

Exzellenzcluster
Cognitive Interaction Technology
Kognitronik und Sensorik
Prof. Dr.-Ing. U. Rückert

Dynamisch partielle Rekonfiguration in fehlertoleranten FPGA-Systemen

zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEUR (Dr.-Ing.)

der Technischen Fakultät
der Universität Bielefeld

genehmigte Dissertation

von

Dipl.-Ing. Sebastian Korf

Referent: Prof. Dr.-Ing. Ulrich Rückert
Korreferent: Prof. Dr. Christian Plessl

Tag der mündlichen Prüfung: 14.09.2017

Bielefeld / September 2017
DISS KS / 13

Gedruckt auf alterungsbeständigem Papier °° ISO 9706

Danksagung

An dieser Stelle möchte ich mich bei allen Personen herzlich bedanken, die mich während meiner Arbeit auf unterschiedlichste Weise unterstützt und zu meiner Dissertation beigetragen haben. Besonderer Dank gilt Prof. Dr.-Ing. Ulrich Rückert, der mir diese Dissertation ermöglicht und mich bereits als Studentische Aushilfskraft in Paderborn in seinem Fachgebiet Schaltungstechnik aufgenommen hat. Ein großer Dank gilt Dr.-Ing. Mario Pormann, der mich zum Schritt der Promotion ermutigt hat und stets eine offene Tür und ein offenes Ohr für unzählige, wertvolle Diskussionen hatte. Insbesondere möchte ich mich für die konstruktive Rückmeldung zu dieser Dissertation bedanken. Ebenfalls möchte ich mich bei Prof. Dr. Christian Plessl für die Begutachtung dieser Dissertation bedanken.

Ein besonderer Dank gilt auch Jens Hagemeyer, der mich bereits seit der Zeit als Studentische Aushilfskraft betreut und anschließend über viele Jahre begleitet und geprägt hat. Ebenfalls möchte ich mich bei Dario Cozzi für die intensive Zusammenarbeit seit unserer Master-/Diplomarbeitszeit bedanken. Gemeinsam haben wir die Herausforderungen bei der Entwicklung von DHHarMa, INDRA 2.0 und OLT(RE)² angenommen und gemeistert.

Weiterer Dank gilt allen Mitarbeitern des Fachgebiets Schaltungstechnik und der Arbeitsgruppe Kognitronik und Sensorik für das gemeinschaftliche Arbeitsklima. Hierbei möchte ich mich insbesondere bei meinen beiden Studien-, anschließenden Arbeitskollegen und Freunden Martin Kaiser und Dirk Jungewelter für die vielen gemeinsamen Jahre mit fachlichen und nicht-fachlichen Diskussionen bedanken.

Zuletzt möchte ich mich bei meiner Familie für den starken Rückhalt und die Unterstützung bedanken. Der größte Dank geht an meine Frau, die mich seit meiner Jugend begleitet, anspornt und fördert sowie an meine beiden wundervollen Kinder, Lea und Ida, die mich stets auf Gedanken fernab von rekonfigurierbarer Hardware bringen. Euch widme ich diese Dissertation und bedanke mich für eure Liebe, Geduld und Nachsichtigkeit während der letzten Jahre.

Kurzfassung

Die Anforderungen an mikroelektronische Systeme steigen kontinuierlich. Rekonfigurierbare Architekturen bieten einen Kompromiss zwischen der Leistungsfähigkeit anwendungsspezifischer Schaltungen (ASICs) und der Flexibilität heutiger Prozessoren. Sogenannte im *Feld programmierbare Gatter-Arrays* (engl. **Field-Programmable Gate Arrays**, FPGAs) haben sich hierbei in den letzten Jahrzehnten besonders etabliert. Die Konfigurationsart dynamisch partielle Rekonfiguration (DPR) moderner SRAM-basierter FPGAs verdeutlicht die gewonnene System-Flexibilität. DPR wird in verschiedensten Anwendungsgebieten aus unterschiedlichsten Motivationen heraus eingesetzt.

Die Hauptanwendung der DPR ist die Erstellung eines Systems, welches Veränderungen an der Schaltung auf dem FPGA zur Laufzeit erlaubt. Obwohl viele FPGA-Familien bereits seit zwei Jahrzehnten DPR hardwareseitig ermöglichen, ist die Unterstützung durch die Hersteller-Software und insbesondere die Eigenschaften des daraus resultierenden DPR-Systems verbesserungswürdig. Um das Potenzial der verfügbaren Hardware-Flexibilität ausnutzen zu können, wird in dieser Dissertation ein neuer Entwurfsablauf (*INDRA 2.0, INtegrated Design Flow for Reconfigurable Architectures*) vorgestellt. Dieser Entwurfsablauf ermöglicht die Erstellung eines flexiblen DPR-Systems mit geringem Speicher-, Verwaltungs- und Wartungsaufwand.

Für Anwendungen mit Homogenitätsanforderungen wird mit *DHHarMa (Design Flow for Homogeneous Hard Macros)* ein Entwurfsablauf vorgestellt, der die Transformation eines zunächst inhomogenen Designs in ein homogenes Design ermöglicht. Bei dieser Design-Homogenisierung ergibt sich die Fragestellung nach möglichen Auswirkungen bezüglich des FPGA-Ressourcenbedarfs und der Leistungsfähigkeit durch die einzelnen Homogenisierungsschritte. Die einzelnen DHHarMa-Softwarekomponenten wurden daher detailliert durch verschiedene Bewertungsmaße analysiert. Hierbei konnte festgehalten werden, dass die Homogenisierungsschritte im Mittel einen, teils deutlichen, positiven Einfluss auf den FPGA-Ressourcenbedarf jedoch teils einen geringen negativen Einfluss auf die Leistungsfähigkeit hat. Die verwendete FPGA-Architektur hat hierbei auf beide Größen einen entscheidenden Einfluss.

Zusätzlich wird in Anwendungsgebieten mit Strahlungseinfluss die DPR-Funktionalität in Verfahren zur Abschwächung von durch Strahlung induzierten Fehlern eingesetzt. In der Dissertation wird mit der *Readback Scrubbing Unit* eine Komponente vorgestellt, welche eine Einbitfehlerkorrektur und Zweibitfehlererkennung im FPGA-Konfigurationsspeicher implementiert. Durch integrierte Fehlerstatistikmechanismen wird eine Analyse des Systems zur Laufzeit realisiert. Zusätzlich ist die Erstellung von Readback Scrubbing Schedules möglich, sodass die Fehlererkennung und -korrektur zum einen autonom und zum anderen zur Laufzeit angepasst werden kann. Zusätzlich wird mit *OLT(RE)² (On-Line on-demand Testing approach for permanent Radiation Effects in REconfigurable systems)* ein Selbsttest für ein SRAM-basiertes FPGA vorgestellt. Dieser Selbsttest ermöglicht zur Systemlaufzeit eine Überprüfung einer FPGA-Fläche vor der Verwendung auf permanente Fehler in den Verdrahtungsressourcen.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Inhaltlicher/Thematischer Überblick	2
1.2	Motivation/Ziele	3
1.2.1	Erstellung eines flexiblen, feingranularen DPR-Systems	3
1.2.2	Erzeugung homogener Designstrukturen für FPGAs	3
1.2.3	Erhöhung der Fehlertoleranz eines FPGA-basierten Systems	3
1.3	Aufbau der Arbeit	4
2	Grundlagen zu FPGAs	7
2.1	Einordnung in und Vergleich mit bekannten Bauteilgruppen	7
2.1.1	Einordnung in bekannte Bauteilgruppen	7
2.1.2	Vergleich von FPGA und ASIC	10
2.1.3	Vergleich von FPGA und Prozessor	11
2.2	Struktureller Aufbau eines FPGAs	11
2.2.1	Basisblöcke eines FPGAs	12
2.2.2	Verdrahtungsinfrastruktur	14
2.3	Konfigurationsspeicher eines FPGAs	15
2.4	Konfigurationsarten eines FPGAs	18
2.5	Anwendungsgebiete	19
3	Dynamisch partielle Rekonfiguration von FPGAs	21
3.1	Einführung	21
3.1.1	Vor- und Nachteile	22
3.1.2	Analyse und Anwendungen	24
3.2	Xilinx FPGA-Familien	31
3.2.1	Virtex-4	31
3.2.2	Virtex-5	33
3.2.3	Virtex-6	35
3.2.4	Spartan-6	36
3.2.5	7-Serie und Zynq-7000	37
3.2.6	Gegenüberstellung der FPGA-Familien	39
3.3	Konfiguration eines Xilinx FPGAs	42
3.3.1	Konfigurationsspeicher	42
3.3.2	Konfigurationsschnittstellen	46
3.3.3	Konfigurationsablauf	49

3.3.4	Dynamisch partielle Rekonfiguration eines Xilinx FPGAs . . .	51
3.4	Kommunikationsinfrastrukturen in DPR-Systemen	56
3.4.1	Eigenschaften der Kommunikationsinfrastruktur	56
3.4.2	Typen der Kommunikationsinfrastruktur	57
3.5	Grob- und feingranulare DPR-Systeme	62
3.5.1	Rekonfigurationsarten	63
3.5.2	Xilinx	63
3.5.3	ReCoBus und GoAhead	67
3.5.4	OpenPR	71
3.5.5	DREAMS	73
3.5.6	INDRA	75
3.5.7	INDRA 2.0	77
3.5.8	Vergleich der Ansätze	87
3.6	Zusammenfassung	90
4	DHHarMa: Automatisierte Erzeugung homogener Strukturen für FPGAs	91
4.1	Anwendungsgebiete mit homogenen Designanforderungen	91
4.1.1	DPR-Kommunikationsinfrastrukturen	92
4.1.2	Time-to-Digital-Converter	92
4.2	Entwurfsablauf	93
4.2.1	Xilinx-basiertes Frontend	94
4.2.2	DHHarMa Backend	95
4.3	Xilinx Design Language	96
4.3.1	XDL-Report	97
4.3.2	XDL-Design-Datei	101
4.4	HDL-Bibliothek für DPR-Kommunikationsinfrastrukturen	104
4.4.1	Kommunikationsprotokoll Encapsulated Wishbone Bus	104
4.4.2	Ein- und zweidimensionale Kommunikationsinfrastrukturen	106
4.4.3	Komponenten der Bibliothek	108
4.4.4	DPR-Systempartitionierungen	111
4.4.5	FPGA-Ressourcenbedarf	114
4.5	Softwarekomponenten des DHHarMa Backends	117
4.5.1	Xilinx FPGA-Datenbanken	118
4.5.2	Parser	123
4.5.3	Packer	125
4.5.4	Placer	129
4.5.5	Router	134
4.5.6	XDL Writer	136
4.6	Auswertung	138
4.6.1	FPGA-Datenbanken	138
4.6.2	Platzierungsalgorithmen	145
4.6.3	DPR-Kommunikationsinfrastrukturen	148

4.6.4	Time-to-Digital-Converter	163
4.7	Zusammenfassung	166
5	Einsatz von FPGAs in Umgebungen mit Strahlung	169
5.1	Strahlung in unterschiedlichen Umgebungen	169
5.1.1	Kosmische Strahlung	170
5.1.2	Terrestrische Strahlung	171
5.2	Eigenschaften von Strahlung	172
5.3	Physikalische Effekte durch Strahlung	174
5.4	Kategorisierung der physikalischen Strahlungseffekte	177
5.4.1	Single-Event Effects (SEEs)	177
5.4.2	Kumulative Effekte	180
5.5	Auswirkung der Strahlungseffekte auf FPGAs	180
5.5.1	SEU/MCU/MBU Effekte	181
5.5.2	SET Effekte	183
5.5.3	SEL und SEGR Effekte	184
5.5.4	Kumulative Effekte	184
5.6	Abschwächungsverfahren zur Behandlung der Strahlungseffekte	186
5.6.1	Fehlermodelle	186
5.6.2	Redundanzverfahren	188
5.6.3	Fehlererkennung und -korrektur	193
5.6.4	Speicher-Scrubbing	194
5.7	Anwendung und Analyse der Abschwächungsverfahren	194
5.7.1	Zuverlässigkeitsmetriken	194
5.7.2	Strahlungsabschätzung	196
5.7.3	SRAM-basierte FPGAs	197
5.7.4	Flash- und Antifuse-basierte FPGAs	207
5.8	Strahlungstolerante und -gehärtete FPGAs	209
5.8.1	Xilinx	210
5.8.2	Atmel	213
5.8.3	Microsemi	214
5.9	Anwendungsbereiche	216
5.9.1	Terrestrische Anwendungsbereiche	217
5.9.2	Kosmische Anwendungsbereiche	226
5.10	Zusammenfassung	235
6	Evaluation der Ziele	239
6.1	Demonstrator für dynamisch rekonfigurierbare Verarbeitungsmodule	240
6.1.1	Übersicht über die DRPM-Hardware	241
6.1.2	Systemarchitektur des DRPMs	244
6.1.3	Verarbeitungsmodul PR-FPGA	245
6.1.4	Kommunikationsmodul DB-SPACE	248

6.1.5	Systeminitialisierung	249
6.1.6	Systemvarianten des DRPMs	250
6.2	Realisierung des INDRA 2.0 DPR-Systems	251
6.2.1	Externer Rekonfigurationskontroller	252
6.2.2	Interner Rekonfigurationskontroller	255
6.2.3	Multi-Port Memory Controller	264
6.2.4	Kommunikation im DPR-System des PR-FPGAs	267
6.3	PR-Module	277
6.3.1	Fast Fourier Transform	279
6.3.2	Finite Impulse Response	287
6.3.3	Bildfilter	290
6.4	Verfahren zur Erhöhung der Fehlertoleranz	292
6.4.1	Behandlung von Single-Event Effects	293
6.4.2	Behandlung von permanenten Fehlern	301
6.5	Zusammenfassung	316
7	Zusammenfassung und Ausblick	319
7.1	Ausblick	322
	Abbildungsverzeichnis	325
	Tabellenverzeichnis	331
	Abkürzungsverzeichnis	337
	Referenzen	343
	Eigene Veröffentlichungen	361
	Betreute Arbeiten	363
	Anhang	365
A	DPR	365
A.1	Xilinx Konfigurationsdatei	365
A.2	Xilinx Konfigurationssequenzen	367
A.3	Kodierung des Konfigurationsmodus	370
B	DHHarMa	371
B.1	Xilinx Vivado	371
B.2	DXF	375
B.3	Simulated Annealing	376
B.4	HDL-Bibliothek für Kommunikationsinfrastrukturen	377
B.5	Auswertung	386

C	Evaluation	390
C.1	Datenübertragungsraten SHRC	390
C.2	Initiale Konfiguration des SpaceWire-RTCs	393
C.3	PR-Module	394
C.4	OLT(RE) ²	395

1 Einleitung

Die Anforderungen an mikroelektronische Systeme hinsichtlich der Leistungsfähigkeit und der Flexibilität steigen kontinuierlich. Parallel hierzu soll die Energieeffizienz dieser Systeme stetig gesteigert werden. Zur Realisierung dieser Systemeigenschaften sind rekonfigurierbare Architekturen zu einer weit verbreiteten Implementierungsplattform für digitale Schaltungen in vielen verschiedenen Anwendungsgebieten geworden. Der Einsatz einer rekonfigurierbaren Architektur hat sich in den letzten Jahrzehnten besonders etabliert: Sogenannte im *Feld programmierbare Gatter-Arrays* (engl. **Field-Programmable Gate Array**, FPGA). Der globale Marktumfang wächst seit der Markteinführung im Jahr 1985 beständig. Konkret stieg der Marktumfang von 1,9 Mrd \$ im Jahr 2005 auf 2,7 Mrd \$ im Jahr 2010 und weiter auf 5,4 Mrd \$ im Jahr 2013 [72]. Bis zum Jahr 2020 wird eine durchschnittliche Wachstumsrate von 9,1 %, resultierend in einem Marktumfang von 9,9 Mrd \$, prognostiziert. Den größten Marktanteil teilen sich die FPGA-Hersteller Xilinx (47 %) und Altera (41 %) (Stand: Jahr 2013, [162]).

Die (grobe) Architektur eines FPGAs besteht aus konfigurierbaren Datenverarbeitungsblöcken, welche durch programmierbare Verbindungselemente über Signalleitungen miteinander verbunden werden können. Je nach Typ des Datenverarbeitungsblocks kann dieser etwa boolesche Gleichungen durch programmierbare Tabellen abbilden oder Multiplikationen durch digitale Signalprozessoren durchführen. Durch die Verschaltung mehrerer Datenverarbeitungsblöcke können beliebig komplexe Schaltungen implementiert werden. Insbesondere parallelisierbare Anwendungen lassen sich auf FPGAs gut abbilden und führen in geringen Stückzahlen, im Vergleich zu anderen (Hardware-) Lösungen, kostengünstig zu einer energieeffizienten, performanten Systemimplementierung bei kurzen Implementierungszeiten. Die FPGA-Architektur hat sich hierbei stufenweise von homogenen Architekturen mit identischen Blöcken hin zu heterogenen Architekturen mit unterschiedlichen Blocktypen entwickelt. Die grundlegende Struktur eines FPGAs ist jedoch immer noch regulär und stellenweise homogen. Diese Eigenschaft der Regelmäßigkeit ist für einige Anwendungsgebiete essenziell. War der Einsatz von FPGAs ursprünglich dem Zweck als sogenannte *Glue Logic* (Verbindung zweier zunächst inkompatibler Hardwarekomponenten) geschuldet, wird heutzutage die Verarbeitungsgeschwindigkeit von FPGAs benötigt, um die stetig anwachsenden Datenmengen verarbeiten zu können.

1.1 Inhaltlicher/Thematischer Überblick

Aktuelle FPGAs erlauben eine Änderung an der Konfiguration eines FPGAs zur Laufzeit. Somit wird die Verwendung der verfügbaren FPGA-Ressourcen im Zeitmultiplex-Betrieb (engl. **Time Division Multiplexing**, TDM) ermöglicht und eine Art *virtuelle Hardware* aufgebaut, in der nur gerade benötigte Logik auf das FPGA konfiguriert wird. Systeme mit einer Aufteilung in statische (stets verfügbare) und dynamische (nur bei Bedarf konfigurierte) Teile werden als *dynamisch partiell rekonfigurierbare* (DPR-) Systeme bezeichnet. Die Überführung eines statischen Systems in ein DPR-System kann zu Produktionskosteneinsparungen (z. B. durch die Verwendung eines kleineren FPGAs), der Reduzierung der Leistungsaufnahme und der Steigerung der System-Flexibilität (z. B. nachträgliche Designänderungen im laufenden Betrieb) zum Vorteil haben. Diese Vorteile werden unter anderem durch einen komplexeren Systementwurf erkauft. Die Implementierung eines flexiblen DPR-Systems, in dem dynamische Teilkomponenten an verschiedene Positionen mit gleicher Ressourcenanordnung platziert werden können (Umplatzierung), erfordert eine homogene Kommunikationsinfrastruktur zwischen den statischen und dynamischen Teilen.

Ein weiteres Anwendungsgebiet, in dem homogene Strukturen Grundlage der Realisierung sind, stellen sogenannte *Time-to-Digital-Converter* (TDC) dar. Die Regelmäßigkeit der Verdrahtungsinfrastruktur wird hierbei zur Messung von Laufzeitunterschieden zwischen Signalen mit einer Auflösung von wenigen Pikosekunden genutzt.

Die Eigenschaft der Rekonfigurierbarkeit von FPGAs wird sich zunehmend auch in Anwendungsgebieten mit hoch-energetischer/kosmischer Strahlung zunutze gemacht. Strahlung führt, je nach Art des Konfigurationsspeichers eines FPGAs, zu unterschiedlichen Anfälligkeiten. Auswirkungen einzelner Partikel werden als *Single Event Effects* (SEEs) klassifiziert und zum Begriff *Soft Error* zusammengefasst. Der Einfluss der Strahlung kann auf Dauer zu permanenten Fehlern, sogenannten *Hard Errors*, führen. Zur Behandlung der durch Strahlung induzierten Fehler können bei SRAM-basierten FPGAs DPR-basierte (Abschwächungs-) Verfahren eingesetzt und somit die Fehlertoleranz des Gesamtsystems erhöht werden. Ein Beispiel für solch ein SEE-Abschwächungsverfahren ist das sogenannte *Readback Scrubbing*, welches den Konfigurationsspeicher eines FPGAs überprüft und einen Fehler durch DPR korrigiert. In FPGA-basierten Weltraummissionen werden in der Regel speziell geschützte FPGAs eingesetzt. Insbesondere in (Klein-) Satelliten schreitet jedoch der Einsatz performanterer und deutlich kostengünstigerer Standard-FPGAs, sogenannter *Commercial Off-The-Shelf* (COTS) FPGAs, voran. Neben der Art Selbstheilung eines fehlerhaften Konfigurationsspeichers durch Scrubbing, wird auch die Implementierung als DPR-System vermehrt eingesetzt. Die aufgeführten Vorteile der DPR-Funktionalität – Produktionskosteneinsparung, Reduzierung der Leistungsaufnahme und Steigerung der System-Flexibilität – werden anhand des Anwendungsgebiets Weltraum besonders verdeutlicht.

1.2 Motivation/Ziele

Der dargestellte Überblick skizziert die Bereiche der Ziele dieser Dissertation. In den folgenden Abschnitten werden die Einzelziele konkretisiert. Die DPR-Funktionalität SRAM-basierter FPGAs steht hierbei in allen Einzelzielen im Fokus.

1.2.1 Erstellung eines flexiblen, feingranularen DPR-Systems

Obwohl viele FPGA-Familien bereits seit zwei Jahrzehnten DPR hardwareseitig ermöglichen, ist die Unterstützung durch die (Hersteller-) Software und insbesondere die Eigenschaften des daraus resultierenden DPR-Systems verbesserungswürdig. Um das Potenzial der verfügbaren Hardware-Flexibilität auszunutzen zu können, muss ein eigener Entwurfsablauf konzipiert und realisiert werden. Primäre Aufgaben hierbei sind die Erstellung einer homogenen Kommunikationsinfrastruktur zwischen den statischen und dynamischen Teilbereichen eines DPR-Systems sowie ein entkoppelter Entwurfsablauf, sodass ein flexibles DPR-System mit Modulumpplatzierung erstellt werden kann.

1.2.2 Erzeugung homogener Designstrukturen für FPGAs

Die Homogenität eines Designs kann in Bezug auf die Packung, Platzierung und Verdrahtung durch die Werkzeuge der Hersteller nicht parametrisiert werden. Aus diesem Grund muss für Anwendungsgebiete mit Homogenitätsanforderungen die Erzeugung homogener Designstrukturen durch einen eigens konzipierten Entwurfsablauf und daraus resultierende Softwarekomponenten ermöglicht werden. Hierbei muss die anschließende Integration der homogenen Designstruktur in den Entwurfsablauf des Herstellers möglich sein. Um typische Implementierungsaufgaben im FPGA-Design – Packung, Platzierung und Verdrahtung – durchführen zu können, muss ferner eine Datenbank verfügbar sein, in welcher der konkrete Aufbau des FPGAs festgehalten wird. Bei dieser Design-Homogenisierung ergibt sich hierbei die Fragestellung nach möglichen Auswirkungen bezüglich des FPGA-Ressourcenbedarfs und der Leistungsfähigkeit durch die einzelnen Homogenisierungsschritte.

1.2.3 Erhöhung der Fehlertoleranz eines FPGA-basierten Systems

In Einsatzorten mit erhöhter Strahlung hat sich bei Verwendung SRAM-basierter FPGAs der Einsatz von Scrubbing-Verfahren als De-facto-Standard gegenüber SEEs etabliert. Der Zugriff auf den Konfigurationsspeicher erfolgt über eine interne oder externe Konfigurationsschnittstelle des FPGAs. Eine autonome, parametrierbare, hardwarebasierte Implementierung würde einerseits das Prozessorsystem entlasten und andererseits adaptive Scrubbing-Profilen (unterschiedliche Scrubbing-Raten für definierbare FPGA-Abschnitte) ermöglichen.

Die Zuverlässigkeit FPGA-basierter Systeme könnte durch weitere Abschwächungsverfahren erhöht werden. So ist die Erkennung von hardwareseitigen, permanenten Fehlern (Hard Errors) zur Laufzeit wünschenswert. In SRAM-basierten FPGA-Systemen

könnte dies durch kleine, autonome Testschaltungen erfolgen, welche durch DPR platziert werden. Durch eine anschließende Testauswertung könnte der aktuelle Zustand des FPGAs festgestellt und somit die Zuverlässigkeit des Gesamtsystems weiter erhöht werden.

1.3 Aufbau der Arbeit

In Kapitel 2 werden allgemeine Grundlagen zu FPGAs beschrieben. Hierbei erfolgt unter anderem eine Einordnung in und ein Vergleich mit bekannte(n) Bauteilgruppen. Weiterhin werden die unterschiedlichen Konfigurationsspeichertypen und Konfigurationsarten von FPGAs sowie aktuelle FPGA-Familien verschiedener Hersteller vorgestellt.

In Kapitel 3 folgt eine Übersicht über die dynamisch partielle Rekonfiguration von FPGAs. Zunächst wird eine Einführung in die Thematik gegeben und dabei mögliche Vor- und Nachteile der Verwendung von DPR beschrieben. Zur Verdeutlichung der Unterschiede zwischen den verschiedenen Konfigurationsarten, wird auf den Aufbau des Konfigurationsspeichers, die Konfigurationsschnittstellen und den allgemeinen Konfigurationsablauf von Xilinx FPGAs eingegangen. Anschließend werden verschiedene Kommunikationsinfrastrukturtypen für die Kommunikation in einem DPR-System vorgestellt. Nach diesen wichtigen Grundlagen folgt eine Übersicht über grob- und feingranulare DPR-Systeme. Zur Bewertung der einzelnen Ansätze werden Bewertungsmaße für DPR-Systeme näher beschrieben, sodass die Ansätze in einem Vergleich gegenübergestellt werden können. Mit *INDRA 2.0 (INtegrated Design Flow for Reconfigurable Architectures)* wird die Realisierung des erstgenannten Ziels vorgestellt. Der Entwurfsablauf ermöglicht die Erstellung eines flexiblen, feingranularen DPR-Systems für eine Vielzahl von FPGA-Familien.

Die benötigte Homogenität der Kommunikationsinfrastruktur für das *INDRA 2.0* DPR-System wird durch die Realisierung des zweiten Ziels ermöglicht. Mit *DHHarMa (Design Flow for Homogeneous Hard Macros)* wird in Kapitel 4 der Entwurfsablauf und die daraus resultierende Softwarekette für die automatisierte Transformation eines anfänglich inhomogenen Designs in ein homogenes Design vorgestellt. Der Entwurfsablauf unterstützt Eingangsdateien, welche durch eine HDL (engl. **H**ardware **D**escription **L**anguage) beschrieben werden. Hierdurch ist es möglich generische, hierarchische und daher wartbare Bibliotheken für Anwendungen mit Homogenitätsanforderungen zu erstellen. Zentraler Punkt von *DHHarMa* sind die Homogenisierungsschritte, welche eine homogene Verwendung der internen FPGA-Ressourcen, konkret der CLB-internen Komponenten, eine homogene Platzierung sowie Verdrahtung beinhalten. Daher wird detailliert auf die *DHHarMa*-Softwarekomponenten der einzelnen Schritte sowie die benötigten Datenbanken eingegangen und im Anschluss eine Auswertung anhand der vorgestellten Anwendungsgebiete durchgeführt. Zur Beurteilung der einzelnen Homogenisierungsschritte sowie der Datenbanken werden Bewertungsmaße für die Auswertung der Leistungsfähigkeit und des Ressourcenbedarfs verwendet.

In Kapitel 5 wird der Einsatz von FPGAs in Umgebungen mit Strahlung betrachtet. Zu Beginn des Kapitels wird auf die Herkunft, Eigenschaften und resultierenden Effekte von Strahlung eingegangen. Basierend auf diesen Grundlagen erfolgt nach einer Kategorisierung der Strahlungseffekte die Beschreibung der Auswirkungen dieser auf unterschiedliche FPGA-Technologien. Wie zuvor erwähnt, erfolgt die Erhöhung der Zuverlässigkeit eines FPGA-basierten Systems über Abschwächungsverfahren. Zur Einordnung der existierenden Abschwächungsverfahren werden bekannte Fehlermodelle und Zuverlässigkeitsmetriken vorgestellt. Im Anschluss erfolgt die Anwendung und Analyse der Abschwächungsverfahren mit Fokus auf SRAM-basierte FPGAs. In Umgebungen mit Strahlung werden in der Regel strahlungstolerante oder -gehärtete FPGAs verwendet. Die aktuell verfügbaren FPGAs dieser Kategorie werden anschließend vorgestellt. Zusätzlich wird auf die Entwicklung der Strahlungstoleranz von COTS-FPGAs eingegangen. Abschließend werden FPGA-basierte Experimente aus terrestrischen und kosmischen Anwendungsbereichen unter Betrachtung der verwendeten Abschwächungsverfahren näher beschrieben. Die aufgeführten Experimente zeigen den Trend zu fehlertoleranten DPR-Systemen auf und motivieren die Anpassung bzw. Hinzunahme der im dritten Ziel definierten Abschwächungsverfahren.

In Kapitel 6 werden die realisierten Ziele der Dissertation auf dem, von der Europäischen Weltraumorganisation (engl. **European Space Agency**, ESA) geförderten *Demonstrator für dynamisch rekonfigurierbare Verarbeitungsmodule* (engl. **Dynamically Reconfigurable Processing Module demonstrator**, DRPM) evaluiert. Zunächst wird zum Verständnis des Demonstrators die Hardware und die skalierbare Systemarchitektur des DRPMs vorgestellt. Im Anschluss wird auf die Realisierung eines flexiblen DPR-Systems eingegangen und die wesentlichen Komponenten des INDRA 2.0 DPR-Systems inklusive der DPR-Kommunikationsinfrastruktur beschrieben. Weiterhin werden drei verschiedene Anwendungstypen – *Finite Impulse Response* (FIR), *Fast Fourier Transform* (FFT) und Bildfilter – für die dynamisch rekonfigurierbaren Verarbeitungsmodule des DPR-Systems vorgestellt. Darauf folgend wird auf die entwickelten Abschwächungsverfahren gegenüber SEEs im Konfigurationsspeicher und permanenten Fehlern in den Verdrahtungsressourcen eines FPGAs, das dritte Ziel dieser Dissertation, eingegangen.

In Kapitel 7 werden die Ergebnisse dieser Dissertation zusammengefasst. Abschließend wird ein Ausblick für die vorgestellten Entwurfsabläufe und Abschwächungsverfahren gegeben.

2 Grundlagen zu FPGAs

Dieses Kapitel beschreibt die Grundlagen eines im Feld programmierbaren Gatter-Arrays (engl. **Field-Programmable Gate Array**, FPGA). Ein FPGA ist ein bereits gefertigter integrierter Schaltkreis (engl. **Integrated Circuit**, IC), welcher eine Matrix verschiedener, konfigurierbarer Blöcke enthält. Der Begriff *Programmierung* eines FPGAs bezieht sich auf die Programmierung der internen Verschaltung und der Konfiguration dieser internen Blöcke, weshalb auch von der Konfiguration eines FPGAs gesprochen wird. Eine Konfiguration wird nach der Bauteilfertigung erstellt und kann beliebige logische Schaltungen abbilden. Durch diese Flexibilität finden FPGAs eine starke Verbreitung und erschließen stetig weitere Anwendungsgebiete.

Im ersten Unterkapitel erfolgt eine Einordnung und ein Vergleich mit bekannten Bauteilgruppen, um die Vor- und Nachteile der Verwendung eines FPGAs aufzuführen zu können. Der strukturelle Aufbau eines FPGAs wird in dem folgenden Unterkapitel beschrieben. FPGAs kategorisieren sich zum einen durch die Verwendung von unterschiedlichen Konfigurationsspeichertypen, sodass auf die Unterschiede kurz eingegangen und eine Übersicht über aktuelle FPGA-Familien verschiedener FPGA-Hersteller gegeben wird. Zum anderen existieren, bedingt durch den Konfigurationsspeichertyp, unterschiedliche Konfigurationsarten des FPGAs. Die einzelnen Konfigurationsarten werden anhand eines Modells beschrieben und somit die, für diese Dissertation wesentliche Konfigurationsart, dynamisch partielle Rekonfiguration, definiert. Abschließend werden exemplarisch einige Anwendungsgebiete von FPGAs aufgeführt.

2.1 Einordnung in und Vergleich mit bekannten Bauteilgruppen

FPGAs gehören aufgrund des Aufbaus und der Struktur zu der Bauteilgruppe der programmierbaren, mikroelektronischen Bauteile. Diese Bauteilgruppe wird in dem ersten Unterkapitel vorgestellt. Anschließend erfolgt ein Vergleich mit anderen Bauteilen, um eine allgemeine Einordnung geben zu können.

2.1.1 Einordnung in bekannte Bauteilgruppen

Mikroelektronischen Bauteile oder auch integrierte Schaltkreise werden in der Literatur in Gruppen unterteilt. Die Kategorisierung und Beschreibung der Bauteilgruppen in dieser Dissertation wird in Abbildung 2.1 dargestellt und beruht auf der Einteilung der Fachgruppe Kognitronik und Sensorik (Lehrveranstaltung *Rekonfigurierbare Parallele*

Rechnerarchitekturen [134]). Eine alternative, leicht abgewandelte Einteilung findet sich zum Beispiel in [97, S. 670 ff.] oder [95, S. 199 ff.].

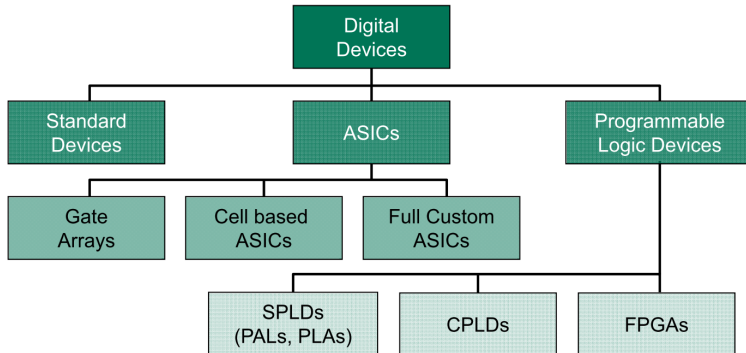


Abbildung 2.1: Kategorisierung der Bauteilgruppen basierend auf der Einteilung der Fachgruppe Kognitronik und Sensorik

Standardbauteile

Standardbauteile (*Standard Devices*) umfassen integrierte Schaltkreise wie z. B. Prozessoren oder Speicherbauteile.

Anwendungsspezifische Schaltungen

Anwendungsspezifische Schaltungen (engl. *Application-Specific Integrated Circuits, ASICs*) machen ein Produkt klein, zuverlässig (weniger Fläche und Anschlusspins), schwer kopierbar und energieeffizient, da nur wirklich benötigte Funktionen implementiert werden. Bei hohen Stückzahlen sind ASICs zudem kostengünstig. ASICs unterteilen sich in die Unterkategorien *Full Custom*, *Cell Based* und *Gate Arrays*, wobei die beiden letztgenannten Gruppen oftmals durch die Unterkategorie *Semi Custom* zusammengefasst werden.

Bei einem Full Custom ASIC werden die benötigten Schritte Zellenentwurf, Zellenplatzierung und Zellenverbindung durch den Anwender durchgeführt. Dies hat den Vorteil, dass die zur Verfügung stehende Technologie vollständig genutzt werden kann um eine optimale Implementierung zu erhalten. Nachteilig ist der hohe Entwicklungsaufwand, die aufwendige Fehlerbehebung und in der Regel ein vollständiger Neuentwurf bei dem Wechsel zu einer neuen Technologie. Aufgrund dieser schwerwiegenden Nachteile, erfolgt lediglich in Sonderfällen ein Full Custom ASIC Entwurf.

Bei einem Semi Custom ASIC wird hingegen z. B. beim Zellenentwurf auf vorgefertigte Bibliotheken (Standardzellen bei *Cell Based ASICs*) oder Makros (Basiszellen in *Masked Gate Arrays, MGAs*) zurückgegriffen. Die Verwendung von Standardzellen bietet eine effiziente Schaltungsentwicklung, insbesondere da größere Zellen wie

RAM (engl. **R**andom-**A**ccess **M**emory), ROM (engl. **R**ead-**O**nly **M**emory) oder IP (engl. **I**ntellectual **P**roperty)-Cores vollständig fertig entwickelt und getestet sind. Im Gegensatz zu einem Full Custom ASIC ist ebenfalls ein Technologiewechsel möglich. Dennoch ist der Fertigungsaufwand insgesamt vergleichbar mit dem eines Full Custom ASICs. Bei den (Masked) Gate Array werden Makros, auch Basiszellen (*Primitive-Cells*) genannt, zum Schaltungsentwurf verwendet. Die Basiszellen bestehen dabei aus definierten Anordnungen von Transistoren. Somit sind lediglich die Verbindungen der Basiszellen mit fester Größe anwendungsspezifisch, wodurch der Entwurf effizient, jedoch zugleich eingeschränkt optimierbar ist. Ein Technologiewechsel ist weniger kosten- und zeitintensiv als bei Cell Based ASICs.

Programmierbare Logikbauteile

Programmierbare Logikbauteile (engl. **P**rogrammable **L**ogic **D**evice, PLDs) beschreiben gefertigte Bauteile, in denen eine vorgegebene Matrix aus Zellen und die Verdrahtung fest vorgegeben sind. Die Zellen und die Verbindungsstruktur sind nach der Fertigung programmierbar, sodass unterschiedliche Schaltungen auf einem solchen Bauteil realisiert werden können. Allgemein bieten PLDs einen schnellen und weitestgehend automatisierten Entwurfsablauf an und sind insbesondere für geringe Stückzahlen kostengünstig. Nachteilig ist die beschränkte Optimierbarkeit und Komplexität durch die vorgegebenen Makrozellen. In modernen Bauteilen wird jedoch durch die Integration von eingebetteten IP-Cores, wie Prozessoren, Speicherbauteilen oder Kommunikationsschnittstellen, eine hohe Komplexität erreicht. Wie in Abbildung 2.1 dargestellt, unterteilen sich PLDs in drei Unterkategorien: *Simple Programmable Logic Devices (SPLDs)*, *Complex Programmable Logic Devices (CPLDs)* und *Field-Programmable Gate Arrays (FPGAs)*.

SPLDs bilden die kleinsten, einfachsten und kostengünstigsten PLDs. Kombinatorische Logik kann dabei durch Arrays von Und- und Oder-Gattern aufgebaut werden. CPLDs kombinieren mehrere SPLDs in einem Bauteil. Die einzelnen Ausgänge der SPLDs können durch programmierbare Verbindungen entweder mit anderen SPLDs oder mit Aus- bzw. Eingängen des Bauteils verschaltet werden.

Der Hersteller Xilinx stellte im Jahr 1983 mit den *Logic Cell Arrays (LCAs)* eine neue Bauteilart vor, in der sich die Gatterfunktionen durch programmierbare Tabellen (engl. **L**ook-**U**p **T**able, LUT) in SRAM-Technik realisieren ließen [95, S. 208 f.]. Im Verlauf wurde diese Klasse von Bauteilen als *FPGA* bezeichnet. Im Vergleich zu den CPLDs ist ein deutlich höherer Verschaltungsbedarf notwendig. Statt zentraler Schaltmatrizen wird in FPGAs eine sogenannte segmentierte Verbindungsarchitektur mit horizontalen und vertikalen Verdrahtungskäufen verwendet. Durch die Kombination von LUTs und Registern sowie eingebetteten (IP-) Cores, wie Prozessoren, Speicherbauteilen oder Kommunikationsschnittstellen, erschließen moderne FPGAs eine stetig steigende Anzahl von Anwendungsgebieten. Der strukturelle Aufbau wird in Abschnitt 2.2 detailliert.

2.1.2 Vergleich von FPGA und ASIC

Zu den wesentlichen Vorteilen einer FPGA-Implementierung zählen die deutlich geringeren Entwicklungskosten im Gegensatz zu einer Implementierung als anwendungsspezifische Schaltung. Unter anderem werden für die weitere Entwicklung keine Produktionsmasken benötigt, welche mit sehr hohen Fixkosten einhergehen, da ein FPGA bereits gefertigt vorliegt. Der Entwurf als ASIC lohnt sich daher erst ab einer hohen Stückzahl. Während ein ASIC-Baustein für eine konkrete Anwendung konzipiert und gefertigt ist, kann ein FPGA eine beliebige Schaltung implementieren. Dieses Merkmal eines FPGAs, die (Hardware-) Flexibilität, bringt gleichzeitig Vor- und Nachteile.

Die Konfiguration des FPGAs erfolgt nach der Fertigung des Bausteins, wodurch sehr kurze Implementierungszeiten für ein Design ermöglicht werden. Weiterhin ist das Design und damit die Funktionalität einfach korrigier- und erweiterbar, im Gegensatz zu einer statischen, fest-verdrahteten ASIC-Implementierung. Änderungen an der Implementierung können innerhalb weniger Minuten bis Stunden realisiert und kostengünstig durchgeführt werden. Weiterhin können Änderungen spät bzw. sogar nachträglich (im Feld) vorgenommen werden.

Die Hardware-Flexibilität resultiert auf der anderen Seite in einem deutlich größeren Flächenbedarf bzw. einer geringeren Logikdichte. Der Großteil der FPGA-Fläche ist durch Speicherelemente und Konfigurationslogik belegt. In [95, S. 222 f.] erfolgt eine ausführliche, theoretische Berechnung der Transistoranzahl in einem FPGA und einem ASIC. Im Vergleich mit einem SRAM (engl. **Static Random-Access Memory**)-basierten FPGA ist die Anzahl der benötigten Transistoren bis zu fünffach höher gegenüber einer ASIC-Implementierung in gleicher Fertigungstechnologie. Durch die flexible Verschaltungsmöglichkeit, welche weitere Speicherzellen benötigt, sowie dem Umstand, dass nicht alle Elemente (z. B. LUTs und Register) des FPGAs benutzt werden, weisen ASICs gegenüber SRAM-basierten FPGAs eine um Faktor 10 höhere Logikdichte auf [95, S. 196 f.]. Aus dem erhöhten Flächenbedarf resultieren längere Verbindungsleitungen und damit höhere Lastkapazitäten, wodurch sich deutlich größere Verzögerungszeiten (Faktor 5 bis 10 im Vergleich zu einer äquivalenten ASIC-Implementierung) ergeben. Der Flächenbedarf für Flash-Speicherzellen ist bei vergleichbaren Technologien etwa um Faktor 7 geringer als eine SRAM-Zelle, sodass die genannten Nachteile bei Flash-basierten FPGAs weniger schwer wiegen. Aufgrund der geringeren Taktraten der FPGAs (bei speziellen FPGAs bis zu 1,5 GHz, typisch sind allerdings 20 bis 500 MHz) gegenüber ASICs (im einstelligen GHz-Bereich) können einige Anwendungsgebiete nicht erschlossen werden. Weiterhin sind FPGAs deutlich empfindlicher gegenüber Teilchenstrahlung und elektromagnetischen Wellen. Für den Einsatz in strahlungsreichen Gebieten/Bereichen existieren daher unterschiedliche Methoden zur Steigerung der Zuverlässigkeit eines FPGA-basierten Systems. Diese Methoden sind zentrales Thema dieser Dissertation und werden detailliert in Kapitel 5 vorgestellt.

2.1.3 Vergleich von FPGA und Prozessor

Moderne Prozessoren sind vorgefertigte Bauteile, die mit vorgefertigter Hardware komplexe Programme abarbeiten. Der wesentliche Vorteil liegt dabei in der Flexibilität. So können Algorithmen aus den verschiedensten Bereichen durch eine Vielzahl von Programmiersprachen implementiert werden. Die Entwicklungszeit ist dabei durch vorgefertigte Bibliotheken sehr kurz, was zu geringen Entwicklungskosten führt. Durch die festgelegten Hardware-Komponenten kann ein Prozessor parallelisierbare Algorithmen jedoch nicht beliebig parallel ausführen, sodass die Ausführungszeit im Vergleich mit einem FPGA um mehrere Größenordnungen größer sein kann. Wird zusätzlich die Leistungsaufnahme betrachtet, bieten FPGAs in vielen Anwendungsfällen eine deutlich effizientere Lösung als Prozessoren [8, 65].

Die Vorteile einer Symbiose von statischen und rekonfigurierbaren Hardwareteilen in einem Rechensystem wurde bereits im Jahr 1960 von Gerald Estrin erkannt und der Begriff *Reconfigurable Computing* eingeführt [68]. Dieser Begriff definiert Rechensysteme, die in ihrer Struktur programmierbar sind und somit die Flexibilität von Software mit der Leistungsfähigkeit von Hardware verknüpfen. Ein Prozessor kann dadurch Aufgaben auf einer dafür optimierten Hardware ausführen. In den letzten Jahren ist diese Kopplung von FPGA und Prozessor immer enger und effizienter geworden. Die FPGA-Hersteller bieten FPGAs mit Hard- und Softcore-Prozessor-Lösungen an. Xilinx integriert in vielen FPGA-Familien (beginnend mit der Virtex-II Pro Familie) ein bis zwei IBM PowerPC-Prozessoren. Mit der Zynq-Familie wurde zu Mehrkernprozessoren mit ARM-Architektur (Dual-Core Cortex-A9) gewechselt. Altera integriert ab den Familien Cyclone V und Arria V den gleichen ARM-MP. In der nächsten Generation des Zynqs (UltraScale+) werden zwei unterschiedliche ARM Prozessoren integriert: Zur Datenverarbeitung ein Quad-Core ARM Cortex-A53 und für Echtzeitanwendungen ein Dual-Core ARM Cortex-R5. Altera erweitert in der Familie Arria 10 erstmals die DSP-Einheiten zur Verarbeitung von Fließkommaarithmetik. Die Stratix 10-Familie integriert, analog zu dem Zynq, den Quad-Core ARM Cortex-A53. Microsemi koppelt in den SmartFusion-Familien ein FPGA mit einem deutlich langsameren aber energiesparenden ARM Cortex-M3 und zielt somit auf ein anderes Anwendungsfeld als Xilinx und Altera. Neben diesen, eher für eingebettete Systeme geeigneten, Prozessoren hat der Prozessor-Hersteller Intel mit dem Xeon E5 im Jahr 2014 einen Server-Prozessor mit einem gekoppelten Altera Stratix V FPGA angekündigt. Auf dem *Intel Developer Forum* (IDF) im Frühjahr 2016 wurde die Auslieferung von Xeon E5-2600 v4 Prozessoren mit einem angeschlossenen Arria 10 FPGA bestätigt [89].

2.2 Struktureller Aufbau eines FPGAs

Die Grundstruktur eines FPGAs besteht aus einer Matrix von spaltenweise angeordneten, konfigurierbaren Basisblöcken, die über unterschiedliche Verdrahtungsinfrastrukturen miteinander verbunden werden können. Abbildung 2.2 visualisiert den allgemeinen

Aufbau anhand eines Xilinx Virtex-II FPGAs. Dieser Aufbau ist allgemein auch auf neue Xilinx FPGA-Familien und sogar auf FPGAs verschiedener Hersteller übertragbar. Die Namensgebung einzelner Basisblöcke variiert und wird in den entsprechenden Abschnitten hervorgehoben. Da der Fokus dieser Dissertation primär auf Xilinx FPGAs liegt, wird detaillierter auf FPGAs dieses Herstellers eingegangen.

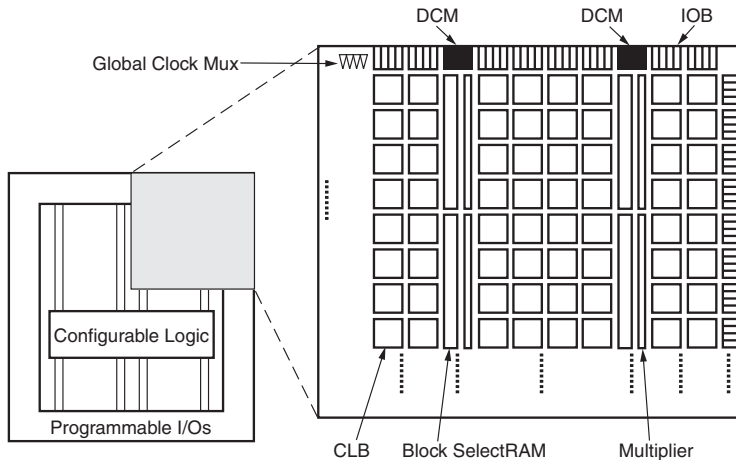


Abbildung 2.2: Anordnung der Basisblöcke eines Xilinx FPGAs [242, S. 3]

2.2.1 Basisblöcke eines FPGAs

Die Basisblöcke unterteilen sich je nach Funktion in unterschiedliche Typen. Die Funktion der wesentlichsten Basisblocktypen wird in diesem Unterkapitel beschrieben.

