rollback eine bereits gesendete Nachricht oder Signal-Propagation nicht erneut erfolgen. Diese Einschränkung führt zu einem *Unkoordinierten* Checkpoint und vermeidet gleichzeitig den Domino Effekt im Gesamtsystem. Es ist das Ziel, die FPGA Schaltung ohne Einfluss auf das Gesamtsystem mittels Rollback wiederherzustellen. In den meisten Fällen ist dieser Einfluss jedoch vorhanden und die zeitlichen Abweichungen der Interaktionen werden mittels der vorgestellten Analysen hervorgehoben.

Satz 4.2 *Für echtzeitfähige FPGA Schaltungen in eingebetteten Systemen gelten folgende Bedingungen und Definitionen für Checkpoint und Rollback:*

(1) *Die Schaltung muss die Zustandsgleichung $Z_{t+1} = f(Z_t, E_t)$ erfüllen.*

(2) *Die Datenpropagation an den Eingängen und Ausgängen muss zu diskreten Zeitpunkten erfolgen.*

(3) *Die Periodendauer des internen Zustandsvektors der Schaltung ist deutlich größer als die Periodendauer des Checkpointing T_{CP}.*

(4) *Es werden unkoordinierte, globale und wertestabile Checkpoints in einem Schaltungsmodul angewendet.*

(5) *Interaktionen sind nicht wiederherstellbar und dürfen nicht wiederholt werden.*

Erfüllt nun eine Schaltung die Punkte (1) bis (3) aus Satz 4.2, dann werden in dieser Arbeit die Effekte von Checkpoint und Rollback auf diese Schaltung untersucht und minimiert. Hierfür soll der Zustandsvektor bei geringen Kosten für die Hardware so schnell wie möglich extrahiert und abgespeichert werden. Die dedizierten Zustandsdaten sind in Speicherelementen wie FF und RAM enthalten und sind erst ab der Netzlistenebene zugreifbar. Daher erfolgen die Untersuchungen zur Erstellung eines Checkpoints auf der Netzliste. Ist die Checkpoint Methode gefunden, dann werden anhand einer Anwendungssimulation die zeitlichen Auswirkungen der Erstellung und Wiederherstellung eines Checkpoints analysiert. Diese Analyseebenen mit den Bedeutungen für Checkpoint und Rollback sind in Tabelle 4.1 aufgeführt. Zudem führt diese Arbeit als Neuheit die Simulation von Checkpoint-Methoden auf der Netzlistenebene ein.

Tabelle 4.1.: Analyseebenen für Checkpoint und Rollback

Verhaltensmodell	Netzliste
Deadline-Analyse von Interaktionen	Zugriff auf Zustandsvektor
Evaluation von Checkpoint Zeitpunkten und Häufigkeit	Kostenfunktion und Optimierung des Zugriffs
Simulation auf RTL Ebene	Simulation auf Netzlistenebene

4.2. Checkpoint-Analyse auf der Netzliste

In diesem Abschnitt werden Methoden und Evaluierungsschritte für Checkpointing in synchronen und sequentiellen Schaltungen beschrieben. Dabei soll der HW Mehraufwand für die Implementierung der Checkpoint Methode und die Beeinflussungen der Laufzeit der Schaltung minimiert werden. Somit werden die funktionalen Eigenschaften der Checkpoint Methoden untersucht. Hierfür werden die Methoden modelliert und auf die Netzliste angewendet. Anschließend kann mittels einer neuartigen Netzlisten-Simulationsmethode das Verhalten während der Erstellung und Wiederherstellung eines Checkpoints ohne hohen Entwicklungsaufwand bei der Produktentstehung evaluiert werden. Der weitere Aspekt ist die Auswahl der Checkpoint Methode anhand von mehreren Kriterien, beispielsweise HW Mehraufwand und Checkpoint Erstellzeit. Hierfür wird die Analyse der Netzliste angepasst und weiterführend mittels multikriterieller Optimierung zur Lösungsfindung angewendet.

4.2.1. Modellierung von Methoden und Schaltungen

Das Auslesen des Zustandsvektors kann nur durch dedizierten Zugriff auf die Speicherstellen FFs und RAMs erfolgen. Der Zugriff und die Modellierung erfolgt demnach ausschließlich auf der Netzlistenebene. So wird auch die Methode der PDR nach Sahraoui et al. (2012) [SGBG12] und Yang et al. (2013) [YHH+13] auf diese Ebene transformiert.

Als Basis für die Schaltungsmodellierung wird die Netzliste nach dem Standard *Electronic Design Interchange Format (EDIF)* 2 0 0[1] verwendet. Aus der Netzliste werden die Zustandsspeicher gefiltert und typisiert. Die Typisierung nach dem Typ FF, LUT-RAM oder BRAM kann nicht aus der HDL Beschreibung erfolgen. Nach der Typisierung wird die Lage der Speicherstellen im Konfigurationsspeicher bestimmt und dedizierten Major Frames zugeordnet. Dies erfolgt durch das Parsen der Xilinx XDL Layout Datei. Das XDL Format ist eine menschlich lesbare Editierung der P&R Ergebnisse bei Xilinx, die aus dem binären Native Circuit Description (NCD) Format konvertiert werden kann. Veranschaulicht wird dieses Format in Anhang A.2.2. Die Lage der FFs wird auf Slices korreliert und über die Slice-Koordinaten dem Major Frame zugeordnet. Die Lage der verwendeten RAM Blöcke kann direkt aus den platzierten RAM-Koordinaten bestimmt werden. Mittels des ermittelten Major Frames kann der entsprechende Zustandsspeicher nun über PDR ausgelesen werden.

Nach der Typisierung und der Zuordnung der Platzierungsinformation wird für die spätere Simulation ein hierarchisches Klassenabbild der Netzliste erstellt. Das Abbild besteht aus Primitiven, Ports, hierarchischen Modulen und Signalen. Das Klassendiagramm ist in Abbildung A.3 aus Anhang A.3.1 dargestellt. Mit dem so erstellten Schaltungsmodell können nun die Checkpoint Methoden in die Netzliste integriert und evaluiert werden.

In Tabelle 4.2 sind die angewendeten Checkpoint Methoden aufgeführt. Es werden nur allgemein anwendbare Methoden für FF von Koch et al. (2007) [KHT07] sowie für RAMs von Sahraoui et al. (2012) [SGBG12] und Yang et al. (2013) [YHH+13] verwendet und modelliert. Aufgrund eines sehr hohen HW Aufwandes für Echtzeitsysteme wurde die Rollback Methode mit Eingangs-FIFO für verteilte Systeme von Chan et al. (2012) [CSNSM12] nach einigen Versuchen nicht weiter verfolgt. Die Kommunikation vieler Schaltungen basiert nicht auf Protokoll-Nachrichten und erzeugt somit eine FIFO-Tiefe von mehreren kbit pro Eingang.

[1] Standard: ANSI/EIA-548-1988

Tabelle 4.2.: Universelle Checkpoint Methoden

Methode	Speicher-typ	Vorteil	Nachteil
Scan-Chain (SC)	FF	• Wenig HW Mehraufwand • Parallelität einfach möglich	• Langsame Datenrate • Direkte Werteänderung der funktionalen FF
Shadow Scan-Chain (SHC)	FF	• Lauffähig im Hintergrund • Kontrollierte Werteänderung der funktionalen FF	• Hoher HW Mehraufwand • FF mit Enable-Eingängen beachten
Direkte Adressierung (MM)	FF, RAM	• Komplette RAMs auslesbar • Hoher Datenrate	• Hoher HW Mehraufwand • Lange Haltezeit der Schaltung
PDR	FF , RAM	• Wenig HW Mehraufwand • Sehr hohe Datenrate	• Nur bei FPGAs möglich (Xilinx) • Schlechte Effizienz (FF: 0,7 %, RAM: 11 %)

Die Repräsentation der Netzliste wird um erweiterte Modelle für die Primitive von Zustandsspeichern ergänzt. In Abbildung 4.2 zeigt das FF Modell die Erweiterungen für SC, SHC und PDR. Die Variable `MODE` legt für die spätere Simulation die Eigenschaften dieser Primitive bei einem Checkpoint fest. In dieser Art und Weise werden auch RAM Primitiven modelliert. Zudem wird der notwendige HW Mehraufwand für diese Erweiterung bestimmt.

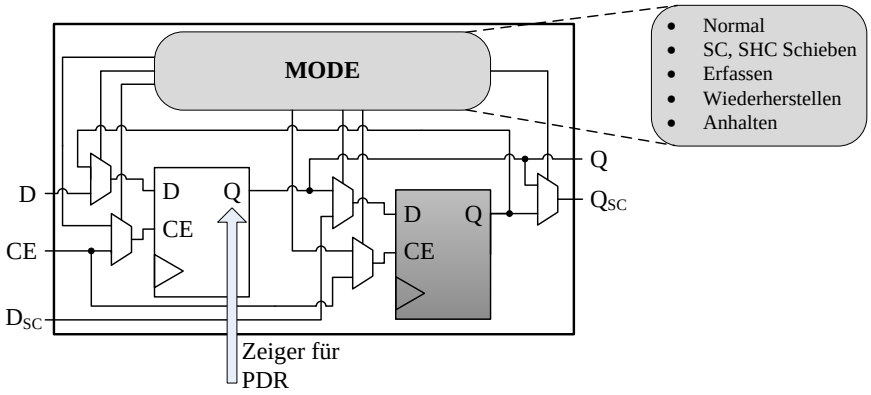


Abbildung 4.2.: Modellierung der Checkpoint Methoden im FF Modell

4.2.2. Simulation der applizierten Methoden

Das erstellte Netzlistenmodell mit den Erweiterungen für Checkpoint und Rollback wird nun in eine SystemC Simulation [2] überführt. SystemC bietet gegenüber weiteren Netzlistensimulatoren wie Mentor ModelSim®und Xilinx ISim einige Vorteile. Zum einen beinhaltet die SystemC Bibliothek den Simulator und die Bibliothek in einem und lässt sich somit in das CaRLOS Framework integrieren. Ferner erleichtern C++ Sprachkonstrukte wie Polymorphie und Vererbung die Erstellung der Checkpoint und Rollback Bibliothek aus den Standard Netzlistenelementen. Für eine Simulation mit SystemC muss zudem keine EDIF Netzliste oder Standard VHDL Bibliothek angepasst werden. Als Alternative könnte die Netzlistensimulation auch mit ModelSim®und dem

[2] Standard IEEE 1666-2005 [IEE06b], OSCI Kernel 2.3.0

Foreign Language Interface (FLI) durchgeführt werden. Hierfür müssten allerdings die herstellerspezifischen Modelle der Netzlistenprimitiven untersucht und verändert werden. Zudem muss das FLI in der Netzliste instanziiert und bei einer Variation der Checkpoint Maßnahme angepasst werden.

Da SystemC ursprünglich für die Abstraktion der Verhaltensbeschreibung auf Transaktionsebene entwickelt worden ist, muss die Leistungsfähigkeit und Effizienz bei einer Netzlistensimulation nachgewiesen werden. Hierfür wurden zwei Schaltungen synthetisiert und die Netzliste modelliert. Die Netzliste der ISCAS s38417 Schaltung hat 3685 Primitiven mit 1349 FF. Die zweite Schaltung, ein Xilinx MCS, hat 1091 Primitiven mit 414 FF und 72 RAM Instanzen. Für eine Verifikation der Simulationsdauern wurden für diese Schaltungen das *Post Synthesis* Simulationsmodell in Form einer VHDL Netzliste und das in dieser Arbeit vorgestellte Klassenmodell gemäß Anhang A.3.1 erstellt.

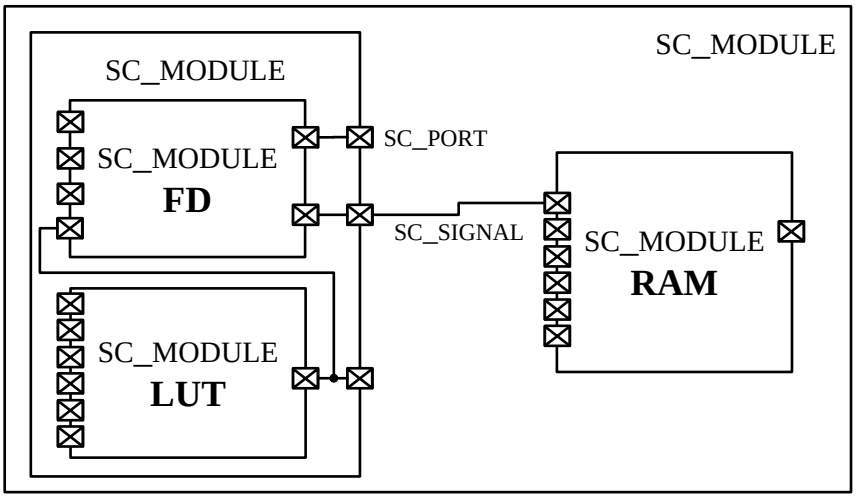
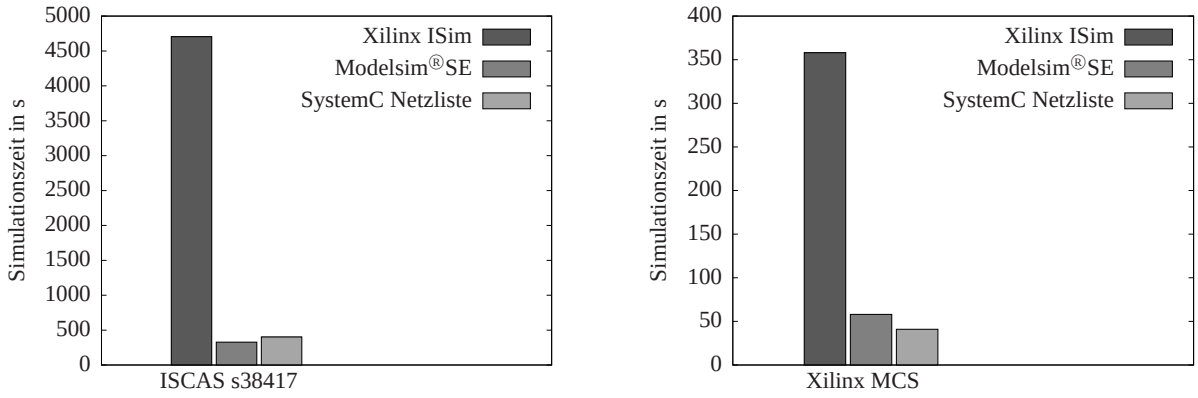


Abbildung 4.3.: Exemplarische SystemC Netzliste mit hierarchischem Aufbau

Nach der Modellierung der Netzliste wird aus jeder Primitive und aus jedem hierarchischen Netzlistenmodul ein `sc_module` erzeugt, siehe Abbildung 4.3. Anschließend wird das oberste Modul der hierarchischen Netzliste in der Methode `sc_main` instanziiert. Nach der Kompilierung mittels GCC in der Version 3.4.5 erfolgt die Ausführung auf einem Intel Core 2 Duo E8400 Prozessor 3GHz sowie 4 GiB RAM unter Windows 7 (64-Bit). Die Dauer der Simulation ist in den Abbildungen 4.4a und 4.4b dargestellt. Die in dieser Arbeit entwickelte Simulationsbibliothek der SystemC Netzliste zeigt keine erhöhten Simulationszeiten bei diesen Schaltungen. Gleichzeitig benötigt die Simulation weniger Arbeitsspeicher gegenüber ModelSim®. Die gezeigte Effizienz der SystemC Simulation der Netzlisten lässt sich mit einer reduzierten Wertegenauigkeit bei den Netzen und den speziell entwickelten Modellen für die Primitiven begründen.

Die SystemC Simulation erlaubt nun ohne eine Schaltungssynthese die schnelle Variation der Netzliste mit Checkpoint und Rollback Maßnahmen. Nach jeder Variation der Netzliste wird jedoch eine neue SystemC Kompilierung notwendig. Diese Rekompilierung kann über eine Dynamische Netzlistensimulation nach Lack (2014) [Lac14] vermieden werden. Dieser Ansatz steigert noch einmal die Effektivität der Checkpoint Simulation und verkürzt die Evaluationszeit der Zustandswiederherstellung. Hierbei liegt der Fokus auf einem zur Laufzeit erzeugten SystemC Modell innerhalb des CaRLOS Frameworks. Mittels vorkompilierten C++ Klassenbibliotheken mit Netzlisten- und Checkpointprimitiven sowie SystemC Schnittstellen kann ein Schaltungsmodell über Objektreferenzen erzeugt und die Simulation über das `Java Native Interface (JNI)` initiiert werden.



(a) ISCAS s38417, 10 ms simulierte Zeit

(b) Xilinx MCS, 1 ms simulierte Zeit

Abbildung 4.4.: Vergleich der Simulationszeiten verschiedener Netzlisten-Simulatoren mit einer Taktfrequenz der Schaltung von 100 MHz

Somit können Hochlauf-Szenarien und dedizierte Checkpoint und Rollback Zeitpunkte charakterisiert werden. Eine vollständige Anwendungssimulation hingegen ist mit der taktgenauen Netzlistensimulation mit tausenden SystemC Modulen nicht sinnvoll.

4.2.3. Analyse der applizierten Methoden

Neben der simulativen Evaluation werden in dieser Arbeit vor allem Kenngrößen der Checkpoint Methoden erzeugt. Diese Kenndaten sind teilweise bereits in der Evaluation von Kwak et al. (2001) [KCK01] enthalten und für echtzeitfähige Schaltungen angepasst. Für die Netzliste werden vier Kenndaten bestimmt. In Abbildung 4.5 sind die vier verwendeten Kenngrößen dargestellt. Eine weitere Kenngröße ist die Veränderung der maximalen Taktfrequenz der Schaltung. Dieser Parameter ist jedoch nicht innerhalb einer schnellen Evaluation bestimmbar und erfordert eine Synthese und Platzierung der Schaltung. Ferner zeigen bereits Streichert et al. (2006) [SKHT06] eine typische Verringerung der Taktfrequenz mit SC auf 95 %, mit SHC auf 85 % und mit MM auf 92 %.

- **Hardware Mehraufwand**
  Prozentualer Anteil für zusätzliche Ressourcen wie FF, Multiplexer (MUX) und LUTs von der originalen Schaltung. Die zusätzlichen Ressourcen reservieren den Platz sowohl für die Implementierung der Scan-Chains und RAM Ansteuerungen als auch für den Checkpoint- und Speichercontroller.

- **Checkpointing Dauer $\mathbf{t}_{CP}$**
  Die Zeitdauer für das Erstellen eines Checkpoints inklusive Filterung des Zustandsvektors und Abspeicherung im Checkpoint Speicher.

- **Checkpointing Haltedauer $\mathbf{t}_{CpHalt}$**
  Die Zeitdauer für den Stillstand des funktionalen Schaltungsteiles während des Checkpointing.

- **Rollback Dauer $\mathbf{t}_{Rb}$**
  Die Zeitdauer für das Wiederherstellen eines Checkpoints in allen Zustandsspeichern.

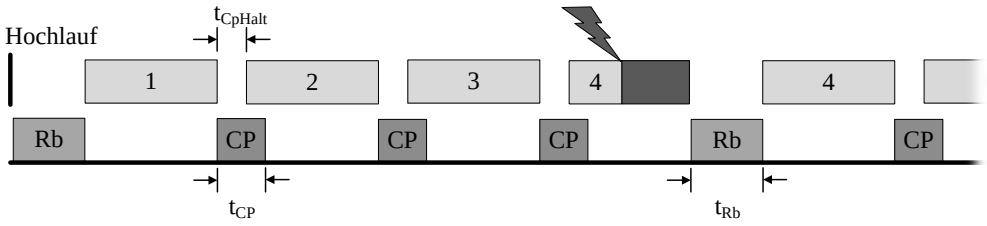


Abbildung 4.5.: Überblick über die Kenndaten des Checkpoints auf Netzlistenebene

Die Analyse der Checkpoint Methoden beginnt mit einer Zuweisung einer Methodik aus Tabelle 4.2 an die Zustandsspeicher. Wie bereits Solana & Manzano (1999) [SM99] zeigten, können mehrere SC parallel als Multi-Scan-Chain existieren. Durch paralleles Schieben der FF Werte steigt die Datenrate linear mit der Anzahl der SC. Nach der Zuweisung der Checkpoint Methode wird für jede SC der Zusatzaufwand an Steuerleitungen bestimmt. Dabei handelt es sich je nach Typ des FF um Enable `CE`, Preset (`PRE`), Clear (`CLR`), Set (`S`) und Reset (`R`) Steuersignale. Jedes dieser Signale benötigt zusätzliche Gatter in der Netzliste.

Wird mehr als ein RAM Block mit einer direkten Speicheradressierung belegt, dann muss die jeweilige Speichertiefe des RAMs beachtet werden. Das Ziel ist die optimale Nutzung des Speicherbusses zum externen Checkpoint Speicher. Daher sollen die angeschlossenen internen RAM Blöcke pro Takt die Datenrate der Speichercontroller zum externen Checkpoint Speicher bereitstellen. Dabei handelt es sich um ein Behälterproblem (Bin-Packing Problem) nach Korte und Vygen (2008) [KV08]. Dieser Ansatz wird für Checkpointing Methoden von Lack (2014) [Lac14] berücksichtigt.

4.2.3.1. Hardware Mehraufwand

Für jede Checkpoint Methode wird der Hardware Mehraufwand A_{CP} in Form von zusätzlichen FFs ($n_{FF,Cp}$) und LUTs ($n_{LUT,Cp}$) bestimmt. Für die spätere Optimierung wird dieser Mehraufwand auf die Anzahl der FF und LUTs der Originalschaltung ($n_{FF,org}$, $n_{LUT,org}$) normiert. Je nach FPGA Typ kann das Verhältnis von FF oder LUTs über Faktoren (w_{FF}, w_{LUT}) gewichtet werden, um beispielsweise die benötigte Siliziumfläche zu berücksichtigen. Die Formel 4.1 beschreibt die Berechnungsvorschrift für den Hardware Mehraufwand.

$$A_{CP} = \frac{\frac{n_{FF,Cp}}{n_{FF,org}} \cdot w_{FF} + \frac{n_{LUT,Cp}}{n_{LUT,org}} \cdot w_{LUT}}{w_{FF} + w_{LUT}} \tag{4.1}$$

Die Anzahl der zusätzlichen FFs $n_{FF,Cp}$ und LUTs $n_{LUT,Cp}$ hängt nun von der gewählten Checkpoint Methode und deren Parametern ab. Einfache SC erhalten pro FF eine LUT als Eingangsmultiplexer. Für SHC werden ein Schattenregister und zwei Multiplexer addiert. Die Steuerleitungen werden mit einer LUT pro FF implementiert. Für eine direkte Speicheradressierung müssen die Adressdekoder sowie Daten-Multiplexer vorgesehen werden. Diese Multiplexer sind mit jeweils einer LUT für Daten- und Adressleitungen vorgesehen.

Für die Checkpoint Methode der PDR wird ein fester Wert für die Ansteuerlogik angenommen, da der Zugriff zentral mittels ICAP erfolgt. Hier werden 100 FF und 100 LUTs für eine Implementierung angenommen. Die für eine Adressierung des CRAMs notwendige Adress-Datenbank der Frames ist momentan nicht in dem Mehraufwand enthalten. Ein Checkpoint Speichercontroller ist aufgrund der undefinierten externen Speicheranbindung ebenfalls nicht in den bisherigen Berechnungen des Mehraufwandes enthalten

4.2.3.2. Checkpoint und Rollback Zeiten

Die Zeiten für Checkpoint t_{Cp}, Haltezeit t_{CpHalt} und Rollback t_{Rb} werden in Anzahl der Taktzyklen berechnet. Nach der Optimierung einer Methode über das Behälter-Problem werden die einzelnen Methoden in der realen Anwendung seriell angesteuert. Somit können die Checkpoint und Rollback Zeiten der einzelnen Methoden zu einer Gesamtzeit nach Gleichung 4.2 aufaddiert werden. Die Haltezeit der Schaltung bestimmt sich aus der Gleichung 4.3, wobei hier die SHC keinen Einfluss auf die Haltezeit hat. In der Zeit für die PDR sind nur Major Frames mit LUT-RAM und BRAM inbegriffen. Die Zustände der FFs werden innerhalb einer Zeit von 10 ns bis 50 ns in den Konfigurationsspeicher kopiert. Diese Zeit wird folglich für die Haltezeit vernachlässigt.

$$t_{Cp} = t_{Rb} = t_{SC} + t_{SHC} + t_{MM} + t_{PDR} \tag{4.2}$$

$$t_{CpHalt} = t_{SC} + t_{MM} + t_{PDR,LUTRAM} + t_{PDR,BRAM} \tag{4.3}$$

t_{SC}	=	Anzahl Taktzyklen für die längste Scan-Chain
t_{SHC}	=	Anzahl Taktzyklen für die längste Shadow Scan-Chain
t_{MM}	=	Summe der Taktzyklen für das Auslesen aller RAMs mittels Direkter Speicheradressierung
t_{PDR}	=	Summe der Taktzyklen für das Auslesen des CRAM aller betroffenen Frames

Die PDR eines Major Frames der Logik wird mit etwa 3.700 Taktzyklen (bis maximal 100 MHz) gemäß Anhang A.4.2 angenommen. Durch die Anzahl von 128 statt 36 Minor Frames pro Major Frame steigt die Zeit für das Rücklesen eines BRAM Major Frames auf etwa 13.000 Taktzyklen. Die Zeit für das Auslesen der Speicher mit direkter Speicheradressierung wird aus der Anzahl dieser Blöcke und den Tiefen dieser Blöcke bestimmt. Die Summe aller Zeiten für die unterschiedlichen Frame Typen ist in den Formeln 4.2 und 4.3 enthalten.

Gleichung 4.2 zeigt die gleiche Zeit für Checkpoint t_{Cp} und Rollback t_{Rb} durch die symmetrische Auslegung der Bandbreite des Speichercontrollers an. Lese- und Schreibzugriffe auf den Checkpoint Speicher werden mit gleicher Datenrate durchgeführt. Für eine Differenzierung kann beispielsweise ein externer Flash Speicher mit einem schnellen Lesezugriff und langsamen Schreibzugriff sorgen. Ferner kann bei der PDR das Rollback länger als der Checkpoint dauern, da der Rollback auch durch Rücklesen der Frames, Modifizieren einzelner Zustandsbits und Schreiben der Frames durchgeführt werden kann.

4.2.4. Minimierung der Methoden-Auswirkungen

Die im vorherigen Abschnitt gezeigte Analyse von Kenngrößen der Checkpoint Methode wird für eine Schaltungsmodifikation angewendet. Mit dem Ziel einer effektreduzierten Zustandswiederherstellung werden nun verschiedene Methoden für das Checkpointing kombiniert. Jede Kombination erzeugt einen neuen Datensatz an Kenngrößen mit dem 4-Tupel A_{CP}, t_{Cp}, t_{CpHalt} und t_{Rb}. Diese vier Optimierungsziele sollen nun mittels Mehrzieloptimierung minimiert werden. Für die Mehrzieloptimierung wurde eine *Evolutionärer Algorithmus (EA)* aus der Reihe der genetischen Verfahren ausgewählt. Hierbei kommt ein NSGA-II Algorithmus nach Deb at al. (2002) [DPAM02] zum Einsatz. Dieser zeichnet sich durch eine schnelle Lösungsfindung mit einer guten Konvergenz aus. Die Hauptaufgabe des EA ist die Optimierung von Zielfunktionen. Im angewendeten Fall der Optimierung der Checkpoint Methode auf Netzlistenebene stellt das ermittelte 4-Tupel aus CP Kenngrößen die Werte der Zielfunktionen dar.

Ein Selektor wählt die besten Individuen anhand von Merkmalen der Zielfunktionen aus. Diese Merkmale sind die *schwache*, die *strenge* Dominanz und das Hypervolumen der Lösung im Lösungsraum. Eine Lösung dominiert die andere schwach, wenn mindestens eine Zielfunktion besser ist und alle weiteren Zielfunktionen gleich gut sind. Eine Lösung dominiert die andere streng, wenn alle Zielfunktionen besser sind. Die Auswahl über das Hypervolumen hängt vom gewählten EA ab. In dem verwendeten Algorithmus wird jedoch die Lösung mit dem kleinsten Volumen gewählt, wenn die Werte der Zielfunktionen im mehrdimensionalen Raum und der Koordinatenursprung die Eckpunkte eines Hypervolumens sind.

Nach einer Auswertung der Analyse für EA Frameworks von Parejo et al. (2011) [PRCLF12] wurde für das CaRLOS Framework das Tool "ECJ" von Luke (2013) [Luk13a] ausgewählt. Die Vorteile von ECJ liegen in der externen Konfiguration ohne Codeänderungen und in der Nutzung variabler Genomlängen. Die Einbindung des EA Frameworks in die Netzlistenanalyse ist in Abbildung 4.6 dargestellt. Innerhalb der `EA Schnittstelle` dekodiert `Dekodierer` aus dem veränderten Genom die Checkpoint Methode jedes Zustandsspeichers. Anschließend wird die im vorherigen Abschnitt beschriebene Analyse der Kenngrößen durchgeführt. Diese Kenngrößen werden in ein Array der Zielfunktionen verpackt und als `Fitness` dem EA zurückgeliefert.

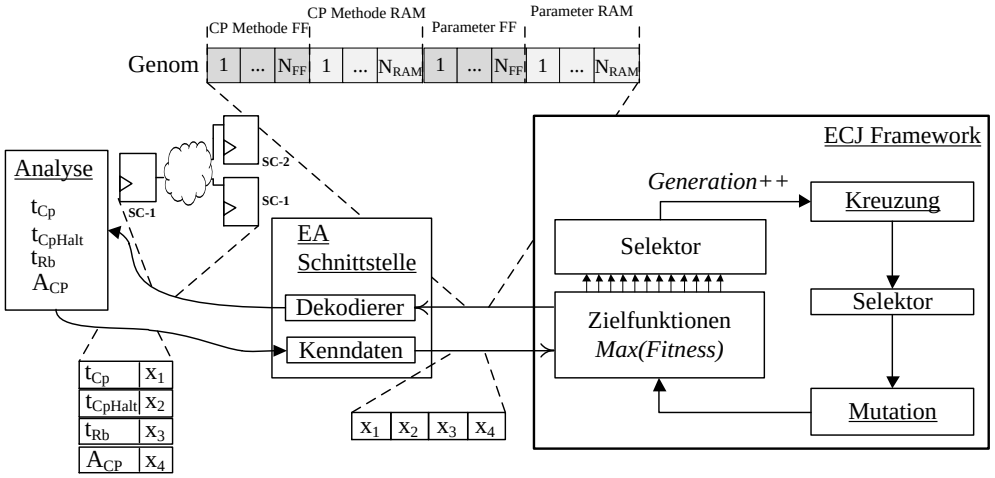


Abbildung 4.6.: Schematische Darstellung der Anbindung der Schaltungsanalyse zum Evolutionären Algorithmus

Das Genom hat eine Länge der zweifachen Speicheranzahl mit einem Datentyp Integer. Jeder Speicherzelle der Schaltung kann eine Checkpoint Methode und ein Methodenparameter zugewiesen werden. Dies ermöglicht eine maximale Flexibilität während der Mutation und Kombination der Genome. Der Methodenparameter entspricht dem Index der SC für FF bei Multi-Scan-Chains. Prinzipiell kann jeder Speicherzelle jede Checkpoint Methode aus Tabelle 4.2 zugewiesen werden. Praktische Versuche zeigten aber, dass für FF die direkte Adressierung sowie für RAMs die Scan-Chain Zuweisung nicht sinnvoll ist. Weiterhin erhalten die SC Parameter nur einen Wertebereich von 1 bis 32, da mehr als 32 SC nicht parallel an einen externen 32 bit Speichercontroller angeschlossen werden können. Für eine reale Implementierung der SC und SHC ist die Zuteilung eines identischen Index auf beide Methoden nicht möglich.

Das initiale Genom besteht unter anderem aus der Zuweisung aller FFs zu einer SC und der Zuweisung der direkten Adressierung an die RAMs. Bei jeder Iteration nach der Berechnung der Zielfunktion werden die dominanten Individuen selektiert, kombiniert und anschließend mit einer Wahrscheinlichkeit von 0,05 oder 0,1 pro Gen mutiert. Das Ergebnis des EA ist eine Checkpoint Methode für jede Speicherzelle, wobei die Kenngrößen zur HW und zeitlicher Mehraufwand minimal sind.

4.3. Dynamische Analyse der Schaltung auf der Ebene von Register Transfer Level

Im vorherigen Abschnitt wurde die optimale Checkpoint Methode auf der Netzlistenebene bestimmt. In diesem Abschnitt wird nun die Anwendung des nicht transparenten Checkpointing und Rollback in der Verhaltensebene gezeigt. Für FPGA Schaltungen in Echtzeitsystemen sind die Veränderung in den Interaktionen zum Gesamtsystem die zentrale Fragestellung. Die Betrachtungen werden daher für zeitgesteuerte Datenflüsse und Schaltungen durchgeführt (engl. Time-Triggered).

Diese Art der Datenverarbeitung beruht auf einer Entwurfsmethodik für Echtzeitsysteme, die durch Heiner und Thurner [HT98] sowie durch Kopetz und Bauer [KB03] präsentiert wurde. Grundlegend zu dieser Methodik ist das Paradigma, dass eine Funktion eine bestimmte Zielzeit (*Deadline*) und festgelegte Ausführungszeit zugesprochen bekommt, unabhängig von der tatsächlichen, kleineren, Ausführungszeit. Daraus folgt ein deterministisches Verhalten und berechenbare Verzögerungen in Regelschleifen für komplexe Echtzeitsysteme. Mit diesem Schema werden beispielsweise HW und SW Tasks, statische Videobearbeitung und Kommunikationskanäle betrieben.

Jede Interaktion erhält eine zeitliche Deadline t_D, bis zu deren Erreichen die Datenpropagation beginnen oder abschließen muss (siehe auch Buttazzo (2011) [But11]). Die Deadline wird während der Entwicklungsphase festgelegt und bestimmt somit das Timing des Gesamtsystems. Zeitlich vor der Deadline liegt die tatsächliche Ausführungszeit t_A.

4.3.1. Checkpoint Gültigkeiten

Nach dem Einfügen eines Checkpoint Mechanismus in eine zeitgesteuerte Echtzeitschaltung werden zwei Charakteristiken erstellt, die *Datengültigkeit* und die *Zeitliche Gültigkeit*. Anhand dieser Gültigkeitsaussagen werden die Checkpoint Periodendauer und Checkpoint Strategie ermittelt werden. Die Strategie bestimmt die Verteilung der Checkpoint Zeitpunkte anhand der Schaltungsaktivität oder über eine konstante Periode.

4.3.1.1. Datengültigkeit

Im Gegensatz zum Checkpointing bei verteilten Systemen nach Elnozahy et al. (2002) [EAWJ02] kann in den hier betrachteten Echtzeitsystemen eine Transmission von Daten nicht wiederholt werden. Eingehende Daten können zudem aufgrund des sehr hohen Ressourcenbedarfs nicht über einen FIFO gepuffert und über einen *Audit Trail* nach Wolter (2008) [Wol08] wiederholt werden. Somit sind die Zustandsdaten innerhalb eines konsistenten Checkpoints nach einer Sende-Interaktion und nach einer Lese-Interaktion ungültig. Diese Datenungültigkeit wird in Abbildung 4.7 dargestellt. Im Zeitraum zwischen dem Erstellen des Checkpoints und der Interaktion werden die Inhalte des Checkpoints als gültig angesehen. Nach einer Interaktion mit dem externen System wird der zuvor erzeugte Checkpoint als ungültig deklariert und muss verworfen werden. Somit wird mittels der Datengültigkeit nach Definition 4.3 die inhaltliche Anwendbarkeit ausgedrückt. Beispielsweise wird vor einer Bestätigungsnachricht (engl. acknowledge) nur in sehr seltenen Fällen ein gültiger Checkpoint existieren. Die zu bestätigende Interaktion wurde davor ausgeführt und setzt den zuvor erstellten Checkpoint auf ungültig.

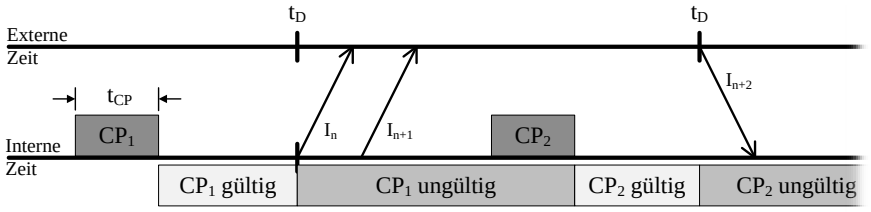


Abbildung 4.7.: Datengültigkeit: Zeitraum für die Anwendung eines Checkpoints

Definition 4.3 *Die Datengültigkeit P_D gibt den Zeitraum der Anwendbarkeit eines Checkpoints im Verhältnis zum betrachteten Zeitintervall an. Im Echtzeitsystem wird ein Checkpoint ungültig, sobald eine Sende- oder Empfangsinteraktion stattfindet.*

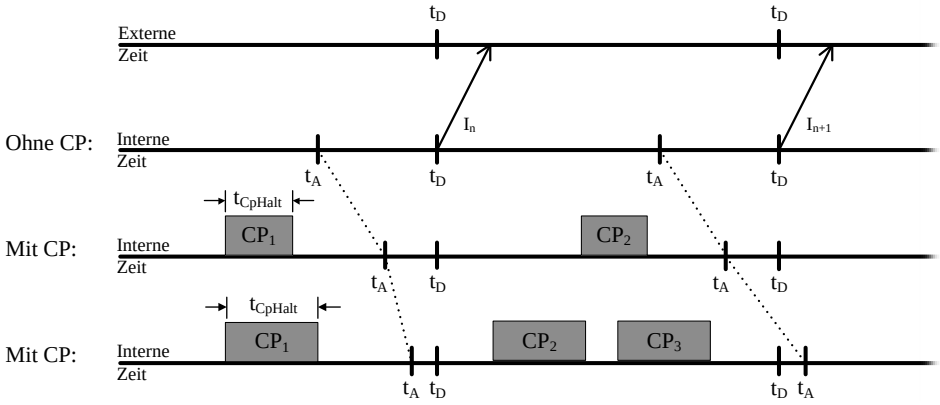
4.3.1.2. Zeitliche Gültigkeit

Die Verschiebung der Ausführungszeit t_A und ein Überschreiten von t_D wird mit der zeitlichen Gültigkeit beschrieben. Die zeitliche Gültigkeit ist eine übergeordnete Modellierung der Deadline Betrachtungen von Kopetz (2011) [Kop11] und den Anwendungen von Checkpoints in Echtzeitsystemen durch Koren & Krishna (2010) [KK10]. Bei den untersuchten Checkpoint Methoden soll ein Domino-Effekt im Gesamtsystem vermieden werden. Daher werden die Veränderungen des Timings anhand der statischen Deadlines verifiziert.

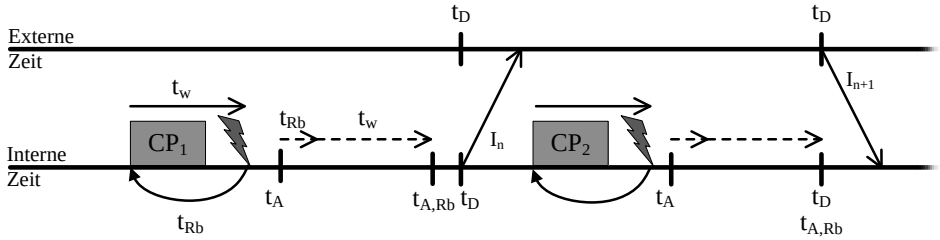
Es werden die Verzögerungen in der Ausführungszeit bei der Erstellung der Checkpoints untersucht. Abbildung 4.8a zeigt die erhöhte Ausführungszeit bei einer längeren Checkpoint Haltezeit t_{CpHalt} und bei verkleinerter Checkpoint Periodendauer. Dieses Szenario ist auch im fehlerfreien Zustand zu beachten. Hier zeigt sich nun der Vorteil des zeitgesteuerten Datenflusses mit der Differenzierung von t_A und t_D. Während die Ausführungszeit variabel an das Checkpoint Szenario angepasst werden muss, bleibt die Deadline stabil.

Im Fehlerfall wird ein Rollback mit einer Zeit t_{Rb} zum letzten gültigen Checkpoint durchgeführt. Anschließend wird die logische lokale Zeit von dem Checkpoint bis zur Fehlerdetektion erneut durchlaufen. Diese Wiederholung der Abarbeitung wird in der Zeit t_w zusammengefasst. Die tatsächlich Ausführungszeit einer Schaltung verzögert sich nun um $t_{Rb} + t_w$. In Abbildung 4.8b ist diese Verzögerung für eine Sende- und Empfangsinteraktion dargestellt. Letztendlich verlängert sich die Ausführungszeit der Funktion t_A um die aus dem Rollback resultierende Verzögerung. Problematisch kann diese Verzögerung bei zeitlich kritischen Deadlines sein, wie beispielsweise bei engen Zeitfenstern für eine Watchdog Antwort oder beim Senden und Empfangen von Empfangsbestätigungen.

Die Zeit für die wiederholte Ausführung, t_w, wird im Wesentlichen durch die Fehlerdetektionszeit FDZ nach Gleichung 4.4 bestimmt, siehe auch Abbildung 4.9. Für eine Fehlerkorrektur ohne erneutes Auftreten des gleichen Fehlers muss mindestens um die FDZ zurückgesprungen werden. Der zusätzliche Offset τ resultiert aus der Checkpoint Periode $t_{CpPeriode}$. Je nach Anwendung können somit 90 % bis 99 % der flüchtigen Fehler beseitigt werden, da diese Fehler je nach Fehlerabdeckung des Detektionsmechanismus innerhalb der FDZ erkannt werden. Tritt nach dem Rollback der Fehler erneut auf, handelt es sich entweder um einen latenten oder permanenten Fehler.



(a) Verzögerung der Ausführung durch die Erstellung des Checkpoints



(b) Doppelte Ausführung durch Rollback

Abbildung 4.8.: Zeitliche Gültigkeit bei Checkpoint und Rollback

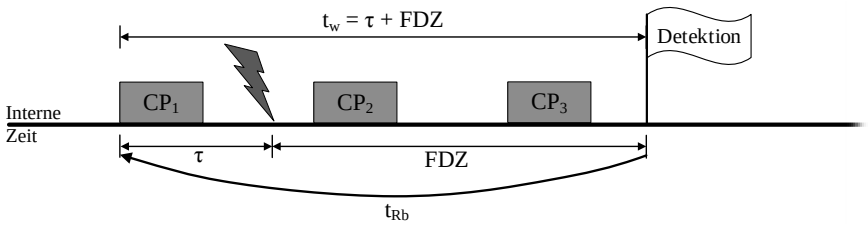


Abbildung 4.9.: Wiederholdauer t_w nach einem Rollback

$$FDZ < (t_w = \tau + FDZ) \leq t_{CpPeriode} + t_{CP} + FDZ \tag{4.4}$$

Satz 4.4 *Für eine Fehlerkorrektur durch einen Rollback muss mindestens um die Fehlerdetektionszeit (FDZ) in der logischen Ausführungszeit zurück gesprungen werden. Somit ergibt sich für einen erfolgreichen Rollback die Bedingung 4.5.*

$$t_{Rb} + FDZ \;<\; |t_D - t_A| \;\leq\; t_{Rb} + t_{CpPeriode} + t_{CP} + FDZ \tag{4.5}$$

Gemäß der Bedingung aus Satz 4.4 verzögert sich die Ausführung nach einem Rollback zum Teil erheblich. Somit kommt einer kurzen FDZ eine hohe Bedeutung für das Checkpoint Prinzip zu.

Neben der Länge der Überschreitung von Deadlines ist die Anzahl der Überschreitungen ein wesentlicher Faktor für die zeitliche Gültigkeit. In Definition 4.5 wird die Wahrscheinlichkeit P_Z für die Überschreitung von Deadlines eingeführt. Werden beispielsweise bei der CP Simulation bei genau 20 von 1000 Interaktionen

aufgrund des Checkpointing und Rollback die Deadlines überschritten, liegt P_Z bei 98 % und führt somit zu einem hohen Erfolg der CP Maßnahme.

Definition 4.5 *Die Wahrscheinlichkeit der Deadline Überschreitung durch die zeitliche Gültigkeit P_Z wird aus eins minus dem schlechtesten Verhältnis der Anzahl überschrittener Deadlines zu der gesamten Anzahl der Interaktionen im Betrachtungszeitraum ermittelt. Hier werden sowohl Überschreitungen bei der Checkpoint Erstellung als auch Überschreitungen nach einem Rollback bestimmt. Für den Rollback wird eine gleich verteilte Fehlerwahrscheinlichkeit in jeder Mikrosekunde angenommen.*

4.3.2. SystemC basierte Evaluation der Interaktionen

Die eben erläuterten Zusammenhänge zwischen Deadlines, Ausführungszeiten und Checkpoint Parametern sollen nun für Echtzeitanwendungen dediziert untersucht und schnell ausgewertet werden. Eine erste Evaluation der Checkpoint Methoden und Parameter in der realen Applikation bedeutete bisher einen sehr hohen Entwicklungsaufwand. Hier verkürzen die nun entwickelten Analyse- und Simulationsmethoden die Entwurfszeit erheblich. Es werden die Auswirkungen auf die Interaktionen bei variierender Checkpoint Periode und Auftretensstrategie aufgezeigt.

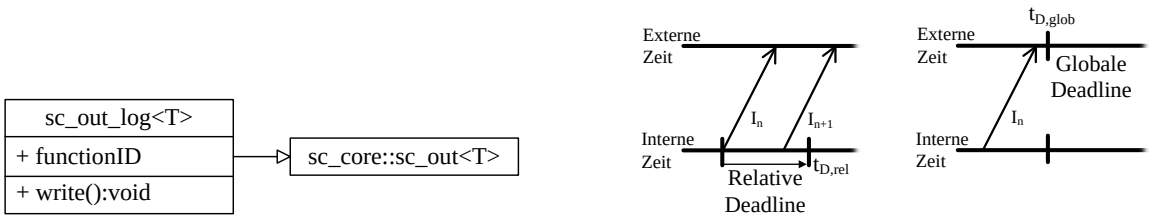
Hierfür wird eine initiale Anwendungssimulation durchgeführt. Aufgrund der Komplexität der Schaltung und der Länge der simulierten Zeit wird ein abstraktes Schaltungsmodell in einer Systemsimulation verwendet. In den aktuellen Anwendungen wird ein abstraktes SystemC[3] für die Systemsimulation verwendet. Für die spätere Analyse wird das Schaltungsmodell als I/O Modul abstrahiert und lediglich die Lese- und Schreibzugriffe auf die Ports `sc_in` und `sc_out` der obersten Hierarchiestufe registriert. Diese Zeitpunkte der Interaktionen mit dem Gesamtsystem stellen die Basis der Verhaltensanalyse dar.

4.3.2.1. Registrierung der Interaktionen und Bestimmung der Checkpoint Zeitpunkte

Die Registrierung der Interaktionen erfolgt nicht über die Trace-Datei `VCD`. Die Trace-Datei enthält lediglich Informationen über Pegelwechsel an den Ports und keine Zeitstempel über explizite Lese- und Schreibzugriffe mit gleichen und veränderten Werten. Daher ist die `VCD` Datei unzureichend für eine Nachverfolgung der Interaktionen mit dem Gesamtsystem. Stattdessen wurden die SystemC Ports um Logging Eigenschaften erweitert, so dass jeglicher Zugriff seitens der Schaltung auf Ein- und Ausgangsports mit einem Zeitstempel versehen werden kann.

Speziell die Standard-Ports `sc_out` können durch `sc_out_log` nach Abbildung 4.10a ersetzt werden. Bei einer Verwendung eines Schaltungsports mit `sc_out_log` wird jeder Schreibzugriff (`write`) mit einem Funktionsindex in eine Logging Datei geschrieben. In den Logging Einträgen entspricht der Zeitstempel des Schreibzugriffes der Ausführungszeit t_A. Zusätzlich wird noch eine dedizierte Deadline t_D für jede Interaktion registriert. Diese Deadline kann einerseits mit einem globalen Zeitstempel angegeben werden und bezieht sich somit auf den Simulationsbeginn. Andererseits kann eine relative Deadline in Bezug auf die vorherige Interaktion angegeben werden. Diese Variante ist beispielsweise bei einem Datentransfer mittels Kommunikationsprotokollen sinnvoll. Beide Deadline Varianten sind in Abbildung 4.10b dargestellt.

[3] Standard IEEE 1666-2005 [IEE06b], OSCI Kernel 2.3.0



(a) Erweiterung (Ableitung) von `sc_out`

(b) Deadline Varianten in der Simulation

Abbildung 4.10.: Zeitliche Gültigkeit bei Checkpoint und Rollback

Nach der Initialsimulation und dem Logging der Interaktionen erfolgt eine Analyse der Checkpoint Ausführungszeiten. In die Loggingdaten der Ausführungszeiten t_A werden virtuelle CP Haltezeiten t_{CpHalt} eingefügt. Die dabei verwendete Verteilungsstrategie entspricht einer Methode aus Abbildung 4.11 und kann mit der Periodendauer parametrisiert werden. Anschließend werden die Datengültigkeit und die zeitliche Gültigkeit in einem fehlerfreien Szenario analysiert. Für eine erste Beurteilung wird zusätzlich ein virtueller Fehler in die Loggingdaten eingefügt, der zum Rollback um die Zeit t_w führt. Die dafür benötigte FDZ wird vor dem Einsatz der Methode spezifiziert und die Fehlerrate liegt bei einem Fehler pro Mikrosekunde. Obwohl die CP Erstellung und der Rollback in diesem ersten Schritt virtuell ausgeführt, lassen sich Ergebnisse der zeitlichen Gültigkeit nach einem Rollback erstellen. Zudem wird die Checkpoint Strategie festgelegt und der Suchraum der Checkpoint Periode deutlich verkleinert.

(a) *Konstant Periodischer Checkpoint*
Checkpoint wird nach konstanter Periodendauer erstellt. Es gibt keine Beachtung der Interaktionen.

(b) *Interaktionsabhängiger Periodischer Checkpoint*
Checkpoint wird unmittelbar nach einer Interaktion erstellt und anschließend periodisch bis zur nächsten Interaktion fortgesetzt.

Abbildung 4.11.: Checkpoint Strategie in Abhängigkeit der Checkpoint Zeitpunkte

4.3.2.2. Simulation und Auswertung

Im nächsten Schritt werden ausgewählte Checkpoint Strategien zusammen mit der optimierten Methode aus Abschnitt 4.2 in der Register Transfer Level (RTL) Simulation auf Verhaltensebene angewendet. Hierfür werden die Checkpoint Zeitpunkte und die Haltezeit t_{CpHalt} in das Simulationsmodell mittels abschaltbarem Takt oder Enable Leitungen eingefügt. In der Simulation kommt es nun zu einer taktgenauen Verschiebung der Ausführungszeiten t_A und der relativen Deadlines im Vergleich zur Initialsimulation. Als Ergebnis lassen sich Deadline Überschreitungen durch erhöhte Haltezeiten unmittelbar erkennen. Als weiteres Ergebnis wird nun die Datengültigkeit genau evaluiert und P_D bestimmt.

Nach der Simulation bestimmt eine Analyse die Deadline Überschreitungen während des Checkpointing. Ferner werden die Überschreitungen von n Interaktionen nach einem Rollback berechnet. Die Anzahl n ist abhängig von der Anwendung und der Synchronisationsfähigkeit des Gesamtsystems. In der aktuellen Version der Evaluation liegt n zwischen 5 und 10. Während der Analyse der Rollback Eigenschaften wird in jeder Mikrosekunde eine Fehlerdetektion angenommen. Nach der Detektion wird zu einem gültigen Checkpoint zurückgesprungen, mindestens aber um die FDZ.

Zugleich wird der Wert für P_{CP} bestimmt. P_{CP} gibt die Gesamtwahrscheinlichkeit für die Anwendbarkeit von CP und Rollback für die gegebene Schaltung nach Gleichung 4.6 an. Hier werden die Wahrscheinlichkeiten der Datengültigkeit und zeitlichen Gültigkeit multipliziert und ergeben eine Metrik zur Anwendbarkeit einer CP Methode.

$$P_{CP} = P_D \cdot P_Z \tag{4.6}$$

$$= \frac{\sum_{t_{sim}} T_{CP,ung\ddot{u}ltig}}{t_{sim}} \cdot \left(1 - \frac{N_{miss,CP}}{N_I}\right) \cdot \left(1 - \frac{N_{miss,Rb}}{N_F}\right) \tag{4.7}$$

P_{CP}	=	Erfolgswahrscheinlichkeit Checkpoint und Rollback innerhalb der simulierten Zeit
P_D	=	Datengültigkeit einer Checkpoint Methode
P_Z	=	Zeitliche Gültigkeit einer Checkpoint Methode
t_{sim}	=	Simulierte Zeit und gleichzeitig Betrachtungsintervall
$T_{CP,ung\ddot{u}ltig}$	=	Intervall mit Datenungültigkeit
N_{miss}	=	Anzahl der Deadline Überschreitungen von Interaktionen bei Checkpointing und Rollback
N_I	=	Gesamtzahl der Interaktionen im Betrachtungsintervall
N_F	=	Gesamtzahl der möglichen Fehlerzeitpunkte mit einer definierten Defektperiode von 1 μs (Entspricht somit einem Jitter in der FDZ von 1 μs)

Die Werte von P_{CP} werden zusammen mit den Verschiebungen von t_A dem Entwickler aufgezeigt. Anhand dieser Ergebnisse muss ein manueller Prozess über die Abnahme der gewählten Methode entscheiden. Bei einer Ablehnung können nun erneute Simulationen mit veränderten Parametern untersucht werden. Der gesamte Ablauf der bisher beschriebenen hybriden Checkpoint Evaluation in der Netzliste und auf der Verhaltensebene ist noch einmal in Abbildung 4.12 zusammengefasst.

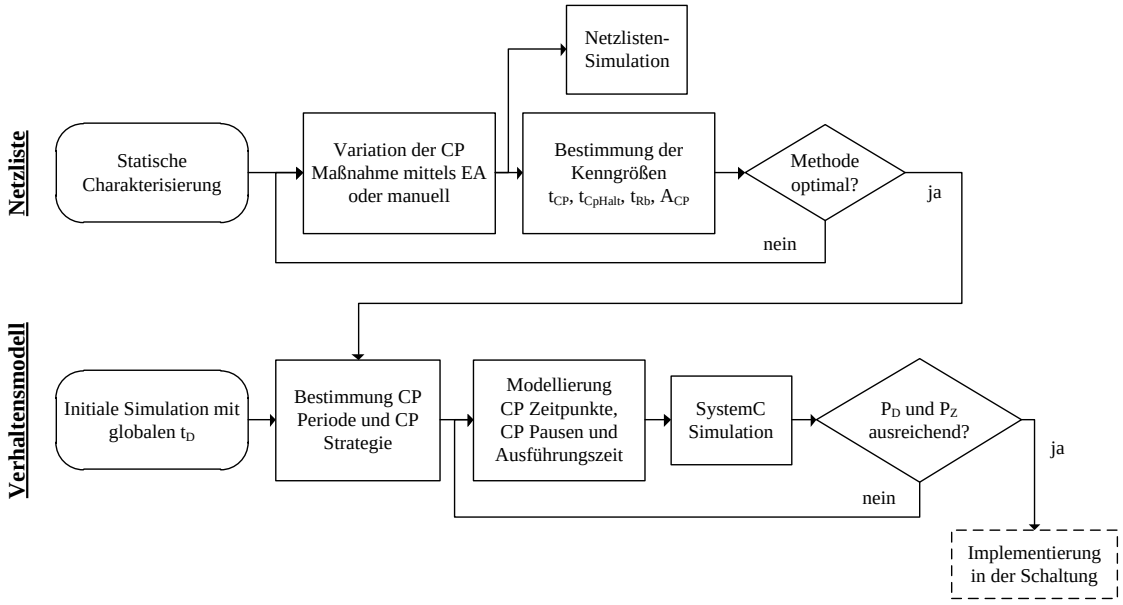


Abbildung 4.12.: Ablauf der Checkpoint Evaluation für FPGA Echtzeitsysteme

4.4. Analysetool und Anwendungsergebnisse für ein Echtzeitsystem

Für die hybride Analyse von Checkpoint Maßnahmen in Echtzeitsystemen aus den Abschnitten 4.2 und 4.3 wurde ein Java Tool entwickelt. Dieses Tool soll Entwickler bei der schnellen Beurteilung der Stärken und Schwächen von Checkpoints in den zu entwickelnden FPGA Schaltungen unterstützen. Ein Blockschaltbild des als CaRLOS bezeichneten Frameworks ist in Abbildung 4.13 zu sehen. Das Java Tool unterteilt sich in jeweils ein Modul für die Netzlistenanalyse und die Anwendungsanalyse. Jedes Modul verwendet eine eigene CP Bibliothek aufgrund unterschiedlicher Abstraktionsebenen. Die beiden Teile des Checkpoint Tools tauschen lediglich die Methodenparameter aus Abschnitt 4.2.3 aus und arbeiten daher unabhängig voneinander.

Zusammen mit dem EDIF Parser "BYU EDIF Tool" der Brigham Young Universität [Uni] und dem XDL Layout Parseranteil des Frameworks "RapidSmith" [LPL+10] wird die Schaltung, die FF- und die BRAM-Lage im CRAM evaluiert. Des Weiteren verwendet die Verhaltenssimulation neben zusätzlichen CP Bibliotheken die in dieser Arbeit entstandene Bibliothek `InteractionMonitor` als SystemC Erweiterung für das Logging.

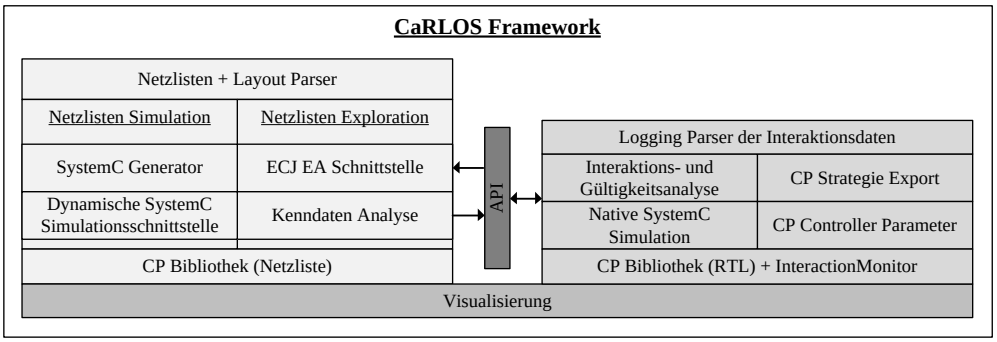


Abbildung 4.13.: Blockschaltbild des Analysetools CaRLOS zur hybriden Evaluation von Checkpoint Methoden in Schaltungen

Anhand von zwei Schaltungen soll diese Checkpoint Evaluation gezeigt werden. Zum einen wird die Netzlistenanalyse und Simulation an einem Xilinx MicroBlaze™Microcontroller System (MCS) System beschrieben. Zum anderen erfolgt die Methodenoptimierung und Verhaltensanalyse an einem industriellen Echtzeit Timermodul.

4.4.1. Anwendung 1: Microcontroller System

Der implementierte Controller MicroBlaze MCS besteht aus einem 32bit Prozessorsystem mit einem Daten- und Instruktionsspeicher von 32KiB sowie einem Timer, UART und I/O Peripherieeinheiten [Xil13d]. In der Anwendung wird ein FreeRTOS mit der Version 7.5.3 implementiert [Ltd14]. Die Schaltung benötigt 414 FF, 64 LUT-RAM (32 Bit $\times$ 1), 51 Schieberegister (16 Bit) sowie 8 BRAM (32 KiBit). Nach dem Hochlauf des FPGAs benötigt die Initialisierung des Betriebssystems und der Start einer Task etwa 260 μs.

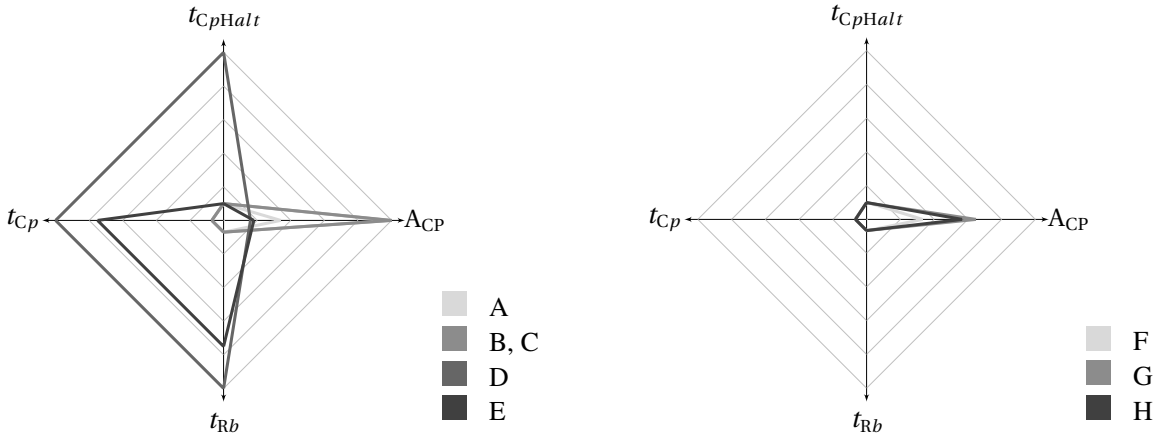
4.4.1.1. Netzlistenanalyse

An dieser Schaltung wird die CP Methodenanalyse und Netzlistensimulation gezeigt. Nach dem Parsen der Schaltung und der Modellierung der Netzliste werden die Methoden händisch und mittels EA evaluiert. Hierbei wird für den BRAM und LUT-RAM jeweils PDR und direkte Speicheradressierung MM als Methode angewendet. Für die FF werden Scan-Chain SC, Shadow Scan-Chain SHC und PDR als Option geführt. Bei dieser Anwendung wurden für FF eine und zehn parallele SC aufgebaut. Die für jeweils einen Parameter optimalen Lösungen sind in Tabelle 4.3 aufgeführt und die ausführliche Ergebnisliste ist in Tabelle A.1 im Anhang A.3.2 zu finden. Abbildung 4.14a zeigt die aggregierten Ergebnisse der manuellen Analyse, normiert auf den jeweils schlechtesten Wert der einzelnen Achsen. An diesem Ergebnis ist eine sehr starke Heterogenität der Werte aufgrund der Größe des Lösungsraumes zu erkennen. Dahingegen zeigt das Ergebnis aus dem Evolutionären Algorithmus eine deutliche Tendenz mit geringer Streuung bei den Resultaten, dargestellt in Abbildung 4.14b. Beide Graphen der Abbildung 4.14 sind auf gleiche Maximalwerte normiert.

Tabelle 4.3.: Zusammenfassung der Ergebnisse aus der Netzlistenanalyse des MicroBlaze MCS

FF	BRAM	LUT-RAM	A_{CP}	t_{CpHalt} [μs]	t_{Cp} [μs]	t_{Rb} [μs]	Lösungsindex
10× SC	PDR	MM	0,446	83,7	**83,7**	**83,7**	A
1× SHC	MM	MM	1,311	**82,9**	87,0	87,0	B
10× SHC	MM	MM	1,318	**82,9**	**83,7**	**83,7**	C
PDR	PDR	PDR	**0,211**	845,0	1285,0	1285,0	D
	MM	MM	0,241	**82,9**	962,9	962,9	E
EA			0,438	87,0	87,0	87,0	F
FF: SC, SHC			0,854	83,5	84,1	84,1	G
RAM: MM, PDR			0,745	83,7	84,0	84,0	H

Für die gegebene Schaltung existiert keine eindeutig optimale Lösung. Vielmehr zeigen die Werte eine Checkpoint Erstellzeit t_{Cp}von etwa 83 μs bis 87 μs für eine gute Lösung an. Ebenso konstant zeigt sich die Checkpoint Haltezeit t_{CpHalt}. Jeder minimalen Verkleinerung dieser zwei Werte folgt ein zum Teil deutlich erhöhter Hardware Mehraufwand, der im Bereich von 21 % bis 130 % liegt. Nun kann je nach Auslegung des Gesamtsystems der Kostenvorteil oder der Zeitvorteil optimiert werden.



(a) Übersicht der besten manuell bestimmten Lösungen **(b)** Übersicht der besten Pareto-Optimalen Lösungen

Abbildung 4.14.: Übersicht der Checkpoint Methoden des MCS

Mittels des Rollback durch die Methoden A bis C und G bis H lässt sich die Hochlaufzeit des Systems von 260 μs auf 84 μs reduzieren. Hierfür wird direkt nach dem ersten Takt der Rollback mittels direkter Speicheradressierung des RAMs und SC gestartet. Somit kann nach kurzer Evaluationsphase die Hochlaufzeit um Faktor 3 verkürzt werden. Eine Fehlerkorrektur nach einem flüchtigen Fehler im Zustandsvektor der Schaltung ist ebenfalls mit etwa 84 μs möglich.

Die händische Evaluation wurde innerhalb von einer Stunde durchgeführt. Die Lösungsfindung mittels EA wurde bei 500 Individuen mit 4000 Generationen ebenfalls innerhalb einer Stunde abgeschlossen. Die konkrete Umsetzung der Checkpoint Methode in der Netzliste kann anschließend beispielsweise mit dem Tool "StateAccess" von Koch et al. (2007) [KHT07] innerhalb der Netzliste erfolgen und ist nicht Teil dieser Arbeit. Für vergleichbare Ergebnisse mit Methoden in der VHDL Beschreibung bei anschließender Synthese nach Koch et al. (2007) [KHT07], Chan et al. (2012) [CSNSM12] und Gong & Diessel (2011) [GD11] werden etwa 8 Stunden benötigt. Danach kann auch hier der HW Mehraufwand abgeschätzt oder mittels Simulation die Zeitanalyse durchgeführt werden.

4.4.1.2. Netzlistensimulation mit Checkpoint Methoden

Noch gravierender fällt der Zeitgewinn in der Simulation der Checkpoint Methoden aus. Für den Nachweis der Funktionalität einer PDR basierten Lösung wurden bisher mit "ReSim" nach Gong und Diessel (2011) [GD11] mehrere Stunden funktionale Verifikation benötigt. Mit dem Fokus auf Checkpointing und Anwendung der PDR auf Netzlistenebene wird für diese Schaltung ein Checkpointing und Rollback mit dem CaRLOS Framework innerhalb von 30 Minuten simuliert. Dies wird durch die dynamische Netzlistensimulation aus Abschnitt 4.2.2 ermöglicht.

Während der Evaluation der Checkpoint Methoden muss das Schaltungsmodell nach einer Variation der Methodik neu erstellt werden. In den klassischen Simulationsverfahren mittels nativem SystemC, Modelsim®und Xilinx ISim erfolgt demnach eine erneute Kompilierung oder Synthese der Schaltung und verzögert dadurch die Entwicklungsphase. Daher wird für die MCS Schaltung die hier entwickelte dynamische Netzlistensimulation gewählt. Die eigentliche Ausführungszeit der Simulation wurde bereits in Abschnitt 4.2.2 dargelegt und zeigt eine konkurrenzfähige Durchführungszeit der SystemC Netzlistensimulation. Der zeitliche Ablauf eines kompletten Evaluationszyklus ist in Tabelle 4.4 eingetragen. Die Checkpoint Evaluation gliedert sich in die Erstellung des Schaltungsmodells, die Kompilierung und die eigentliche Simulation mit 2 ms simulierter Zeit. Das Einfügen der Checkpoint Maßnahme am User Interface oder im VHDL ist hier nicht berücksichtigt.

Tabelle 4.4.: Vergleich der nativen und dynamischen SystemC Netzlistensimulationen am Beispiel MCS

Simulationsumgebung[1]	Kompilierung [s]	Modellaufbau [s]	Simulation [s]	GESAMT
Native SystemC Simulation (-O3)	125,2	0,08	47	**172,3**
Native SystemC Simulation (-O0)	11,6	0,06	57,3	**68,9**
Dynamische SystemC Simulation	0	0,32	50,1	**50,4**

[1] 2x Intel Xeon E5620 64-bit, 8 x 2,4 GHz, 96 GiB RAM, Ubuntu 12.04.4 LTS, GCC 4.6.3

Wie erwartet zeigt die dynamische Simulation eine langsamere Geschwindigkeit gegenüber dem optimierten (-O3 Compilereinstellung) und kompilierten Schaltungsmodell. Es entfällt jedoch die Kompilierzeit und

der Modellaufbau ist vernachlässigbar klein. Im Vergleich zur nicht optimierten (-O0) klassischen SystemC Simulation zeigt die dynamische Ausführung in jedem Fall eine verbesserte Geschwindigkeit. Nach der Netzlistensimulation kann die `VCD` Datei ausgewertet werden. Das Ergebnis einer Simulation ist im Anhang in Abbildung A.4 zu finden.

Ein Vergleich der Evaluationsmethoden für Checkpointing und Rollback zeigt den Geschwindigkeitsvorteil des CaRLOS Frameworks gegenüber den Methoden aus der Literatur. In Abbildung 4.15 sind die Zeiten der Modellbildung bzw. Kodierung der Methodik, der Synthese oder Kompilierung und die Zeiten der Simulation aufgeführt. Die angegebenen Zeiten sind experimentelle Erfahrungen und Abschätzungen. Die Werte gelten für die modellierte Anwendung genau einer Checkpoint Methode auf die Schaltung bis zur Bestimmung der Kenngrößen inklusive zyklenakkuraten simulativen Nachweis. Dabei werden die Tools “StateAccess” von Koch et al. (2007) [KHT07] und “CpR Verilog” von Chan et al. (2012) [CSNSM12] für HDL- und Netzlisten Variationen angewendet. Die Bibliothek “ReSim” von Gong & Diessel (2011) [GD11] wird für die Evaluation der PDR Checkpoint Methodik zum Vergleich herangezogen. Das in dieser Arbeit entwickelte CaRLOS Tool zeigt durch die schnelle Modellbildung und die fehlende Kompilierzeit eine deutliche Verringerung des zeitlichen Entwicklungsaufwandes. Die tatsächliche Simulation mit SystemC zeigt keine längere Laufzeit gegenüber den klassischen Netzlisten Simulatoren. Somit können etwa achtmal so viele Iterationen von Checkpoint Methoden in einer Evaluationsphase durchlaufen werden.

Im Gegensatz zu den bisherigen Methoden basiert das CaRLOS Framework auf einem geschlossenen objektorientierten Modell mit angepasster Simulationsbibliothek. Bei einer Rücktransformation in das Netzlistenformat EDIF muss nun Zeitaufwand für Konvertierung und Verifikation berücksichtigt werden. Dieser Aufwand ist in Abbildung 4.15 nicht enthalten. Im Vergleich zu dem Checkpoint Evaluationsaufwand ist die Zeit für die Rücktransformation jedoch vernachlässigbar.

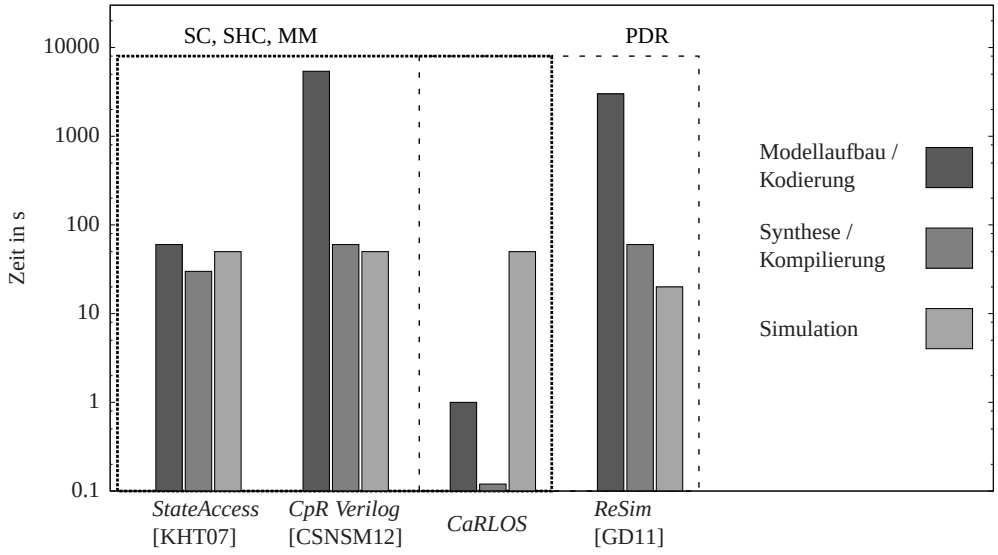
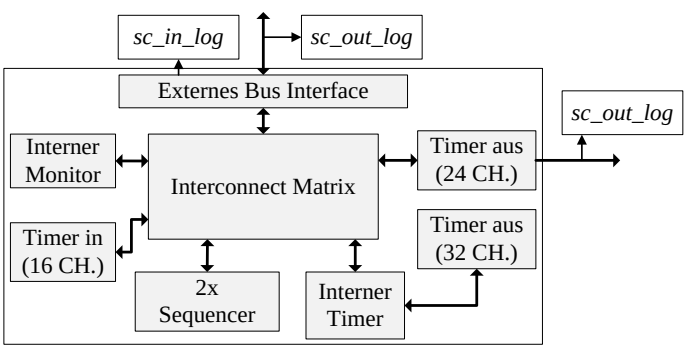


Abbildung 4.15.: Vergleich der Evaluationszeiten verschiedener Tools für eine Checkpoint Methode am Beispiel MCS

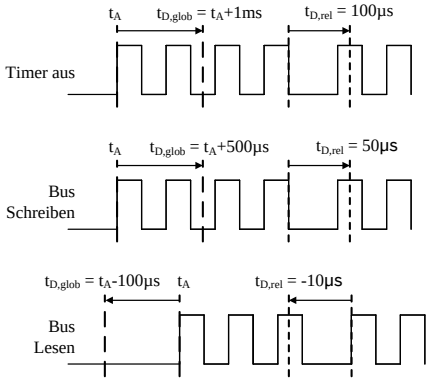
4.4.2. Anwendung 2: Echtzeit Timermodul

Nun soll anhand einer komplexen Echtzeit Anwendung die Auswirkungen des Checkpoint und Rollback auf das Systemverhalten aufgezeigt werden. Hierfür wird ein komplexes Echtzeit Timermodul auf Netzlisten-

und Verhaltensebene untersucht. Ein Überblick über das Modul ist in Abbildung 4.16a gegeben [Rob13]. Das Modul enthält mehrkanalige Zähler für den Ausgang `Timer aus`, einen mehrkanaligen Zähler `Timer in` für die Charakterisierung von Eingangssignalen und eine interne Zeitbasis `Interner Timer`. Ferner sind zwei Sequencer zur Verarbeitung von Assembler Microcode und Zusatzmodule für die interne Überwachung und für die externe Schnittstelle implementiert. Die eben genannten Einzelkomponenten kommunizieren intern über die `Interconnect Matrix`, die einen Großteil der verwendeten Si-Fläche ausmacht. Der gesamte Zustandsspeicher in der Netzliste besteht nach der Synthese für einen Xilinx Kintex7 XC7K325T aus 30.534 FF, 10 Block RAM, 87 Latches und 8 Schieberegistern.



(a) Blockschaltbild des Timermoduls mit SystemC Interaktionsüberwachung



(b) Deadlines der Interaktionen

Abbildung 4.16.: Blockschaltbild mit Deadlines des Timermoduls

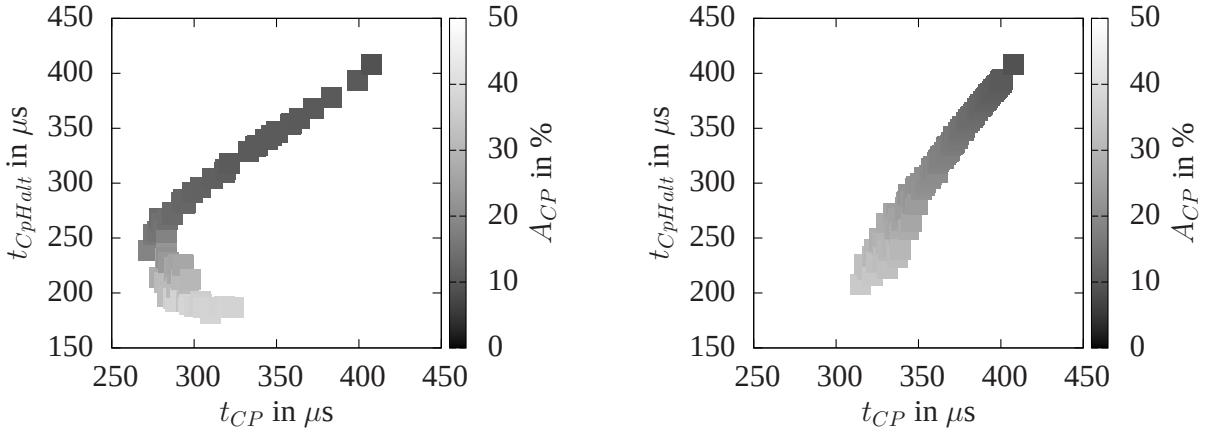
Die Schaltung erzeugt am Ausgang `Timer aus` eine periodische I^2C Kommunikation und verwendet für diese Aufgabe hauptsächlich die Blöcke `Timer aus`, `Interconnect Matrix`, einen Sequencer und das externe Bus Interface. Die Kommunikation wird in einer 20 ms Task mit einer globalen Deadline von 21 ms ausgeführt. Innerhalb des I^2C Datenpaketes wird zwischen den Datenbits eine relative Deadline von t_A + 100 μs angenommen. Neben dem Timerausgang hat auch die Buskommunikation eine absolute Deadline pro Datenpaket und eine relative Deadline innerhalb des Datenpaketes. Die Deadlines sind in Abbildung 4.16b zusammengefasst.

4.4.2.1. Netzlistenanalyse

Zur Erstellung eines vollständig persistenten Checkpoints wird ein wertestabiler und globaler Checkpoint erzeugt. Im Gegensatz zum bereits gezeigten Beispiel des MCS wird bei dem Timermodul keine händischer Vergleich der Checkpoint Lösungen erzeugt. Durch die große Anzahl der Zustandsspeicher werden hier zwei Ergebnisse des EA Algorithmus “ECJ” gemäß Abschnitt 4.2.4 mit verschieden EA Parametern verglichen. Die Pareto-optimalen Ergebnisse der 4000 Generationen mit je 500 Individuen sind in den Abbildungen 4.17a und 4.17b dargestellt. Hier wird die Zeit für die Checkpoint Erstellung t_{CP} und die Haltezeit der Schaltung t_{CpHalt} ebenso wie der Hardware Mehraufwand als Anteil von der gesamten Schaltung A_{CP} dargestellt. Die EA Läufe unterscheiden sich durch unterschiedliche Parameter in den Kreuzungseigenschaften der Individuen während des EA Algorithmus. Die Netzlistenanalyse mittels EA benötigt single-threaded etwa 48 Stunden auf einem Rechner mit zwei Intel Xeon E5620 Prozessoren (je vier Prozessorkerne mit 2,4 GHz) sowie 96 GiB RAM unter Ubuntu 12.04.4 LTS (64-Bit).

Wie erwartet steigt in beiden dargestellten Pareto-Fronten der Hardware Mehraufwand bei sinkender Check-

point Dauer durch eine erhöhte Anzahl an SHC und parallelen SC an. Dabei zeigt der Datenpunkt oben rechts mit $t_{CP} = 408\ \mu$s, $t_{CpHalt} = 408\ \mu$s und $A_{CP} = 0{,}092$ die Referenz einer einfachen SC mit direkter Speicheradressierung für die RAM Blöcke an. Alternativ zur SC bietet die auf PDR basierende Checkpoint Methode eine sehr günstige Alternative mit nur 0,2 % Hardware Mehraufwand an. Diese Methode liegt jedoch mit einer Checkpoint Erstellzeit von 21,2 ms und einer Haltezeit von 1,09 ms weit außerhalb der echtzeitfähigen Lösungen aus Abbildung 4.17.



(a) NSGA-II Lauf 1 **(b)** NSGA-II Lauf 2

Abbildung 4.17.: Pareto-Optimale Checkpoint Kenngrößen für die Netzliste des Timermoduls

Im Zuge der einheitlichen Modellierung der Checkpoint Methoden lassen sich nun die Ergebnisse des CaRLOS Framework mit den bekannten Methoden aus der Literatur vergleichen. Hier werden die Hardware Methoden der einfachen SC, SHC oder direkter Adressierung von Koch et al. (2007) [KHT07] und die PDR Methodik nach Sahraoui et al. (2012) [SGBG12] und Yang et al. (2013) [YHH+13] mit einem exemplarischen Ergebnis der Pareto Front verglichen ($t_{CP} = 272\ \mu$s, $t_{CpHalt} = 239\ \mu$s und $A_{CP} = 0{,}186$). Der quantitative Vergleich ist in Abbildung 4.18 mit einer Normierung auf das CaRLOS Ergebnis abgebildet, wobei jeweils ein kleiner Faktor besser ist. Zwei von drei Parametern sind gegenüber dem Stand der Technik verbessert. Die Beschleunigung der Checkpoint Zeiten wird jedoch durch eine erhöhte Flächennutzung in Kauf genommen. Die PDR Methodik benötigt lediglich einen kleinen ICAP Controller und ist demnach sehr kostengünstig mit einer deutlich höheren Checkpoint Dauer gegenüber den Netzlisten-invasiven Verfahren. Hierbei wird von einer Speicherung der Konfigurationsdaten in einem externen Speicher ausgegangen und nicht von einer Zwischenablage im CRAM. Je nach Auswahl eines Ergebnisses der Pareto-Front des CaRLOS Frameworks verändern sich die Faktoren zum Stand der Technik.

4.4.2.2. Verhaltensanalyse

Nach der Optimierung der Checkpoint Methode in der Netzlistenanalyse erfolgt nun die Anwendung der Methode in der Schaltung mit der Analyse der Interaktionen gemäß Abschnitt 4.3. Hierbei sollen die Zeitpunkte der Checkpoint Erstellung evaluiert und die Auswirkungen auf die Deadlines der Schaltung reduziert werden. Für die Verhaltensanalyse werden zwei Checkpoint Methoden aus dem Lösungsraum favorisiert, siehe Tabelle 4.5. Methode 1 zeigt eine kostengünstige Variante und Methode 2 zeigt die geringste Haltezeit t_{CpHalt} aller Lösungen. Die Lösungen enthalten jeweils SC und SHC für FF sowie PDR und MM für BRAMs.

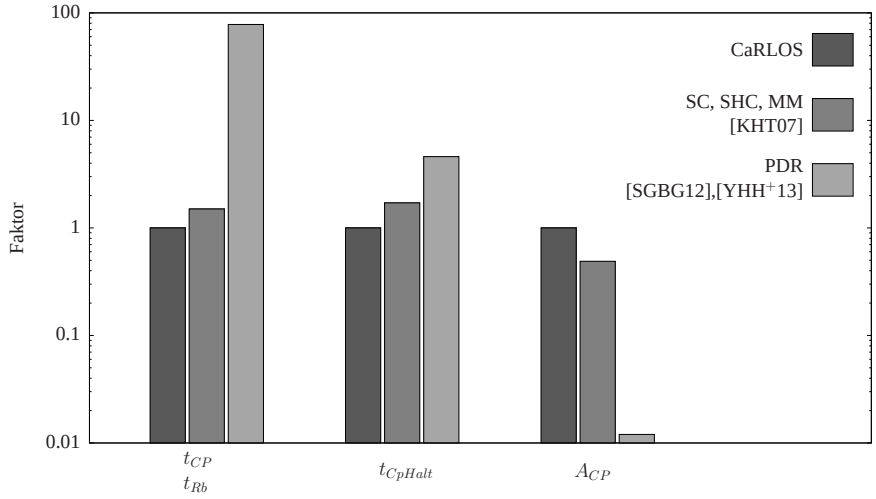


Abbildung 4.18.: Vergleich einzelner Checkpointing Methoden anhand von Parametern

Tabelle 4.5.: Methodenauswahl mit Parametern

CP Methode	t_{CP}, t_{CpHalt} in μs	t_{CpHalt} in μs	A_{CP} in %	CP Periode in ms
1	408	408	9,2	2,66
2	310	181	37,8	1,75

Zu Beginn der Verhaltensanalyse wird eine Initialsimulation der Anwendung mit SystemC RTL durchgeführt. Hier werden für 500 ms simulierte Zeit innerhalb einer Simulationsdauer von 152 Minuten die Interaktionen mit dem Gesamtsystem aufgezeichnet. Dieser Monitor der Interaktionen wird durch die dedizierten SystemC Ports `sc_out_log` und `sc_in_log` durchgeführt. Die Aufrufe dieses Loggings innerhalb des Schaltungsmodells sind in Abbildung 4.16a enthalten. Der Port `sc_out_log` speichert die Schreibzugriffe auf den Ausgangsports des Bus Interfaces und des Timerausganges. Die Lesezugriffe auf den Bus werden explizit im Bus Interface mittels `sc_in_log` registriert. Für die Lesezugriffe existiert kein Überladener Zuweisungsoperator und der Monitor muss demnach dediziert im Modell beschrieben und aufgerufen werden.

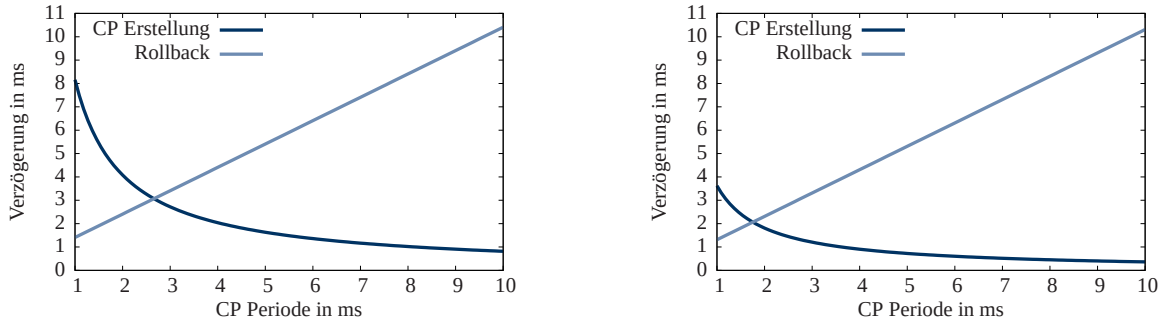
Anhand der Initialsimulation des Gesamtsystems mit der Schaltung kann nun eine theoretische Verzögerung der Ausgangssignale bei einer variablen Checkpoint Periode und bei einer Checkpoint Strategie ermittelt werden. Die Checkpoint Strategie besteht zum einen aus einer konstanten Periode und zum anderen aus einer interaktionsabhängigen Periode nach Abschnitt 4.3.2.1. Im Laufe der Evaluation zeigt sich, dass die konstante Periode einen um 0,02 bis 0,04 geringere Datengültigkeit P_D gegenüber der interaktionsabhängigen Periode hat. Zudem erzeugen die CP Erstellungen während der I^2C Sendung mehrere Überschreitungen der relativen Deadlines. Somit kommt für diese Anwendung nur die interaktionsabhängige periodische CP Strategie in Betracht.