Konfigurierbarer Logikblock

Der konfigurierbare Logikblock ist der, für einen FPGA charakteristischste, Basisblocktyp. Er besteht mindestens aus der Kombination einer programmierbaren LUT und einem Register. Die LUT besitzt drei bis sechs Eingänge und kann durch Definition einer Wahrheitstabelle eine boolesche Funktion abbilden. Xilinx verwendet für diesen Blocktyp den Terminus *Configurable Logic Block* (CLB). Ein CLB teilt sich in eine familienspezifische Anzahl an *Slices* (Teile) auf und enthält Funktionsgeneratoren, Multiplexer, *Carry Chains* und 1-Bit-Register, welche entweder als taktgesteuertes *Flip Flop* (FF) oder zustandsgesteuertes Latch konfiguriert werden können. Eine schematische Übersicht über die genannten Komponenten innerhalb eines CLBs ist in Abbildung 2.3 dargestellt. Der Funktionsgenerator besteht aus einer LUT (*LUT G/F* in der Abbildung), welche durch Definition einer Wahrheitstabelle in den SRAM-Konfigurationszellen eine boolesche Funktion abbilden kann. Die Dimension der Wahrheitstabelle ist durch die Anzahl der LUT-Eingängen begrenzt und variiert bei den Xilinx FPGAs familienspezifisch

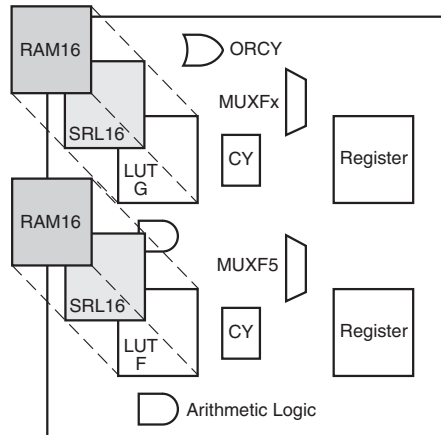


Abbildung 2.3: Allgemeine Übersicht über die CLB-internen Komponenten [242, S. 12]

zwischen vier und sechs. Alternativ können einige Funktionsgeneratoren als dedizierter Speicher (*RAM16*) oder Schieberegister (*SRL16*) verwendet werden. Die Carry Chain (*CY*) verbindet übereinanderliegende Slices direkt miteinander und ermöglicht somit eine schnelle Kommunikation zwischen benachbarten CLBs. Die einzelnen Multiplexer (*MUXFx* / *MUXF5*) mehrerer Slices können zur Realisierung größerer Multiplexer kombiniert werden. Details hierzu und ein Vergleich zwischen konkreten Xilinx FPGA-Familien werden in Abschnitt 3.2 aufgeführt. Das Pendant zu Xilinx CLBs lautet bei FPGAs des Herstellers Altera *Logic Array Block* (LAB), das Pendant zu Slices *Adaptive Logic Modules* (ALM). Die Anzahl an ALMs pro LAB variiert dabei ebenfalls familien-spezifisch. Microsemi verwendet je nach Familie unterschiedliche Begriffe (*VersaTiles*, *Logic Elements* oder *C-cells*), Atmel den Begriff *core cell*.

RAM-Block

Zusammenhängender FPGA-interner SRAM-Speicher wird bei Xilinx *Block RAM* (BRAM), oder auch Select RAM, genannt. Atmel verwendet den Begriff *FreeRAM*, Altera integriert oftmals die Größe des RAM-Blocks in den Namen, wie z. B. M9K oder M20K. Die Bezeichnung bei Microsemi variiert je nach Familie oder Größe (*Embedded RAM*, *RAM Blocks*, *LSRAM* oder *uSRAM*). Die Größe eines einzelnen RAM-Blocks beträgt wenige Kilobit, die verfügbare Gesamtspeichermenge variiert FPGA-familienintern von wenigen Kilobyte bis zu mehreren Megabyte.

Multiplizierer

Ein Multiplizierer-Basisblock (engl. *Multiplier*), in einigen FPGAs auch als Digitaler Signalprozessor (engl. *Digital Signal Processor*, DSP) oder *Mathblock* bezeichnet, ermöglicht eine schnelle Multiplikation oder Addition von Signalen.

Digitaler Taktmanager

Für die Taktgenerierung oder Taktaufbereitung steht ein weiterer Basisblock zur Verfügung. Der Digitale Taktmanager (engl. *Digital Clock Manager*, DCM) enthält *Phase-Locked Loops* (PLL) und *Delay-Locked Loops* (DLL), mit denen Takteilungen oder -vervielfachungen erzeugt werden können oder die Phase eines Taktsignals verändert werden kann.

Eingangs-/ Ausgangsblock

Der Eingangs-/ Ausgangsblock (engl. *Input Output Block*, IOB) wird für die Kommunikation mit der Umwelt benötigt. Bei älteren FPGA-Familien (Xilinx: Virtex II) sind diese Blöcke als äußerer Ring angeordnet. Bei neueren FPGA-Familien (Xilinx ab Virtex-4) wurde der Ring durch spaltenweise angeordnete Blöcke ersetzt. Dies ist insbesondere in dem stetigen Anstieg an verfügbaren Basisblöcken in einem FPGA begründet.

Einzelne FPGA-Familien unterteilen sich oftmals in Unterfamilien, welche auf spezielle Anwendungsgebiete zielen. So existieren Unterfamilien, welche über eine Vielzahl eines oder mehrerer Basisblocktypen verfügen oder weitere komplexe, fest verdrahtete Funktionsblöcke, sogenannte *Hard-IP-Cores*, integrieren. Beispiele hierfür sind (IBM PowerPC- oder ARM¹-) Prozessoren sowie (Multi-) Gigabit-Transceiver. Diese integrierten Funktionsblöcke bieten zwei Vorteile gegenüber den, durch Logikblöcken auf dem FPGA realisierten Komponenten: Zum einen sind sie wesentlich performanter und zum anderen belegen sie nur einen Bruchteil der Chipfläche.

2.2.2 Verdrahtungsinfrastruktur

Die einzelnen Basisblöcke können durch programmierbare Verbindungspunkte (engl. *Programmable Interconnect Point*, PIP) über lokale Verdrahtungsressourcen an eine benachbarte Schaltmatrix (engl. *Switch Matrix*, SM) angeschlossen werden. Abbildung 2.4 visualisiert dies für einige der genannten Basisblocktypen.

Die Schaltmatrizen eines FPGAs sind untereinander über globale Verdrahtungsressourcen verbunden. Durch die Konfiguration mehrerer programmierbarer Verbindungspunkte innerhalb der Schaltmatrix kann schließlich eine Verbindung zwischen den Basisblöcken über eine oder mehrere Schaltmatrix/-matrizen hergestellt werden. Die globalen Verdrahtungsressourcen durchziehen einen FPGA sowohl horizontal als auch vertikal und variieren in der Länge sowie der Anzahl angeschlossener Schaltmatrizen.

Anders als in dem schematischen Bild 2.2 dargestellt, belegt die Verdrahtungsinfrastruktur, insbesondere bedingt durch die flexible Verschaltungsmöglichkeiten, den

¹Acorn RISC Machine, ARM

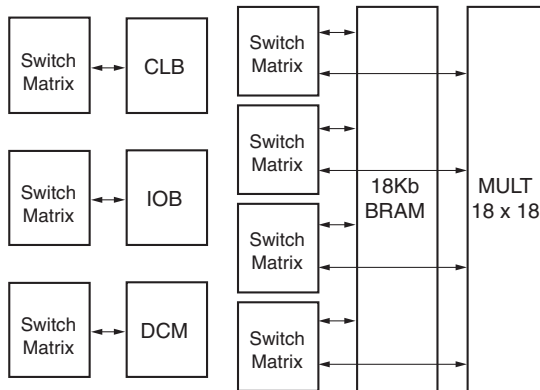


Abbildung 2.4: Anbindung der Basisblöcke an die Switch Matrix eines Xilinx FPGAs [242, S. 32]

Großteil der FPGA-Chipfläche. Laut DeHon ([51] aus dem Jahr 1996) werden ca. 89 % der Chipfläche von der Verdrahtungsinfrastruktur, 10 % von dem Konfigurationsspeicher und nur 1 % von den Logikgattern belegt. Angaben für aktuelle FPGAs sind nicht verfügbar. Durch die Verwendung von Hard-IP-Cores hat die Verdrahtungsinfrastruktur bei aktuellen FPGAs keinen nachteiligen Einfluss auf die Verarbeitungsgeschwindigkeit.

2.3 Konfigurationsspeicher eines FPGAs

Die Konfiguration und die Verschaltung der Basisblöcke eines FPGAs werden allgemein in einem Konfigurationsspeicher abgelegt. Durch unterschiedliche Speichertechnologien ergeben sich Vor- und Nachteile, die in der Literatur (z. B. [61, S. 10 ff.], [95, S. 175 ff.] oder [15, S. 14 ff.]) diskutiert und in diesem Unterkapitel zusammengefasst werden.

SRAM-basierte FPGAs

Der Großteil der existierenden FPGAs verwendet SRAM-Speicherezellen als Konfigurationsspeicher. Die Vorteile der SRAM-basierten FPGAs werden in [61, S. 9 f.] aufgeführt. Diese sind zum einen die (nahezu) unbegrenzte Anzahl an Programmierungen und zum anderen die Verwendung der Standard CMOS-Prozesstechnologie, was wiederum folgende Vorteile mit sich bringt: Takterhöhung und Senkung der dynamischen Verlustleistung bei neuen Prozesstechnologien. Die Verwendung von SRAM bringt allerdings auch Nachteile mit sich. So werden sechs Transistoren benötigt, um eine SRAM-Speicherezelle aufzubauen, was zu einem erhöhten Flächenbedarf im Vergleich mit anderen Speichertechnologien führt. Ein weiterer Nachteil ist, dass der FPGA die

Konfiguration verliert, sobald die Stromzufuhr unterbrochen wird. Die Konfigurationsdatei muss daher auf einem anderen, nicht flüchtigen Speichermedium abgelegt sein. Moderne FPGAs ermöglichen eine automatische Konfiguration aus einem externen Speicher, einige FPGAs müssen hingegen von einem dedizierten, externen Rekonfigurationskontroller konfiguriert werden.

Flash-basierte FPGAs

Darüber hinaus existieren Flash-basierte FPGAs, welche nichtflüchtigen Flash-Speicher (EEPROM²) als primären Speicherort für die Konfiguration verwenden. Dies ist zugleich der wesentliche Vorteil gegenüber der SRAM-Speichertechnologie [95, S. 196 ff.]: Ein zusätzlicher Konfigurationsspeicher ist nicht notwendig und das FPGA ist bei Einschalten der Versorgungsspannung funktional. Im englischen wird der Begriff *live at power-up* verwendet. Ein weiterer Vorteil ist der geringe Flächenbedarf einer Flash-Speicherzelle. Im Vergleich zu einer SRAM-Speicherzelle ist diese um etwa einen Faktor 7 geringer. Zusätzlich ist es bei Flash-basierten FPGAs möglich, kopiergeschützte Konfigurationsdateien in dem Flash-Speicher abzulegen. Die Nachteile der Flash-basierten FPGAs sind zum einen die begrenzte Anzahl der Programmierungen und zum anderen der aufwendige, nicht CMOS-basierte Herstellungsprozess, welcher drei zusätzliche Fertigungsschritte benötigt [15, S. 14 f.]. Weiterhin ist die statische Verlustleistung höher als bei SRAM-basierten FPGAs.

SRAM-basierte FPGAs mit zusätzlichem Flash-Speicher

Eine hybride Variante wird durch die Kombination der SRAM- und Flash-Speichertechnologie ermöglicht [1]. Der zusätzlich integrierte Flash-Speicher enthält initiale Konfigurationsdateien für den SRAM-basierten Konfigurationsspeicher. Der Nachteil eines zusätzlichen, externer Rekonfigurationskontroller entfällt somit.

Antifuse-basierte FPGAs

Antifuse-basierte FPGAs unterscheiden sich deutlich von den bisher genannten FPGA-Typen, da diese nur ein einziges Mal konfiguriert werden können. Im englischen Sprachraum wird der Begriff *One-Time Programmable* (OTP) verwendet. Der Vorteil dieser Technologie ist die, im Vergleich zu den anderen Technologien, geringe Chip-Fläche [61, S. 10 f.]. Weiterhin sind Antifuse-basierte FPGAs von Natur aus nichtflüchtig, was den Einsatz in bestimmten Einsatzgebieten mit rauen Umweltbedingungen ermöglicht. Der offensichtliche Nachteil bei der Verwendung ist, dass nachträglich keine Änderungen an der Schaltung durchgeführt werden können.

Übersicht über aktuelle FPGA-Familien

Tabelle 2.1 führt aktuelle FPGA-Familien unter Betrachtung der Konfigurationsspeicherart und Strahlungstoleranz der fünf führenden FPGA-Hersteller auf. Die Kategorisierung der Strahlungstoleranz ist vorgegriffen und wird in Abschnitt 5.8 detaillierter erläutert.

²engl. Electrically Erasable Programmable Read-Only Memory, EEPROM

Tabelle 2.1: Übersicht über aktuelle FPGA-Familien

Hersteller	FPGA-Familie	Konfigurationsspeicher
Xilinx	Spartan-3AN	SRAM+Flash
	Virtex-4 ¹ , -5 ² , -6, Spartan-6	SRAM
	7-Serie (Artix-, Kintex-, Virtex, Zynq)	SRAM
	Ultrascale (Kintex, Virtex)	SRAM
	Ultrascale+ (Kintex, Virtex, Zynq ¹)	SRAM
Altera	Stratix (I-V und 10)	SRAM
	Arria (GX, II, V, 10)	SRAM
	Cyclone (I-V)	SRAM
Microsemi	IGLOO (1-2), SmartFusion(1-2)	Flash
	(RT) ¹ ProASIC3, RTG4 ¹	Flash
	Axcelerator, SX-A, eX, MX	Antifuse
	RTAX ¹ , RTSX-SU ¹	Antifuse
Lattice Semiconductor	LatticeXP2, MACHXO2	SRAM+Flash
	ECP3/5	SRAM
Atmel	AT40K/AT40KAL/AT40KEL040 ²	SRAM
	ATF280 ²	SRAM

Legende:

¹ Strahlungstolerantes FPGA verfügbar (engl. Radiation Tolerant, RT)

² Strahlungsgehärtetes FPGA verfügbar (engl. Radiation Hardend, RH)

Eine Definition beider Begriffe erfolgt in Abschnitt 5.8.

Der FPGA-Marktführer Xilinx verwendet fast ausschließlich die SRAM-Technologie in den Virtex-, Artix-, Kintex- und den meisten Spartan-Familien. Altera bietet ausschließlich SRAM-basierte FPGAs in den Arria-, Stratix- und Cyclone-Familien an. Der Hersteller Atmel verwendet ebenfalls in allen Familien, AT40K(AL/EL) und ATF280, diese Speichertechnologie. Lattice Semiconductor produziert mit den FPGA-Familien iCE40LM, ECP3 und ECP5 ebenfalls, aber nicht ausschließlich, SRAM-basierte FPGAs. Der Fokus von Lattice Semiconductor liegt allgemein auf kleinen und besonders stromsparenden FPGAs. Xilinx vertreibt mit der älteren Spartan-3AN-Familie einen FPGA mit hybrider Speichertechnologie. Der Hersteller Lattice Semiconductor bietet mit den LatticeXP2-, iCE40- und MACHXO-Familie ebenfalls Bausteinreihen an. Microsemi, ehemals Actel, hat mehrere FPGA-Familien mit reiner Flash-Speicher-Technologie im Produktportfolio: IGLOO, SmartFusion, (RT) ProASIC3 und RTG4. In dem Bereich der Antifuse-FPGAs ist Microsemi weiterhin Marktführer und bietet für verschiedene Anwendungsgebiete unterschiedliche Familien an: Die Axcelerator-, SX-A, eX- und MX-

FPGA-Familien für Bereiche, die besonders sicherheitskritisch sind. Zusätzlich werden strahlungstolerante FPGAs, die RTAX- und RTSXA-SU-Familie, angeboten.

Im Rahmen dieser Dissertation werden insbesondere die Xilinx FPGA-Familien Virtex-4 bis 7-Serie untersucht. In dem Abschnitt 3.2 folgt daher eine detaillierte Vorstellung und Gegenüberstellung dieser FPGA-Familien.

2.4 Konfigurationsarten eines FPGAs

Lysaght et al. haben im Jahr 1993 eine Nomenklatur, basierend auf der Konfigurierbarkeit der FPGAs, vorgestellt [116]. Es erfolgt eine Aufteilung in vier Mengen, wobei eine Untermenge die Eigenschaften einer bzw. der Elternmenge(n) erbt: *Programmierbar*, *Rekonfigurierbar*, *Partiell Rekonfigurierbar* und *Dynamisch Rekonfigurierbar*. In der Abbildung 2.5 werden die ursprünglich definierten Mengen dargestellt und durch alternative Namen erweitert.

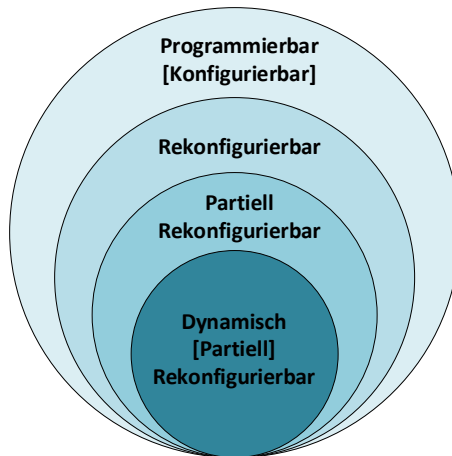


Abbildung 2.5: Klassifizierung von FPGAs auf Basis der Konfigurierbarkeit (basierend auf [116])

Programmierbar [Konfigurierbar]

Alle FPGAs sind per definitionem programmierbar und gehören somit der Menge *Programmierbar* an. Alternativ kann, zur Verwendung einer einheitlichen Bezeichnung, der Begriff *Konfigurierbar* verwendet werden.

Rekonfigurierbar

Rekonfigurierbar sind wiederum nur solche FPGAs, welche sich nach einer Konfiguration mit einer anderen Konfiguration überschreiben lassen. OTP FPGAs erfüllen diese

Anforderung nicht und gehören damit eindeutig zu der Menge der programmierbaren FPGAs.

Partiell Rekonfigurierbar

Die Klassifikation als *Partiell Rekonfigurierbar* erfüllen FPGAs, welche eine Änderung der Konfiguration eines Teils des FPGAs erlauben, während der restliche Bereich inaktiv bleibt und die Konfiguration beibehält. Die Änderung der Konfiguration wird somit erst nach einem erneuten Start des FPGAs wirksam.

Dynamisch [Partiell] Rekonfigurierbar

Dynamisch [Partiell] Rekonfigurierbar ist ein FPGA, wenn eine Änderung an der Konfiguration durchgeführt werden kann, ohne die Ausführungen im restlichen Bereich zu stören. Dies impliziert, dass der restliche Bereich aktiv ist und verschärft somit die Bedingung der zuvor definierten Menge *Partiell Rekonfigurierbar*. Um alle Eigenschaften dieser Konfigurationsart in den Begriff zu integrieren, kann dieser zu *Dynamisch Partiell Rekonfigurierbar* erweitert werden.

In der Literatur werden die Begriffe *Partielle Rekonfiguration* und *Dynamisch Partielle Rekonfiguration* oftmals synonym verwendet, obwohl beide Konfigurationsarten in modernen FPGAs eingesetzt werden. In der englischen Literatur existiert zusätzlich die Variante *run-time reconfiguration*. Die FPGA-Hersteller, welche die Konfigurationsart *Dynamisch Partielle Rekonfiguration* bereitstellen (Xilinx und Altera), verwenden die drei genannten Begriffe ebenfalls synonym. In dieser Dissertation wird die zuletzt aufgeführte Konfigurationsart *Dynamisch Partielle Rekonfiguration* betrachtet und mit DPR abgekürzt. Diese Konfigurationsart ist wesentlicher Bestandteil der Dissertation und wird im Kapitel 3 ausführlich beschrieben.

2.5 Anwendungsgebiete

Ein wesentliches Anwendungsgebiet von FPGAs ist in dem sogenannten (*Rapid*) *Prototyping*, der Verifizierung von Entwürfen neuer Bauteile. Durch das Prototyping können Fehler in einer Schaltung frühzeitig erkannt und bereits vor der kostenintensiven Bauteilfertigung behoben werden. Ein Beispiel hierfür ist die Überprüfung von Prozessoren (z. B. Intel), auch *Hardware Emulation* genannt. Durch die größeren Verzögerungszeiten gegenüber einer ASIC-Implementierung, muss die Emulation jedoch mit verringerter Taktfrequenz durchgeführt werden [95, S. 196]. In der Arbeitsgruppe Kognitronik und Sensorik wird das FPGA-basierte Rapid-Prototyping-System *RAPTOR* entwickelt, welches durch den modularen Aufbau eine flexible Integration von unterschiedlichen FPGAs und Schnittstellen ermöglicht [135].

Allgemein bieten sich FPGAs, durch die flexible Hardware-Architektur, für die parallele Datenverarbeitung an. So werden FPGAs in vielen Bereichen der Bildverarbeitung eingesetzt. Beispiele hierfür sind in eingebetteten Systemen, in Digitalkameras, (Miniatur-)

Robotern (z. B. *Autonomous Mini Robot*, AMiRo [81]) oder Assistenzsystemen in der Automobilindustrie zu finden.

In dem Serverbereich werden FPGAs vermehrt in dem Bereich *High-Performance Computing* (HPC) zur Steigerung der Energieeffizienz eingesetzt. Durch die enge Kopplung von Prozessor und FPGA wird der Begriff zu *High-Performance Reconfigurable Computing* (HPRC) erweitert. Als Beispiel kann das im Jahr 2010 gestartete *Project Catapult* von Microsoft genannt werden, in dem rekonfigurierbare Strukturen/Architekturen für Rechenzentren entwickelt und untersucht werden [136]. Ein weiteres Beispiel aus dem universitären Bereich stellt das Projekt FiPS (*Developing Hardware and Design Methodologies for Heterogeneous Low Power Field Programmable Servers*) dar [73].

Durch das Anwendungsgebiet der Datenverarbeitung in Satelliten, wird die Anforderung nachträglich auf Änderungen eingehen zu können, besonders verdeutlicht, da hier in der Regel die Hardware nicht ausgetauscht werden kann. Wie bereits erwähnt, wird heutzutage die Verarbeitungsgeschwindigkeit von FPGAs benötigt, um die stetig anwachsende Datenmenge in einem Satelliten verarbeiten zu können. Der Einsatz von FPGAs im Weltraum stellt besondere Anforderungen an diese, welche in dem Kapitel 5 dargestellt werden.

3 Dynamisch partielle Rekonfiguration von FPGAs

Besteht ein System aus Komponenten, die nicht alle gleichzeitig zur Laufzeit benötigt werden, kann ein System in statische und dynamische Komponenten unterteilt werden. Einige moderne FPGAs ermöglichen eine teilweise (partielle) Änderung des Systems zur Laufzeit (dynamisch). Dies entspricht der Konfigurationsart der dynamisch partiellen Rekonfiguration (DPR), wie sie in Abschnitt 2.4 beschrieben wurde. Diese Konfigurationsart bildet die Grundlage dieser Dissertation und wird daher in diesem Kapitel detailliert beschrieben.

Zunächst erfolgt eine Einführung in die Thematik. Dabei werden die möglichen Vor- und Nachteile der Verwendung der DPR beschrieben und durch konkrete Anwendungsbeispiele verdeutlicht. Im Anschluss werden aktuellen FPGA-Familien des Herstellers Xilinx vorgestellt und Neuerungen bei Generationssprüngen hervorgehoben. Darauf folgend wird ein Überblick über die allgemeine Konfiguration eines Xilinx FPGAs gegeben. Dabei werden insbesondere der Konfigurationsspeicher, die Konfigurationsschnittstellen sowie der Konfigurationsablauf eines Xilinx FPGAs beschrieben. Diese Grundlagen sind zum Verständnis der Unterschiede zwischen einer vollständigen und einer dynamisch partiellen Rekonfiguration essentiell. Im anschließenden Abschnitt werden verschiedene Kommunikationsinfrastrukturtypen für die Kommunikation in einem DPR-System vorgestellt und anhand ihrer Merkmale analysiert. Abschließend werden grob- und feingranulare Ansätze für DPR-Systeme vorgestellt und in einem Vergleich gegenübergestellt.

3.1 Einführung

Die dynamisch partielle Rekonfiguration beschreibt den Austausch einzelner Komponenten des FPGAs während des Betriebs. Die nicht vom Austausch betroffenen statischen Komponenten führen ihre Aktion während der Rekonfiguration weiter aus. Somit ergibt sich eine Partitionierung des Systems in statische und dynamische Komponenten. Die dynamisch partielle Rekonfiguration kann dabei über einen externen oder FPGA-internen Rekonfigurationskontroller durchgeführt werden. Die Konfigurationsdateien werden, wie in Abbildung 3.1 dargestellt, in einem nicht flüchtigen Speichermedium abgelegt und durch den Rekonfigurationskontroller zur Konfigurationsschnittstelle transferiert. Als interner Rekonfigurationskontroller kann eine eingebaute Hard-IP-Core CPU (z. B. PowerPC oder ARM), aber auch eine Soft-IP-Core CPU wie der MicroBlade

ze oder ein, im FPGA realisierter, Zustandsautomat verwendet werden. Als externe Rekonfigurationskontrolller können zum Beispiel ein Mikrokontroller oder ein CPLD dienen.

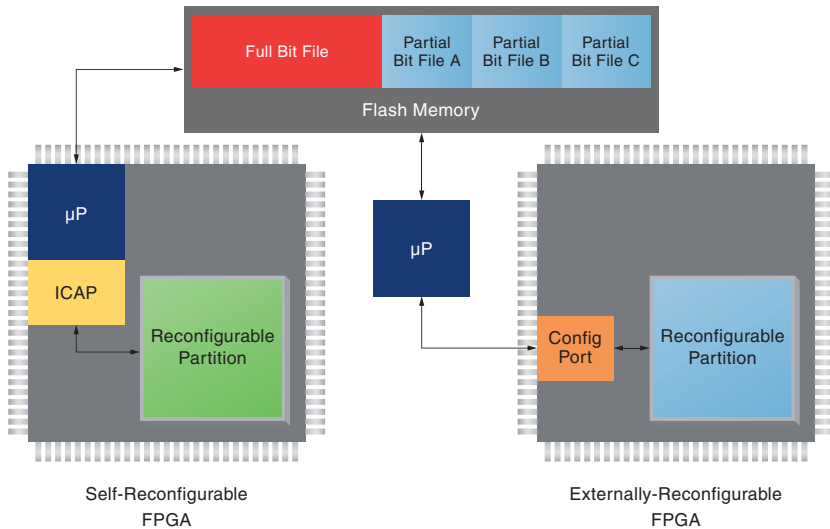


Abbildung 3.1: Dynamisch partielle Rekonfiguration durch einen internen (links) und externen (rechts) Rekonfigurationskontrolller [210, S. 6]

3.1.1 Vor- und Nachteile

Xilinx und Altera sind derzeit die einzigen FPGA-Hersteller, die eine partielle Änderung an dem FPGA zur Laufzeit durch Hard- und Software unterstützen. Die Vor- und Nachteile der Verwendung der DPR werden von den genannten Herstellern in [210] und [5] herausgestellt und folgend zunächst zusammengefasst. Anschließend erfolgt eine Analyse der genannten Vor- und Nachteile durch konkrete Anwendungsbeispiele.

Kosten: Produktionskosteneinsparung vs. Mehrkosten durch Komplexitätsanstieg

Durch die Aufteilung des FPGAs in statische, also stets benötigte, und dynamische, nur zu bestimmten Zeitpunkten benötigte, Komponenten, entsteht auf dem FPGA eine höhere virtuelle Logikdichte. Werden nicht alle Teilkomponenten eines Systems zu jedem Zeitpunkt benötigt, kann also mehr Logik auf einen FPGA platziert werden. Abbildung 3.2 visualisiert diesen Aspekt. Die Systemkosten können daher durch die Verwendung kleinerer, kostengünstigerer FPGAs und/oder durch die Verringerung der Anzahl benötigter FPGAs reduziert werden. Dies kann ebenfalls einen positiven

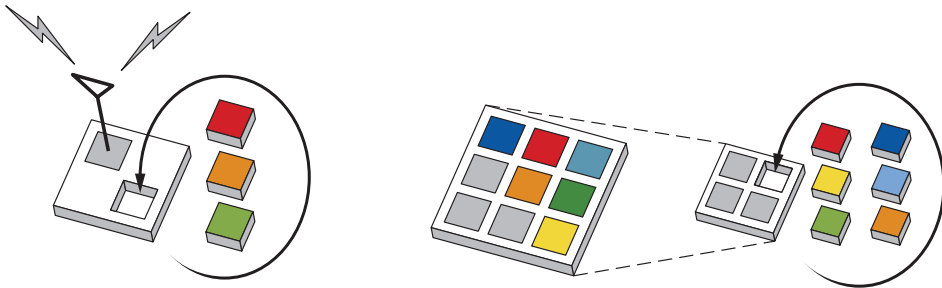


Abbildung 3.2: Visualisierung der Vorteile des Zeitmultiplex-Betriebs und der Verwendung eines kleineren FPGAs [210, S. 3]

Einfluss auf die Produktionskosten (Fertigung und Entwicklungszeit) der benötigten Leiterplatte(n) haben. Durch diesen Zeitmultiplex-Betrieb des FPGAs wird auch von *virtueller Hardware* gesprochen, da nur gerade benötigte Logik auf das FPGA geladen wird. Das aufwendige Multiplexen von Signalen zu den verschiedenen Ausführungseinheiten in einem rein statischen Design entfällt. Die Mehrkosten eines DPR-Systems werden lediglich von Altera angedeutet [5]. In dem Dokument wird abstrakt die erhöhte Komplexität des Designs und der Implementierung aufgeführt, welche sich auf die System-Architektur, die -Beschreibung und den -Test auswirkt.

Reduzierung der Leistungsaufnahme Weiterhin kann die Verlustleistung reduziert werden und somit die Energieeffizienz eines Systems erhöht werden. Die Verlustleistung teilt sich allgemein in statische und dynamische Anteile auf. Die statische Verlustleistung entsteht durch Kriechströme an den Transistorübergängen und kann bereits durch die Verwendung eines kleineren FPGAs reduziert werden. Die dynamische Verlustleistung entsteht allgemein durch Schaltvorgänge innerhalb des FPGAs. Die Verlustleistung kann weiterhin durch die Erstellung von unterschiedlichen Implementierungsvarianten und anschließender Auswahl der optimalen Implementierungsvariante für eine Verarbeitungsaufgabe gesenkt werden. Nach erfolgter Bearbeitung wird die Anwendung wieder vom FPGA gelöscht und somit die Verlustleistung des Gesamtsystems reduziert.

Steigerung der System-Flexibilität Ein weiterer Vorteil ist die Steigerung der System-Flexibilität. So können durch DPR Designänderungen nachträglich, und sogar im laufenden Betrieb, vorgenommen werden. Eine vollständige Neustartphase des gesamten FPGA-Systems entfällt somit (siehe Abschnitt 3.3.3). Durch die Wiederverwendung der statischen Komponenten muss nur die neue dynamische Komponente implementiert werden. Dies führt zu einer deutlich reduzierten Entwicklungs-/Synthesezeit.

Erschließung neuer Anwendungsgebiete Durch die stark ansteigende Menge an Basisblöcken in einem FPGA, steigt proportional die Größe der Konfigurationsdatei und

damit wiederum die benötigte Konfigurationszeit (siehe Abschnitt 3.3.3). Dies kann dazu führen, dass Echtzeitbedingungen von Anwendungen oder Protokolle nicht eingehalten werden können.

Erhöhung der Fehlertoleranz Die Erhöhung der Fehlertoleranz von SRAM-basierten FPGA-Systemen ist eine weitere DPR-Anwendung, welche in [210] jedoch nur abstrakt aufgezählt und nicht weiter detailliert wird. Durch die Erkennung und Korrektur von, durch Strahlung induzierten Fehlern im SRAM-basierten Konfigurationsspeicher kann eine Art Selbstheilung des FPGAs durchgeführt werden.

3.1.2 Analyse und Anwendungen

In diesem Abschnitt werden die zuvor aufgeführten Vor- und Nachteile anhand von wissenschaftlichen Beiträgen analysiert. Hierbei wird insbesondere auf die (Mehr-) Kosten eines DPR-Systems und die Leistungsaufnahme eingegangen. Die Erhöhung der Flexibilität und der Fehlertoleranz eines Systems sind die Kernpunkte dieser Dissertation und werden in diesem Abschnitt nur kurz beschrieben. Details zu diesen Themengebieten folgen somit in den entsprechenden verwiesenen Abschnitten.

Analyse der Kosten

Die erwähnten, möglichen Kosteneinsparungen müssen stets den Mehrkosten zur Bereitstellung der DPR-Funktionalität gegenübergestellt werden. Diese werden jedoch von keinem Hersteller konkretisiert. Aus diesem Grund folgt eine Analyse dieser Mehrkosten. Diese Kosten beinhalten

- einen komplexeren Entwurf des Systems
- die Kommunikationsinfrastruktur zwischen dem statischen und dynamischen Bereich
- einen (komplexeren) Rekonfigurationskontroller
- die Verwaltung der dynamischen Komponenten

Der wesentliche Nachteil von DPR ist der komplexere Entwurf im Vergleich mit einer rein statischen Implementierung. Die Hersteller Xilinx und Altera haben in den letzten Jahren die Entwicklung eines DPR-Systems durch verschiedene Entwurfsabläufe nebst dedizierter Software unterstützt (siehe Abschnitt 3.5.2). Das Abstraktionsniveau wurde dabei immer weiter erhöht. Das vollständige Potenzial der Hardware kann mit diesen Entwurfsabläufen jedoch nicht ausgeschöpft werden. Eine wesentliche Einschränkung ist hierbei, dass die dynamischen Module durch die Kommunikationsinfrastruktur eine feste Platzierung auf dem FPGA erhalten. Im akademischen Bereich sind daher Projekte entstanden, welche eine freie Modulplatzierung auf dem FPGA ermöglichen, wenn Positionen gleiche FPGA-Ressourcen bereitstellen. Weitere Details hierzu folgen im Abschnitt 3.5.

Ebenfalls muss die Kommunikationsinfrastruktur zwischen dem statischen und dynamischen Bereich im Vergleich zu einer rein statischen Implementierung angepasst werden. Allgemein kann durch die Anpassung der Kommunikationsinfrastruktur ein

Vorteil gegenüber einer rein statischen Implementierung entstehen. Der Grund hierfür ist, dass in einem DPR-System die Kommunikationsinfrastruktur nicht von der Gesamtanzahl aller Komponenten, sondern lediglich von der Anzahl maximal benötigter Komponenten zu einem Zeitpunkt abhängt. Details zu unterschiedlichen Arten von Kommunikationsinfrastrukturen werden in Abschnitt 3.4 beschrieben.

Weiterhin bieten aktuelle FPGAs oftmals die Möglichkeit der initialen Rekonfiguration ohne zusätzlichen Rekonfigurationskontroller aus einem angeschlossenen Festwert- oder Flash-Speicher (siehe Abschnitt 3.3.2). In vielen Systemen existiert jedoch ein dedizierter Rekonfigurationskontroller. Der Grund hierfür variiert je nach Anwendung. Als Beispiele können die Beschleunigung der initialen Rekonfiguration des FPGAs, z. B. zur Einhaltung von Echtzeitbedingungen, oder die Erhöhung der Fehlertoleranz des Systems genannt werden (Details zu beiden Anwendungen folgen weiter unten in diesem Abschnitt). In jedem Fall muss ein existierender Rekonfigurationskontroller erweitert werden, um die DPR-Funktionalität realisieren zu können.

Einen zusätzlichen Mehraufwand bringt ebenfalls die Verwaltung der dynamischen Komponenten. Die Verwaltung beinhaltet die Erstellung eines Ablaufplans (engl. *schedule*) und die Platzierung einer dynamischen Komponente. Um die zuletzt genannte Aufgabe ermöglichen zu können, muss die Kontrollkomponente den Speicherort und die Größe der dynamischen Komponente kennen und die DPR mittels Rekonfigurationskontroller steuern. Im besten Fall, kann ein existierender Prozessor diese Verwaltungsaufgaben übernehmen, wodurch es jedoch zu Verzögerungen in der Ausführung anderer Aufgaben des Prozessors kommen kann. Ist dies nicht der Fall, muss dem System eine dedizierte Komponente, in Form eines Zustandsautomats oder (Hard- oder Soft-Core) Prozessors, hinzugefügt werden.

Analyse des Ressourcenbedarfs

Kettelhoit analysiert in [96] unter anderem ob bzw. unter welchen Umständen die zeitversetzte Verwendung der FPGA-Ressourcen in DPR-Systemen im Vergleich mit einem statischen System zu einem verminderten Ressourcenbedarf führt. Bei den DPR-Systemen wird ferner zwischen einem System mit festen Modulplätzen (*FM*) und einem System mit freier Modulplatzierung unterschieden (siehe auch Abschnitt 3.5). In der Dissertation werden 14 typische Anwendungen als dynamische Komponenten ausgeführt. In der Analyse werden anschließend 9 Anwendungen ausgewählt. Die Anwendungen lassen sich in drei Gruppen bezüglich des Ressourcenbedarfs (klein, mittel und groß) einteilen und bestehen jeweils aus drei Anwendungen. Hierbei ist anzumerken, dass die Anzahl der Anwendungen für die Analyse reduziert werden musste, da die verfügbaren FPGA-Ressourcen (Xilinx Virtex -II FPGA XC2V4000 mit 23 040 Slices) für eine statische Implementierung nicht für alle 14 Komponenten ausreichten. Soll ein statisches System alle Anwendungen bereitstellen, müsste daher auf ein größeres FPGA ausgewichen werden. Das Einsparpotential von DPR-Systemen gegenüber einem statischen System hängt im Wesentlichen davon ab, wie viele Anwendungen zur Laufzeit gleichzeitig auf

das FPGA verfügbar sein müssen. Aufgrund des unterschiedlichen Ressourcenbedarf der Anwendungen werden für das DPR-System mit freier Modulplatzierung die beiden möglichen Extremfälle für einen Ablaufplan untersucht. In dem *Best Case* (BC) sind die benötigten Anwendungen mit ansteigender Anzahl an FPGA-Ressourcen sortiert. In dem *Worst Case* (WC) ist die Reihenfolge invertiert.

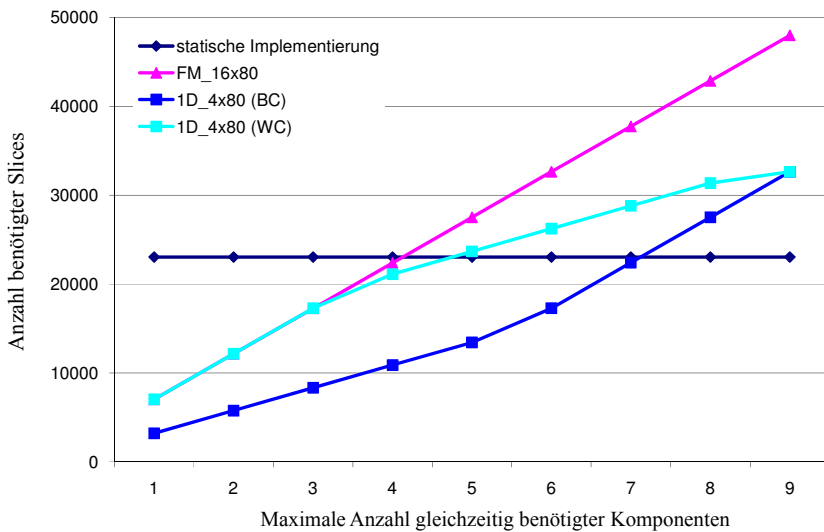


Abbildung 3.3: Gegenüberstellung des Ressourcenbedarfs eines statischen Systems und DPR-Systemen [96, S. 146]

Abbildung 3.3 stellt die Ergebnisse der Analyse gegenüber. Auf der Ordinate sind die Anzahl benötigter Slices und auf der Abszisse die Anzahl gleichzeitig benötigter Anwendungen dargestellt. Da die statische Implementierung stets alle 9 Anwendungen enthält, ist diese als horizontale Linie dargestellt.

Insgesamt ergeben sich folgende, wesentliche Erkenntnisse:

1. Der Ressourcenbedarf der DPR-Systeme ist bei einer geringen Anzahl gleichzeitig benötigter Anwendungen deutlich geringer als der Bedarf der statischen Implementierung.
2. Bei 3 gleichzeitig benötigten Anwendungen ist der Ressourcenbedarf eines DPR-Systems mit festen Modulplätzen identisch zu der WC-Reihenfolge der freien Modulplatzierung.
3. Bei 4 gleichzeitig benötigten Anwendungen können kaum noch FPGA-Ressourcen gegenüber einer statischen Implementierung eingespart werden (Betrachtung der WC-Reihenfolge für die freie Modulplatzierung).

4. Das DPR-System mit freier Modulplatzierung ist ab 4 gleichzeitig benötigter Anwendungen besser als DPR-System mit festen Modulplätzen.
5. Je nach Größe und Typ der Anwendungen können bei dem DPR-System mit freier Modulplatzierung zwischen 4 (WC) und 7 (BC) Anwendungen zu einem Zeitpunkt auf das FPGA geladen werden.
6. Da das untersuchte FPGA insgesamt über 23 040 Slices verfügt, ist es bei den DPR-Systemen nicht möglich alle Anwendungen gleichzeitig zur Verfügung zu stellen.

Zu der letzten Erkenntnis sei angemerkt, dass die Anzahl der Anwendungen initial aufgrund der statischen Implementierung von 14 auf 9 verringert wurde. Werden nicht alle Anwendungen zur gleichen Zeit benötigt, liefert nur ein DPR-System aufgrund der höherer virtuellen Logikdichte eine Lösungsmöglichkeit bei der Verwendung des besagten FPGAs.

Insgesamt fasst Kettelhoit in seiner Dissertation daher zusammen, dass sich der Einsatz von DPR-Systemen gegenüber statischen Systemen bezüglich des Ressourcenbedarfs lohnt, wenn zu jeder Zeit nur wenige Anwendungen gleichzeitig benötigt werden. DPR-Systeme mit freier Modulplatzierung haben weiterhin dann einen Vorteil gegenüber festen Modulplätzen, wenn viele dynamische Komponenten mit unterschiedlichen Ressourcenanforderungen zur Laufzeit benötigt werden. Anzumerken ist abschließend, dass die Analyse auf einem Xilinx Virtex-II FPGA basiert. Aufgrund des vergleichbaren Aufbaus des FPGAs, können die Ergebnisse, so Kettelhoit, auch auf modernere FPGA-Architekturen übertragen werden.

Als konkrete Beispiele für DPR-Systeme im Rahmen dieser Dissertation können der *Heinrich Hertz* oder der *RadSat* Satellit genannt werden (siehe Abschnitt 5.9.2). Die dort verwendeten Algorithmen werden sequentiell auf dem FPGA ausgeführt und erlauben daher einen Zeitmultiplex-Betrieb. Die Verwendung von strahlungstoleranten bzw. -gehärteten FPGAs (siehe Abschnitt 5.8) in diesem Anwendungsgebiet verdeutlicht das Kosteneinsparungspotential durch DPR. Die Kosten für diese speziellen Bauteile sind laut Battezzati et al. um mindestens eine Größenordnung höher als die Kosten vergleichbarer kommerzieller Bauteile [15].

Analyse der Leistungsaufnahme

Leray et al. analysieren in [113, S. 200 ff.] den Trend in der Entwicklung der Verlustleistung eines FPGAs. Durch die Miniaturisierung der Transistorstrukturgröße kommt es ab einer Strukturgröße von etwa 65 nm zu einem Wechsel der primären Größe der Verlustleistung. In kleinen Strukturgrößen dominiert die statische Verlustleistung gegenüber der dynamischen Verlustleistung.

In [115] erfolgt eine Analyse, ob DPR verwendet werden kann, um die Energieeffizienz eines rekonfigurierbaren (FPGA-) Systems zu steigern. Das ideale Szenario in einem solchen System wurde bereits eingangs beschrieben. Ein System wird durch einen Hardware-Beschleuniger erweitert, welcher eine bestimmte Verarbeitungsaufgabe des

Systems beschleunigen kann. Nach Vollendung der Ausführung soll die Erweiterung wieder vom FPGA gelöscht werden, um die statische und dynamische Verlustleistung zu minimieren. Der Löschvorgang selbst entspricht jedoch wiederum einer DPR (siehe Abschnitt 3.3.4), wodurch zusätzliche Energie benötigt wird. Ob und wann sich der Einsatz dieser zusätzlichen Energie lohnt, wird von Liu et al. durch drei Fragestellungen untersucht:

1. Kann die Ausführungszeit minimiert werden und wird dadurch auch die benötigte Energie minimiert?
2. Ist es lohnenswert DPR einzusetzen, um eine Reduzierung der Verlustleistung zu erlangen?
3. Kann DPR andere Techniken wie *clock gating*(CG) übertreffen?

Diesen Fragestellungen wird anhand des Virtex-4-basierten Testsystem *eMIPS* unter Zuhilfenahme analytischer Modelle [115, S. 6 ff.] bei Betrachtung unterschiedlicher Systemkonfigurationen nachgegangen. Das System besitzt einen rekonfigurierbaren Bereich, welcher sich über zwei Taktregionen erstreckt und 12 % der FPGA-Fläche belegt. Als Hardware-Erweiterung wird eine 64-Bit Division bereitgestellt mit einem Anteil von 5 % an der Gesamtverlustleistung. Die Auswertung der analytischen Modelle haben gezeigt, dass die Konfigurationsdatenrate einen starken Einfluss auf die Energieeffizienz hat. Eine performante Anbindung der Konfigurationsschnittstelle an den Speicherort der Konfigurationsdateien ist somit essentiell. Liu et al. beziffern die Anbindung in *eMIPS* mit 399 MB/s, was somit sehr nah an dem theoretischen Maximum mit 400 MB/s (siehe Abschnitt 3.3.2) liegt. Die Auswertungen der Fragestellung führten zu dem Ergebnis, dass die Ausführungszeit und die benötigte Energie minimiert werden kann. Allgemein hängt dies jedoch von den Eigenschaften der jeweiligen Anwendung ab.

Die zweite Fragestellung kann positiv beantwortet werden, wenn die erzielbare Energieminimierung die benötigte Energie für die DPR mindestens ausgleichen kann. Dies wird insbesondere durch eine hohe Konfigurationsdatenrate ermöglicht. Im Testsystem *eMIPS* kann die DPR in weniger als einer Millisekunde erfolgen, sodass die zusätzlich benötigten Energiekosten vernachlässigt werden können und DPR somit zu einer effizienten Technik zur Energieminimierung des Systems wird. Konkret wird festgehalten, dass die Verwendung der Hardware-Erweiterung bereits ab einem Anteil von 8 % am Gesamtprogramm zu einer Energieminimierung führt. Ferner lohnt sich, basierend auf den analytischen Modellen, ein Löschen der Hardware-Erweiterung ab einer Ruhezeit von 9,1 ms.

Liu et al. stellen zur Beantwortung der dritten Fragestellung eine Implementierung des *clock gating* Verfahrens auf einem Virtex-4 FPGA vor. Sie fassen zusammen, dass die Technik einfach auf einem FPGA implementiert und innerhalb von zwei Takten aktiviert werden kann. CG kann jedoch nur den dynamischen Anteil der Verlustleistung eliminieren, während ein Löschvorgang durch DPR den statischen und dynamischen Anteil reduziert. Die Auswertung am *eMIPS* Demonstrator zeigt, dass ab einer Ruhezeit von 27 ms DPR energieeffizienter als die *clock gating* Technik ist. Liu et al. unterstreichen, dass bei der Auswertung der Fragestellungen ein kleines DPR-System verwendet

wird. Als Diskussion wird daher eine Extrapolation der Ergebnisse vorgestellt [115, S. 18 ff.], in dem die Hardware-Erweiterungen einen Anteil von 50 % an der Gesamtverlustleistung haben. In diesem Fall bietet DPR ein deutlich größeres Potenzial zur Energieminimierung als CG.