Für diese Strategie wird die Periodendauer innerhalb der Taskbearbeitung anhand der theoretischen Verteilungen der CP über die Initialsimulation vorbestimmt. Hierbei wird die Verzögerung der Ausgangssignale bei der CP Erstellung durch die Haltezeit t_{CpHalt} ermittelt. Die akkumulierte Haltezeit steigt bei kleinerer Periodendauer für die CP Erstellung. Jedoch kann die Verzögerung der Ausgangssignale nicht durch eine große Periodendauer behoben werden. Hierfür ist die Verzögerung bei einem Rollback verantwortlich, die sich mit steigender Periodendauer vergrößert. Nach einem Rollback muss im schlechtesten Fall die gesamte Ausführung während einer Periodendauer erneut durchgeführt werden. Für beide angewendeten CP Methoden sind diese Verzögerungen in den Abbildungen 4.19a und 4.19b dargestellt. Der Schnittpunkt zeigt in etwa das

Optimum zwischen CP Erstellung und Rollback Verzögerung an.

Bei der Bestimmung der CP Periode mit dieser Methode ist nicht die reduzierte Performance der Schaltung zur Einhaltung der Deadlines einbezogen. Ferner muss eine angewendete CP Periode die Bedingung aus Satz 4.4 erfüllen. Ist die CP Periode jedoch kleiner als die FDZ, muss die Schaltung zeitlich um mehrere CP Perioden zurückspringen. In dem betrachteten Timermodul wird die FDZ mit 2 ms definiert.

Die CP Periodendauer ist nun als Parameter in Tabelle 4.5 gesetzt. Das CaRLOS Framework exportiert daraufhin eine Liste mit Zeitpunkten für den Checkpoint. In dem Checkpoint Controller des SystemC Modells wird die Liste gelesen und während eines Checkpoint Zeitpunktes werden die Takt und Enable Leitungen für die Zeit t_{CpHalt} deaktiviert.



(a) CP Methode 1 **(b)** CP Methode 2

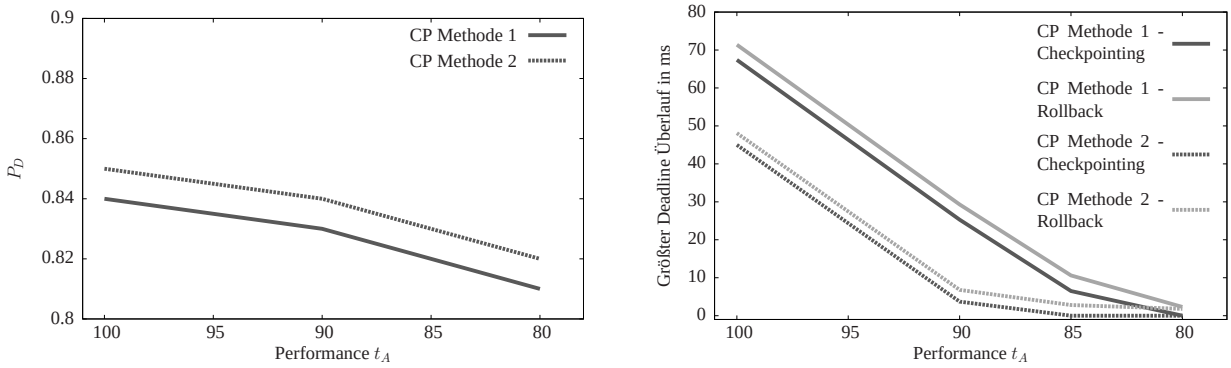
Abbildung 4.19.: Theoretische Verzögerung des Timings bei CP Erstellung und nach einem Rollback

Durch die Virtualisierung der Haltezeiten und Erstellzeitpunkte kann zusammen mit dem Schaltungsmodell eine Simulation und Auswertung des Checkpointing durchgeführt werden. Im Gegensatz zum Stand der Technik werden nun anhand der initialen Deadlines die Datengültigkeit P_D und die zeitliche Gültigkeit P_Z bestimmt. Die Gültigkeiten sind in Abbildung 4.20 zusammengefasst. Je höher die Datengültigkeit P_D ist, desto wahrscheinlicher ist die störungsfreie Verwendung eines Checkpoints während der Laufzeit. Der Deadline Überlauf wird sowohl für die Erstellphase als auch für den Fall eines Rollback angegeben. In der Erstellphase verzögert die CP Haltezeit die Ausführung und erzeugt somit Deadline Überläufe. Für den Rollback wird eine Fehlerrate von einem Fehler pro Mikrosekunde angenommen und nach jedem Fehler wird analytisch ein virtueller Rollback durchgeführt und die Verschiebungen der Interaktionen beobachtet.

Für die initiale Performance der Schaltung ohne CP von 100 % zeigt sich die höchste Datengültigkeit, da eine verlängerte Ausführungszeit t_A umgekehrt proportional zu der Datengültigkeit ist (Abbildung 4.20a). Eine Reduzierung der Funktionalität (Performance) der Schaltung bedeutet eine verkürzte Ausführungszeit t_A, wobei die Deadlines konstant bleiben gegenüber der Initialsimulation mit 100 % Performance. Somit verbleiben bis zum Erreichen der Deadline innerhalb der Ausführungszeit größere Zeitschlitze für das Checkpointing. Es sinkt jedoch das Verhältnis von Ausführungszeit zu Kommunikationszeit und somit sinkt auch P_D.

Einen deutlich größeren Einfluss hat die Reduzierung der Performance auf die zeitliche Gültigkeit in Abbildung 4.20b. Sowohl für das Erstellen der CP als auch nach einem Rollback müssen erhebliche Deadline Überschreitungen bei 100 % Ausführungszeit berücksichtigt werden. Daher muss die Anwendung an die Haltezeit t_{CpHalt} und an die Wiederholzeit t_w angepasst werden. Für die CP Methode 1 mit t_{CpHalt} gleich 408 ms wird bei einer Ausführungszeit von 80 % keine Deadline mehr durch eine Erstellung von CP überschritten. Wie erwartet verbleibt jedoch nach einem Rollback eine maximale Deadline Überschreitung von 2,3 ms

aufgrund der Wiederholdauer t_w von 3,1 ms. Die CP Methode 2 zeigt durch eine kürzere Haltezeit und Periodendauer eine optimale Überschreitung während des Checkpointing bei 85 % Ausführungszeit. Auch hier kann durch t_w von 1,9 ms die Überschreitung nach einem Rollback nicht vermieden werden. Zeitliche Ungültigkeiten können durch die engen Deadlines im Vergleich zur Rücksprungzeit erst bei deutlich größeren Einbußen in der Performance komplett vermieden werden.



(a) Datenungültigkeit **(b)** Zeitliche Ungültigkeit

Abbildung 4.20.: Datengültigkeiten und zeitliche Gültigkeiten unter Anpassung der Schaltungsperformance

4.4.2.3. Methodenvergleich

Nach der detaillierten Erstellung werden die CaRLOS Ergebnisse der Verhaltensanalyse nun mit Checkpoint Methoden nach dem Stand der Technik verglichen. Hierbei werden die hardware-basierende SC bzw. MM Methoden nach Koch et al. (2007) [KHT07] und die PDR Methodik nach Sahraoui et al. (2012) [SGBG12] und Yang et al. [YHH+13] mit zwei verschiedenen Checkpoint Strategien verknüpft. Bei Kwak et al. (2001) wird eine optimale konstante CP Periode bestimmt. Dagegen wird bei Wolter (2008) [Wol08] eine interaktionsabhängige CP Strategie verfolgt. Da das hier vorgestellte CaRLOS Framework alle diese Ansätze vereint, werden die Ergebnisse der Erfolgswahrscheinlichkeit für Checkpoint und Rollback P_{CP} und der Verfügbarkeit A in Tabelle 4.6 miteinander verglichen.

Tabelle 4.6.: Vergleich der Ergebnisse für eine Ausführungszeit von 85 % gegenüber der Anwendung ohne Checkpoint

Methodik	P_D	P_Z	P_{CP}	A[1]
Reset	-	-	-	$8,33 \cdot 10^{-5}$
CP1 [KHT07], [KCK01]	0,82	0,92	0,76	$9,22 \cdot 10^{-5}$
CP2 [KHT07], [Wol08]	0,82	0,92	0,76	$9,22 \cdot 10^{-5}$
CP3 [YHH+13], [KCK01]	0,58	0	0	$8,74 \cdot 10^{-5}$
CaRLOS	0,83	0,92	0,77	$9,27 \cdot 10^{-5}$

[1] λ_F = 500 FIT, Stückzahl = 1 Mio. => MTBF = 0,5 h; $t_{Detektion}$ = 1ms

Insgesamt zeigt das CaRLOS Framework eine 77 %ige Wahrscheinlichkeit für den Erfolg der Checkpoint und Rollback Maßnahme mit einer leicht verbesserten Verfügbarkeit gegenüber den Einzelmethoden und einer erheblichen Verbesserung gegenüber dem Reset. Trotz der verbesserten zeitlichen Gültigkeit während des

Checkpointings bleibt die Gesamtwahrscheinlichkeit für den Erfolg der CP Methodik annähernd konstant. Begründet wird dies mit der konstanten Anzahl der Überschrittenen Deadlines, trotz verbessertem zeitlichen Überlauf. In Folge der Evaluierung zeigt sich zudem die schlechte Verwendbarkeit der PDR für das Echtzeit Timermodul aufgrund der konstanten Überschreitungen von Deadlines im Falle eines Rollbacks.

Die Ergebnisse für diese Anwendung zeigen, dass eine Fehlerkorrektur mittels Checkpoint und Rollback in dem Timermodul mit den gegebenen Randbedingungen kaum sinnvoll einzusetzen ist. Durch die geringe Fehlerrate von 500 FIT steht die Erhöhung der Verfügbarkeit in keinem guten Verhältnis zu der Erfolgswahrscheinlichkeit des CP und Rb. Mit einer Verdoppelung der Deadlines für Interaktionen steigt P_{CP} auf über 90 % an. Mit dieser Maßnahme lassen sich folglich bei gleichzeitiger Reduzierung der Ausführungszeit auf 85 % Checkpoints zur Fehlerkorrektur einsetzen. Wie gezeigt müssen in diesem Fall die Parameter und die Checkpoint Methode optimiert werden, um einen zu starken Performanceverlust zu vermeiden. Gleichzeitig muss die Fehlerdetektionszeit kleiner sein als die CP Periode.

4.5. Zusammenfassung und Erweiterungen

Im Rahmen des in diesem Kapitel vorgestellten CaRLOS Frameworks können nun echtzeitfähige Schaltungen auf die redundanzfreie Fehlerkorrektur mittels Checkpoint und Rollback untersucht werden. Entwickler können nun mittels einer hybriden Netzlistenanalyse und Netzlistensimulation das hardwarenahe Verhalten taktgenau bestimmen. An dieser Stelle kann die CP Methodik durch Anwendung von SC, SHC, MM und PDR Maßnahmen auf bis zu vier Kenngrößen in der Netzlistenebene optimiert werden. Ferner wird bei einer MCU die Hochlaufzeit um Faktor drei gegenüber dem klassischen Boot und Initialisierungsverfahren verkürzt. Für ein komplexes Timermodul kann hier eine Verbesserung einzelner CP Eigenschaften um den Faktor zwei bis drei erreicht werden. Die Parameter der Checkpoint Dauer, Checkpoint Haltezeit, Rollback Dauer und Hardware Mehraufwand werden anschließend an die Verhaltensanalyse und Simulation übertragen.

Die zweite Säule der effektreduzierten Zustandswiederherstellung mittels Checkpoint (CP) und Rollback (Rb) besteht aus einer Aussage der Erfolgswahrscheinlichkeit P_{CP} für die Datengültigkeit und Einhaltung von Deadlines. Mittels einer mit CP Bibliotheken und Interaktionslogging erweiterten SystemC RTL Verhaltenssimulation werden diese Gültigkeiten für Daten und Zeit bei Anwendungssimulationen bestimmt. Anhand von Deadlines der Interaktionen mit dem Gesamtsystem wird nach den Analysen und Simulationen die Bewertungsmetrik P_{CP} erstellt. Auf dieser Datenbasis kann nun parallel zur Veränderung der Ausführungszeit t_A die optimale CP Strategie bestimmt werden. Das vorgestellte Echtzeit Timermodul erreicht zusammen mit einer CP Erfolgswahrscheinlichkeit von 77 % eine Verbesserung der Verfügbarkeit um etwa $1 \cdot 10^{-5}$ gegenüber dem Reset Verhalten.

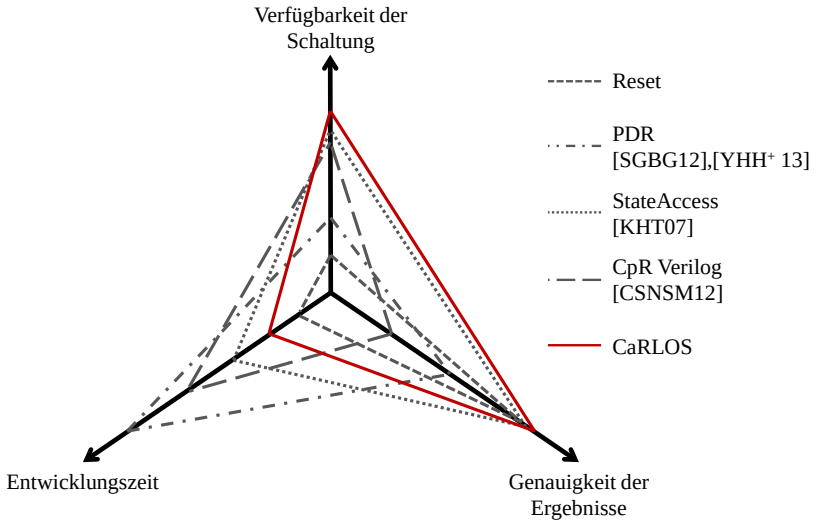


Abbildung 4.21.: Qualitativer Vergleich des CaRLOS Frameworks mit konventionellen CP Entwicklungstools

Ein wichtiger Mehrwert des CaRLOS Framework zeigt sich zudem in der Evaluationsgeschwindigkeit der CP Methodiken. Teil-automatisierte Auswertungen und Modellbildungen der Netzliste bieten für erfahrene Entwickler eine Ergebnisdarstellung des Lösungsraumes nach wenigen Stunden. Mit dieser Maßgabe wird die Eintrittshürde für den Einsatz von CP und Rb in industriellen (FPGA) Schaltungen deutlich herabgesetzt. Dementsprechend zeigt Abbildung 4.21 eine qualitative Beurteilung der Tools zur CP Erstellung und Evaluation. Es zeigt sich, dass trotz konstanter Genauigkeit und Qualität der Ergebnisse die Entwicklungszeit deutlich verkürzt werden konnte.

Ferner stellt diese Arbeit 6 Merkmale für eine CP Anwendung auf Echtzeitsysteme heraus.

1. *Verwendung von Deadlines*
   Interaktionen zum Gesamtsystem müssen mit Deadlines behaftet sein. Anderenfalls führt jeder Rollback zwangsläufig zu einer Asynchronizität der Schaltung.

2. *Time-Triggered Entwicklung*
   Die gebündelten Interaktionen am Ende einer Ausführung im Time-Triggered Entwicklungsansatz führen zu weniger Deadline Überläufen und erhöhen die Datengültigkeit.

3. *Performanceanpassung der Anwendung*
   Zum Zeitpunkt der CP Erstellung wird die Anwendung verzögert. Diese Zeit muss innerhalb der Schaltung oder des Gesamtsystems kompensiert werden.

4. *Verringerung der FDZ*
   Die Fehlerdetektionszeit (FDZ) bestimmt maßgeblich die Rücksprungzeit während eines Rollbacks. Eine kurze FDZ kann eine kurze CP Periode hervorbringen.

5. *CP Methode optimieren*
   Es verkürzt sich die Rücksprungzeit, CP Erstellzeit und/oder der HW Mehraufwand. Dies hat unmittelbare positive Auswirkungen auf die zeitliche Gültigkeit und die Ausführungszeit der Anwendung.

6. *Verwendung interaktionsabhängiger CP*
   Bei Interaktionen der Schaltung mit gekoppelten CP Erstellungszeitpunkten vergrößert sich die Datengültigkeit der Checkpoints.

Werden diese Merkmale in einer sequentiellen Echtzeitschaltung erfüllt, ist eine CP Erfolgswahrscheinlichkeit von über 90 % möglich. Das CaRLOS Framework unterstützt den Entwickler bei der Umsetzung der Punkte 5 und 6. Die Punkte 1 bis 3 müssen während der Systementwicklung berücksichtigt werden und sind nicht Teil dieser Arbeit. Einen Ansatz zur redundanzfreien Verringerung der FDZ aus Punkt 4 wird im folgenden Kapitel vorgestellt.

Für einen praktischen Einsatz von CP Methoden muss die gesamte Methodik unter dem Aspekt der funktionalen Sicherheit entwickelt werden. Hierfür müssen zukünftig Selbsttest Methoden wie Ring-SC, Rücklesen des CRAMs oder Testpattern in den Zustandsspeichern implementiert werden. Ferner muss die Rücktransformation der Netzlisten Modelle in das EDIF Format mit validierten Tools oder mit einer ausführlichen nachträglichen Validierung erfolgen, siehe auch Abschnitt 3.3.

5. Priorisierte und datenflussorientierte Fehlerdetektion im Konfigurationsspeicher

Für FPGA Systeme in automobilen Anwendungen trägt eine kurze Fehlerdetektionszeit (FDZ) im Konfigurationsspeicher zu einer hohen Verfügbarkeit und zur Einhaltung der funktionalen Sicherheit bei. Diese Zusammenhänge sind in den Abschnitten 3.1 und 3.2 beschrieben und betreffen SEU und Haftfehler.

Ferner zeigen die Ergebnisse der Checkpoint und Rollback Untersuchungen aus Abschnitt 4.3 eine direkte Beeinflussung der Rücksprungzeit durch die FDZ. Je kürzer die FDZ, desto kleiner ist die Wiederholzeit nach einem Rollback. Für einen FPGA bedeutet eine kleine FDZ eine notwendige Verkleinerung der Testzeit für das CRAM zur Erstellung netzlistenkonformer Checkpoints.

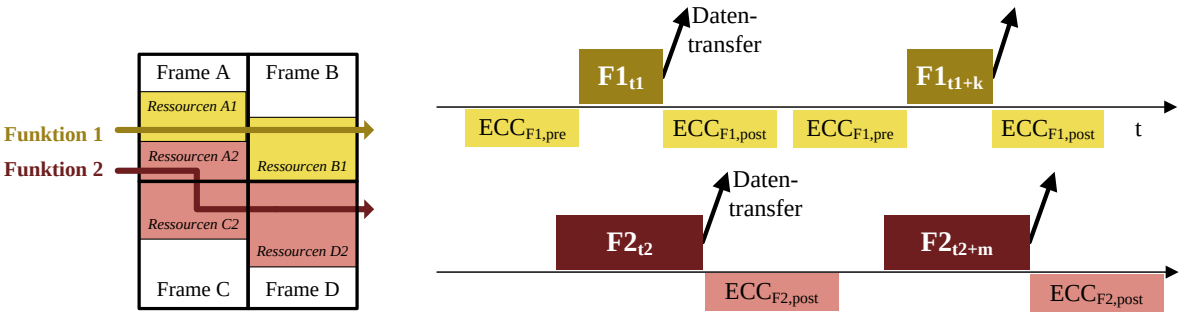
In diesem Kapitel wird nun das Verfahren eines Priorisierten Konfigurationstest (PKT) erläutert, der bereits in [FRS14] dargestellt wurde. Die Voraussetzung hierfür ist der Zugriff auf Teilbereiche des Konfigurationsspeichers und die Nutzung von Paritätsbits für diese Teilbereiche. Das Ergebnis des PKT ist eine Testsequenz, die zur Laufzeit zyklisch abgearbeitet wird, die Teilbereiche priorisiert ausliest und überprüft. Während der Entwicklung stand eine kostengünstige Umsetzung im eingebetteten System im Fokus. Für diese Methodik werden neben den Grundprinzipien auch die Schaltungsanalyse und die Partitionierung der Testbereiche erklärt. Ferner wird die Erzeugung der Testsequenz über Scheduling und Verteilungsfunktionen sowie die Einbettung in das Echtzeitsystem beschrieben. Abschließend werden Ergebnisse und Analysen vorgestellt.

5.1. Grundlegende Auslegung

Ein priorisierter Konfigurationstest benötigt zwei grundlegende Elemente. Zum einen werden Ressourcen zum Priorisieren benötigt. Zum anderen benötigt der PKT eine Datengrundlage und Freiheitsgrade für den Vorgang der Priorisierung. Die hier verwendeten Ressourcen sind Major Frames *MFs* und enthalten die essentiellen Konfigurationsbits der verwendeten Instanzen, Primitiven und Settings. Dabei belegt jede eingebettete Funktion im FPGA eine Gruppe von MFs, die getestet werden muss. Als Datengrundlage für die Priorisierungen der MFs werden Vorgaben zur FDZ herangezogen, die aus der ISO 26262 oder der Verfügbarkeitsanforderung abgeleitet werden. Für diese Arbeit wird angenommen, dass in zukünftigen FPGA-Anwendungen mehrere größtenteils unabhängige Funktionen eingebettet sind.

5.1.1. Priorisierter Konfigurationstest

Das grundlegende Prinzip des PKT ist an den ECC normaler RAM Bausteine angelehnt. Hier wird eine Prüfung des Datenwortes erst vorgenommen, wenn ein Lesezugriff auf dieses Wort erfolgt. Dieser Ansatz ist für den statischen CRAM des FPGAs erweitert und auf MFs angewendet. Die Propagation der Ausgangsdaten einer Funktion entscheidet hier, wann eine ECC Prüfung der korrespondierenden MFs durchgeführt wird. In dieser Arbeit werden Xilinx FPGAs untersucht und die kostengünstige interne SECDED ECC Prüfung mittels *FrameECC* wird angewendet [Xil13b]. Dieses Verfahren wurde bereits in Abschnitt 2.3 erläutert. Als Grundlage für weitere Erläuterungen des zeitlichen Verlaufes des PKT wird die exemplarische Schaltung aus Abbildung 5.1a verwendet. Diese besteht aus zwei Funktionen mit verteilter Nutzung der Frames, wobei MF A von beiden Funktionen verwendet wird. Die möglichen Ausführungszeiten der ECC Prüfung sind in Abbildung 5.1b dargestellt. Die Fehlerkorrektur vor der Ausführung einer Funktion auf Ressourcen wird mit $\mathrm{ECC}_{F,pre}$ bezeichnet und die Fehlerdetektion nach der Ausführung oder Propagation des Ausgangs wird mit $\mathrm{ECC}_{F,post}$ eingeführt. Eine Anwendung des PKT als $\mathrm{ECC}_{F,pre}$ kann als Fehlerprävention oder für die Ansteuerung von partiellem Überschreiben angewendet werden (*Scrubbing*). Innerhalb dieser Arbeit wird jedoch ein $\mathrm{ECC}_{F,post}$ bevorzugt, da somit Fehler der Netzliste während der Abarbeitung der Funktion F nachträglich detektiert werden können.



(a) Vereinfachtes FPGA Layout **(b)** Ausführungszeit des PKT

Abbildung 5.1.: Grundprinzip eines PKT

Mit Hilfe des Datenfluss-Timings erhalten die Ressourcen einer Funktion eine Ausführungszeit. Der Datenfluss gibt dabei vor, zu welchen Zeitpunkten bestimmte Hardware-Funktionen bearbeitet und die Ergebnisse propagiert werden. In Abbildung 5.1 ist zu erkennen, dass es durch die unterschiedlichen Ausführungszeiten zu einer Überschneidung der Anforderungen von $\mathrm{ECC}_{F,post}$ kommen kann. Hierfür wird ein *Scheduler* eingesetzt,

um diese Konflikte zu lösen. Die Verknüpfung des Datenflusses mit der Schaltung und den verwendeten Ressourcen wird in einem PKT-Tool hergestellt, das auch den Scheduler enthält. Die resultierende Testsequenz, bestehend aus MFs, kann anschließend in den RAM des FPGAs abgelegt und zur Laufzeit abgearbeitet werden. Diese Anwendungskette ist in Abbildung 5.2 dargestellt. Für die Optimierung der Testsequenz wird ein Datenfluss mit *Hyperperiode* benötigt. Eine Hyperperiode kann eine Periodendauer von mehreren internen Funktionsdurchläufen haben.

Wie dargestellt, wird die Testsequenz nicht zur Laufzeit erstellt und als *Offline*-Scheduling bezeichnet. Dies bringt den Vorteil, dass kein Scheduler in das FPGA System implementiert werden muss. Zudem ermöglicht ein Offline-Scheduling pareto-optimale Verteilungen der MF in der Testsequenz. Anders als beim Offline-Scheduling wird bei der *Online*-Testverteilung die Reihenfolge der Frames in der Sequenz zur Laufzeit bestimmt. Die echtzeitfähige Verteilung wird ebenfalls durch Deadlines und Prioritäten bestimmt. Mit diesem Prinzip lassen sich beispielsweise interrupt-lastige Funktionen oder Einzelfunktionen im FPGA sehr gut testen. Hier kann jedoch nur eine schlechte Vorhersage über die Einhaltung der gesetzten FDZ getroffen werden. Zudem kann es zu Bereichen oder einzelnen MF kommen, die aufgrund niedriger Priorität nicht getestet werden. Der implementierte Scheduler muss ferner eine Komplexität für die Einhaltung der Deadlines und Priorisierung einzelner MF aufweisen. Dies erhöht zugleich die Fehlerrate des Scheduler. Insgesamt widerspricht eine Online-Testverteilung der Anforderung einer kostengünstigen Lösung, kann aber bei nicht vorhersehbaren Datenflüssen in interrupt-gesteuerten Systemen eine Alternative sein. Nach Abwägung der genannten Faktoren wurden in dieser Arbeit mehrere Offline-Verteilungsverfahren gewählt und weiterentwickelt.

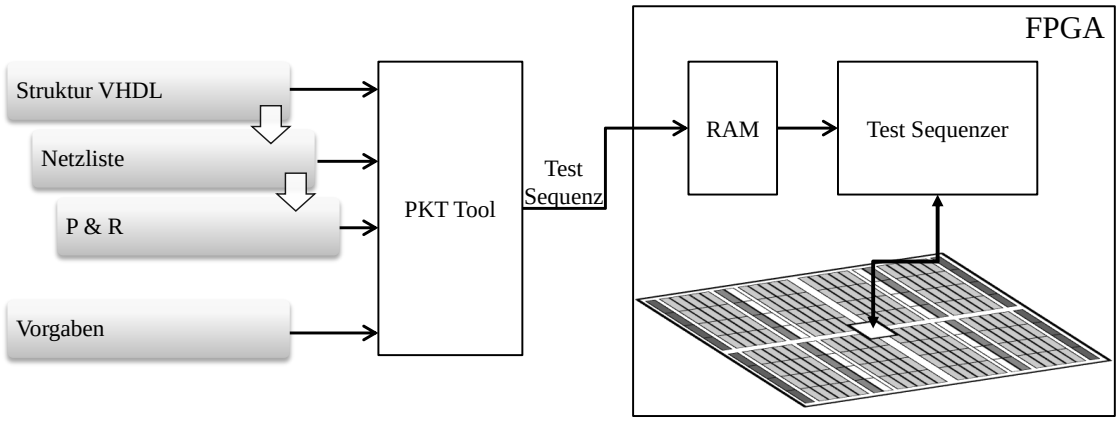
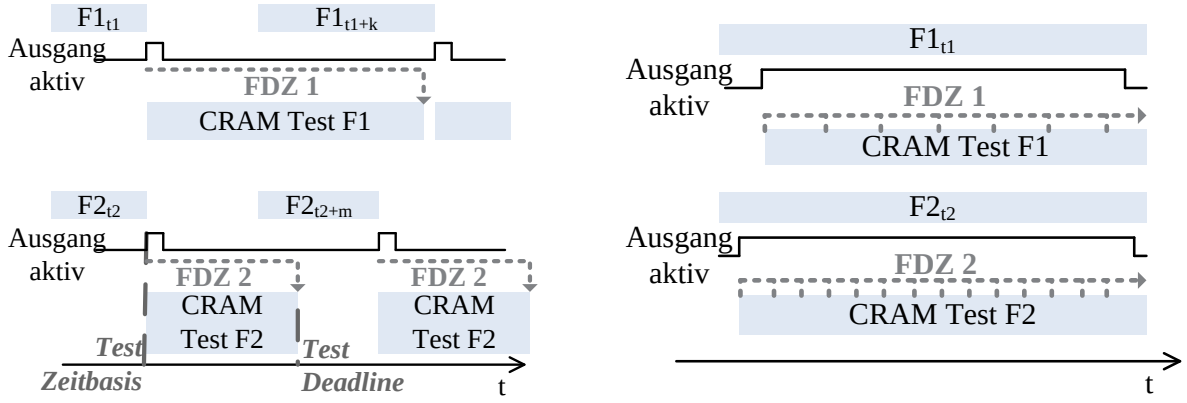


Abbildung 5.2.: Implementierungsvorgang für den Offline PKT

5.1.2. Datenfluss-Varianten

Für eine Übertragung der Zeitinformationen auf den PKT müssen die Aufrufe-, Bearbeitungs- und Datenausgabezeiten für bestimmte Funktionen innerhalb des FPGA bekannt sein. Hierfür betrachtet diese Arbeit zum einen die *zeitgesteuerten* Datenflüsse (engl. Time-Triggered). Die Anwendung des zeitgesteuerten Datenflusses für einen PKT ist in Abbildung 5.3a dargestellt. Die Funktionen erhalten Anforderungen an die Testzeit FDZ mit einer Zeitbasis B_i und Deadline D_i. Zudem ist der Ausgang periodisch aktiv und es erfolgt keine zusätzliche Testanforderung solange die FDZ nicht erreicht ist.



(a) Zeitgesteuert (Time-Triggered) **(b)** Kontinuierlich

Abbildung 5.3.: Gruppierung der Datenflüsse für PKT

Entspricht der Datenfluss nicht den Zeitanforderungen aus Abbildung 5.3a oder sind die Zeiten für die Basis und Deadline nicht bekannt, dann wird ein *kontinuierlicher* Datenfluss angenommen. Diese Art des Datenflusses ist der zweite in dieser Arbeit betrachtete Typ und schematisch in Abbildung 5.3b dargestellt. Dabei wird versucht, über eine homogene Testverteilung und Priorisierung die gesetzte FDZ zu jedem möglichen Fehlerzeitpunkt einzuhalten. Hierbei handelt es sich zum Beispiel um Schaltungen zur Erzeugung von Pulsweitenmodulation (PWM) Signales oder Streaming Dienste.

5.1.3. Partitionierung von Testbereichen

Die grundlegende Testeinheit für den PKT ist ein MF. Ein MF kann innerhalb eines Testslots S mittels ECC geprüft werden. Ein MF enthält entweder die Konfiguration der CLBs, I/O Blöcke, DSP und BRAM Einstellungen oder globale Takteinstellungen. Die in den Platzierungsinformationen enthaltenen Koordinaten lassen sich auf Zeilen und Spalten des FPGAs projizieren [NR08]. Die Zeilen und Spalten wiederum können einer Adresse für ein MF zugeordnet werden. Eine feiner granulierte Lokalisierung ist aufgrund der ungenauen Zuordnung zu Minor Frames nicht möglich. Ferner wäre eine aus Minor Frames bestehende Testsequenz etwa 40 mal länger und würde demnach auch mehr RAM in der Anwendung beanspruchen.

Innerhalb eines Testslots müssen die Minor Frames eines MF mittels ICAP ausgelesen werden. Die überwachende Primitive `FrameECC` liefert daraufhin das Ergebnis der Paritätsüberprüfung sowie die Lokalisierung des Fehlers im Minor Frame [Xil10]. Das Auslesen und die Überprüfung eines MF der Logik und somit eines Testslots werden nach den Erläuterungen im Anhang A.4.2 innerhalb von $t_S = 40\ \mu s$ durchgeführt. Weiterführende Major Frames anderer Block Typen mit globalen FPGA Einstellungen können mit zwei zusätzlichen Mikrosekunden gelesen werden.

5.2. Datenflussorientierte Schaltungsanalyse und Erstellung der Testbereiche

Im Folgenden wird die Schaltungsanalyse zur Korrelation der implementierten Funktion mit den verwendeten FPGA-Ressourcen dargelegt. Als Basis hierfür wird die bereits aus dem CaRLOS Framework bekannte

Netzliste nach dem Standard *EDIF* 2 0 0[1] verwendet. In dieser Arbeit wird auf die elektrischen Netze und Signale fokussiert, da beim FPGA Layout kein MF ausschließlich mit Instanzen und ohne elektrische Netze konfiguriert ist. Abbildung 5.4 zeigt die Kette der Verknüpfungen auf. Da Schaltungsentwickler in der Regel nicht mit Netzlistensignalen arbeiten, wird eine Korrelation zur HDL Sprache angeboten. Dabei werden VHDL Signale mit der Netzliste über die hierarchischen Netznamen korreliert. Die Netzliste wiederum wird auf das platzierte und verdrahtete Design abgebildet. Somit erhalten elektrische Signale und Instanzen einer Funktion eine Platzierungsinformation.

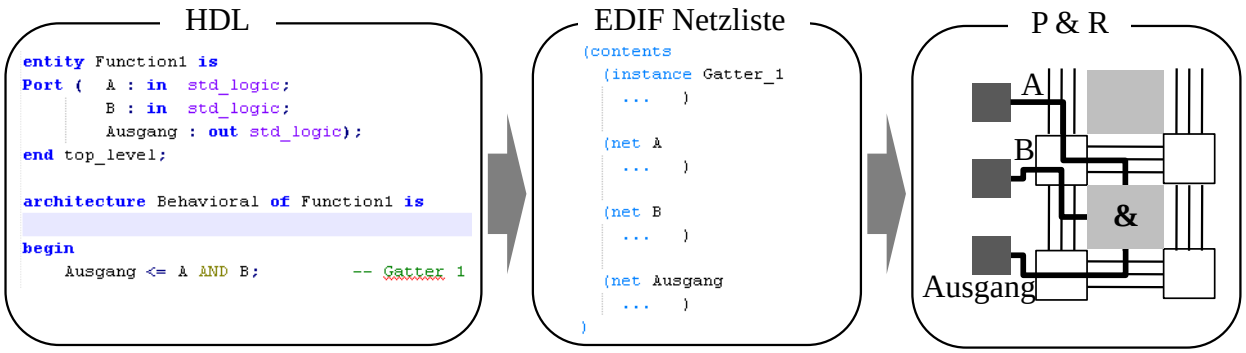


Abbildung 5.4.: Verknüpfung der HDL Signale über die Netzliste mit den FPGA Ressourcen

Wie in Abschnitt 3.1 erläutert, müssen für dedizierte Funktionen die FDZ verringert werden. Somit müssen für diese einzelnen Funktionen die Platzierungsinformationen über den Abhängigkeitspfad in der Netzliste gefiltert werden. Unter der Annahme einer korrekten Netzliste beeinflussen nur die Elemente den Ausgang, die fehlerhafte Zustände weiter propagieren können. Wie auch Asadi et al. (2007) [ATT07] belegen, müssen bei unbekanntem Zeitpunkt eines Fehlers alle Elemente einer Netzliste berücksichtigt werden.

Satz 5.1 *Für einen PKT einer Funktion müssen die Konfigurationselemente aller Instanzen und Signale auf dem Abhängigkeitspfad der Netzliste dieser Funktion getestet werden.*

Nach Satz 5.1 können keine logischen Maskierungen angewendet werden, da diese vom Eingangspattern abhängen und zur Zeit der Fehlerentstehung nicht vorhersehbar sind. Ferner kann keine zeitliche Maskierung angewendet werden, da die Wahrscheinlichkeit eines persistenten Fehlers immer über 0 liegt. Bezogen auf die Platzierungsinformationen bedeutet das, dass alle Konfigurationselemente für eine Funktion erfasst werden müssen, die bei einem Fehler den Ausgang der Funktion beeinflussen können. Aufgrund der Island- Struktur des FPGA sind die Abhängigkeitspfade der Netzliste hier ausreichend. Auf diesen Punkt wird in Abschnitt 5.2.2 vertieft eingegangen.

5.2.1. Korrelation der Funktionen zum Layout und zu Testregionen

Die Filterung der Platzierungsinformation einer Funktion basiert auf der schrittweisen Korrelation der HDL Signale über die Netze der Netzliste zu den P&R Signalen (Abbildung 5.4). Innerhalb der Schaltungsanalyse werden in erster Linie Netze gefiltert und zugeordnet. Instanzen der Netzliste werden nur optional bei einem fehlenden MF hinzugezogen.

[1] Standard: ANSI/EIA-548-1988

5.2.1.1. Korrelation von VHDL Signalen zur Netzliste

Als Grundlage der Anwendung des PKT wird momentan eine VHDL Beschreibung verwendet. Mittels der proprietären C/C++ Schnittstelle von ModelSim®werden alle HDL Signale der Schaltung extrahiert und gelistet. Hier werden mittels des FLI iterativ über alle Module die hierarchischen Signalnamen in eine Datei geschrieben [Men12b]. Der VHDL Code aus Anhang A.1 beschriebt die Instanziierung des Signal-Monitors innerhalb einer VHDL Schaltung. Die Funktion `void InitMonitor` aus der dynamischen Bibliothek `FLI_Interface.dll` initiiert die Filterung der hierarchischen Signalliste.

Anhand der VHDL Signalliste werden die Anforderungen an die FDZ mit der Schaltung korreliert. Es werden die angeforderten hierarchischen Signalnamen gefiltert und als Ausgangsbasis für die weitere Netzlistenanalyse verwendet. Somit arbeiten die Anwender auf den beschriebenen Ports und Signalen im weiteren Verlauf der Anforderungsbeschreibung. Jeder Modulport und jedes deklarierte Signal entspricht einem Netz in der Netzliste insofern es während der Synthese nicht optimiert wurde. Im weiteren Verlauf dieser Arbeit wird daher eine hierarchische Synthese anstelle einer nicht-hierarchischen Synthesevorgabe angenommen. Bei der Anwendung dieser Vorgabe auf Xilinx FPGAs wird die Synthese-Eigenschaft `Keep Hierarchy = YES` gesetzt. Die erzeugte Netzliste wird aus dem proprietären Format Xilinx Native Generic Database (NGC) in das EDIF Format konvertiert. Das anschließende Parsen in das PKT Tool übernimmt das Framework *BYU EDIF Tool* der Brigham Young Universität [Uni].

5.2.1.2. Korrelation der Netzliste zu Layoutinformationen

Die Repräsentation der Netze auf die Platzierungsinformation wird über die hierarchischen Signalnamen im XDL Format hergestellt. Das XDL Format ist eine menschlich lesbare Editierung der P&R Ergebnisse bei Xilinx, die aus dem binären NCD Format konvertiert werden kann. Die XDL Datei einer Schaltung enthält physikalisch platzierte Netze mit Koordinaten von programmierten PIPs. Ferner sind die Einstellungen und Koordinaten von Slice Instanzen aufgeführt. Veranschaulicht wird dieses Format in Anhang A.2.2. Mittels des Parseranteils des Frameworks *RapidSmith* werden die Platzierungsinformation in das PKT Tool eingelesen [LPL+10]. Danach wird eine Korrelation der platzierten Netze mit den Netzen der Netzliste nach [NR08] erstellt. Aufgrund mehrerer Optimierungsschritte im Entwicklungsablauf gelten folgende Aussagen.

1. Ein platziertes Signal kann mehreren Netzen und Ports der Netzliste entsprechen.

2. Platzierte Netze mit einer Netzquelle und alle Netzsenken innerhalb einer Slice-Instanz werden durch Slice-interne Einstellungen aufgelöst. Diese können nicht mehr korreliert werden.

3. Portnamen der Netzliste bei hierarchischen Designs werden entfernt und müssen über die elektrische Integrität korreliert werden.

4. Statische Signalpegel der Netzliste werden mittels statisch platzierter Netze abgebildet.

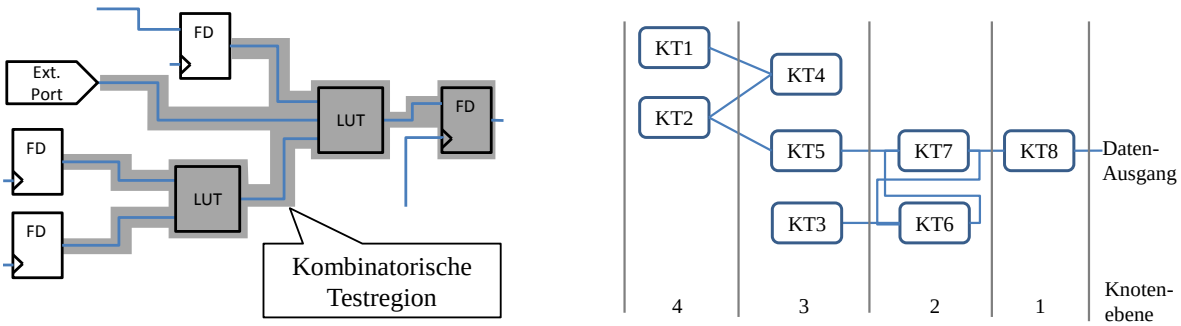
Die Zuweisung der platzierten Logiknetzen, Slice-Instanzen und statischen Netzen zu Elementen aus der Netzliste ermöglicht zusammen mit der Koordinatenbestimmung dieser platzierten Objekte eine Zuordnung von Funktionen zu Major Frames (MF). Die Zuweisung der platzierten Objekte zu einem MF erfolgt über einen X-Y Koordinatenabgleich der MF-Adresse zu der globalen Platzierungsadresse des Objektes. Variationen im

Koordinatensystem der Objekte werden über Korrekturtabellen angepasst. Empirische Versuche offenbaren, dass mit dieser Korrelationsmethode über 99,99 % der logischen Netze und Instanzen der Netzliste sich physikalisch einem MF zuordnen lassen. Für eine Verbesserung der Fehlerabdeckung werden ferner auch alle weiteren Elemente der Netzliste wie beispielsweise BRAM, DSP und I/O Blöcke korreliert. Für global verteilte Netze wie Takt- oder Enable-Signale werden die MFs für die gesetzten PIPs ermittelt.

5.2.1.3. Einhüllende Testregion

Nach der Korrelation der Netzliste mit den Platzierungsinformationen muss eine Filterung von Schaltungsteilen dedizierter Funktionen erfolgen. Die Anforderungen einer FDZ beziehen sich auf Schaltungsausgänge respektive die gesamten Abhängigkeitskette einer Funktion. Daher müssen alle MFs erfasst werden, die den betreffenden Funktionsausgang bei einem Einzelfehler mit einer Wahrscheinlichkeit größer 0 verändern können. Zu diesem Zweck wird die *Einhüllende Testregion* ermittelt. Die einhüllende Testregion ermittelt alle Ressourcen der platzierten Schaltung, die eine Funktion beeinflussen können. Dazu zählen neben PIPs und Slice Instanzen auch I/O Blöcke, BRAM Einstellungen, DSP Einstellungen, Takttreiber und weitere Blöcke. Durch die Anordnung der PIPs in Verbindungsknoten werden die gesamten Knoten der Funktion zugeordnet.

Zunächst werden alle *Kombinatorischen Testregionen* (KT) einer Funktion ermittelt und die zugehörigen Ressourcen bestimmt. Eine kombinatorische Testregion enthält alle Netzlistenelemente, die innerhalb einer Taktperiode das Ausgangssignal eines Zustandsspeichers beeinflussen können. Abbildung 5.5a verdeutlicht den Rahmen dieser Region, beginnend mit den Ausgängen von Zustandsspeichern bis zum Ausgang des FD Flip-Flops der nachfolgenden Taktstufe. Die Testregion enthält mit einem Anteil von über 90 % vor allem Verdrahtungsinformationen. Zudem ist eine kombinatorische Testregion die kleinste zu testende Region zu einem bestimmten Zeitpunkt. Durch Traversierung der Abhängigkeitskette sind zunächst alle kombinatorischen Testregionen erhältlich, welche die Zielfunktion beeinflussen. Diese werden anschließend über einen Abhängigkeitsbaum nach Abbildung 5.5b verkettet. Das Taktnetz und dessen Ressourcen werden im Abhängigkeitsbaum bei vielen Schaltungen an die Knotenebene 1 gesetzt.



(a) Kombinatorischer Testregion **(b)** Abhängigkeitskette durch Traversierung

Abbildung 5.5.: Bestimmung der *Einhüllenden Testregion*

Durch die Aufteilung in kombinatorische Testregionen ist die Takt-Periodendauer T_{clk} die maximale zeitliche Auflösung für den Testzeitpunkt eines MFs. Diese maximale Auflösung kann aufgrund der Größe der MF nicht ausgereizt werden. Eine Beispielrechnung hierzu ist in Anhang A.4.1 zu finden. Hier müsste eine Schaltung mit mindestens 4000 sequentiellen nachgeschalteten Regionen existieren, um für ein einzelnes MF gezielt eine Testzeit zu definieren. Diese Bedingung wird von den gängigen Schaltungen nicht erfüllt, die lediglich 5 bis 10 kombinatorische Testregionen aufweisen. Das liegt vor allem an den kombinatorischen Brücken

durch Feedback-Pfade zwischen den Gattern. Somit werden für die gesamte Teilschaltung einer Funktion die kombinatorischen Regionen zusammengefasst und nach Satz 5.2 eine dedizierte einhüllende Testregion definiert und bestimmt. Die kombinatorischen Testregionen bleiben dennoch latent erhalten und bieten im späteren Verlauf der Erstellung der Testsequenz die Möglichkeit der Variation der Fehlerabdeckung.

Satz 5.2 *Für einen PKT einer Funktion wird eine Testregion mit* $R_F :\subseteq \{M_1, M_2, \ldots, M_N\}$ *pro Funktion definiert, wobei* N *die Anzahl aller verfügbaren MFs ist. Es werden diejenigen MFs* M *eine Teilmenge von* R_F*, die bei einem Einzelfehler den Ausgang der Funktion beeinflussen können.*

Für eine Beurteilung der Fehlerabdeckung im CRAM einer Funktion während des Scheduling und zur Vereinfachung des Umgangs mit dem Ressourcenverbrauch bedarf es einer normierten Metrik für die essentiellen Konfigurationsbits einer Funktion innerhalb eines MFs und einer Testregion R_F. Die Metrik soll herstellerunabhängig sein und die Anzahl der verwendeten Konfigurationsbits ersetzen. Es wurden alle kritischen Instanzen des FPGAs in Hinblick auf die Anzahl ihrer verwendeten Konfigurationsbits verglichen und auf das Maß eines CLB normiert. Somit kann in einfacher Schreibweise die Menge der verwendeten Konfigurationsbits einer Funktion in einem MF angegeben und mit anderen Funktionen in diesem Frame verglichen werden.

Definition 5.3 *Die Metrik zur Konfigurationsnormierung* $\Phi_{MF,F}$ *basiert auf dem Verhältnis der Anzahl der Konfigurationsbits verschiedener Instanzen des FPGA.* $\Phi_{MF,F}$ *bezeichnet demnach die Menge der Konfigurationselemente innerhalb einer Funktion und innerhalb eines MFs, normiert auf die Konfigurationselemente eines CLB. Die Grundeinheit ist 10.*

$$\Phi_{MF,F} = N_{MF,F,CLB} \cdot 10 + N_{MF,F,INT} \cdot 30 + N_{MF,F,CLB_{PIP}} \cdot 5 + N_{MF,F,Inst} \cdot 50 \tag{5.1}$$

$\Phi_{MF,F}$	=	Normierte Anzahl an Konfigurationselementen pro Funktion in einem MF
$N_{MF,F,CLB}$	=	Anzahl der verwendeten CLBs im MF
$N_{MF,F,INT}$	=	Anzahl der verwendeten Verbindungsknoten im MF
$N_{MF,F,CLB_{PIP}}$	=	Anzahl der verwendeten lokalen CLB Verbindungsknoten im MF
$N_{MF,F,Inst}$	=	Anzahl weiterer Instanzen im MF

Die Gewichtungsfaktoren der Instanzen sind anhand der Verhältnisse der zugehörigen Konfigurationsbits festgelegt. So hat beispielsweise ein Verbindungsknoten etwa dreimal so viele Konfigurationsbits wie ein CLB. Bei den Gewichtungen ist das Verhältnis der Konfigurationsbits einzelner Instanzen entscheidend und nicht die absolute Größe.

Mit Hilfe von $\Phi_{MF,F}$ kann die Testabdeckung des Scheduling überwacht werden. Die Summe aller $\Phi_{MF,F}$ einer Funktion entspricht einer vollständigen Fehlerabdeckung durch den PKT dieser Funktion. Ferner werden kritische Instanzen oder Primitiven mit wenigen CRAM Bits innerhalb eines MFs bei einem Konflikt im Scheduling durch einen hohen Gewichtungsfaktor priorisiert.

5.2.1.4. Priorisierung der Testregionen anhand der zeitlichen Anforderungen

Der Einsatz von PKT ist bei Systemen mit unterschiedlichen FDZ pro Funktion geeignet. Somit sind parallel mehrere einhüllende Testregionen nach Satz 5.2 für eine Priorisierung vorhanden. Bei überlagerten Funktionen mit $\Phi_{i,m} > 0$ und $\Phi_{i,n} > 0$ wird dieser Frame i der Region mit der kleinsten FDZ zugeordnet. Die Entstehung

der priorisierten Testregionen R'$_F$ ist in Abbildung 5.6 dargestellt und über Gleichung 5.2 beschrieben. Hier werden MFs eindeutig einer Testregion R'$_F$ zugewiesen.

(a) Einzelnutzung der Frames **(b)** Priorisierte Testregionen

Abbildung 5.6.: Priorisierung der Testregionen mit $\mathrm{FDZ}_{F1} < \mathrm{FDZ}_{F2}$

$$R'_m = R_m \setminus \{M \mid \{M \in R_m, \exists F \text{ mit } M \in R_F, FDZ_F < FDZ_m\}\} \tag{5.2}$$

5.2.2. Abdeckung der CRAM Fehlermodelle

Platzierte Instanzen lassen sich lokalisieren und mittels eines PKT abdecken. Somit sind die CRAM Fehlermodelle für konfigurierbare Logikinstanzen wie Slices, BRAM, DSP Einstellungen sowie I/O Blöcke, Signaltreiber und weitere dedizierte Blöcke innerhalb des FPGAs zu 100 % berücksichtigt. Die globalen logik-unabhängigen FPGA Einstellungen mit dem Block Typ 3 und 4 haben nach Tabelle A.2 im Anhang A.4.2 einen Anteil von 0,2 % des gesamten ECC Testbereiches. Diese Frames können für einen PKT eine eigene Testregion R$_F$ zugewiesen bekommen oder werden einer dedizierten Funktion zugewiesen. Für Verbindungsleitungen müssen die Fehlermodelle aus Abschnitt 2.3.2 eingehalten werden. Diese betreffen SEU und Haftfehler für logische Verbindungen, statische Signale und verteile Netze.

5.2.2.1. Theoretische Fehlerabdeckung von Verbindungen

Die PIPs der logischen Verbindungen und statischen Signale werden zu 100 % erfasst. Abbildung 5.7 illustriert die Abdeckung der unterschiedlichen Signalformen innerhalb eines PKT. Durch die Punkt-zu-Punkt Verbindung der Verbindungsknoten der Logik können Defekte außerhalb der konfigurierten Verbindungsknoten keine Fehler hervorrufen. Dieser Abschirmungseffekt bei direkten Verbindungen wird in Abbildung 5.7 illustriert. Ferner sind auch alle Verbindungsknoten im PKT berücksichtigt, die das statische Routing weiterleiten. Hier liegt nur eine lokale Ausbreitung vor und der gesamte MF ist im Test inkludiert. Somit sind alle weiteren potentiellen Senken und PIPs außerhalb der erfassten Knoten bereits durch den MF abgedeckt.

Die verteilten Netze im FPGA Layout sind nach der ersten Korrelation des platzierten Designs nicht zu 100 % abgedeckt. Die Ursache liegt in den verteilten Takttreibern einer Taktregion. Hier sind nur die zur Verteilung konfigurierten Takttreiber erfasst. Somit ist das Fehlermodell des Taktausfalls abgedeckt. Ein Kurzschluss im Taktnetz wird mit einer Abdeckung von 100 % innerhalb des PKT detektiert, da die betroffenen Verbindungsknoten bereits in der Testregion enthalten sind. Latent angeschlossene Treiber ohne zugeschalteten PIP sind potentielle Einzelfehler für eine Jittererhöhung und sind nicht in der einhüllenden Testregion enthalten. Nach ISO 26262 werden diese Fehlerursachen als *Residual Faults* eingestuft. Die globalen

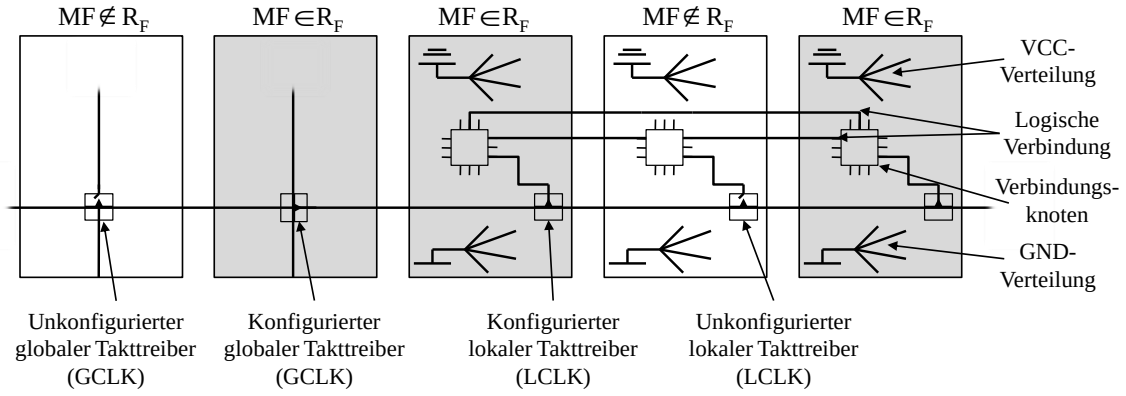


Abbildung 5.7.: Abdeckung des Verbindungsgitters innerhalb des PKT

Takttreiber (GCLK) haben gemäß Anhang A.3 einen Anteil von 3,2 % an der Logikkonfiguration. Davon sind die essentiellen globalen Takttreiber in R_F enthalten, jedoch nicht die latenten globalen Elemente. Sie reduzieren die Fehlerabdeckung des PKT nach Gleichung 5.4. Im Gegensatz dazu liegen die lokalen Takttreiber (LCLK) im CLB. Der Anteil der lokalen Takttreiber pro CLB MF liegt bei kleiner als 0,1 %[2] gegenüber den gesamten Konfigurationselementen. Zudem werden bei platzierten Funktionen zwangsläufig latente Takttreiber innerhalb von MF mit getestet. Somit können die latenten lokalen Takttreiber im CLB MF wie die globalen Takttreiber eingestuft werden und müssen lediglich im Rahmen einer Vermeidung von Fehlerakkumulationen über einen Fahrzyklus getestet werden. In kritischen Fällen kann für MFs mit latenten Takttreibern eine eigene Testregion mit FDZ Vorgaben erstellt werden.

In dem vorgestellten Verfahren des PKT werden mehrere Funktionen priorisiert und die verwendeten Ressourcen weitestgehend unabhängig getestet. Dadurch ergibt sich eine unterschiedliche Fehlerabdeckung pro Zeitintervall für jede Funktion. Für eine Evaluierung der Testzeiten und Abdeckungen einzelner Funktionen sowie die Einstellung der maximalen Fehlerabdeckung pro Funktion wird die *Funktionale Konfigurations-Fehlerabdeckung* $Q_{F,Det}$ definiert.

Definition 5.4 *Die Funktionale Konfigurations-Fehlerabdeckung (FKF)* $Q_{F,Det}$ *gibt die Detektionswahrscheinlichkeit eines Fehlers im essentiellen CRAM einer Funktion an. Sie entspricht dem Verhältnis aus getesteten FPGA Konfigurationselementen einer Funktion zu den gesamten Konfigurationselementen einer Funktion. Die Konfigurationselemente einer Funktion können bei einem Einzelfehler im CRAM den Ausgang einer dedizierten Funktion beeinflussen.*

Aufgrund der detektierbaren CRAM Fehlermodelle aus Abschnitt 5.2.2 eines PKT kann im schlechtesten Fall eine maximale Fehlerabdeckung FKF von 96,7 % für die Detektion erreicht werden. Die Wahrscheinlichkeit einer Fehlerkorrektur mittels SECDED ECC liegt für den PKT bei mindestens 58,5 %. Die Korrekturwahrscheinlichkeit kann durch nachträgliche Verifikation der Korrektur über eine goldene Referenz oder mittels partiellem Scrubbing deutlich erhöht werden.

[2] Bestimmung über Xilinx Tool `xdl.exe -report`

$$Q_{F,Det} = \frac{\nu}{N} \cdot 100 \cdot Q_{ECC,Det} = \frac{\sum_{i \in R_F} \Phi_{i,F}}{\Phi_F} \cdot 100 \cdot Q_{ECC,Det} \tag{5.3}$$

$$\begin{aligned} Q_{F,Det,max} &\geq \left(1 - \frac{N_{GCLK} + N_{LCLK}}{N_{CLB} + N_{IO} + N_{DSP,BRAM} + N_{GCLK} + N_{LCLK}}\right) \cdot 100 \cdot Q_{ECC,Det} \\ &\geq 96,7\,\% \end{aligned} \tag{5.4}$$

$$\begin{aligned} Q_{F,Korr,max} &= Q_{F,Det,max} \cdot Q_{ECC,Korr} \\ &\approx 58,5\,\% \end{aligned} \tag{5.5}$$

ν	=	Anzahl der getesteten Konfigurationselemente pro Funktion
N_X	=	Anzahl der gesamten Konfigurationselemente (eines Typs X) einer Funktion
$\Phi_{i,F}$	=	Konfigurationsnormierung der Funktion F in einem MF
Φ_F	=	Konfigurationsnormierung der Funktion F aller Instanzen und Primitiven
$Q_{ECC,Det}$	=	Wahrscheinlichkeit für eine Fehlerdetektion mit ECC. Nach Abschnitt 3.1.2: 99,99 %
$Q_{ECC,Korr}$	=	Wahrscheinlichkeit für eine Fehlerkorrektur mit ECC. Nach Abschnitt 3.1.2: 60,53 %

5.2.2.2. Varianz der Fehlerabdeckung

In bestimmten Anwendungsfällen kann die Fehlerabdeckung $Q_{F,Det}$ zugunsten einer kürzeren FDZ weiter verringert werden. Hier wird die einhüllende Testregion um eine Anzahl von MFs verringert. Hierfür stehen zwei Methoden zur Verfügung.

- MF mit geringem Anteil an Konfigurationselementen - Die Frames mit der kleinsten Beitrag an Konfigurationselementen werden exkludiert.
- MF mit geringer Abhängigkeit auf den Ausgang - Hier wird die Abhängigkeitskette aus Abbildung 5.5b mit den Frames korreliert. Die Wahrscheinlichkeit einer Fehlerpropagation am Ausgang sinkt mit jeder Knotenebene. Daher werden MF exkludiert, dessen Ressourcen die höchste Ebenen im Abhängigkeitsbaum haben.

Die Flexibilität der Testregionen ermöglicht eine feingranulare Exkludierung von MF für eine Funktion. Diese ignorierten Ressourcen können entweder ungetestet bleiben oder sie werden einer Testregion mit flexiblerer FDZ zugeordnet. Für einen sicherheitsrelevanten Test sollten jedoch nicht mehr als 10 %[3] der Konfigurationselemente exkludiert werden. In der Praxis haben sich 1 - 5 % exkludierte Ressourcen als wirksame Methode für die Reduzierung der FDZ bewährt.

5.3. Zeitgesteuerte Testverteilung

In diesem Abschnitt werden zeit-deterministische Systeme nach der *Time-Triggered* Methode betrachtet. Hierbei sind die Ausgänge einer Funktion oder eines Schaltungsmodules gemäß Abbildung 5.3a lediglich in gesteuerten Zeitschlitzen aktiv. In Anwendungen wie Bildverarbeitung oder Kommunikation sind die Ausgänge periodisch aktiv und erfüllen die Anforderung nach Satz 5.5. Bei sicherer Auslegung der Ausgangs-Ansteuerung

[3]Entspricht einer Fehlerabdeckung für Einzelfehler von kleiner 90 % mit 10 % Residual Faults für CRAM SEU und Haftfehler

kann auch im Fehlerfall innerhalb eines Schaltungsmodules der Fehler nicht nach außen propagieren. Hierfür können die gesicherten Ports und Enable-Signale mittels TMR zuverlässig ausgelegt werden.

Die Anforderungen an eine zeitgesteuerte Testverteilung sind in Satz 5.5 zusammengefasst.

Satz 5.5 *Für einen vollständigen PKT können zeitgesteuerte Datenflüsse einer Funktion dann angewendet werden, wenn*

(1) Für alle Aktivierungen und Propagationen des Ausgangs die Zeitbasis (B) und die Deadline (D) innerhalb einer Hyperperiode bekannt sind. Die Länge der Hyperperiode aller Funktionen bestimmt die Periodendauer der Testsequenz des PKT und somit auch deren Länge.

(2) Nach einer Datenausgabe mit der Zeitbasis (B_i) für die Zeitdauer der gesetzten FDZ mit der Deadline (D_i) keine weitere Datenausgabe erfolgt.

(3) Die minimale Testzeit der Testregion R_F kleiner als das Zeitintervall zwischen B_i und D_i einer Testanforderung ist.

Ist die minimale Testzeit für R_F größer als die FDZ, kann mit einer Verzögerung $t_{d,i+1}$ des nächste Datenausgangs erfüllt werden. Es muss sichergestellt sein, dass diese minimale Testzeit kleiner wird als die Summe aus der gesetzten FDZ und der Verzögerung $t_{d,i+1}$. Ein neuer PKT derselben Funktion mit B_{i+1} wird erst nach einer kompletten Abarbeitung nach einer Basis B_i gestartet.

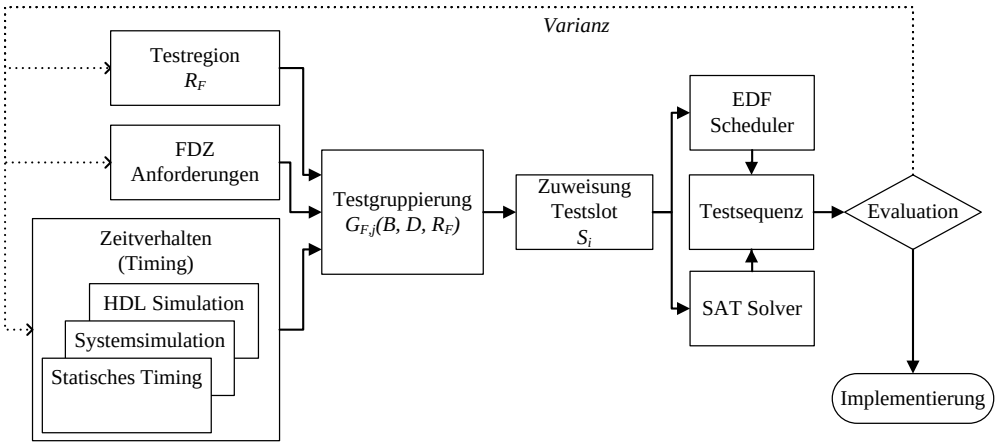


Abbildung 5.8.: Ablaufgraph für Zeitgesteuerte Testverteilung

Bei Beachtung der Anforderungen aus Satz 5.5 reduziert sich die maximale Testzeit für den CRAM gegenüber der Testzeit t_{LKT} aus Gleichung 2.4 auf die gesetzte FDZ. Die erforderliche Testsequenz auf Basis der Daten für B und D wird mittels Scheduling erstellt. Die notwendigen Schritte sind in Abbildung 5.8 dargestellt. Nach der Gruppierung der Testanforderung in zeitlich arrangierten Testgruppen erfolgt die Verteilung auf die Testslots. Anschließend werden mittels Scheduling Algorithmen die Testslots mit MF belegt. In dieser Arbeit werden zwei Scheduling Methoden näher erläutert. Zum einen kommt ein an den Konfigurationstest angepasster *Earliest Deadline First (EDF)* Scheduler zum Einsatz. Dieser als EDF-KT bezeichnete Variante basiert auf heuristischen deadline-sensitiven Methoden nach Liu und Layland (1973) [LL73]. Zum anderen wird eine aussagenlogische Methode über *Satisfiable (SAT) Problem Solver* verwendet. Diese beruht auf der booleschen Kodierung von Scheduling Problemen nach Hebrard (2012) [Heb12]. Abschließend wird die

Fehlerdetektionszeit aus der erstellten Testsequenz bestimmt und mit den FDZ verglichen. Wird die FDZ überschritten, kann eine Variation der Testregionen oder der Testanforderungen das Scheduling beeinflussen.

5.3.1. Gruppierung der Testanforderung und Verteilung auf Testslots

Die Testverteilung beginnt mit der Erstellung von Testgruppen $G_{F,j}$ innerhalb der Hyperperiode. Hierfür wird für jede Funktion mit gesetzter FDZ eine zeitlich abhängige Basis $B_{F,j}$ und Deadline $D_{F,j}$ bestimmt. Der Funktionsindex wird mit F bezeichnet und der zeitliche Index mit j. Wie in Abbildung 5.8 dargestellt, können $B_{F,j}$ und $D_{F,j}$ aus statischen Angaben oder aus simulativen Zeitverhalten gefiltert werden. Ferner werden jeder Testgruppe $G_{F,j}$ die zu testenden Ressourcen R_F zugewiesen.

Anschließend werden die Testgruppen auf die physikalisch möglichen Testslots S_i nach Gleichung 5.6 und Abbildung 5.9 verteilt. Die Testslots werden mit $S_i := \{S_0, S_1, \ldots, S_i\}, i = \lceil \frac{t_{HP}}{t_S} \rceil$ definiert. Hier entspricht t_{HP} der Periodendauer der Hyperperiode aller Funktionen und somit die Länge der Testsequenz. Gleichung 5.6 besagt eine Zuordnung von einer Testgruppe $G_{F,j}$ zu S_i unter der Bedingung, dass die zu G gehörende Basis kleiner gleich und die zu G gehörende Deadline größer als die Zeit des aktuellen Testslots ist.

$$S_i := \left\{ G_{F,j} \middle| \frac{B_{F,j}}{t_S} \leq i \ \wedge \ \frac{D_{F,j}}{t_S} > i \right\} \tag{5.6}$$

Nun kommt es zu einem Scheduling, da bei einem FPGA mit verteilten Funktionen in jedem Testslot mehrere Testgruppen liegen können.

5.3.2. EDF Scheduling

Die heuristische Testverteilung für den PKT wird mittels einer adaptierten EDF Methodik durchgeführt. Der EDF Ansatz wird in vielen Software Scheduling Problemen für Einzel- und Mehrkernprozessoren verwendet und auch als Earliest Due Date (EDD) bezeichnet [SAA+04]. Für die vorliegende Testverteilung werden die Deadlines $D_{F,j}$ als Entscheidungskriterium festgelegt. Die Testgruppe mit der kleinsten Deadline innerhalb von S_i wird in diesem Testslot favorisiert. Es wird das häufigste MF aus allen konkurrierenden Testgruppen dieses Slots ausgewählt. Dieses Vorgehen ist in Abbildung 5.9 illustriert. Ist eine Testgruppe bereits vor Ablauf der Deadline verteilt, dann wird sie bis zu dessen Deadline ignoriert. Bei gleicher Deadline wird die Testgruppe mit den wenigsten offenen MF favorisiert. Die wichtigsten Schritte sind in Algorithmus 1 in der Funktion `EDF_Scheduling` enthalten.

Neben der gemeinsamen Deadline Behandlung gibt es in EDF-KT dedizierte Änderungen gegenüber dem Standard-Algorithmus. Nicht ausgewählte Testgruppen eines Slots werden im Falle einer prognostizierten Deadline Überschreitung zu den Slots außerhalb der Deadline hinzugefügt. So geht trotz Überschreitung der Deadline die Testanforderungen nicht verloren. Ferner sind die Testgruppen durch gemeinsam genutzte MF nicht unabhängig und beeinflussen den Testverlauf voneinander. Eine weitere Adaption des EDF-KT ist das *präemptive* Testen der Testgruppen. Diese können jederzeit unterbrochen werden zugunsten einer höher priorisierten Gruppe. Es wird jeder Testslot während des PKT ohne Kontextwechsel angesteuert und somit müssen die Testgruppen nicht in einem zusammenhängenden Intervall bearbeitet werden.

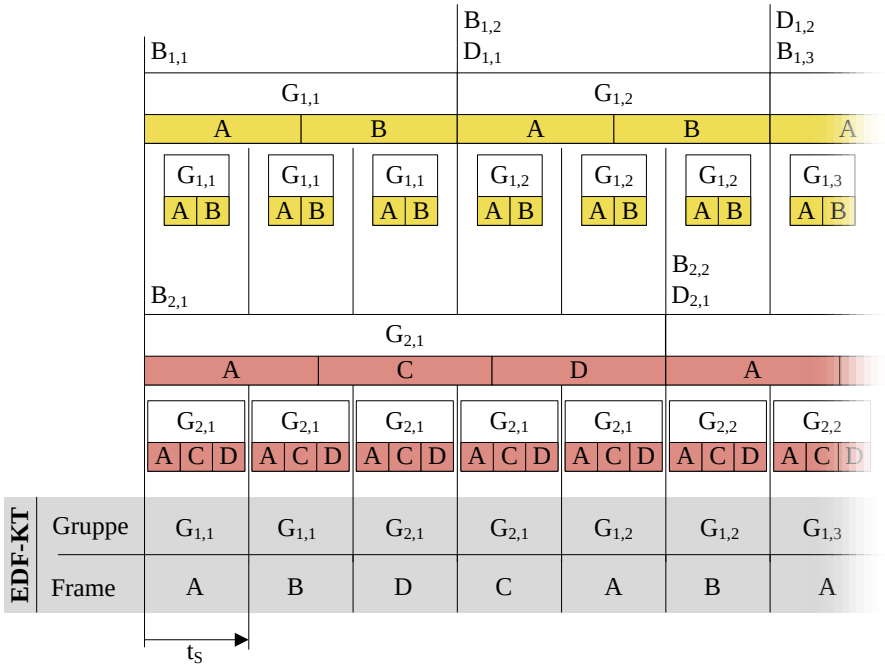


Abbildung 5.9.: EDF-KT Scheduling

Im Zuge der Untersuchungen wurden dem EDF-KT zusätzliche Eigenschaften hinzugefügt. Zum einen wurde bei Überschreitung der Deadlines ein Malus in Form von einer virtuellen Verkürzung der Deadlines eingeführt. Damit sollen überzogene Deadlines höher priorisiert werden. Zum anderen können Testgruppen mit langen Deadlines zufällig priorisiert werden. Dadurch wird eine dauerhafte Vernachlässigung dieser Testgruppen zugunsten von Gruppen mit kürzeren Deadlines vermieden. Beide Ansätze zeigten in den Ergebnissen keine Verbesserung des gesamten Scheduling in Bezug auf Überschreitung von Deadlines. Die Umpriorisierung der Testgruppen durch Malus und zufälliger Auswahl widerspricht dem EDF-Prinzip und verursacht somit zum Teil schlechtere Ergebnisse. Dennoch sind diese Optionen im PKT-Tool enthalten und können je nach Anwendungsfall angewendet werden.

5.3.3. Scheduling mit SAT

Der verwendete EDF Algorithmus basiert auf heuristischer Verteilung und kann je nach Anwendungsfall zu vielen Überschreitungen der Deadlines liefern. Bei der Aussage der Erfüllung der gesetzten Deadlines handelt es sich auch um ein entscheidbares Erfüllbarkeitsproblem (SAT). Mittels SAT kann bei entsprechender Kodierung die Erfüllbarkeit des Scheduling überprüft werden [HH12, Heb12]. Für entsprechende Lösungsalgorithmen muss das Scheduling in eine aussagenlogische Formel mit booleschen Werten gebracht werden. Die Lösungsfindung hingegen ist NP-vollständig und kann je nach Komplexität eine tolerable Rechenzeit überschreiten.

SAT Probleme werden mit booleschen Ausdrücken beschrieben. Als Standardformatierung hat sich hier die Konjunktive Normalform (KNF) etabliert [DIM93]. In der KNF werden ODER-verknüpfte Variablen (Disjunktionstherme) über UND-Operatoren verbunden (Konjunktion) und in der Form $\bigwedge_i \bigvee_j (\neg) X_{ij}$ kodiert. Für einen PKT wird jeder verfügbare MF aller R_F für jeden Testslot der Sequenz durch eine neue Variable repräsentiert. Diese Kodierung wird in Abbildung 5.10 dargestellt. Die Einträge der entstandenen Variablenmatrix werden nun mit den Eigenschaften der Testgruppen $G_{F,j}$ und Testslots S_i verknüpft. Ein

Algorithm 1 Angewendeter EDF Algorithmus

```
1:  function EDF_Scheduling
2:      for k=0 to i do
3:          G_min = null;
4:          for all G ∈ S_k do
5:              if D_G_min > D_G UND R_G ≠ ∅ then          ▷ Finde G mit der kürzesten Deadline
6:                  G_min = G
7:              else if D_G_min == D_G UND R_G ≠ ∅ then    ▷ Bei gleicher Deadline
8:                  if R_G < R_G_min then                  ▷ Kleinere Testregion priorisieren
9:                      G_min = G
10:                 end if
11:             end if
12:         end for
13:         M = null;
14:         for all G ∈ S_k do
15:             if R_G ∩ R_G_min ≠ ∅ then                  ▷ Finde gemeinsamen MF
16:                 M = R_G ∩ R_G_min
17:             end if
18:         end for
19:         for all G ∈ S_k do
20:             R_G = R_G \ M                              ▷ Reduziere verbleibende Testgruppen um M
21:             if (R_G + k) · t_test,MF > D_G then
22:                 S_k+R_G+1 = S_k+R_G+1 ∪ R_G            ▷ Verzögere Testgruppen bei potentiellem Deadline Miss
23:             end if
24:         end for
25:     end for
26: end function
```

Testslot kann maximal einen MF testen und wird somit über eine `AtMostOne` Kodierung nach Heule und Hunt (2012) [HH12] repräsentiert. Für einen Testslot S_i gilt demnach Gleichung 5.7. Die `AtMostOne` Kodierung erzeugt 3n-6 Klauseln und (n-3)/2 Hilfsvariablen, wobei n die Anzahl der Variablen ist. Des Weiteren werden die Deadlines $D_{F,j}$ einer Testgruppe $G_{F,j}$ durch eine ODER Verknüpfung nach Gleichung 5.8 kodiert. Die Kodierung der Testgruppe G_{11} und des Testslots S_1 ist in den Gleichungen mit aufgeführt.

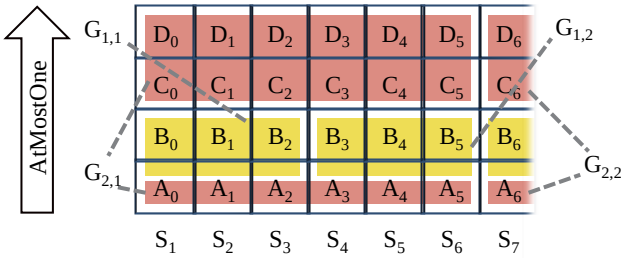


Abbildung 5.10.: Variablen der SAT Kodierung des PKT Scheduling

Nach der KNF-Kodierung kann die Lösung mittels SAT Solver ermittelt werden. Die Berechnung eines SAT Solvers kann je nach Problemgröße mehrere Tage beanspruchen und dabei Lösungsräume größer $2^{1.000.000}$ bearbeiten. In dieser Arbeit wird der SAT Solver "Treengeling" von Biere (2013) [Bie13] verwendet, da dieser parallel auf Mehrkern Prozessoren arbeitet und somit eine schnelle Berechnung liefert. Das Ergebnis ist eine WAHR/FALSCH Entscheidung. Sollte das Ergebnis WAHR sein, dann wird eine mögliche Lösung angegeben. Die Lösung wird dekodiert und liefert aus den Variablen die MFs nach dem Schema aus Abbildung 5.10 in MF dekodiert werden. Hierfür wird während der Kodierung des SAT Problems die Zuordnung der einzelnen booleschen Variablen zu den Frameadressen in eine externe Datei geschrieben. Anschließend kann in der Dekodierungsphase die Testsequenz aus dem booleschen Lösungsraum rekonstruiert werden.

$$\begin{aligned} \text{Exklusivität in } \mathrm{S}_i &= \texttt{AtMostOne}(M_{i0}, M_{i1}, \ldots, M_{iF}) \\ \text{Exklusivität in } \mathrm{S}_1 &= \texttt{AtMostOne}(A_0, B_0, C_0, D_0) \end{aligned} \tag{5.7}$$

$$\begin{aligned} D_{F,j,kodiert} &= (M_{k,S_j,G_F} \vee M_{k,S_{j+1},G_F} \vee \ldots \vee M_{k,S_{j+D},G_F}) \\ D_{11,kodiert} &= (A_0 \vee A_1 \vee A_2) \wedge (B_0 \vee B_1 \vee B_2) \end{aligned} \tag{5.8}$$

M_k = MF einer Testregion R_F
M_{iF} = MF der SAT Kodierungsmatrix aus Testslot i und Funktion F

5.4. Homogene Testverteilung

Nach der Einführung der zeitgesteuerten Testverteilung im vorigen Abschnitt wird nun eine priorisierte Testverteilung ohne Deadlines vorgestellt. Gemäß Abbildung 5.3b aus Abschnitt 5.1 können Funktionen im FPGA auch kontinuierlich Daten ohne Enable-Triggerung propagieren. Sobald der Ausgang einer Funktion nach einer Datenausgabe innerhalb der FDZ erneut aktiv werden kann, muss für einen PKT die homogene Testverteilung angewendet werden. Mit diesem Ansatz werden zwei Ziele verfolgt.

1. Zu jedem Zeitpunkt der Laufzeit soll ein Fehler im CRAM einer Funktion F innerhalb der gesetzten FDZ detektiert werden. Die schlechteste Testabdeckung erfolgt so schnell wie nötig und nicht so schnell wie möglich.

2. Die Fehlerabdeckung soll zu jedem Zeitpunkt linear ansteigen mit $\frac{d^2 Q_{F,Det}(t)}{dt^2} \approx 0$. Das garantiert eine homogene mittlere FDZ.

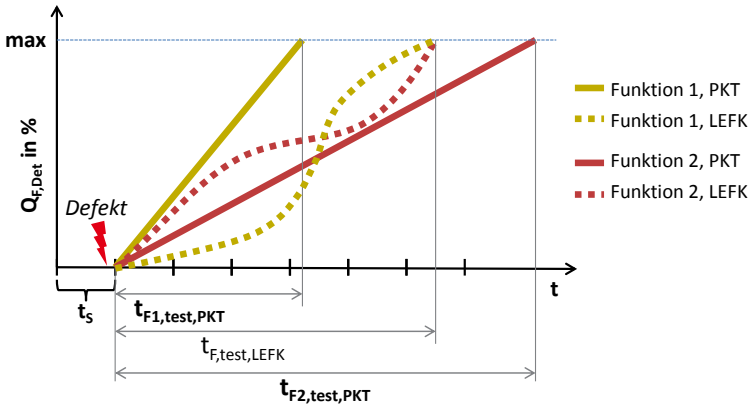


Abbildung 5.11.: Idealisierte homogene Testverteilung des PKT

Abbildung 5.11 zeigt diese Ziele anhand der Fehlerabdeckung über der Zeit. Nach einem Defekt im CRAM, dargestellt durch den Blitz, steigt die Fehlerabdeckung gegenüber dem klassischen LKT linear an. Zudem wird $\mathrm{Q}_{F,Det,max}$ je nach gesetzter FDZ erreicht. Der Gradient von $\mathrm{Q}_{F,Det}$ soll über die gesamte Laufzeit annähernd gleich sein, um eine deterministische und skalierbare Fehlerabdeckung zu gewährleisten. In diesem Zusammenhang kann bei einer skalierbaren Fehlerabdeckung der Wert von $\mathrm{Q}_{F,Det,max}$ gemäß Abschnitt 5.2.2.2 ohne Auswirkung auf $dQ_{F,Det}/dt$ reduziert werden.

In Abbildung 5.12 ist der Ablauf der homogenen Testverteilung aufgezeigt. Die Testregionen werden anhand der gesetzten FDZ mittels eines Verteilungsalgorithmus oder eines Optimierungsverfahrens der Testsequenz

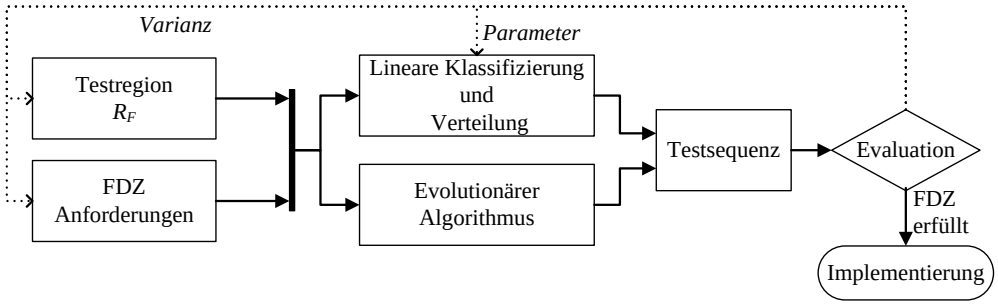


Abbildung 5.12.: Ablaufgraph für Homogene Testverteilung

zugeteilt. Die Ergebnisse werden evaluiert und können anschließend durch Beeinflussung der Platzierung, der Anforderungen oder der Algorithmenparameter variiert werden.

5.4.1. Verteilungsalgorithmus

Die heterogenen Anforderungen an die FDZ verschiedener Funktionen müssen auf eine lineare Testsequenz verteilt werden. Die erste Priorisierung der Testregionen R_F wird durch die Erstellung von R'$_F$ nach Abschnitt 5.2.1 vorgenommen. Durch diese Zuordnung werden die Vorkommnisse eines MFs in der Testsequenz über das Verhältnis des kleinsten gemeinsamen Vielfaches (kgV) aller FDZ zu der dedizierten FDZ dieser Testregion bestimmt. Diese Häufigkeit eines MFs in der Testsequenz ist in Abbildung 5.13 mit N bezeichnet. Die Zuordnung von Frames basiert auf dem vorher gezeigten Beispiel von Abbildung 5.6. Die Häufigkeit aller MFs bestimmt demnach die Anzahl der Einträge in der Testsequenz L_{seq} nach Gleichung 5.9, wobei F die Anzahl der Funktionen mit gesetzter FDZ ist. Die Testperiode wird aus $L_{seq} \cdot t_S$ bestimmt.

MF	Φ_1	Φ_2	N
A	$\Phi_{A,1}$	$\Phi_{A,2}$	n
B	$\Phi_{B,1}$		n
C		$\Phi_{C,2}$	m
D		$\Phi_{D,2}$	m

(a) Zuordnung von $\Phi_{MF,F}$ und der Häufigkeit von Frames

MF	$\overline{\Phi_M}$	N
C	$\Phi_{C,2}$	m
A	$f(\Phi_{A,1}, \Phi_{A,2})$	n
D	$\Phi_{D,1}$	m
B	$\Phi_{B,1}$	n

(b) Sortierung der Frames durch Gewichtung der Konfigurationsnormierung

Abbildung 5.13.: Übergang paralleler Testanforderungen zu einer linearen Klassifizierung mittels Konfigurationsnormierung

$$L_{seq} = \sum_{j=1}^{F} \frac{kgV(FDZ_1, FDZ_2, \ldots, FDZ_F)}{FDZ_j} \cdot R'_j \tag{5.9}$$

Die Häufigkeit eines Frames in der Testsequenz bestimmt die Testzeit unter der Voraussetzung einer Gleichverteilung. Durch eine *lineare Klassifizierung* können die MFs nun verteilt werden. Dabei sollen die Frames mit einem hohen Einfluss auf die Fehlerabdeckung beginnend in die Sequenz einsortiert werden. Bei späterer Einsortierung kann aufgrund der Vielzahl der MFs eine Gleichverteilung nicht mehr gewährleistet werden. Der Einfluss dieser Inhomogenität auf die Fehlerabdeckung soll durch die Klassifizierung verringert werden und die Abweichung vom Ziel $\frac{d^2 Q_{F,Det}(t)}{dt^2} \approx 0$ wird verkleinert. Für die lineare Klassifizierung der MFs wird die Konfigurationsnormierung aus Definition 5.3 des Abschnittes 5.2.1.3 zur einhüllenden Testregion angewendet.

Diese Metrik bestimmt den Beitrag eines Frames zu einer Funktion und wird in Tabelle 5.13a für jeden Frame explizit aufgeführt.

5.4.1.1. Lineare Klassifizierung der Frames

Für Frames mit mehreren implementierten Funktionen, wie beispielsweise Frame A, muss die Klassifizierung über alle beteiligten Funktionen hinweg berechnet werden. Aufgrund der geforderten Homogenität für alle Funktionen wird hier eine Abwägung von Φ_{A1} und Φ_{A2} für die lineare Klassifizierung erfolgen. Hierfür sind drei Wägeverfahren angewendet und das qualitative Ergebnis der Gewichtung ist in Tabelle 5.13b in der Spalte $\overline{\Phi}_M$ dargestellt.