Anwendungen mit Anforderung erhöhter System-Flexibilität

Das zuvor beschriebene Szenario der Wiederverwendung der statischen Komponenten, sodass nur neue dynamische Komponente implementiert werden müssen, stellt den Idealfall dar. Dies ist nur durch eine vollständige Entkopplung der Implementierung der statischen und dynamischen Komponenten möglich, welche zur Zeit von keinem der beiden Hersteller unterstützt wird. Aus diesem Grund sind im akademischen Bereich Projekte entstanden, welche Entwurfsabläufe für diese benötigte Entkopplung anbieten. Weitere Details hierzu folgen im Abschnitt 3.5.

Anwendungen aus dem *Software-Defined Radio* (SDR) Bereich sind typische und historisch gesehen die ersten Anwendungsbeispiele für dynamisch partielle Rekonfiguration [210] oder [94, S. 65 ff.]. Die Änderung des Kommunikationskanals zur Laufzeit stellt hierbei eine besondere Anforderung an die System-Flexibilität (siehe Abbildung 3.2). Ein weiteres Beispiel ist in sogenannten *Optical Transport Networks* (OTN) zu finden [7]. Dort werden unterschiedliche Protokolle eingesetzt, wobei pro Kanal nur ein konkretes Protokoll aktiv sein kann. Zusätzlich zu diesem Zeitmultiplexing ist es, ähnlich wie bei SDR, nötig, im Betrieb neue Protokolle aufnehmen zu können.

Das Anwendungsgebiet Weltraum ist ebenfalls ein ideales Beispiel, in dem nachträgliche Designänderungen wünschenswert sind. In FPGA-basierten Weltraummissionen ist der Trend zu späten Designänderungen, wie zum Beispiel dem Austausch von Kommunikationsprotokollen zur Laufzeit, eindeutig erkennbar (siehe Abschnitt 5.9.2).

Anwendungen mit Echtzeitanforderungen

Am Beispiel von PCI Express wird die Einhaltung von Echtzeitbedingungen konkretisiert [210]. Durch das benötigte *Linktraining* nach maximal 100 ms nach dem Systemstart. Durch die Konfiguration eines initial sehr kleinen Designs, welches lediglich den PCIe-Block sowie die zugehörige Logik konfiguriert, kann die geforderte harte Echtzeit von 100 ms eingehalten werden. Anschließend können die weiteren Systemkomponenten durch DPR nachgeladen und somit das gesamte System initialisiert werden.

Eine konkrete Implementierung für eine solche zweistufige Initialisierung ist z. B. in [122] zu finden. Meyer et al. haben die Startzeit eines CAN-basiertes Steuergeräts (engl. Electronic Control Unit, ECU), basierend auf einem Spartan-6 Evaluierungsboards (SP605), durch das vorgestellte *Fast FPGA Start-up* um 78 % beschleunigen können.

Anwendungen mit Anforderung erhöhter Fehlertoleranz

Diese Selbstheilung des Konfigurationsspeichers erfolgt durch eine teilweise Überschreibung der durch Strahlung verursachte fehlerhafte Speicherbereich zur Laufzeit. In der

Literatur hat sich der englische Begriff *Scrubbing* (dt. schrubben, reinigen) durchgesetzt. Dieses Verfahren wird somit erst durch die DPR-Funktionalität ermöglicht, ist ein zentraler Punkt dieser Dissertation und wird detailliert in Abschnitt 5.7.3 beschrieben.

Zusätzlich wurde im Rahmen der Dissertation von Cozzi [43] und dieser Dissertation ein neuartiges Verfahren zur Erkennung von permanenten Fehlern in den Verdrahtungsressourcen eines FPGAs entwickelt. Die dynamisch partielle Rekonfiguration wird in diesem Verfahren für die Platzierung von kleinen, autonomen Testschaltungen verwendet. Weitere Details zu diesem neuen Verfahren folgen in Abschnitt 6.4.2.

Konkrete Anwendungsbeispiele im terrestrischen und kosmischen Bereich sind in Abschnitt 5.9 und insbesondere in Abschnitt 6.1 durch einen eigenen Demonstrator für dynamisch rekonfigurierbare Verarbeitungsmodule, einem Prototypen für die Datenverarbeitung in einem Satelliten, beschrieben.

Gegenüberstellung der Vor- und Nachteile

In diesem Abschnitt werden die, durch die Analyse erweiterten, möglichen Vor- und Nachteile zusammengefasst. Die Verwendung der DPR-Funktionalität bietet folgende mögliche Vorteile:

- Reduzierung der Produktionskosten: Anzahl der oder Größe des FPGAs kann reduziert werden
- Reduzierung der Leistungsaufnahme: Je nach Anwendung möglich und teils bessere Ergebnisse als existierende Stromspartechniken (z. B. *clock gating*)
- Erhöhung der System-Flexibilität: Erhöhte Wiederverwendbarkeit des Designs und nachträgliche Designänderungen
- Erschließung neuer Anwendungsgebiete: z. B. Einhaltung von Echtzeitbedingungen
- Erhöhung der Fehlertoleranz: Selbstheilung durch Scrubbing-Verfahren und Erkennung permanenter Fehler

Die aufgeführten Vorteile werden durch folgende Nachteile erkaufte:

- Mehrkosten durch komplexeren Systementwurf
- Entwurfsabläufe der Hersteller ermöglichen nur feste Modulplätze
- Notwendigkeit eines (komplexeren) Rekonfigurationskontrollers
- Erhöhter Verwaltungsaufwand: Verwaltung der dynamischen Komponenten

Als neutral können die folgenden Punkte festgehalten werden:

- DPR-Kommunikationsinfrastruktur: Komplexer für DPR-Systeme, Ressourcenbedarf abhängig von der Anzahl maximal benötigter Komponenten zu einem Zeitpunkt
- Gesamt-Ressourcenbedarf des Systems: Einsatz von DPR-Systemen gegenüber statischen Systemen lohnt, wenn zu jeder Zeit nur wenige Anwendungen gleichzeitig benötigt werden

3.2 Xilinx FPGA-Familien

Im Rahmen dieser Dissertation werden primär die FPGA-Familien des Herstellers Xilinx betrachtet. In diesem Abschnitt werden die aktuellen FPGA-Familien vorgestellt und die Neuerungen bei Generationssprüngen hervorgehoben. Insbesondere wird auf den Aufbau der konfigurierbaren Logikblöcke (CLB) und die Slice-Architektur eingegangen. Diese Informationen sind essentiell für die Generierung einer eigenen Datenbank sowie der Entwicklung eigener Algorithmen zur Packung, Platzierung und Verdrahtung, welche in Abschnitt 3.5.7 und insbesondere Kapitel 4 vorgestellt werden. Aufgrund von Änderungen in der Hersteller-Software unterstützen die eigenen Softwarekomponenten die FPGA-Familien bis zur 7-Serie. Die unterstützten Familien ab der Virtex-4-Familie werden daher kurz vorgestellt. Die Übersicht basiert auf [103, S. 29 ff.], wird an dieser Stelle zum einen zusammengefasst und zum anderen durch aktuelle FPGA-Familien ergänzt. Weiterhin erfolgt im Unterschied zur genannten Quelle eine detaillierte Beschreibung der FPGA-Ressourcen. Abschließend werden die FPGA-Familien bezüglich der Slice-Architektur und den verfügbaren FPGA-Ressourcen gegenübergestellt. Zur Darstellung der Entwicklung der FPGA-Familien werden ebenfalls die neueren FPGA-Familien UltraScale und UltraScale+ berücksichtigt.

3.2.1 Virtex-4

Die Basisblöcke der FPGAs ab der Virtex-4-Familie werden in sogenannte *Advanced Silicon Modular BLock* (ASMBL) Generationen zusammengefasst. Die erste Version der ASMBL baut auf den Basisblöcken der Virtex, Virtex-E sowie Virtex-II (Pro) auf und erweitert diese um neue Funktionen. Die Virtex-4-Familie ist in drei Unterfamilien aufgeteilt: LX, SX und FX. Eine Übersicht über die FPGA-Familie und den konkreten Funktionen der FPGAs erfolgt in [227]. Die LX-Unterfamilie besteht primär aus CLBs, während in der SX-Unterfamilie mehr DSP-Blöcke zur Verfügung stehen. Die FX-Unterfamilie wiederum bietet sich aufgrund von bis zu zwei eingebauten RISC (engl. *Reduced Instruction Set Computer*)-Prozessoren, PowerPC 405, und performanten IO-Schnittstellen (*Multi-Gigabit Transceiver*, *MGT*, *Xilinx RocketIO* mit bis zu 6,5 Gbit/s) für eingebettete Systeme mit Bedarf an performanten, externen Anbindungen an.

Interner Aufbau des CLBs Ein konfigurierbarer Logikblock (CLB) ist, wie in Abbildung 3.4 an eine Switch Matrix angeschlossen und enthält vier Slices. Die Slices entsprechen einem von zwei möglichen Typen, *Typ L* und *Typ M*, und sind entsprechend des Typs gruppiert: Zwei Slices vom Typ M sind innerhalb eines CLBs links und zwei Slices vom Typ L rechts angeordnet. Beide Slice-Gruppen besitzen eine dedizierte Leitung für die Implementierung einer Carry Chain (Anschlüsse *COUT* und *CIN*). Weiterhin ist in der Abbildung 3.4 zu erkennen, dass nur Slices vom Typ M eine Leitung für Schieberegister (Anschlüsse *SHIFTIN* und *SHIFTOUT*) besitzen. Die Konfigurationsmöglichkeiten einer CLB-Slice werden in [229, S. 183 ff.] vorgestellt.

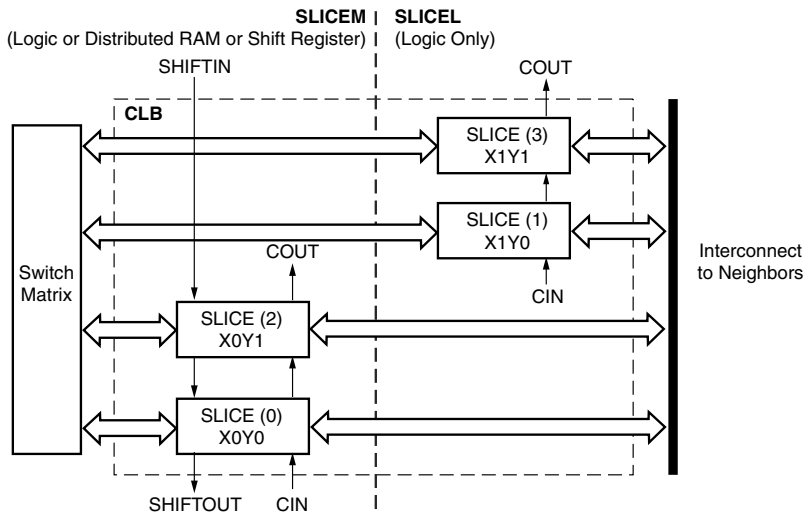


Abbildung 3.4: Darstellung des Aufbaus eines CLB der Virtex-4-Familie [229, S. 183]

Part Eine Virtex-4 Slice besteht, unabhängig von dem Typen, aus zwei sogenannten *Parts*. In einem Part steht ein Funktionsgenerator (basierend auf einer Look-Up-Table, LUT), eine Carry-Logik, arithmetische Gatter (UND- und XOR-Gatter), Multiplexer und ein Register zur Designimplementierung zur Verfügung. Eine Virtex-4 LUT kann eine Boolesche Funktion mit vier Eingängen implementieren. Da diese logische Funktion durch einen Speicher abgebildet wird, ist die Verarbeitungszeit konstant und somit unabhängig von der Funktion. Ein Funktionsgenerator in einem Slice vom Typ M kann, familienübergreifend, zusätzlich als verteilter Speicher (engl. distributed RAM, auch LUTRAM) oder Schieberegister (engl. Shift Register) konfiguriert werden.

Familienübersicht Die verfügbaren Ressourcen aller FPGAs der Virtex-4-Familie sind tabellarisch in [230] aufgeführt und in fünf Gruppen unterteilt: Logik, Speicher, Takt, Ein-/Ausgang und eingebettete Hard IP-Cores. Die Logik-Ressourcen werden in Slices, Flip-Flops und Logic Cells angegeben. Die Definition der Logic Cells wurde von Xilinx zum Vergleich mit anderen FPGA-Herstellern erstellt und wird durch das Produkt der Anzahl der Slices und einem (Familien-) spezifischen Parameter, R_{LB2S} , berechnet. Der Parameter R_{LB2S} spiegelt dabei die Anzahl der äquivalenten Logikgatter wieder. Die zwei LUTs einer Slice sowie die arithmetischen Logikgatter (AND und XOR) ergeben für die Virtex-4-Slice-Architektur den kumulierten Wert 2,25. Die FPGAs der Virtex-4-Familie bieten bis zu 200 448 Logic Cells, realisiert durch 89 088 Slices mit bis zu 178 176 internen Registern und LUTs. Die Speicherressourcen sind in LUTRAM, Block RAM

FIFO und Block RAM aufgeteilt. In der Virtex-4-Familie sind bis zu 9 936 kbit BRAM und bis zu 1 392 kbit LUTRAM verfügbar. Die Anzahl der DCM (Digital Clock Manager) variiert zwischen 4 und 20 und ist in dem Abschnitt Takt-Ressourcen aufgeführt. Für die IO-Ressourcen sind die Anzahl der Ein- und Ausgänge für asymmetrische (engl. single-ended) und differenziellen (engl. differential) Signalübertragung sowie die jeweils unterstützen Protokolle angegeben. Die eingebetteten Hard-IP-Ressourcen liefern eine Auflistung der verfügbaren DSP-, PowerPC-, Ethernet- sowie RocketIO-Blöcke. Die Anzahl der DSP-Blöcke variiert von 32 bis zu 512 in der auf DSP-optimierten SX-Unterfamilie. Die drei letztgenannten IP-Ressourcen sind, wie eingangs erwähnt, nur in der FX-Unterfamilie verfügbar. Es können bis zu zwei PowerPC-Prozessoren, vier Ethernet-MACs und 24 RocketIO-Transceiver in ein Design integriert werden.

3.2.2 Virtex-5

Die Virtex-5-Familie basiert auf der Virtex-4-Familie und bietet teils überarbeitete sowie neue Basisblöcke und integriert somit neue Funktionen in ein System. Ein Beispiel hierfür ist die Integration von PCI Express (PCIe) durch einen Hard-IP-Core. Die Basisblöcke der Virtex-5-Familie werden zu der zweiten ASMBL-Generation zusammengefasst. Die Virtex-5-Familie teilt sich in fünf Unterfamilien auf [231]: LX, LXT, SXT, TXT und FXT. Das *T* am Namensende weist auf zusätzliche serielle Verbindungsmöglichkeiten hin: PCIe (Spezifikation 1.1) und Ethernet MACs (10/100/1 000 Mbit/s). Die LXT und SXT Unterfamilie integrieren zusätzlich MGTs (RocketIO), Transceiver mit bis zu 3,75 Gbit/s, die TXT und FXT Unterfamilie mit bis zu 6,5 Gbit/s. Die Unterfamilienbezeichnung ist an die Bezeichnung der Virtex-4-Familie angelehnt. So enthält die LX-Unterfamilie primär CLBs, während in der SX-Unterfamilie mehr DSP-Blöcke zur Verfügung stehen. Die FX-Unterfamilie integriert den neueren Prozessor PowerPC440 (V4: PowerPC405). Die Unterfamilie TXT bildet eine neue Unterfamilie mit einem Fokus auf externe, serielle Verbindungsblöcke. Die größte Veränderung bei dem Generationswechsel von Virtex-4 zu Virtex-5 liegt in der internen Slice-Architektur.

Interner Aufbau des CLBs FPGAs der Virtex-5-Familie besitzen bis zu 51 840 Slices. Obwohl die Slice-Anzahl geringer als bei der Virtex-4-Familie ist, sind mit bis zu 207 360 internen Registern und LUTs größere Designs möglich. Grund hierfür ist eine strukturelle Veränderung innerhalb der Slices. Ein CLB wird aus zwei Slices gebildet, die Anzahl der Parts verdoppelt sich dafür von zwei auf vier. Die Anzahl der Parts und somit auch die Anzahl der Funktionsgeneratoren und Register für ein CLB im Vergleich zu der Virtex-4-Familie ist daher konstant geblieben. Die genauen Konfigurationsmöglichkeiten eines CLBs werden in [234, S. 173 ff.] beschrieben. In Abbildung 3.5 ist eine Übersicht des Aufbaus und der Verbindungen eines Virtex-5 CLBs visualisiert.

Part Ein Funktionsgenerator einer Slice enthält eine LUT (*LUT6*), welche in zwei unabhängige LUTs (*LUT5*) aufgeteilt werden und entsprechend *zwei* Boolesche Funktionen beschreiben kann (siehe Abbildung 3.6).

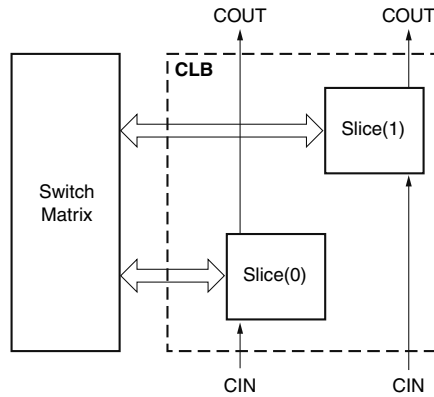


Abbildung 3.5: Darstellung des Aufbaus eines CLB der Virtex-5-Familie [234, S. 173]

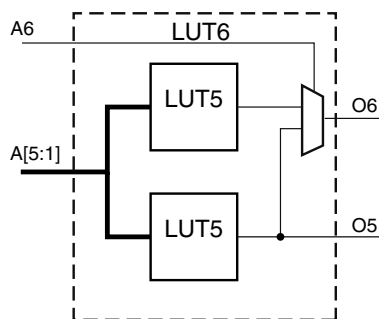


Abbildung 3.6: Interner Aufbau der LUT der Virtex-5-Familie [218, S. 13]

Die LUT6 besitzt nun sechs anstatt vier Eingänge ($A1 - A6$) und zwei unabhängige Ausgänge ($O5$ und $O6$). Die Verwendung beider LUTs bedingt, dass die beiden Funktionen durch die gleichen Signale an den ersten fünf Eingängen ($A1 - A5$) oder durch fünf unabhängige Signale gebildet werden können. Nur der sechste Eingang ($A6$) ist an die logische Funktion des Ausgangs $O6$ gebunden. Durch drei Multiplexer können die vier Funktionsgeneratoren kombiniert werden, um Slice-intern eine Funktion mit acht Eingängen zu realisieren. Funktionen mit sieben Eingängen werden durch die Kombination zweier Funktionsgeneratoren aufgebaut. Eine Slice vom Typ M erweitert die Funktionalität einer Slice vom Typ L. Einzig die Funktionsgeneratoren einer Slice vom Typ M unterstützen die, bereits aus der Virtex-4-Familie bekannte, Speicherfunktionalität

als Schieberegister oder verteilter Speicher (LUTRAM). Die strukturelle Veränderung gegenüber einem Virtex-4-Part ist besonders durch die klare Abgrenzung der einzelnen Parts erkennbar. Jedem Part einer Slice wird ein Buchstabe (A-D) zugeordnet. Der obere Part trägt den Buchstaben D, die Buchstaben der weiteren Parts werden in absteigender Anordnung dekrementiert.

Familienübersicht Die Anzahl der verfügbaren Logikressourcen in der Virtex-5-Familie steigt auf bis zu 331 776 Logic Cells, realisiert in 51 840 Slices mit bis zu 207 360 internen Registern und LUTs [235]. Der Parameter R_{LB2S} erhöht sich durch die gestiegene Anzahl von Parts und LUTs sowie der Möglichkeit der doppelten Nutzung einer LUT (LUT5 und LUT6) auf 6,4. Die FPGAs der Virtex-5-Familie integrieren bis zu 18 576 kbit BRAM und bis zu 4 200 kbit LUTRAM. Bei den eingebetteten Hard-IP-Ressourcen ist der Eintrag für PCIe-Blöcke hinzugekommen. Wie eingangs beschrieben, integrieren vier der fünf Unterfamilien mindestens einen und maximal vier PCIe-Block/Blöcke. Die Anzahl der verfügbaren DSP-Blöcke steigt auf 1 056 und hat sich somit in dem Generationsprung mehr als verdoppelt. Ebenfalls verdoppelt wird die Anzahl der Gigabit-Ethernet-MACs (auf nun acht) und die Anzahl der RocketIO-Transceiver (48).

3.2.3 Virtex-6

Die Virtex-6-Familie baut wiederum auf die Virtex-5-Familie auf und integriert, analog zum Generationswechsel von Virtex-4 nach Virtex-5, überarbeitete Blöcke der dritten ASMBL-Generation. Die Virtex-6-Familie teilt sich in drei Unterfamilien auf: LXT, SXT und HXT. Die Namensgebung entspricht ebenfalls der, der Virtex-4- und Virtex-5-Familien. Die neue Unterfamilie HXT bietet eine höhere Bandbreite bei seriellen Hard-IP Verbindungsblöcken. Eine Übersicht über die Familie ist in [237] gegeben.

Interner Aufbau des CLBs Die Konfigurationsmöglichkeiten eines CLBs werden ab der Virtex-6-Familie in einem separaten Dokument aufgeführt [238]. Die einzelnen XDL-Konfigurationsschalter sind in [103, S. 41 ff.] beschrieben. Der interne Aufbau eines Virtex-6 CLBs hat sich bei dem Generationsprung nicht verändert. Ein CLB wird, wie bereits in Abbildung 3.5 dargestellt, aus zwei Slices aufgebaut. Eine Veränderung zu der Virtex-5-Familie ist jedoch innerhalb der Slices bzw. Parts zu finden und wird im folgenden Abschnitt vorgestellt.

Part Die Anzahl der verfügbaren Register innerhalb eines Parts wird erhöht. Das zusätzliche Register kann jedoch nur als Flip-Flop benutzt werden. Sind die ersten Register in einer Slice als Latches konfiguriert, können die zusätzlichen Register nicht verwendet werden [238, S. 12]. Weiterhin ist der Eingang diese zusätzlichen Flip-Flops auf zwei mögliche Eingänge begrenzt: Ausgang O5 des Funktionsgenerators und dem Bypass-Eingang des Parts. Ab der Virtex-6-Familie besteht zudem die Möglichkeit das Register als Logikgatter zu verwenden. Dabei ist die Realisierung als OR-Gatter (OR2L) oder AND-Gatter (AND2L) mit zwei Eingängen möglich. Diese Funktion ist in [238] nicht beschrieben, jedoch als HDL-Primitive verfügbar. Die Funktionsgeneratoren kön-

nen, wie schon in der Virtex-5-Familie, entweder als LUT6 mit sechs Eingängen und einem Ausgang oder als zwei LUTs (LUT5 und LUT6) mit fünf (gemeinsam benutzten) Eingängen und zwei separaten Ausgängen konfiguriert werden.

Familienübersicht Die Virtex-6-FPGAs stellen bis zu 758 784 Logic Cells durch 118 560 Slices zur Verfügung [240]. Die Werte haben sich bei dem Generationssprung somit mehr als verdoppelt. Die Anzahl der verfügbaren LUTs steigt auf 474 240, die Anzahl der maximal verfügbaren Flip-Flops durch die Verdopplung innerhalb eines Parts auf 948 480. Die Verdopplung der Register hat keinen Einfluss auf den Parameter R_{LB2S} , sodass dieser, analog zur Virtex-5-Familie, bei 6,4 liegt. Der maximal verfügbare interne Speicher wird bei dem Generationssprung für beide Kategorien verdoppelt: Bis zu 38 304 kbit BRAM und 8 280 kbit LUTRAM können in einem Design integriert werden. Bei der Gruppe der eingebetteten Hard-IP-Ressourcen ist bei der Anzahl der verfügbaren DSP-Blöcken abermals nahe zu eine Verdopplung auf nun 2 016 Blöcken erkennbar. Die Anzahl der anderen Ressourcen ist, mit Ausnahme der maximalen Anzahl an Ethernet-MACs (maximal vier), identisch.

3.2.4 Spartan-6

Bei der Spartan-Familie ist ein Sprung von drei auf sechs in der Namensbezeichnung zu erkennen. Dies ist der Ähnlichkeit zur Virtex-6-Familie geschuldet. Die Spartan-6-Familie ist in nur zwei Unterfamilien aufgeteilt [217]: LX und LXT. Die LXT-Unterfamilie erweitert die LX-Unterfamilie um energiesparsame, serielle Verbindungsblöcke. Die Spartan-6-Familie unterscheidet sich von der Virtex-6-Familie deutlich in Bezug auf die Kosten und Komplexität (FPGA-Ressourcen).

Interner Aufbau des CLBs Die Konfigurationsmöglichkeiten eines Spartan-6-CLBs sind in [218] beschrieben. Die Spartan-6-Familie definiert neben den bereits bekannten zwei Slice-Typen (L und M) einen weiteren Slice-Typ: X. Dieser neue Typ beschneidet die Funktionalität einer Slice vom Typ L um die Carry Chain- und Multiplexer-Funktionalität. Die Integration des Slice-Typs X führt zu einer Veränderung in den Verbindungen eines CLBs, welche in Abbildung 3.7 dargestellt ist. Ein CLB besteht aus zwei Spalten, wobei die zweite Spalte stets mit einer Slice vom Typ X und die erste Spalte alternierend Slices vom Typ L oder M integriert. Somit besitzen FPGAs der Spartan-6-Familie ca. 50 % Slices vom Typ X und je ca. 25 % Slices vom Typ L und M.

Part Bei dem Vergleich mit dem Aufbau einer Virtex-6 Slice fallen weitere Änderungen bzw. Einschränkungen auf. So fällt zum Beispiel der Multiplexer vor dem zweiten Register weg, sodass lediglich der Ausgang O5 des Funktionsgenerators in dem Flip-Flop abgespeichert werden kann. Weiterhin wird für das zweite Register der initiale Wert und der Wert nach einem Zurücksetzen auf einen gemeinsamen Wert beschränkt.

Familienübersicht Die FPGAs der Spartan-6-Familie bieten bis zu 23 038 Slices mit 147 443 Logic Cells [219]. Die Parts bestehen aus den gleichen Komponenten wie die

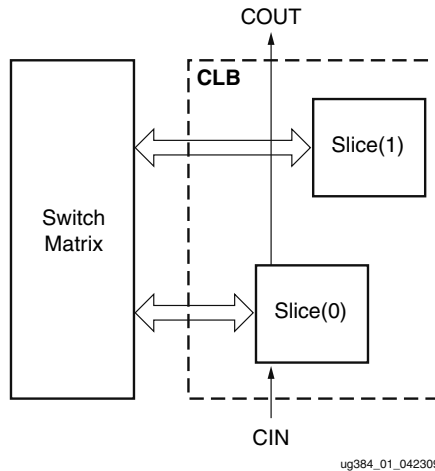


Abbildung 3.7: Darstellung des Aufbaus eines CLBs der Spartan-6-Familie [218, S. 7]

eines Virtex-6 Parts, sodass die Anzahl der bereitgestellten LUTs 92 152 und die Anzahl der verfügbaren Flip-Flops 184 304 beträgt. Der Parameter R_{LB2S} ist auch in dieser Familie konstant geblieben und beträgt 6,4. Die Größen der eingebetteten Speicher liegen für den LUTRAM auf dem Niveau der Virtex-4-Familie (1 355 kbit). Die Größe des integrierten BRAMs ist mit 4 824 kbit sogar weniger als die Hälfte des, in der Virtex-4-Familie bereitgestellten, Speichers. Bei der Anzahl der Hard-IP-Ressourcen wird ebenfalls die Kosteneinsparung deutlich. Die maximale Anzahl der verfügbaren DSP-Blöcke liegt bei 180. Ferner wird nur in der LXT-Familie ein PCIe-Block integriert. Ebenfalls ist die Anzahl der seriellen Gigabit-Transceiver auf maximal acht in der LXT-Familie beschränkt.

3.2.5 7-Serie und Zynq-7000

Mit der 7-Serie werden erstmals FPGA-Familien durch eine Bezeichnung zusammengefasst [194]. Die 7-Serie besteht aus drei Familien, welche sich für unterschiedliche Anwendungsszenarien anbieten:

- Artix-7: Familie mit kleinen, kostengünstigen FPGAs für Anwendungen mit hohen Stückzahlen
- Kintex-7: Neue Familie optimiert auf das Preis-Leistungsverhältnis
- Virtex-7: Familie optimiert für Anwendungen mit hoher Anforderung an Leistung und/oder Kapazität (FPGA-Ressourcen)

Während die FPGAs der Artix- und Kintex-7-Familie nicht in weitere Unterfamilien aufgeteilt sind, existieren für die FPGAs der Virtex-7-Familie drei Unterfamilien:

T, XT und HT. Die Unterteilung innerhalb der Virtex-7-Familie ist abermals an den Funktionsumfang der Gigabit-Transceiver geknüpft. Die T-Unterfamilie integriert GTX-Transceiver mit 12,5 Gbit/s, die XT-Unterfamilie GTH-Transceiver mit 13,1 Gbit/s und die HT-Unterfamilie GTZ-Transceiver mit 28,05 Gbit/s. Alle FPGAs der 7-Serie verfügen zusätzlich über einen 12 bit Analog-Digital-Umsetzer (engl. **Analog-to-Digital Converter**, ADC). Alle Blöcke bilden die vierte Generation der ASMBL-Architektur.

Ein weiteres Novum ist in den größeren Virtex-7-FPGAs zu finden, welche physikalisch aus mehreren Teilkomponenten (Dies) bestehen und über spezielle Verbindungen, sogenannten *Stacked Silicon Interconnect* (SSI), verbunden werden. So besteht der größte FPGA der 7-Serie (XC7V2000T) aus vier Einzelkomponenten.

Die Zynq-7000-Familie bildet ebenfalls eine neue, eigene Familie in der ein ARM Cortex-A9 Zweikern-Prozessor mit bis zu 1 GHz und ein FPGA zu einem (MP)SoC kombiniert werden. Der prozessorbasierte Teil bildet das sogenannte *Processing System* (PS), das FPGA die *Programmable Logic* (PL). Die PL eines Zynqs ist (je nach Ausführung) durch einen FPGA der Artix- oder Kintex-7-Familie realisiert. Da die PL somit identisch zu den 7-Serie FPGAs ist, gelten die folgenden Eigenschaften ebenfalls für die Zynq-Familie und sind in diesem Abschnitt zusammengefasst aufgeführt.

Interner Aufbau des CLBs Der Funktionsumfang und der interne Aufbau eines CLBs der 7-Serie ist in [192] beschrieben und identisch zu dem Aufbau eines Virtex-6 CLBs. Eine Neuerung ist in der Anordnung der Blöcke innerhalb des FPGAs zu finden. Die Anbindung der Blöcke in den vorherigen FPGA-Familien erfolgt stets durch die jeweils linksseitige Switch-Matrix. In den 7-Serie FPGAs sind die Blöcke hingegen alternierend an links- und rechtsseitige Switch-Matrizen angeschlossen. Diese Änderung führt zu einer weiteren Inhomogenität in dem strukturellen FPGA-Aufbau.

Familienübersicht Die verfügbaren FPGA-Ressourcen der 7-Serie- und Zynq-7000-FPGAs werden in vier einzelnen Tabellen dargestellt (Artix-7: [197], Kintex-7: [206], Virtex-7: [241] und Zynq-7000: [253]). Die FPGAs der 7-Serie bestehen aus bis zu 305 400 Slices und 1 954 560 Logic Cells. Dies entspricht abermals mehr als eine Verdopplung (Faktor 2,58) der verfügbaren Logikressourcen bei dem Generationssprung. Die Anzahl der verfügbaren FGs steigt durch den identischen Aufbau auf 1 221 600 und die Anzahl maximal verfügbaren Flip-Flops auf 2 443 200. Der maximal verfügbare interne Speicher in Form von LUTRAM wird durch die Anzahl der verfügbaren FGs ebenfalls mehr als verdoppelt: 21 550 kbit. Der eingebettete Speicherplatz steigt um einen Faktor 1,77 auf 67 680 kbit. Bei den Hard-IP-Cores sind zwei neue Blocktypen aufgeführt: Der eingangs erwähnte Analog-Digital-Umsetzer (XADC) und die Verschlüsselungskomponenten zum Schutz der FPGA-Konfiguration durch AES-256 und HMAC (engl. **Hashed Message Authentication Code**). Die AES-Komponente ist laut [215, S. 8 f.] allerdings bereits seit der Virtex-4-Familie und HMAC-Komponente seit der Virtex-6-Familie als Hard-IP-Core verfügbar. Die PCIe-Blöcke werden in unterschiedlichen Breiten (engl. lanes) und Spezifikationen integriert: Artix-7 (x4 Gen2), Kintex-7 (x8 Gen2), Virtex-7 (T x8 Gen2, XT und HT x8 Gen3), Zynq-7000 (x4 - x8

Gen2, Ausnahmen Z-7010 und Z-7020 ohne PCIe). Bei den Gigabit-Transceivern sind maximal 96 GTH, und 16 der schnelleren GTZ integrierbar.

3.2.6 Gegenüberstellung der FPGA-Familien

Die zuvor beschriebenen Unterschiede werden in diesem Abschnitt tabellarisch gesammelt und gegenübergestellt. Dabei wird separat auf die Änderungen in der Slice-Architektur und der Entwicklung der FPGA-Ressourcen eingegangen. Zur Darstellung der Entwicklung der FPGA-Familien werden ebenfalls die FPGA-Familien UltraScale und UltraScale+ berücksichtigt [222]. Diese neuen FPGA-Familien unterteilen sich jeweils in die Unterfamilien Kintex, Virtex und Zynq.

Slice-Architektur

Die Tabelle 3.1 stellt Informationen über die internen Komponenten eines CLB für die unterstützten FPGA-Familien gegenüber. Die Informationen werden dabei in unterschiedlichen Kategorien vorgestellt: Slices, Parts, LUT, Register, arithmetische Gatter, Carry Chain (CC) und R_{LB2S} .

Tabelle 3.1: Gegenüberstellung der Slice-Architekturen der FPGA-Familien (basierend auf [103, S. 47])

		V4	V5	S6, V6, 7S, Z7	US (+)
Slice	# Slices/CLB	4	2	2	1
	Slice-Typen	L/M	L/M	X ¹ /L/M	L/M
Part	# Parts/Slice	2	4	4	8
	# LUTs/Slice	2	4 - 8	4 - 8	8 - 16
LUT	# LUTs/Part	1	2	2	2
	# LUT-Ausgänge	1	2	2	2
	LUT-Typen	LUT4	LUT5,LUT6	LUT5,LUT6	LUT5,LUT6
	# Register/Slice	2	4	8	16
Register	Register-Typen	R1:L/FF	R1:L/FF	R1:L/FF/AO2L R2:FF	R1:L/FF/AO2L R2:FF
	Arithm. Gatter	# Gatter/Slice	4	4	4
	Gatter-Typen	AND, XOR	XOR	XOR	XOR
CC	# CC/CLB	2	2	1 ¹ /2	1
R_{LB2S}		2,25	6,4	6,4	6,4

Legende:

¹ nur in Spartan-6 FPGAs

Bei den Slice-Informationen wird die Anzahl der Slices pro CLB sowie die unterschiedlichen Slice-Typen aufgeführt. Die Part-Informationen beschreiben die Anzahl der Parts

pro Slice und führt die internen Namen der Parts auf. Für die Informationen der LUT sind die Anzahl der LUT pro Slice und Part sowie die jeweilig unterstützten LUT-Typen und die Anzahl an Ausgängen angegeben. Die Register-Informationen enthalten die Anzahl der Register pro Slice und die unterstützten Register-Typen. Ebenfalls wird die Anzahl der arithmetischen Gatter und dessen Typen sowie die Anzahl der Carry Chains in einem CLB dargestellt. Die letzte Zeile stellt den Parameter R_{LB2S} für jede FPGA-Familie gegenüber.

Die größte strukturelle Veränderung ist bei dem Generationswechsel von Virtex-4 zu Virtex-5 erkennbar, in der die Anzahl der Slices und Parts pro CLB getauscht und die Implementierung zweier LUT5 oder einer LUT6 in einem Funktionsgenerator ermöglicht wurde. Der Generationswechsel von Virtex-5 zu Virtex-6 fügt zusätzliche Register für den LUT5-Ausgang oder den Bypass-Eingang ein. Weiterhin wird die Implementierung der ersten Register als Zweitorlogikgatter (Und- und Oder-Gatter, *AO2L*) ermöglicht. In den Spartan-6 FPGAs wird die Virtex-6-Slice um die Carry Chain- und Multiplexer-Funktionalität reduziert und bildet somit einen neuen Slice-Typen, Slice-X, welcher in jedem CLB enthalten ist.

Ab der UltraScale-Familie besteht ein CLB nur noch aus einer Slice, sodass die Verwendung des Namens nur noch historisch begründet ist. In [223, S. 11 f.] führt Xilinx die Unterschiede zur CLB-Architektur der 7-Serie auf. Wie in der Tabelle dargestellt, ist die Anzahl der internen Ressourcen einer UltraScale-Slice identisch zu zwei 7-Serie-Slices. Neu sind vier Clock Enable Signale pro CLB, sodass Clock Gating effizienter eingesetzt werden kann. Durch die Verwendung einer Slice pro CLB sinkt gleichzeitig die Anzahl der Carry Chain auf 1. Durch die Verdopplung der Datensignalleitungsweite wird jedoch eine identische Übertragung ermöglicht.

FPGA-Ressourcen

Eine Gegenüberstellung der FPGA-Ressourcen der einzelnen FPGA-Familien erfolgt in Tabelle 3.2. Zusätzlich sind fertigungsspezifische Angaben, die Strukturgröße und das Fertigungsjahr, dargestellt. Grundlage für das Fertigungsjahr sind die Jahresberichte (*Form 10-K - Annual report*) von Xilinx aus den Jahren 2011 ([195], Virtex-4) und 2016 ([196], ab Virtex-5). Die Anzahl der Slices kann für eine Gegenüberstellung der Xilinx FPGA-Familien aufgrund der zuvor beschriebenen Unterschiede in der Slice-Architektur nicht als Parameter für die Logikressourcen verwendet werden. Ein Vergleich ist jedoch über die Einheit Logic Cell möglich. Der FPGA-interne Speicher ist in LUTRAM und BRAM aufgeteilt.

Abbildung 3.8 stellt exemplarisch die tabellarisch aufgeführten Werte für die Logic Cells, LUTRAM und BRAM als Diagramm dar. Auffällig ist der Unterschied der Logic Cells und des BRAM-Speichers im letzten Generationssprung. Die geringere BRAM-Größe lässt sich hierbei durch die Hinzunahme des URAMs erklären. Durch diesen Rückgang ergibt sich bei Betrachtung aller FPGA-Familien das beste Bestimmtheitsmaß R^2 durch Verwendung potenzieller Trendlinien. Werden hingegen die FPGA-Familien bis

Tabelle 3.2: Gegenüberstellung der aktuellen Xilinx FPGA-Familien unter Betrachtung der maximal verfügbaren FPGA-Ressourcen

		Virtex-4 [230]	Virtex-5 [235]	Virtex-6 [240]	Spartan-6 [219]	7-Serie [241]	UltraScale [222]	UltraScale+ [222]
Fertigung	Strukturgr. Jahr	90 nm 2004	65 nm 2006	45 nm 2009	45 nm 2009	28 nm 2010	20 nm 2013	16 nm 2015
Logik (in Tsd)	Slices	89	52	119	23	305	317	216
	Logic Cells	200	332	759	147	1955	5541	3780
	Flip-Flops	178	207	948	184	2443	5066	3456
Speicher (in kbit)	LUTRAM	1,4	4,1	8,1	1,3	21,0	28,7	48,3
	BRAM	9,7	18,1	37,4	4,7	66,1	132,9	94,5
	URAM	0	0	0	0	0	0	360
Hard-IP	DSP	512	1056	2016	180	3360	5520	12288
	PCI Express	0	4	4	1	4	6	4
	Ethernet MAC	4	8	4	0	0	9	12
	RocketIO/GT	24	48	48	8	96	64	128

zur UltraScale-Familie betrachtet, ergeben exponentielle Trendlinien teils ein deutlich besseres Bestimmtheitsmaß von nahezu 1 (LUTRAM = 0,999 und Logic Cells = 0,985).

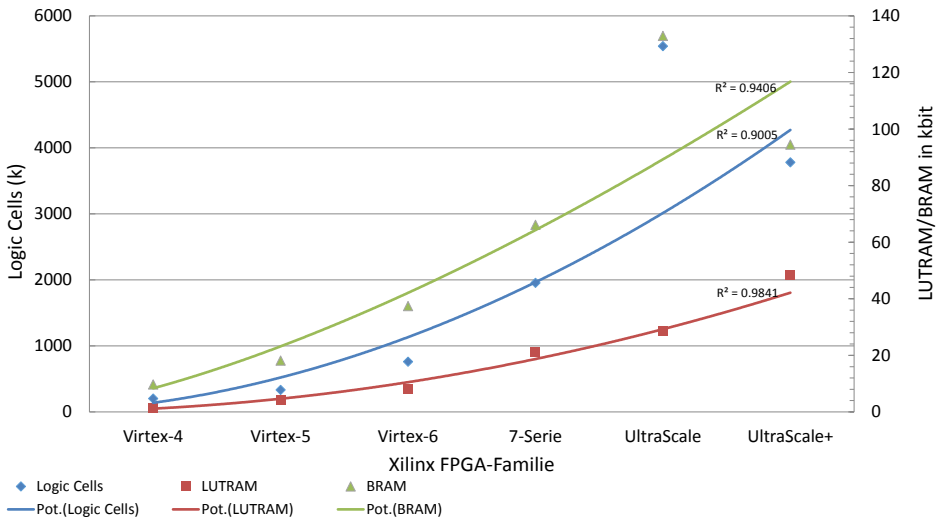


Abbildung 3.8: Entwicklung der FPGA-Ressourcen bei den Generationswechseln

3.3 Konfiguration eines Xilinx FPGAs

Die DPR stellt eine besondere Konfigurationsart eines FPGAs dar. Um die Unterschiede zu der Standard-Konfiguration verstehen zu können, wird in diesem Abschnitt zunächst auf den Konfigurationsspeicher, den existierenden Konfigurationsschnittstellen und den Konfigurationsablauf eingegangen. Anschließend wird die DPR von Xilinx FPGAs vor- und die Besonderheiten und Unterschiede herausgestellt.

Die Grundlagen zu der Konfiguration von Xilinx FPGAs werden in den *Configuration User Guides* der jeweiligen FPGA-Familie beschrieben: Virtex-4 [228], Virtex-5 [232], Virtex-6 [239], 7-Serie [193] und UltraScale [224]. Die folgenden Abschnitte fassen die Gemeinsamkeiten der genannten FPGA-Familien zusammen. Sollten Unterschiede innerhalb der FPGA-Familien existieren, so werden diese explizit textuell herausgestellt.

3.3.1 Konfigurationsspeicher

Eine Konfigurationsdatei besteht neben den eigentlichen FPGA-Konfigurationsdaten aus einem Header, Konfigurationsanweisungen für den Konfigurationsprozessor und einem Footer. Grundlagen zu dem Aufbau können in Anhang A.1 nachgeschlagen werden. Eine grafische Übersicht einer Xilinx Konfigurationsdatei ist dort in Abbildung A.1 dargestellt. Die Konfiguration eines Xilinx FPGAs ist in einem SRAM-basierten Konfigurationsspeicher abgelegt. Dieser Konfigurationsspeicher ist in sogenannte *Configuration Frames* organisiert. Ein solcher *Configuration Frame* ist die atomare Konfigurationseinheit und entspricht dem kleinsten adressierbaren (Speicher-) Segment mit einer, (seit Virtex-4) pro FPGA-Familie, festgelegten Anzahl an Bits. Ein *Configuration Frame* spannt in der Höhe immer über eine Taktregion. Die Höhe einer Taktregion variiert FPGA-Familien-spezifisch. Orientiert man sich an den CLBs, kann man allgemein für die aktuellen Xilinx-FPGA-Familien einen ansteigenden Trend bei der Anzahl der CLBs erkennen. In Tabelle 3.3 ist die Entwicklung des *Configuration Frames* durch die Anzahl an Bits, (32-Bit) Wörtern und CLBs dargestellt.

Tabelle 3.3: Entwicklung des *Configuration Frames* für aktuelle Xilinx FPGA-Familien

FPGA-Fam.	Configuration Frame		
	in bit	in Wörtern	in Anzahl CLBs
Virtex-4	1 312	41	16
Virtex-5	1 312	41	20
Virtex-6	2 592	81	40
7-Serie	3 232	101	50
UltraScale	3 936	123	60
UltraScale+	2 976	93	60

In Abbildung 3.9 ist der Inhalt eines *Configuration Frames* für eine Virtex-4 FPGA dargestellt. Diese FPGA-Architektur wurde gewählt, da ein Virtex-4 FPGA in der späteren Evaluierungsplattform verwendet wird Abschnitt 6.1. Auf der linken Seite wird die Reihenfolge in der Konfigurationsdatei (*Bitstream*) und auf der rechten Seite eine grobe Gliederung der einzelnen Bits innerhalb der Wörter visualisiert. Mehrere *Configuration Frames* wiederum enthalten die Konfigurationsinformation für eine Spalte innerhalb einer Taktregion. Diese Anzahl der *Configuration Frames* in der *Breite* richtet sich nach dem, in der Spalte vorliegenden, Basisblocktyp.

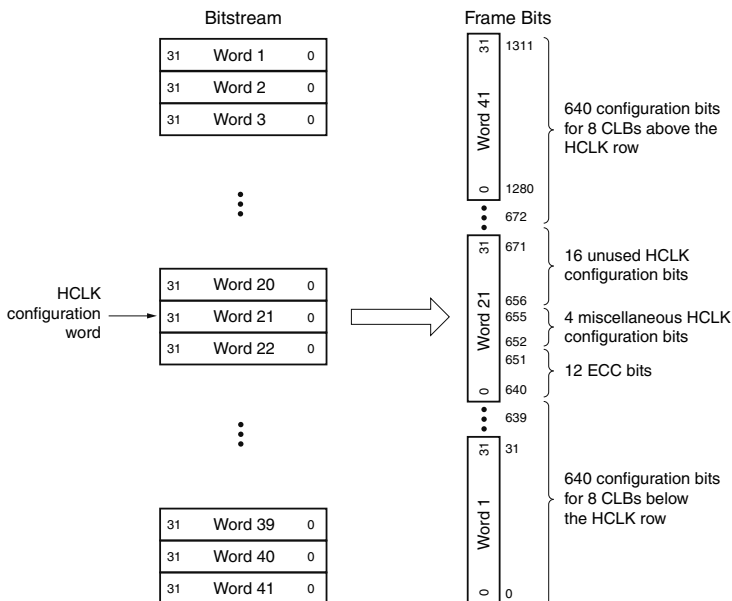


Abbildung 3.9: Aufteilung eines Xilinx Virtex-4 Configuration Frames [35, S. 2]

Tabelle 3.4 führt die Anzahl der *Configuration Frames* für die unterschiedlichen Basisblöcke auf. Diese Daten sind nicht in den *Configuration Guides* für die einzelnen FPGAs aufgeführt, sondern wurden aus den Software-Treibern des HWICAP-IP-Cores (Version 8.01) für Xilinx EDK extrahiert bzw. für den Virtex-6 FPGA aus [110] entnommen. Angaben zu den aktuellen FPGA-Familien sind nicht verfügbar. Für ein Virtex-4-FPGA bedeutet dies: Eine Zelle, eine Spalte innerhalb einer Taktregion, mit 16 CLBs wird aus 22 *Configuration Frames*, eine IOB-Zelle hingegen aus 30 *Configuration Frames* aufgebaut.

Die Adressierung der einzelnen *Configuration Frames* erfolgt über die sogenannte *Frame Address*. Diese ist in fünf Abschnitte unterteilt: *Block Type*, *Top/Bottom*, *Row*

Tabelle 3.4: Anzahl der *Configuration Frames* pro Basisblock für einige Xilinx FPGA-Familien

FPGA-Fam. / Basisblocktyp	Virtex-4	Virtex-5	Virtex-6
CLB	22	36	36
IOB	30	54	44
DSP	21	28	28
GCLK	3	4	38
MGT/GTX	20	32	30
BRAM_Interconnect	20	30	28
BRAM_Content	64	128	128
Overhead	2	2	2
BRAM_Overhead	2	2	2

Address, *Column Address* und *Minor Address*. Bei den Xilinx FPGA-Familien ergeben sich dabei Unterschiede in der (Bitweisen) Aufteilung dieser fünf Abschnitte, welche in Tabelle 3.5 dargestellt sind.