1. *Mittelwert aller* $\Phi_{M,F}$: Es wird der Mittelwert aller $\Phi_{M,F}$ in einem Frame bestimmt. Hierbei wird der Anteil an der Gesamtfunktion vernachlässigt.

$$\overline{\Phi}_M = \frac{1}{F} \sum_{j=1}^{F} \Phi_{M,j} \tag{5.10}$$

2. *Maximales Verhältnis von* $\Phi_{M,F}$ *zu* Φ_{ges}: Es wird das Verhältnis jedes $\Phi_{M,F}$ in einem Frame zu der Gesamtsumme aller $\Phi_{M,F}$ jeder Funktion und jedes Frames gebildet. Anschließend wird der Maximalwert pro Frame für die Klassifizierung weiter verwendet. Hierbei wird die Gewichtung einzelner Funktionen in der Gesamtschaltung vernachlässigt.

$$\overline{\Phi}_M = \max \left(\frac{\Phi_{M,f}}{\sum_{j=1}^{M} \sum_{k=1}^{f} \Phi_{j,k}} \right), \forall f \in F \tag{5.11}$$

3. *Maximales Verhältnis von* $\Phi_{M,F}$ *zu* Φ_F: Es wird das Verhältnis jedes $\Phi_{M,F}$ in einem Frame zu der Gesamtsumme aller $\Phi_{M,F}$ jeder Funktion gebildet. Anschließend wird der Maximalwert pro Frame für die Klassifizierung weiter verwendet. Hier werden große Unterschiede in der Beteiligung eines Frames zu unterschiedlichen Funktionen vernachlässigt.

$$\overline{\Phi}_M = \max \left(\frac{\Phi_{M,f}}{\sum_{j=1}^{M} \Phi_{j,f}} \right), \forall f \tag{5.12}$$

Das jeweilige Wägeverfahren hat für bestimmte Anwendungen Vor- und Nachteile. Diese hängen stark von den Beiträgen eines Frames zu unterschiedlichen Funktionen und von den Größenverhältnissen der Funktionen untereinander ab. Im PKT Tool können alle drei Optionen angewendet werden.

Nach der Klassifizierung werden die MFs in die Testsequenz mit einer Periode P_M und der Anzahl N eingefügt. Die Periode bestimmt sich durch Gleichung 5.13. Die Frames mit dem höchsten Wert für $\Phi_{M,F}$ und somit mit dem höchsten Einfluss auf alle Funktionen starten mit der Verteilung in der Testsequenz mit der Periode P_M. Bei kleiner werdender Klassifizierung eines Frames in Tabelle 5.13b steigt die Wahrscheinlichkeit auf eine inhomogene Verteilung in der Testsequenz. Diese Inhomogenität wird durch die Umgebungssuche nach freien Slots in der Sequenz hervorgerufen. Kann ein Frame aufgrund einer besetzten Position nicht mit der Periode

P_M in die Testsequenz eingetragen werden, dann wird nach einer freien benachbarten Position gesucht.

$$P_M = \frac{L_{seq}}{N} = \sum_{k=1}^{F} \frac{FDZ_M}{FDZ_k} \cdot R'_k \tag{5.13}$$

Diese heuristische Verteilung ermöglicht eine schnelle Erstellung der Testsequenz und nimmt Rücksicht auf die Fehleranfälligkeit eines MFs in Bezug auf die Gesamtfunktion oder Gesamtschaltung. Bei Schaltungen mit vielen Funktionen und unterschiedlichen FDZ kommt es jedoch zu einer Inhomogenität in der Fehlerabdeckung in Form eines Jitters bei der FDZ über die Laufzeit.

5.4.1.2. Trimmung der Testverteilung

Nach der linearen Klassifizierung und vor der tatsächlichen Verteilung in die Testsequenz können die Einträge in der gewichteten Liste aus Tabelle 5.13b noch variiert werden. Dadurch können bestimmte Funktionen oder Verläufe von Verteilungen hervorgehoben werden. Eine effektive Maßnahme ist die EDF-Verteilung gemäß dem Prinzip der Priorisierung der kürzesten FDZ. Dazu werden nach der Sortierung durch $\overline{\Phi}_M$ die Frames mit der kürzesten FDZ beginnend in die Sequenz eingefügt. Das führt zu einer homogenen Verteilung dieser Frames und vermeidet für kritische Funktionen eventuelle Artefakte in der Verteilung durch eine gestörte Periode P_M. Diese Artefakte vergrößern die schlechteste Testzeit $t_{test,F}$.

Für eine weitere Reduzierung des Jitters der FDZ wurden zwei zusätzliche Lösungsansätze evaluiert. Zum einen wurden Frames mit hohen Werten von $\overline{\Phi}_M$ zusätzlich in die Testsequenz verteilt. Der Fokus lag hier in der Erhöhung der Fehlerabdeckung in möglichst kurzer Zeit. Damit sollte $dQ_{F,Det}$ aus Abbildung 5.11 steigen und die maximale Fehlerabdeckung schneller erreichen. Dieser Ansatz priorisierte ausschließlich einzelne Frames und erzielte keine maximale Fehlerabdeckung in kürzerer Zeit.

Ein zweiter Ansatz zur Trimmung der Testverteilung geht auf die Annahme zurück, dass zum Erreichen einer kürzeren FDZ einer Funktion meist nur wenige Frames besser verteilt werden müssen. Diese fehlenden Frames sind unabhängig von $\overline{\Phi}_M$. Somit wurden zusätzliche Testslots zur freien Verfügung in die Sequenz eingefügt. Hier wählte ein Optimierungsalgorithmus bestimmte MFs mit dem größten Einfluss auf die FDZ für diesen zusätzlichen Slot aus. Im Folgenden verlängert sich jedoch die Länge der Testsequenz und der verkürzende Effekt auf eine FDZ erhöht die FDZ anderer Funktionen. Somit mussten weitere Algorithmen zur Optimierung aller FDZ evaluiert werden.

5.4.2. Optimierung der Testzeit

Der Verteilungsalgorithmus im vorigen Abschnitt kann Testsequenzen erzeugen, die im schlechtesten Fall die FDZ für eine oder mehrere Funktionen überschreiten. Alternativ dazu wird eine metaheuristische Optimierungsmethode für eine Einhaltung aller FDZ angewendet. Hierzu wurden eine Einzieloptimierung und eine Mehrzieloptimierung aus Luke (2013) [Luk13b] untersucht. Für die Einzieloptimierung wurde das Verfahren *Simulated Annealing* umgesetzt. Dieser Algorithmus ist an das physikalische Abkühlen von Werkstoffen angelehnt und imitiert die Erreichung eines möglichst energiearmen Zustandes. In der Anwendung auf den PKT wird während der Abkühlphase die Testsequenz variiert. Während der Energieberechnung nach

jeder Variation wird zuerst die schlechteste Testzeit $max(t_{F,test})$ einer Funktion über die Laufzeit bestimmt. Anschließend werden diese Zeiten zum Erhalt einer eindimensionalen Zielfunktion zu einer Summe aufaddiert mit $\sum_F max(t_{F,test})$. Somit wird die Summe aller schlechtesten Testzeiten optimiert. Dies führte zu dem Nachteil, dass es bei zwei oder mehr Funktionen zu Verschlechterungen einzelner $max(t_{F,test})$ kommen kann auch wenn die Gesamtsumme sinkt. Daraufhin wurde eine Mehrzieloptimierung implementiert.

Für das PKT-Tool kommt ebenso wie für die Checkpoint Methodenoptimierung aus Abschnitt 4.2.4 ein NSGA-II Algorithmus nach Deb et al. (2002) [DPAM02] zum Einsatz. Für einen PKT ist eine Zielfunktion die schlechteste Testzeit $max(t_{F,test})$ einer Funktion. Eine alternative Zielfunktion kann die maximale Überschreitung der FDZ sein. Die Optimierung wird durch Kombination und Mutation des Genoms erreicht. Für den homogenen PKT besteht das Genom aus MFs mit einer variablen Länge L_{seq}. In Abbildung 5.14 ist der Ablauf einer Iteration des EA dargestellt. Nach dem Start wird eine initiale Population erstellt und die Zielfunktionen von jedem Individuum berechnet.

In Algorithmus 2 ist die Berechnung der Zielfunktionen in der Funktion `Berechnung_Zielfunktionen` beschrieben. Hier enthält das Array `Schlechteste_Abdeckung[F]` die längste Zeit $max(t_{F,test})$ bis zur maximalen Fehlerabdeckungen jeder Funktion. Es wird für jeden Fehlerzeitpunkt eine vollständige Testsequenz geprüft, wobei der Fehlerzeitpunkt jeweils einem Testslot entspricht. Optional wird der minimale und maximale Gradient der zeitlichen Verläufe der Fehlerabdeckungen bestimmt. Die maximale Inhomogenität bestimmt sich dann aus der absoluten Differenz dieser Werte pro Funktion.

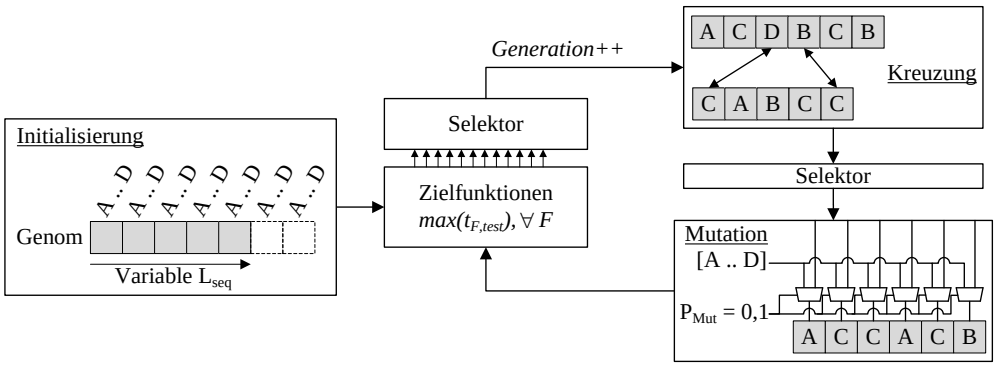


Abbildung 5.14.: Verlauf des Evolutionären Algorithmus zur Erstellung eines PKT

Nach der Selektion werden die besten Lösungen durch Kreuzung von zwei Genen kombiniert. Anschließend wird erneut eine der beiden Lösungen selektiert und mutiert. Die Mutationswahrscheinlichkeit P_{Mut} bestimmt die Variation der einzelnen Gene und die Wahrscheinlichkeit einer Längenänderung des Genoms. Der Vorgang der Selektion, Kombination und Mutation wird solange aus der Vorgängerpopulation fortgeführt, bis eine neue Population entstanden ist. Aus dieser werden wieder die Zielfunktionen der einzelnen Individuen berechnet und die Iteration startet von neuem.

An dieser Stelle wird das Tool “ECJ” ausgewählt [Luk13a], dass bereits für die Optimierung der Checkpoint Kenngrößen aus Abschnitt 4.2.4 angewendet wird. Die Anbindung des Frameworks an das PKT-Tool ist in Abbildung 5.15 dargestellt. Die Java Bibliothek ECJ ist in ein eigenes `PKT EA Tool` eingebettet und verwendet eine Nutzerklasse zur Evaluation der Zielfunktionen, hier `PktEaProblem`. Das EA Tool erhält über einen Dateiaustausch das Scheduling Problem und erzeugt nach Ablauf aller Generationen eine Liste von Dateien mit der Pareto Front. Jede Datei enthält die dazugehörige Testsequenz zur späteren Auswertung oder Verwendung.

Algorithm 2 Bestimmung der Zielfunktionen eines Individuums I für den PKT

```
1: function Berechnung_Zielfunktionen(Array I)
2:     const Maximale_Fitness = 30000
3:     Abdeckung[F][M]
4:     Schlechteste_Abdeckung[F]
5:     Maximale_Inhomogenitaet[F]
6:     Groesster_Gradient[F]
7:     Kleinster_Gradient[F]
8:     for k=0 to M-1 do                ▷ Iteration über alle MFs einer Lösung, jeder Slot kann fehlerhaft sein
9:         Abdeckung[F][k] = 0
10:        for j=k+1 to M-1 do          ▷ Vollständiger Durchlauf der Testsequenz nach dem Fehlerzeitpunkt
11:            for f=0 to F-1 do                          ▷ Iteration über alle gesetzten Funktionen
12:                Abdeckung[f][k] += I[j]
13:                if (Abdeckung[f][k] == R_f) UND (j > Schlechteste_Abdeckung[f]) then
14:                    Schlechteste_Abdeckung[f] = j
15:                end if
16:                if j % 4 == 0 then                     ▷ Gradient wird alle 4 Testslots bestimmt
17:                    gradient = (Abdeckung[F][k] - Testabdeckung vor 4 Slots) /4
18:                    if Groesster_Gradient[f] > gradient then
19:                        Groesster_Gradient[f] = gradient
20:                    end if
21:                    if Kleinster_Gradient[f] < gradient then
22:                        Kleinster_Gradient[f] = gradient
23:                    end if
24:                end if
25:            end for
26:        end for
27:        if ∃ Abdeckung[f][M] ≠ R_f, ∀f ∈ F then
28:            Schlechteste_Abdeckung[f] = Maximale_Fitness    ▷ Zuweisung der schlechtesten Fitness
29:        end if
30:    end for
31:    Maximale_Inhomogenitaet[f] = Groesster_Gradient[f] - Kleinster_Gradient[f], ∀f ∈ F
32:    Fitness[F] = Schlechteste_Abdeckung[F]
33:    Fitness[2 · F] = Maximale_Inhomogenitaet[F]
34:    return Fitness
35: end function
```

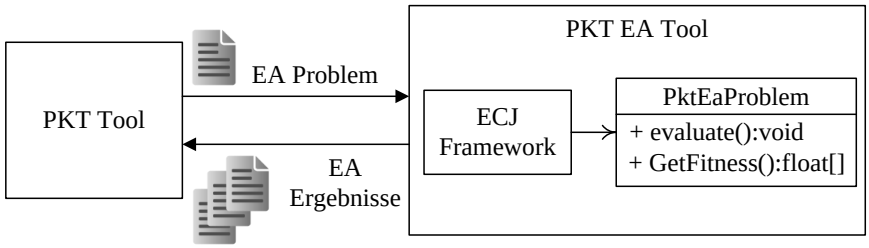


Abbildung 5.15.: Anbindung des EA an das PKT Tool

5.4.3. Abhängige Testzeit bei gemeinsamen Ressourcen

Die bisherige Testverteilung ging von einem statischen Design nach dem P&R und statischen Anforderungen aus. Obwohl dies dem normalen Entwicklungsfluss entspricht, können Änderungen in der Platzierung oder in den Anforderungen an die FDZ die Testzeiten deutlich verbessern. Dieser Varianzpfad ist bereits in Abbildung 5.12 dargestellt. Die Veränderung der Platzierung oder der FDZ einer FPGA Funktion beeinflusst das Ergebnis des PKT anderer Funktionen.

5.4.3.1. Varianz der Platzierung

Bei einer Änderung in der Platzierung von Funktionen im FPGA kommt es zu einer Änderung der Überschneidungen der Testregionen R_F und der priorisierten Testregionen R'_F. Für diesen Vorgang werden keine Veränderungen der Größe von R_F angenommen. Ferner werden für die folgenden Untersuchungen immer nur zwei Testregionen gegeneinander verschoben. Zur Erläuterung der Vorgänge dient erneut das Beispiel aus Abbildung 5.6 mit $FDZ_1 < FDZ_2$.

Der Test einer Funktion wird durch den zusätzlichen Testaufwand für weitere Funktionen verlängert. Die Häufigkeit der Unterbrechung ist durch das Verhältnis beider FDZ bestimmt. Somit ergibt sich für die Testdauer einer Funktion $t_{f,test}$ die Summenformel aus Gleichung 5.14. Die Summe ist abhängig von R'_j und somit unmittelbar von den Überschneidungen der Testregionen.

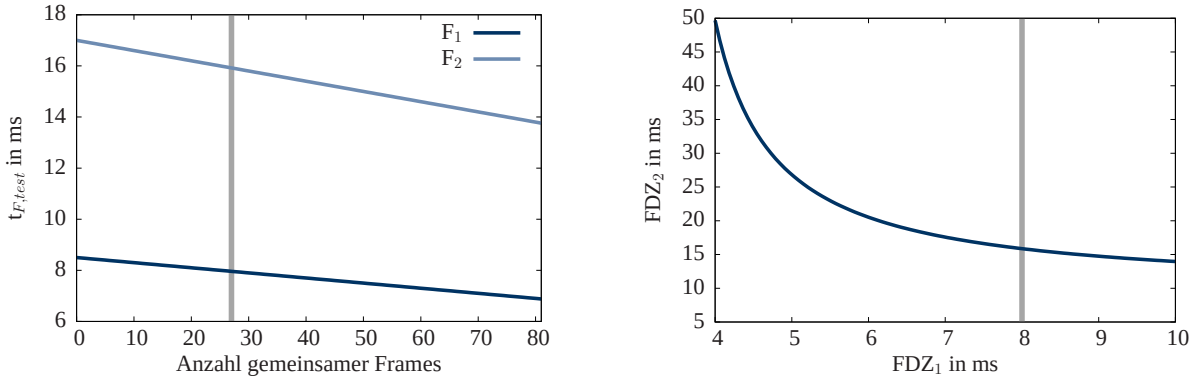
$$t_{1,test} = N_{R'_1} \cdot t_S + \left\lceil \frac{FDZ_1}{FDZ_2} \cdot N_{R'_2} \right\rceil \cdot t_S$$
$$t_{2,test} = N_{R'_2} \cdot t_S + \left\lceil \frac{FDZ_2}{FDZ_1} \cdot N_{R'_1} \right\rceil \cdot t_S$$

$$t_{f,test} = t_S \cdot \sum_{j=1}^{F} \left\lceil \frac{FDZ_f}{FDZ_j} \cdot N_{R'_j} \right\rceil \tag{5.14}$$

$t_{f,test}$	=	Mittlere Testzeit einer Funktion F bei homogener Testverteilung
t_S	=	Mittlere Testzeit eines Testslots mit einem Major Frame (MF). Hier: 40 μs
N_R	=	Anzahl der MFs in der Region R

Zur Illustration wurde eine weitere Beispielanwendung evaluiert. Der gegenseitige Einfluss der Anzahl der gemeinsam genutzten Frames auf die Testzeit ist im Graph der Abbildung 5.16a dargestellt. Bei Vergrößerung der gemeinsamen Fläche reduziert sich die Testzeit beider dargestellten Funktionen. Der Anstieg dieser

Funktion wird durch die Verhältnisse der FDZ bestimmt. Die senkrechte Linie bei 27 Frames gibt den aktuellen Platzierungszustand wieder. Diese Kurven sind ideale Testzeiten bei homogener Verteilung. Durch Inhomogenitäten in der Testsequenz werden die schlechtesten Zeiten größer sein.



(a) Variation der gemeinsam genutzten Frames

(b) Variation der FDZ Anforderung

Abbildung 5.16.: Veränderung der Testzeiten bei Varianzen der Anzahl gemeinsam genutzter Frames und FDZ Anforderungen. Die senkrechte Markierung zeigt die aktuell gesetzten Anforderungen.

5.4.3.2. Varianz der Anforderungen

Eine Änderung in den Anforderungen an die FDZ ist in der Praxis zwar selten, aber denkbar. Die Testzeit einer Funktion wird bei verschärfter Anforderung an eine andere Funktion länger. Dies ist bei unabhängig platzierten Funktionen mit wenigen gemeinsamen MFs der Fall. Aus der Gleichung 5.14 für $t_{f,test}$ ist die Gleichung 5.15 abgeleitet und zeigt die Abhängigkeit der FDZ einer Funktion von den FDZ weiterer Funktionen. Hier wird angenommen, dass die Platzierung konstant bleibt und die FDZ variiert werden.

$$\frac{R'_1}{FDZ_i} = \frac{1}{t_{test,MF}} - \sum_{k=1,k\neq i}^{F} \frac{R'_k}{FDZ_k} \tag{5.15}$$

Ein Beispiel für zwei Funktionen in einem FPGA zeigt Abbildung 5.16b. Hier wird FDZ_1 variiert und verändert somit die ideale FDZ_2.

5.5. Realisierung des Testsequenzers im FPGA

Die Anwendung der erstellten Testsequenz zur Laufzeit erfordert eine Implementierung des PKT im FPGA. Einen Überblick über dieses IP Modul verschafft Abbildung 5.17 und enthält einige Elemente aus Heiner et al. (2008) [HCW08] und Dutton & Stroud (2009) [DS09b]. Der lesende und schreibende Zugriff auf das CRAM erfolgt über den ICAP. Die interne Zustandsmaschine des ICAP wird durch die `Detektions- und Korrektureinheit (DKE)` bedient. Aufgrund der internen Architektur aktueller Xilinx FPGAs ermittelt das Hardmacro `FrameECC` für jeden ausgelesenen Frame automatisch die Syndrome Bits und vergleicht diese mit den gespeicherten Paritätsbits des CRAMs. Die Zustandsinformationen der ECC Berechnung werden von der DKE übernommen und ausgewertet, beispielsweise die korrekte Berechnung der Syndrome (`SYNDROMEVALID`),

das Fehlerflag (`ECCERROR`), die Frameadresse (`FAR`) und die Bitposition des Fehlers (`SYNDROME`). Ferner speichert die DKE den letzten gelesenen Minor Frame in dem Speicher `Frame-Puffer`, welcher für eine eventuelle Fehlerkorrektur benötigt wird. Die Testsequenz ist in einem separaten Speicher abgelegt und stellt die Frameadressen zum Auslesen bereit.

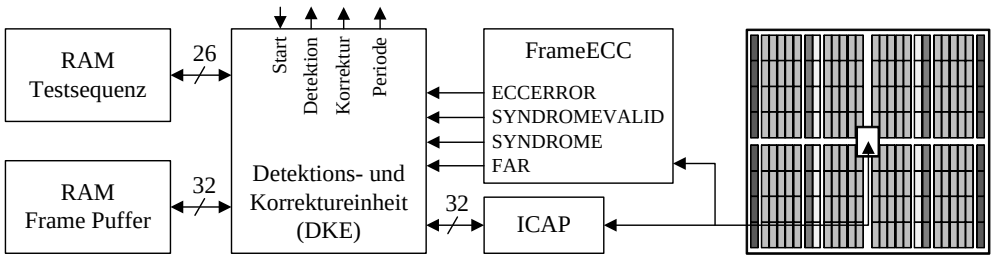


Abbildung 5.17.: Blockschaltbild der PKT Implementierung im FPGA

5.5.1. Detektions- und Korrektureinheit

Für eine kostengünstige Lösung wurde die DKE als dedizierte Hardwarelösung ohne Prozessoranteil implementiert. Die Grundaufgaben lauten

1. Ansteuerung des ICAP zum Lesen und Schreiben von CRAM Frames
2. Bestimmung der Frameadressen aus der Testsequenz und aus den Frametypen
3. Auswertung von Paritätsfehlern und Korrektur von Einzelbitfehlern

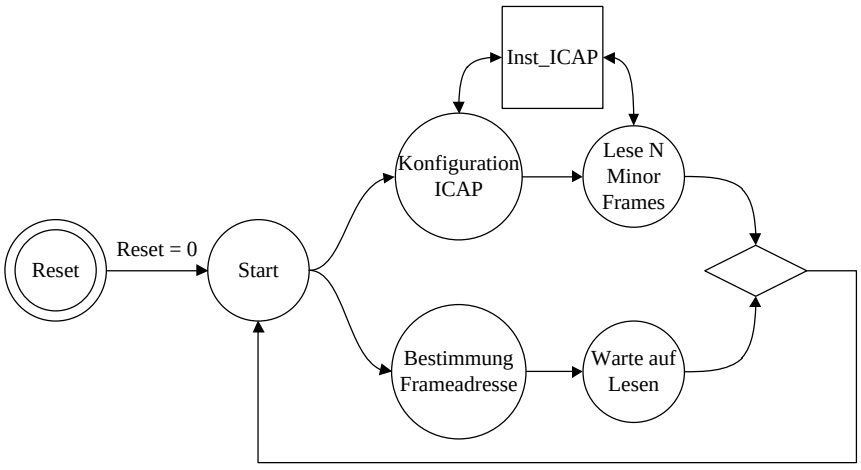


Abbildung 5.18.: Vereinfachter Ablauf der Fehlerdetektion DKE

Der vereinfachte Ablauf für eine Fehlerdetektion ist im Zustandsgraphen der Abbildung 5.18 zu erkennen. Nach dem `Start` werden die Ermittlung der neuen Frameadresse und das Auslesen der alten Frameadresse nebenläufig betrieben. Die Konfiguration des ICAP und das Auslesen von N Frames ist als VHDL Snippet in Anhang A.4.4 abgebildet. Zum Auslesen eines Frames muss zusätzlich ein Dummy Frame ausgelesen werden, da die interne ICAP Pipeline voll belegt werden muss. Nach dem Abspeichern dieser Framedaten kann der nächste Frame gelesen werden. Die neue Frameadresse wird entweder aus der Testsequenz entnommen oder die alte Adresse wird erneut mit anderem Block Typ und veränderter Anzahl der Minor Frames gesetzt. Diese

Maßnahme dient der Erhöhung der Fehlerabdeckung durch Einbindung von Frames ohne Logikbezug aber mit CLB Einstellungen. Die genaue Aufteilung der Frameadresse ist in Anhang A.4.3 dargestellt. Nach einer kompletten Periode der Testsequenz wird diese von Speicheradresse Null neu gestartet.

Nach der Detektion eines Fehlers über das Fehlerflag `ECCERROR` wird der Inhalt des Frame-Puffers korrigiert und zurück in den CRAM geschrieben. Das Rückschreiben erfolgt durch erneutes Einfügen eines Dummy Frames gemäß den Erläuterungen aus Anhang A.4.5 innerhalb von etwa 2 μs. Für sicherheitsrelevante Anwendungen kann die gesamte DKE mit einem Selbsttest getestet werden. Die FPGA-internen ICAP und FrameECC Hardmacros können ebenfalls über Selbsttest Erweiterungen nach Dutton & Stroud (2009) [DS09a] geprüft werden oder mittels Toleranzmethoden nach Heiner et al. (2008) [HCW08] gehärtet werden.

5.5.2. Speicherbedarf

Der eigentliche Unterschied zu klassischen LKT basierenden Tests ist die Ablage der Testsequenz im Speicher. Die Speicherstrategie ist in Abbildung 5.19 dargestellt. Zum einen werden die physikalischen MF Adressen in einem Speicherbereich inklusive der Anzahl zugehöriger Minor Frames abgelegt. Ferner wird der Typ-Wechsel (TW) gespeichert und zeigt ein Auslesen dieser Adresse jeweils mit Block Typ 0 und 2 an. In diesem Fall werden beide Blocktypen nacheinander gelesen, ohne dass der Sequenzzähler inkrementiert wird. Für eine Zuordnung der Fehlerquelle auf die Fehlerreaktion des Systems werden die beteiligten Funktionen dieses Frames gespeichert. Hiermit können für jeden Frame nach einem Fehler dedizierte Schaltungsfunktionen in den sicheren Zustand geführt werden.

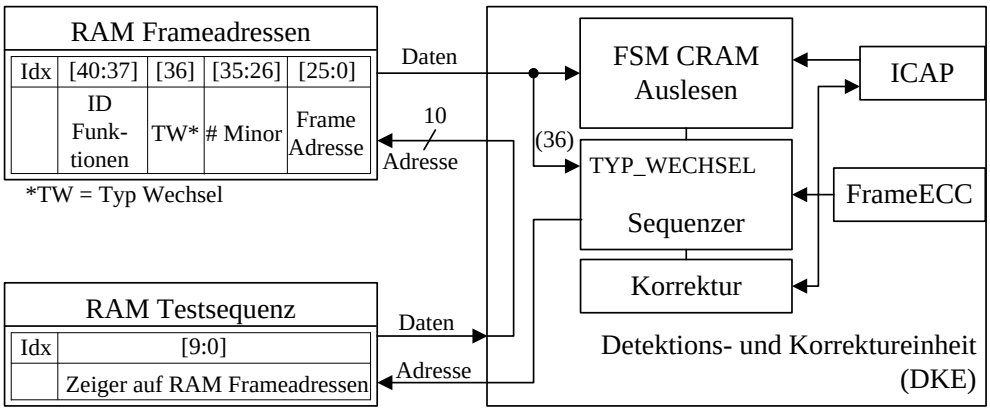


Abbildung 5.19.: Speicherverteilung der Testsequenz

Zur Optimierung des Speicherbedarfs wird die Testsequenz als Zeiger auf die physikalischen Frameadressen abgelegt. Somit dient der Datenausgang der `RAM-Sequenz` gleichzeitig als Adressbus für den Block `RAM-Frameadressen`. Ferner reduziert der PKT Controller die Frameadressen um den Minor Frame Anteil. Diese Optimierung ist in Abbildung 5.19 nicht dargestellt und sie gibt 8 Bit pro Eintrag in der physikalischen Adressliste frei. Somit ergibt sich Gleichung 5.16 für den Speicherbedarf innerhalb der PKT Implementierung.

5.5.3. Empirischer Nachweis der PKT Fehlerabdeckung

Nach dem theoretischen Nachweis der Fehlerabdeckung des PKTs von über 96 % aus Abschnitt 5.2.2 wurde eine Fehlerinjektion in den CRAM durchgeführt. Hierfür wurde mittels PDR zur Laufzeit ein Frame ausgelesen,

$$\text{Mem} \approx N_{EF} \cdot (\overbrace{17}^{\text{Frameadresse kurz}} + \overbrace{\lceil log_2(F) \rceil}^{\text{MF} \rightarrow \text{F}} + \overbrace{8}^{\text{\# Minor Frames}} + \overbrace{1}^{\text{TW}} + \overbrace{1}^{\text{Parität}}) + \underbrace{(1 + \lceil log_2(N_{EF}) \rceil) \cdot L_{seq}}_{\text{Testsequenz mit Parität}} \tag{5.16}$$

N_{EF} = Anzahl essentieller Frames der Schaltung
F = Anzahl der gesetzten Funktionen
L_{seq} = Anzahl der Einträge in der Testsequenz

verändert und wieder geschrieben. Diese Art der SEU Emulation wurde bereits erfolgreich von Dutton et al. (2009) [DASS09] und Gosheblagh & Mohammadi (2013) [GM13] angewendet. Durch eine zufällige Auswahl des Ortes der Fehlerinjektion ist die Wahrscheinlichkeit der Korrumpierung einer Speicherzelle über den gesamten CRAM gleich verteilt. Gemäß dem Abschnitt 2.3.1 über flüchtige Fehler im CRAM liegt die Fehlerwahrscheinlichkeit für Einzelbit- oder Zweibitfehler in einem Frame bei etwa 98 %. Für die eingesetzte SECDED Maßnahme ist somit eine Einzelfehler-Injektion für den Nachweis der Fehlerabdeckung unter Berücksichtigung der 2 % MBU Testlücke in der Auswertung ausreichend.

Der Versuchsaufbau ist in Abbildung 5.20 dargestellt. Das Testsystem besteht aus zwei FPGAs, einem Master (Xilinx Virtex 7 XC7CX485T) und einem Slave (Xilinx Kintex 7 XC7K325T). Der Master enthält den Mehrheitsentscheider (Voter) zur Überwachung eines Logikfehlers in der Schaltung nach der Fehlerinjektion. Die Testschaltung (Unit Under Test (UUT)) ist dreifach über TMR ausgelegt, um die Fehlerquelle eindeutig zu identifizieren. Davon sind zwei Instanzen im Master als Kontrollinstanzen implementiert, die dritte Instanz (`TMR_3`) wird der Fehlerinjektion im Slave ausgesetzt. Sowohl im Master als auch im Slave wird die UUT von einem gleich getakteten Test Pattern Generator (TPG) mit Input Signalen versorgt. Im Slave sind neben der Testschaltung noch das PKT IP mit der Detektions- und Korrektureinheit (DKE) vorhanden. Ferner wird für die Fehlerinjektion ein echter Zufallszahlengenerator nach Schellekens et al. (2006) [SPV06] implementiert. Dieser sorgt nach einer Neukonfiguration für eine zufällige Auswahl des Frames und des dazugehörigen Bits für die Manipulation. An den Slave ist extern ein Flash Speicher mit dem Bitstrom angeschlossen.

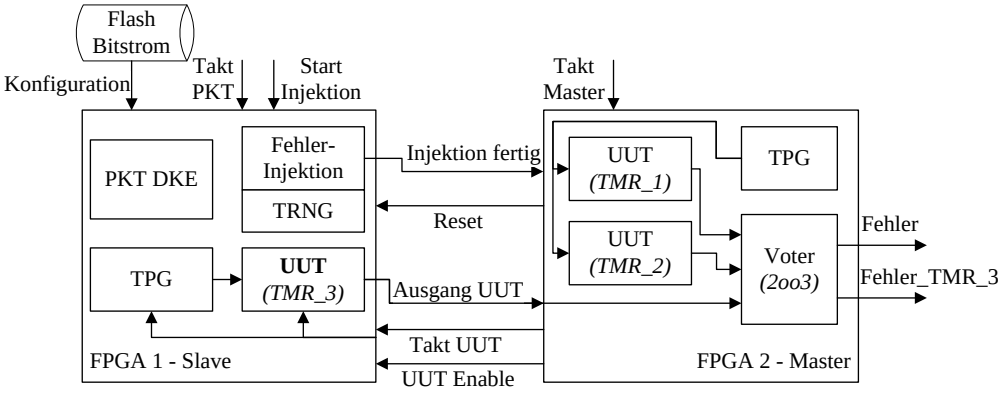


Abbildung 5.20.: Blockschaltbild zur Fehlerinjektion und Fehlerdetektion der Fehlerauswirkung inklusive PKT

Zur Erläuterung der Fehlerinjektion dient der Ablaufgraph aus Abbildung 5.21. Nach der Konfiguration des FPGAs aus dem Flash Speicher wird die Fehlerinjektion gestartet. Mittels True Random Number Generator (TRNG) wird der Ort der Bit-Manipulation festgelegt. Anschließend wird der entsprechende Frame ausgelesen, ein Bit wird gekippt und der Frame wird zurück in den CRAM geschrieben. Diese Vorgänge werden im Slave durchgeführt und danach wird der Master mittels `Injektion fertig` Signal über eine erfolgreiche

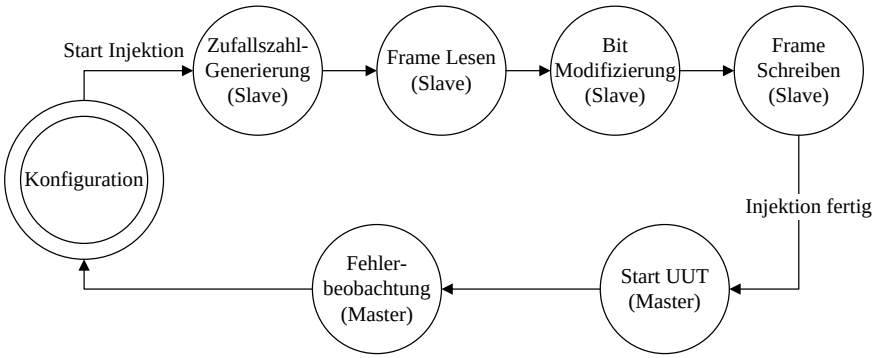


Abbildung 5.21.: Ablauf der Fehlerinjektion und Verifikation der Testschaltung

Tabelle 5.1.: Fehlerabdeckung des PKT CRAM Test nach Fehlerinjektion

	# Injizierte Fehler	# Schaltungsfehler `TMR_3`	# Detektierter Fehler durch PKT nach Schaltungsfehler	Fehlerabdeckung $Q_{F,Det,max}$
Schaltung 1 [Fed14]	7312	159	158	99,4 %
Schaltung 2 [Ele02]	8613	209	206	98,6 %

Fehlerinjektion informiert. Nun startet der Master den Takt und das Enable der UUT. Innerhalb des Masters werden die zwei Referenzausgänge der UUT und der potentiell korrumpierte Ausgang des Slave mittels Voter für etwa 10 Sekunden beobachtet. Ein Fehler im TMR Modul 3 wird an dem Signal `Fehler_TMR_3` angezeigt. Für eine erneute Fehlerinjektion wird der Master und der Slave erneut konfiguriert, um für die Instanzen der Fehlerinjektion, `TPG` und `UUT` gleiche Startbedingungen herzustellen.

Mit diesem Versuchsaufbau sind zwei Testschaltungen evaluiert worden. Zum einen die Schaltung von Fedorov (2014) [Fed14] und zum anderen die Schaltung B18 des ITC99 Benchmarks [Ele02]. In Tabelle 5.1 sind die quantitativen Ergebnisse der Fehlerinjektion aufgelistet. Für die Detektion mittels PKT sind die Fehler in die Fehlerabdeckung eingerechnet, die auch zu einem Ausfall der Schaltung führten. Es zeigt sich, dass durch Fehlermaskierungen und Teilauslastungen des FPGAs nur zwei bis drei Prozent der injizierten Fehler zu einem Schaltungsausfall führten. Davon erkannte der PKT im schlechtesten Fall 98,6 % Einbit- und Zweibitfehler. Diese Fehlerabdeckung entspricht den erwarteten theoretischen Zahlen und zeigt eine hohe Zuverlässigkeit des PKTs für eine Fehlerdetektion im Konfigurationsspeicher.

5.6. Anwendungsergebnisse für verteilte und einzelne FPGA Funktionen

In den vorherigen Abschnitten wurden das Prinzip und das Vorgehen zur Erstellung eines PKTs erläutert. Nun werden anhand von drei Schaltungen die Vorteile und Beschränkungen dieser Methodik gezeigt. Zum einen wird eine FPGA Schaltung mit zwei unabhängigen Microcontrollern evaluiert. Diese Schaltung kann auch in einem ASIL A System nach ISO 26262 eingesetzt werden. Zum zweiten wird die entwickelte Methode auf einen nach ASIL B konzipierten redundanten Microcontroller nach dem DCR Prinzip angewendet. Abschließend werden die Grenzen des PKTs anhand einer komplexen industriellen Timerschaltung aufgezeigt.

Die Anwendung des PKTs und die Auswertung der Schaltungen laufen in mehreren Schritten ab.

1. Zu Beginn wird das Layout der Schaltung auf dem FPGA ermittelt und dargestellt. Das Layout bezeichnet hier die Lage der verwendeten PIPs und Instanzen im CRAM in Form von logischen `x-y` Koordinaten. Mit Hilfe dieser Koordinaten lassen sich die Konfigurationselemente auf Major Frames MF projizieren und adressieren.

2. Anschließend werden für Funktionen mit gesetzter FDZ die einhüllenden Testregionen bestimmt und dargestellt. Hierbei können auch Isolationen und Überschneidungen von Testbereichen mit Hinblick auf eine unabhängige Platzierung im FPGA überprüft werden, analog zu dem Tool von Corbett (2013) [Cor13].

3. Nun folgt die Erstellung von Testsequenzen mittels Scheduling nach der zeitgesteuerten Testverteilung aus Abschnitt 5.3. Hierbei ist die Anzahl und die Länge der Überschreitungen von Deadlines die Maßeinheit für einen schnellen Konfigurationstest.

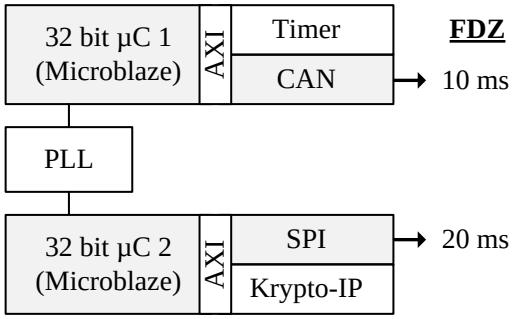
4. Im Falle eines nichtdeterministischen Datenflusses werden die Ergebnisse für die homogene Testverteilung bestimmt. Hier wird die schlechteste FDZ für einen undefinierten Fehlerzeitpunkt über die Laufzeit ermittelt. Für den PKT werden hierfür zwei aufeinanderfolgende Perioden der Testsequenz evaluiert.

5. Die Trimmung der realisierten FDZ kann abschließend über die Reduzierung der Fehlerabdeckung einer Testregion erfolgen. Hierfür werden MFs nach den Methodiken aus Abschnitt 5.2.2.2 in eine Testregion mit einer FDZ von 50 ms überführt. Durch diese Maßnahme wird die funktionale Fehlerabdeckung $Q_{F,Det}$ in einem Maße von 0,1 % bis 10 % gesenkt. Eine Trimmung der Ergebnisse über eine veränderte Platzierung der Funktion ist in praktischen Anwendungen kaum sinnvoll, da die Platzierung durch übergeordnete Anforderungen der Anwendung vorgegeben wird. Diese Anforderungen sind vor allem Signallaufzeiten (Timing), aber auch die Nutzung dedizierter Primitiven innerhalb des FPGA, der Einsatz partieller dynamischer Rekonfiguration oder die unabhängige Platzierung und Verdrahtung aus Gründen der funktionalen Sicherheit.

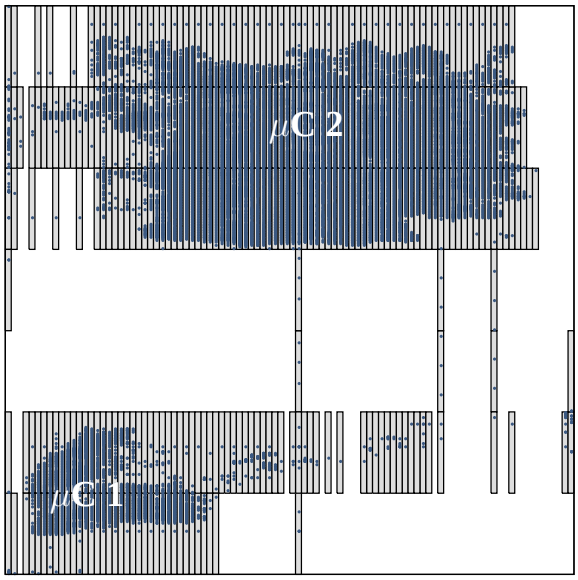
Die ermittelten Ergebnisse aus zeitgesteuerter und homogener Testverteilung werden mit den Methoden aus Tabelle 2.6 in Abschnitt 2.3.3 verglichen. Für eine bessere Übersichtlichkeit werden hier die zusammenfassenden Methodenbezeichnungen LVKT und LEFK verwendet.

5.6.1. Dual Microcontroller Schaltung

Eine typische Anwendung in einem FPGA ist eine Architektur mit dedizierten Controllern für unabhängige Aufgaben. Die Schaltung in Abbildung 5.22a entspricht dieser ASIL-A konformen Architektur. Die beiden Microcontroller werden von einer gemeinsam verwendeten Phase-Locked Loop (PLL) mit unterschiedlichen Taktleitungen versorgt. Microcontroller 1 ist hier für die externe Kommunikation mittels Controller Area Network (CAN) verantwortlich und Controller 2 steuert die Leiterplatten-Kommunikation mittels SPI und hat Aufgaben in der Cybersicherheit über Encryption- und Decryption-Algorithmen. Beide Controller arbeiten unabhängig voneinander. Der weitere Funktionsinhalt der Schaltung ist für den priorisierten Konfigurationstest nicht relevant. Es wird lediglich das Timing des Datenflusses benötigt.



(a) Blockschaltbild mit gesetzter FDZ für zwei Schaltungsausgänge



(b) Vereinfachtes Layout mit Konfigurationselementen und MFs

Abbildung 5.22.: Dual Controller Anwendung im FPGA

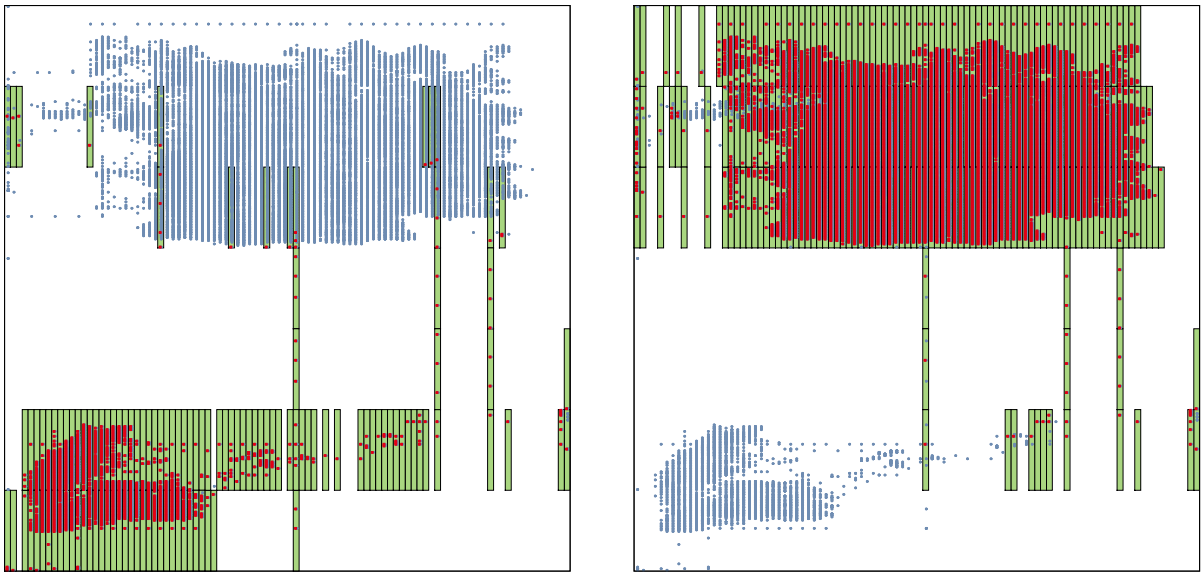
Tabelle 5.2.: Eigenschaften der Testregionen

	R_F	R'_F
F1	126	126
F2	256	223

Die gegebene Schaltung besteht aus 270.629 HDL Signalen. Diese Signale wurden nach dem Vorgehen aus Abschnitt 5.2.1 gefiltert und für den Anwender als Referenz zur Verfügung gestellt. Ferner besteht die hierarchische Netzliste aus 41.293 Netzen. Nach dem P&R verbleiben noch 24.853 Netze in der Schaltung und die Schaltung ist auf 367 MFs mit dem Frametyp 0 verteilt. Das Layout mit den verwendeten Konfigurationselementen in dunklen Punkten und den implizit verwendeten MFs in hellen Rechtecken ist in Abbildung 5.22b dargestellt. In einem LEFK-basierten Konfigurationstest werden demnach 367 Frames linear auf Fehler geprüft.

Aus dem vollständigen Layout und der Schaltungsbeschreibung werden nun über die Anforderungen an die FDZ die einhüllenden Testregionen für die Funktion 1 und 2 bestimmt. Die Testregionen erstrecken sich über die einzelnen μC inklusive aller Peripherie. Die AXI-Busanbindung sorgt für diesen großen Abhängigkeitsbaum in der Netzliste innerhalb des gesamten Controllers. Abbildung 5.23a zeigt die Testregion des CAN-IP Ausganges mit 126 Frames anhand einer Layout-Karte. Die markierten MFs entsprechen der Testregion R_1. In Abbildung 5.23b ist die Testregion R_2 mit 256 MF für den SPI Ausgang dargestellt. Die platzierten Funktionen überlappen sich in 33 MF und sind somit sehr gut voneinander isoliert. Lediglich einige Verbindungsleitungen zu den I/O Treiber verlaufen hier architekturbedingt in der jeweils anderen Testregion.

Für die Testverteilung ergeben sich nun die Größen der Testregionen R'_F. In Tabelle 5.2 werden die Testregionen durch die Anzahl der MFs charakterisiert. Die Testregionen werden nun durch die zeitgesteuerte und homogene Testverteilung in eine Testsequenz prozessiert. Dabei werden die priorisierten Testregionen R'_1 und R'_2 mit den entsprechenden Größen verteilt.

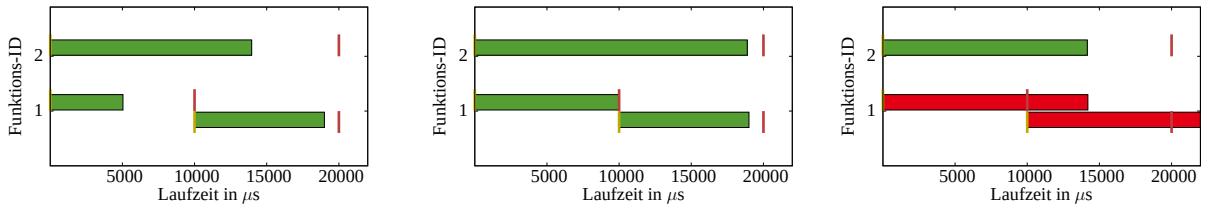


(a) Testregion R_1 für Funktion 1 (CAN) **(b)** Testregion R_2 für Funktion 2 (SPI)

Abbildung 5.23.: Testregionen im Layout mit allen (blaue, helle Punkte) und funktional essentiellen (rote, dunkle Punkte) Konfigurationselementen inklusive essentieller MF (grüne, dunkle Rechtecke)

5.6.1.1. Ergebnis der zeitgesteuerten Testverteilung

Für eine zeitgesteuerte Testverteilung wird das Zeitverhalten des Datenflusses verwendet. Für die gegebene Anwendung wird angenommen, dass die Zeitbasis eine Periode von $P_1 = 10$ ms und $P_2 = 20$ ms hat. Somit fällt auf jede Deadline D_i eine neue Zeitbasis B_{i+1}. Dieses Verhalten lässt sich in typischen periodischen Tasks einer Automobilanwendung wiederfinden. Für den PKT bedeutet diese Verteilung der Zeitbasen und Deadlines eine Hyperperiode des Datenflusses von 20 ms und somit eine Länge der Testsequenz von 500 Einträgen bei 40 μs Testzeit pro Eintrag. Nach Gleichung 5.16 entspricht das einem Speicherbedarf von etwa 15,4 kbit.



(a) EDF-KT Scheduler **(b)** SAT Solver **(c)** LEFK-basierend

Abbildung 5.24.: Scheduling der Testregionen als Gantt-Diagramm

Die Einteilung der MFs auf die Testeinheit wird durch den EDF-KT und zum Vergleich auch mit dem SAT Solver ermittelt. Die Ergebnisse dieses Scheduling sind in Abbildung 5.24a und 5.24b dargestellt. Die Funktionen haben eine durch senkrechte rote Linien dargestellte Deadline von 10 ms und 20 ms. Die Zeitbasen mit einer Periode von 10 ms und 20 ms sind durch die gelben senkrechten Linien gekennzeichnet. Das Gantt-Diagramm zeigt die Testdauer für eine Testregion, wobei die Testdauer jeweils mit der Zeitbasis beginnt und mit dem letzten MF der erforderlichen Region endet. Beide PKT Lösungen halten die Vorgabe der FDZ ein. Die FDZ für diesen Datenfluss liegt bei der EDF-KT Lösung im schlechtesten Fall bei 9,00 ms für Funktion 1 und 13,96 ms für Funktion 2. Die Lösung des SAT-Solvers zeigt zwar eine langsamere FDZ, ist jedoch gemäß der Aussagenlogik innerhalb der Vorgaben und somit eine gültige Lösung. Dem gegenüber ist

das Scheduling eines LEFK in Abbildung 5.24c dargestellt. Hier überschreitet der Konfigurationstest für Funktion 1 beide gesetzten Deadlines um etwa 3,5 ms.

Die vorgestellten Lösungen werden zur Entwicklungszeit mit einem Rechner mit zwei Intel Xeon E5620 Prozessoren (je vier Prozessorkerne mit 2,4 GHz) sowie 96 GiB RAM unter Ubuntu 12.04.4 LTS (64-Bit) ermittelt. Das Scheduling benötigt sowohl für den EDF-KT als auch für den SAT Solver Lösungsweg weniger als eine Minute. Der SAT Solver "Treengeling" von Biere (2013) [Bie13] untersucht in dieser Zeit einen Lösungsraum von $2^{261.522}$ mit 522.049 Klauseln im KNF Format.

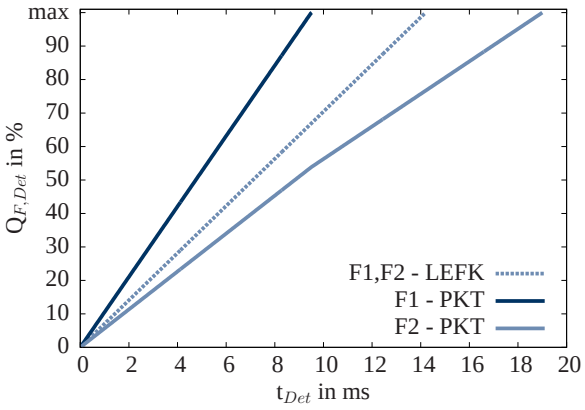
5.6.1.2. Ergebnis der homogenen Testverteilung

Bei der homogenen Testverteilung wird ein Fehlerzeitpunkt nach dem Modell des weißen Rauschens angenommen. Somit kann zu jedem Zeitpunkt ein Fehler entstehen und auch am Schaltungsausgang propagieren. Zur Vereinfachung wird ein Fehler in jedem Testslot angenommen und daraufhin die Testzeit für jeden fehlerhaften Slot bestimmt. Durch eine priorisierte und dennoch gleichmäßige Verteilung der MFs nach dem Vorgehen aus Abschnitt 5.4.1 werden die Zeiten für eine Fehlerdetektion für kritische Funktionen verkleinert. Das Nebenziel nach der Einhaltung der FDZ ist der lineare Anstieg der Fehlerabdeckung mit $\frac{d^2 Q_{F,Det}(t)}{dt^2} \approx 0$. Dieser Anstieg und auch die FDZ bei Erreichung von $Q_{F,Det,max}$ hängt bei mehr als einer implementierten Funktion unmittelbar von dem gewählten Algorithmus für die lineare Klassifizierung und der optionalen EDF-Verteilung ab. In dieser Anwendung wurde für die lineare Klassifizierung das maximale Verhältnis von $\Phi_{M,F}$ zu Φ_F nach Abschnitt 5.4.1.1 bestimmt.

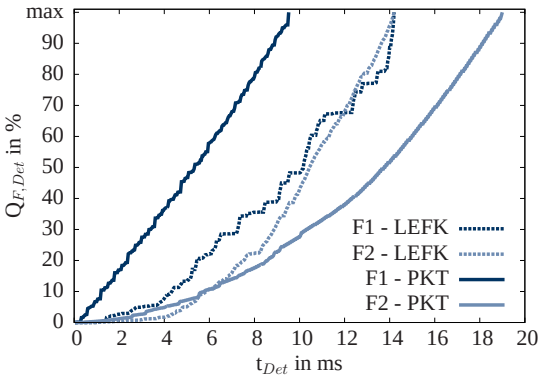
Eine Gesamtübersicht über die Ergebnisse ist in Abbildung A.5 im Anhang A.4.6 dargestellt. Die aggregierten Ergebnisse sind in Abbildung 5.25a und 5.25b dieses Abschnittes illustriert. Es werden jeweils die Verläufe der Fehlerabdeckungen nach einem Fehler $Q_{F,Det}$ über der Laufzeit dargestellt. Mit steigender Laufzeit nach einem Fehler steigt auch die Anzahl der getesteten MFs und somit die Fehlerabdeckung. Der in Abbildung 5.25a gezeigte Mittelwert der Fehlerabdeckung zeigt die homogene Verteilung der Frames über die Testsequenz, sowohl für den LEFK als auch für den hier vorgestellten PKT. In dieser Mittelwertberechnung wird ein Fehler in jedem Testslot angenommen und der Mittelwert von Q aller nachfolgenden Detektionszeiten t_{Det} ermittelt. Die unterschiedlichen Gradienten der Funktion 1 und 2 für einen PKT resultieren aus der verschiedenen Häufigkeit der angehörigen Frames in der Testsequenz.

Für die Anwendung wichtiger als der Mittelwert ist die schlechteste Fehlerabdeckung mit der längsten Detektionszeit t_{Det} nach dem Fehlerzeitpunkt. In Abbildung 5.25b erreichen die Funktionen ihr $Q_{F,Det,max}$ unterhalb der gesetzten FDZ. Diese Zeit t_{Det} bei einer maximalen Fehlerabdeckung entspricht genau der Testzeit $t_{F,test}$ für den PKT. Durch die Priorisierung von Funktion 1 wird diese maximale Fehlerabdeckung nach 9,52 ms erreicht und Funktion 2 wird in jedem Fall nach 19,0 ms vollständig getestet. Der nichtlineare Verlauf für die Funktion 2 wird durch die starke Priorisierung der Funktion 1 mittels EDF-Verteilung hervorgerufen. Die Zeiten für einen LEFK-basierten Test sind jeweils mit 14,2 ms gleich lang. Ferner zeigt der LEFK eine deutliche Nichtlinearität im Verlauf beider Funktionen des schlechtesten Falles, insbesondere für Funktion 2. Das bedeutet eine niedrige Fehlerabdeckung für den Konfigurationsspeicher unmittelbar nach einem Fehler.

In diesem Anwendungsbeispiel werden die Vorgaben für die FDZ mit 10 ms für F1 und 20 ms für F2 erreicht. Falls die Funktionen die Vorgaben nicht erreichen würden, können nun mehrere Parameter der Schaltung



(a) Mittelwert von $Q_{F,Det}$

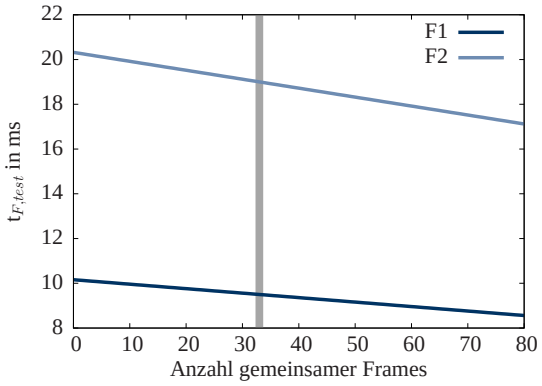


(b) Schlechtester Fall von $Q_{F,Det}$

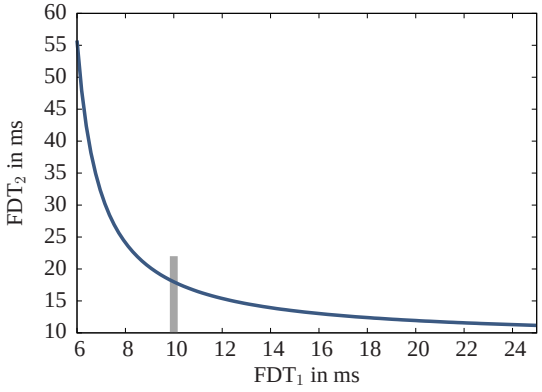
Abbildung 5.25.: Fehlerabdeckung über die Zeit nach einem auftretenden Fehler zum beliebigen Zeitpunkt

verändert werden. Der Entwickler kann das Layout verändern, die Vorgaben der FDZ verlängern oder die Fehlerabdeckung innerhalb der FDZ senken. Die Auswirkungen einer Veränderung der Platzierung und somit der Überschneidungen ist in Graphen der Abbildung 5.26a dargestellt. Die senkrechte graue Linie deutet die Anzahl der gemeinsam genutzten Frames der aktuellen Schaltung an. Durch eine Variation des Layouts können mehr oder weniger Frames von beiden genutzt werden. Hier werden die idealen Abhängigkeiten auf die Fehlerdetektionszeit bestimmt. Die realen Testverteilungen bringen jedoch die bereits erwähnten Inhomogenitäten mit und somit ist der Graph aus Abbildung 5.26a ein Hilfsmittel zur Erreichung neuer FDZ.

Neben dem Layout können in seltenen Fällen auch die Anforderungen an die FDZ verändert werden. Hierfür gibt der Graph aus Abbildung 5.26b einen Hinweis auf die Veränderung der FDZ der konkurrierenden Funktion bei Variation der FDZ der anderen Funktion. In diesem Bild stellt die graue Linie die aktuelle Konstellation dar. Hier hätte eine Reduzierung der FDZ von Funktion 1 um etwa 2 ms eine doppelt so hohe Verlängerung von FDZ_2 zur Folge. Die Variation dieser Werte ist jedoch eine übergeordnete Systemeigenschaft und wird daher hier nur als Option ohne detailliertere Analyse aufgeführt.



(a) Variation der Anzahl gemeinsam genutzter Frames



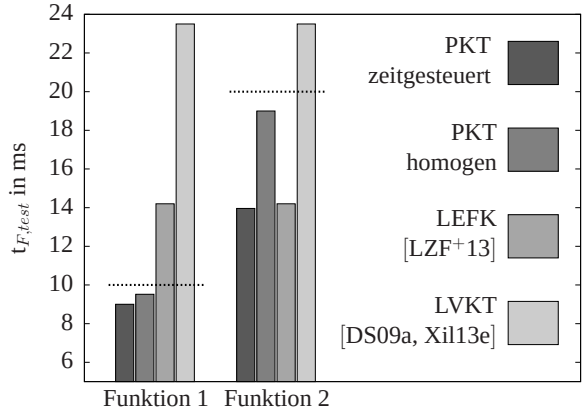
(b) Variation der Anforderungen an die FDZ

Abbildung 5.26.: Ideale Veränderung der Testzeit durch Variation des Layouts und der Anforderungen. Die senkrechte Markierung zeigt die aktuell gesetzten Anforderungen.

Ein Vergleich der FDZ verschiedener Methoden zum Testen des CRAM ist in Abbildung 5.27a gegeben. Für die Funktionen 1 und 2 sind die Testzeiten $t_{F,test}$ für die Methoden PKT, LVKT nach Dutton & Stroud (2009) [DS09b] und nach Xilinx SEM-IP (2013) [Xil13e] und die Methode LEFK nach Lanuzza et al. (2009)

[LZF+09] dargestellt. Die gesetzte Fehlerdetektionszeit ist durch eine schwarze gestrichelte Linie markiert. Der in dieser Arbeit vorgestellte Ansatz des PKTs erreicht die FDZ für beide Funktionen. Die Testzeiten für einen LEFK erfüllen nicht die Anforderungen für Funktion 1 und die Anwendung eines LVKT ist bei diesen Vorgaben für beide Funktionen nicht geeignet.

Die Verringerung der Testzeit wird durch die notwendige Speicherung der Testsequenz erkauft. Das Ergebnis der homogenen Testverteilung ist eine Testsequenz mit der Länge von 475 Einträgen und einem daraus resultierenden Speicherbedarf im RAM von 15,1 kbit. In dem Überblick über die verwendeten Ressourcen der verschiedenen Methoden in Tabelle 5.27b ist ein leicht höherer Speicherbedarf gegenüber weiteren Fehlerdetektionsverfahren erkennbar. Die Anzahl der verwendeten RAM Blöcke des PKTs mit je 18 kbit resultiert aus einem Speicher für die Testsequenz mit 36 kbit, einem Speicher für die physikalischen Frameadressen mit 18 kbit und einem Frame-Puffer für die spätere Korrektur eines Einzelbitfehlers mit 18 kbit. Die Größe des IP in Bezug auf LUTs und FFs für die Fehlerdetektion ist ähnlich der Größe anderer Testverfahren. Eine weitere Methodik zur Fehlerdetektion ist das DCR-Prinzip. Die Einfügung einer DCR Redundanz verringert zwar die Fehlerdetektionszeit auf einen Taktzyklus. Im Gegenzug muss unverhältnismäßig viel Ressourcenaufwand investiert werden und die Erkennung latenter Fehler und eine Fehlerkorrektur ist dennoch nicht möglich.



(a) Vergleich der schlechtesten Fehlerdetektionszeit

	# LUT	# FF	RAM in kbit
PKT	≈ 300	≈ 250	4 · 18
LEFK [LZF+09]	≈ 250	≈ 200	2 · 18
LVKT [DS09a]	≈ 200	≈ 200	1 · 18
LVKT [Xil13e]	≈ 400	≈ 300	3 · 18
DCR [Sut13]	≈ 21100	≈ 6800	28 · 18

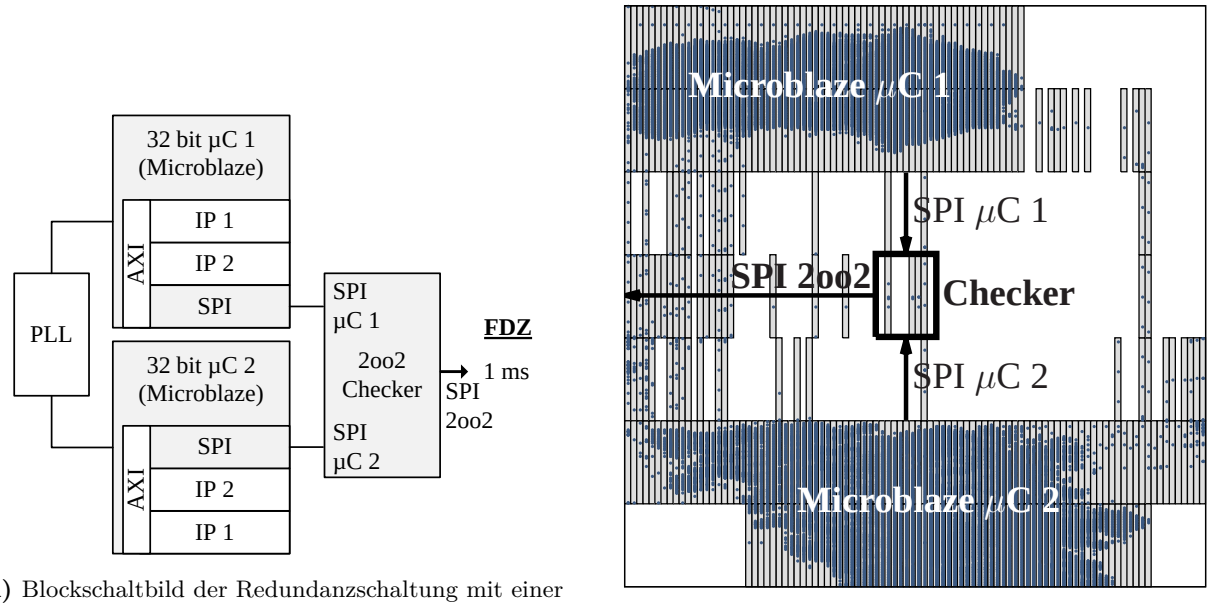
(b) Vergleich der Kosten in Ressourcenaufwand

Abbildung 5.27.: Ergebnisse auf einem Xilinx Kintex 7 XC7K325T

5.6.2. Redundanzschaltung mit Überprüfer

Das Anwendungsbeispiel des Dual Microcontrollers entspricht einer ASIL A konformen Schaltung im Automobilbereich. Eine ASIL B konforme Schaltung benötigt eine interne Redundanz zur Erfüllung der Ausfallraten in aktuell kommerziell verfügbaren FPGAs. Dieses Szenario ist durch die Schaltung in Abbildung 5.28a realisiert. Hier ist ein 32 bit Microcontroller vom Typ Microblaze doppelt implementiert und unabhängig platziert. Der sicherheitsrelevante Ausgang `SPI` ist über einen Überprüfer (Checker) durch eine Zwei-aus-Zwei Entscheidung (`2oo2`) überwacht. Der Checker ist ebenfalls unabhängig platziert und erzeugt neben den Ausgangsdaten noch ein Fehlersignal. Das entsprechende Layout der platzierten FPGA Schaltung ist in Abbildung 5.28b dargestellt. Diese Schaltung verwendet 373 MF, wobei der Checker von dem I/O Block bis zum Eingang beider überprüfender Signale lediglich 6 MFs verwendet.

In dieser Anwendung muss die Korrektheit des Ausgangssignales innerhalb 1 ms nach der Propagation überprüft sein. Hier sind jedoch nicht die redundant ausgelegten Microcontroller die kritischen Module,



(a) Blockschaltbild der Redundanzschaltung mit einer gesetzten FDZ

(b) Layout mit Konfigurationselementen und MFs

Abbildung 5.28.: Redundanzschaltung mit Überprüfer

sondern der Checker. Bei einem Einzelfehler im Microcontroller wird der Checker diesen Fehler detektieren und den sicheren Zustand auslösen. Im Falle eines Fehlers im Checker kann trotz korrekter Signale `SPI µC 1` und `SPI µC 2` eine fehlerhafte Sendung die Schaltung verlassen. Somit muss für den PKT ausschließlich die Netzliste des Checkers innerhalb der FDZ geprüft werden. Der Konfigurationsspeicher der Mikrocontroller wird zur Vermeidung latenter Fehler mit einer längeren Testzeit von 25 ms überprüft.

Tabelle 5.3.: Ergebnis für die schlechteste Testzeit $t_{F,test}$ das Ausgangssignales `SPI 2oo2`

	SPI 2oo2 & Checker	SPI µC 1	SPI µC 2
# MF	6	199	222
Gesetzte FDZ	1 ms	25 ms	25 ms
Zeitgesteuerter PKT	0.3 ms	11.6 ms	18.8 ms
Homogener PKT	1.0 ms	20,2 ms	20,2 ms
LEFK [LZF+09]	14.9 ms	14.9 ms	14.9 ms
LVKT [DS09a, Xil13e]	23,5 ms	23,5 ms	23,5 ms

In Tabelle 5.3 sind die zeitlichen Ergebnisse für den PKT im Vergleich zu weiteren Testverfahren angegeben. Durch die Priorisierung auf den Checker wird eine FDZ von 1 ms und darunter erreicht. Der redundante Microcontroller wird in einer 10 bis 20-fachen Testzeit gegenüber dem Checker überprüft. Die beiden alternativen Verfahren zeigen aufgrund einer fehlenden Korrelation der Schaltung zum Konfigurationsspeicher eine deutlich erhöhte Testzeit für die kritischen Bereiche. In dieser Anwendung enthält die Testsequenz für die zeitgesteuerte Verteilung 625 Einträge und für die homogene Testverteilung 505 Referenzen auf Frameadressen. Somit liegt der Speicherbedarf im Bereich vom vorherigen Beispiel des Dual Controllers.

5.6.3. Echtzeit Timermodul

Nun soll anhand einer Anwendung mit Einzelinstanz die Grenzen des PKTs aufgezeigt werden. Hierfür wird das bereits im Abschnitt der Zustandswiederherstellung vorgestellte komplexe Echtzeit Timermodul in einen

Tabelle 5.4.: Eigenschaften der Testregionen

	F	R_F	R'_F	$\Phi_{MF,F,max}$	$\Phi_{MF,F,min}$	Maximale Knotenebene
`Timer aus`	1	636	636	5190	570	6
`Bus Interface`	2	636	0	5190	570	6

Kintex 7 XC7K325T implementiert. Insgesamt besteht die VHDL Beschreibung aus etwa 235.000 Signalen, die Netzliste aus etwa 227.000 Netzen und das platzierte und verdrahtete Layout aus etwa 180.000 Netzen. Es wird mit 636 MFs beinahe der gesamte FPGA belegt.

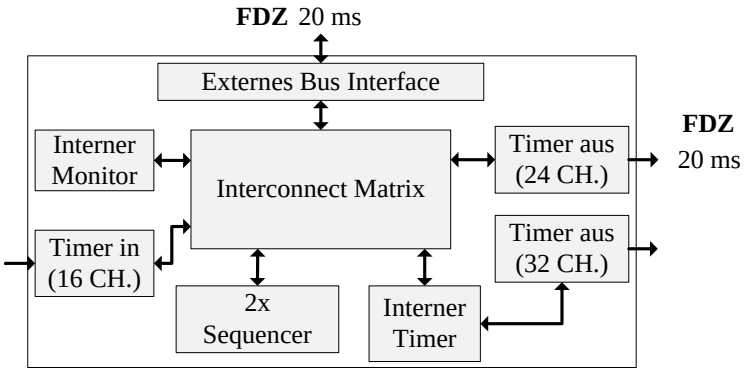


Abbildung 5.29.: Blockschaltbild des Timermoduls mit einer gesetzten FDZ

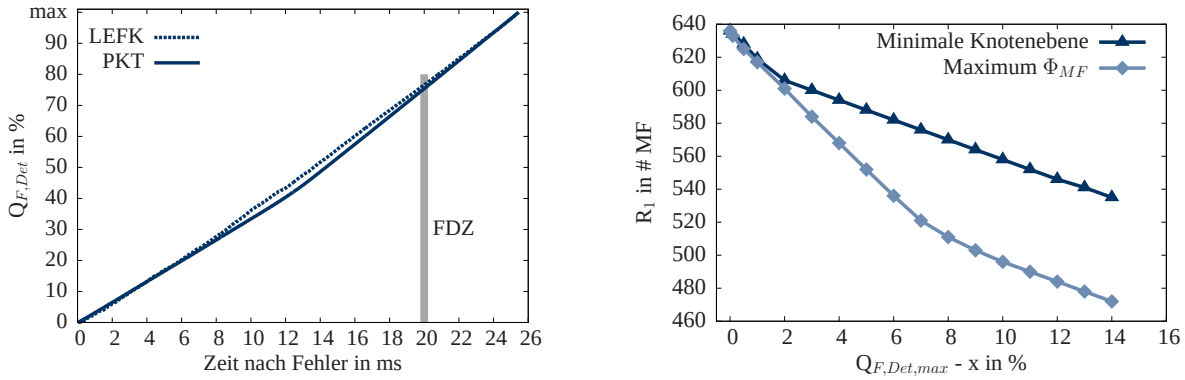
Für einen Konfigurationstest werden nun für den Ausgang von `Timer aus` und für die externe Bus-Schnittstelle eine FDZ von 20 ms für den Konfigurationsspeicher verlangt, siehe Abbildung 5.29. Die einhüllenden Testregionen R_1 und R_2 zeigen nach der Schaltungsanalyse beide eine Größe von 636 MFs und enthalten somit beinahe die komplette Schaltung. Die Ursache hierfür liegt in der Interconnect Matrix, die bei einem Defekt der Netzliste alle angeschlossenen Module korrumpieren kann. Somit wird trotz differenzierter Anforderungen für die FDZ nur eine Testregion R_1 geprüft und es entfällt die zeitgesteuerte Testverteilung aufgrund fehlender Freiheitsgrade für das Scheduling. Tabelle 5.4 zeigt zusammengefasst die Kenndaten der Testregionen mit den minimalen und maximalen Konfigurationsnormierungen der Frames und die maximale Knotenebene eines Frames im Abhängigkeitsbaum der vollständigen einhüllenden Testregion. Die Region R_2 geht komplett in Region R'_1 auf.

Nach der Evaluierung der Testregionen mit einer verbleibenden Einzelfunktion kann nur die homogene Testverteilung angewendet werden und nun mit dem Linearen Essentiellen Frame Konfigurationstest (LEFK) verglichen werden. Der Vergleich der Fehlerabdeckung von einem PKT und LEFK über die Zeit nach einem Fehler ist in Abbildung 5.30a dargestellt. Die Verläufe sind äquivalent und enden jeweils bei der Testzeit $t_{F,test}$ von 25,4 ms. Innerhalb der gesetzten FDZ von 20 ms kann eine maximale Fehlerabdeckung von 76,7 % erreicht werden.

Zum Erreichen der Testzeit von 20 ms führt eine Neuplatzierung der Schaltung nicht zu dem gewünschten Ergebnis, da die Größe der Schaltung eine Reduzierung der Testregion nicht zulässt. Demnach kann die gesetzte FDZ nur durch Reduzierung der maximalen Fehlerabdeckung nach der Methode aus Abschnitt 5.2.2.2 erreicht werden. Bei dieser Methode werden nach zwei Verfahren bestimmte Frames aus der Testregion herausgenommen und einer Testregion mit einer FDZ von 50 ms zugewiesen.

Die detaillierten Kompositionen der Frames von R'$_1$ mit Hinblick auf die Konfigurationsnormierung und die Knotenebenen des Abhängigkeitsbaumes ist im Anhang A.4.7 in Abbildung A.6 zu finden. Die Werte für die Konfigurationsnormierung und die Knotenebene werden bei einer Verringerung der Fehlerabdeckung nach Abschnitt 5.2.2.2 berücksichtigt. In diesem Fall werden die Frames mit dem kleinsten $\Phi_{MF,F}$ aus der Testregion exkludiert und in eine neue Testregion mit einer FDZ von 50 ms eingefügt. Alternativ können die Frames auch auf Grundlage der Knotenebene exkludiert werden, beginnend mit der größten Ebene. Die verbleibenden MFs erreichen nach der Exkludierung von N Frames eine Fehlerabdeckung von $Q_{F,Det,max}$ subtrahiert um die Anteile von $\Phi_{MF,F}$ aller N Frames im Verhältnis zur Gesamtmenge Φ_F. Die exkludierten Frames entsprechen hier einem Anteil von 0 bis 14 Prozent der gesamten Fehlerabdeckung. Die Größe der Testregion verringert sich dagegen überproportional. In Abbildung 5.30b ist die Reduzierung der Größe der Testregion über den exkludierten Anteil `x` dargestellt. Zum einen können die MFs mit der geringsten Konfigurationsnormierung Φ_{MF} bis zum Erreichen des Zielwertes von $Q_{F,Det,max}$ exkludiert werden. Demnach verbleiben in der Testregion die Frames mit maximalem Φ_{MF}. Es zeigt sich, dass innerhalb der geforderten FDZ durch diese Methode viele Frames mit jeweils einem sehr kleinen Anteil an der Gesamtfunktionalität nicht getestet werden. Das erklärt den starken negativen Anstieg der Kurve des maximalen Φ_{MF}.

Eine weitere Möglichkeit zur Reduzierung der Testregion durch Verringerung der Testabdeckung ist die Exkludierung der Frames mit der höchsten Knotenebene im Abhängigkeitsbaum der Netzliste. Die Knotenebenen aus dem Abhängigkeitsbaum geben die sequentielle Hierarchiestufe bezüglich des Ausganges der Schaltung wieder. Je größer die Knotenebene, desto weniger wahrscheinlich ist aufgrund von logischer Maskierung die Fehlerpropagation eines Fehlers in diesem Frame zum Ausgang. Ein Fehler in einem Frame der Knotenebene 1 kann sich unmittelbar kombinatorisch ohne FF Stufe auf den Ausgang auswirken. Abbildung 5.30b zeigt die Veränderung der Testregion bei der Exkludierung der höchsten Knotenebenen. In diesem Fall können auch Frames mit einem großen Wert für Φ_{MF} aufgrund einer hohen Knotenebene ausgefiltert werden. Somit sinkt die Fehlerabdeckung bei geringerer Verkleinerung der Testregion gegenüber der Methode des maximalen Φ_{MF}. Somit zeigt sich eine größere Reduzierung der Testregion bei kleinerem Verlust an Testabdeckung mit der Methode des maximalen Φ_{MF}.

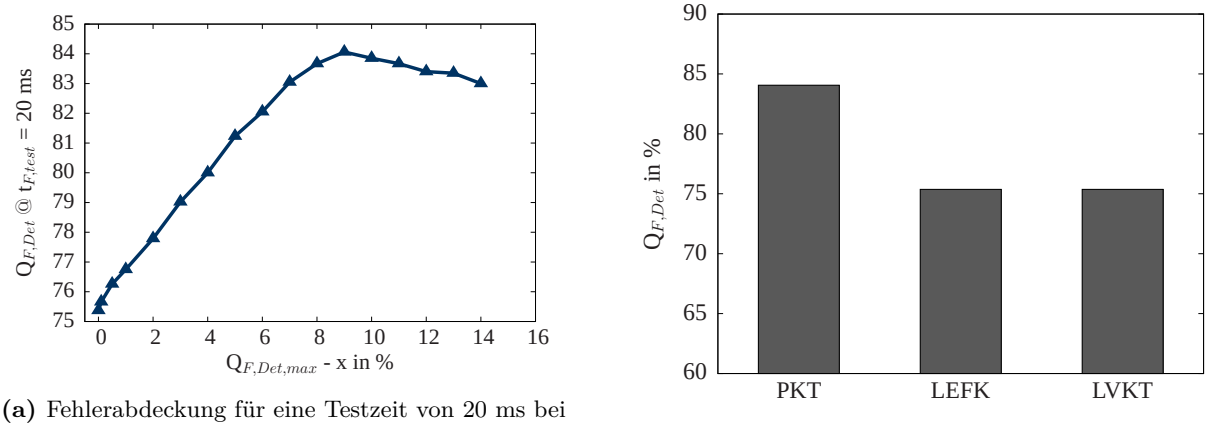


(a) Schlechteste Fehlerabdeckung nach einem Fehler **(b)** Reduzierung der Größe der Testregion

Abbildung 5.30.: Verlauf der Fehlerabdeckung und Verkleinerung der Testregion

Nach der Filterung von Frames mittels der Methode des maximalen Φ_{MF} wird die verbleibende Testregion mit einer FDZ von 20 ms priorisiert. Die exkludierten Frames erhalten eine eigene Testregion mit einer FDZ von 50 ms. Die Größe der Testregion mit exkludierten Frames reicht gemäß der Kurve des Maximalen Φ_{MF} aus Abbildung 5.30b von 0 bis 164 MFs. Diese zwei Testregionen werden nun mit verschiedenen Größen homogen verteilt. Für jede dieser Verteilungen wird nun die Fehlerabdeckung $Q_{F,Det}$ bei einer Testzeit von

20 ms evaluiert. Das entspricht genau der Fehlerabdeckung für die gesetzte FDZ. In Abbildung 5.31a ist die schlechteste Fehlerabdeckung für eine Testzeit von 20 ms nach einem zufällig verteilten Fehler gegenüber der Größe der Testregion mit exkludierten Frames aufgetragen. Wie erwartet erhöht sich bei steigender Größe der zusätzlichen Testregion die Fehlerabdeckung für die ursprüngliche Testregion bei 20 ms Testzeit. Dieser Effekt sättigt sich jedoch bei einer Exkludierung von 9 % der Konfigurationselemente, das entspricht einer Frameanzahl von 133 MFs. Hier wird eine Fehlerabdeckung von 84 % innerhalb der FDZ von 20 ms erreicht. Ab dieser Größe beeinflusst die zusätzliche Testregion die homogene Verteilung der ursprünglichen Testregion stärker als die Reduzierung der Fehlerabdeckung kompensieren kann. Ein detaillierter Verlauf der Fehlerabdeckungen der zwei Testregionen ist in Abbildung A.7 im Anhang A.4.7 zu finden.



(a) Fehlerabdeckung für eine Testzeit von 20 ms bei verkleinerter Testregion nach Maximaler Φ_{MF} Methodik

(b) Vergleich der Fehlerabdeckung nach einer Testzeit von 20 ms

Abbildung 5.31.: Ergebnis der Aufteilung des Timermoduls in zwei Testgruppen

Für die hier vorgestellten Echtzeit-Timer Anwendung mit einer gesetzten FDZ von 20 ms kann keine Fehlerabdeckung von 100 % ($Q_{F,Det,max}$) innerhalb dieser Zeit erreicht werden, sowohl für den PKT als auch für den LEFK und LVKT. Daher wurde eine zusätzliche Testregion mit einer FDZ von 50 ms erzeugt und geringfügig verwendete MFs in diese Region delegiert. Bei einer Exkludierung von 9 % der Konfigurationsbits in diese zusätzliche Testregion kann eine Fehlerabdeckung der gesamten Schaltung von 84 % innerhalb von 20 ms nach Eintritt eines Fehlers erreicht werden. Wie in Abbildung 5.31b dargestellt, ist das eine Steigerung von 8 % gegenüber den bisher angewendeten Methoden. Die in dieser Zeit nicht getesteten Ressourcen haben eine maximale Testzeit von etwa 120 ms.

Es zeigt sich, dass für alleinstehende Funktionen ein Scheduling nicht sinnvoll ist. Eine homogene Testverteilung kommt dann in Betracht, wenn die Anforderung an die FDZ eine Reduzierung der Fehlerabdeckung erforderlich macht. In diesem Fall kann über Auswahlpfade in dem PKT Tool die Fehlerabdeckung für den streng priorisierten Bereich reduziert werden, ohne jedoch Frames komplett aus dem Konfigurationstest zu entfernen. In dieser Anwendung führt eine nominelle Anforderungsreduzierung der Fehlerabdeckung zu einer Erhöhung der tatsächlichen Fehlerabdeckung während des Tests. Bei einer Anforderung an die FDZ des CRAM von größer gleich t_{LKT} nach Gleichung 2.4 aus Abschnitt 2.3.3 bringt der hier vorgestellte PKT keine Vorteile in der Testzeit.

5.7. Zusammenfassung und Erweiterungen

In diesem Kapitel wurde die Methode des Priorisierten Konfigurationstest PKT zur effizienten Fehlerdetektion im Konfigurationsspeicher eines FPGA vorgestellt. Mit dem Ziel einer schnellen FDZ wird die implementierte Schaltung mit dem Speicherlayout des CRAM korreliert. Die dafür notwendige Überprüfung der Netzlisten ist ebenfalls beschrieben worden. Daraus resultieren Testregionen mit Major Frames, die anhand der Vorgaben für die FDZ priorisiert über einen ECC Kern geprüft werden. Momentan erfolgt mit dem ECC Kern eine Fehlererkennung nach dem SECDED Prinzip. Für den ECC Kern wird eine Testsequenz in den Arbeitsspeicher des FPGA Systems abgelegt, die mittels eines ICAP Controllers verarbeitet wird. Hierbei werden einzelne Frames sequentiell aus dem CRAM ausgelesen. Die Systemperformance wird nicht beeinflusst. Ferner kann die PKT Steuerung sowohl für zyklische Rekonfigurationen als auch für eine Verifikation der Netzliste herangezogen werden.

Zur weiteren Effizienzsteigerung wurde das Zeitverhalten des Datenflusses herangezogen, um die Prüfzeitpunkte der Testregionen einzugrenzen. Daraus gehen zwei Verteilungsmethoden für die einzelnen MFs hervor, die zeitgesteuerte Testverteilung und die homogene Testverteilung. Die zeitgesteuerte Testverteilung benötigt die Zeitpunkte der Datenkommunikation von kritischen Ausgangsports aus der Schaltung heraus. Das anschließende Scheduling mittels EDF-KT Scheduler und SAT Solver erstellt die Testsequenz. Für die homogene Testverteilung hingegen werden keine expliziten Zeitpunkte zum Testen vorausgesetzt. Hier werden die Frames der Funktionen über einen heuristischen und einen metaheuristischen Ansatz homogen in der Testsequenz verteilt.