Tabelle 3.5: (Bitweise) Kodierung der *Configuration Frame Address* für aktuelle Xilinx FPGA-Familien

FPGA-Fam. / Abschnitt	Virtex-4	Virtex-5/-6	7-Serie	UltraScale (+)
Block Type	21-19	23-21	25-23	25 - 23
Top/Bottom	22	20	22	-
Row Address	18-14	19-15	21-17	22-17
Column Address	13-6	14-7	16-7	16-7
Minor Address	5-0	6-0	6-0	6-0

Die Blocktypen variieren von FPGA-Familie zu FPGA-Familie. Typischerweise enthält der erste Blocktyp alle wesentlichen Basisblocktypen, wie CLBs oder IOBs und deren Verbindungsmatrizen. Für den Inhalt des Basisblocks Block-RAM existiert in jeder FPGA-Familie ein separater Blocktyp. Details zu den konkreten Blocktypen sind in den einzelnen *Configuration User Guides* aufgeführt. Bis zur 7-Serie gilt FPGA-Familienübergreifend, dass ein Xilinx FPGA in zwei Hälften (Top/Bottom) aufgeteilt ist. Hervorzuheben ist der Tausch der Position des Blocktyps und des Top/Bottom-Bits von Virtex-4 zu den folgenden FPGA-Familien. Ab der UltraScale-Familie erfolgt die Adressierung ohne eine Aufteilung in einen oberen und unteren Bereich. Die Row Address gibt die aktuelle Zeile innerhalb des FPGAs an. Wie bereits beschrieben, spannt ein Configuration Frame immer über eine Taktregion, sodass sich die Zeilenadresse ebenfalls nach

diesem Schrittmaß richtet. Die Column Address adressiert die aktuelle Spalte und die Minor Address den aktuellen Configuration Frame innerhalb der Gesamtkonfiguration eines Basisblocks.

Die Frame Adresse kann durch einen Schreibzugriff auf das interne Register *FAR* des Xilinx Paket-Prozessors direkt gesetzt werden oder sie wird automatisch am Ende eines *Configuration Frames* erhöht. In einer vollständigen Konfigurationsdatei wird die Adresse zu Beginn auf 0 gesetzt und anschließend automatisch inkrementiert.

Abbildung 3.10 visualisiert eine Aufteilung des Virtex-4 FX100 FPGAs in die fünf unterschiedlichen Abschnitte der *Frame Address*¹. Dieses FPGA wird in der späteren Evaluierungsplattform verwendet Abschnitt 6.1. Die dargestellten Blöcke sind in vier Blocktypen eingeteilt. Die einzelnen Basisblocktypen sind farblich kodiert und spannen teilweise über mehrere Spalten, wie in der Column Address angedeutet ist. Das Maximum der Basisblock-spezifischen Minor Address entspricht den Werten aus Tabelle 3.4. Die mit X markierte Zelle entspricht der Startadresse 0. Diese Adresse zeigt auf den Block in der ersten Spalte der ersten Taktregion der oberen Hälfte des FPGAs.

		Block Type														1		2		6		
		Column Address														0-9		0-9		0-9		
		Minor Address														0-19		0-63		0-1		
		MGT	CLB	IOB	CLB	DSP	CLB	IOB	CLK	CLB	DSP	CLB	IOB	CLB	MGT	Over-head	BRAM Int	Over-head	BRAM	Over-head	BRAM Data Integrity	Over-head
Top/Bottom	Row Address																					
0	4																					
	3																					
	2																					
	1																					
	0	X																				
1	0																					
	1																					
	2																					
	3																					
	4																					

Abbildung 3.10: Darstellung der *Frame Address* anhand des Xilinx Virtex-4 FX100 (basierend auf [35])

Konfiguration des FPGAs durch inkrementieren der Frame Address

Wie in Tabelle 3.5 dargestellt, definieren die unteren Bits die *Minor Address*. Diese Adresse zeigt auf ein Configuration Frame innerhalb der Gesamtkonfiguration eines Basisblocks. Wird die Anzahl der *Configuration Frames* für den aktuellen Basisblock, wie in Tabelle 3.4 dargestellt, erreicht, wird mit der nächsten Spalte fortgefahren. Dies führt zur Erhöhung der *Column Address*. Nachdem alle Spalten eines FPGAs beschrieben wurden, wird die *Row Address* inkrementiert. Dies bedeutet einen Sprung in die nächste Taktregionen. Für FPGAs bis zur 7-Serie wird nach der letzten Taktregion in der oberen Hälfte des FPGAs das *Top/Bottom Bit* gesetzt und damit nach dem gleichen Schema der

¹Die Darstellung basiert auf der Excel-Tabelle aus den Referenz-Dateien aus [35]. Die Daten und die Visualisierung für das aufgeführte FPGA wurden eigenständig erarbeitet.

untere Teil des FPGAs - beginnend mit der Mitte des FPGAs - konfiguriert. Nachdem auch die untere Hälfte des FPGAs durchlaufen wurde, ist der vollständige FPGA für einen *Blocktyp* beschrieben. Der Blocktyp wird daher geändert (erhöht) und anschließend der gesamte FPGA abermals nach dem gleichen Prozedere durchlaufen. Dies geschieht für alle unterstützten Blocktypen.

3.3.2 Konfigurationsschnittstellen

Xilinx stellt externe und interne Konfigurationsschnittstellen zur Verfügung. Zur externen Konfiguration werden Schnittstellen bzw. Protokolle in verschiedenen Modi mit unterschiedlicher Datenbreite (in Klammern dargestellt) angeboten:

1. Serielle Konfiguration im Master- oder Slave-Modus (1)
2. Parallele Konfiguration mit SelectMAP im Master-(8, 16) oder Slave-(8, 16, 32) Modus
3. JTAG² / Boundary-Scan Konfiguration (1)
4. SPI³ Master Flash Konfiguration (1, 2, 4) ab Virtex-5
5. BPI⁴ Master Flash Konfiguration (8, 16) ab Virtex-5

Der Master- oder Slave-Modus ist durch die Richtung des Konfigurationstakts CCLK festgelegt. Im Master-Modus wird der Takt von einem FPGA-internen Oszillator erzeugt; CCLK ist daher ein Ausgang. Im Slave-Modus hingegen wird der Konfigurationstakt CCLK als Eingang verwendet. Die beiden letztgenannten Modi sind mit FPGAs ab der Xilinx Virtex-5-Familie möglich und erlauben eine Konfiguration durch einen angeschlossenen Flash-Speicher ohne einen zusätzlichen externen Controller.

Für die interne Konfiguration wird die ICAP⁵-Schnittstelle angeboten. Diese interne Schnittstelle entspricht von der Signalgebung und Funktionalität der SelectMAP-Schnittstelle. Einzig die in SelectMAP bidirektionalen Datenleitungen sind in der ICAP-Schnittstelle in separate Lese- und Schreibdatenbus aufgeteilt. Für die Zynq-Familie wird zusätzlich die PCAP⁶-Schnittstelle zum integrierten ARM-Prozessor angeboten.

Signalbelegung der SelectMAP- und ICAP-Schnittstelle

Die SelectMAP- und die ICAP-Schnittstelle wurden in Rahmen dieser Dissertation für die Implementierung eines externen sowie internen Rekonfigurationskontroller verwendet (siehe Abschnitt 6.2.2 und Abschnitt 6.2.1). Daher wird in diesem Unterkapitel diese Konfigurationsschnittstelle genauer vorgestellt. Wie bereits erwähnt, ist die Signalgebung und Funktionalität beider Schnittstellen identisch. Bei der externen SelectMAP-Schnittstelle können einige Ein- und Ausgänge nach der statischen Rekonfiguration eine andere Funktion übernehmen. Sollen die Ein-/Ausgänge ihre Funktion behalten, muss dies explizit in der UCF (engl. User Constraint File) durch die Bedingung

²JTAG: Joint Test Action Group

³SPI: Serial Peripheral Interface

⁴BPI: Byte Peripheral Interface

⁵ICAP: Internal Configuration Access Port

⁶PCAP: Processor Configuration Access Port

CONFIG_MODE und bei der Erzeugung der Konfigurationsdatei durch die Xilinx Software *BitGen* durch die Option *-g persist* deklariert werden. Wird die SelectMAP- oder ICAP-Schnittstelle in einer 8-Bit Konfiguration verwendet, sind die Bits innerhalb eines Bytes gedreht. In Abbildung 3.11 ist die Signalbelegung für die in dieser Dissertation untersuchten Xilinx FPGA-Familien dargestellt.

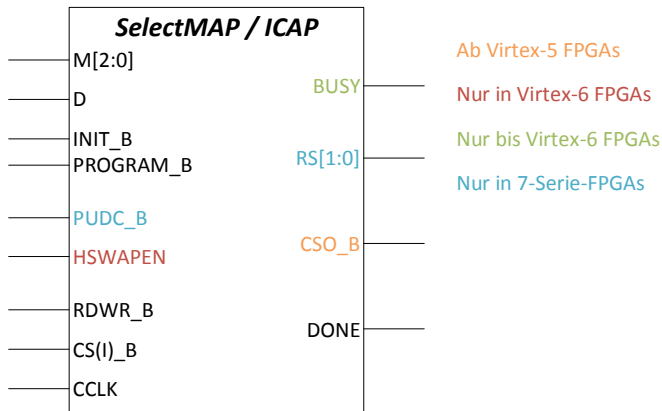


Abbildung 3.11: Signalbelegung der SelectMAP- und ICAP-Schnittstelle

Eine Beschreibung und konkrete Typ-Zuordnung zu den einzelnen Ein- und Ausgängen ist in Tabelle 3.6 dargestellt. Das Signal RDWR_B darf den Pegel nur ändern, wenn das Signal CS(I)_B nicht gleichzeitig gesetzt ist. Ansonsten wird die Rekonfiguration abgebrochen (engl. abort). Die Tabelle A.4 in Anhang A.3 fasst die Kodierung des Konfigurationsmodus für die untersuchten FPGA-Familien zusammen.

Konfigurationsdatenraten und -zeiten

Die maximalen Konfigurationsdatenraten für die genannten Konfigurationsschnittstellen sind in Tabelle 3.7 dargestellt. Der maximale Takt bei SPI und BPI hängt zum einen von der gewählten Taktquelle (intern über CCLK mit maximal 60 MHz oder extern über EMCCLK ab 7-Serie) und zum anderen von der Setup-Time des FPGAs sowie dem maximal unterstützten Takt des Flash- oder PROM-Bauteils (ca. 90 MHz). Eine konkrete Formel zur Bestimmung der maximalen Konfigurationsgeschwindigkeit wird für SPI in [225, S. 9] und BPI in [199, S. 10] beschrieben.

Die Konfigurationszeit hängt neben der Konfigurationsdatenrate nur von der Größe der Konfigurationsdatei ab. Exemplarisch sind in Tabelle 3.8 die Größe des Konfigurationsspeichers und daraus resultierende Konfigurationszeiten unter Verwendung der jeweiligen unterstützten Konfigurationsschnittstellen für den jeweils kleinsten und größten FPGA der untersuchten FPGA-Familien dargestellt. Da zu der Konfigurationsdatei neben den eigentlichen FPGA-Konfigurationsdaten ein Header, Konfigurationsanwei-

Tabelle 3.6: Ein- und Ausgänge der Konfigurationsschnittstelle SelectMAP/ICAP

Pin-Name	Typ	Beschreibung
M[2:0]	I	Kodierung des Konfigurationsmodus
D	3-State I/O	Konfigurationsdatenleitungen
INIT_B	I/O	Vor Konfiguration: Verzögerung der Konfiguration Nach Konfiguration: CRC-Fehler (0 = Fehler)
PROGRAM_B	I	Zurücksetzen der Konfiguration des FPGAs
PUDC_B	I	Pull-Up During Configuration (nur 7-Serie) (0 : interne Pull-Up Register werden aktiviert) (1 : interne Pull-Up Register werden deaktiviert)
HSWAPEN	I	Pull-Up During Configuration (nur Virtex-6)
RDWR_B	I	Datenrichtung des SelectMAP-Datenbusses (0 = Eingang / Schreiben)
CS(I)_B	I	Aktivierung des SelectMAP-Datenbusses (0 = aktiviert)
CCLK	I/O	Konfigurationstakt
BUSY	3-State O	Signalisiert, dass der FPGA keine Konfigurationsdaten zurücklesen kann (engl. readback)
RS[1:0]	O	Revision Select (ConfigFallback bei 7-Serie)
CSO_B	O	Chip Select (nur für die Konfiguration in einer parallelen Daisy-Chain)
DONE	I/O	Konfigurationszustand des FPGAs (1 = FPGA konfiguriert)

Tabelle 3.7: Maximale Konfigurationsdatenraten für die Konfigurationsschnittstellen eines Xilinx FPGAs

Konfigurations- schnittstelle	Max. Takt in MHz	Max. Datenbreite in bit	Konfigurationsdatenrate in Mbit/s
JTAG	66	1	66
Seriell	100	1	100
SPI	66–90	4	264–360
BPI	66–90	16	1 056–1 440
SelectMAP	100	32	3 200
ICAP	100	32	3 200

sungen und ein Footer gehört, handelt es sich bei den Angaben um Approximationen. Für den größten FPGA der UltraScale+-Familie (XC7VU13P) sind bis zum Zeitpunkt der Fertigstellung dieser Dissertation keine Informationen über die Größe der Konfigurationsdatei verfügbar.

Tabelle 3.8: Approximierte Konfigurationszeiten für den jeweils kleinsten und größten FPGA der untersuchten FPGA-Familien (mit Ausnahme UltraScale+)

FPGA-Familie	FPGA-Typ	Konfigurations-speichergröße in bit	JTAG in s	Seriell in s	SPI in s	BPI in s	SelectMAP/ICAP in s
V4	FX12	4 723 200	0,068	0,045	-	-	0,001
	LX200	51 325 440	0,742	0,489	-	-	0,015
V5	LX20T	6 251 200	0,090	0,060	0,066	0,006	0,002
	LX330	79 704 832	1,152	0,760	0,845	0,072	0,024
S6	LX4	2 731 488	0,039	0,026	0,029	0,002	0,001
	LX150T	33 909 664	0,490	0,323	0,359	0,031	0,010
V6	LX75T	26 239 328	0,379	0,250	0,278	0,024	0,008
	LX760	184 823 072	2,671	1,763	1,958	0,167	0,055
7S	7A15T	17 536 096	0,253	0,167	0,046	0,012	0,005
	7V2000T	447 337 216	6,464	4,266	1,185	0,296	0,133
US	KU025	128 055 264	1,850	1,221	0,339	0,085	0,038
	VU440	1 031 731 104	14,908	9,839	2,733	0,683	0,307
US+	XCKU3P	123 432 576	1,784	1,177	0,327	0,082	0,037

3.3.3 Konfigurationsablauf

Der Konfigurationsablauf ist für alle Xilinx FPGAs identisch und unterteilt sich wie in Abbildung 3.12 dargestellt in 8 einzelne Schritte, welche wiederum in die 3 Abschnitte *Setup*, *Bitstrom Loading* und *Startup* gruppiert sind. Dieser Abschnitt fasst den Konfigurationsablauf kurz zusammen (Details können den *Configuration User Guides* der FPGAs entnommen werden).

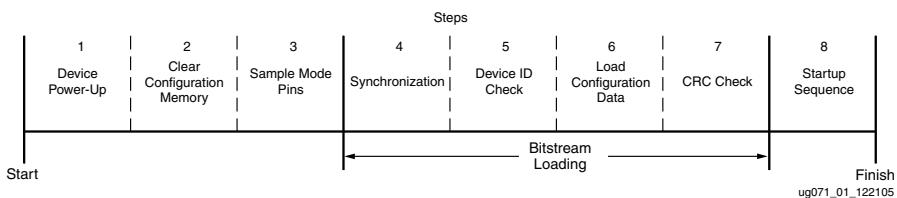


Abbildung 3.12: Konfigurationsablauf eines Xilinx FPGAs [228, S. 15]

Setup In dem ersten Schritt *Device Power-Up* wird auf stabil anliegende Versorgungsspannungen gewartet. Anschließend wird im Schritt *Clear Configuration Memory* der Konfigurationsspeicher gelöscht. Der Löschvorgang ist durch die Verwendung eines SRAM-basierten Konfigurationsspeichers notwendig, da initial undefinierte Speicherinhalte existieren und damit eine undefinierte Konfiguration für den FPGA vorliegt. Das Löschen kann neben einem Neustart auch durch externe Konfigurationsschnittstellen forciert werden (per SelectMAP mit dem Signal *PROGRAM_B* oder per JTAG mit der *JPROGRAM* Instruktion). Die Konfiguration kann weiterhin durch Setzen (auf 0) des Konfigurationssignals *INIT_B* verzögert werden. Ein Pegelwechsel auf diesem Konfigurationssignal hat den Wechsel zu dem letzten Setup-Schritt zur Folge. In *Sample Mode Pins* werden die Konfigurationssignale (Mode-Pins) für die Wahl der Konfigurationsschnittstelle (auch Modus genannt) abgetastet und im Master-Modus die CCLK getrieben. Ab diesem Zeitpunkt werden die Konfigurationsdaten von den Dateneingängen mit dem Konfigurationstakt abgetastet.

Bitstream Loading Das Laden der Konfiguration in den Konfigurationsspeicher beginnt mit dem *Synchronization*-Schritt, der Erkennung des Synchronisationsworts *0xAA995566*. Die Konfigurationsdaten können aufgrund des Headers in der Konfigurationsdatei nicht-Wort-ausgerichtet sein. Die interne Konfigurationslogik erkennt diese Verschiebung anhand des Synchronisationsworts und ordnet die Konfigurationsdaten entsprechend um. Anschließend wird im Schritt *Device ID Check* die in den Konfigurationsdaten übermittelte Geräte-ID überprüft. Dies stellt sicher, dass ein FPGA nur mit für den FPGA passenden Konfigurationsdaten beschrieben wird. Ein Virtex-4 LX15 lässt sich also nicht mit den Konfigurationsdaten eines Virtex-4 LX80 konfigurieren. Das Synchronisationswort und die Geräte-ID sind ein wesentlicher Bestandteil der Konfigurationsdatei (Abbildung A.1 in Anhang A.1). Nach der Überprüfung des FPGAs wird in den Schritt *Load Configuration Data* übergegangen, in dem die Konfigurationsdaten, aufgeteilt in die beschriebenen Configuration Frames in den Konfigurationsspeicher übertragen werden. Während dieser Datenübertragung wird eine zyklische Prüfsumme (CRC) berechnet. Die Software zur Erstellung der Konfigurationsdatei (*BitGen*) berechnet typischerweise bei der Erstellung ebenfalls eine CRC-Prüfsumme. Dieser Wert sowie ein Befehl zur Überprüfung der CRC-Prüfsumme steht am Ende der Konfigurationsdatei. Stimmen die beiden Werte nicht überein, wird das Konfigurationssignal *INIT_B* gesetzt und die Konfiguration abgebrochen. Die Überprüfung durch den CRC kann deaktiviert werden, was zur Folge hat, dass korrupte/verfälschte Daten auf den FPGA geladen werden können. Die Deaktivierung erfolgt durch Schreiben des *Magic-Words* *0x0000DEFC* an das CRC-Konfigurationsregister.

Startup Nachdem die Konfigurationsdaten im Konfigurationsspeicher geladen wurden, wird durch einen Befehl am Ende der Konfigurationsdatei die Start-Sequenz des FPGAs initialisiert. Die Start-Sequenz wird durch einen Zustandsautomaten mit maximal sieben Zuständen bestimmt. Die einzelnen Phasen und deren Ereignis sind in der Tabelle 3.9

aufgeführt und können in der Konfigurationsdatei, über die Software *BitGen*, (de-) aktiviert und zu einem Ablaufplan angeordnet werden.

Tabelle 3.9: Mögliche Startup-Phasen während der Konfiguration

Phase	Voreinst.	Ereignis
1-6	-	Warten auf Lock der MMCM ¹ - oder DCM ² -Komponenten
1-6	-	Warten auf Übereinstimmung der DCI ³ -Komponenten
1-6	4	Freistellung des DONE-Ausgangspins
1-6	5	Negierung von GTS (Global 3-State), Aktivierung der Ein-/Ausgänge
1-6	6	Setzen von GWE (Global Write Enable), Zustandsänderungen in RAM und Register
7	7	Setzen von EOS (End Of Startup)

Legende:

¹ MMCM: Mixed-Mode Clock Manager

² DCM: Digital Clock Manager

³ DCI: Digitally Controlled Impedance

Die Reihenfolge in dem Ablaufplan kann bis auf die siebte Phase beliebig festgelegt werden. Die ersten beiden Phasen sind optional. In der Voreinstellung erfolgt zuerst die Freistellung des DONE-Ausgangspins, was, an den FPGA angeschlossenen Komponenten, die erfolgreiche Konfiguration des FPGAs signalisiert. In der nächsten Phase werden alle weiteren Ein- und Ausgänge freigeschaltet. Anschließend werden Zustandsänderungen für die internen Speicher (BlockRAM, Block-interne Register und LUTRAM) aktiviert, wodurch das geladene Design funktionsbereit ist. Die einzelnen Phasen der Voreinstellung haben in der Regel eine Dauer von einem Takt. Bei FPGAs der 7-Serie empfiehlt Xilinx jedoch weitere 24 Takte nach Setzen des DONE-Signals zu warten, um das erfolgreiche Erreichen der siebten und letzten Phase sicherzustellen.

3.3.4 Dynamisch partielle Rekonfiguration eines Xilinx FPGAs

Die dynamisch partielle Rekonfiguration ist bei Xilinx prinzipiell bereits seit dem Jahr 1996 mit der XC6200-Familie, möglich [247]. Zum Zeitpunkt dieser Dissertation werden hardwareseitig die Virtex-4-, Virtex-5-, Virtex-6- und Virtex-7-Familie (außer in SSI⁷-Technologie gefertigte FPGAs), die Kintex-7-, Artix-7- und die Zynq-7000-Familie (alle in [211] beschrieben) sowie die UltraScale-Familie unterstützt [244]. Die Spartan-3- und -6-Familie unterstützen mit Einschränkungen hardwareseitig die DPR. In diesem Abschnitt werden die Besonderheiten im Bezug auf die Hardware, den Konfigurationsprozess sowie allgemeine Richtlinien für den Entwurf eines DPR-Systems aufgeführt.

⁷SSI: Stacked Silicon Interconnect

Hardwareseitige Besonderheiten

Xilinx weist in [211] auf einige, teils FPGA-Familien-spezifische, hardwareseitige Besonderheiten hin.

Non-Glitching Die zuvor erwähnte Einschränkung für Spartan-FPGAs betrifft die sogenannte *Non-Glitching* Technologie, welche für die offiziell unterstützten Virtex und die als 7-Serie (Artix-7, Kintex-7 und Virtex-7) benannten FPGAs sowie Zynq-7000 (basierend auf Artix-7 und Kintex-7) verwendet wird [210]. Diese stellt sicher, dass es durch die DPR während der Überschreibung von statischer Logik innerhalb eines rekonfigurierten Configuration Frames nicht zu *Glitches* kommen kann [211, S. 112].

Verschlüsselung Als integrierte Lösung bieten die FPGAs ab der Virtex-6-Familie, sowie als Makro für den Virtex-5 erhältlich, eine Verschlüsselung im AES (engl. Advanced Encryption Standard) an. Hierbei muss die Datenbreite der Konfigurationsschnittstellen allerdings reduziert werden.

Zyklische Redundanzprüfung Zur Überprüfung von Fehlern im partiellen Bitstrom kann eine CRC-Überprüfung durchgeführt werden. Die 7-Serie unterstützt dabei einen *Frame-by-Frame* CRC, sodass nur korrekte Configuration Frames in den Konfigurationsspeicher geladen werden [211, S. 99]. Für ältere Familien kann für jedes partielle Modul ein eigener CRC-Wert in den Bitstrom aufgenommen werden. Mithilfe eines zusätzlichen CRC-Checkers kann die Konfigurationsdatei vor dem Laden auf Fehler geprüft werden. Ein Xilinx IP-Core für den CRC ist geplant, bis dato allerdings noch nicht verfügbar, sodass der Anwender auf eine eigene Implementierung angewiesen ist.

Besonderheiten bei und Hinweise zu dem Konfigurationsprozess

Bei dem Konfigurationsprozess sind ebenfalls einige Abweichungen gegenüber den zuvor beschriebenen Konfigurationseigenschaften zu beachten. Folgend wird daher auf den Konfigurationsablauf einer dynamisch partiellen Rekonfiguration eingegangen, die unterstützten Konfigurationsschnittstellen sowie weitere DPR-spezifische Konfigurationsdetails aufgeführt.

Konfigurationsablauf

Der Konfigurationsablauf einer dynamisch partiellen Rekonfiguration unterscheidet sich von dem Ablauf der statischen Rekonfiguration. Ein vollständiger Bitstrom enthält, wie in Abschnitt 3.3.3 beschrieben, einen Header, Konfigurationsanweisungen, Konfigurationsdaten und eine CRC-Prüfsumme zur Fehlerüberprüfung. Der FPGA befindet sich während der Konfiguration im Konfigurationsmodus (engl. Configuration Mode) und wechselt nach erfolgreicher CRC-Überprüfung in den Benutzermodus (engl. User Mode). Dies wird dem Benutzer durch das externe *DONE*-Signal signalisiert. Eine negative Überprüfung der CRC-Prüfsumme führt dazu, dass die Konfiguration des FPGA nicht aktiv und somit der Benutzermodus nicht erreicht wird. Das *DONE*-Signal

wird in diesem Fall ebenfalls nicht gesetzt, sodass ein Konfigurationscontroller diese Fehlkonfiguration erkennen kann.

Konfigurationsschnittstellen

Die dynamisch partielle Rekonfiguration kann von fast allen Konfigurationsschnittstellen durchgeführt werden. Extern können die SelectMAP-, die serielle oder JTAG-Schnittstelle, intern die ICAP-Schnittstelle verwendet werden. Bei der Zynq-Familie kann extern lediglich die JTAG-Schnittstelle und intern die ICAP- oder PCAP-Schnittstelle verwendet werden. Wie bereits beschrieben, müssen für die externen SelectMAP- und Seriell-Schnittstellen einige Ein-/Ausgänge reserviert werden, sodass diese nach der Konfiguration weiterhin aktiv bleiben. Dies geschieht zum einen durch Verwendung einer Randbedingung (engl. constraint) `CONFIG_MODE`, welche in einer Datei (User Constraint File, UCF) eingetragen wird und zum anderen durch eine *BitGen*-Option (`-g persist`) [211, S. 93 ff.].

Die dynamisch partielle Rekonfiguration wird hingegen ausschließlich im Benutzermodus durchgeführt. Der FPGA wurde somit schon durch eine statische Rekonfiguration in den Benutzermodus versetzt. Der nicht von der Konfiguration betroffene Teil des FPGAs führt die aktuelle Funktion weiter aus, während der rekonfigurierbare Teil modifiziert wird. Der partielle Bitstrom besteht aus einem Header, Konfigurationsanweisungen (hauptsächlich zum Setzen der *Frame-Adresse*), Konfigurationsdaten (Configuration Frames) und einer CRC-Prüfsumme. Der Header und die CRC-Prüfsumme können über *BitGen*-Einstellungen deaktiviert werden und sind daher optional. Wird der Header nicht erstellt, erhält die Konfigurationsdatei die Dateiendung *bin* (anstatt *bit*). Der entscheidende Unterschied zwischen der statischen und der dynamisch partiellen Rekonfiguration ist der aktive Modus. Durch den Benutzermodus wird zum einen die optionale CRC-Prüfsumme erst nach der Durchführung der Rekonfiguration überprüft. Ein defekter, partieller Bitstrom kann somit den FPGA beschädigen. Ob ein defekter Bitstrom geladen wurde, kann durch das Status Register (STAT) des Paket-Prozessors detektiert werden. Zum anderen muss zur Abschlusserkennung der Rekonfiguration der Transfer des partiellen Bitstroms zum Paket-Prozessor überwacht werden, da der erfolgreiche Abschluss nicht signalisiert wird.

In der Tabelle A.2 im Anhang A.2 ist die Sequenz für die dynamisch partielle Rekonfiguration eines Virtex-4 FPGAs dargestellt. Dies entspricht dem Inhalt einer partiellen Konfigurationsdatei. Im Vergleich zu der statischen Rekonfigurationssequenz werden deutlich weniger Konfigurationsanweisungen an den Paket-Prozessor geschickt. In der Tabelle A.3 erfolgt ferner eine Gegenüberstellung der Sequenzen einer statischen und einer partiellen Rekonfiguration.

Black-Box-Implementierung

Sogenannte *Black-Box*-Implementierungen für die dynamischen Komponenten enthalten lediglich die Kommunikationsinfrastruktur zum statischen Bereich und keinerlei FPGA-Ressourcen (Logik und Verdrahtung) der dynamischen Komponente [211, S. 114].

Die Konfiguration einer Black-Box entspricht einer leeren Konfiguration für einen dynamischen Bereich und wird daher als *Blanking Bitstream* bezeichnet. Die Verwendung der Black-Box-Implementierungen bringt zwei Vorteile. Zum einen kann eine Reduzierung der Größe einer vollständigen Konfigurationsdatei erfolgen, was zu einer Reduzierung der initialen Konfigurationszeit führt. Zum anderen wird die Leistungsaufnahme bei einem Wechsel von dynamischen Komponenten, also der dynamisch partiellen Rekonfiguration, durch Verwendung der Blanking Bitstreams reduziert. Auf die Verwendung der Blanking Bitstreams weist Xilinx indirekt bereits in der Einleitung (siehe Abschnitt 3.1) hin. In [20] wird der Unterschied zwischen einem direkten Überschreiben mit einer neuen dynamischen Komponente und einem vorherigen Löschen der Konfiguration durch einen Blanking Bitstream untersucht. Beckhoff et al. konnten nachweisen, dass es bei einem direkten Überschreiben zu temporären Kurzschlüssen kommen kann, welche zu einer erhöhten Leistungsaufnahme führt. Die Abbildung 3.13 stellt diesen Sachverhalt dar.

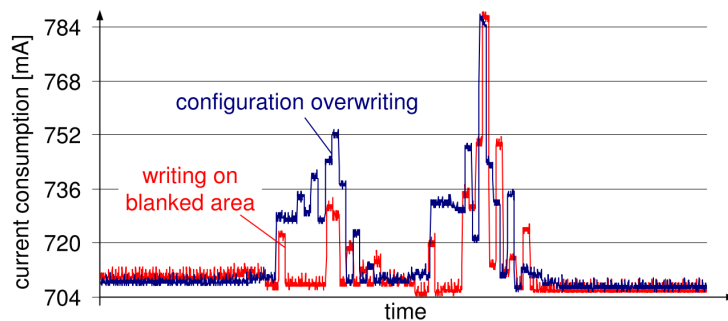


Abbildung 3.13: Unterschied in der Stromaufnahme zwischen einem direkten Überschreiben und einem vorherigen Löschen der Konfiguration [20, S. 5]

Bitstromkompression

Die Größe der Konfigurationsdatei – und damit auch die Konfigurationszeit – kann weiter verringert werden, indem eine Kompression bei der Erstellung der Konfigurationsdateien mit der Xilinx Software *BitGen* durchgeführt wird. Dadurch werden sich wiederholende Konfigurationsinformationen erkannt und nur einmalig in die Konfigurationsdatei geschrieben. Die Kompression wird in beiden, vollständigen und partiellen, Konfigurationsdateien unterstützt, muss allerdings explizit bei der Erzeugung mit der Software *BitGen* aktiviert werden [200, S. 229]. Einzig bei der Erstellung differenzbasierter Bitströme (siehe Abschnitt 3.5.2) erfolgt automatisch eine Kompression.

Allgemeine Richtlinien für den Entwurf

Bei der Entwicklung eines DPR-Systems sollten einige allgemeingültige Hinweise bezüglich der Design-Hierarchie, der Taktverteilung, der Entkoppelung und der Partitionierung des FPGAs in statische und dynamische Partitionen beachtet werden.

Statische und rekonfigurierbare Komponenten

Nicht alle initial konfigurierbaren Basisblöcke können nachträglich durch eine partielle Rekonfiguration des Konfigurationsspeichers geändert werden. Komponenten, die als statische Komponenten erstellt werden müssen, sind:

- Komponenten zur Modifikation des Taktes (MMCM, PLL und DCM)
- BUFG-Takt-Buffer/Speicher, BUFR-Takt-Buffer/Speicher können mit Einschränkungen verwendet werden (Details in [211, S. 106 ff.])
- Spezielle Gerätekomponenten wie BSCAN, STARTUP, FRAME_ECC oder ICAP

Um dennoch Änderungen an diesen statischen Komponenten vornehmen zu können, werden für viele dieser Komponenten DRP (engl. **D**ynamic **R**econfiguration **P**ort)-Schnittstellen angeboten. Hierdurch kann das Verhalten durch Änderung der Steuerregister der entsprechenden Komponenten während der Laufzeit geändert werden. Die Vorgehensweise und die unterstützten Komponenten sind im jeweiligen *Configuration User Guide* der FPGA-Familie beschrieben. In einem DPR-Bereich können folgende Komponententypen rekonfiguriert werden:

- Switch-Matrix (Routingressourcen)
- CLB/Slice (Logik: LUT, FFs, SRLs, RAMs, ROMs)
- BRAM/FIFO
- DSP
- IO (mit Einschränkungen)

Entkoppelung der rekonfigurierbaren Partitionen

Da die partielle Rekonfiguration im Betrieb durchgeführt wird, sollten Daten von rekonfigurierbaren Modulen während der Rekonfiguration ignoriert werden. Eine solche Entkopplung zwischen den statischen und den rekonfigurierbaren Partitionen kann durch eine aktivierbare (z. B. durch Chip Enable Signale) Registerstufe auf der statischen Seite erfolgen [211, S. 109]. Um einen definierten Startzustand in der rekonfigurierten Partition nach der Rekonfiguration zu erreichen, sollte ein lokales Reset die platzierte Schaltung zurücksetzen [211, S. 110].

Partitionierung des FPGAs

Die Partitionierung des FPGAs in statische und dynamische Bereiche erfolgt durch sogenannte *AREA_GROUP_RANGE* Randbedingungen (engl. constraints, [202, S. 46 ff.]). Diese Bedingungen für den Entwurf werden in einer UCF gespeichert. Um möglichst kleine Bitströme zu erzeugen, sollten diese Bedingungen nur für wirklich verwendete Komponententypen erstellt werden. Sollte zum Beispiel eine DSP-Spalte zu der rekonfigurierbaren Partition gehören, diese jedoch nicht von der Schaltung verwendet werden, sollte diese nicht zu der rekonfigurierbaren Partition hinzugefügt werden.

Weiterhin werden nicht alle Blocktypen durch die Bedingung unterstützt. Eine Tabelle in der genannten Referenz schlüsselt die unterstützten Blocktypen für unterschiedliche FPGA-Familien auf. Als konkrete Beispiele können die DCM oder PLL aufgeführt werden. Eine `AREA_GROUP` bietet weiterhin folgende Einstellungsmöglichkeiten an:

- *GROUP*: Kontrolliert die Packung der Slices von Schaltungsteilen die dem Bereich nicht angehören (CLOSED: keine Kombination erlaubt, OPEN: Kombination erlaubt; Standard: OPEN)
 - *PLACE*: Kontrolliert die Platzierung der Slices von Schaltungsteilen die dem Bereich nicht angehören (CLOSED: keine Kombination erlaubt, OPEN: Kombination erlaubt; Standard: OPEN)
 - *PRIVATE=ROUTE* (nicht dokumentiert und nur für dynamische Bereiche): Verwendung der Verdrahtungsressourcen nur für die internen Komponenten erlaubt
- Irreguläre Formen der Partitionen, wie eine T- oder L-Form, sind allgemein erlaubt. Da sich die Platzierung und Verdrahtung wesentlich komplizierter gestaltet, wird allerdings von solchen Formen abgeraten. Eine rechteckige Form ist daher die beste Wahl. Eine Form wird daher über zwei Punkte definiert: die untere, linke und die obere, rechte Ecke. Ähnliches gilt im Bezug auf den Abstand zwischen Partitionen: Die Grenzen von Partitionen können direkt nebeneinander liegen, was allerdings zu Verdrahtungsproblemen führen kann. Daher ist ein (geringer) Abstand empfehlenswert. Ein Überlappen oder Verschachteln von rekonfigurierbaren Partitionen ist nicht möglich.

Durch die Verwendung eines DPR-Systems kann es zu einer Takt-Reduktion von bis zu 10 % kommen [211, S. 14]. Weiterhin ist bei dem Design anzunehmen, dass die maximale Ausnutzung der Slices bei 80 % liegt.

3.4 Kommunikationsinfrastrukturen in DPR-Systemen

DPR-Systeme benötigen spezielle Kommunikationsinfrastrukturen für die Kommunikation zwischen Komponenten des statischen und dynamischen Bereichs, sogenannten *PR-Modulen*, bzw. für die Kommunikation der PR-Module untereinander. In [101] wird eine Kategorisierung der existierenden Kommunikationsinfrastrukturen für DPR-Systeme in zwei Gruppen vorgestellt. Bevor auf diese zwei Typen eingegangen wird, werden typische Eigenschaften der Kommunikationsinfrastruktur durch zwei *Attribute* festgelegt.

3.4.1 Eigenschaften der Kommunikationsinfrastruktur

Eine Kommunikationsinfrastruktur oder ein Teil einer Kommunikationsinfrastruktur in Form eines einzelnen Signals besitzt die beiden folgenden Attribute *Kommunikationsrichtung* und *Signalart*.

Attribut Kommunikationsrichtung Die Richtung der Kommunikationsstruktur kann *unidirektional* oder *bidirektional* sein. Eine unidirektionale Struktur kann nur einseitig

betrieben werden: von einem statischen Modul zu einem rekonfigurierbaren Modul oder andersherum. Eine bidirektionale Struktur erlaubt hingegen mehrere Treiber und kann in beide Richtungen betrieben werden, wobei stets nur ein Treiber aktiv ist, sodass eine Arbitrierung implementiert werden muss.

Attribut Signalart Die Art des Signals ist ein weiteres Attribut. So kann ein Signal entweder *dediziert* (engl. *dedicated*), also einem konkretem Modul zugewiesen, oder *gemeinsam* verwendet (engl. *shared*) an mehrere Module gesendet werden. Dedizierte Signale sind stets Punkt-zu-Punkt Verbindungen.

Durch die Kombination beider Attribute ergeben sich typische Signaltypen. Bidirektionale gemeinsam verwendete Signale werden zum Beispiel in den Datenleitungen einer busbasierten Kommunikationsinfrastruktur verwendet. Ein Anwendungsbeispiel von gemeinsam verwendeten, unidirektionalen Signalen ist beispielsweise in den Adressleitungen zu finden. Kontroll- oder Arbitrierungssignale sind wiederum Beispiele für dedizierte, unidirektionale Signale. Als konkretes Beispiel in dem DPR-Kontext kann die (De-) Aktivierung einzelner Module genannt werden.

3.4.2 Typen der Kommunikationsinfrastruktur

Weiterhin kann die Kommunikationsinfrastruktur verschiedenen Typen zugeordnet werden. Koester et al. unterteilen diese Typen in sogenannte *Link Makros* und *Eingebettete Makros* [101].

Link Makros

Eine Verbindung vom statischen Bereich in die PR-Region wird als *Link Makro* bezeichnet. Ein PR-Modul nutzt ausschließlich diese Verbindung für die Kommunikation zum statischen Bereich. Die Abbildung 3.14 (a) zeigt diese einfache Verbindung zwischen der PR-Region und dem statischen Bereich. Um mehrere PR-Module durch diese starren Verbindungen auch untereinander verbinden zu können, werden *Link Makros zwischen Modulen* benötigt. Diese verbinden zusätzlich benachbarte Module miteinander, wie in Abbildung 3.14 (b) dargestellt.

Der Vorteil der einfachen Implementierung für eindimensionale und zweidimensionale Anordnungen von Modulen, wird durch eine Vielzahl an Nachteilen egalisiert. Zunächst sinkt die Bandbreite durch die hohe Anzahl von Verbindungssprüngen (*Hops*), um ein Signal von dem statischen Modul über mehrere PR-Module zum letzten PR-Modul zu schicken. Darüber hinaus muss die Kommunikationsinfrastruktur in dem Modul selbst implementiert werden. Dadurch ist die Verbindung abhängig von der Realisierung (insbesondere der Verdrahtung) eines jeden Moduls. Durch das Austauschen eines Moduls bei einer Rekonfiguration ändern sich daher die Eigenschaften der Kommunikationsinfrastruktur. So kann die Kommunikation unterbrochen werden, wenn nicht exakt die gleichen Verdrahtungsressourcen in dem neu-platzierten Modul zur Realisierung der Kommunikationsinfrastruktur benutzt werden. Ferner ist ein Lö-

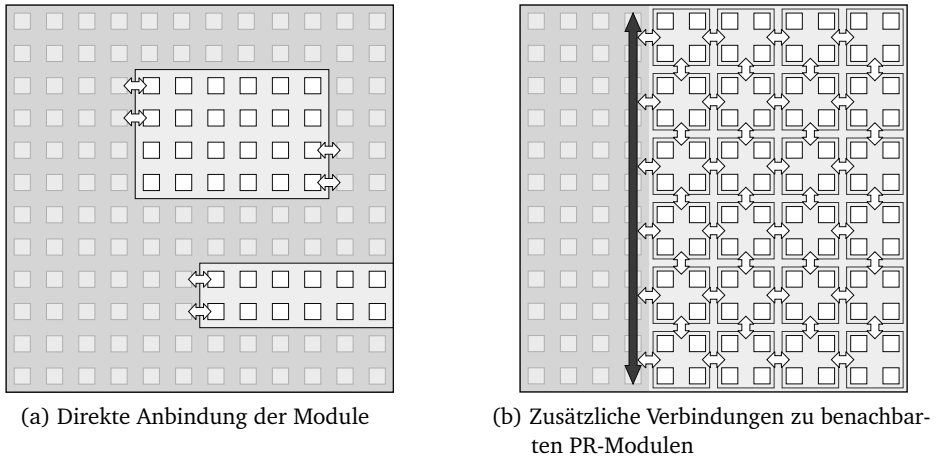


Abbildung 3.14: Link Makros zur Anbindung von PR-Modulen [101, S. 4 f.]

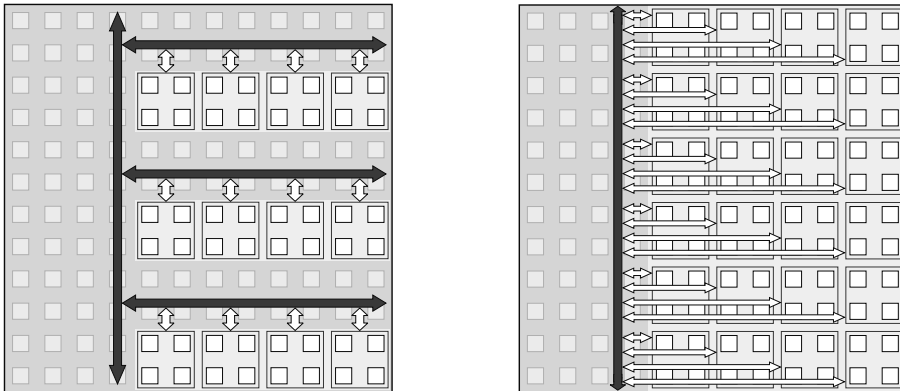
schen eines Moduls nicht möglich, da die Kommunikationsinfrastruktur in dem Falle unterbrochen ist.

Um diesen Nachteil zu umgehen, kann jedes Modul mit einem Kommunikationskanal, der direkt mit dem statischen Bereich verbunden ist, angeschlossen werden. Diese Kommunikationsstruktur wird als *Link Makro mit direktem Kommunikationskanal* bezeichnet und ist in Abbildung 3.15 (a) dargestellt. Diese Kommunikationsstruktur weist allerdings einen neuen, entscheidenden Nachteil auf: Die direkten Kommunikationskanäle gehören zum statischen Bereich wodurch der PR-Bereich in mehrere PR-Regionen aufgeteilt wird. Die für die PR-Module zur Verfügung stehende Fläche wird dadurch deutlich verkleinert.

Eine weitere Kommunikationsstruktur umgeht alle bisher genannten Nachteile, das sogenannte *Wormhole Routing* (dt. Wurmlochverdrahtung). Dabei wird jedes Modul direkt mit dem statischen Bereich über einen eigenen Kanal verbunden (siehe Abbildung 3.15 (b)). Daraus resultiert, dass Kanäle über mehrere andere Module hindurch geroutet sind. Dieses hat wiederum zwei neue entscheidende Nachteile: Zum einen steigt die Anzahl der insgesamt benötigten Verdrahtungsressourcen stark an, zum anderen sinkt die Anzahl der verfügbaren Verdrahtungsressourcen eines Moduls, je näher dieses an dem statischen Bereich lokalisiert ist.

Eingebettetes Makro

Ein sogenanntes *eingebettetes Makro* bietet den PR-Modulen identische Anschlusspunkte (engl. connection point oder auch port) zur Kontaktierung an den Kanal. Dieses Makro ist daher keine Punkt-zu-Punkt Verbindung, sondern eine homogene, monolithische



(a) Verbindung zu einem direkten Kommunikationskanal

(b) Wormhole Makro mit direktem Anschluss an den Kommunikationskanal

Abbildung 3.15: Weitere Link Makros zur Anbindung von PR-Modulen [101, S. 5]

Verbindung zwischen dem statischen Bereich und allen verbundenen PR-Regionen. Durch die Homogenität in der Auswahl der (Logik- und Verdrahtungs-) Ressourcen in den einzelnen PR-Regionen wird gewährleistet, dass stets die gleichen Anschlusspunkte bereitgestellt werden. Eine unterbrechungsfreie Kommunikation kann dadurch während der Rekonfiguration gewährleistet werden. Diese Kommunikationsstruktur kombiniert alle Vorteile der drei Link Makro Varianten ohne deren Nachteile zu teilen. Prinzipiell können auch inhomogene, eingebettete Kommunikationsinfrastrukturen zur Realisierung verwendet werden. Durch die Inhomogenität ist eine Umplatzierung (engl. relocation) von PR-Modulen nicht möglich. Abbildung 3.16 zeigt die Realisierung der Kommunikationsinfrastruktur durch Verwendung von eingebetteten Makros.

Der allgemeine Aufbau einer bidirektionalen, gemeinsam verwendeten, 1-Bit breiten Kommunikationsinfrastruktur mit dem Typ eingebettetes Makro ist in Abbildung 3.17 dargestellt. Jede PR-Region (*Tile*) verwendet die gleichen FPGA-Ressourcen um identische Verbindungspunkte für den Dateneingang (D_{in}) und Datenausgang (D_{out}) inklusive eines Freigabesignals (en) zur Verfügung zu stellen.

Der Aufbau von unidirektionalen, dedizierten 1-Bit breiten Signalen eines eingebetteten Makros sind für beide Kommunikationsrichtungen in den Abbildungen 3.18 dargestellt. Um die Homogenität der gesamten Kommunikationsinfrastruktur zu wahren, müssen ebenfalls die gleichen FPGA-Ressourcen in den einzelnen PR-Regionen verwendet werden. Zur Adressierung erhält jede PR-Region eine eindeutige Adresse, welche in internen Registern, binär (a) oder One-Hot (b), kodiert ist [79].

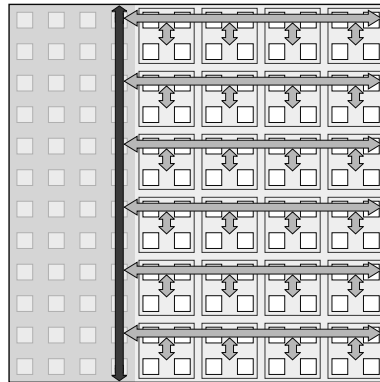


Abbildung 3.16: Eingebettetes Makro mit definierten Anschlusspunkten an den Kommunikationskanal [101, S. 5]

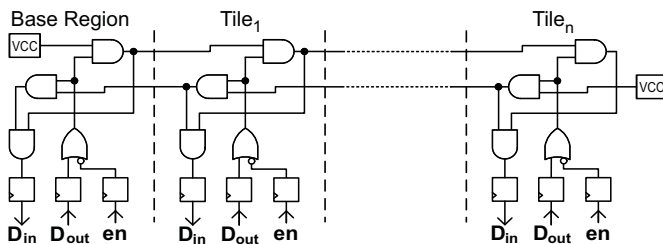


Abbildung 3.17: 1-Bit breite, bidirektionale, gemeinsam verwendete Kommunikationsinfrastruktur als eingebettetes Makros [101, S. 5]

Gegenüberstellung der Kommunikationsmakros

In Tabelle 3.10 sind zusammenfassend die Funktionen der vorgestellten Kommunikationsmakros gegenübergestellt. Ein Wormhole Makro erlaubt als einziger Kommunikationsmakrotyp lediglich Punkt-zu-Punkt-Verbindungen und unterstützt somit keine Kommunikationsinfrastruktur, in welcher gemeinsam verwendete Signale benötigt werden. Link Makros wiederum unterstützen keine Punkt-zu-Punkt-Verbindungen, sodass keine dedizierten Signale realisiert werden können. Die Homogenität der Kommunikationsinfrastruktur kann durch die Punkt-zu-Punkt-Verbindungen des Wormhole Makros ebenfalls nicht eingehalten werden. Dieser Nachteil kann auch durch eine, im Vergleich, höhere Taktfrequenz in DPR-Systemen nicht ausgeglichen werden. Die Auswertung der FPGA-Ressourcenauslastung durch die einzelnen Kommunikationsmakros ist in [101] beschrieben. An dieser Stelle werden nur die Ergebnisse festgehalten, welche besagen, dass diesbezüglich das Eingebettete Makro zu der geringsten Auslastung führt. Der

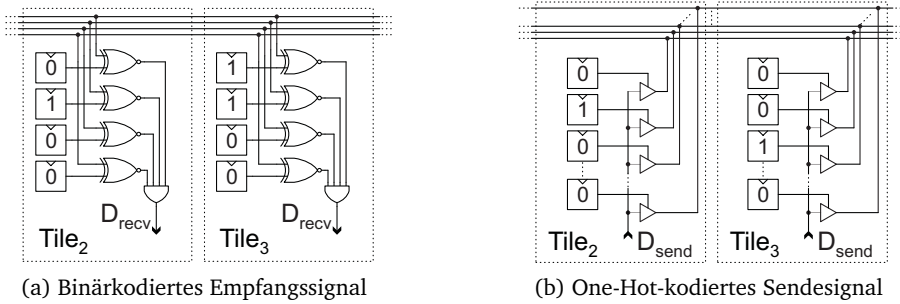


Abbildung 3.18: Dedizierte, unidirektionale Signale im eingebetteten Makro [101, S. 6]

Tabelle 3.10: Vergleich der Funktionen unterschiedlicher Kommunikationsmakrotypen (basierend auf [101, S. 7])

	Link	Wormhole	Eingebettet
Gemeinsam verwendete Signale	ja	nein	ja
Dedizierte Signale	nein	ja	ja
Homogenität	ja	nein	ja
Interrupt-freie Rekonfiguration	nein	ja	ja
Taktfrequenz	-	+	o
FPGA-Ressourcenauslastung	o	-	+

Vergleich der Kommunikationsmakros zeigt, dass ein eingebettetes Makro insgesamt die beste Lösung für den Kommunikationsinfrastrukturtypen darstellt.

3.5 Grob- und feingranulare DPR-Systeme

DPR-Systeme können unterschiedliche Rekonfigurationsarten implementieren und unterteilen sich daher in grobgranulare oder feingranulare Ansätze. In diesem Abschnitt wird auf Entwurfsabläufe für die Xilinx ISE Werkzeuge eingegangen. Hierdurch werden FPGAs bis zur 7-Serie unterstützt. Der Xilinx Vivado-basierte Entwurfsablauf mit einer Unterstützung von UltraScale FPGAs wurde erstmalig im April des Jahres 2016 veröffentlicht [244].

Drei unterschiedliche Xilinx Entwurfsabläufe werden in dem ersten Unterabschnitt beschrieben und die Limitierungen aufgeführt. In dem akademischen Bereich sind mehrere Projekte entstanden, welche die vorgestellten Limitierungen der Xilinx Entwurfsabläufe aufheben, weitere Rekonfigurationsarten unterstützen und somit die vorhandene Hardware-Flexibilität des FPGAs nutzbar machen. Diese Projekte verfolgen daher einen feingranularen Ansatz, indem der FPGA in mehrere homogene, rekonfigurierbare Bereiche aufgeteilt wird. Konkret haben die folgenden, vorgestellten akademischen Projekte die Entwicklung eines flexiblen DPR-System mit einer Umplatzierung von Modulen zum Ziel. Ferner, soll ein entkoppelter Entwurf von PR-Modulen und dem statischen Design möglich sein, sodass eine Änderung am statischen Design nicht dazu führt, dass alle Module neu erstellt werden müssen. Zwei dieser Projekte basieren auf freien APIs (engl. Application Programming Interface, dt. Anwendungsprogrammierschnittstelle) für Xilinx FPGAs: Torc (*Tools for open reconfigurable computing*) [175] und RapidSmith [140]. Diese ermöglichen den Zugriff auf FPGA-interne Informationen und die Manipulation von existierenden Designs. Durch diese Informationen wird die Entwicklung von eigenen Algorithmen (z. B. zur Packung, Platzierung oder Verdrahtung) bzw. eigener Software zur Erweiterung der Standard Xilinx Werkzeugkette ermöglicht.

Zusätzlich existieren Projekte, welche den Fokus auf die Integration der DPR-Funktionalität in einem (Echtzeit-) Betriebssystem und somit auf eine andere Abstraktionsebene legen. Agne et al. stellen mit *ReconOS* ein Betriebssystem für die Verarbeitung durch hybride Software- und Hardware-Threads, welche auf einem FPGA ausgeführt werden, vor [3]. Eine Anwendung wird in drei Schritten von einer reinen multithreaded Software-Implementierung in eine Hardware-Implementierung in VHDL überführt. Das *R3TOS (Reliable Reconfigurable Real-time Operating System)* von Iturbe et al. stellt ein Echtzeit-fähiges Betriebssystem dar, in dem Anwendungen ebenfalls als Hardware-Threads ausgeführt werden [87]. Besonders innovativ ist die Realisierung der Kommunikation in dem DPR-System. Anstelle von typischen, wie in Abschnitt 3.4 vorgestellten, Kommunikationsinfrastrukturen wird BRAM- oder LUTRAM-basierter Speicher als Eingangs- und Ausgangsspeicher der Hardware-Threads verwendet und dieser über die interne Konfigurationsschnittstelle ICAP geschrieben und ausgelesen. Hierdurch können zum einen FPGA-Ressourcen für die Kommunikationsinfrastruktur eingespart und zum anderen das Problem von statischen Signalen, die den dynamischen Bereich durchkreuzen und somit eine Umplatzierung von Modulen verhindern, umgangen werden. Der Nachteil dieser Lösung liegt in der Teilung der verfügbaren Bandbreite

der Konfigurationsschnittstelle. Eine kommerzielle Lösung in diesem Umfeld bietet das Unternehmen TOPIC mit dem Produkt Dyplo [174]. Dyplo bietet eine out-of-the-box Integration der CPU- und FPGA-Funktionalität für Xilinx Zynq-Bausteine an und ermöglicht die Verwendung von Hardware-Beschleunigern, welche mittels DPR auf dem FPGA-Teil des Zynqs platziert werden.

3.5.1 Rekonfigurationsarten

Bevor auf die konkrete Entwurfsabläufe eingegangen wird, erfolgt zunächst eine Kategorisierung der Rekonfigurationsarten. Koch definiert in [98, S. 3 f.] die vier folgenden Rekonfigurationsarten.

Single Island Ermöglicht die Platzierung eines PR-Moduls in einem bestimmten rekonfigurierbaren Bereich. Dies ist vorteilhaft in Systemen, wo Funktionen im Wechsel verwendet werden.

Multi Island Ermöglicht die Platzierung eines PR-Moduls in mehrere identische rekonfigurierbare Bereiche. Identisch sind die Bereiche nur dann, wenn die gleichen FPGA-Ressourcen (Basisblöcke und Verdrahtung) für die PR-Module verfügbar sind. Konkret müssen in identischen PR-Bereichen die Basisblöcke an den gleichen, relativen Positionen vorhanden sein (*resource footprint*) und identische Verbindungen zu dem statischen Bereich existieren (*connection footprint*).

1D-Slot Ermöglicht die Aufteilung des rekonfigurierbaren Bereichs in mehrere, nebeneinanderliegende, eindimensionale Slots. Ein PR-Modul belegt, entsprechend dem FPGA-Ressourcenbedarf, eine optimale Anzahl dieser Slots. Ein DPR-System mit dieser Eigenschaft führt zu einem geringen internen Verschnitt im Vergleich zu den zuvor aufgeführten Island-Rekonfigurationsarten. Dort bestimmt das größte Modul die Größe des PR-Bereichs, was bei der Platzierung von kleineren PR-Modulen zu einem hohen internen Verschnitt führt (engl. *internal fragmentation*).

2D-Grid Ermöglicht die Aufteilung des rekonfigurierbaren Bereichs in ein nebeneinanderliegendes, zweidimensionales Gitter (engl. *grid*). Durch diese weitere Unterteilung kann der interne Verschnitt weiter minimiert werden.

3.5.2 Xilinx

Xilinx teilt die Entwurfsabläufe eines DPR-Systems in zwei Kategorien auf: differenzbasiert [226] und modulbasiert [205, 211]. Wie der Name suggeriert, werden in einem differenzbasierten Entwurf nur Unterschiede zwischen zwei Designs betrachtet. Wenn sich von Design zu Design nur kleine Veränderungen ergeben, ist diese Entwurfsart von Vorteil. Die Dokumentation [226] beschreibt, wie mit Hilfe des Xilinx FPGA Editors, die einzelnen partiellen Bitströme für drei Veränderungsfälle erstellt werden können:

1. Änderungen in der Gleichung einer LUT zur Realisierung einer logischen Funktion

2. Änderungen im dedizierten Speicher
3. Änderungen des IO Standards

Diese Entwurfsart schließt damit (als alleiniger Entwurfsablauf) automatisch große, komplexe DPR-Systeme aus. In einem modulbasierten Entwurf werden hingegen eindeutig definierte Teile des Designs als rekonfigurierbar deklariert. Diese Unterscheidung von statischen und rekonfigurierbaren Teilen eines Designs ermöglicht eine abweichende Behandlung und Implementierung der jeweiligen Teile und dadurch den Entwurf von größeren DPR-Systemen.

Xilinx EA PR

Der modulbasierte Entwurfsablauf wurde erstmals in der Version 5.2 der Hersteller-Software ISE (Integrated Software Environment) von Xilinx eingeführt. Mit Hilfe des zugangsbeschränkten Entwurfsablaufs *EA PR* (*Early Access Partial Reconfiguration*) wurden die Standardwerkzeuge erweitert. Der Name ist historisch bedingt, da der Entwurfsablauf nur über ein entsprechendes *Early Access* Programm von Xilinx verfügbar war. Initial wurden nur die FPGA-Familien Virtex-II und Virtex-II Pro unterstützt. Eine detaillierte Dokumentation ist in [205] gegeben.

Ein Update des *EA PR* Entwurfsablaufs wurde für die ISE-Versionen 8.1 und 8.2 sowie 9.1 vollzogen, um auch die FPGAs der Virtex-4-Familie zu unterstützen. Die Erstellung des DPR-Systems erfolgt bis zu diesen Versionen ausschließlich über Skripte [205, S. 49 f.].

Die letzte *EA PR* Variante wurde für die ISE-Version 9.2.04i vorgestellt. Neben einer (eingeschränkten) Integration der Virtex-5-Familie [205, S. 13 f.] wird die Unterstützung bei der Erstellung eines DPR-Systems durch die Xilinx *PlanAhead* Software vorgestellt. Die Software unterstützt bei der Erstellung der (Rand-) Bedingungen und steuert die Implementierung des statischen Designs und der PR-Module durch eine grafische Benutzeroberfläche (engl. *Graphical User Interface*, GUI). Die Kommunikation in diesen DPR-Systemen wird über sogenannte *Bus-Makros* realisiert. Dabei stellen Bus-Makros lediglich eine Punkt-zu-Punkt Verbindung und keinen Kommunikationsbus zur Verfügung und entsprechen somit einfachen Link-Makros. Abbildung 3.19 zeigt den vollständigen *EA PR* Entwurfsablauf.

Das System wird durch eine HDL (engl. *Hardware Description Language*) beschrieben. Die Randbedingungen für den Entwurf werden in einer *UCF* gespeichert, welche von allen Konfigurationen benutzt werden. Zu den wesentlichen Randbedingungen gehören die bereits beschriebenen *AREA_GROUPS*. Für jede rekonfigurierbare Region muss die Größe und Form der Partition vorgegeben werden (siehe Abschnitt 3.3.4), wodurch gleichzeitig die Grenzen zum statischen Bereich bestimmt werden. Die rekonfigurierbare Logik wird einzeln für jedes Modul synthetisiert, anschließend folgt die Synthese des statischen Bereichs und abschließend ein Zusammenführen (engl. *merge*) aller partiellen Designs.

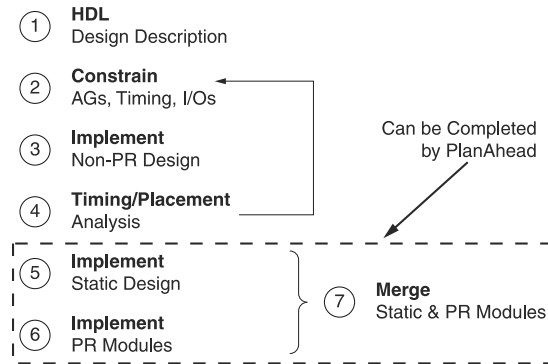


Abbildung 3.19: Xilinx EA PR Entwurfsablauf [205, S. 20]

Xilinx PR

Mit der Version 12 von ISE wurde ein Entwurfsablauf von Xilinx vorgestellt, welcher eine neuartige Kommunikationsinfrastruktur (*proxy logic*) für die Implementierung eines DPR-Systems einführt. Obwohl DPR nun fester Bestandteil der Xilinx ISE ist, wird eine zusätzliche Lizenz zum Erstellen eines DPR-Systems benötigt. Der Entwurfsablauf wird im *Partial Reconfiguration User Guide* [211, S. 23 ff.] beschrieben und ist in Abbildung 3.20 dargestellt.

Ein Design wird in *Partitionen* eingeteilt, wobei eine Partition ein vom Benutzer definierter logischer Bereich ist. Eine Partition kann von verschiedenen PR-Modulen über die Zeit verteilt benutzt werden. Eine Partition garantiert, dass geteilt benutzte Module, wie zum Beispiel statische Module unter allen Konfigurationen identisch sind. Eine Partition wird daher in Form eines Attributs, *RECONFIGURABLE* oder *STATE*, an ein Modul angehaftet. Die Xilinx Software erkennt dieses Attribut und implementiert das Modul entsprechend des gesetzten Attributs. So wird beispielsweise automatisch ein partieller Bitstrom für ein Modul mit dem Attribut *RECONFIGURABLE* erstellt. Das *STATE* Attribut gibt Aufschluss über den Zustand eines statischen Moduls. Ein statisches Modul kann dabei entweder erstmalig implementiert (Wert: *implement*) oder von einer existierenden Konfiguration importiert werden (Wert: *import*).

Sogenannte *Partition Pins* verbinden den statischen Bereich mit den PR-Modulen. Ein Partition Pin hat eine definierte unidirektionale Verbindungsrichtung und fungiert daher entweder als Eingang oder Ausgang zum PR-Modul. Neben der einfachen Deklaration des Pins können zeitliche Bedingungen (engl. *timing constraints*) für die Verbindungen von oder zu den Partition Pins spezifiziert werden. Ein Partition Pin wird durch eine LUT (sogenannte *proxy logic*) realisiert und kann daher als *entartetes* Link-Makros angesehen werden, da Anschlusspunkte ohne Verdrahtung zum statischen Bereich bereitgestellt werden.

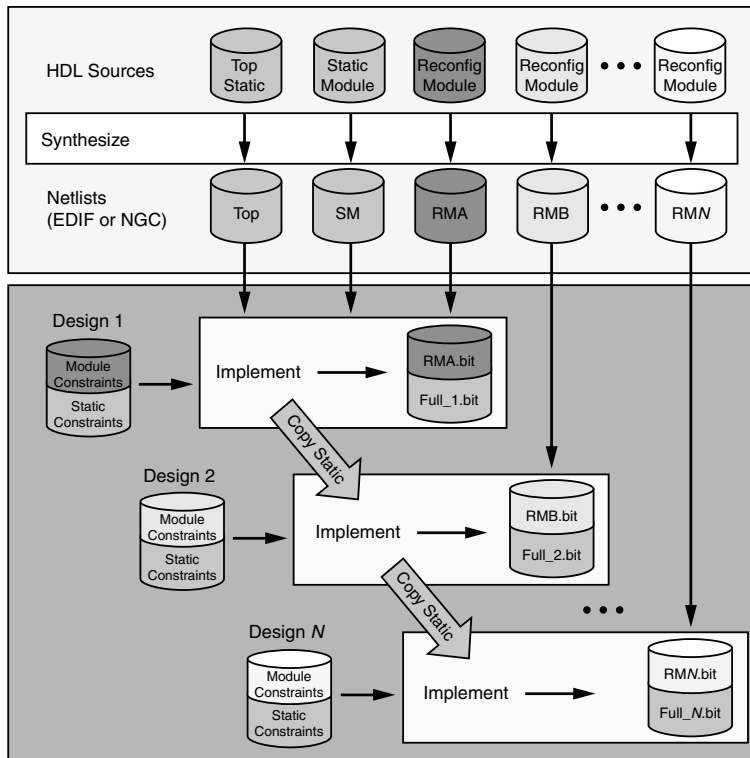


Abbildung 3.20: Xilinx PR Entwurfsablauf [211, S. 23]

Es werden *Konfigurationen* (engl. configurations) erstellt, welche aus der statischen Logik und einer Auswahl von PR-Modulen bestehen. Jedes PR-Modul wird in unabhängigen Projekten synthetisiert. Eine vollständige Konfiguration wird durch das Werkzeug *NGDBuild* zusammengeführt. Anschließend wird der statische Anteil extrahiert und für weitere Konfigurationen, das heißt eine andere Belegung der PR-Module oder generell andere PR-Module, verwendet. Abschließend existiert für jede Konfiguration eine vollständige Konfigurationsdatei und für jede durch die Konfigurationen definierte Belegung der PR-Module partielle Konfigurationsdateien. Dabei ist es nicht nötig Konfigurationen mit allen möglichen Kombinationen der PR-Module zu erstellen wenn ein einheitliches statisches System verwendet wird. Es genügen die Konfigurationen, in denen jedes benutzte PR-Modul einmal implementiert ist. Ein PR-Modul kann dann allerdings nur in diese eine Partition platziert werden. Eine Umplatzierung in andere Partitionen ist durch die unterschiedliche Platzierung und Verdrahtung der Kommunikationsinfrastruktur nicht möglich. Das heißt, wenn ein PR-Modul in mehreren Partitionen

platzierbar sein soll, müssen entsprechende Konfigurationen erstellt werden, um einen passenden partiellen Bitstrom für die Partitionen zu erhalten.

Der Entwurfsablauf kann wie bei der letzten *EA PR* Version durch *PlanAhead* unterstützt oder vollständig gesteuert werden [211, S. 57 ff.]. Alternativ können Skripte in Tcl (engl. Tool command language) verwendet werden, um den Aufruf der einzelnen ISE-Werkzeuge zu parametrieren und den Entwurfsablauf zu koordinieren [211, S. 83 ff.].

Softwareseitig werden seit der ISE-Version 12 Virtex-4, -5 und -6 unterstützt. Mit der Version 13 wurden die ersten 7-Serie-FPGAs aufgenommen. Vollständig integriert wurden alle 7-Serie-FPGAs sowie die Zynq-7000-Familie mit der ISE-Version 14.5 [210]. Die Spartan-Familien werden, wie bereits erwähnt, softwareseitig offiziell nicht unterstützt.

3.5.3 ReCoBus und GoAhead

Die *ReCoBus-Builder* Softwarekette für flexible DPR-Systeme wurde im Jahr 2008 durch Koch et al. veröffentlicht [100]. *GoAhead* ist eine Neuentwicklung der *ReCoBus-Builder* Softwarekette von Beckhoff et al. aus dem Jahr 2012 [19].

ReCoBus

In [100] wird eine Einführung in ein ReCoBus DPR-System gegeben. Ein ReCoBus DPR-System kann in mehrere rekonfigurierbare und statische Bereiche eingeteilt werden. Ein rekonfigurierbarer Bereich ist wiederum in kleine Teilabschnitte, sogenannte *Resource Slots*, aufgeteilt und kann, je nach Größe der einzelnen rekonfigurierbaren Module, gleichzeitig mehrere Module aufnehmen. Es existieren zwei Arten von Kommunikationsinfrastrukturen in einem ReCoBus-DPR-System. Sogenannte *IOBars* ermöglichen den Anschluss an Ein- oder Ausgangspins oder stellen Punkt-zu-Punkt Verbindungen zu anderen Modulen innerhalb des DPR-Systems zur Verfügung. Dafür werden horizontal verlaufende Leitungen reserviert. Nur wenn in einem *Resource Slot* eine Verbindung zu einem Ein- oder Ausgangspin oder zu einem anderen Modul benötigt wird, kann durch zusätzliche Logik eine Verbindung zu der IOBar hergestellt werden. Parallel zu den IOBars verläuft der sogenannte ReCoBus, welcher die eigentliche Kommunikation zwischen dem statischen Bereich und den rekonfigurierbaren Modulen bereitstellt. Abbildung 3.21 zeigt den Entwurfsablauf eines ReCoBus DPR-Systems.

Als erster Schritt muss das später verwendete Bus-Protokoll ausgewählt werden. Hierbei werden die Busprotokolle AMBA, CoreConnect, AVALON und Wishbone unterstützt. Weiterhin muss z. B. die Breite des Daten- und Adressbusses, die Anzahl der Kontroll-Signale und die Anzahl der Master- und Slave-Module spezifiziert werden. Eine Spezifikation der definierten Kommunikationsinfrastruktur kann, unabhängig von dem Ziel-FPGA, in mehreren Designs wiederverwendet werden und wird über eine Benutzeroberfläche der ReCoBus-Suite erstellt und abschließend als RTL-Modell (engl. Register Transfer Level, Registertransferebene) oder VHDL-Beschreibung abgespei-

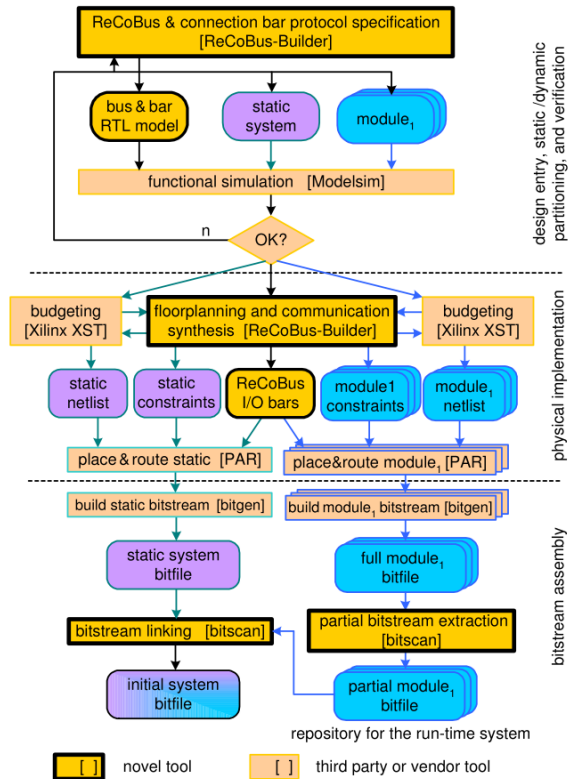


Abbildung 3.21: Entwurfsablauf eines ReCoBus DPR-System [99, S. 9]

chert. Die Synthese, Platzierung und das Routing der einzelnen PR-Module sowie des vollständigen Designs wird durch Standard Xilinx ISE-Werkzeuge ausgeführt.

Nachdem die Simulation erfolgreich durchgeführt wurde, wird die Synthese gestartet, um die benötigten Ressourcen des Designs zu bestimmen. Basierend auf diesen Ressourcen-Belegungen wird der sogenannte Floorplanning-Schritt durchgeführt, in dem zunächst das Ziel-FPGA gewählt und dieses anschließend in den statischen und rekonfigurierbaren Bereich partitioniert wird. Für jeden rekonfigurierbaren Bereich können Höhe, Breite und Anzahl von Resource Slots (in CLBs) spezifiziert werden. Nach der Spezifikation des gesamten Systems wird die Kommunikationsinfrastruktur ReCoBus von dem *ReCoBus-Builder* erstellt. Dabei werden zunächst die Slices zur Implementierung der Logik der Kommunikationsinfrastruktur der Resource Slots zugeordnet. Anschließend wird die Logik homogen in allen Resource Slots platziert, wobei die relative Position der Slices in allen Slots eingehalten werden muss. Die interne Verbindung

der Logik sowie die Verbindung zwischen zwei benachbarten Resource Slots, wird über einen eigenen Router (unter Verwendung eines Breitensuche-Algorithmus) erstellt. Abschließend wird der ReCoBus (Logik- und Verdrahtungsressourcen) und die IOBars in Xilinx konformen Hard-Makros (im Format NMC) abgespeichert.

Zur strikten Trennung zwischen statischen und dynamischen Bereichen werden sogenannte *Blocker-Makros* von dem ReCoBus-Builder aufgebaut [99, S. 22 ff.]. Die Blocker-Makros können alle Ausgangssignale von CLBs oder BRAMs verwenden, um den Xilinx Router in der Auswahl der verfügbaren Verdrahtungsressourcen einzugrenzen. Dies ist notwendig, da Xilinx selbst keine entsprechenden Einstellungsmöglichkeiten anbietet. Ohne die Verwendung der Blocker-Makros kommt es dazu, dass statische Signale den dynamischen Bereich kreuzen. Im letzten Schritt werden die Konfigurationsdateien des vollständigen DPR-Systems generiert. Die partiellen Bitströme werden dabei durch eine eigene Software (*bitscan*) aus dem vollständigen Bitstrom extrahiert. Abschließend kann die gleiche Software eine vollständige Konfigurationsdatei eines initialen DPR-System mit platzierten Modulen erstellen. ReCoBus ist unabhängig vom Xilinx PR Flow und benötigt somit keine entsprechende Lizenz.

Die ReCoBus-Software ist in der Programmiersprache C# (Microsoft Visual Studio 2005) geschrieben und unterstützt die FPGA-Familien Virtex-II (Pro) und Spartan-3.

GoAhead

Die Neuimplementierung der ReCoBus-Software hatte zum Ziel aktuelle Xilinx FPGA-Familien zu unterstützen und die Benutzbarkeit der Software zu erhöhen [19]. Neben einer Benutzeroberfläche sollte weiterhin ein Scripting-Interface bereitgestellt werden, sodass Aufgaben im Stapelbetrieb (engl. batch mode) ausgeführt werden können. In [98] werden die Unterschiede und die neuen Funktionen von GoAhead vorgestellt und im folgenden zusammengefasst.

Abbildung 3.22 zeigt den Entwurfsablauf eines GoAhead DPR-Systems. Als erster Schritt muss das (*Initial*) *Planning* durchgeführt werden. In diesem Schritt muss das System manuell in statische und dynamische Komponenten aufgeteilt werden. Weiterhin müssen dynamische Bereiche und damit die Schnittstellen zwischen dem dynamischen und statischen Bereich definiert werden. Die Schnittstellen werden dabei als VHDL Entities abgespeichert. Abschließend erfolgt eine Abschätzung der benötigten Ressourcen, z. B. in LUTs und BRAMs, für die einzelnen Module.

In dem zweiten Schritt *Floorplanning* wird die konkrete Anzahl der verfügbaren Ressourcen in jedem Modul festgelegt. Anschließend werden die zuvor definierten Schnittstellen eingelesen und Platzierungsinformationen gesammelt.

Wie in Abbildung 3.22 dargestellt, erfolgt die weitere Entwicklung des statischen Systems und der dynamischen Module unabhängig voneinander. Bei der Entwicklung müssen die vorhandenen FPGA Ressourcen eindeutig dem dynamischen oder dem statischen Teil zugeordnet sein. GoAhead verwendet für die Slices, BRAMs und andere Blöcke die Xilinx Randbedingung *CONFIG PROHIBIT*. Für die Implementierung des statischen Systems werden alle Blöcke innerhalb der dynamischen Bereiche mit dieser

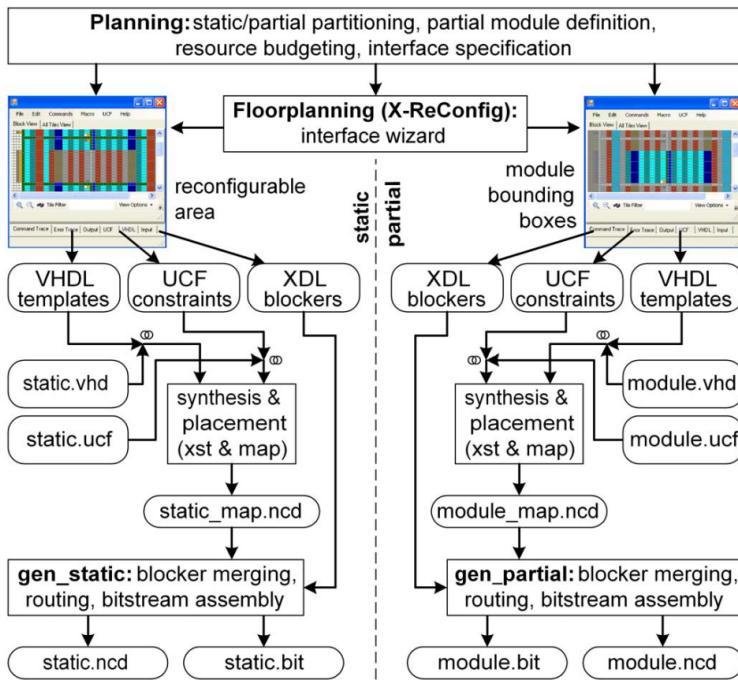


Abbildung 3.22: Entwurfsablauf eines GoAhead DPR-Systems [19, S. 3]

Randbedingung ausmaskiert. Da es keine analoge Randbedingung für die Verdrahtungsressourcen gibt, werden wie bei ReCoBus Blocker-Makros verwendet, welche in dem Fall der Implementierung des statischen Systems sämtliche Verdrahtungsressourcen in dynamischen Bereichen belegen. Anschließend werden die ISE Werkzeuge (bis zum Technologie-Mapping) verwendet. Die letzten Schritte beim Entwurf werden von einem Skript (*gen_static*) ausgeführt. Dazu gehört das Hinzufügen der Blocker-Makros, das Verdrahten durch FPGA Edline, eine Timing Analyse und die Erstellung des Bitstroms.

Die Entwicklung der dynamischen Module erfolgt ähnlich der beschriebenen Implementierung des statischen Systems. Der Unterschied ist hierbei, dass die Randbedingungen und Blocker-Makros nun um die dynamischen Bereiche gelegt werden. Das Skript *gen_partial* führt die gleichen Aufgaben wie das *gen_static* Skript aus, mit der Ausnahme der Erstellung von partiellen Bitströmen. Durch die Option *BitGen -r* wird dabei, wie in Abschnitt 3.5.2 beschrieben, ein *differentieller* Bitstrom erzeugt. GoAhead unterstützt zwei Methoden bei der Erzeugung partieller Bitströme:

- Inkrementell: Ein partielles Modul wird als Inkrement eines existierenden Designs erstellt. Dies entspricht dem Xilinx PR flow.
- Isoliert: Ein partielles Modul wird als Inkrement eines vollständig leeren Designs erstellt. Dies entspricht dem GoAhead Standard.

In [19] wird die Unterstützung der FPGA-Familien Virtex-4 bis Virtex-7 und Spartan-6 aufgeführt. Als Programmiersprache wurde, analog zur *ReCoBus-Builder* Softwarekette, C# verwendet. Die Software (in Form einer ausführbaren Datei) sowie eine Dokumentation ist frei verfügbar [71].

3.5.4 OpenPR

OpenPR [158] wurde primär von Sohahngpurwala entwickelt, basiert auf der quelloffenen (GNU GPL 3.0 Lizenz) API *Torc* (*Tools for open reconfigurable computing*, [175]) und ist seit dem Jahr 2011 als quelloffene Software verfügbar [130]. Abbildung 3.23 zeigt den Entwurfsablauf eines OpenPR DPR-Systems.

Als erster Schritt erfolgt ein Floorplanning, in dem der FPGA in statische und dynamische Regionen aufgeteilt wird. Dies kann manuell über die Platzierungsbedingungen (*AREA_GROUP*) in der UCF oder mit Hilfe der Xilinx PlanAhead Software geschehen.

Die Aufteilung der Ressourcen in dynamisch und statisch erfolgt im *genPlaceConstraints* Schritt. Laut Sohahngpurwala et al. reichen die Platzierungsbedingungen der *AREA_GROUP* bei der Generierung des statischen Teils nicht aus, um die Ressourcen für die ISE Software klar abzugrenzen. In OpenPR werden daher weitere Bedingungen (*CONFIG_PROHIBIT*) verwendet, wodurch spezifische Ressourcen (einzelne oder eine rechteckige Anordnung) als nicht verwendbar markiert werden. Die besagten Bedingungen werden durch eine eigene Software erstellt, sodass dieser Schritt für den Benutzer transparent erfolgt. Da es keine entsprechenden Bedingungen für die Verdrahtungsressourcen gibt, werden in OpenPR, analog zu ReCoBus und GoAhead, bestimmte Verdrahtungsressourcen geblockt. Für das statische Design werden alle Verdrahtungsressourcen, die eine Verbindung in den dynamischen Bereich herstellen können, einem Netz (blocking net) hinzugefügt. Die Kommunikationsinfrastruktur zwischen dem statischen und dynamischen Bereich erfolgt, ähnlich wie bei dem Xilinx *EA PR* Entwurfsablauf, durch vorgefertigte Bus-Makros. Für jedes Signal werden dabei zwei zusätzliche LUTs eingebaut. Dies führt im Vergleich zu dem aktuellen Xilinx *PR* Entwurfsablauf mit einer LUT (proxy logic) zu einem höheren Ressourcenaufwand und einer größeren Verzögerungszeit. Die Instanziierung der Makros muss dabei manuell in der VHDL-Datei erfolgen und einem bestimmten Schema entsprechen. Die benötigten Platzierungsinformationen für die einzelnen Makros werden anschließend automatisch der UCF hinzugefügt. Die vorgefertigten Bus-Makros erlauben aktuell nur eine vertikale Verbindung. Sohahngpurwala et al. erwähnt, dass horizontale Verbindungen durch zusätzliche Bus-Makros und Platzierungsstrategien erweitert werden könnten.

Nach der Instanziierung der Bus-Makros erfolgt die Generierung des statischen Designs (*Static Design Build*). Diese erfolgt über ein Makefile und abstrahiert die

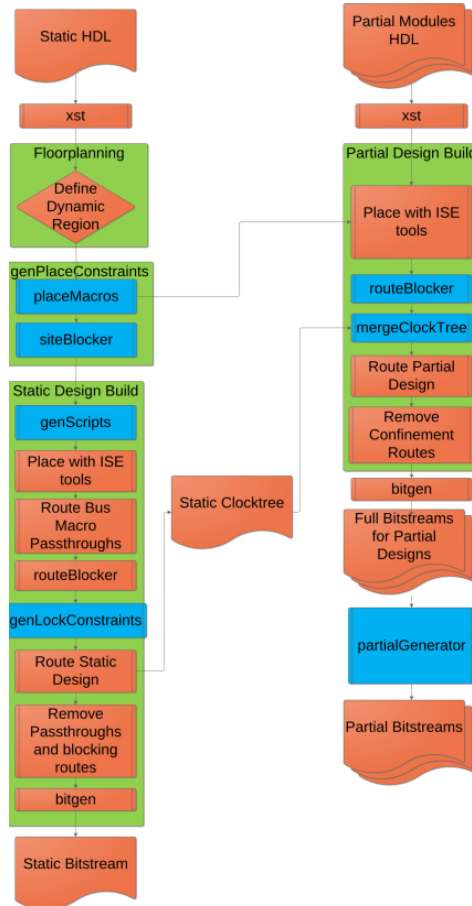


Abbildung 3.23: Entwurfsablauf eines OpenPR DPR-Systems [158, S. 5]

folgenden Schritte von einem Anwender. Die Platzierung erfolgt mit den Standard Xilinx ISE Werkzeugen (*XST*, *NGDBuild* und *MAP*). Die Implementierung des Designs erfolgt unter Einhaltung der erstellten UCF. Die anschließende Verdrahtung erfolgt nicht mit dem ISE Werkzeug *PAR*, sondern mit dem Xilinx *FPGA Editor*. Bevor das Design verdrahtet wird, müssen zunächst zwei Vorschritte ausgeführt werden. Der erste Schritt führt die Verdrahtung vom statischen Bereich in den dynamischen Bereich durch. Diese Verbindungen werden als *passthroughs* bezeichnet. Anschließend werden alle weiteren Verdrahtungsressourcen in dem dynamischen Bereich durch ein *blocking net* belegt. Alle Netze werden mit einer zusätzlichen LOCK-Randbedingung belegt (*genLockConstraints*).

Dies ist notwendig, da der Router des *FPGA Editors* ansonsten versuchen kann die Netze wieder aufzulösen, um die Verdrahtungsressourcen zu verwenden. Nach diesen Vorschriften wird das gesamte Design verdrahtet. Abschließend werden die *passthroughs* und *blocking nets* wieder entfernt, das Design im NCD-Format gespeichert und das Xilinx Werkzeug *BitGen* verwendet, um den vollständigen Bitstrom zu erzeugen. Die Verwendung des *FPGA Editor* Routers ist ein wesentlicher Nachteil, da dieser nicht so gute Ergebnisse erzielt, wie das ISE Werkzeug *PAR*.

Bei der Generierung der partiellen Module muss darauf geachtet werden, dass nur die Ressourcen in dem dynamischen Bereich verwendet werden. In diesem Fall reicht die Randbedingung *AREA_GROUP* für die Auswahl der verfügbaren Blöcke aus. Das Blockieren der Verdrahtungsressourcen erfolgt abermals über *blocking nets*. Vor der Verdrahtung des partiellen Moduls mit dem *FPGA Editor*, wird der Taktbaum aus dem statischen Design extrahiert und in das Design des partiellen Moduls eingefügt (*mergeClockTree*). Abschließend können die *blocking nets* wieder entfernt werden. Die Generierung der partiellen Bitströme erfolgt in zwei Schritten. Zunächst wird ein vollständiger Bitstrom erstellt und anschließend die Konfiguration für das partielle Modul aus dem Bitstrom extrahiert. Die Extraktion erfolgt dabei durch eine eigene Software (*partialGenerator*), welche den Bitstrom einliest und die relevanten *Configuration Frames* detektiert. Eine ausführliche Beschreibung der Software erfolgt in [157].

OpenPR basiert, wie eingangs geschrieben, auf Torc [175] und ist daher in der Programmiersprache C++ geschrieben. Die Software unterstützt die FPGA-Familien Virtex-4 und Virtex-5.

3.5.5 DREAMS

DREAMS wurde von Otero et al. im Jahr 2012 veröffentlicht, basiert auf RapidSmith [140] und ist im Rahmen des gleichnamigen Projektes **D**ynamically **R**econfigurable **E**mdedded **P**latforms for **N**etworked **C**ontext-**A**ware **M**ultimedia **S**ystems entstanden [132]. Otero et al. erwähnen, dass die Software öffentlich zugänglich gemacht werden soll, dies ist bis dato allerdings nicht der Fall.

Ein DREAMS-DPR-System teilt sich in sogenannte *Virtual Regions* (VR), wobei eine VR eine rechteckige, auf CLB-Koordinaten definierte Region beschreibt, welche entweder statische oder dynamische Module aufnehmen kann. Die VRs können dabei feingranular definiert werden und müssen sich nicht in der Höhe über eine Taktregion erstrecken. Ferner ist es möglich, dass ein *Virtual Modul* (VM) mehrere VRs belegt. Die Gesamtheit der VRs bilden die *Virtual Architecture* (VA), welche somit ein vollständiges DREAMS-System beschreibt. Ein VA wird in Form einer XML-Datei abgespeichert. Ein VM kann entweder in einer HDL- oder in einer NCD-Beschreibung vorliegen. Die VMs müssen in einer weiteren XML-Datei spezifiziert werden. Eine vollständige Spezifikation beinhaltet den Typ (statisch oder dynamisch), den Namen, den Namen der HDL- oder NCD-Datei, mögliche VR-Positionen sowie VR-Verbindungen. Um die zuletzt genannte Kommunikation realisieren zu können, werden sogenannte *Virtual Borders* (VB)

verwendet. Eine VB ist eine Datenstruktur, die Informationen über die verwendeten Verdrahtungsressourcen speichert. Für jedes Signal, das VRs kreuzt, werden VBs erstellt. Durch die Verwendung von diesen VBs können identische Verdrahtungsschritte in VRs durchgeführt werden, sodass diese ähnlich wie Masken in der Fotolithographie fungieren. Dies ist nur möglich, da im Gegensatz zu den zuvor aufgeführten Projekten ein eigener, parametrierbarer Router erstellt wurde.

Wie in Abbildung 3.24 dargestellt, teilt sich der Entwurfsablauf eines DREAMS DPR-Systems in zwei Teile auf. Das *Front-End* besteht aus zwei Schritten. In einem ersten Schritt (*Netlist Generation*) werden alle vorliegenden HDL-Dateien in Netzlisten übersetzt. Dies erfolgt über den Kommandozeilenmodus von Xilinx PlanAhead. Somit werden die gleichen Standard-Werkzeuge der Xilinx ISE verwendet. Bei den erzeugten Netzlisten handelt es sich um bereits platzierte und verdrahtete Netzlisten im NCD-Format. Die Platzierungsinformation werden dabei der XML-Datei System Modules entnommen. Der zweite Schritt analysiert das spezifizierte System und identifiziert VBs und deren Anwendung in VRs.

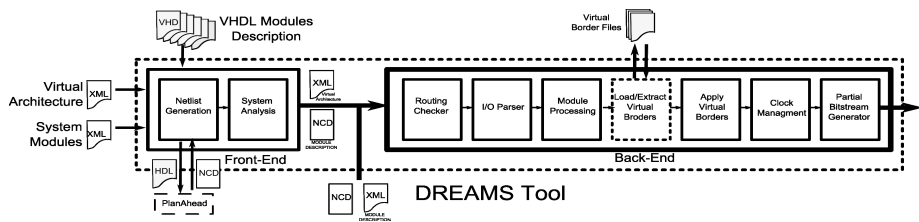


Abbildung 3.24: Entwurfsablauf eines DREAMS DPR-Systems [132, S. 6]

Das *Back-End* besteht aus sieben Schritten. Der erste Schritt überprüft die Verdrahtung der Module auf Konflikte. Treten diese auf, werden sie durch den eigenen Router behoben. Der *IO Parser* erkennt welche Ports im System rekonfigurierbare Verbindungen enthalten und somit Teil einer VB sind. Im dritten Schritt werden die Module eingelesen. Die VBs können in einem vierten Schritt abgespeichert oder gelesen und somit wiederverwendet werden. Die Anwendung der VBs erfolgt anschließend im fünften Schritt. Der sechste Schritt stellt sicher, dass alle VM, die die gleichen VRs verwenden mit dem gleichen Takt versorgt werden. Die Änderungen an der Verdrahtung der Taktleitungen erfolgt dabei ebenfalls durch den eigenen Router. Abschließend werden die (partiellen) Bitströme erstellt. Die partiellen Bitströme werden dabei, analog zu OpenPR, durch eine eigene Software aus dem vollständigen Bitstrom extrahiert.

DREAMS basiert auf RapidSmith [140] und ist daher in der Programmiersprache Java geschrieben. In [132] wird nur auf die Xilinx Virtex-5-Familie eingegangen, daher ist unklar, inwiefern die Software zu anderen FPGA-Familien kompatibel ist.

3.5.6 INDRA

Mit INDRA (**I**ntegrated **D**esign **F**low for **R**econfigurable **A**rchitectures) von Hagemeyer et al. wurde im Jahr 2007 ein Entwurfsablauf zur Erstellung von DPR-Systemen vorgestellt [75]. Der Entwurfsablauf enthält Software für das Floorplanning, die automatische Generierung der Konfigurationsdateien für die statischen und rekonfigurierbaren Komponenten des Systems und die Erstellung einer homogenen Kommunikationsinfrastruktur. Die PR-Region wird in mehrere Teilabschnitte, sogenannte *PR-Tiles* (dt. Kacheln), unterteilt.

Ein rekonfigurierbares Modul belegt keine einzelne PR-Tile, sondern die minimale Anzahl an benötigten PR-Tiles, wodurch der interne Verschnitt minimiert werden kann. Diese Flexibilität bei der Platzierung der rekonfigurierbaren Module wird durch die Notwendigkeit einer komplexeren, homogenen Kommunikationsinfrastruktur erkauft.

Die Abbildung 3.25 visualisiert den Ablauf der Erstellung eines INDRA DPR-Systems. Als erster Schritt muss ein System in statische und dynamische Komponenten partitioniert werden. Dieser Schritt muss manuell ausgeführt werden. Die statischen und dynamischen Systemkomponenten werden durch eine HDL-Datei oder eine Netzliste beschrieben.

Ist die Platzierung und Laufzeit der einzelnen rekonfigurierbaren Module vor der Laufzeit bekannt, kann ein Ablaufplan (engl. *schedule*) zur Design-Zeit erstellt werden. Ein Ablaufplan kann mit Hilfe der Software *SARA* (**S**imulation **F**ramework for **R**econfigurable **A**rchitectures) auf die Ausführbarkeit des Ablaufplans und der Modulplatzierung getestet werden. Darüber hinaus wird über *SARA* der Algorithmus zur Platzierung der rekonfigurierbaren Module zur Laufzeit bestimmt.

Der nächste Schritt, das *Floorplanning/Layout*, partitioniert die statischen und PR-Regionen entsprechend ihres Ressourcenbedarfs (CLB, BRAM, IO ...) und wird durch die Software *ArchGen* unterstützt. Die Synthese des statischen Bereichs sowie aller rekonfigurierbaren Module wird aufgrund dieser Partitionierung durchgeführt. Anschließend erfolgt die Erstellung der homogenen Kommunikationsinfrastruktur mit der, in der Programmiersprache C++ geschriebenen, Software *X-CMG* (*XDL*⁸ *based Communication Macro Generator*). Die Homogenität garantiert, dass in den PR-Tiles der PR-Region stets die gleichen FPGA-Ressourcen für die Kommunikationsinfrastruktur verwendet werden, sodass identische Anschlusspunkte für die PR-Module zur Verfügung stehen. Dieser Schritt ist der zentrale Punkt zur Erzeugung eines flexiblen DPR-Systems mit Modulplatzierung. Die Kommunikationsinfrastruktur wird, wie bei ReCoBus oder OpenPR, als eingebettetes Makro dem statischen Bereich hinzugefügt. Alle bisher genannten Schritte sind Teile des INDRA Frontends.