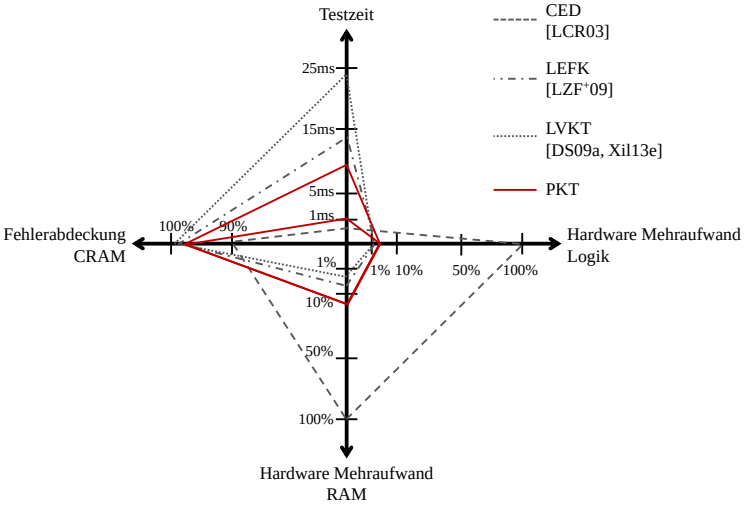


Abbildung 5.32.: Quantitativer Vergleich des PKT mit konventionellen CRAM Fehlerdetektionsmethoden

Die beiden Vorteile einer kostengünstigen Implementierung und einer reduzierten Fehlerdetektionszeit werden anhand von zwei Anwendungen gezeigt und aggregiert in Abbildung 5.32 zusammengefasst. Für den PKT reduziert sich die Testzeit für eine dedizierte Funktion um den Faktor 1,3 bis 10 gegenüber aktuellen CRAM Testverfahren bei ähnlichem Hardwareaufwand. An dieser Stelle wird mit einem Linearen Vollständigen Konfigurationstest (LVKT) [DS09b], [Xil13e], einem Linearen Essentiellen Frame Konfigurationstest (LEFK) [LZF+09] und mit einer Concurrent Error Detection (CED) [LCR03] verglichen. Eine Kombination aus PKT und redundanten Schaltungsmodulen zeigt über dieses Ergebnis hinaus eine sehr gute Kombination aus kurzer Testzeit und Absicherung sicherheitsrelevanter Anwendungen.

In dem Anwendungsbeispiel einer alleinstehenden Funktion im FPGA stoßen die Vorteile des PKT an Grenzen. Hier fehlen die Freiheitsgrade für eine Priorisierung mittels Scheduling und somit erzeugt der PKT keine verbesserte Fehlerdetektionszeit. In dieser Anwendung kann jedoch eine höhere Fehlerabdeckung um 8 % gegenüber aktuellen CRAM Testverfahren innerhalb der FDZ erzielt werden. Des Weiteren wird angenommen, dass zukünftige FPGA basierte Systeme die Komplexität und Vielfalt an parallel laufenden Funktionen mitbringen werden.

Das vorgestellte Verfahren lässt sich durch eine detailliertere Korrelation der Netzliste mit dem Konfigurationsspeicher des FPGAs verfeinern. Somit kann die Größe der kleinsten Testeinheit variiert und das Scheduling erweitert werden. Für alle vorgestellten Anwendungen gilt zudem, dass aufgrund unvollständiger Zuordnung essentieller Bits zum Testbereich nur eine maximale Fehlerabdeckung von 96,7 % bis 99 % erreicht werden kann. Im Sinne der funktionalen Sicherheit könnte hier die Einbindung aller globalen Takttreiber GCLK und aller essentiellen FPGA Einstellungen in globalen Registern des CRAM gezielter erfolgen und die Fehlerabdeckung weiter verbessern. Hierfür werden jedoch detaillierte Informationen über den Aufbau des Speicherarrays seitens der Hersteller benötigt. Ferner wurde die Methode aktuell nur auf einzelnen FPGAs angewendet, bei denen ein framebasierter Zugriff auf den Konfigurationsspeicher durch den Anwender möglich ist. Für eine Verbreitung der Methode sollten weitere FPGA Hersteller diesen Zugang zum CRAM ermöglichen und eine transparente Toolkette mit Schaltungs- und Layoutinformationen bereitstellen.

6. Zusammenfassung und Ausblick

Die vorliegende Arbeit beschäftigt sich mit der Reduzierung der mittleren Fehlerdetektions- und Reparaturzeit (MTTR) durch die kostengünstige und redundanzfreie Anwendung von Fehlerdetektions- und Fehlerkorrekturmaßnahmen in FPGA Schaltungen. Für das Gesamtsystem erhöht sich damit die Verfügbarkeit und auch die Qualität des Steuergerätes. Hierzu wurden jedoch keine FPGA internen Toleranzmechanismen entwickelt, da diese in vielen sicherheitsrelevanten Anwendungen entweder unnötig sind oder durch externe Redundanzen implementiert werden.

6.1. Zusammenfassung und Diskussion

Aus Sicht der Verfügbarkeit ist die Synchronisation der Schaltung mit dem Gesamtsystem nach einer Fehlerkorrektur ein entscheidender Einflussfaktor. Um Reset Anwendungen zu vermeiden, ist in Kapitel 4 ein Framework zur Auswahl und Untersuchung von Checkpoint und Rollback Methoden in echtzeitfähigen FPGA Schaltungen entstanden, genannt *Checkpoint and Rollback EvaLuation, Optimization and Simulation* (CaRLOS). In echtzeitfähigen Schaltungen müssen Deadlines von Interaktionen mit dem Gesamtsystem eingehalten werden. Hierfür werden die Stillstandszeiten und HW Mehrkosten in der Schaltung optimiert. Anschließend wird als Neuerung die Anwendungswahrscheinlichkeit eines Checkpoints anhand von Deadline Überläufen und der Datengültigkeit bestimmt. Mit dieser Methodik können persistente Fehler mit kleinstmöglichen Auswirkungen auf die Echtzeitfähigkeit korrigiert werden.

Während der Anwendung der Checkpoint Methodik entsteht auch hier der Bedarf nach einer kurzen Fehlerdetektionszeit (FDZ). Ferner ist für die funktionale Sicherheit eine effektive Fehlerdetektion von kritischen Komponenten, anstelle einer Toleranz, zur Erreichung des sicheren Zustands meist ausreichend. Mit diesem Hintergrund kann nun der für flüchtige und permanente Fehler anfällige Konfigurationsspeicher CRAM durch den vorgestellten *Priorisierten Konfigurationsspeicher Test* (PKT) effektiv getestet werden. In Kapitel 5 wird die Erstellung der CRAM Testsequenz aus dem Datenfluss erläutert und angewendet.

Verwendet man nun die entwickelten Methoden in einer echtzeitfähigen FPGA Schaltung und vergleicht sie mit konventionellen Methoden, ist eine deutliche Reduzierung der MTTR bei annähernd gleicher Fehlerabdeckung und sehr geringen Hardware Mehraufwand feststellbar. Abbildung 6.1 zeigt die Verringerung der Detektionszeit im CRAM und der anschließenden Zustandswiederherstellung einer im FPGA implementierten MCU. Die konventionellen CRAM Testmethoden Linearer Vollständiger Konfigurationstest (LVKT) und Linearer Essentieller Frame Konfigurationstest (LEFK) werden mit dem PKT ebenso verglichen wie der Schaltungsreset mit der Checkpoint Methode aus dem CaRLOS Framework. Der Reset wird inklusive Synchronisation mit 1 ms bemessen. Als Referenz wird die Kombination LVKT plus Reset mit 100 % Zeit angenommen. Nach der Anwendung der CRAM Fehlerdetektion mittels PKT und der CP Einführung sinkt die MTTR auf ca. 37 %.

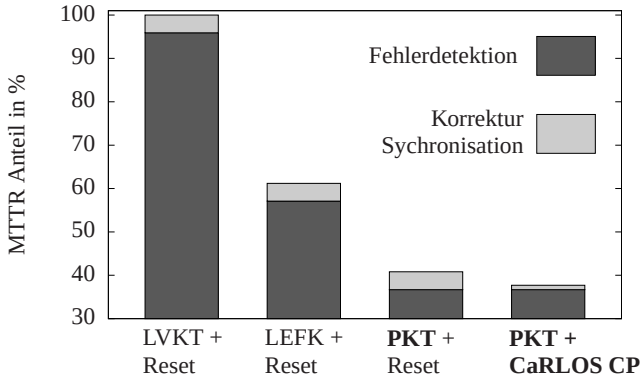


Abbildung 6.1.: Vergleich der Mittleren Fehlerdetektions- und Reparaturzeit (MTTR) für unterschiedliche CRAM Testmethoden und Zustandskorrekturen am Beispiel eines MCU

Zur Anwendung dieser redundanzfreien Methoden müssen jedoch zwei Randbedingungen erfüllt werden. Zum einen bedarf der bei großen FPGAs sinnvolle PKT durch das verwendete Scheduling mehrere Testbereiche mit unterschiedlichen Testanforderungen. Zukünftige komplexe Systeme auf großen FPGAs ab einer Größe des CRAM von 50 MBit werden jedoch meist unabhängige Funktion an Bord haben. Zum anderen muss der frame-basierte Zugriff auf den CRAM möglich sein. Dies ist aktuell nicht bei allen FPGA Herstellern gegeben.

Ferner werden für das anschließende Checkpoint und Rollback mehrere Voraussetzungen verlangt. Zum Beispiel muss die Anwendung im Time-Triggered Entwicklungsstil mit Deadlines entwickelt werden und die Ausführungszeit einer Funktion muss sich an den optimierten zeitlichen Mehraufwand durch das Checkpointing anpassen. Liegen diese Bedingungen vor, dann kann die CP Methodik und Strategie optimiert und die Erfolgswahrscheinlichkeit für den erfolgreichen Einsatz bestimmt werden.

Insgesamt lag der Fokus auf redundanzfreien und kostengünstigen Methoden zur Verbesserung der Einsatzfähigkeit von FPGAs in hohen Stückzahlen im automobilen Sektor. Dieses Ziel wurde sowohl für die Fehlerdetektion als auch für die Zustandskorrektur bei einer gleichzeitigen Optimierung der Verfügbarkeit erreicht.

6.2. Ausblick

Um die Restriktionen für die Checkpoint Anwendung innerhalb der FPGA Schaltungen zu minimieren, sollte das Gesamtsystem konzeptuell an die Effekte des Rollbacks angepasst werden. Hier bietet sich bei rein digitalen Systemen ein systemweiter und koordinierter Checkpoint an. Analoge Umgebungen in ASICs oder anderen Sensor-Aktor Systemen sollten auf verfügbare Latenzzeiten überprüft werden. Zusammen mit der deadline-unterstützen Time-Triggered Methodik können zukünftig Echtzeitsysteme entwickelt werden, die in ihrer Performance die Kapazitäten für Checkpoint und Rollback mitbringen. Eine weitere Beschleunigung der Checkpoint Erstellzeit und der Zeit für den Rollback kann durch eine schnelle externe Speicheranbindung oder On-Chip Speicher erreicht werden. Zudem können bereits vorhandene Boundary-Scan Teststrukturen innerhalb des ICs für die Netzlisten-Verfahren verwendet werden. Einmal ausgewählte Checkpoint Methoden müssen ferner für den produktiven Einsatz durch validierte Tools in die Netzliste überführt werden.

Neben den Checkpoint Erweiterungen kann auch der PKT verbessert werden. Hierfür könnte für große

Einzelfunktionen innerhalb des FPGAs eine zeitliche Korrelation von Teilfunktionalitäten erfolgen, die eine gewisse Netzlistenabhängigkeit zeigen. Dafür wird der Datenfluss statt auf Funktionsebene auf Gatterebene bestimmt und mit den CRAM Frames korreliert. Andererseits können FPGAs auch feingranularer mit dedizierten Funktionen genutzt werden und somit kann die Testzeit mittels PKT optimal klein gehalten werden. Überdies werden aktuell viele nicht essentielle Bits unnötigerweise mittels ECC geprüft. Diese Granularität kann auf Kosten des Speicherbedarfs für die Testsequenz verkleinert werden.

Zur redundanzfreien Erhöhung der Verfügbarkeit können neben den Konfigurationstests und den Checkpoint und Rollback Konzepten auch anderweitige Methoden weiterentwickelt werden. Speziell bei der Anwendung formaler Ausdrücke zur Plausibilisierung von Ausgangsdaten liegt viel Potential. Eine Fehlerkorrektur mittels prädiktivem Roll-Forward oder einer Kombination aus Reset und partiellem Rollback kann für bestimmte periodische oder interruptgesteuerte Anwendungen eine bessere Vorhersagbarkeit zeigen. Dennoch liefert der PKT und das CaRLOS Framework für die Testbarkeit und Verfügbarkeit in echtzeitfähigen FPGA Schaltungen bei verteilten und eingebetteten Funktionen mit nicht ausgeschöpfter Latenz in der Regelschleife einen sehr guten Beitrag.

A. Anhang

A.1. Code Templates

```
entity for_model is
end for_model;

architecture fli of for_model is
  -- pass clock signal
  attribute foreign : string;
  attribute foreign of a : architecture is "InitMonitor FLI_Interface.dll; clk";
begin
end fli;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top_level is
  port (clk : in std_logic
  );
end top_level;

architecture STRUCTURE of top_level is

  component for_model is
  end component;

  for all : for_model use entity work.for_model(fli);

begin

  Model1 : for_model;

  -- top_level architecture here

end architecture STRUCTURE;
```

Listing A.1: Instanziierung der Funktion `InitMonitor` Ã¼ber Modelsim®FLI in VHDL

A.2. Design Dateien

A.2.1. EDIF Netzlistenformat

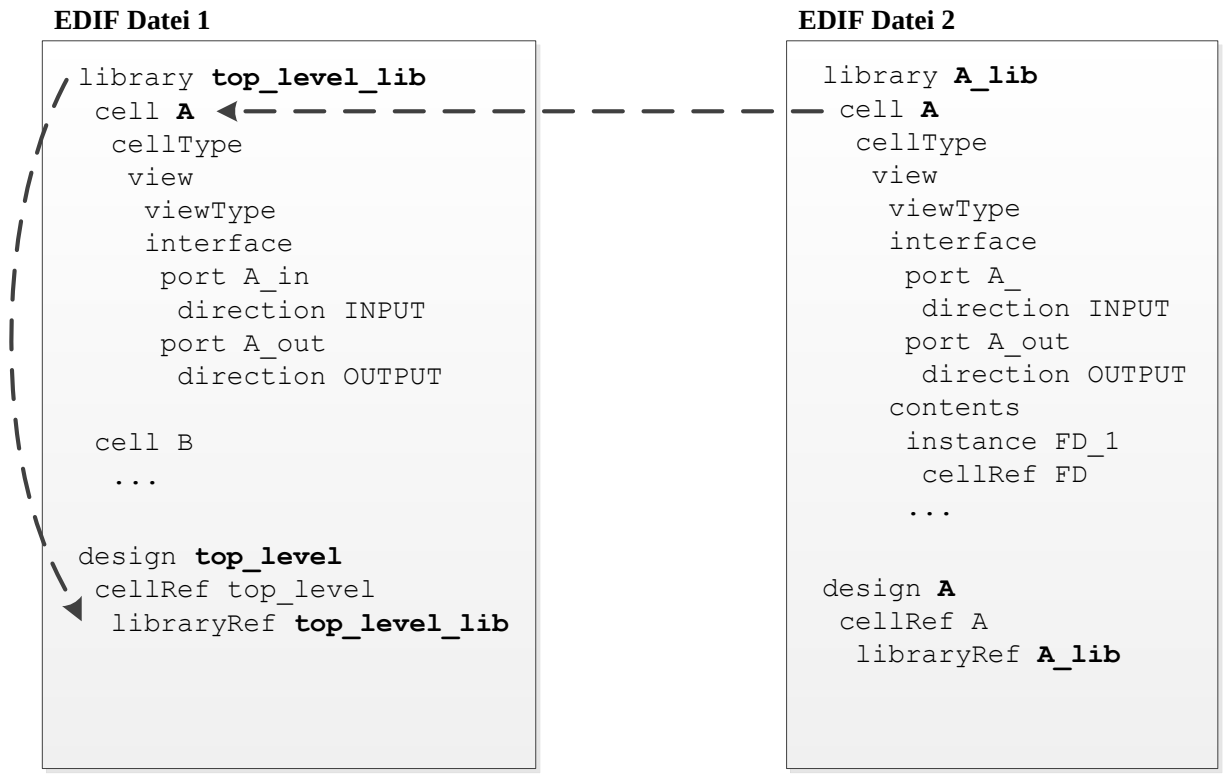


Abbildung A.1.: EDIF Netzlistenformat bei hierarchischen Schaltungen (verkürzt). Die Beschreibung des Schaltungsmoduls `A` ist in eine zweite EDIF Datei ausgelagert. Dieser hierarchische Aufbau wird mittels gleicher Bibliotheksnamen aufgelöst. Der Inhalt von Modul `A` befindet sich ebenfalls in Datei 2. Datei 1 enthält lediglich den Rumpf von `A`. Die Pfeile deuten die Referenzen bis zum `top_level` an.

A.2.2. XDL Platzierungsformat

XDL Datei

```
...

inst „top_level/A/slice“ „SLICEL“,placed CLBLM_L_X26Y161 SLICE_X_38YY161
 cfg „
     FF, LUT, MUX, Init, Carry Einstellungen
     „;

...

net „A_in“ ,
 outpin „top_level/input_pin_IBUF“ A1 ,
 inpin „top_level/A/slice“ B5 ,
 pip CLBLM_L_X26Y161 CLBLL_IMUX26 -> CLBLM_L_B5 ,
 pip INT_L_X26Y161 LOGIC_OUTS_L10 -> IMUX_L37 ,
 ...
 ;
```

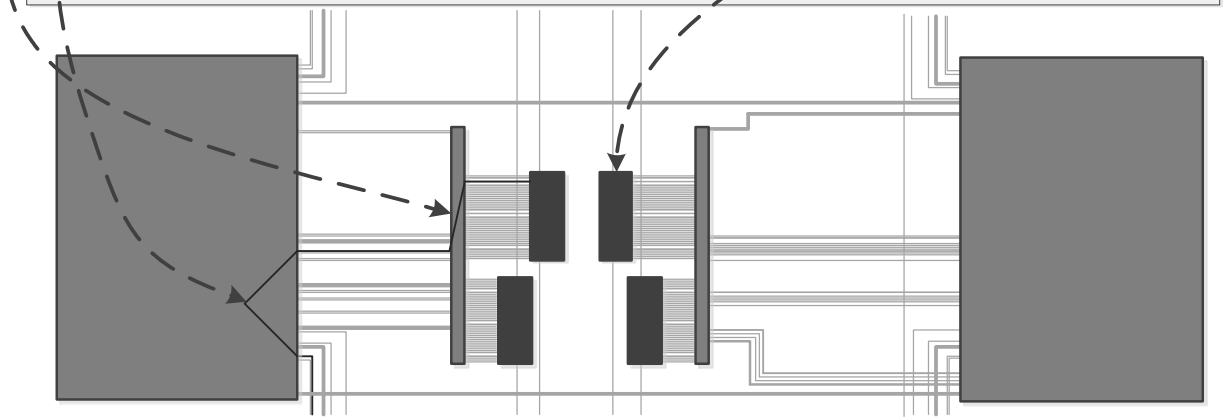


Abbildung A.2.: XDL Dateiformat mit skizzierter Zugehörigkeit von Layout Ressourcen. Aus den Koordinaten platzierter Logikinstanzen und Netzinstanzen lassen sich die MF Koordinaten bestimmen.

A.3. Checkpoint und Rollback

A.3.1. Netzlistenrepräsentation

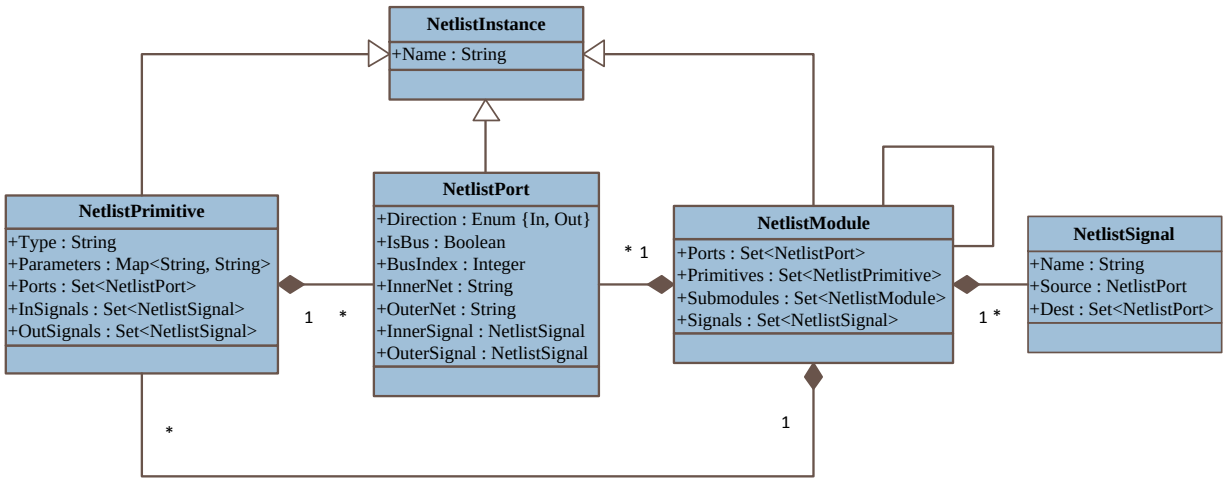


Abbildung A.3.: Klassendiagramm der internen Netzlistenrepräsentation

A.3.2. Ergebnisse Fallbeispiel MicroBlaze MCS

Tabelle A.1.: Ergebnisse der Netzlistenanalyse des MicroBlaze MCS

FF	BRAM	LUT-RAM	A_{CP}	t_{CpHalt} [μs]	t_{Cp} [μs]	t_{Rb} [μs]	Lösungsindex
1× SC	PDR	PDR	0,620	849,1	849,1	849,1	
	PDR	MM	0,635	410,1	410,1	410,1	
	MM	PDR	0,648	526,1	526,1	526,1	
	MM	MM	0,438	87,0	87,0	87,0	
10× SC	PDR	MM	0,627	845,8	845,8	845,8	
	MM	PDR	0,642	406,8	406,8	406,8	
	MM	MM	0,655	522,8	522,8	522,8	
	PDR	MM	0,446	83,7	**83,7**	**83,7**	A
1× SHC	PDR	PDR	1,493	845,0	849,1	849,1	
	PDR	MM	1,508	405,9	410,1	410,1	
	MM	PDR	1,520	521,9	526,1	526,1	
	MM	MM	1,311	**82,9**	87,0	87,0	B
10× SHC	PDR	PDR	1,500	845,0	845,8	845,8	
	PDR	MM	1,515	405,9	406,8	406,8	
	MM	PDR	1,528	521,9	522,7	522,7	
	MM	MM	1,318	**82,9**	**83,7**	**83,7**	C
PDR	PDR	PDR	**0,211**	845,0	1285,0	1285,0	D
	PDR	MM	0,226	405,9	1285,9	1285,9	
	MM	PDR	0,239	521,9	961,9	961,9	
	MM	MM	0,241	**82,9**	962,9	962,9	E
EA			0,438	87,0	87,0	87,0	F
FF: SC, SHC			0,854	83,5	84,1	84,1	G
RAM: MM, PDR			0,745	83,7	84,0	84,0	H

In Abbildung A.4 sind 4 Signale der MCS Schaltung dargestellt. Bei diesem Beispiel wurde die Methode D aus Abbildung 4.14a auf die Schaltung angewendet. Das entspricht einer vollständigen PDR und wurde hier aufgrund der sichtbaren Effekte auf die Schaltung ausgewählt. Während der globale Takt `clk` aktiv bleibt, ist die Schaltung für eine Haltezeit von $t_{CpHalt} = 845$ μs im Checkpoint Modus. Diese inaktive Phase während des Checkpoints zeigt sich am Signal `cnt` bei 2 ms. Ferner muss für einen Rollback die Schaltung ebenfalls angehalten werden, sichtbar bei 5 ms simulierter Zeit. Der wertestabile Rollback ist an dem Zählerwert `gpo0` erkennbar.

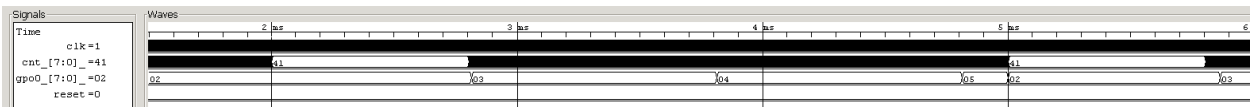


Abbildung A.4.: PDR Checkpoint bei 2 ms und PDR Rollback bei 5 ms

A.4. Priorisierter Konfigurationstest

A.4.1. Kombinatorische Testbereiche

Praktische Nutzung der Aufteilung in Kombinatorische Testbereiche:

$$T_{clk} = 10ns$$
$$t_{Test,MF} = 40\mu s$$
$$\text{Komb. Testbereiche pro MF} = 4000$$

$t_{Test,MF}$ = Testzeit für ECC eines Major Frames

Demnach müssen bei einer Frequenz von 100 MHz etwa 4000 kombinatorische Testbereiche durchlaufen werden, um einen MF zu testen.

A.4.2. Herleitung der mittleren Testdauer eines Major Frames der Xilinx 7 Serie

Es gibt 5 verschiedene Arten von Major Frame Typen. Diese Arten sind in Tabelle A.2 mit den entsprechenden Häufigkeiten und Minor Frames aufgeführt. Für den ECC Prüfung werden nur 4 Blocktypen verwendet, da der BRAM Inhalt keinen statischen ECC besitzt. Jeder Major Frame eines Blocktyps hat eine entsprechende Anzahl von Minor Frames. Die Frame Adressen von Blocktyp 0 und Typ 2 sind abgesehen vom Blocktyp identisch. Somit muss mit jedem Blocktyp 0 auch ein Typ 2 während der ECC Prüfung ausgelesen werden.

Tabelle A.2.: Arten der Blocktypen von Major Frames für den ECC (Xilinx 7 Serie)

	Typ 0 (CLB)	Typ 1 (BRAM Inhalt)	Typ 2 (CLB Konfig)	Typ 3	Typ 4
Anzahl Major Frames in Kintex 7 XC7K325T:	648	-	648	45	1
Enthaltene Minor Frames:	siehe Tabelle A.3	128	1	1	1

Die Informationen über die Logik enthält der Blocktyp 0. Dieser Blocktyp konfiguriert sowohl die CLBs als auch die I/O Blöcke, DSP und BRAM Einstellungen und globale Takteinstellungen. Eine Übersicht ist in Tabelle A.3 gegeben. Die Anzahl der enthaltenen Minor Frames sind für die gesamte Xilinx 7 Serie identisch. Obwohl die Anzahl der Major Frames pro FPGA über die Produktfamilie variiert, ist der Anteil der Arten ähnlich über die Familie. Aus dieser Vielfalt hat die Konfiguration der CLBs einen Anteil von 83,6 % und nimmt demnach den Hauptteil der ECC Prüfung ein.

Tabelle A.3.: Arten der Major Frames für den Blocktyp 0 (Xilinx 7 Serie)

	CLB	I/O	DSP und BRAM Settings	GCLK	andere
Enthaltene Minor Frames:	36	42	28	30	32
Anzahl in Kintex 7 XC7K325T:	523	10	87	24	4
Anteil A in %:	83,6	1,9	10,8	3,2	0,6

Die mittlere Testdauer eines Major Frames wird durch den Blocktyp 0 bestimmt, da dieser viele Minor Frames enthält. Die Formel A.1 beschreibt die mittlere Zeit für das Auslesen der Daten eines Major Frames vom Typ 0.

$$\begin{aligned} t_{MF} &= \frac{1}{100} \sum_{i=0}^{4} A_i \cdot N_{Minor,i} \cdot N_{Words/Minor} \cdot T_{clk,ICAP} \\ &= 35,1\mu s \end{aligned} \tag{A.1}$$

t_{MF}	=	Mittlere Auslesezeit der Daten eines Major Frames vom Typ 0
A_i	=	Anteil der Art des Major Frames am gesamten Typ 0
$N_{Minor,i}$	=	Anzahl der Minor Frames pro Major Frame vom Typ 0
$T_{clk,ICAP}$	=	10 ns. Periodendauer der Auslesefrequenz via ICAP
$N_{Words/Minor}$	=	101 Wörter mit 32 bit pro Wort

Die mittlere Testzeit eines Major Frames enthält neben dem Lesen der Framedaten zudem das Auslesen eines Minor Frames vom Typ 2 und die Ansteuerlogik des ICAP.

$$\begin{aligned} t_{test,MF} &= t_{MF} + t_{minor,Typ2} + t_{FSM,ICAP} \\ &\approx 40\mu s \end{aligned} \tag{A.2}$$

$t_{test,MF}$	=	Mittleren Testdauer eines Major Frames
t_{MF}	=	35,1 μs
$t_{minor,Typ2}$	$\approx$	1,01 μs. Auslesezeit der Daten eines Minor Frames
$t_{FSM,ICAP}$	$\approx$	1,6 μs. Ansteuerlogik zum Setzen der internen FSM im ICAP.

Für eine konservative Anwendung des ECCs eines Major Frames wird somit eine Testzeit von 40 μs angenommen.

A.4.3. Aufbau der Frameadressen der Xilinx 7er Serie

Die Frameadresse besteht aus 26 Bit, verwendet nach folgendem Pattern:

Die Frame Adressen von Blocktyp 0 und Typ 2 sind identisch

Tabelle A.4.: Aufbau der Frameadresse

Bit:	[25:23]	22	[21:17]	[16:7]	[6:0]
Inhalt:	Blocktyp	Oberer / Unterer Bereich	Reihe	Major Adresse (Spalte)	Minor Adresse

A.4.4. Frame auslesen mittels ICAP

```
process(clk)
begin
  if rising_edge(clk) then
    if cnt = 63 then
      cnt <= 0;
    elsif cnt = 0 then
      if Start = '1' then  -- wait till start
        cnt <= cnt+1;
      end if;
    elsif (cnt /= 52) then
      cnt <= cnt +1;
    elsif word_count = 0 then
      cnt <= cnt +1;
    end if;
  end if;
end process;

process(clk)
begin
  if rising_edge(clk) then
    case cnt is
    when 0 =>  data2icap <= DUMMY;
               icap_rw <= ICAP_write;
               done <= '0';
               ram_we <= "0000";
               buffer_address <= "0000000000";
               if Start = '1' then
                 word_count <= unsigned(FrameCount) * to_unsigned(words_per_minor, 7);
                 frame_address <= StartFrame;
               end if;
    when 1 =>  data2icap <= DUMMY;
                icap_cs <= ICAP_active;
    when 2 =>  data2icap <= SYNC;
    when 3 | 4 =>  data2icap <= NOP;

    when 5 =>  data2icap <= Type1Write(CMD_REG, 1);
    when 6 =>  data2icap <= expand(CMD_RCFG, 32);
    when 7 | 8 | 9 => data2icap <= NOP;

    when 10 => data2icap <= Type1Write(FAR_REG, 1);
    when 11 => data2icap <= expand(frame_address, 32);

    when 12 => data2icap <= Type1Read(FDRO_REG, 0);
    when 13 => data2icap <= Type2Read(expand(word_count, 27));
    when 14 to 45 => data2icap <= NOP;
    when 46 => icap_cs <= ICAP_inactive;
     when 47 => icap_rw <= ICAP_read;
    when 48 => data2icap <= NOP;
```

```
                icap_cs <= ICAP_active;
    when 49 | 50 | 51 => data2icap <= NOP;
    when 52 =>
      icap_cs <= ICAP_active;
      ram_we <= "1111";
      if word_count /= 0 then
        word_count <= word_count - 1;
        buffer_address <= buffer_address + 1;
      else
        icap_cs <= ICAP_inactive;
        ram_we <= "0000";
      end if;
    when 53 => icap_rw <= ICAP_write;
               ram_we <= "0000";
    when 54 => icap_cs <= ICAP_active;
               data2icap <= NOP;
    when 55 => data2icap <= Type1Write(CMD_REG, 1);
    when 56 => data2icap <= expand(CMD_DESYNC, 32);
    when 57 to 61 => data2icap <= NOP;
    when 62 => data2icap <= NOP;
               done <= '1';
    when 63 => icap_cs <= ICAP_inactive;
               done <= '0';
    when others => null;
    end case;
  end if;
end process;
```

Listing A.2: VHDL Sequenz zum Auslesen von Frames mittels ICAP

A.4.5. Frame schreiben mittels ICAP

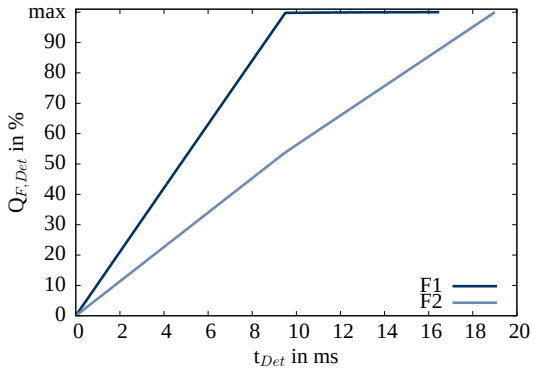
```
process(clk)
begin
  if rising_edge(clk) then
    if cnt = 74 then
      cnt <= 0;
    elsif cnt = 0 then
      if Start = '1' then  -- wait till start
        cnt <= cnt+1;
      end if;
    elsif (cnt /= 61 AND cnt /= 62) then      --write data
      cnt <= cnt +1;
    elsif (word_count = 0) then
      cnt <= cnt +1;
    end if;
  end if;
end process;

process(clk)
begin
  if rising_edge(clk) then
    case cnt is
    when 0 =>  data2icap <= DUMMY;
               icap_rw <= ICAP_write;
```

```
            done <= '0';
            ram_we <= "0000";
            if Start = '1' then
                word_count := WORDS_PER_MINOR;
            end if;
    when 1 =>  data2icap <= DUMMY;
               icap_cs <= ICAP_active;
    when 2 =>  data2icap <= SYNC;
    when 3 | 4 =>  data2icap <= NOP;

    when 5 =>  data2icap <= Type1Write(CMD_REG, 1);
    when 6 =>  data2icap <= expand(CMD_RCRC, 32);
    when 7 | 8 | 9 => data2icap <= NOP;

    when 10 => data2icap <= Type1Write(IDCODE_REG, 1);
    when 11 => data2icap <= expand(DEVICE_ID, 32);
    when 12 to 20 => data2icap <= NOP;

    when 21 => data2icap <= Type1Write(FAR_REG, 1);
    when 22 => data2icap <= expand(frame_address, 32);
    when 23 | 24 => data2icap <= NOP;

    when 25 =>  data2icap <= Type1Write(CMD_REG, 1);
    when 26 =>  data2icap <= expand(CMD_WCFG, 32);
    when 27 to 58 => data2icap <= NOP;

    when 59 => data2icap <= Type1Write(FDRI_REG, 0);
    when 60 => data2icap <= Type2Write(expand(to_unsigned(2*WORDS_PER_MINOR, 9), 27));
            frame_buffer_address <= "0000000000";

    when 61 =>                  --write data frames
            if (flip = '1' and word_count = 10) then
              data2icap(31 downto 1) <= frame_content_word(31 downto 1);
              data2icap(0) <= not frame_content_word(0);
            else
              data2icap <= frame_content_word;
            end if;


            frame_buffer_address <= std_logic_vector(unsigned(frame_buffer_address) + 1);
            if (word_count > 0) then
                word_count := word_count - 1;
            else
              word_count := WORDS_PER_MINOR;
            end if;

    when 62  =>     --write dummy frames
           data2icap <=  x"00000000";
           if (word_count > 0) then
                word_count := word_count - 1;
            else
              word_count := WORDS_PER_MINOR;
            end if;

    when 63  => data2icap <= NOP;
    when 64 => data2icap <= NOP;
    when 65 => data2icap <= Type1Write(CMD_REG, 1);
```

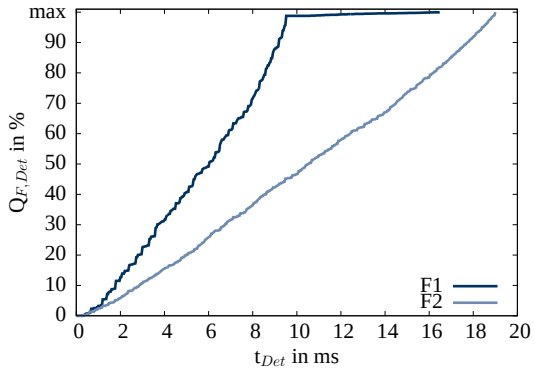
```
81      when 66 => data2icap <= expand(CMD_DESYNC, 32);
        when 67 to 72 => data2icap <= NOP;
83      when 73 => data2icap <= NOP;
                   done <= '1';
85      when 74 => icap_cs <= ICAP_inactive;
                   done <= '0';
87      when others => null;

89      end case;
    end if;
91  end process;
```

Listing A.3: VHDL Sequenz zum Schreiben eines Frames mittels ICAP

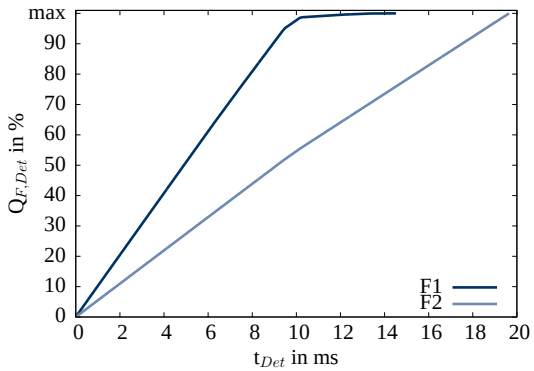
A.4.6. Detaillierte Ergebnisse der homogenen Testverteilung für Dual Core Microcontroller



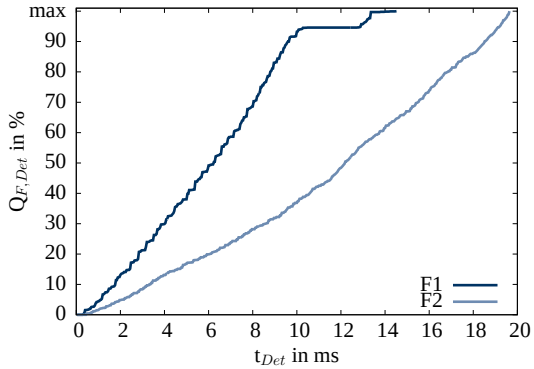
(a) Mittelwert von $Q_{F,Det}$ bei der linearen Klassifizierung über den Mittelwert $\overline{\Phi}_M$



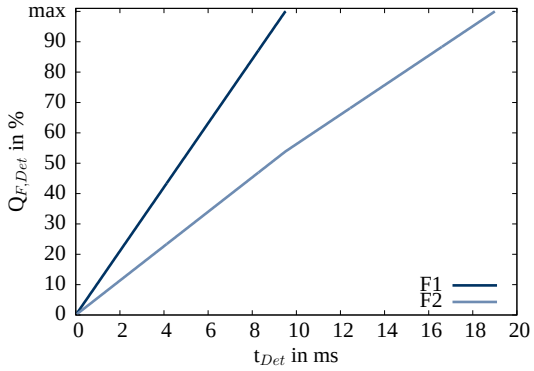
(b) Schlechtester Fall von $Q_{F,Det}$ bei der linearen Klassifizierung über den Mittelwert $\overline{\Phi}_M$



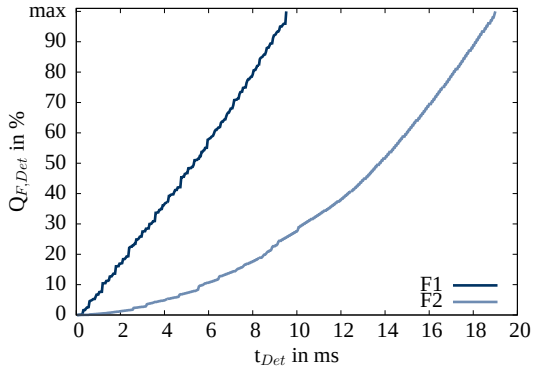
(c) Mittelwert von $Q_{F,Det}$ des NSGA-II Resultats nach 4000 Generationen, Population = 700



(d) Schlechtester Fall von $Q_{F,Det}$ des NSGA-II Resultats nach 4000 Generationen, Population = 700



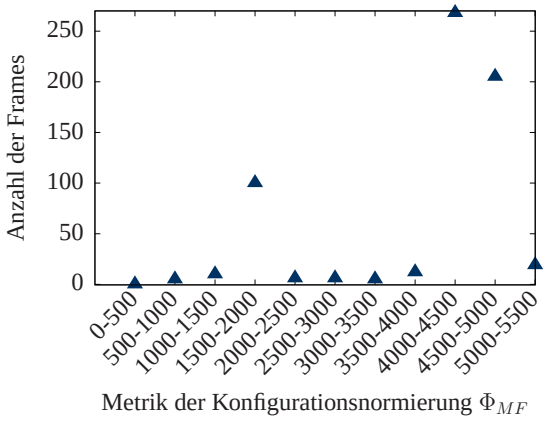
(e) Mittelwert von $Q_{F,Det}$ bei der EDF-Verteilung nach der linearen Klassifizierung



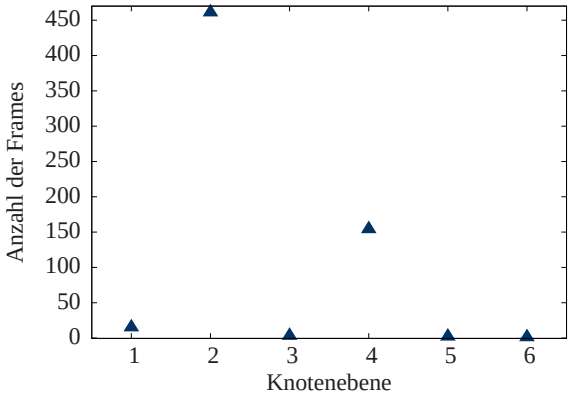
(f) Schlechtester Fall von $Q_{F,Det}$ bei der EDF-Verteilung nach der linearen Klassifizierung

Abbildung A.5.: Fehlerabdeckung über die Zeit nach einem auftretenden Fehler zum beliebigen Zeitpunkt

A.4.7. Detaillierte Ergebnisse der homogenen Testverteilung für den Echtzeit Timer

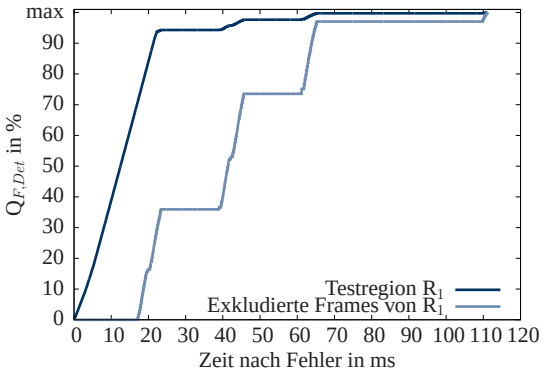


(a) Anzahl der Frames für Wertebereiche von Φ_{MF}

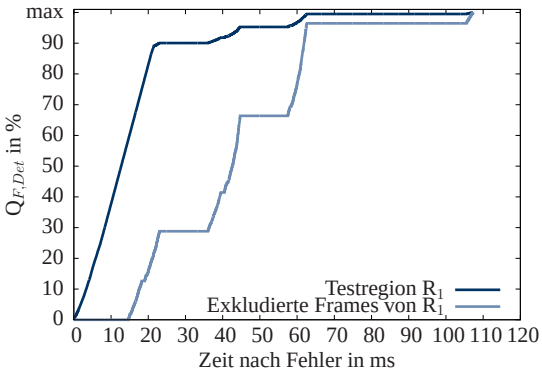


(b) Anzahl der Frames für die Knotenebenen im Abhängigkeitsbaum

Abbildung A.6.: Eigenschaften der MFs mit Fokus auf Konfigurationsnormierung und Knotenebene