Die nachfolgenden Schritte sind zur Erzeugung der (partiellen) Bitströme notwendig und sind Teil des INDRA Backends. *MiDesires* (**M**odule **I**mplementation **D**esign flow for **R**econfigurable **S**ystems) basiert auf den, in Abschnitt 3.5.2 beschriebenen, Xilinx *EA PR* Entwurfsablauf zur Erzeugung der rekonfigurierbaren Module und besteht

⁸Xilinx Design Language

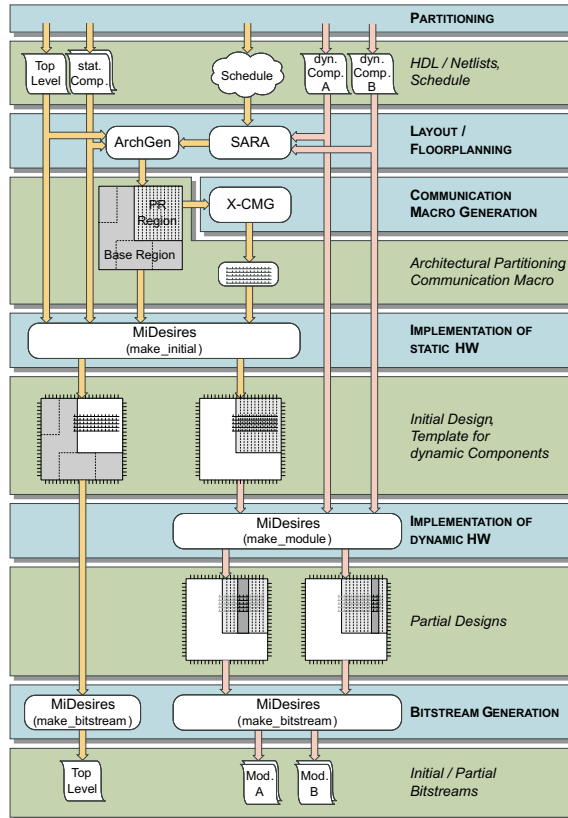


Abbildung 3.25: INDRA – **I**ntegrated **D**esign **F**low for **R**econfigurable **A**rchitectures [75, S. 2]

aus mehreren (Batch/Shell) Skripten. *MiDesires* bietet zusätzlich die Möglichkeit den aktuellen Stand des Entwurfsablaufs zu speichern und anschließend erneut zu laden. Weiterhin werden Abhängigkeiten in dem Entwurfsablauf erkannt und aufgelöst sowie die verwendeten Xilinx Werkzeuge konfiguriert.

Die Trennung zwischen statischen und dynamischen Bereichen erfolgt durch die Blockierung von, nur durch das Design verwendeten, statischen Verdrahtungsressourcen. In den entsprechenden ISE Versionen wurde dies durch eine spezielle Datei (*exclude.arcs*) ermöglicht. Die aufgeführten Verdrahtungsressourcen wurden in dem Verdrahtungsschritt mit dem Werkzeug *PAR* nicht verwendet. Der Vorteil dieser Implementierung ist, dass nicht alle möglichen Verdrahtungsressourcen mit einem potenziellem Übergang zwischen dynamischen und statischen Bereichen blockiert werden müssen.

Durch die Verwendung des Xilinx *EA PR* Entwurfsablaufs werden die FPGA-Familien Virtex-II (Pro), Spartan-3 sowie Virtex-4 unterstützt. Der Entwurfsablauf unterstützt prinzipiell auch einzelne Virtex-5-FPGAs (siehe Abschnitt 3.5.2). Aufgrund des nachfolgend beschriebenen, aktualisierten INDRA 2.0 Entwurfsablaufs, wurden einzelne Teilkomponenten wie z. B. X-CMG nicht weiterentwickelt, sodass Virtex-5-FPGAs offiziell nicht unterstützt werden.

3.5.7 INDRA 2.0

INDRA 2.0 ist die Weiterentwicklung des INDRA Ansatzes mit den Zielen der Unterstützung aktueller Xilinx FPGA-Familien und der Kompatibilität zum Xilinx *PR* Entwurfsablauf. Der Entwurfsablauf wurde im Rahmen der Dissertation von Cozzi und dieser Dissertation konzipiert und durch Softwarekomponenten realisiert. In diesem Abschnitt wird INDRA 2.0 detailliert vorgestellt und auf die Änderungen zur ersten Version eingegangen. Die einzelnen Schritte zur Erstellung eines INDRA 2.0 DPR-Systems sind in der Abbildung 3.26 dargestellt und werden in den folgenden Abschnitten vorgestellt. Der Entwurfsablauf ist im Vergleich zur ersten INDRA Version näher an dem Ansatz des Herstellers angelehnt. Anstelle des Xilinx *EA PR* wird der Xilinx *PR* Entwurfsablauf verwendet. Hierdurch werden die aktuellen FPGA-Familien und Software Werkzeuge von Xilinx unterstützt. Der Tcl-basierte *PR* Entwurfsablauf (siehe Abschnitt 3.5.2) wurde in einzelnen Schritten angepasst bzw. erweitert. So wird die dort angesprochene Problematik von statischen Netzen, die Verdrahtungsressourcen des dynamischen Bereichs verwenden und somit die Homogenität des DPR-Systems zerstören, durch einen Nachbearbeitungsschritt gelöst. In der Abbildung 3.26 ist die Problematik durch eine gestrichelte rote Linie in dem *Initialdesign* dargestellt. In dem darauffolgendem Nachbearbeitungsschritt wird dafür gesorgt, dass diese Inhomogenität durch eine Neuverdrahtung der betroffenen Netze behoben wird, sodass eine Umplatzierung von dynamischen Komponenten ermöglicht wird.

Partitionierung

In einem ersten Schritt (*DPR-Part*) muss ein existierendes, rein statisches System in statische und dynamische Komponenten partitioniert werden. Dieser Schritt muss, analog zu den existierenden, in Abschnitt 3.5 vorgestellten, Entwurfsabläufen manuell ausgeführt werden. Die statischen und dynamischen Systemkomponenten sind hierbei entweder durch eine HDL-Datei oder eine Netzliste beschrieben.

Layout/Floorplanning

Für das Layout oder Floorplanning, die Erstellung von Partitionen für dynamische und statische Bereiche, kann die Xilinx Software *PlanAhead* verwendet werden. Einzelne Partitionen können hierbei in der FPGA-Darstellung als Rechteck eingezeichnet werden, wobei die resultierenden FPGA-Ressourcen (z. B. CLBs, BRAMs oder DSPs) simultan zur Zeichnung angezeigt werden. Ein vollständiges Layout kann abschließend in Form einer UCF mit entsprechenden AREA_GROUP Bedingungen abgespeichert werden. Eine

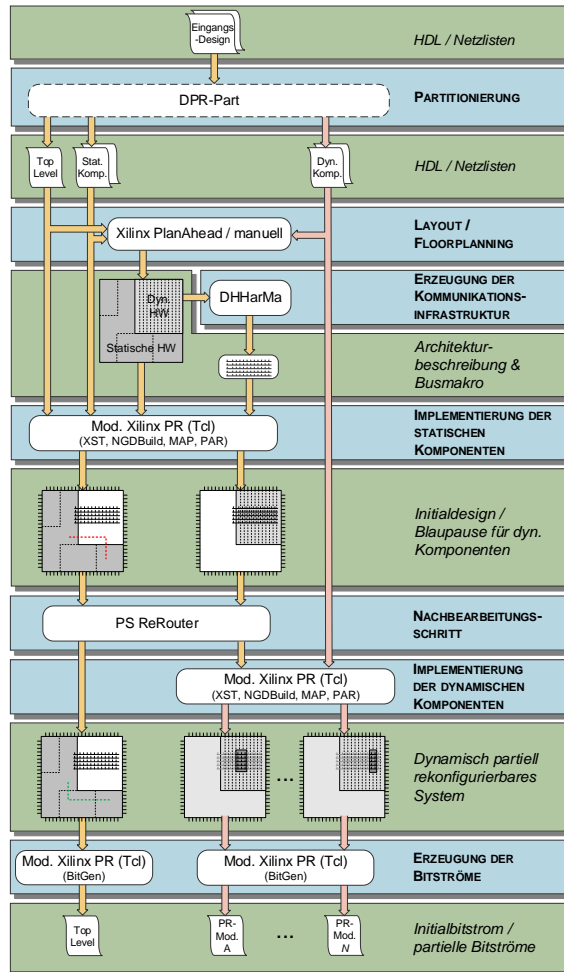


Abbildung 3.26: INDRA 2.0 – Integrated Design Flow for Reconfigurable Architectures

Partition sollte dabei unter Einhaltung der, in Abschnitt 3.3.4 beschriebenen, Hinweise erstellt werden und sich daher an der Grenze der *Configuration Frames*, der kleinsten rekonfigurierbaren Einheit, orientieren. Diese stimmt mit der Höhe der Taktregion überein, ist, wie in Abschnitt 3.3.1 beschrieben, familienspezifisch und variiert bei den unterstützten FPGA-Familien zwischen 16 CLBs (Virtex-4) und 50 CLBs (7-Serie).

Erzeugung der Kommunikationsinfrastruktur

Um die Flexibilität des Systems durch die Unterstützung aller Rekonfigurationsarten (siehe Abschnitt 3.5.1) erhöhen zu können, wird eine homogene Kommunikationsinfrastruktur als eingebettetes Makro zwischen dynamischen und statischen Bereichen benötigt. Die Homogenität garantiert, dass in Tiles vom gleichen Typ stets die gleichen FPGA-Ressourcen verwendet werden, wodurch identische Anschlusspunkte für die PR-Module zur Verfügung stehen. Dieser Schritt ist somit der zentrale Punkt zur Erzeugung eines flexiblen DPR-Systems mit Modulplatzierung.

Die Erstellung der Kommunikationsinfrastruktur eines INDRA DPR-Systems erfolgt durch die Software *X-CMG*, welche detailliert in [76, S. 49 ff.] von Hagemeyer beschrieben wird. Die Software verwendet zur Erzeugung der homogenen Struktur sogenannte *Bus-Makro Primitive*. Diese Primitive müssen in einem ersten Schritt händisch mit Hilfe der Xilinx Software *FPGA Editor* erzeugt werden. Ein Bus-Makro Primitive beschreibt dabei einen Teil der Kommunikationsinfrastruktur. Die Software *X-CMG* konkateniert mehrere dieser erstellten Bus-Makro Primitive zur Erzeugung eines vollständigen Bus-Makros (siehe Abbildung 3.27).

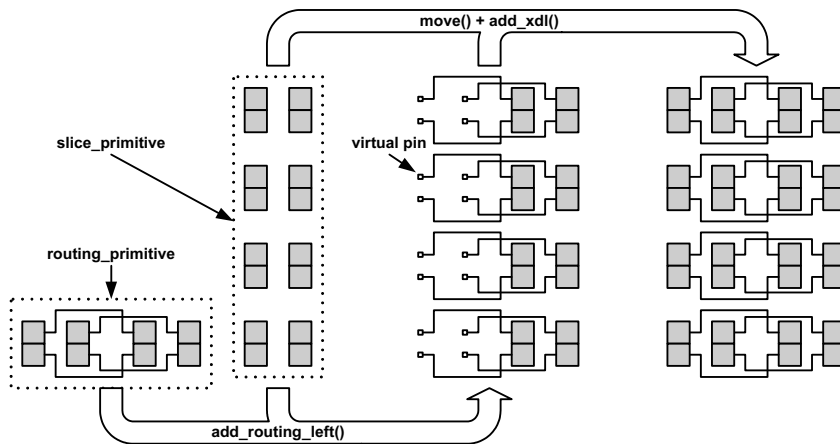


Abbildung 3.27: Erstellung eines Bus-Makros mit der Software *X-CMG* [76, S. 59]

Ein Bus-Makro Primitive ist FPGA-Familien-spezifisch, oftmals aufgrund der von FPGA zu FPGA variierenden, spaltenweise angeordneten FPGA-Ressourcen sogar FPGA-spezifisch. Somit ist die Wiederverwendbarkeit der Bus-Makro Primitive stark eingeschränkt. Die Erstellung eines Bus-Makro Primitives ist weiterhin sehr zeitintensiv, erfordert fundierte Kenntnisse der jeweils zugrunde liegenden FPGA-Architektur (insbesondere der Verdrahtungsinfrastruktur) und ist durch den manuellen Entwurf mit dem *FPGA Editor* fehleranfällig.

Dies sind die vorrangigen Gründe des Wunsches einer Automatisierung dieses Schrittes zur Erstellung der homogenen Kommunikationsinfrastruktur, welche durch die Softwarekomponenten von DHHarMa (Design Flow für **H**omogene **H**ard-**M**akros), zur automatisierten Erzeugung homogener Strukturen für Xilinx FPGAs, realisiert und detailliert in Kapitel 4 vorgestellt werden. Die Bestandteile des Kommunikationsmakros können nun durch eine Hardwarebeschreibungssprache (engl. **H**ardware **D**escription **L**anguage, HDL) spezifiziert werden, sodass FPGA-unabhängige Primitive deklariert werden können. Dadurch ist eine uneingeschränkte Wiederverwendbarkeit gegeben und es kann eine Bibliothek von HDL-Primitiven erstellt werden, welche unabhängig von der verwendeten FPGA-Familie ist. Durch die DHHarMa-Software ist es ebenfalls nicht mehr nötig, die Verdrahtungsinfrastruktur im Detail zur Implementierung des Kommunikationsmakros kennen zu müssen.

Die homogene Kommunikationsinfrastruktur wird von DHHarMa in einem Xilinx-konformen Dateiformat erzeugt und den weiteren Xilinx Softwarekomponenten des PR Entwurfsablaufs hinzugefügt.

Implementierung der statischen Komponenten

Die Implementierung der statischen Komponenten des DPR-Systems erfolgt durch eine modifizierte Version des Tcl-basierten Xilinx PR Entwurfsablaufs. Durch die Verwendung von (AREA_GROUP) Attributen (siehe Abschnitt 3.3.4 oder [202, S. 46 ff.]) kann die Packung und Platzierung für Bereiche vorgegeben werden. Eine Parametrierung des Xilinx Routers (PAR) ist nahezu nicht möglich. Die Meidung eines rekonfigurierbaren Bereichs ist prinzipiell durch das nicht dokumentierte (AREA_GROUP) Attribut *PRIVATE=ROUTE* möglich. Durch dieses Attribut wird lediglich den zugewiesenen Komponenten einer Partition die Verwendung der Verdrahtungsressourcen erlaubt. Das Attribut muss daher den AREA_GROUP Bedingungen der rekonfigurierbaren Partitionen hinzugefügt werden. Das eingebettete Kommunikationsmakro zerstört jedoch diese klare Abgrenzung, sodass dieses Attribut nicht verwendet werden kann. Somit ist es möglich, dass die Software PAR zur Verdrahtung von Verbindungen in statischen Bereichen Verdrahtungsressourcen der dynamischen Regionen verwendet, wodurch eine Inhomogenität im DPR-System entsteht. In diesem Fall wird ein Nachbearbeitungsschritt benötigt, welcher eine Neuverdrahtung der betroffenen Signalleitungen durchführt. Bevor auf diesen Schritt eingegangen wird, wird zuvor der Tcl-basierte Xilinx PR Entwurfsablauf sowie Modifikationen und Erweiterungen dieses Entwurfsablaufs beschrieben.

Tcl-basierter Xilinx PR Entwurfsablauf

Der Tcl-basierte Xilinx PR Entwurfsablauf ist in dem *Partial Reconfiguration User Guide* [211, S. 83 ff.] beschrieben. Neben der Dokumentation wird eine Tcl-Skriptsammlung bereitgestellt, welche für INDRA 2.0 modifiziert wurde.

Die Tcl-Skriptsammlung besteht aus einem Hauptskript (*xpartition.tcl*) welches den allgemeinen Entwurfsablauf zur Erstellung des DPR-Systems mit mehreren Konfigu-

rationen (siehe Abschnitt 3.5.2) kontrolliert. Der Ablauf wird hierbei durch Aufrufe dreier weiterer Skripte realisiert:

- *gen_xp.tcl*: Erstellung von (XML-basierten) Partitionierungsdateien für das DPR-System
- *implement.tcl*: Kontrollierter Aufruf einzelner Xilinx Werkzeuge (*XST*, *NGDBuild*, *MAP*, *PAR* und *BitGen*; eine kurze Beschreibung der ersten drei Werkzeuge folgt in Abschnitt 4.2.1, Details zu allen Werkzeugen in [200])
- *export.tcl*: Exportiert generierte Dateien der statischen Module für eine zukünftige Verwendung (Attribut *STATE=import*, siehe Abschnitt 3.5.2)

Die Konfiguration des DPR-Systems erfolgt durch eine weitere Tcl-Datei (*data.tcl*), welche dem Hauptskript als Eingangsdatei übergeben wird. Der Aufruf zum Start des Entwurfsablaufs erfolgt über das folgende Kommando:

```
xtclsh xpartition.tcl data.tcl
```

Die Abbildung 3.28 stellt den Inhalt einer Konfigurationsdatei des Xilinx *PR* Entwurfsablaufs dar. Die Konfigurationsdatei enthält zehn mögliche Konfigurationsparameter, welche in vier Abschnitte unterteilt sind. Der erste Abschnitt enthält die ersten fünf projektspezifischen Konfigurationsoptionen. Im zweiten Abschnitt werden die einzelnen Module aufgeführt, welche anschließend synthetisiert werden, und die Partitionsattribute pro Modul angegeben. Die Xilinx *configurations* werden in dem dritten Abschnitt definiert und in einer Liste die Implementierungsreihenfolge vorgegeben. Im vierten und letzten Abschnitt werden Implementierungsoptionen definiert, sodass die einzelnen Xilinx Werkzeuge (de-) aktiviert und parametrisiert werden können.

Richtlinien des Xilinx PR Entwurfsablaufs

Bei der Verwendung des Tcl-basierten *PR* Entwurfsablaufs müssen einige Richtlinien eingehalten werden. Aufgrund der festen, relativen Pfade in den Tcl-Skripten muss eine strikte Ordnerstruktur eingehalten werden (siehe Abbildung 3.29). Die Skriptsammlung liegt in dem Ordner *Tools*, die Syntheseergebnisse in dem Ordner *Synth* und die Implementierungsergebnisse jeder Xilinx *configuration* in dem Ordner *Implementation*. Für jedes rekonfigurierbare Modul (*RM*) muss ein dedizierter Unterordner in dem Ordner *Synth* erstellt werden. Weiterhin muss für jedes *RM* eine ISE-Projektdatei (*PRJ*) sowie eine Datei mit den Einstellungen für das Xilinx Synthese-Werkzeug (*XST*) in dem entsprechenden Unterordner angelegt werden. Weiterhin muss für jede Xilinx *configuration* ein Unterordner in dem Ordner *Implementation* händisch hinzugefügt werden. Die Ordner mit dem Prefix *X* enthalten eine Kopie der Implementierungsergebnisse einer entsprechenden Xilinx *configuration* und dienen zum Importieren in weiteren Konfigurationen.

```

# 1:environment variables for all configurations
# set ::env(XIL_TIMING_ALLOW_IMPOSSIBLE) 1

# 2:part definition
set PART xc4vfx100-11-ff1152

# 3:constraints file
set UCF ../../../../vhdl/pr_fpga.ucf

# 4:Partition names:
# These names must match the actual instance names in the design
set TOP_PART /pr_fpga_left
set SLOT_A ${TOP_PART}/slot_a
...

# 5:RM names
set PR_FPGA pr_fpga_left
set SLOT_A_IMAGE Slot_A_Image
set SLOT_A_BB Slot_A_Blank
...

# 6:RM list: Each RM in the list will be synthesized with bottom-up synthesis.
# You must create a directory for each of the RMs in the list
set RMs [list $SLOT_A_IMAGE $SLOT_B_FIR $SLOT_C_FFT $SLOT_D_BRAM $PR_FPGA ]

# 7:Partition Attributes List
set PartitionAttrsList {
  {/pr_fpga_left}
  {/pr_fpga_left/slot_a {Reconfigurable true}}
  ...
}

# 8:Configuration Information
set CONFIG_BlankConfig {
  {ConfigName CfgBlank}
  {Settings
    {/pr_fpga_left {State implement} }
    ...
  }
}
set CONFIG_Init {
  {ConfigName CfgInit}
  {Settings
    {/pr_fpga_left {State import} {ImportLocation ../XCfgBlank} }
    ...
  }
}

# 9:List of configurations in order of implementation
set ALL_CFGS [list $CONFIG_BlankConfig $CONFIG_Init ]

# 10:Implementation options
# set optional implementation data flags
array set IMPLEMENTATION_DATA { \
    SYNTH_TOOL xst \
    RUN_RM_SYNTH YES \
    ...
}

```

1. Projektspezifische Konfigurationsoptionen

2. Modulliste und Partitionsattribute

3. Definition und Implementierungsreihenfolge der Xilinx configurations

4. Implementierungsoptionen

Abbildung 3.28: Konfigurationsdatei des Xilinx PR Entwurfsablaufs

Modifikationen und Erweiterungen des Tcl-basierten Xilinx PR Entwurfsablaufs

Bei der Verwendung der Xilinx Tcl-Skriptsammlung ergaben sich einige Änderungen in einzelnen Tcl-Skripten, welche zum einen durch den benötigten Nachbearbeitungsschritt und zum anderen durch fehlende Einstellungsmöglichkeiten begründet sind.

Fehlende Einstellungsmöglichkeiten Zu den letztgenannten fehlenden Einstellungsmöglichkeiten gehört die Übergabe des Pfades für die erstellten Netzlisten jedes RMs für die Implementierung des Gesamtsystems. Die Pfadübergabe erfolgt durch die Hin-

```
Implementation # Implementierungsergebnisse jeder configuration
- CfgBlank
- CfgInit
- ...
- XCfgBlank
- XCfgInit
Synth # Syntheseergebnisse jedes RMs
- Slot_A_Image
- ...
Tools # Bibliothek der Tcl-Skripte
- xpartition.tcl
- gen_xp.tcl
- implement.tcl
- export.tcl
- data.tcl
```

Abbildung 3.29: Ordnerstruktur des Xilinx PR Entwurfsablaufs

zunahme einer weiteren Variablen in dem Skript *implement.tcl* und der Modifikation des Aufrufs in der Prozedur *implement* des Tcl-Skripts *xpartition.tcl*.

Eine weitere Änderung ist die Unterstützung mehrerer UCFs. Insbesondere bei modularen, IP-Core-basierten Systemen ist dies wünschenswert und wird durch die Xilinx Werkzeuge unterstützt. Die Änderung wurde ebenfalls in dem Tcl-Skript *implement.tcl* in dem Implementierungsschritt *Translate* durch den Aufruf des Werkzeugs *NGDBuild* vorgenommen.

Der vierte Abschnitt der Konfigurationsdatei (*data.tcl*) enthält die Implementierungsoptionen für die einzelnen Xilinx Werkzeuge. Für das Werkzeug *BitGen* fehlt dies jedoch und wurde daher dem existierenden Tcl-Array *IMPLEMENTATION_DATA* hinzugefügt und in dem Skript *implement.tcl* bei dem Aufruf des Werkzeugs berücksichtigt.

Die letzte Modifikation ist die Entkopplung eines prozessorbasierten DPR-Systems, welches durch die Xilinx Software *Embedded Development Kit* (EDK) unterstützt wird (eine kurze Einführung zu EDK folgt in Abschnitt 6.1.2). Für die Entkopplung werden die wesentlichen, EDK-basierten Dateien in den Ordner der jeweiligen Xilinx *configuration* kopiert. Der Kopiervorgang wird in dem Tcl-Skript *xpartition.tcl* durchgeführt. Durch diese Entkopplung ist die benötigte Konsistenz zwischen Hardware- und Softwareteil des EDK-Systems gewährleistet, sodass z. B. nachträglich ein Softwareupdate erfolgen kann.

Integration des Nachbearbeitungsschritts Ein benötigter Nachbearbeitungsschritt des statischen Bereichs erfolgt durch zusätzliche Werkzeugaufrufe. Die Detektion von statischen Netzen, welche Verdrahtungsressourcen eines dynamischen Bereichs verwenden, erfolgt durch ein Perl-Skript (*checkRouting.plx*). In dem Falle detektierter Netze, wird das Xilinx Werkzeug *TRACE* (*Timing Reporter And Circuit Evaluator*, [200, S. 179 ff.]) zur statischen Zeitanalyse ausgeführt und der Nachbearbeitungsschritt initialisiert.

Nachbearbeitungsschritt / Post Synthesis ReRouter

Kreuzende Signalleitungen führen zur Inhomogenität im DPR-System, wodurch eine Umplatzierung von PR-Modulen verhindert wird. Kreuzende Signale treten dabei z. B. durch vorgegebene IOB-Belegungen auf, welche die Partitionierung des FPGAs einschränken. Zur Behandlung dieser Inhomogenität wurden unterschiedliche Herangehensweisen untersucht, welche folgend beschrieben werden.

Directed Routing Ein erster Ansatz ist die Verwendung von sogenannten *DIrected RouTing (DIRT)*, [202, S. 85 f.] Bedingungen, welche einer UCF hinzugefügt werden können. Diese Bedingungen enthalten Informationen über die verwendeten Verdrahtungsressourcen und können über den Xilinx *FPGA Editor* für ein Netz erstellt werden. Zusätzlich ist die Hinzunahme von Platzierungsinformationen der verwendeten Blöcke möglich. Ein betroffenes Netz kann manuell oder automatisch von dem Router des *FPGA Editors* neu verbunden werden. Je größer der Abstand zwischen der Quelle und der (oder den) Senke(n) des Netzes, desto geringer ist die Erfolgswahrscheinlichkeit der Neuverdrahtung. In dem Fall ist häufig eine zeitintensive, manuelle Neuverdrahtung nötig. Ein Anwender muss daher über detaillierte Kenntnisse der zugrundeliegenden Verdrahtungsinfrastruktur verfügen. Die erzeugten DIRT eines Netzes sind nicht direkt aufschlüsselbar:

```
NET "net_name"ROUTE="2;1;-4!-1;-53320;2920;14;...
```

Die Verwendung der DIRT-Bedingungen wird weiterhin nur für eine geringe Anzahl an Netzen empfohlen. Darüber hinaus, kann es bei Änderungen an statischen Designkomponenten zu weiteren kreuzenden Netzen kommen, sodass die Anzahl der DIRT-Bedingungen in dicht gepackten DPR-Systemen die Xilinx Werkzeuge stark einschränkt. Aufgrund dieser Einschränkungen und da eine Automatisierung dieser Herangehensweise nicht möglich ist, stellen DIRT-Bedingungen keine zufriedenstellende Lösung für einen vollständigen DPR-Entwurfsablauf dar.

Blockierung der Verdrahtungsressourcen Eine Alternative bieten die Blockierung sämtlicher Verdrahtungsressourcen zwischen statischen und dynamischen Bereichen. Dieser Ansatz wird erstmals durch *Blocker-Makros* in den *ReCoBus* DPR-Systemen (siehe Abschnitt 3.5.3) vorgestellt und auch in *OpenPR* DPR-Systemen (siehe Abschnitt 3.5.4) durch *blocking nets* praktiziert. Der wesentliche Nachteil der Verwendung dieses Ansatzes ist die Auswirkung auf den abschließenden Verdrahtungsschritt. So kann die Blockierung von Verdrahtungsressourcen bei komplexeren DPR-Systemen zu langen Ausführungszeiten, bis hin zur Terminierung bzw. Blockierung des Werkzeugs *PAR* führen.

XDL-Manipulation Eine proprietäre Xilinx Design-Datei kann mittels der Xilinx Software *XDL (Xilinx Design Language)* in eine lesbare ASCII-Datei überführt werden. Ebenfalls ist eine Überführung einer XDL-Datei in eine Design-Datei möglich. Daher wird dieses Zwischendateiformat oftmals für Modifikationen an Designs verwendet. Details zu

der Sprache folgen in Abschnitt 4.3. Eine Manipulation über diese Zwischendatei XDL für ein vollständiges Design ist jedoch nicht möglich, da die Konfiguration einiger Blocktypen (z. B. DCM oder IOB) nicht in die XDL-Datei exportiert wird und somit durch die Konvertierung verloren geht.

Manipulation durch den FPGA Editor Eine Manipulation an einem Design kann somit nur an der proprietären Design-Datei selbst durchgeführt werden. Die Software *FPGA Editor* ermöglicht eine skriptbasierte Design-Manipulation, wie z. B. eine automatische Verdrahtung eines existierenden Netzes an einer solchen geöffneten Design-Datei. Wie bereits erwähnt, kann bei einer automatischen Verdrahtung nicht sichergestellt werden, dass das neuverdrahtete Netz Verdrahtungsressourcen einer dynamischen Region verwendet. Die Verdrahtung des Netzes muss daher konkret von der Quelle über die einzelnen Leitungen und PIPs bis zu der (oder den) Senke(n) des Netzes vorgegeben werden. Bei der manuellen Verdrahtung durch den *FPGA Editor* muss bei der Auswahl der FPGA-Ressourcen die Reihenfolge genau eingehalten werden, da die neue Verbindung ansonsten nicht übernommen wird. Der Nachteil dieser Lösung ist, dass keine freie Datenbank für den *FPGA Editor* existiert, sodass eine Automatisierung der manuellen Neuverdrahtung nicht direkt möglich ist. Weiterhin ist die Erkennung der betroffenen Netze sowie ein Router, welcher während der Verdrahtung die Grenzen der dynamischen Bereiche respektiert, zur Realisierung dieses Ansatzes nötig. Durch die existierenden DHHarMa-Softwarekomponenten (siehe Kapitel 4) ist die Behandlung der letzten Punkte durch Software-Erweiterungen umsetzbar.

Aus den dargestellten Gründen ist die Entscheidung zugunsten der letzten Herangehensweise gefallen. Eine wesentliche Komponente stellt hierbei der, von dem DHHarMa-Router abgeleitete, *PS ReRouter* (**P**ost **S**ynthesis **R**eRouter (siehe Abbildung 3.26) von Cozzi dar, welcher in [43] beschrieben wird. Die Erstellung der benötigten *FPGA Editor* Datenbank ist in Abschnitt 4.5.1 beschrieben und erweitert die DHHarMa-Softwarekette. Abschließend wurde der existierende *FPGA Editor* Skript-Generator der DHHarMa-Softwarekette erweitert, sodass die ermittelten, neu-verdrahteten Netze in einem geöffneten Design integriert werden können.

Abbildung 3.30 stellt die wesentlichen Teile des erstellten *FPGA Editor* Skripts zur Neuverdrahtung betroffener Netze dar. Der erste Teil enthält den Befehl zum Öffnen eines spezifischen Designs und setzt allgemeine Einstellungen für den *FPGA Editor* bzw. der Kommandozeilenversion *FPGA Edline*. Im zweiten Teil sind die Befehle zur Neuverdrahtung betroffener Netze aufgeführt. Exemplarisch ist ein Ausschnitt des Netzes *RX_CLK_IBUF* dargestellt. Vor der eigentlichen Neuverdrahtung müssen zunächst die FPGA-Verdrahtungsressourcen des existierenden Netzes freigegeben werden (*unroute*). Anschließend werden die neuen FPGA-Verdrahtungsressourcen (beginnend von der Quelle) ausgewählt und über das Kommando *route* übernommen. Wie dargestellt, ist die Auswahl des Ein- und Ausgangspins sowie aller PIPs (*arc*) hierbei optional. Nachdem alle ermittelten Netze angepasst wurden, wird abschließend das nun modifizierte

```
#####
# 1. Properties:
open design D:\work\SVN\indra\examples\reroute\V4\FX100\ESA\CfgBlank.ncd
setattr main edit_mode read-write
setattr main auto_deselect true
setattr main auto_post false
setattr main case_sensitive false
setattr main wire_info on
setattr main show_tiles on
setattr main auto_route off
setattr main enhanced_routing off
setattr main stub_trim off
#setwin list1 state min
#setwin world1 state min
#setwin array1 state min
unselect -all
#####

#####
# 2. Reroute nets:
# - 1. Net: RX_CLK_IBUF
select net 'RX_CLK_IBUF'
unroute
unselect -all
#select pin 'J6.I'
select wire -id 3952164; # IOIS_LC_X76Y119 IOIS_IBUF0 -> IOIS_IBUF_PINWIRE0
#select arc -k PADOUTPUT(146824,135656)--->IOBIN2OUT(146680,135656)
select wire -id 3949768; # IOIS_LC_X76Y119 IOIS_IBUF_PINWIRE0 -> IOIS_I0
#select arc -k IOBIN2OUT(146680,135656)--->IOBOUTPUT(146600,135792)
select wire -id 3943454; # IOIS_LC_X76Y119 IOIS_I0 -> BEST_LOGIC_OUTS0_INT
#select arc -k IOBOUTPUT(146600,135792)--->OUTBOUND(146045,135368)
select wire -id 3918149; # INT_X76Y119 BEST_LOGIC_OUTS0 -> OMUX0

...

select wire -id 2132222; # INT_X42Y28 W2MID6 -> CLK_B3
#select arc -k DOUBLE(-1736,-175031)--->PINFEED(-411,-175912)
select wire -id 2139634; # DCM_BOT_X42Y28 CLK_B3_INT0 -> DCM_ADV_CLKIN
#select arc -k PINFEED(-411,-175912)--->CLKPIN(136,-176376)
select wire -id 2156288
#select pin 'DCM_ADV_X0Y5.CLKIN'
route
unselect -all

...
#####

save
quit!
```

Abbildung 3.30: Ausschnitte eines *FPGA Editor* Skripts zur Neuverdrahtung von statischen Netzen

Design gespeichert und die Ausführung des Skripts terminiert. Die Ausführung des Skripts erfolgt durch die Kommandozeilenversion des *FPGA Editors*, *FPGA Edline*, durch den Aufruf:

```
fpga_edline -p reroute.scr
```

Nachdem dieser Nachbearbeitungsschritt durchgeführt wurde, kann der allgemeine Entwurfsablauf weiter fortgeführt werden. Um Änderungen am Zeitverhalten durch die Modifikationen am Design erkennen zu können, wird das Xilinx Werkzeug *TRACE* ein zweites mal ausgeführt und dem Zeitverhalten vor der Modifikation gegenübergestellt. Nur bei einer Übereinstimmung wird der Entwurfsablauf weiter fortgesetzt. Ergibt sich

eine Abweichung im Zeitverhalten aufgrund der Modifikationen muss der Router nach manuellen Änderungen an den Router-Parameters erneut ausgeführt werden.

Implementierung der dynamischen Komponenten

Die Synthese der einzelnen Module erfolgt in der, in *RM list* (siehe Abbildung 3.28) festgelegten Reihenfolge. Die erste Xilinx *configuration* (*CfgBlank*) enthält lediglich die *Blank/Black-Box*-Implementierungen (siehe Abschnitt 3.3.4). Die Implementierung der realen dynamischen Komponenten erfolgt somit nach einem potentiellen Nachbearbeitungsschritt am statischen Design. Das (modifizierte) statische Design wird in einer weiteren Xilinx *configuration* importiert und die dynamischen Module konkreten *Partitionen* zugeordnet. Die Anzahl benötigter Konfigurationen hängt neben der Anzahl dynamischer Komponenten von der DPR-Systempartitionierung und dem damit verbundenen Typ der Kommunikationsinfrastruktur ab (siehe Abschnitt 4.4.4).

Erzeugung der Bitströme

Die Konfigurationsdateien werden für jede Xilinx *configuration* in dem dazugehörigem Ordner erstellt. Die Erstellung der partiellen Konfigurationsdateien erfolgt automatisch für als rekonfigurierbar deklarierte Partitionen (siehe Modulliste *PartitionAttrsList* in Abbildung 3.28). Die partiellen Konfigurationsdateien zur Löschung einer rekonfigurierbaren Partition (*Blank/Black-Box*-Implementierungen, siehe Abschnitt 3.3.4) sind in dem Ordner der *configuration CfgBlank* abgelegt. Die Konfigurationsdatei einer Xilinx *configuration* enthält stets die enthaltenen RMs. Wird eine initiale Konfiguration mit einer konkreten Belegung von RMs gewünscht, so kann die entsprechende Konfigurationsdatei ausgewählt werden. Ist die Belegung der Partitionen zum Startzeitpunkt noch nicht bekannt (z. B. durch Ereignisse/Umwelteinflüsse beeinflusst) kann das rein statische Design (Konfiguration *CfgBlank*) als Initialdesign verwendet werden.

3.5.8 Vergleich der Ansätze

Ein DPR-System basierend auf dem Xilinx *PR* Entwurfsablauf eignet sich für ein DPR-System mit einer geringen Anzahl von Regionen für rekonfigurierbare Module. Der Fokus liegt auf einer Überführung eines statischen Designs in ein DPR-System auf ein kleineres, kostengünstigeres FPGA, wodurch auch die Verlustleistung eines Gesamtsystems abnimmt. Die Erstellung eines großen DPR-Systems ist nur schwer realisierbar und mit einem hohen Wartungsaufwand verbunden. Durch die Verwendung einer LUT (*proxy logic*) als Verbindungspunkt, welche in Partitionen unterschiedlich verdrahtet sein kann, ergibt sich eine feste Position der PR-Module. Die Verbindungen zur *proxy logic* kann sich mit jeder Änderung am statischen Design ebenfalls ändern. Dies hat zur Folge, dass alle PR-Module neu synthetisiert und die entsprechenden Bitströme generiert werden müssen. In [18, S. 4] wird dieses Problems durch ein konkretes Beispiel eines DPR-Systems mit 10 möglichen Modulpositionen (*Partitions*) und 5 unterschiedlichen Modulen verdeutlicht. Eine Änderung an dem statischen System hätte zur Folge, dass 50 neue partielle Bitströme generiert werden müssen. Weiterhin müssen

alle unterschiedlichen Versionen eines Moduls gespeichert werden. Dies kann in einem kleinen, eingebetteten Design mit geringen Speicherkapazitäten zu einem Problem führen.

Die DPR-Systeme von ReCoBus, GoAhead, OpenPR, DREAMS, INDRA und INDRA 2.0 hingegen verfolgen das Ziel, feingranulare, flexible DPR-Systeme mit einer größeren Anzahl an Modulpositionen und Modulen realisieren zu können. Anwendungen, wie ein Co-Prozessor mit Modulen für optionale Erweiterungssätze oder Verschlüsselungsalgorithmen, benötigen ein solches flexibles DPR-System. Durch feingranulare, kachelbasierte Ansätze wird eine höhere Flexibilität in der Platzierung von Modulen erreicht. Insbesondere bei unterschiedlich großen Modulen ergibt sich eine bessere Ressourcenausnutzung, da der Verlust durch den internen Verschnitt geringer ist als beim Xilinx-basierten Ansatz. Die Verwendung von Eingebetteten Makros ermöglichen einen kachelbasierten Ansatz. Der manuelle Entwurf ist jedoch aufwändig und erfordert Expertenwissen. Die meisten Schritte bei dem Entwurf sind in den genannten Entwurfsabläufen jedoch automatisiert und daher für den Anwender transparent. Entscheidend für die Eingebetteten Makros ist die Homogenität, welche erst eine Umplatzierung von Modulen erlaubt und somit zentraler Punkt eines jeden feingranularen, flexiblen DPR-Systems ist. In den genannten Projekten werden unterschiedliche Verfahren zur Einhaltung der Homogenität in der Kommunikationsinfrastruktur zwischen dem dynamischen und dem statischen Bereich verwendet. Den geringsten Einfluss auf die Xilinx Werkzeuge hat der INDRA 2.0 Ansatz durch einen Nachbearbeitungsschritt in Form einer Neuverdrahtung von Signalen, die Verdrahtungsressourcen einer PR-Region verwenden. In [18, S. 4 ff.] erfolgt eine Gegenüberstellung einiger genannter PR-Projekte. In Anlehnung wurde Tabelle 3.11 erstellt, um aktuelle Projekte erweitert und die zuvor textuell herausgearbeiteten Unterschiede zusammengefasst.

Tabelle 3.11: Gegenüberstellung grob- und feingranularer DPR-Systeme

	Xilinx PR	OpenPR	DREAMS	ReCoBus	GoAhead	INDRA	INDRA 2.0
Floorplanning	manuell (PlanAhead)	manuell (PlanAhead)	manuell (PlanAhead)	manuell (GUI)	manuell (GUI) / autom.	manuell	manuell (PlanAhead)
Kommunikation	proxy logic	Eingebettetes Makro	Virtual Border	Eingebettetes Makro	Eingebettetes Makro	Eingebettetes Makro	Eingebettetes Makro
Entkopp. Entwurf	Nein	Ja	Ja	Ja	Ja	Ja	Ja
Unterst. FPGA-Fam.	V4-6, 7S	V4-5	V5	VII(Pro), S3	V4-6, S6, 7S	VII(Pro)-4	V4-6, 7S
Rekonf.-Art							
-Single Island	Ja	Ja	Ja	Ja	Ja	Ja	Ja
-Multi Island	Nein	Ja	Ja	Ja	Ja	Ja	Ja
(Verfahren)		(block. nets)	(Router)	(Blocker-Makros)	(Blocker-Makros)	(Exclude Arcs)	(Post-Router)
-1D-Slot	Nein	Ja	Ja	Ja	Ja	Ja	Ja
-2D-Grid	Nein	Nein	Ja	Ja	Ja	Ja	Ja
PR-Lizenz	Ja	Nein	Nein	Nein	Nein	Nein	Ja
Program.-Sprache	-	C++	Java	C#	C#	C++ (X-CMG)	C++
FPGA Datenbank	Xilinx	Torc	RapidSmith	ReCoBus	GoAhead	-	DHHarMa

3.6 Zusammenfassung

In diesem Kapitel wurde ein Überblick über die dynamisch partielle Rekonfiguration von FPGAs gegeben. Dabei erfolgte zunächst eine Einführung in die Thematik, in der die möglichen Vor- und Nachteile bei der Verwendung von DPR aufgeführt und anschließend durch Anwendungsbeispiele analysiert wurden. Hierbei wurden die Kosten (Produktionskosteneinsparung, Mehrkosten durch DPR-Infrastruktur), die Leistungsaufnahme, die System-Flexibilität und die Erhöhung der Fehlertoleranz erörtert.

Darauffolgend wurden die aktuellen FPGA-Familien des FPGA-Herstellers Xilinx vorgestellt und auf deren Entwicklung eingegangen. Hierbei wurden insbesondere die Änderungen in der Slice-Architektur und die verfügbaren FPGA-Ressourcen beleuchtet. Anschließend wurde auf die Konfiguration von Xilinx FPGAs eingegangen. Dabei wurden der Aufbau des Konfigurationsspeichers, die unterstützten Konfigurationsschnittstellen und der allgemeine Konfigurationsablauf vorgestellt. Mit diesem Wissen konnten die Unterschiede zwischen einer vollständigen und einer dynamisch partiellen Rekonfiguration sowie die Besonderheiten der DPR von Xilinx FPGAs herausgestellt werden. Unterschiede in den Xilinx FPGA-Familien wurden ebenfalls aufgezeigt.

Die beiden vorherigen Abschnitte bilden die Grundlage für die Entwicklung eines eigenen Entwurfsablaufs und eigener Softwarekomponenten, um den Entwurf von flexiblen DPR-Systemen zu ermöglichen. Ein zentraler Punkt in einem DPR-System ist die Kommunikation zwischen dem statischen Bereich und den dynamischen Bereichen. In den vergangenen Jahren wurden unterschiedliche Kommunikationsinfrastrukturtypen entwickelt, welche sich in zwei Kategorien, Link Makros und Eingebettete Makros, einordnen lassen.

Abschließend wurden die Entwurfsabläufe von Xilinx und Entwurfsabläufe aus dem akademischen Bereich vorgestellt. Um eine feingranulare Gegenüberstellung der daraus resultierenden DPR-Systeme zu ermöglichen, wurden vier verschiedene Rekonfigurationsarten (Single Island, Multi Island, 1D-Slot und 2D-Grid) vorgestellt. Der Vergleich zeigte, dass ein Xilinx DPR-System die vorhandene Hardware-Flexibilität nicht ausnutzt und dass der Speicher- und Wartungsaufwand eines Xilinx DPR-Systems bereits bei kleinen DPR-Systemen hoch ist. Dies liegt insbesondere an der gewählten Kommunikationsinfrastruktur, welche in dynamischen Bereichen unterschiedlich verdrahtet sein kann. Hierdurch ergibt sich eine feste Position für die dynamischen Komponenten. Überdies führen Änderungen am statischen Design in der Regel auch zu Änderungen an der Platzierung und Verdrahtung der Kommunikationsinfrastruktur, sodass die dynamischen Komponenten ebenfalls neu implementiert werden müssen. Die Aufhebung dieser deutlichen Einschränkung war das primäre Ziel der vorgestellten Projekte aus dem akademischen Bereich. Der Vergleich zeigte dabei, dass insbesondere der im Rahmen dieser Dissertation entstandene Entwurfsablauf von INDRA 2.0 den Entwurf von flexiblen und wartbaren DPR-Systemen ermöglicht, ohne die Xilinx Werkzeuge einzuschränken. Die wesentliche Voraussetzung für die Unterstützung der übrigen vorgestellten Rekonfigurationsarten ist eine homogene DPR-Kommunikationsinfrastruktur.

4 DHHarMa: Automatisierte Erzeugung homogener Strukturen für FPGAs

DHHarMa (*Design Flow for Homogeneous Hard Macros*) ermöglicht die Transformation eines anfänglich inhomogenen Designs in ein homogenes Design. Die bereits im vorherigen Kapitel aufgeführten DPR-Kommunikationsinfrastrukturen stellen hierbei die Hauptanwendung dar und werden zur Realisierung eines flexiblen, feingranularen (INDRA 2.0) DPR-Systems benötigt. Der Begriff *Hard-Makro* basiert auf dem Xilinx-Dateityp (NMC) und beschreibt eine bereits platzierte und verdrahtete Komponente, welche direkt in ein Hardware-Design integriert werden kann. Um ein *homogenes* Hard-Makro erzeugen zu können, muss eine homogene Verwendung der internen FPGA-Ressourcen, konkret der (CLB-) internen Komponenten, eine homogene Platzierung sowie Verdrahtung erfolgen. Die Software-Werkzeuge von Xilinx lassen sich bezüglich der Homogenität nicht parametrieren, sodass die Entwicklung eigener Softwarekomponenten notwendig wird.

Das Kapitel beginnt mit einer kurzen Beschreibung von typischen Anwendungsgebieten, in denen Homogenität für die Designimplementierung benötigt wird. Anschließend wird der allgemeine Entwurfsablauf von DHHarMa beschrieben. Eine wesentliche Funktion erfüllt hierbei die *Xilinx Design Language* (XDL), eine von Xilinx eingeführte Beschreibungssprache, welche im Anschluss vorgestellt wird. Darüber hinaus wird die entstandene generische, hierarchische und daher wartbare Bibliothek für DPR-Kommunikationsinfrastrukturen vorgestellt. In dem darauffolgenden Abschnitt erfolgt eine Auswertung der vorgestellten DHHarMa-Softwarekomponenten. Zur Auswertung aller DHHarMa Werkzeuge, erfolgt eine Gegenüberstellung von homogenen, durch DHHarMa erzeugte, Implementierungen mit inhomogenen, durch Xilinx Werkzeuge erzeugte, Implementierungen für unterschiedliche FPGA-Familien.

4.1 Anwendungsgebiete mit homogenen Designanforderungen

In diesem Abschnitt werden zwei Anwendungsgebiete vorgestellt, in denen gleichartige Strukturen für die Designimplementierung benötigt werden: Neben DPR-Kommunikationsinfrastrukturen werden diese in sogenannten *Time-to-Digital-Converters* (TDCs) verwendet.

4.1.1 DPR-Kommunikationsinfrastrukturen

Das erste Anwendungsgebiet ist in dem Bereich von DPR-Systemen zu finden. Wie in Abschnitt 3.5 erörtert wurde, muss für die Erzeugung eines flexiblen DPR-Systems mit Unterstützung aller dort aufgeführten Rekonfigurationsarten die Kommunikation zwischen den PR-Modulen sowie zum statischen Bereich über homogene Kommunikationsinfrastrukturen erfolgen. Die Erstellung dieser homogenen Kommunikationsinfrastrukturen ist die zentrale Aufgabe von DHHarMa. Details zu den verschiedenen Arten der DPR-Kommunikationsinfrastrukturen wurden bereits in Abschnitt 3.4 aufgeführt. In diesem Kapitel ist die DPR-Kommunikationsinfrastruktur, aus den sich ergebenden Vorteilen dieser Implementierungsart, stets als eingebettetes Makro realisiert. In Abschnitt 4.4 werden die einzelnen Komponenten der Kommunikationsinfrastruktur sowie die erwähnte Bibliothek vorgestellt. In Abschnitt 4.6.3 folgen konkrete Implementierungen für ein- und zweidimensionale Kommunikationsinfrastrukturen für unterschiedliche FPGA-Familien.

4.1.2 Time-to-Digital-Converter

Eine zweite Anwendung bilden TDCs, in denen homogene Verzögerungsleitungen (engl. delay lines) eingesetzt werden, um Laufzeitunterschiede zwischen Signalen mit einer hohen Auflösung im Bereich von Piko- bis Femtosekunden messen zu können [189]. Weiterhin können TDCs aufgrund der Implementierungsart in zwei Untergruppen aufgeteilt werden: *Single-Hit-TDCs* und *Multi-Hit-TDCs*.

Single-Hit-TDC Ein Single-Hit-TDC hat nach einer Messung eine, im Vergleich zum gemessenen Intervall, relativ hohe Totzeit. In einigen Anwendungen, wie z. B. in der Triggerlogik eines Oszilloskops, kann die Zeitmessung jedoch indirekt erfolgen und somit eine Auflösung von ca. 100 Femtosekunden erreicht werden.

Multi-Hit-TDCs Sollen mehrere Ereignisse dicht hintereinander gemessen werden, werden Multi-Hit-TDCs mit selbstkalibrierenden Gatterverzögerungsketten verwendet, wodurch Auflösungen bis zu etwa 10 Pikosekunden erricht werden. Im Vergleich zum Single-Hit-TDC, gibt ein Multi-Hit-TDC nicht die Größe eines einzelnen Zeitintervalls, sondern mehrerer Puls-Ankunftszeiten aus. Einsatzgebiete für Multi-Hit-TDCs sind in einem Massenspektrometer oder in der experimentellen Grundlagenforschung, insbesondere der Atomphysik, zu finden.

TDC-Implementierungen

In [159] erfolgt eine Gegenüberstellung von TDC-Implementierung in ASICs und FPGAs. Song et al. resümieren, dass in Anbetracht der Kosten, Entwicklungszeit und Flexibilität eine Implementierung auf einem FPGA bei geringen Stückzahlen zu bevorzugen ist. Zur Implementierung der Verzögerungsleitung auf einem FPGA können die dedizierten Leitungen einer Carry Chain benutzt werden. Diese Leitungen werden allgemein zur

Implementierung für arithmetische Funktionen (Addierer, Zähler oder Vergleichsschaltungen) benutzt, sodass die Verzögerung auf diesen Leitungen zum einen sehr kurz ist und zum anderen für eine FPGA-Technologie (FPGA-Familie und Fertigungsgröße), bei gleichbleibender Spannung und Temperatur, als konstant angesehen werden kann. In Abbildung 4.1 wird die TDC-Implementierung auf einem FPGA durch die Verwendung einer Carry Chain dargestellt. Die linke Abbildung zeigt das vereinfachte logische Blockdiagramm, die mittlere Abbildung die Realisierung auf einem Virtex-5 FPGA und die rechte Abbildung die interne Verschaltung der Slices (mit den Eingängen CIN und COUT) dar. Y

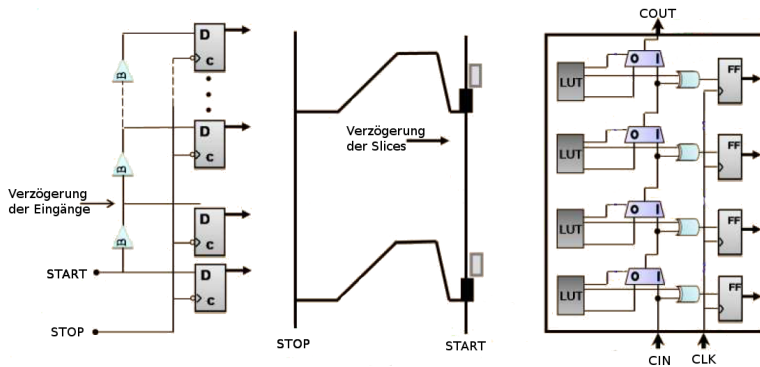


Abbildung 4.1: FPGA-basierte TDC-Implementierung durch Verwendung einer Carry Chain (modifizierte Darstellung aus [40])

In [62] wird eine FPGA-basierte Implementierung mit einer Auflösung von bis zu 17 Pikosekunden beschrieben. In [191] werden weitere Techniken vorgestellt, die eine Implementierung auf eine Genauigkeit von 10 Pikosekunden bringen. Alle genannten Quellen teilen den Umstand der manuellen Erzeugung der benötigten Verzögerungsleitung. Eine Automatisierung des Entwurfs ist wünschenswert und wird mit der DHHarMa-Software ermöglicht. Konkrete Implementierungen werden in Abschnitt 4.6.4 vorgestellt.

4.2 Entwurfsablauf

Der Entwurfsablauf von DHHarMa lässt sich in zwei Abschnitte, dem sogenannten *Xilinx-basierten Frontend* und dem *DHHarMa Backend*, unterteilen und ist in Abbildung 4.2 dargestellt.

Das Frontend überführt ein Design in Form von HDL-Dateien in eine Xilinx spezifische Design-Datei für ein konkretes FPGA. Die Ausgangsdatei des Frontends dient dem Backend als Eingangsdatei. Die Aufgabe des DHHarMa Backends ist die Homo-

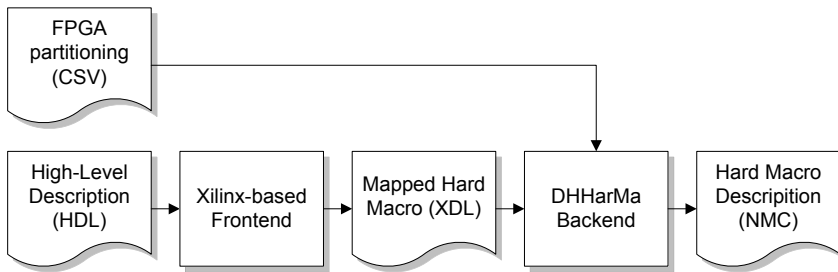


Abbildung 4.2: Entwurfsablauf von DHHarMa zur Erzeugung von homogenen Hard-Makros [105, S. 2]

genisierung des initial inhomogenen Designs. Hierzu erfolgt eine homogene Packung, Platzierung und Verdrahtung. Der Entwurfsablauf wurde in [105] veröffentlicht, ist ausführlich in der Diplomarbeit [103, S. 21 ff.] beschrieben und wird an dieser Stelle zusammengefasst dargestellt.

4.2.1 Xilinx-basiertes Frontend

Das Frontend des DHHarMa Design Flows verwendet vier Werkzeuge von Xilinx, um die Abbildung eines Designs auf einen spezifischen FPGA durchzuführen. Abbildung 4.3 stellt das Frontend mit den einzelnen Werkzeugen dar. Diese Werkzeuge und deren Aufgabe werden im Folgenden beschrieben, um die benötigten Einzelschritte zu verdeutlichen.

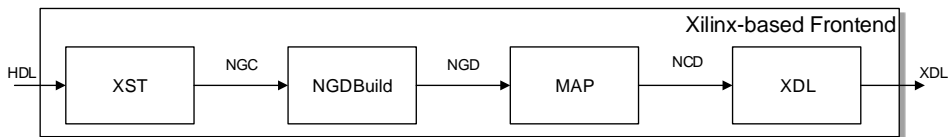


Abbildung 4.3: Xilinx-basiertes Frontend von DHHarMa zur Erzeugung von homogenen Hard-Makros [103, S. 22]

Xilinx Synthesis Technology (XST) Das erste Werkzeug von Xilinx, *XST* (Xilinx Synthesis Technology, [200, S. 81 ff.]), transformiert die HDL-Datei in eine architektur-spezifische Netzliste (engl. netlist) im Format *NGC* (Native Generic Circuit). Die Synthese analysiert die HDL-Eingangsdateien und übersetzt diese in designspezifische Blöcke, wie zum Beispiel LUT, RAM oder Multiplexer (MUX). Zusätzliche Randbedingungen (engl. constraints) bezüglich Takt oder Implementierung können in Form von einer *XCF* (Xilinx Constraint File) oder *UCF* (User Constraint File) berücksichtigt und der

NGC-Datei hinzugefügt werden. Die NGC-Datei kann weiterhin noch unbeschriebene Blöcke (Black-Box-Implementierungen, siehe Abschnitt 3.3.4) enthalten. Eine ausführliche Dokumentation der Software XST für die Virtex-5-Familie und ältere FPGA-Familien ist in [252] und für die Virtex-6-, Spartan-6 und die 7-Serie-Familie in [251] gegeben.

Translate (NGDBuild) Die Netzliste wird von dem Werkzeug *NGDBuild* [200, S. 71 ff.] eingelesen und in eine logische Datenbankdatei im Format NGD (Native Generic Database) übersetzt. Dieser Schritt wird daher auch Übersetzung (engl. translate) genannt. Eine NGD-Datei beschreibt das Design in Basis-Elementen wie Logikgatter (AND, OR usw.), Flip-Flops und RAM-Blöcken. Das Design kann aus mehreren Netzlisten im NGC- oder EDIF-Format sowie UCFs bestehen, welche alle zu einer NGD-Datei zusammengeführt werden.

(Technology) Mapping (MAP) Das dritte Werkzeug, *MAP* [200, S. 81 ff.], führt die Technologieabbildung (engl. technology mapping) der logischen Design-Datei aus. Dabei wird die Datenbank im NGD-Format auf FPGA-spezifische Blöcke, wie CLBs, BRAMs oder IOBs abgebildet. Die Ausgangsdatei beschreibt das durch eine HDL beschriebene Design in einer für eine FPGA-Familie spezifischen Schaltungsspezifikation im NCD-Format (Native Circuit Description). Ab der Virtex-5-Familie wird zusätzlich eine Platzierung des Designs durchgeführt. Für frühere FPGA-Familien kann diese durch Modifizierung der Standardeinstellungen forciert werden.

XDL Mit Hilfe des Werkzeugs *xdl* von Xilinx kann die proprietäre Schaltungsspezifikation im NCD-Format in eine textuelle Beschreibung des Designs umgewandelt werden. Diese Datei bildet die Schnittstelle zwischen dem Frontend und dem Backend und enthält alle FPGA-spezifischen Blöcke in einer von Xilinx definierten Beschreibungssprache (engl. Xilinx Design Language, XDL). Eine Erläuterung zu dieser Sprache folgt in Abschnitt 4.3.

4.2.2 DHHarMa Backend

Die Ausgangsdatei des DHHarMa Frontends dient dem, in Abbildung 4.4 dargestellten, DHHarMa Backend als Eingangsdatei. Die einzelnen benötigten Komponenten zur Homogenisierung der inhomogenen Design-Datei werden an dieser Stelle kurz zusammengefasst. Eine detailliertere Vorstellung der Hauptkomponenten von DHHarMa folgt in Abschnitt 4.5.

FPGA-Datenbank Die Werkzeuge des DHHarMa Backends (Packer, Placer und Router) benötigen Informationen über die verfügbaren FPGA-Ressourcen sowie deren interne Verdrahtung. Diese Information wird durch zuvor ebenfalls mit DHHarMa erstellte Datenbanken bereitgestellt. Details zu den Datenbanken folgen in Abschnitt 4.5.1.

Parser In einem ersten Schritt wird das inhomogene Design in der XDL-Beschreibung durch einen *XDL Parser* in eine interne Datenstruktur eingelesen. Zusätzlich wird die Partitionierung des FPGAs durch den *FPGA Partitioning Parser* importiert. Die Partitio-

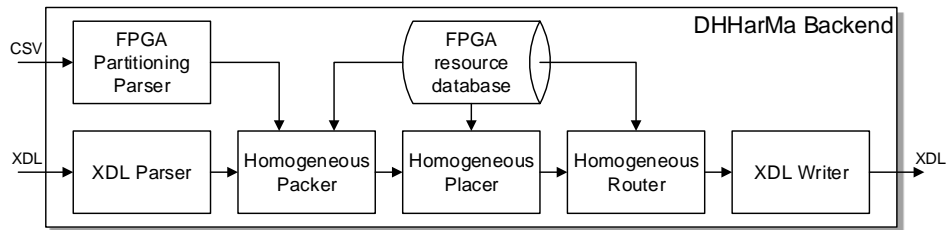


Abbildung 4.4: DHHarMa-Backend zur Erzeugung von homogenen Hard-Makros [105, S. 3]

nierungsdatei folgt dem CSV-Format (engl. **Comma-Separated Values**, durch Komma getrennte Werte), trägt die Dateiendung FPF (engl. **FPGA Partitioning File**) und definiert Regionen durch zwei zweidimensionale Koordinatenpaare. Eine Region hat somit stets eine rechteckige Form. Die Partitionierungsdatei einer Kommunikationsinfrastruktur eines DPR-Systems basiert auf der UCF des Layouts. Details zu den Parsern werden in Abschnitt 4.5.2 beschrieben.

Homogenisierung Nachdem die Eingangsdateien eingelesen sind, führt der *Packer* Transformations- und Packungsschritte innerhalb der CLBs durch, um identische Slice-Konfigurationen in, als homogen definierten, Regionen zu realisieren. Die genaue Vorgehensweise des Packers folgt in Abschnitt 4.5.3. Der *Placer* platziert die nun identischen Slices einer jeden homogenen Region in gleicher Art und Weise (siehe Abschnitt 4.5.4). Die Platzierung wird für jeden Regionstypen einmalig durchgeführt und anschließend in allen Regionen des gleichen Typs repliziert. Nachdem alle Instanzen des Designs homogen gepackt und platziert sind, werden die Netze durch den *Router* homogen verdrahtet. Details hierzu werden in Abschnitt 4.5.5 vorgestellt.

XDL-Dateigenerierung Der letzte Schritt des DHHarMa-Backends ist die Generierung einer XDL-konformen Textdatei. Die nun homogene Design-Beschreibung in der internen Datenstruktur der DHHarMa Softwarekette wird durch den *XDL Writer* in eine XDL-konforme Ausgangsdatei geschrieben, mit Hilfe des gleichnamigen Xilinx Werkzeugs *xdl* in ein Hard-Makro (NMC-Format) transformiert und ist somit für eine weitere Verwendung in einem Design verfügbar. Alternativ kann der XDL Writer die XDL-Ausgangsdatei entsprechend einer NCD-Datei erstellen (siehe Abschnitt 4.5.6).

4.3 Xilinx Design Language

Die Xilinx Design Language (XDL) benutzt eine, von Xilinx eingeführte Syntax und Semantik, um zum einen ein *Design* und zum anderen die interne Verdrahtung und die internen Ressourcen eines Xilinx FPGAs in Form eines *Reports* zu beschreiben.

Die Sprache wird daher in diesen zwei unterschiedlichen Dateitypen benutzt, wobei einzelne Schlüsselwörter in beiden Dateitypen verwendet werden. XDL wird nur von den Werkzeugen der Xilinx ISE verwendet und von Xilinx lediglich rudimentär, in Form einer HTML-Dokumentation, beschrieben [221]. Nähere Details zu der Sprache sind in den laufenden Jahren durch wissenschaftliche Projekte gesammelt worden. Eine Übersicht über die Sprache erfolgt z. B. in [21]. Da der Support für diese Softwareumgebung mit der Version 14.7 eingestellt wurde, werden FPGAs bis zu der 7-Serie unterstützt.

4.3.1 XDL-Report

Ein XDL-Report besitzt die Dateiendung *xdlrc* und beschreibt textuell den Aufbau eines Xilinx FPGAs. Die Informationen dieses XDL-Reports werden für die Erzeugung einer effizienten Datenstruktur eines Xilinx FPGAs benötigt, welche ein essentieller Bestandteil der DHHarMa Werkzeugkette ist. Details zu dem XDL-Report werden z. B. in [21, 102, 103] beschrieben. Im folgenden Abschnitt wird eine aktualisierte Zusammenfassung und gleichzeitig eine Terminologie für dieses Kapitel gegeben.

Ein Report beschreibt den gesamten FPGA durch sogenannte *tiles* (dt. Kachel), welche in einer zweidimensionalen Matrix angeordnet sind sowie sogenannte *primitive_defs*, welche Informationen über den internen Aufbau der einzelnen Kacheltypen enthalten. Auf die zuletzt genannten *primitive_defs* wird nicht weiter eingegangen, da diese für die Erstellung der FPGA-Datenstruktur irrelevant sind. Die *tiles* des Reports sind nicht mit den PR-Tiles eines INDRA (2.0) DPR-Systems in Abschnitt 3.5.6 zu verwechseln.

Eine Kachel beschreibt einen konkreten Block eines Basisblocks (siehe Abschnitt 2.2) in einem FPGA. Für jeden Basisblocktyp existiert ein äquivalenter Kacheltyp. So ist einem CLB der gleichnamige Kacheltyp, der Switch Matrix z. B. dem Kacheltyp *INT* (engl. interconnection matrix) zugeordnet. Eine Kachel ist in drei Abschnitte aufgeteilt, welche fünf Komponententypen enthält: *Primitive Site*, *Pinwire*, *Wire*, *Connection* und *PIP* (Programmable Interconnection Point). Die Abbildung 4.5 stellt Ausschnitte des XDL-Reports für den größten Virtex-7 FPGA (2000T) dar. Allgemein ist die Klammerung zur Abgrenzung der einzelnen Abschnitte erkennbar. Die Abbildung 4.6 zeigt die Komponenten des XDL-Reports aus Sicht des Xilinx FPGA Editors.

Tile Dem Schlüsselwort *tile* folgen zunächst zwei Zahlen, welche die Anordnung innerhalb der zweidimensionalen FPGA-Matrix wiedergeben. Die erste Zahl beschreibt die Zeile, die zweite Zahl die Spalte. Beide Zahlen bilden die sogenannte *Tile-Grid-Koordinate*. Die nachfolgende Zeichenfolge beschreibt einen eindeutigen Kachelnamen (*CLBLM_L_X134Y553*) und enthält in der Regel den Namen des Kacheltyps. Zusätzlich zu den Tile-Grid-Koordinaten enthält der Name *XY-Koordinaten*, welche für jeden Kacheltyp ein separates Koordinatensystem aufbaut. Anschließend an den Kachelnamen wird der Kacheltyp aufgeführt (*CLBLM*). Die letzte Zahl in der Tile-Definition gibt die Anzahl der folgenden Primitive Sites an.

```

(xdl_resource_report v0.2 xc7v2000tflg1925-2 virtex7
# *****
# * Tile Resources
# *****
(tiles 628 677
  (tile 0 0 NULL_X0Y627 NULL 0
    (tile_summary NULL_X0Y627 NULL 0 0 0)
  )
  ... weitere tiles
  (tile 48 309 CLBML_L_X134Y553 CLBML_M 2
    (primitive_site SLICE_X243Y553 SLICEM internal 45
      (pinwire A1 input CLBML_L_A1)
      (pinwire A output CLBML_L_A)
      ... weitere pinwires
    )
    ... weitere primitive_site
    (wire CLBML_M_A 0)
    (wire CLBML_WRL1END1 4 # Wire#1
      (conn INT_R_X133Y553 WRL1END1)
      (conn INT_L_X134Y553 WRL1BEG1) # Conn#1
      ... weitere conns
    )
    ... weitere wires
    (pip CLBML_L_X134Y553 CLBML_IMUX6 -> CLBML_M_A1) # PIP#1
    (pip CLBML_L_X134Y553 CLBML_M_A -> CLBML_LOGIC_OUTS12) # PIP#2
    ... weitere pips
    (tile_summary CLBML_L_X134Y553 CLBML_L 95 314 151)
  )
  (tile 48 310 INT_L_X134Y553 INT_L_SLV 1
    (primitive_site TIEOFF_X139Y553 TIEOFF internal 2
      (pinwire HARD0 output GND_WIRE)
      (pinwire HARD1 output VCC_WIRE)
    )
    (wire WRL1BEG1 4 # Wire#2
      (conn INT_R_X133Y553 WRL1END1)
      (conn CLBML_L_X134Y553 CLBML_WRL1END1) # Conn#2
      ... weitere conns
    )
    ... weitere wires
    (pip INT_R_X135Y553 IMUX_L7 -> SW2END3) # PIP#3
    (pip INT_R_X135Y553 LOGIC_OUTS_L12 -> WRL1BEG1) # PIP#4
    ... weitere pips
    (tile_summary INT_R_X135Y553 INT_R_SLV 2 588 3744)
  )
  ... weitere tiles
)
(primitive_defs 86
...
)
# *****
# * Summary
# *****
(summary tiles=425156 sites=490428 sitedefs=86 numpins=16543816 numpips=659152432)

```

Abbildung 4.5: Ausschnitte des XDL-Reports des Virtex-7 2000T FPGAs

Primitive Site Eine *Primitive Site* beschreibt Teilkomponenten einer Kachel und kapselt wiederum die Ein- und Ausgänge (*Pinwires*) dieser. Ein Virtex-7-CLB wird z. B. aus *zwei* Slices aufgebaut. Die Slices entsprechen daher atomaren Teilkomponenten einer CLB-Kachel. Der Primitive Site Name enthält ebenfalls ein XY-Koordinatenpaar (*SLICE_X243Y553*). Einzig in dem Fall, dass in dem Kacheltyp stets nur eine Primitive Site existiert, stimmen die Kachelnamenkoordinaten mit diesen XY-Koordinaten überein.

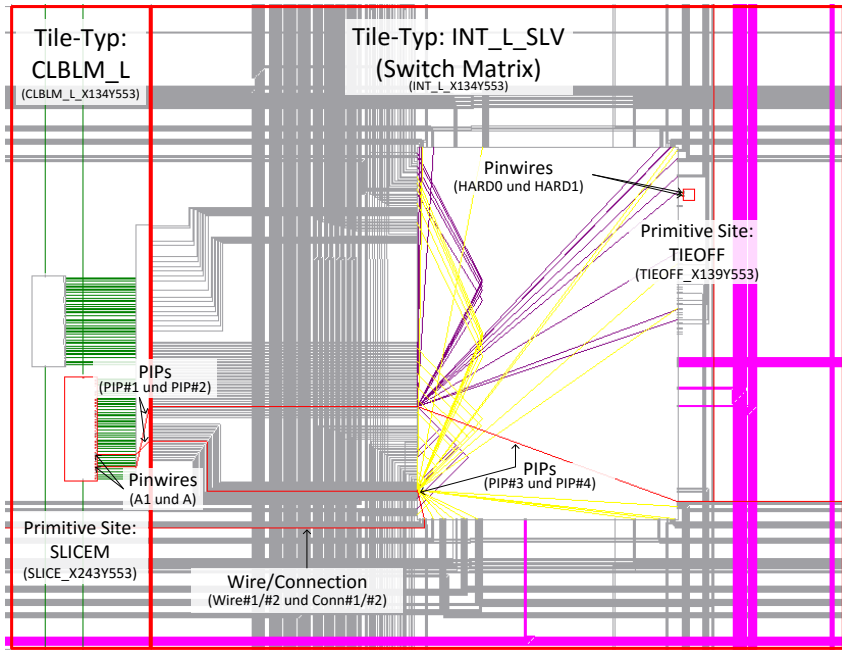


Abbildung 4.6: Visualisierung der Komponenten des XDL-Reports einer Virtex-7 Switch-Matrix mit benachbartem CLB

Insgesamt existieren daher drei Koordinatensysteme. Die Zahl am Ende weist auf die Anzahl der folgenden Pinwire-Definitionen hin.

Pinwire Die *Pinwires* definieren die Ein- und Ausgänge einer Primitive Site. Im Anschluss an das Schlüsselwort folgt der Pinname (z. B. *A1*). Die nachfolgende Zeichenfolge besagt, ob der Anschlusspin ein Eingang (engl. *input*) oder Ausgang (engl. *output*) ist. Die letzte Zeichenfolge bestimmt den Namen einer Leitung (engl. *wire*). Ein Pinwire wird daher textuell mit einer Leitung verbunden. Da die Leitungen erst in dem folgenden Abschnitt definiert werden, sind die Leitungsnamen an dieser Stelle vorgegriffen.

Wire Die Leitungsdefinition enthält den Namen der Leitung (z. B. *CLBLM_WR1END1*) und die Anzahl der Verbindungen (engl. *connection*) zu anderen Kacheln.

Connection Eine Leitungsverbindung, im Report mit dem Schlüsselwort *conn* abgekürzt, beschreibt physikalisch die gleiche Leitung aus Sicht einer anderen Kachel und enthält zwei Zeichenfolgen: Den Namen der anderen Kachel (z. B. *INT_L_X134Y553*) und den darin verwendeten Leitungsnamen (*WR1BEG1*). Alle Verbindungen beschrei-

ben daher zusammen eine *physikalische* Leitung.

In der Übersicht in Abbildung 4.5 und in der *FPGA Editor* Ansicht in Abbildung 4.6 wird die Relation zwischen *Wire* und *Connection* durch die Aufführung von zwei Leitungsdefinitionen (*Wire#1* und *Wire#2*) in zwei unterschiedlichen Kacheln für eine physikalische Leitung dargestellt. Wichtig ist, dass nicht alle aufgeführten Leitungsverbindungen eine physikalische Anschlussmöglichkeit bieten müssen. Ein Beispiel hierfür stellt die Leitungsdefinition *Wire#1* in Kachel CLBLM_L_X134Y553 dar. Leitungstypen, die physikalisch einzelne Kacheln überspringen, können je nach Detailstufe (siehe folgenden Abschnitt) des XDL-Reports auch in diesen Kacheln definiert sein.

PIP Ein sogenannter *Programmable Interconnection Point* (PIP) beschreibt eine programmierbare Verbindungsmöglichkeit zweier Leitungen. Insbesondere in den Switch Matrizen (*INT*) ist dieser Abschnitt sehr groß. Die zweite und vierte Zeichenfolge nach dem Schlüsselwort *pip* referenziert die beiden verschaltbaren Leitungen über ihre Leitungsnamen. Das dritte Symbol bestimmt die Richtung, in der ein Signal getrieben werden kann. Die Richtung kann je nach Leitungstyp, welcher in der Regel FPGA-Familien-spezifisch ist, uni- oder bidirektional sein.

Am Ende einer jeder Kachelinstanz folgt eine Zusammenfassung, in welcher der Kachelname, der Kacheltyp sowie die Anzahl der Pinwires, Wires und PIPs aufgeführt wird. Am Ende des Reports wird weiterhin die Anzahl der einzelnen Komponententypen (mit Ausnahme der Leitungen) festgehalten.

Detailstufen des XDL-Reports

Seit der Xilinx ISE Version 10.1 existieren drei verschiedene Detailstufen für den XDL-Report. Zuvor waren lediglich die ersten beiden Detailstufen verfügbar. Ein XDL-Report wird über den folgenden Kommandozeilenaufruf erstellt:

```
xdl -report [-pips] [-all_conns] <part> [<filename>]
```

Die aufgeführten (optionalen) Parameter veranlassen das Werkzeug *xdl* einen XDL-Report in einer bestimmten Detailstufe zu erstellen:

1. *-report*: Erstellt einen Report mit rudimentären FPGA-Informationen (Tiles und Primitive Sites)
2. *-pips*: Durch diesen additiven Parameter werden zusätzlich alle Ein- und Ausgänge der Primitive Sites, alle Verbindungen (Wire) zu *benachbarten* Kacheln sowie deren Verschaltungsmöglichkeiten (PIPs) aufgeführt
3. *-all_conns*: Erst durch diesen weiteren, additiven Parameter wird ein vollständiger Report mit *allen* Leitungsverbindungen erstellt

Der Report der ersten Detailstufe ist nur wenige MB groß und liefert eine grobe Übersicht über den Aufbau des FPGAs. Für den größten Virtex-7 FPGA (2000T) beträgt die Größe dieses Reports ca. 49 MB.

Die zweite Detailstufe beschreibt *annähernd* einen vollständigen FPGA. Jede Kachel enthält nun die Ein- und Ausgänge (Pinwire) der zugehörigen Teilkomponenten einer Kachel (Primitive Site), den Leitungsabschnitt mit den Leitungen (Wire) und Leitungsverbindungen (Connection) benachbarter Kacheln und den Abschnitt der Verschaltungsmöglichkeiten (PIP) dieser Leitungen. In diesem Report sind jedoch nur die Verbindungen zu *benachbarten* Kacheln in einer Leitungsdefinition aufgeführt. Dies führt dazu, dass einige, längere Leitungstypen über mehrere Kacheln verteilt definiert sind. Um eine vollständige Beschreibung für die physikalische Leitung eines solchen Leitungstyps zu erhalten, müssen alle benachbarten Kacheln, welche in den Verbindungen aufgeführt sind, durchlaufen und deren Informationen gesammelt werden. In einigen Fällen ist eine benachbarte Kachel eine leere Kachel ohne Komponenten (Typ *NULL* oder *Empty*). Die Verbindungsdefinition ist hierdurch unterbrochen und somit nicht vollständig. Da die Leitungen dennoch physikalisch existieren, wurden Verfahren entwickelt, um diese fehlerhaften Trennungen zu überbrücken [102, S. 36 ff.]. Die Größe dieses zweiten XDL-Report-Typs beträgt für den größten Virtex-7 FPGA ca. 44 GB.

Der dritte Report beschreibt einen *vollständigen* FPGA, mit allen Verbindungsinformationen. Jede Leitungsdefinition enthält alle Leitungsverbindungen, wodurch das Sammeln der Verbindungsinformationen entfällt. Die Report-Größe steigt auf ca. 77 GB für den Virtex-7 FPGA an. Die vollständigen Reports aller Virtex-7 FPGAs belegen ca. 1 097 GB an Speicherplatz.

4.3.2 XDL-Design-Datei

Eine XDL-Design-Datei beschreibt ein Design textuell. Das Design kann dabei teilweise oder vollständig verdrahtet und platziert sein. Eine XDL-Design-Datei ist in der Regel in vier, teilweise optionale, Abschnitte eingeteilt. Abbildung 4.7 visualisiert einen Ausschnitt einer XDL-Design-Datei eines Virtex-6 FPGAs. Die Einflüsse der DHHarMa Werkzeuge sind farblich gekennzeichnet: Packer (blau), Placer (grün) und Router (orange). Die in diesem Abschnitt beschriebenen Beispiele werden anhand eines Virtex-6 FPGAs erläutert, sind jedoch in der Regel auch auf andere FPGA-Familien übertragbar.

Properties Der erste Abschnitt enthält allgemein geltende Einstellungen (engl. *Properties*). Dieser Abschnitt wird als Konfigurationsabschnitt bezeichnet. In der ersten Zeile einer jeden XDL-Design-Datei wird zunächst der Name des Designs angegeben, folgend von dem vollständigen Namen des verwendeten FPGAs, inklusive Gehäuse (engl. *package*), Speedgrade und der verwendeten NCD-Versionsnummer.

Ports Der zweite Abschnitt ist nur in einer bestimmten Untermenge der XDL-Beschreibungen vorhanden, den *Hard-Makro-Dateien*. Dieser Abschnitt führt sogenannte *Ports* auf, welche Ein- und Ausgänge des Designs beschreiben. Die Ports dienen als Anschluss-

The image shows a snippet of an XDL-Design-File with four main sections: Properties, Ports, Instances, and Nets. Each section is highlighted with a different background color and has a corresponding label on the right side. The code is color-coded to match these sections. A legend on the right side of the Nets section identifies three tools: Packer (blue), Placer (yellow), and Router (orange).

```

design "bm_top" xc6vcx75tff484-2 v3.2 ,
#All Ports
port "CLK_int" "Base0_[11]_ (Slice_2/3)" "CLK";
...

#All instances (SLICE, IOB, DSP, BRAM ...)
inst "Rec0_a_[0]_ (Slice_1/2)" "SLICEL",placed CLBLM_X12Y118 SLICE_X19Y118 ,
cfg "
A5FFINIT::#OFF A5FFMUX::#OFF A5FFSR::#OFF A5LUT::#OFF A6LUT:Rec0_a/inst_g[0].base_element_1/
LUT2_4.A6LUT:#LUT:O6=(A5*A6) ACY0::#OFF AFF::#OFF AFFINIT::#OFF AFFMUX::#OFF AFFSR::#OFF
AOUTMUX::#OFF AUSED::0
B5FFINIT::#OFF B5FFMUX::#OFF B5FFSR::#OFF B5LUT::#OFF B6LUT:Rec0_a/inst_g[0].base_element_1/
LUT2_1.B6LUT:#LUT:O6=(A5*A6) BCY0::#OFF BFF:Rec0_a/inst_g[0].base_element_1/FD_2.BFF#FF
BFFINIT::INIT1 BFFMUX::O6 BFFSR::SRLOW BOUTMUX::#OFF BUSED::0
C5FFINIT::#OFF C5FFMUX::#OFF C5FFSR::#OFF C5LUT::#OFF C6LUT:Rec0_a/inst_g[0].base_element_1/
LUT2_2.C6LUT:#LUT:O6=(A5*A6) CCY0::#OFF CFF:Rec0_a/inst_g[0].base_element_1/FD_3.CFF:#FF
CFFINIT::INIT1 CFFMUX::O6 CFFSR::SRLOW COUTMUX::#OFF CUSED::0
D5FFINIT::#OFF D5FFMUX::#OFF D5FFSR::#OFF D5LUT::#OFF D6LUT:Rec0_a
inst_g[0].base_element_1/LUT2_3.D6LUT:#LUT:O6=(A5+A6) DCY0::#OFF DFF:Rec0_a/
inst_g[0].base_element_1/FD_1.DFF:#FF DFFINIT::INIT1 DFFMUX::O6 DFFSR::SRLOW DOUTMUX::#OFF
DUSED::#OFF
CLKINV::CLK COUTUSED::#OFF PRECINIT::#OFF SYNC_ATTR::ASYNC "
;
inst "Rec1_b_[0]_ (Slice_2/4)" "SLICEL",placed CLBLM_X12Y119 SLICE_X19Y119 ,
...

#All nets connecting the instances
net "static_right_out<7>" ,
outpin "Rec0_a_[0]_ (Slice_1/2)" BQ,
inpin "Base0_[0]_ (Slice_1/10)" AX,
inpin "Rec1_b_[0]_ (Slice_2/4)" B6,
inpin "Rec0_a_[0]_ (Slice_1/2)" DS,
pip CLBLM_X12Y118 CLBLM_L_BQ -> CLBLM_LOGIC_OUTS2 ,
pip INT_X12Y118 LOGIC_OUTS2 -> IMUX_B20 ,
pip CLBLM_X13Y117 CLBLM_IMUX_B20 -> CLBLM_L_B6 ,
...
;
net "static_right_out<8>" ,
...
    
```

Abbildung 4.7: Ausschnitt einer XDL-Design-Datei mit farblicher Kennzeichnung der Einflüsse der DHHarMa Werkzeuge [105, S. 2]

punkte für andere Designkomponenten, die ein solches Hard-Makro integrieren. Die erste Zeichenfolge definiert den Namen des Ports. Anschließend folgt der Name einer konkreten Instanz einer Komponente, welche im nächsten Abschnitt definiert und daher vorgegriffen wird. Abschließend folgt über den Namen des Ein- bzw. Ausgangs der Komponente eine eindeutige Port-Zuordnung.

Instances Der dritte Abschnitt listet alle Instanzen (engl. *instance*, abgekürzt *inst*) von Komponenten auf, die für die Realisierung des Designs benötigt werden. Eine Instanz entspricht dabei einer Primitive Site des XDL-Reports. Die Abbildung 4.7 stellt die Konfiguration eines Virtex-6 Slices eines CLB dar. Ein CLB wird durch zwei (ab Virtex-5) oder vier (bis Virtex-4) Slices beschrieben. Jede Slice wird in einer eigenen Instanz beschrieben und über textuelle (Konfigurations-) Schalter (engl. *switch*) konfiguriert. Die erste Zeile einer Instanz-Definition weist der Instanz einen eindeutigen Namen zu, gefolgt von dem Instanztyp. Die Informationen über die Platzierung folgen in den beiden, bereits im XDL-Report beschriebenen, zweidimensionalen XY-Koordinatensystemen, zunächst für den übergeordneten Block-Typ (*CLBLM*), dann für die eigentliche Instanz (*SLICE*). Anschließend erfolgt die Konfiguration der Instanz über die XDL-Konfigurationsschalter. Eine Slice besteht ab der Virtex-5-Familie wiederum aus vier sogenannten *Parts*. Ein Part enthält einen Funktionsgenerator (engl. *function generator*), mit zwei, je nach Slicetyp konfigurierbaren LUTs, einen Anschluss an die Carry Chain sowie zwei Register. Der erste Buchstabe eines Schalters legt dabei die Zuordnung zu dem Part (A-D) fest. Der unterschiedliche Slice-Aufbau aller unterstützter FPGA-Familien ist in Abschnitt 3.2 beschrieben.

Nets Der letzte Abschnitt enthält Verbindungen in Form von Netzen zwischen den zuvor definierten Instanzen. Ein Netz (engl. *net*) enthält einen eindeutigen Namen, gefolgt von der Auflistung der (Netz-) Anschlüsse (engl. *pin*), den Ein- und Ausgängen des Netzes. Ein Anschluss ist textuell an den Ein- oder Ausgang einer Instanz gebunden. Diese Anschlüsse stimmen mit den Pinwire der Primitive Site in dem XDL-Report überein. In der Regel ist der erste Pin eines Netzes ein Ausgangsanschluss (*outpin*). Die Anzahl der Eingangsanschlüsse ist von der Anzahl der Netzverzweigungen (engl. *fanout*) abhängig. Ein nicht verdrahtetes Netz beinhaltet nur die Anschlüsse, wohingegen ein verdrahtetes Netz zusätzlich die zu aktivierenden *PIPs*, für die Erstellung eines Pfads zwischen den Anschlüssen, enthält.

In der Abbildung 4.7 ist der Zusammenhang zwischen der Slice-Konfiguration und der Netz-Konfiguration anhand des Netzes *static_right_out<7>* dargestellt. Der Anschlusspin *BQ* beschreibt den Ausgang des Registers *Rec0_a/inst_g[0]_base_base_element_1/FD_2* im Part B der Slice *Rec0_a_[0]_(Slice_12)*, welche im dritten Abschnitt beschrieben ist. Der Ausgang des Registers ist durch eine Raute gekennzeichnet, welche im dritten und vierten Abschnitt abgebildet ist. Ein Eingangsanschluss des gleichen Netzes, *D5* (Kreis in der Abbildung), ist in der gleichen Slice lokalisiert und wird im Part D in der LUT *Rec0_a/inst_g[0]_base_base_element_1/LUT2_3* als Boolescher Operand *A5* verwendet.

4.4 HDL-Bibliothek für DPR-Kommunikationsinfrastrukturen

Durch die Verwendung einer HDL-Beschreibung als Eingangsdatei erhöht DHHarMa die Flexibilität des Entwurfs von homogenen Strukturen deutlich. Während in INDRA die Kommunikationsinfrastrukturen durch Konkatenierung händisch erstellter Busmakro-Primitive erfolgte (siehe Abschnitt 3.5.6), kann mit DHHarMa die Unterstützung einer Vielzahl von Kommunikationsprotokollen und Kommunikationsinfrastrukturen für unterschiedlich dimensionierte DPR-Systeme mit einer variablen Anzahl von PR-Regionen erfolgen. Die einzelnen Varianten der Kommunikationsinfrastruktur können durch eine generische, hierarchische und somit wartbare HDL-Bibliothek abgebildet werden. Die aktuelle, VHDL-basierte Bibliothek implementiert ein, an das Kommunikationsprotokoll *Wishbone* angelehntes Protokoll, welches im folgenden Abschnitt vorgestellt wird. Durch die Verwendung von VHDL *Records* können weitere Protokolle einer existierenden DPR-Kommunikationsinfrastruktur mit geringem Aufwand hinzugefügt werden. Die Kommunikationsinfrastrukturen sind aus Bibliothekskomponenten aufgebaut und in ein- oder zweidimensionale Varianten unterteilt. Aufgrund der ein- und zweidimensionalen Kommunikationsinfrastrukturen ergeben sich unterschiedliche Partitionierungen für DPR-Systeme. Exemplarisch werden für den eindimensionalen Fall drei und für den zweidimensionalen Fall vier Partitionierungen vorgestellt. Ein Wechsel des Kommunikationsprotokolls oder eine Anpassung der Datenbusbreite kann innerhalb der Top-Entity durch Zeilenmodifikationen erfolgen. Abschließend wird auf den Ressourcenbedarf der Kommunikationsinfrastrukturen für diese Partitionierungen eingegangen.

4.4.1 Kommunikationsprotokoll Encapsulated Wishbone Bus

Das Wishbone Busprotokoll ist ein on-chip Kommunikationsprotokoll, welches von der *Silicore Corporation* entworfen wurde und quelloffen ist. Eine direkte Verwendung als Kommunikationsprotokoll in einem DPR-System mit einer großen Anzahl von Teilnehmern (PR-Regionen) kann zu langen Verzögerungen führen. Diese Verzögerungen können dazu führen, dass der Bus nur mit einem geringen Takt betrieben werden kann. In [76, S. 69 ff.] wird daher der *Encapsulated Wishbone Bus* (EWB) vorgestellt, welcher eine Entkopplung des Busses und der Teilnehmer durch FIFOs und Registerstufen implementiert. Abbildung A.4 im Anhang B.4 stellt die Umsetzung des EWBs auf den typischen Wishbone dar.

Weiterhin werden zwei Protokollvarianten, Master-Slave (MEWB) und Slave (SEWB), unterstützt. Die Master-Slave-fähige Implementierung der Kommunikationsinfrastruktur ist 115 bit breit und setzt sich, wie in Tabelle 4.1 dokumentiert, aus geteilt verwendeten (s: shared) und dedizierten (d: dedicated) Eingangs- (I: Input) und Ausgangssignalen (O: Output) zusammen. Bei der Slave-fähigen Variante ergibt sich ein 95 bit breiter, asymmetrischer Bus. Der Datenbus ist hierbei der einzige Eingangs- und Ausgangsbus. Die konkrete Signalbelegung ist in Tabelle 4.2 aufgeführt.

Tabelle 4.1: Signalbelegung der Master Encapsulated Wishbone Bus (MEWB) Kommunikationsinfrastruktur

Bit	Signalname	Signaltyp	Signalbedeutung
31-0	EWB_DATA	sIO	Datenbus
63-32	EWB_ADR	sIO	Adressbus
67-64	EWB_SEL	sIO	Byte Enable
75-68	EWB_BLK_CNT	sIO	Block Counter
79-76	LED	sIO	LED
80	EWB_CYC	sIO	Cycle
81	EWB_WE	sIO	Write Enable
82	EWB_ACK	sIO	Acknowledge
86-83	EWB_GNT	dI	Grant
90-87	EWB_STB	dI	Strobe
94-91	RST_MOD	dI	Reset Module
98-95	TILE_EN	dI	Tile Enable
114-99	EWB_REQ	dO	Request

Tabelle 4.2: Signalbelegung der Slave Encapsulated Wishbone Bus (SEWB) Kommunikationsinfrastruktur

Bit	Signalname	Signaltyp	Signalbedeutung
31-0	EWB_DATA	sIO	Datenbus
32	EWB_ACK	sO	Acknowledge
36-33	LED	sO	LED
40-37	RST_MOD	dI	Reset Module
44-41	EWB_STB	dI	Strobe
48-45	TILE_EN	dI	Tile Enable
80-49	EWB_ADR	sI	Adressbus
88-81	EWB_BLK_CNT	sI	Block Counter
89	EWB_CYC	sI	Cycle
93-90	EWB_SEL	sI	Byte Enable
94	EWB_WE	sI	Write Enable

4.4.2 Ein- und zweidimensionale Kommunikationsinfrastrukturen

Die DHHarMa-Kommunikationsinfrastrukturen sind als eingebettetes Makro realisiert (siehe Abschnitt 3.4). Eine Übertragung der dort vorgestellten eindimensionalen Kommunikationsinfrastruktur auf eine zweidimensionale Kommunikationsinfrastruktur ist in der Abbildung 4.8 für gemeinsam verwendete Signale dargestellt. Da in dem zweidimensionalen Fall der Kommunikationsbus horizontal und vertikal verläuft, werden zusätzliche Ein- und Ausgänge benötigt, welche in der Abbildung kursiv gekennzeichnet sind. Da die LUTs stets mindestens vier Eingänge bereitstellen, ergibt sich für diese Signalart keine Mehrkosten bezüglich der FPGA-Ressourcen LUT und Register für die zweidimensionale Implementierung im Vergleich zum eindimensionalen Pendant (siehe Abbildung 3.17 auf Seite 60). Die zusätzlich benötigten Verbindungen sind in der Abbildung durch gestrichelte Linien dargestellt.

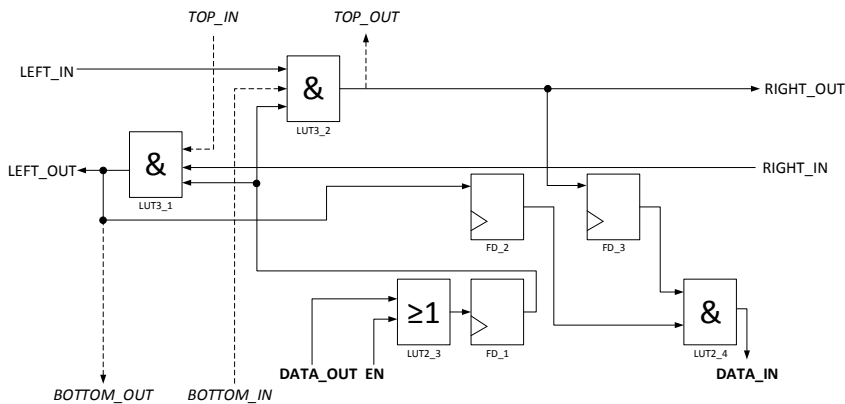


Abbildung 4.8: Architekturabbildung für ein bidirektionales, geteilt verwendetes Signal in einem eingebetteten Makro

Analog ergeben sich Änderungen für die dedizierten Signale. In Abbildung 4.9 ist die Implementierung für ein eindimensionales, dediziertes Signal dargestellt, welche der Implementierungen in den Abbildungen 3.18 entspricht. Die vier Register (*FDRE_1* - 4) kodieren bis zu 16 PR-Regionen eindeutig. Abbildung 4.10 stellt die Implementierung für eine zweidimensionale Kommunikationsinfrastruktur dar. Im Gegensatz zu der eindimensionalen Implementierung erfolgt vor der Adressdekodierung zunächst separat die Auswertung der 4-Bit breiten Eingangssignale der vier Datenrichtungen. Die Adressdekodierung wird anschließend durch zwei zusätzliche LUTs (*LUT4_5* und *LUT4_6*) mit jeweils vier Eingängen durchgeführt.

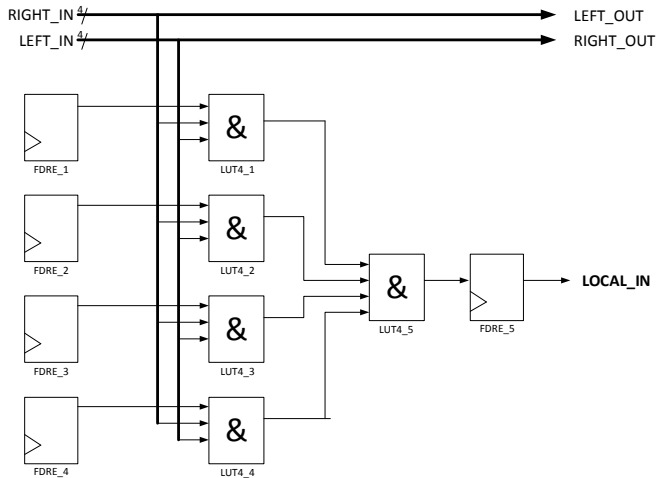


Abbildung 4.9: Architekturabbildung für ein eindimensionales, dediziertes Signal in einem eingebetteten Makro

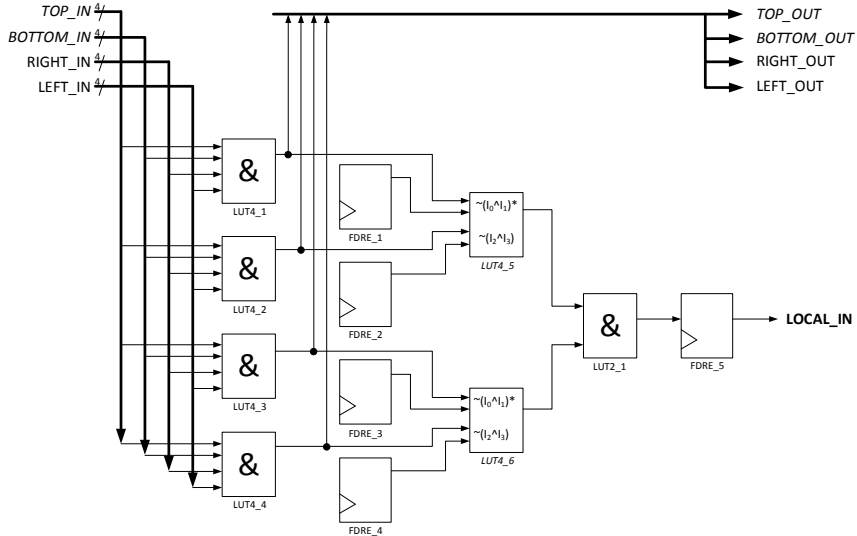


Abbildung 4.10: Architekturabbildung für ein zweidimensionales, dediziertes Signal in einem eingebetteten Makro

4.4.3 Komponenten der Bibliothek

Die Tabelle 4.3 führt die erstellten Bibliothekskomponenten für die unterschiedlichen DPR-Kommunikationsinfrastrukturen auf. Neben dem EWB-Kommunikationsprotokoll wurde ein reiner Datenbus (DB) in die Bibliothek aufgenommen. Wie in der ersten Spalte dargestellt, existieren insgesamt drei Hauptbibliotheken: *common*, *DB* und *EWB*. Die beiden letztgenannten Bibliotheken unterteilen sich in weitere Unterbibliotheken, welche in der zweiten Spalte aufgeführt sind: *DB* in *common* und *top* sowie *EWB* in *common*, *master*, *slave* und *top*. Die weiteren Spalten enthalten für jede VHDL-Komponente der Bibliothek den (Entity-) Namen, die FPGA-Ressourcen in LUTs und FFs sowie eine kurze Beschreibung. Die einzelnen Bibliothekskomponenten verwenden maximal vier LUT-Eingänge, sodass die dargestellten FPGA-Ressourcen unabhängig von einer konkreten FPGA-Familie sind.