(a) Schlechtester Fall von $Q_{F,Det}$ mit 9% exkludierten Frames



(b) Schlechtester Fall von $Q_{F,Det}$ mit 14% exkludierten Frames

Abbildung A.7.: Fehlerabdeckung über die Zeit nach einem auftretenden Fehler zum beliebigen Zeitpunkt

Abbildungsverzeichnis

2.1. Grundstruktur der Island-Based Architektur mit Konfigurationsspeicher 7
2.2. Typische Leitungsarten zur Verbindung von Logikblöcken 7
2.3. Grundprinzip Partielle Rekonfiguration 9
2.4. FPGA spezifisches V-Modell in Industrieapplikationen 11
2.5. Strahlungsdefekt durch Single Event Upset nach [NYKB08] 13
2.6. SEU in SRAM Zelle 13
2.7. Fehlerausbreitung und Fehlermaskierungen nach SEE in der Logik 13
2.8. Redundanzen zur Fehlerdetektion und Fehlertoleranz 15
2.9. Fehlerdetektion in der Logik einer FPGA Schaltung 16
2.10. Mittelwert SER für Xilinx FPGAs [Xil13c] 17
2.11. Logische und physikalische Anordnung von CRAM Frames 18
2.12. Checkpoint Varianten für Echtzeitsysteme 23
2.13. Schematische Darstellung der verwendeten Checkpoint Methoden 25

3.1. Fehlertoleranzzeit nach ISO 26262 28
3.2. Architekturvorschläge von *fail-safe* Steuergeräten mit FPGA 30
3.3. Reduzierung der Verfügbarkeit im Fehlerfall 33

4.1. Bedingungen für den Einsatz von Checkpoint und Rollback 39
4.2. Modellierung der Checkpoint Methoden im FF Modell 42
4.3. Exemplarische SystemC Netzliste mit hierarchischem Aufbau 43
4.4. Vergleich der Simulationszeiten verschiedener Netzlisten-Simulatoren mit einer Taktfrequenz der Schaltung von 100 MHz 44
4.5. Überblick über die Kenndaten des Checkpoints auf Netzlistenebene 45
4.6. Schematische Darstellung der Anbindung der Schaltungsanalyse zum Evolutionären Algorithmus 47
4.7. Datengültigkeit: Zeitraum für die Anwendung eines Checkpoints 49
4.8. Zeitliche Gültigkeit bei Checkpoint und Rollback 50
4.9. Wiederholdauer t_w nach einem Rollback 50
4.10. Zeitliche Gültigkeit bei Checkpoint und Rollback 52
4.11. Checkpoint Strategie in Abhängigkeit der Checkpoint Zeitpunkte 52
4.12. Ablauf der Checkpoint Evaluation für FPGA Echtzeitsysteme 53
4.13. Blockschaltbild des Analysetools CaRLOS zur hybriden Evaluation von Checkpoint Methoden in Schaltungen 54
4.14. Übersicht der Checkpoint Methoden des MCS 55
4.15. Vergleich der Evaluationszeiten verschiedener Tools für eine Checkpoint Methode am Beispiel MCS 57
4.16. Blockschaltbild mit Deadlines des Timermoduls 58
4.17. Pareto-Optimale Checkpoint Kenngrößen für die Netzliste des Timermoduls 59
4.18. Vergleich einzelner Checkpointing Methoden anhand von Parametern 60
4.19. Theoretische Verzögerung des Timings bei CP Erstellung und nach einem Rollback 61
4.20. Datengültigkeiten und zeitliche Gültigkeiten unter Anpassung der Schaltungsperformance . . 62
4.21. Qualitativer Vergleich des CaRLOS Frameworks mit konventionellen CP Entwicklungstools . 64

5.1. Grundprinzip eines PKT 68
5.2. Implementierungsvorgang für den Offline PKT 69
5.3. Gruppierung der Datenflüsse für PKT 70
5.4. Verknüpfung der HDL Signale über die Netzliste mit den FPGA Ressourcen 71
5.5. Bestimmung der *Einhüllenden Testregion* 73
5.6. Priorisierung der Testregionen mit $\mathrm{FDZ}_{F1} < \mathrm{FDZ}_{F2}$ 75
5.7. Abdeckung des Verbindungsgitters innerhalb des PKT 76

5.8. Ablaufgraph für Zeitgesteuerte Testverteilung . . . 78
5.9. EDF-KT Scheduling . . . 80
5.10. Variablen der SAT Kodierung des PKT Scheduling . . . 81
5.11. Idealisierte homogene Testverteilung des PKT . . . 82
5.12. Ablaufgraph für Homogene Testverteilung . . . 83
5.13. Übergang paralleler Testanforderungen zu einer linearen Klassifizierung mittels Konfigurationsnormierung . . . 83
5.14. Verlauf des Evolutionären Algorithmus zur Erstellung eines PKT . . . 86
5.15. Anbindung des EA an das PKT Tool . . . 88
5.16. Veränderung der Testzeiten bei Varianzen der Anzahl gemeinsam genutzter Frames und FDZ Anforderungen. Die senkrechte Markierung zeigt die aktuell gesetzten Anforderungen. . . . 89
5.17. Blockschaltbild der PKT Implementierung im FPGA . . . 90
5.18. Vereinfachter Ablauf der Fehlerdetektion DKE . . . 90
5.19. Speicherverteilung der Testsequenz . . . 91
5.20. Blockschaltbild zur Fehlerinjektion und Fehlerdetektion der Fehlerauswirkung inklusive PKT 92
5.21. Ablauf der Fehlerinjektion und Verifikation der Testschaltung . . . 93
5.22. Dual Controller Anwendung im FPGA . . . 95
5.23. Testregionen im Layout mit allen (blaue, helle Punkte) und funktional essentiellen (rote, dunkle Punkte) Konfigurationselementen inklusive essentieller MF (grüne, dunkle Rechtecke) . . . 96
5.24. Scheduling der Testregionen als Gantt-Diagramm . . . 96
5.25. Fehlerabdeckung über die Zeit nach einem auftretenden Fehler zum beliebigen Zeitpunkt . . . 98
5.26. Ideale Veränderung der Testzeit durch Variation des Layouts und der Anforderungen. Die senkrechte Markierung zeigt die aktuell gesetzten Anforderungen. . . . 98
5.27. Ergebnisse auf einem Xilinx Kintex 7 XC7K325T . . . 99
5.28. Redundanzschaltung mit Überprüfer . . . 100
5.29. Blockschaltbild des Timermoduls mit einer gesetzten FDZ . . . 101
5.30. Verlauf der Fehlerabdeckung und Verkleinerung der Testregion . . . 102
5.31. Ergebnis der Aufteilung des Timermoduls in zwei Testgruppen . . . 103
5.32. Quantitativer Vergleich des PKT mit konventionellen CRAM Fehlerdetektionsmethoden . . . 104

6.1. Vergleich der Mittleren Fehlerdetektions- und Reparaturzeit (MTTR) für unterschiedliche CRAM Testmethoden und Zustandskorrekturen am Beispiel eines MCU . . . 108

A.1. EDIF Netzlistenformat bei hierarchischen Schaltungen (verkürzt). Die Beschreibung des Schaltungsmoduls `A` ist in eine zweite EDIF Datei ausgelagert. Dieser hierarchische Aufbau wird mittels gleicher Bibliotheksnamen aufgelöst. Der Inhalt von Modul `A` befindet sich ebenfalls in Datei 2. Datei 1 enthält lediglich den Rumpf von `A`. Die Pfeile deuten die Referenzen bis zum `top_level` an. . . . 113
A.2. XDL Dateiformat mit skizzierter Zugehörigkeit von Layout Ressourcen. Aus den Koordinaten platzierter Logikinstanzen und Netzinstanzen lassen sich die MF Koordinaten bestimmen. . . . 114
A.3. Klassendiagramm der internen Netzlistenrepräsentation . . . 115
A.4. PDR Checkpoint bei 2 ms und PDR Rollback bei 5 ms . . . 116
A.5. Fehlerabdeckung über die Zeit nach einem auftretenden Fehler zum beliebigen Zeitpunkt . . . 123
A.6. Eigenschaften der MFs mit Fokus auf Konfigurationsnormierung und Knotenebene . . . 124
A.7. Fehlerabdeckung über die Zeit nach einem auftretenden Fehler zum beliebigen Zeitpunkt . . . 124

Literaturverzeichnis

[AEC07] AEC-Q100 Rev.G. Failure mechanism based stress test qualification for integrated circuits, May 2007.

[Alt12] Altera Corporation. Error correction code in SoC FPGA-based memory systems. Technical Report WP-01179-1.2, April 2012.

[Alt13a] Altera Corporation. SEU mitigation for stratix v devices. Technical Report Stratix V Device Handbook Volume 1: Device Interfaces and Integration, May 2013.

[Alt13b] Altera Corporation. Stratix v device datasheet. Technical report, November 2013.

[Alt14] Altera Corporation. Stratix v device handbook. Technical report, January 2014.

[ATT07] H. Asadi, M.B. Tahoori, and C. Tirumurti. Estimating error propagation probabilities with bounded variances. In *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on*, pages 41–49, 2007.

[BBD+08] Eva Beckschulze, David Boymanns, Ramona Dülks, Thomas Gatterdam, Prof. Dr.-Ing. Stefan Kowalewski, Martin Lang, Dr.-Ing. Falk Salewski, and Thomas Siegbert. Zuverlässigkeit von automotive embedded systems. FAT/AK31-UAK Zuverlässigkeit: Projektbericht 2007-2008 FAT 231, Forschungsvereinigung Automobiltechnik e.V. (FAT), RWTH Aachen University, 2008.

[Bie13] Armin Biere. Lingeling, plingeling and treengeling entering the SAT competition 2013. In *Institute for Formal Models and Verification*, Johannes Kepler University Linz, 2013.

[BMS11] C. Bolchini, A. Miele, and C. Sandionigi. A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs. *Computers, IEEE Transactions on*, 60(12):1744 –1758, December 2011.

[BMVBK10] I. Bahri, E. Monmasson, F. Verdiers, and M.-A. Ben Khelifa. Design and validation methodology of FPGA-based motor drive for high-temperature environment. In *Electrical Systems for Aircraft, Railway and Ship Propulsion (ESARS), 2010*, pages 1 –6, October 2010.

[Bos13] Bosch Engineering GmbH. Engine control unit MS 5.1, October 2013.

[BP93] N.S. Bowen and D.K. Pradham. Processor- and memory-based checkpoint and rollback recovery. *Computer*, 26(2):22–31, February 1993.

[BPP+07] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. LaBel, M. Friendlich, H. Kim, and A. Phan. Effectiveness of internal vs. external SEU scrubbing mitigation strategies in a xilinx FPGA: Design, test, and analysis. In *Radiation and Its Effects on Components and Systems*, pages 1 –8, September 2007.

[BRM99] Vaughn Betz, Jonathan Rose, and Alexander Marquardt, editors. *Architecture and CAD for Deep-Submicron FPGAs.* Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[BSV11] Niccolo Battezzati, Luca Sterpone, and Massimo Violante. *Reconfigurable field programmable gate arrays for mission-critical applications.* Springer, 2011.

[But11] G.C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Real-Time Systems Series. Springer, 2011.

[Cad12] Cadence Design Systems, Inc. Taming the challenges of 20nm custom/analog design, 2012.

[Car89] William S. Carter. Configurable logic element, June 1989. US Patent 4,870,302.

[CB09] Carl H. Carmichael and Phil Edward Brinkley. Techniques for mitigating, detecting, and correcting single event upset effects, November 2009. US Patent 7,620,883.

[Chu02] Naoya Chujo. Fail-safe ECU system using dynamic reconfiguration of FPGA. In *R & D Review of Toyota CRDL, Vol. 37, No. 2*, pages 54–60, April 2002.

[Cor13] John D. Corbett. The xilinx isolation design flow for fault-tolerant systems. Technical Report Xilinx Inc., UG191, October 2013.

[CPL+02] G. Cellere, A. Paccagnella, L. Larcher, A. Chimenton, J. Wyss, A. Candelori, and A. Modelli. Anomalous charge loss from floating-gate memory cells due to heavy ions irradiation. *Nuclear Science, IEEE Transactions on*, 49(6):3051–3058, 2002.

[CR07] Nianen Chen and Shangping Ren. Building a coordination framework to support behavior-based adaptive checkpointing for open distributed embedded systems. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 257b–257b, 2007.

[CS98] Guohong Cao and Mukesh Singhal. On coordinated checkpointing in distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 9(12):1213–1225, December 1998.

[CSNSM12] Carven Chan, Daniel Schwartz-Narbonne, Divjyot Sethi, and Sharad Malik. Specification and synthesis of hardware checkpointing and rollback mechanisms. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1226–1232, New York, NY, USA, 2012. ACM.

[CWFH12] C. Cullinan, C. Wyant, T. Frattesi, and X. Huang. Computing performance benchmarks among CPU, GPU, and FPGA. 2012.

[CYR09] Nianen Chen, Yue Yu, and Shangping Ren. Checkpoint interval and system's overall quality for message logging-based rollback and recovery in distributed and embedded computing. In *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, pages 315–322, May 2009.

[DASS09] B. Dutton, M. Ali, J. Sunwoo, and C. Stroud. Embedded processor based fault injection and seu emulation for fpgas. In *Proc. Int. Conf. on Embedded Systems and Applications*, pages 183–189, 2009.

[DIM93] DIMACS. Satisfiability suggested format. Technical report, Rutgers University, 1993.

[DLGO04] Eric Dupont, Olivier Lauzeral, Rémi Gaillard, and Marcos Olmos. Radiation results of the SER test of actel, xilinx and altera FPGA instances. Test report, iRoC Technologies, Grenoble, France, April 2004.

[DPAM02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, April 2002.

[Dri11] Cuthbert Dribble. Fundamentals: FPGAs 101 — part 1: Fundamental concepts - electronic products, January 2011.

[DS09a] Bradley F. Dutton and Charles E. Stroud. Built-in self-test of embedded SEU detection cores in virtex-4 and virtex-5 FPGAs. In Hamid R. Arabnia and Ashu M. G. Solo, editors, *ESA*, pages 149–155. CSREA Press, November 2009.

[DS09b] Bradley F Dutton and Charles E Stroud. Single event upset detection and correction in virtex-4 and virtex-5 FPGAs. In *Proc. ISCA Int. Conf. on Computers and Their Applications*, pages 57–62, 2009.

[DW12] Sinjin Dixon-Warren. A review of TSMC 28 nm process technology, December 2012.

[EAWJ02] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, September 2002.

[Ele02] Electronic CAD and Reliability Group. ITC'99 benchmark, b18, 2002.

[Fed14] Alexey Fedorov. High load configurable test project, May 2014.

[FF12] Francisco Fons and Mariano Fons. FPGA-based automotive ECU design addresses AUTOSAR and ISO 26262 standards. *XCELL Journal*, 78(Q1 2012):20 – 31, 2012.

[FMM12] Umer Farooq, Zied Marrakchi, and Habib Mehrez. *Tree-based Heterogeneous FPGA Architectures*. Springer Science+Business, New York, 2012.

[FPV10] G. Foucard, P. Peronnard, and R. Velazco. Reliability limits of TMR implemented in a SRAM-based FPGA: Heavy ion measures vs. fault injection predictions. In *Test Workshop (LATW), 2010 11th Latin American*, pages 1 –5, March 2010.

[Fre89] R.H. Freeman. Configurable electrical circuit having configurable logic elements and configurable interconnects, September 1989. US Patent 4,870,302.

[FRS14] M. Frischke, A.J. Rohatschek, and W. Stechele. Towards low-cost fault detection strategy of FPGA configuration memory in real-time systems. In *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*, pages 81–86, July 2014.

[GBN+11] N. Gaspard, B. L. Bhuva, B. Narasimham, A. Oates, K. Patterson, N. Tam, M. Vilchis, S.-J. Wen, R. Wong, and Y.Z. Xu. Alpha particle induced soft error rates for FF designed in a 28 nm bulk CMOS process. In *3rd annual IEEE Santa Clara Valley SER Workshop*, San Jose, October 2011.

[GD11] Lingkan Gong and O. Diessel. Modeling dynamically reconfigurable systems for simulation-based functional verification. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pages 9 –16, May 2011.

[GEBT05] M.J. Gadlage, P.H. Eaton, J.M. Benedetto, and T.L. Turflinger. Comparison of heavy ion and proton induced combinatorial and sequential logic error rates in a deep submicron process. *Nuclear Science, IEEE Transactions on*, 52(6):2120–2124, 2005.

[GM13] R Omidi Gosheblagh and Karim Mohammadi. Dynamic partial based single event upset (SEU) injection platform on FPGA. *International Journal of Computer Applications*, 76(3):19 – 24, August 2013.

[GNW+06] Michael Gabrick, Rick Nicholson, Frank Winters, Bruce Young, and Jim Patton. FPGA considerations for automotive applications. In *SAE Technical Paper*, April 2006.

[HAP+07] A. Haggag, G. Anderson, S. Parihar, D. Burnett, G. Abeln, J. Higman, and M. Moosa. Understanding SRAM high-temperature-operating-life NBTI: Statistics and permanent vs recoverable damage. In *Reliability physics symposium, 2007. proceedings. 45th annual. ieee international*, pages 452 –456, April 2007.

[Har01] Reiner Hartenstein. Coarse grain reconfigurable architecture (embedded tutorial). In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, ASP-DAC '01, pages 564–570, New York, NY, USA, 2001. ACM.

[Hau08] Karsten Haug. Nachtsichtgerät, October 2008.

[HC06] Edvin Hjortland and Li Chen. Fault-tolerant FPGAs by online ECC verification and restoration. In *Region 5 Conference, 2006 IEEE*, pages 91 –93, April 2006.

[HCW08] J. Heiner, N. Collins, and M. Wirthlin. Fault tolerant ICAP controller for high-reliable internal scrubbing. In *Aerospace Conference, 2008 IEEE*, pages 1 –10, March 2008.

[Heb12] Emmanuel Hebrard. Scheduling and SAT, May 2012.

[HH12] Marijn J.H. Heule and Warren A. Hunt. Encoding applications into SAT, November 2012.

[HMS+10] M. Huebner, J. Meyer, O. Sander, L. Braun, J. Becker, J. Noguera, and R. Stewart. Fast sequential FPGA startup based on partial and dynamic reconfiguration. In *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, pages 190 –194, July 2010.

[HRSH11] R. Hartl, A.J. Rohatschek, W. Stechele, and A. Herkersdorf. Improved backwards analysis for architectural vulnerability factor estimation. In *Semiconductor Conference Dresden (SCD), 2011*, pages 1 –4, September 2011.

[HT98] G. Heiner and T. Thurner. Time-triggered architecture for safety-related distributed real-time systems in transportation systems. In *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pages 402–407, 1998.

[ICW+06] E. Ibe, S.S. Chung, ShiJie Wen, H. Yamaguchi, Y. Yahagi, H. Kameyama, S. Yamamoto, and T. Akioka. Spreading diversity in multi-cell neutron-induced upsets with device scaling. In *Custom Integrated Circuits Conference, 2006. CICC '06. IEEE*, pages 437–444, 2006.

[IEE06a] IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001). IEEE standard for verilog hardware description language. pages 0_1–560, 2006.

[IEE06b] IEEE Std 1666-2005. IEEE standard system c language reference manual. pages 0_1–423, April 2006.

[IEE09] IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002). IEEE standard VHDL language reference manual. pages c1–626, January 2009.

[ISO11a] ISO 26262-1. Road vehicles functional safety, part 1: Vocabulary, November 2011.

[ISO11b] ISO 26262-5. Road vehicles functional safety, part 5: Product development at the hardware level, November 2011.

[JTW07] S. Jovanovic, C. Tanougast, and S. Weber. A hardware preemptive multitasking mechanism based on scan-path register structure for FPGA-based reconfigurable systems. In *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, pages 358–364, August 2007.

[JW10] Jonathan M. Johnson and Michael J. Wirthlin. Voter insertion algorithms for FPGA designs using triple modular redundancy. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '10, pages 249–258, New York, NY, USA, 2010. ACM.

[KB03] Hermann Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[KCK01] Seong-Woo Kwak, Byung-Jae Choi, and Byung-Kook Kim. An optimal checkpointing-strategy for real-time control systems under transient faults. *Reliability, IEEE Transactions on*, 50(3):293–301, 2001.

[KFC06] F.L. Kastensmidt, C.K. Filho, and L. Carro. Improving reliability of SRAM-based FPGAs by inserting redundant routing. *Nuclear Science, IEEE Transactions on*, 53(4):2060 –2068, August 2006.

[KFV+11] F.L. Kastensmidt, E.C.P. Fonseca, R.G. Vaz, O.L. Goncalez, R. Chipana, and G.I. Wirth. TID in flash-based FPGA: Power supply-current rise and logic function mapping effects in propagation-delay degradation. *Nuclear Science, IEEE Transactions on*, 58(4):1927 –1934, August 2011.

[Khr14] Khronos Group. OpenCL - the open standard for parallel programming of heterogeneous systems, 2014.

[KHT07] Dirk Koch, Christian Haubelt, and Jürgen Teich. Efficient hardware checkpointing: concepts, overhead analysis, and implementation. In *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, FPGA '07, pages 188–196, New York, NY, USA, 2007. ACM.

[KK10] Israel Koren and C. Mani Krishna. *Fault-Tolerant Systems*. Elsevier Science, July 2010.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer-Verlag, zweite auflage edition, 2011.

[Kot06] Arun Kottolli. The economics of structured- and standardcell-ASIC designs. Technical Report Electronic News, March 2006.

[KPCC97] R. Koga, S.H. Penzin, K.B. Crawford, and W.R. Crain. Single event functional interrupt (SEFI) sensitivity in microcircuits. In *Radiation and Its Effects on Components and Systems, 1997. RADECS 97. Fourth European Conference on*, pages 311 –318, September 1997.

[KR09] Ian Kuon and Jonathan Rose. *Quantifying and Exploring the Gap Between FPGAs and ASICs*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[KRS13] Michael Kuhnert, Andreas J. Rohatschek, and Walter Stechele. Fast parameter-based checkpoint, rollback and synchronization prediction for FPGA designs. In *25. GI/GMM/ITG-Workshop: Testmethoden und Zuverlässigkeit von Schaltungen und Systemen*, Dresden, Germany, February 2013.

[Kru08] M. Krueger. *Grundlagen der Kraftfahrzeugelektronik: Schaltungstechnik*. Hanser Verlag, 2008.

[KSL+06] Suk Joon Kim, Poong Hyun Seong, Jun Seok Lee, Man Cheol Kim, Hyun Gook Kang, and Seung Cheol Jang. A method for evaluating fault coverage using simulated fault injection for digitalized systems in nuclear power plants. *Reliability Engineering & System Safety*, 91(5):614 – 623, 2006.

[KTR08] Ian Kuon, Russell Tessier, and Jonathan Rose. FPGA architecture: Survey and challenges. *Found. Trends Electron. Des. Autom.*, 2(2):135–253, February 2008.

[KV08] Bernhard Korte and Jens Vygen. *Kombinatorische Optimierung*. Springer, 2008.

[Lac14] Johannes Lack. *CaRLOS – Ein Framework zur Checkpoint and Rollback Evaluation, Optimization & Simulation*. PhD thesis, HOCHSCHULE PFORZHEIM, April 2014.

[Lat11] Lattice Semiconductor Corp. LatticeECP3 soft error detection (SED) usage guide. Technical Report TN1184, February 2011.

[LBN10] Uros Legat, Anton Biasizzo, and Franc Novak. Automated SEU fault emulation using partial FPGA reconfiguration. In *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, pages 24 –27, April 2010.

[LBN11] U. Legat, A. Biasizzo, and F. Novak. Self-reparable system on FPGA for single event upset recovery. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, pages 1 –6, June 2011.

[LBW+13] K. Lilja, M. Bounasser, S.-J. Wen, R. Wong, J. Holst, N. Gaspard, S. Jagannathan, D. Loveless, and B. Bhuva. Single-event performance and layout optimization of flip-flops in a 28-nm bulk technology. *Nuclear Science, IEEE Transactions on*, 60(4):2782–2788, August 2013.

[LCR03] F. Lima, L. Carro, and R. Reis. Designing fault tolerant systems into SRAM-based FPGAs. In *Design Automation Conference, 2003. Proceedings*, pages 650–655, June 2003.

[LD10] Huiming Li and Yuyuan Du. Research on the vehicle ESP system in FPGA. In *Control and Decision Conference (CCDC), 2010 Chinese*, pages 294–298, 2010.

[LDF+05] A. Lesea, S. Drimer, J.J. Fabula, C. Carmichael, and P. Alfke. The rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs. *Device and Materials Reliability, IEEE Transactions on*, 5(3):317–328, September 2005.

[Les12] Austin Lesea. Single-event effects in 28 nanometer configuration and dual-port RAM block memories. In *Silicon Errors in Logic - System Effects*, University of Illinois, March 2012.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[LOGVPGEA05] Celia Lopez-Ongil, Mario Garcia-Valderas, Marta Portela-Garcia, and Luis Entrena-Arrontes. Autonomous transient fault emulation on FPGAs for accelerating fault grading. In *Proceedings of the 11th IEEE International On-Line Testing Symposium*, pages 43–48, Washington, DC, USA, 2005. IEEE Computer Society.

[LPL+10] C. Lavin, M. Padilla, P. Lundrigan, B. Nelson, and B. Hutchings. Rapid prototyping tools for FPGA designs: RapidSmith. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 353 –356, December 2010.

[Ltd14] Real Time Engineers Ltd. FreeRTOS - market leading RTOS (real time operating system) for embedded systems with internet of things extensions, July 2014.

[Luk13a] Sean Luke. ECJ 21, a java-based evolutionary computation research system. Technical report, 2013. http://cs.gmu.edu/~eclab/projects/ecj/.

[Luk13b] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/$\sim$sean/book/metaheuristics/.

[LZF+09] Marco Lanuzza, Paolo Zicari, Fabio Frustaci, Stefania Perri, and Pasquale Corsonello. An efficient and low-cost design methodology to improve SRAM-based FPGA robustness in space and avionics applications. In *5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications*, pages 74–84, Berlin, Heidelberg, 2009. Springer-Verlag.

[Mar07] Wayne Marx. Addressing today's embedded design challenges with FPGAs. Xilinx, September 2007.

[Mat12] MathWorks, Inc. MATLAB - the language of technical computing, September 2012.

[MCG+05] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin. SEU-induced persistent error propagation in FPGAs. *Nuclear Science, IEEE Transactions on*, 52(6):2438–2445, 2005.

[Men12a] Mentor. Precision hi-rel - mentor graphics, July 2012. mentor_precision_2012.

[Men12b] Mentor Graphics Corporation. ModelSim SE user's manual, v10.1d, 2012.

[MG12] Q. Martin and A.D. George. Scrubbing optimization via availability prediction (SOAP) for reconfigurable space computing. In *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, pages 1–6, 2012.

[Mic13] Microsemi. IGLOO2 FPGA fabric user guide, 2013.

[MK11] V. Mikolasek and Hermann Kopetz. Roll-forward recovery with state estimation. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011 14th IEEE International Symposium on*, pages 179–186, 2011.

[MS13] D. May and W. Stechele. A resource-efficient probabilistic fault simulator. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–4, September 2013.

[MWE+03] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 29–, Washington, DC, USA, 2003. IEEE Computer Society.

[Naz13] G.L. Nazar. *Fine-Grained Error Detection Techniques for Fast Repair of FPGAs*. PhD thesis, Universidade Federal Do Rio Grande Do Sul, Porto Alegre, 2013.

[NMGT06] Jun Nakano, Pablo Montesinos, Kourosh Gharachorloo, and Josep Torrellas. ReViveI/o: Efficient handling of i/o in highlyavailable rollback-recovery servers. In *In HPCA*, pages 203–214, 2006.

[NR08] Jean-Baptiste Note and Éric Rannaud. From the bitstream to the netlist. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, FPGA '08, pages 264–264, New York, NY, USA, 2008. ACM.

[NSB12] M. Niknahad, O. Sander, and J. Becker. Fine grain fault tolerance- a key to high reliability for FPGAs in space. In *Aerospace Conference, 2012 IEEE*, pages 1 –10, March 2012.

[NYKB08] Takashi Nakamura, Eishi Yahagi, Hideaki Kameyama, and Mamoru Baba. *Terrestrial Neutron-Induced Soft Errors in Advanced Memory Devices*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2008.

[PDH11] Kyprianos Papadimitriou, Apostolos Dollas, and Scott Hauck. Performance of partial reconfiguration in FPGA systems: A survey and a cost model. *ACM Trans. Reconfigurable Technol. Syst.*, 4(4):36:1–36:24, December 2011.

[PM11] Daniel Platzker and Ehab Mohsen. Using FPGA synthesis to protect against radiation effects and soft errors. Technical Report Mentor Graphics Corporation, 2011.

[PMN+09] F.R. Palomo, J.M. Mogollon, J. Napoles, H. Guzman-Miranda, A.P. Vega-Leal, M.A. Aguirre, P. Moreno, C. Mendez, and J.R.V. de Aldana. Pulsed laser SEU cross section measurement using coincidence detectors. *Nuclear Science, IEEE Transactions on*, 56(4):2001 –2007, August 2009.

[PRCLF12] José Antonio Parejo, Antonio Ruiz-Cortés, Sebastián Lozano, and Pablo Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, 16(3):527–561, 2012.

[PS96] Ravi Prakash and Mukesh Singhal. Low-cost checkpointing and failure recovery in mobile computing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 7(10):1035–1048, 1996.

[Ren12] Renesas Electronics America Inc. Mastering functional safety and ISO-26262. In *DevCon*, Hyatt Regency Orange County, October 2012.

[RFW10] N. Rollins, M. Fuller, and M.J. Wirthlin. A comparison of fault-tolerant memories in SRAM-based FPGAs. In *Aerospace Conference, 2010 IEEE*, pages 1–12, 2010.

[RMR09] O. Ruano, J.A. Maestro, and P. Reviriego. A methodology for automatic insertion of selective TMR in digital circuits affected by SEUs. *Nuclear Science, IEEE Transactions on*, 56(4):2091 –2102, August 2009.

[Rob13] Robert Bosch GmbH. Generic timer module (GTM) IP product information, February 2013.

[SAA+04] Lui Sha, Tarek Abdelzaher, Karl-Erik Arzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K Mok. Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2-3):101–155, 2004.

[Sau10] Peter Sauer. *Hardware-Design mit FPGA*. Elektor-Verlag GmbH, Aachen, 2010.

[SFRB05] V. Srinivasan, J. W. Farquharson, W. H. Robinson, and B. L. Bhuva. Evaluation of error detection strategies for an FPGA-based self-checking arithmetic and logic unit. 2005.

[SGBG12] Fouad Sahraoui, Fakhreddine Ghaffari, Mohamed El Amine Benkhelifa, and Bertrand Granado. An efficient BER-based reliability method for SRAM-based FPGA. In *Proceedings of 7th IEEE International Design and Test Workshop IDT*, pages 15 – 17, Doha, Qatar, December 2012.

[SHSF11] A.G. Schmidt, Bin Huang, R. Sass, and M. French. Checkpoint/restart and beyond: Resilient high performance computing with FPGAs. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pages 162 –169, May 2011.

[SKHT06] Thilo Streichert, Dirk Koch, Christian Haubelt, and Jürgen Teich. Modeling and design of fault-tolerant and self-adaptive reconfigurable networked embedded systems. *EURASIP J. Embedded Syst.*, 2006(1):9–9, January 2006.

[SKK11] M. Straka, J. Kastil, and Z. Kotasek. SEU simulation framework for xilinx FPGA: First step towards testing fault tolerant systems. In *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pages 223 –230, September 2011.

[SKM+08] S. Srinivasan, R. Krishnan, P. Mangalagiri, Yuan Xie, V. Narayanan, M.J. Irwin, and K. Sarpatwari. Toward increasing FPGA lifetime. *Dependable and Secure Computing, IEEE Transactions on*, 5(2):115 –127, June 2008.

[Sla11] C. Slayman. Soft error trends and mitigation techniques in memory devices. In *Reliability and Maintainability Symposium (RAMS), 2011 Proceedings - Annual*, pages 1–5, 2011.

[SM99] J. M. Solana and M. A. Manzano. PASE-scan design: a new full-scan structure to reduce test application time. *Computers and Digital Techniques, IEE Proceedings* -, 146(6):283–293, 1999.

[SMHW02] D.J. Sorin, M.M.K. Martin, M.D. Hill, and D.A. Wood. SafetyNet: improving the availability of shared memory multiprocessors with global checkpoint/recovery. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 123 –134, 2002.

[Sou13] SourceTech411. Top FPGA companies for 2013, April 2013.

[SPV06] D. Schellekens, B. Preneel, and I. Verbauwhede. FPGA vendor agnostic true random number generator. In *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–6, August 2006.

[SSC08] E. Stott, P. Sedcole, and P. Cheung. Fault tolerant methods for reliability in FPGAs. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 415 –420, September 2008.

[Ste09] Luca Sterpone. Timing driven placement for fault tolerant circuits implemented on SRAM-based FPGAs. In *Proceedings of the 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications*, ARC '09, pages 85–96, Berlin, Heidelberg, 2009. Springer-Verlag.

[Sut13] Angela Sutton. How reliable is your FPGA-based system?, August 2013.

[SWC10] E. Stott, J.S.J. Wong, and P.Y.K. Cheung. Degradation analysis and mitigation in FPGAs. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 428 –433, September 2010.

[Tah06] M. Tahoori. Application-dependent testing of FPGAs. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(9):1024 –1033, September 2006.

[TK10] Simon Tam and Martin Kellermann. Fast configuration of PCI express technology through partial reconfiguration. In *XAPP883*, November 2010.

[Uni] Brigham Young University. FPGA reliability studies - BYU EDIF tools home page.

[VCG+05] F. Vargas, D.L. Cavalcante, E. Gatti, D. Prestes, and D. Lupi. On the proposition of an EMI-based fault injection approach. In *On-Line Testing Symposium, 2005. IOLTS 2005. 11th IEEE International*, pages 207 – 208, July 2005.

[WBR11] Henry Wong, Vaughn Betz, and Jonathan Rose. Comparing FPGA vs. custom cmos and the impact on processor microarchitecture. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 5–14, New York, NY, USA, 2011. ACM.

[WJW+13] Michael Wirthlin, Josh Jensen, Alex Wilson, Will Howes, Shi-Jie Wen, and Rick Wong. Placement of repair circuits for in-field FPGA repair. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, pages 115–124, New York, NY, USA, 2013. ACM.

[Wol08] Katharina Wolter. *Stochastic models for restart, rejuvenation and checkpointing.* PhD thesis, Habilitation thesis, Humboldt-University, Berlin, Germany, 2008.

[WTH13] M.J. Wirthlin, H. Takai, and A. Harding. Soft error rate estimations of the kintex-7 FPGA within the ATLAS liquid argon (LAr) calorimeter. In *TOPICAL WORKSHOP ON ELECTRONICS FOR PARTICLE PHYSICS*, PERUGIA, Italien, September 2013.

[Xil09] Xilinx, Inc. Virtex-4 FPGA configuration user guide. Technical Report UG071, June 2009.

[Xil10] Xilinx, Inc. SEU strategies for virtex-5 devices. Technical Report XAPP864, April 2010.

[Xil12a] Xilinx, Inc. MicroBlaze processor reference guide. Technical Report UG081, April 2012.

[Xil12b] Xilinx, Inc. XILINX TMRTOOL. Technical report, 2012.

[Xil13a] Xilinx, Inc. 7 series FPGAs configurable logic block. Technical Report UG474, August 2013.

[Xil13b] Xilinx, Inc. 7 series FPGAs configuration. Technical Report UG470, February 2013.

[Xil13c] Xilinx, Inc. Device reliability report. Third Quarter 2013 UG116 (v9.6), Xilinx, Inc., November 2013.

[Xil13d] Xilinx, Inc. LogiCORE IP MicroBlaze micro controller system v1.4. Technical Report PG048, March 2013.

[Xil13e] Xilinx, Inc. LogiCORE IP soft error mitigation controller v4.0. Technical Report PG036, June 2013.

[Xil13f] Xilinx, Inc. Virtex-7 t and XT FPGAs data sheet: DC and AC switching characteristics. Technical Report DS183, November 2013.

[Xil14] Xilinx, Inc. 7 series FPGAs overview. Technical Report DS180, February 2014.

[XR95] Jie Xu and Brian R. Responsive roll–forward recovery in embedded real–time systems. Technical report, In Proceedings of the 1996 International Conference on Parallel and Distributed Systems (ICPADS '96, 1995.

[YHH+13] Enshan Yang, Keheng Huang, Yu Hu, Xiaowei Li, Jian Gong, Hongjin Liu, and Bo Liu. HHC: Hierarchical hardware checkpointing to accelerate fault recovery for SRAM-based FPGAs. In *On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International*, pages 193–198, 2013.

[YM01] Shu-Yi Yu and E.J. McCluskey. On-line testing and recovery in TMR systems for real-time applications. In *Test Conference, 2001. Proceedings. International*, pages 240 –249, 2001.

[YWZ10] Wenlong Yang, Lingli Wang, and Xuegong Zhou. CRC circuit design for SRAM-based FPGA configuration bit correction. In *Solid-State and Integrated Circuit Technology (ICSICT), 2010 10th IEEE International Conference on*, pages 1660 –1664, November 2010.

Abkürzungsverzeichnis

ABS	Antiblockiersystem
ALM	Adaptive Logic Module
ASIC	Application Specific Integrated Circuit
ASIL	Automotive Safety Integrity Level
AVF	Architectural Vulnerability Factor
BIST	Build-In Self-Test
BRAM	Block Random Access Memory
CAN	Controller Area Network
CaRLOS	**C**heckpoint **a**nd **R**ollback Eva**l**uation, **O**ptimization, **S**imulation
CED	Concurrent Error Detection
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CP	Checkpoint
CPU	Central Processing Unit
CRAM	Configuration Random Access Memory
CRC	Cyclic Redundancy Check
DCR	Double Checker Redundancy
DKE	Detektions- und Korrektureinheit
DSP	Digital Signal Processor
EA	Evolutionärer Algorithmus
ECC	Error-Correcting Code
EDF	Earliest Deadline First
EDIF	Electronic Design Interchange Format

EMV Elektromagnetische Verträglichkeit

ESD Elektrostatische Entladung (engl. Electrostatic Discharge)

FDZ Fehlerdetektionszeit

FF Flip-Flop

FIFO First In - First Out

FIT Failure In Time

FKF Funktionale Konfigurations-Fehlerabdeckung

FLI Foreign Language Interface

FMEDA Failure Modes Effects and Diagnostic Analysis

FPGA Field Programmable Gate Array

FTZ Fehlertoleranzzeit

GND Masse

GPU Graphics Processing Unit

HC Hot Carrier

HDL Hardware Description Language

HLS High Level Synthese

HW Hardware

IC Integrated Circuit

ICAP Internal Configuration Access Port

IP Intellectual Property

ISO International Organization for Standardization

JNI Java Native Interface

Kfz Kraftfahrzeug

KNF Konjunktive Normalform

LEFK Linearer Essentieller Frame Konfigurationstest

LKT Linearer Konfigurationsspeicher Test

LUT Lookup Table

LVKT Linearer Vollständiger Konfigurationstest

MBU Multi-Bit Upset

MCS Microcontroller System

MCU Microprocessor Control Unit

MF Major Frame

MM Memory-Mapped

MOSFET Metall-Oxid-Halbleiter Feldeffekttransistor

MTBF Mean Time Between Failures

MTTR Mean Time To Repair

MUX Multiplexer

NBTI Negative Bias Thermal Instability

NCD Native Circuit Description

NGC Native Generic Database

NRE Non Recurring Engineering

PDR Partielle Dynamische Rekonfiguration

PIP Programmable Interconnect Point

PKT Priorisierter Konfigurationsspeicher Test

PLL Phase-Locked Loop

PROM Programmable Read Only Memory

PWM Pulsweitenmodulation

RAM Random Access Memory

Rb Rollback

ROM Read Only Memory

RTL Register Transfer Level

SAT Satisfiable

SC Scan-Chain

SEB Single Event Burnout

SECDED Single Error Correction Double Error Detection

SEE Single Event Effect

SEFI Soft Error Functional Interrupt

SEGR Single-Event Gate Rupture

SEL Single Event Latch-up

SER Soft Error Rate

SET Single Event Transient

SEU Single Event Upset

SHC Shadow Scan-Chain

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

TDDB Time-Dependent Dielectric Breakdown

TMR Triple Modular Redundancy

TPG Test Pattern Generator

TRNG True Random Number Generator

TSMC Taiwan Semiconductor Manufacturing Company

UUT Unit Under Test

VHDL Very High Speed Integrated Circuit Hardware Description Language, VHSIC Hardware Description Language

VLSI Very Large Scale Integration

XDL Xilinx Design Language

Erklärung

Ich erkläre an Eides statt, dass ich diese der Fakultät für Elektrotechnik und Informationstechnik am Lehrstuhl für Integrierte Systeme der Technischen Universität München zur Promotionsprüfung vorgelegte Arbeit selbstständig angefertigt habe.

Die Arbeit trägt den Titel „Redundanzfreie Fehlerbehandlung in echtzeitfähigen FPGA Schaltungen“ unter der Betreuung durch Prof. Dr.-Ing. Walter Stechele.

Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.

Wörtlich oder inhaltlich übernommene Stellen sind als solche gekennzeichnet.

Ich habe die Dissertation in dieser oder ähnlicher Form in keinem anderen Prüfungsverfahren als Prüfungsleistung vorgelegt.

Die Promotionsordnung der Technischen Universität München vom 01.08.2001 in der Fassung der Neunten Änderungssatzung vom 02.08.2010 ist mir bekannt.

München, 6. Januar 2016 Michael Frischke

Lebenslauf

Persönliche Daten

Vorname:	Michael
Nachname:	Frischke (geb. Kuhnert)
Geburtsdatum:	17. Januar 1983
Geburtsort:	Bad Muskau
E-Mail:	michael.frischke@gmail.com

Akademische Bildung

2011 - 2015	Promotion an der Technischen Universität München (Lehrstuhl für Integrierte Systeme)
2002 - 2008	Studium der Elektrotechnik an der Brandenburgischen Technischen Universität Cottbus, Vertiefungsrichtung: Mikroelektronik und Mikrosystemtechnik Diplomarbeit: Optimierung der Testmethodik zur Messung des Einflusses von Plasmaprozessen auf die Zuverlässigkeit von MOS – Transistoren
1995 - 2001	Landau-Gymnasium in Weißwasser

Beruflicher Werdegang

seit 09/2014	Hardware- und System Entwicklungsingenieur bei Berlin Heart GmbH
08/2011 - 07/2014	Promotion bei Robert Bosch GmbH in der Forschungsgruppe Digitale Hardware in Schwieberdingen
03/2008 - 07/2011	Entwicklungsingenieur Messtechnik bei BIOTRONIK SE & Co. KG
04/2007 - 03/2008	Praktikum und Diplomarbeit bei X-FAB Dresden GmbH & Co. KG in der Abteilung Zuverlässigkeit und elektrische Analyse