Die aufgeführten *Basiselemente* repräsentieren atomare Bestandteile der Kommunikationsinfrastruktur. Die *common*-Bibliothekskomponente *primitiv_shared(_2d)* realisiert beispielsweise die Verschaltung eines gemeinsam verwendeten Signals, wie sie in Abbildung 4.8 dargestellt ist. Analog repräsentiert die Komponente *primitiv_dedi_bin* die Verschaltung eines dedizierten Signals (siehe Abbildung 4.9). Die zweidimensionale Variante *primitiv_dedi_bin_2d* entspricht der Abbildung 4.10. Durch die Verwendung von VHDL-Primitiven in Kombination mit Xilinx VHDL-Attributen werden Optimierungen innerhalb der PR-Regionen verhindert, sodass stets die gleichen FPGA-Ressourcen verwendet werden. Konkret werden die folgenden VHDL-Attribute verwendet:

- *S* (SAVE NET FLAG) [202, S. 245 ff.]: Verhindert das Entfernen von unverbundenen Signalen während des Packens, Platzierens und Verdrahtens.
- *KEEP* [202, S. 135 ff.]: Unterbindet die Absorption eines Netzes in einen Logikblock, welches zur Entfernung des Netzes führen würde.

Optimierungen in den statischen Regionen sind hingegen bezüglich der Homogenität unkritisch und werden daher nicht unterbunden. Die Komponenten *primitiv_base(_2d)* entsprechen daher den Komponenten *primitiv_shared(_2d)* ohne die aufgeführten VHDL-Attribute.

Die VHDL-Komponenten *tile_base(_2d)* und *tile_pr(_2d)* fassen mehrere Basiselemente zu einer vollständigen eindimensionalen oder zweidimensionalen Anschlussstelle in einer statischen bzw. PR-Region zusammen. Die Top-Entities in der Unterbibliothek *top* kombinieren mehrerer dieser Anschlussstellen, sowie zusätzliche (optionale) Terminierungskacheln (*tile_term*), zu einem vollständigen Kommunikationsbus. Die Terminierung bezieht sich hierbei auf eine *logische* Terminierung, welche den Ruhepegel des Busses bestimmt. Erwähnenswert ist, dass die FPGA-Ressourcen für eine zweidimensionale MEWB-Anschlussstelle mit jeweils acht LUTs nur geringfügig höher sind als die des eindimensionalen Pendantes. Aufgrund der Partitionierung des DPR-Systems können sich Optimierungen ergeben (siehe Abschnitt 4.4.5), die dazu führen, dass FPGA-Ressourcen eingespart werden können, sodass diese Werte Höchstwerte darstellen.

Tabelle 4.3: Komponenten der HDL-Bibliothek

Bibliothek	VHDL-Komponente	FPGA-Ressourcen		Beschreibung	
		LUTs	FFs		
common	ext_pin_tag	1	0	Anschlusspunkt der Kommunikationsinfrastruktur	
	primitiv_base	4	3	primitiv_shared ohne Attribute	
	primitiv_base_2d	4	3	primitiv_shared_2d ohne Attribute	
	primitiv_shared	4	3	1D Basiselement für gemeinsam verwendete Signale	
	primitiv_shared_2d	4	3	2D Basiselement für gemeinsam verwendete Signale	
	primitiv_term	0	2	Basiselement zur Terminierung	
DB	common				
	tile_base	32 128 256	24 96 192	1D Element der statischer Region	
	tile_base_2d	32 128 256	24 96 192	2D Element der statischer Region	
	tile_pr	32 128 256	24 96 192	1D Element der PR-Region	
	tile_pr_2d	32 128 256	24 96 192	2D Element der PR-Region	
top	tile_term	0	16 64 128	Element zur Terminierung	
	bm_top			Top-Entities für 1D und 2D Komm.	
EWB	common				
	bm_pkg			Package (Funktionen und Konstanten)	
	primitiv_dedi_bin	5	5	1D Basiselement für dedizierte Signale	
	primitiv_dedi_bin_2d	7	5	2D Basiselement für dedizierte Signale	
	primitiv_dedi_bin_tog	6	5	1D Basiselement für ded. Tile-Aktivierungssignal	
	primitiv_dedi_bin_tog_2d	8	5	1D Basiselement für ded. Tile-Aktivierungssignal	
	term_tile	0	190 230	Element zur Terminierung	
	master	ewb_master_bus_pkg			Busdefinition durch VHDL-Records
		primitiv_dedi_master	48	32	1D Basiselement für dediziertes Busanfragesignal
		primitiv_dedi_master_2d	48	32	2D Basiselement für dediziertes Busanfragesignal
		tile_base	460	345	1D Element der statischer Region
	tile_base_2d	460	345	2D Element der statischer Region	
	tile_base_ext_pins	115	0	Anschlusspunkte der statischen Region	
	tile_pr	401	301	1D Element der PR-Region	
	tile_pr_2d	409	301	2D Element der PR-Region	
	tile_pr_ext_pins	115	0	Anschlusspunkte der PR-Region	
	slave	ewb_slave_bus_pkg		Busdefinition durch VHDL-Records	
	tile_base	380	285	1D Element der statischer Region	
	tile_base_2d	380	285	2D Element der statischer Region	
	tile_base_ext_pins	95	0	Anschlusspunkte der statischen Region	
	tile_pr	210	172	1D Element der PR-Region	
	tile_pr_2d	308	218	2D Element der PR-Region	
	tile_pr_ext_pins	95	0	Anschlusspunkte der PR-Region	
	tile_term_asym	0	121	Element zur Terminierung	
top	bm_top			Top-Entities für 1D und 2D Komm.	

Die Zusammensetzung der FPGA-Ressourcen der Anschlussstellen wird exemplarisch für einen zweidimensionalen MEWB vorgestellt. Die Tabelle 4.4 und Tabelle 4.5 listen für jedes MEWB-Bussignal die verwendete VHDL-Komponente, die jeweilige Bitbreite und die daraus resultierenden FPGA-Ressourcen in LUTs und FFs auf. Die vorletzte Zeile enthält die kumulierte Gesamtanzahl, welche den Werten aus Tabelle 4.3 entsprechen. Der zuletzt aufgeführte Mittelwert $\bar{\varnothing}_{bit}$ normiert die FPGA-Ressourcen pro Bit. Die Zusammensetzungen der Anschlussstellen für die weiteren EWB-Kommunikationsinfrastrukturen sind in entsprechenden Tabellen im Anhang B.4 aufgeführt. Auf eine tabellarische Darstellung der DB-Kommunikationsinfrastrukturen wird aufgrund der simplen Zusammensetzung (Multiplikation der Bitbreite mit den FPGA-Ressourcen der VHDL-Komponenten $base_element(_base)(_2d)$) verzichtet.

Tabelle 4.4: Zusammensetzung der FPGA-Ressourcen für die 2D MEWB-Anschlussstelle in der statischen Region (*tile_base_2d*)

Bit	Signal Name	VHDL-Komponente	Breite in bit	FPGA-Ressourcen	
				LUTs	FFs
31-0	EWB_DATA	primitiv_base_2d	32	128	96
63-32	EWB_ADR	primitiv_base_2d	32	128	96
67-64	EWB_SEL	primitiv_base_2d	4	16	12
75-68	EWB_BLK_CNT	primitiv_base_2d	8	32	24
79-76	LED	primitiv_base_2d	4	16	12
80	EWB_CYC	primitiv_base_2d	1	4	3
81	EWB_WE	primitiv_base_2d	1	4	3
82	EWB_ACK	primitiv_base_2d	1	4	3
86-83	EWB_GNT	primitiv_base_2d	4	16	12
90-87	EWB_STB	primitiv_base_2d	4	16	12
94-91	RST_MOD	primitiv_base_2d	4	16	12
98-95	TILE_EN	primitiv_base_2d	4	16	12
114-99	EWB_REQ	primitiv_base_2d	16	64	48
Summe			115	460	345
\emptyset_{bit}				4,00	3,00

Tabelle 4.5: Zusammensetzung der FPGA-Ressourcen für die 2D MEWB-Anschlussstelle in der PR-Region (*tile_pr_2d*)

Bit	Signal Name	VHDL-Komponente	Breite in bit	FPGA-Ressourcen	
				LUTs	FFs
31-0	EWB_DATA	primitiv_shared_2d	32	128	96
63-32	EWB_ADR	primitiv_shared_2d	32	128	96
67-64	EWB_SEL	primitiv_shared_2d	4	16	12
75-68	EWB_BLK_CNT	primitiv_shared_2d	8	32	24
79-76	LED	primitiv_shared_2d	4	16	12
80	EWB_CYC	primitiv_shared_2d	1	4	3
81	EWB_WE	primitiv_shared_2d	1	4	3
82	EWB_ACK	primitiv_shared_2d	1	4	3
86-83	EWB_GNT	primitiv_dedi_bin_2d	4	7	5
90-87	EWB_STB	primitiv_dedi_bin_2d	4	7	5
94-91	RST_MOD	primitiv_dedi_bin_2d	4	7	5
98-95	TILE_EN	primitiv_dedi_bin_tog_2d	4	8	5
114-99	EWB_REQ	primitiv_dedi_master_2d	16	48	32
Summe			115	409	301
\emptyset_{bit}				3,56	2,62

4.4.4 DPR-Systempartitionierungen

Für die Erstellung von konkreten Kommunikationsinfrastrukturen wurden DPR-Systempartitionierungen erstellt, welche sich in ein- und zweidimensional unterteilen. Die logische Terminierung des Busses kann insbesondere in zweidimensionalen Partitionierungen zu einem limitierenden Faktor werden. Zum einen ergibt sich ein deutlicher Anstieg der zusätzlich benötigten FPGA-Ressourcen für die Kommunikationsinfrastruktur. Zum anderen werden für die Terminierung zweidimensionaler oder vertikaler, eindimensionaler Kommunikationsinfrastrukturen FPGA-Ressourcen benötigt, die nur durch eine ober- oder unterhalb liegende Taktregion bereitgestellt werden können. Insbesondere in neueren FPGA-Familien, in denen die Anzahl der CLBs in einer Taktregionsspalte steigt (siehe Abschnitt 3.3.1), limitiert dieser Umstand die Partitionierung eines DPR-Systems deutlich.

Untersuchungen haben gezeigt, dass unverbundene Verdrahtungsressourcen einen logischen Eins-Pegel treiben (siehe Abschnitt 6.4) und somit keinen Einfluss auf eine Und-basierte Kommunikationsinfrastruktur haben. Aus diesem Grund kann bei der Verwendung einer Und-basierten Kommunikationsinfrastruktur auf die logische Terminierung in den Partitionierungen prinzipiell verzichtet werden. Als Folge ergibt sich jedoch eine Inhomogenität in der Verdrahtung der Kommunikationsinfrastruktur. Diese Inhomogenität kann durch die Bestimmung einer Referenzregion mit einer vollständigen Kommunikationsanbindung ausgeglichen werden. Dabei muss durch die Partitionierung sichergestellt werden, dass für jeden PR-Regionstyp eine solche Referenzregion existiert. Ein PR-Modul wird anschließend stets für diese Referenzregion erstellt. In den folgenden Abbildungen ist die Referenzregion durch Textformatierungen hervorgehoben und die mögliche Terminierung durch gestrichelte Linien dargestellt.

Partitionierungen für eindimensionale Kommunikationsinfrastrukturen

Für die eindimensionalen Kommunikationsinfrastrukturen wurden drei Partitionierungsvarianten erstellt: *Links-Rechts*, *Links* und *Mitte*. Weiterhin werden bis zu vier unterschiedliche PR-Regionstypen (Rec0 - Rec3) in insgesamt maximal acht PR-Regionen verwendet. Die hier abgebildeten Partitionierungen resultieren in eine horizontal verlaufende Kommunikationsinfrastruktur. Generell sind jedoch auch eindimensionale, vertikal verlaufende Kommunikationsinfrastrukturen realisierbar. Eine Anwendung hierfür erfolgt in Abschnitt 6.2.4.

Partitionierung Links-Rechts Die Partitionierung *Links-Rechts* verbindet acht PR-Regionen: Jeweils vier links und vier rechts von dem statischen Bereich. Diese acht PR-Regionen werden durch einen statischen Bereich in der Mitte geteilt. Zwei der PR-Regionstypen existieren einmalig (Rec0 und Rec3), die beiden anderen Regionstypen mehrfach im DPR-System (Rec1 zweifach und Rec2 vierfach). Diese Partitionierung wurde bereits in Abschnitt 4.5.2 als Beispiel verwendet (siehe Abbildung 4.17). Die Abbildung 4.11 stellt die Referenzregionen dieser Partitionierung dar.

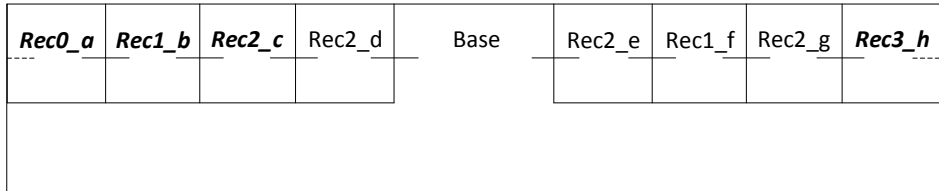


Abbildung 4.11: Partitionierung *LR* für 1D DPR-Kommunikationsinfrastrukturen

Partitionierung Links Die Partitionierung *Links* enthält vier PR-Regionen, welche links von dem statischen Bereich angeordnet sind. Wie in Abbildung 4.12 dargestellt, sind zwei PR-Regionen vom gleichen Typ (Rec2). Aufgrund der drei unterschiedlichen Regionstypen existieren drei Referenzregionen.

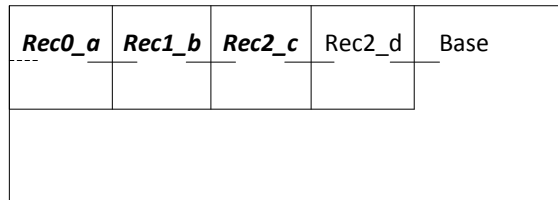


Abbildung 4.12: Partitionierung *L* für 1D DPR-Kommunikationsinfrastrukturen

Partitionierung Mitte Die Partitionierung *Mitte* verbindet zwei PR-Regionen vom gleichen Typ, welche links und rechts von dem statischen Bereich angeordnet sind. Die Abbildung 4.13 stellt diese Partitionierung dar. Um eine Referenzregion zu erhalten, muss für diese Partitionierung eine Buserminierung an einer der beiden Seiten erfolgen.

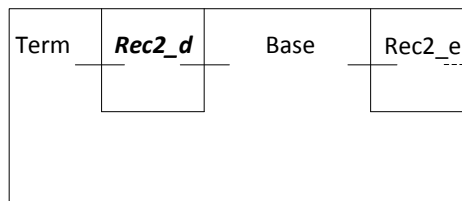
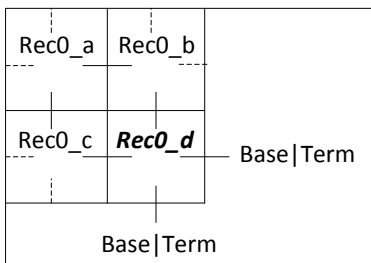


Abbildung 4.13: Partitionierung *M* für 1D DPR-Kommunikationsinfrastrukturen

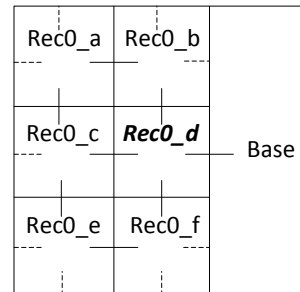
Partitionierungen für zweidimensionale Kommunikationsinfrastrukturen

Für die zweidimensionalen Kommunikationsinfrastrukturen wurden vier unterschiedliche Partitionierungen erstellt, wobei sich das Kürzel auf die Anzahl der Zeilen und Spalten bezieht: 2×2 , 3×2 , 3×3 und 5×2 . Anders als bei den eindimensionalen Partitionierungen wird lediglich ein PR-Regionstyp (*RecO*) in insgesamt maximal zehn PR-Regionen verwendet. Die Wahl eines einzigen PR-Regionstyps wurde primär getroffen, um die Anforderungen an die einzelnen DHHarMa Software-Werkzeuge zu maximieren. Die vorgestellten Partitionierungen stellen vier konkrete Partitionierungen dar, die zum Beispiel durch eine Spiegelung variiert werden könnten.

2x2 Die Partitionierung 2×2 besteht aus vier identischen PR-Regionen und ist, wie in Abbildung 4.14 (a) dargestellt, die einzige zweidimensionale Partitionierung, in der eine zusätzliche Buserminierung für die Erstellung einer Referenzregion hinzugefügt werden muss. Die Position der statischen Anschlussstelle und der Buserminierung kann beliebig gewählt werden.



(a) Partitionierung 2×2



(b) Partitionierung 3×2

Abbildung 4.14: Partitionierungen für 2D DPR-Kommunikationsinfrastrukturen

3x2 Die Partitionierung 3×2 besteht aus sechs identischen PR-Regionen und ist in Abbildung 4.14 (b) dargestellt. Durch die zusätzliche Zeile mit zwei PR-Regionen kann auf eine Buserminierung verzichtet werden, wenn die statische Anschlussstelle rechts neben *Rec0_d* positioniert wird.

3x3 Die Partitionierung 3×3 erweitert die Partitionierung 3×2 um eine weitere Spalte, sodass sich neun identische PR-Regionen ergeben. Wie in Abbildung 4.15 (a) dargestellt, liegt die Referenzregion in der Mitte der PR-Regionen. Die Position der statischen Anschlussstelle kann somit frei gewählt werden.

5x2 Die Partitionierung 5×2 erweitert die Partitionierung 3×2 um zwei weitere Zeilen und besteht daher aus zehn identischen PR-Regionen. Wie in Abbildung 4.15 (b)

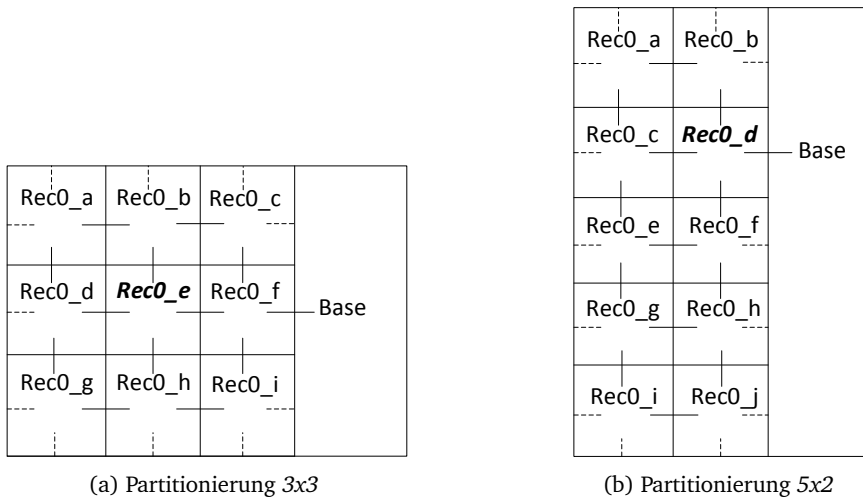


Abbildung 4.15: Weitere Partitionierungen für 2D DPR- Kommunikationsinfrastrukturen

dargestellt, liegt die Referenzregion neben der statische Anschlussstelle. Im Gegensatz zu der Partitionierung 3x2 kann diese wahlweise auch an die zwei darunterliegenden PR-Regionen (*Rec0_f* und *Rec0_h*) angeschlossen werden, wodurch sich die Referenzregion entsprechend ändert.

4.4.5 FPGA-Ressourcenbedarf

Neben der Partitionierung eines DPR-Systems kann der FPGA-Ressourcenbedarf der DPR-Kommunikationsinfrastruktur einen entscheidenden Einfluss auf das resultierende DPR-System haben. Da die DPR-Kommunikationsinfrastrukturen durch die vorgestellten Bibliothekskomponenten aufgebaut sind, beruht der in diesem Abschnitt vorgestellte FPGA-Ressourcenbedarf auf den FPGA-Basiselementen FF und LUT (maximal LUT4) und ist somit unabhängig von einer konkreten FPGA-Familie.

Der Ressourcenbedarf ist in der Tabelle 4.6 für die eindimensionalen und in Tabelle 4.7 für die zweidimensionalen DPR-Kommunikationsinfrastrukturen dargestellt. Die ersten vier Spalten in den Tabellen enthalten designspezifische Eigenschaften der Kommunikationsinfrastruktur. Konkret wird der Typ, die Konfiguration (aufgeteilt in Partitionierung und Busbreite) und die Anzahl der PR-Regionen aufgeführt. Die weiteren Spalten enthalten Informationen bezüglich der FPGA-Ressourcen. Neben der Gesamtanzahl wurden jeweils zwei Durchschnittswerte ermittelt:

- \emptyset_{Reg} : Durchschnittliche Anzahl der FPGA-Ressourcen pro Region

- \emptyset_{Reg_bit} : Durchschnittliche Anzahl der FPGA-Ressourcen pro Region und bit

Der Mittelwert \emptyset_{Reg} gibt Aufschluss über die benötigten FPGA-Ressourcen für einen Kommunikationsinfrastrukturtyp bei einer bestimmten Bitbreite. Die Partitionierung stellt durch die Anzahl der Regionen (inklusive Buserminierungen) einen weiteren Einfluss dar. Der Durchschnittswert \emptyset_{Reg_bit} ermöglicht einen von der Bitbreite unabhängigen Vergleich und resultiert in einen, für einen Kommunikationsinfrastrukturtyp und eine Partitionierung, konstanten Wert. Die Abbildung auf drei verschiedene FPGA-Architekturen mit einer Analyse der resultierenden Anzahl benötigter Slices erfolgt in Abschnitt 4.6.3. Bei den LUTs ergeben sich teilweise Optimierungen aufgrund der Partitionierung. Aus diesem Grund werden zwei Spalten für die Gesamtanzahl der LUTs dargestellt, die auf Basis der Bibliothek ermittelte Anzahl (*Orig.*) und die auf Basis von Optimierungen ermittelte Anzahl (*Opt.*).

Beide Tabellen stellen eine Zusammenfassung der FPGA-Ressourcen dar. Eine Aufschlüsselung der dargestellten, kumulierten Werte kann im Anhang B.4 für die fünf dargestellten Kommunikationstypen in einzelnen Tabellen nachgeschlagen werden. Die Auswertung des FPGA-Ressourcenbedarfs erfolgt für die ein- und zweidimensionalen Kommunikationsinfrastrukturen in separaten Abschnitten.

Eindimensionale Kommunikationsinfrastrukturen

Die Tabelle 4.6 stellt den FPGA-Ressourcenbedarf der eindimensionalen Kommunikationsinfrastrukturen inklusive aller optionalen Buserminierungen dar. In der Partitionierung *L* ergeben sich durch die einseitige Anbindung Optimierungen an der Anzahl der LUTs innerhalb der statischen Anschlussstelle. Da keine Signale von rechts verarbeitet werden müssen, kann pro Bit eine LUT eingespart werden (*LUT3_1* in Abbildung 4.8). Zusätzlich sind durch unidirektionalen Kontrollsignale in den SEWB-Kommunikationsinfrastrukturen weitere Optimierungen der FPGA-Ressourcen möglich.

Der Durchschnittswert \emptyset_{Reg} liegt für die Register bei einer Bitbreite von 8 bei etwa 23 FFs/Region und steigt auf maximal 297 FFs/Region für einen 115-Bit breiten Bus. Die Mittelwerte der LUTs liegen aufgrund der höheren Anzahl der LUTs innerhalb der verwendeten Bibliothekskomponenten über diesen Werten. Bei einer Bitbreite von 8 werden etwa 25 LUTs/Region, bei einer Bitbreite von 115 etwa 362 LUTs/Region benötigt. Designübergreifend variiert der Bitbreiten-unabhängige Mittelwert \emptyset_{Reg_bit} für die Register zwischen 1,76 FFs/Region/bit und 2,83 FFs/Region/bit sowie für die LUTs zwischen 1,17 FFs/Region/bit und 3,27 FFs/Region/bit. Mit steigender Anzahl an PR-Regionen nähern sich beide Mittelwerte entsprechend den jeweiligen Durchschnittswerten der Anschlussstelle *tile_pr* an. Der Vergleich der SEWB- und MEWB-Implementierungen zeigt, dass zwischen 62,1 % und 77,5 % mehr Register sowie 101,3 % und 126,2 % mehr LUTs für eine Master-fähige Kommunikationsinfrastruktur benötigt werden.

Tabelle 4.6: FPGA-Ressourcen der 1D DPR-Kommunikationsinfrastrukturen

Typ	Design			FPGA-Ressourcen						
	Konfiguration Part.	Busbr.	PRR	FFs			LUTs			
				\emptyset_{Reg}	\emptyset_{Reg_bit}	Orig.	Opt.	\emptyset_{Reg}	\emptyset_{Reg_bit}	
DB	L	8	4	136	22,7	2,83	160	152	25,3	3,17
		32	4	544	90,7	2,83	640	608	101,3	3,17
		64	4	1088	181,3	2,83	1280	1216	202,7	3,17
	LR	8	8	248	22,5	2,82	288	288	26,2	3,27
		32	8	992	90,2	2,82	1152	1152	104,7	3,27
		64	8	1984	180,4	2,82	2304	2304	209,5	3,27
SEWB	L	95	4	1002	167,0	1,76	1220	968	161,3	1,70
	LR	95	8	1840	167,3	1,76	2060	1788	162,5	1,71
	M	95	2	868	173,6	1,83	800	558	111,6	1,17
MEWB	L	115	4	1779	296,5	2,58	2064	1949	324,8	2,82
	LR	115	8	3213	292,1	2,54	3668	3668	333,5	2,90
	M	115	2	1407	281,4	2,45	1262	1262	252,4	2,19
\emptyset				163,8	2,49			168,0	2,65	

Zweidimensionale Kommunikationsinfrastrukturen

Tabelle 4.7 führt den FPGA-Ressourcenbedarf der zweidimensionalen Kommunikationsinfrastrukturen exklusive aller optionalen Buserminierungen auf. Da in allen Partitionierungen eine einseitige Anbindung des dynamischen Bereichs erfolgt, ergeben sich in allen Designs Optimierungen. Erkennbar ist, dass die Kommunikationsinfrastrukturen *DB 2x2* aufgrund der gleichen Anzahl an Regionen (inklusive der Buserminierung) identische Werte zu der eindimensionalen Kommunikationsinfrastruktur *DB L* ergeben.

Der Durchschnittswert \emptyset_{Reg} liegt für die Register bei einer Bitbreite von 8 bei etwa 23 FFs/Region und steigt auf maximal 305 FFs/Region für einen 115-Bit breiten Bus. Die Mittelwerte der LUTs variieren von 25 LUTs/Region bis etwa 403 LUTs/Region. Der Bitbreiten-unabhängige Mittelwert \emptyset_{Reg_bit} liegt zwischen 2,08 FFs/Region/bit und 3,0 FFs/Region/bit für die Register und für die LUTs zwischen 2,39 LUTs/Region/bit und 3,86 LUTs/Region/bit.

Der Vergleich des FPGA-Ressourcenbedarfs der SEWB- und MEWB-Implementierungen fällt bezüglich der Register ähnlich zu den Ergebnissen der eindimensionalen Kommunikationsinfrastrukturen aus. So werden zwischen 50,0 % (2x2) und 59,8 % (5x2) mehr Register für eine Master-fähige Kommunikationsinfrastruktur benötigt. Der LUT-Vergleich ist mit Werten zwischen 39,6 % (5x2) und 45,7 % (2x2) deutlich geringer.

Wie bereits in Abschnitt 4.4.3 geschrieben, ist der Mehraufwand der zweidimensionalen MEWB-Kommunikationsinfrastruktur (2x2) mit vier PR-Regionen zu einer

Tabelle 4.7: FPGA-Ressourcen der 2D DPR-Kommunikationsinfrastrukturen

Typ	Design		PRR	FPGA-Ressourcen							
	Konfiguration Part.	Busbr.		FFs		LUTs		Orig.	Opt.	\emptyset_{Reg}	\emptyset_{Reg_bit}
				\emptyset_{Reg}	\emptyset_{Reg_bit}	\emptyset_{Reg}	\emptyset_{Reg_bit}				
DB	2x2	8	4	136	22,7	2,83	160	152	25,3	3,17	
		32	4	544	90,7	2,83	640	608	101,3	3,17	
		64	4	1 088	181,3	2,83	1 280	1 216	202,7	3,17	
	3x2	8	6	168	24,0	3,00	224	216	30,9	3,86	
		32	6	672	96,0	3,00	896	864	123,4	3,86	
		64	6	1 344	192,0	3,00	1 792	1 728	246,9	3,86	
SEWB	2x2	95	4	1 186	197,7	2,08	1 612	1 360	226,7	2,39	
	3x3	95	9	2 036	203,6	2,14	3 152	2 875	287,5	3,03	
	5x2	95	10	2 244	204,0	2,15	3 460	3 178	288,9	3,04	
MEWB	2x2	115	4	1 779	296,5	2,58	2 096	1 981	330,2	2,87	
	3x3	115	9	3 054	305,4	2,66	4 141	4 026	402,6	3,50	
	5x2	115	10	3 585	298,8	2,60	4 550	4 435	369,6	3,21	
\emptyset				176,0	2,64			219,7	3,26		

vergleichbaren eindimensionalen MEWB-Kommunikationsinfrastruktur (L) bezüglich der FPGA-Ressourcen sehr gering. So werden lediglich 1,6 % mehr LUTs für die zweidimensionale Kommunikationsinfrastruktur benötigt. Ein Vergleich der SEWB-Kommunikationsinfrastrukturen führt hingegen zu einem anderen Ergebnis. Die zweidimensionale Kommunikationsinfrastruktur (2×2) benötigt 18,4 % mehr Register und 40,5 % mehr LUTs.

4.5 Softwarekomponenten des DHHarMa Backends

In diesem Abschnitt werden die wesentlichen Komponenten des DHHarMa Backends (siehe Abschnitt 4.2.2) vorgestellt. Bevor die einzelnen Homogenisierungswerkzeuge – Packer, Placer und Router – beschrieben werden, wird auf die FPGA-Datenbanken eingegangen. Diese enthalten Informationen über die verfügbaren Ressourcen und die Verdrahtungsinfrastruktur eines FPGAs und sind daher essentiell für die aufgeführten DHHarMa Werkzeuge. Alle Softwarekomponenten wurden in der Programmiersprache C++ implementiert. Zusätzlich zu den *Standard Template Library* (STL) Klassen, wurden Bibliotheken der freien C++ Bibliothek *boost* [28] verwendet. Wesentliche Bibliotheken sind hierbei *Unordered* und *Serialization*. Die erstgenannte Bibliothek stellt ungeordnete Container (Datenstrukturen, z. B. Hash-Tabellen) zur Verfügung, welche gegenüber den STL-Containern den Vorteil besitzen, dass diese nicht entarten können. Hierdurch ist ein performanter Zugriff in $O(\log(n))$ auf interne Daten garantiert. Die interne Datenstruktur der STL-Containern kann hingegen zu einer Liste entarten, wodurch die

Zugriffszeit im schlechtesten Fall $O(n)$ beträgt. Die Bibliothek *Serialization* wird zur Speicherung der Software-internen Datenbank auf der Festplatte verwendet. Weiterhin erfolgt eine Komprimierung der Datenbank mit der Bibliothek *bzip2* [32].

4.5.1 Xilinx FPGA-Datenbanken

Die Software-Werkzeuge von Xilinx verwenden eine Datenbank im proprietären NPH-Format. Wie in Abschnitt 4.3.1 beschrieben, können die Informationen aus dieser Datenbank in Form eines XDL-Reports generiert werden und sind somit frei verfügbar. In dem ersten Abschnitt wird die FPGA-Datenbank basierend auf diesen XDL-Reports vorgestellt. Der Xilinx *FPGA Editor* basiert auf der gleichen Xilinx Datenbank, verwendet jedoch für die Referenzierung der Leitungen anstelle des XDL-Namens eine konkrete ID. Für die Erstellung von *FPGA Editor* Skripten zur nachträglichen Manipulation eines Designs wurde daher eine weitere Datenbank erstellt, welche im zweiten Abschnitt beschrieben wird. Abschließend erfolgt eine Gegenüberstellung beider FPGA-Datenbanken.

Die Datenbanken der bereits in Abschnitt 3.5 genannten akademischen Projekte Torc (*Tools for open reconfigurable computing*) [175] und RapidSmith [140] basieren ebenfalls auf den XDL-Reports. Die im folgenden Abschnitt vorgestellte Datenbank bietet jedoch als einzige Datenbank eine Kompatibilität zum Xilinx *FPGA Editor*.

XDLRC-basierte FPGA-Datenbank

Eine direkte Verwendung der XDL-Reports ist aus mehreren Gründen nicht empfehlenswert. Wie bereits in Abschnitt 4.3.1 beschrieben, enthält ein Report redundante Informationen (insbesondere durch die Leitungs- und Verbindungsdefinitionen), wodurch der benötigte Speicherplatz sehr hoch ist. Dies führt wiederum zu einem langsamen Zugriff auf eine bestimmte Information aus dem Report. Dies sind die wesentlichen Gründe für die Erstellung einer effizienten Datenbank, der *Datenbank für Xilinx FPGAs* (DXF).

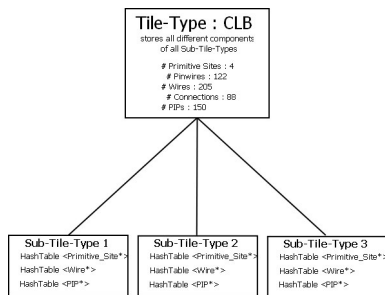
DXF-V1 und allgemeine Phasen zur Erstellung der Datenbank

Die erste Version der DXF wurde im Jahr 2009 erstellt und wird in [102] vorgestellt. Die DXF basiert auf XDL-Reports der zweiten Detailstufe und unterstützt die Xilinx Virtex-4-Familie. Die Erstellung der Datenbank geschieht in zwei, respektive drei Phasen, welche ausführlich in [102, S. 47 ff.] und zusammengefasst in [103, S. 49 ff.] beschrieben sind. Die folgende Zusammenfassung der Phasen basiert auf der zuletzt genannten Quelle.

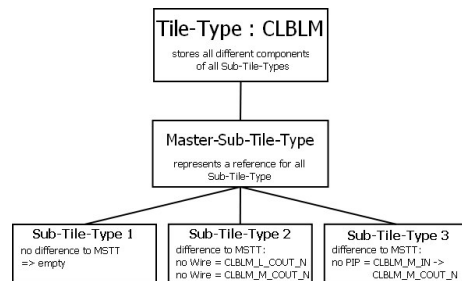
In *Phase I* wird der XDL-Report durch einen Parser eingelesen und für jeden Kacheltyp (engl. Tile Type, TT) ein Objekt angelegt. Während des Einlesevorgangs werden die Leitungsamen der Leitungen, welche durch PIPs verbunden werden können, in dem dazugehörigen TT-Objekt abgespeichert. Diese Leitungen repräsentieren die physikalischen Leitungen eines FPGAs. Leitungen, die nicht mit einem PIP verbunden sind, enthalten lediglich redundante Verbindungsinformationen und werden für die FPGA-Repräsentation nicht benötigt.

In *Phase II* wird der Report ein weiteres Mal eingelesen. Nun werden alle Kacheln in einem temporären Objekt, einem sogenannten *Sub-Tile Type* (STT)-Objekt, eingelesen.

Dabei werden die Informationen aller enthaltenen Komponententypen (Primitive Site, Pinwire, Wire, Connection und PIP) zunächst in temporären Objekten abgebildet. Alle sich unterscheidenden STTs eines TTs werden in dem übergeordneten TT in einer Hash-tabelle permanent abgespeichert. Nachdem ein STT vollständig eingelesen wurde, wird es mit allen abgespeicherten STTs verglichen. Existiert bereits ein STT mit exakt den gleichen temporären Komponentenobjekten, wird das temporäre STT verworfen und für die aktuelle Kachel die Referenz auf das existierende, vollständige STT-Objekt abgespeichert. Enthält das aktuell eingelesene STT hingegen wenigstens einen Unterschied und stellt damit ein STT dar, welches in der Kombination von Komponentenobjekten noch nicht in dem TT existiert, wird dieses STT-Objekt in die Datenstruktur des TT aufgenommen. Die Abbildung 4.16 (a) zeigt die daraus resultierende Datenstruktur anhand des Tile-Typs CLB.



(a) DXF mit den Objekten Tile-Type und Sub-Tile-Type [102, S. 52]



(b) DXF-V2 mit den Objekten Tile-Type, Master-Sub-Tile-Type und Sub-Tile-Type [103, S. 56]

Abbildung 4.16: Datenstruktur der DXF des Kacheltyps CLB

Mit den temporären Komponentenobjekten wird analog verfahren: Nach dem Einlesen einer Komponente wird diese mit existierenden Komponentenobjekten des gleichen Typs verglichen. Neue Komponenten werden in einer Hash-tabelle in dem TT permanent gespeichert. Existiert bereits eine Komponente in der Hash-tabelle wird das temporäre Objekt gelöscht und das existierende Objekt referenziert. Generell wird in dem STT-Objekt stets ein Pointer zu der Komponente aus dem TT-Objekt in einer eigenen Hash-tabelle abgespeichert. Durch das beschriebene Vorgehen wird der redundante Anteil des Reports herausgefiltert und nicht mit in die Datenbank übernommen. Mit Beendigung der Phase II enthält die Datenbank alle Information zur Beschreibung eines vollständigen FPGAs und wird in komprimierter Form, unter der Verwendung der bzip2-Bibliothek, auf der Festplatte abgespeichert. Die DXF kann innerhalb weniger Sekunden eingelesen und somit wiederhergestellt werden.

In der optionalen *Phase III* werden Wire- und Connection-Objekte analysiert und Objekte, welche die physikalischen Leitungen repräsentieren, erstellt. Dies ist insbesondere für den *Router* der DHHarMa-Implementierung notwendig, da die Wire- und Connection-Objekte von mehreren STTs referenziert werden. Durch die Erzeugung der physikalischen Leitungen steigt der Speicherbedarf aufgrund der großen Anzahl physikalischer Leitungen eines FPGAs stark an. In der ersten Version der DXF wurde diese Detailstufe der Datenbank als zusätzliche DXF erzeugt. Da Phase III in einem zeitlichen Rahmen von Sekunden bis wenigen Minuten liegt, wird diese Detailstufe für die folgenden Versionen jedoch nicht mehr erzeugt.

DXF-V2

Eine Überarbeitung der DXF erfolgte im Jahr 2011. Die Änderungen sind detailliert in [103, S. 49 ff.] beschrieben. Die DXF-V2 basiert nun auf XDL-Reports der dritten Detailstufe, sodass die Verfahren zur Überbrückung der fehlerhaften Trennungen der Leitungen (siehe [102, S. 36 ff.]) nicht mehr benötigt werden. Zusätzlich wurde die Datenstruktur optimiert. Während des ersten Lesevorgangs des Reports wird für jeden TT ein sogenanntes *Master-Sub-Tile-Type* (MSTT) Objekt erstellt. In diesem Objekt werden Komponenten, die in allen Kacheln dieses TTs vorkommen, gespeichert. Das MSTT-Objekt erzeugt eine neue Hierarchieebene innerhalb der Datenstruktur. Die Abbildung 4.16 (b) zeigt diese Veränderung innerhalb der Datenstruktur anhand des Kacheltyps CLB. Das MSTT-Objekt enthält die Pointer aller Komponentenobjekte, welche von jedem STT verwendet werden und stellt somit eine Referenz für die STTs dar.

Weiterhin wurden die Xilinx FPGA-Familien Virtex-5, Spartan-6 und Virtex-6 aufgenommen. Die Tabelle A.6 und Tabelle A.7 im Anhang B.2 stellt eine Auswertung des Speicherplatzes und der Ausführungszeit für das jeweils größte FPGA der Unterfamilien aller unterstützten FPGA-Familien bis zur Virtex-6-Familie vor. Diese Ergebnisse wurden in [105] veröffentlicht.

DXF-V3

Neben der Integration der 7-Serie- und Zynq-7000-FPGA-Familie wurden in der dritten Version der DXF Anpassungen durchgeführt, um eine Kompatibilität mit der Xilinx *FPGA Editor* Datenrepräsentation herzustellen. Hierzu müssen für die FPGA-Repräsentation irrelevante Leitungsdefinitionen bzw. Verbindungsdefinitionen der Datenbank hinzugefügt werden. Details hierzu werden nach der Vorstellung beider Datenbanken beschrieben.

FPGA Editor Datenbank

Die *FPGA Editor* Datenbank wurde primär zur Skriptgenerierung für den *FPGA Editor* für eine (nachträgliche) Manipulation von Designs implementiert. In INDRA 2.0 wird dieser Nachbearbeitungsschritt an dem DPR-Design durchgeführt, um statische Signale die eine PR-Region durchkreuzen unter Einhaltung der PR-Region-Grenzen neu zu

verdrahten (siehe Abschnitt 3.5.7). Die Darstellung der FPGA-Komponenten in dem *FPGA Editor* weicht in den Standardeinstellungen der Software *FPGA Editor* stark von der XDL-Darstellung ab, sodass diese zunächst erläutert wird. Durch Modifikationen der Konfiguration der Software können jedoch Bindeglieder zur XDL-Repräsentation erstellt werden. Abschließend werden die Schritte zur Extraktion der Komponenteninformationen beschrieben.

Komponentendarstellung im *FPGA Editor*

Die Repräsentation von FPGA-Komponenten basiert im *FPGA Editor* auf separaten, feingranularen kartesischen Koordinatensystemen. Wird beispielsweise eine Leitung per Mausklick selektiert, erfolgt die folgende Ausgabe in das Ausgabefeld der GUI (*History*) und der Logdatei:

```
node = BENTQUAD(541, -3528)
```

Diese Ausgabe enthält den Leitungstyp (*BENTQUAD*) und einen eindeutigen Referenzpunkt der physikalischen Leitung im Koordinatensystem des *FPGA Editors* (541 und -3528). Im Falle der Leitungen wird das Koordinatensystem in vier Quadranten aufgeteilt, wobei der Referenzpunkt (0,0) auf den Koordinatenursprung, die FPGA-Mitte, zeigt. Die erste Koordinate (x) beschreibt die Lokalisierung auf der Abszissenachse, die zweite Koordinate (y) die Lokalisierung auf der Ordinatenachse.

Diese Informationen allein sind jedoch unzureichend für eine automatisierte Skriptgenerierung, da eine Zuordnung zu der XDL-Repräsentation fehlt. Durch die Aktivierung eines Modus über die Kommandozeile (*Command*) oder einer zusätzlichen Einstellungsdatei können zusätzliche Leitungsinformationen aktiviert werden:

```
setattr main wire_info on
```

Eine Selektion der gleichen Leitung liefert anschließend folgende Ausgabe:

```
node = BENTQUAD(541, -3528) (nodeidx:5687973 wirename:NW6E3 pin=0  
long:0 vert:0 segmented:1 cost:3 speedmodel:RC_NW6E speedidx:542)
```

Basierend auf dieser Ausgabe ist erkenntlich, dass der *FPGA Editor* für jede physikalische Leitung eine eindeutige ID (*nodeidx*) speichert. Der darauffolgende Leitungsname entspricht dem Leitungsnamen in der XDL-Repräsentation und stellt daher ein benötigtes Bindeglied zwischen beiden Datenbanken dar. Die anderen angegebenen Informationen werden derzeit nicht ausgewertet, da sie weder dokumentiert, noch eine direkt aus- bzw. verwertbare Information enthalten. Zusammenfassend sind daher folgende Leitungsinformationen relevant:

1. Leitungstyp: BENTQUAD
2. Referenzpunkt der Leitung in *FPGA Editor* Koordinaten: (541,-3528)
3. Leitungs-ID: 5687973
4. Leitungsnamen: NW6E3 (identisch zu XDLRC)

Da in der XDL-Repräsentation wiederum keine eindeutige ID für eine Leitung existiert, wird eine andere Lokalisierungsinformation benötigt. In Anlehnung an den XDL-Report wird hierfür der Name der Kachel verwendet, in der die Leitung lokalisiert ist. Die Kachelinformationen müssen hierfür ebenfalls durch ein nicht dokumentiertes Kommando aktiviert werden:

```
setattr main show_tiles on
```

Anschließend erhält man folgende Informationen zu einer Kachel:
TILE(115,181) name=INT_R_X47Y175 type=INT_R

Diese Darstellung entspricht der Kachelinformation der Repräsentation im XDL-Report (siehe Abschnitt 4.3.1) und stellt somit das letzte benötigte Bindeglied dar.

Extraktion der Komponenteninformationen

Für die Extraktion der erwähnten Informationen wurden generische Skripte für die freie Automatisierungssoftware AutoIt [11] erstellt. Die Skripte sind in einer BASICähnlichen Sprache geschrieben und können in eine, unter Windows, ausführbare Datei übersetzt werden.

Die eigentliche Extraktion der Komponenteninformationen erfolgt in mehreren Phasen. In *Phase I* werden die ersten beiden Leitungsinformationen aus einem Fenster für die Leitungseinstellungen des *FPGA Editor* extrahiert und in eine eigene Logdatei ("*FPGA*"_wirename_simple.log) gespeichert. Dies erfolgt über folgende Kommandos des FPGA Editors:

- `select wire -id nodeidx`
- `post attr`

Das erste Kommando selektiert eine bestimmte Leitung über die ID, das zweite Kommando öffnet ein Fenster mit den Leitungsinformationen. Der *FPGA Editor* erlaubt hierbei maximal 20 gleichzeitig geöffnete Fenster, welche nach der Extraktion der Daten durch das AutoIt-Skript geschlossen werden. Der Extraktionsschritt kann durch ein oder mehrere Skript(e) durchgeführt werden, in der die ID inkrementiert wird. Durch die Aufteilung in mehrere Skripte kann diese Phase auf verschiedenen Systeme zur Beschleunigung des Prozesses parallelisiert werden. Eine parallele Ausführung auf dem gleichen System ist aufgrund der Detektion von geöffneten *FPGA Editor* Fenstern mittels AutoIt nicht möglich.

Mit Hilfe der ersten Logdatei können in *Phase II* die detaillierten Leitungsinformationen ermittelt und in einer weiteren Logdatei ("*FPGA*"_wirename_advanced.log) gespeichert werden. Die Ausgaben dieser Leitungsinformationen werden über das Kommando *pick* durch Verwendung der jeweiligen Referenzpunkte aus der Logdatei "*FPGA*"_wirename_simple.log erzeugt: `pick -a x y`

Identisch zu Phase I, kann die Ausführung durch ein oder mehrere Skript(e) erfolgen. Da keinerlei Operationen auf GUI-Elementen wie Fenstern durchgeführt werden, kann die Ausführung der AutoIt-Skripte auch auf einem System erfolgen. Ferner ist es

möglich die Skripte in der Kommandozeilenversion des FPGA Editors (`fpga_edline`) durchzuführen. Dies beschleunigt die Erstellung der zweiten Logdatei deutlich, da keinerlei Anzeigeoperationen ausgeführt werden müssen.

Die Kachelinformationen werden in der *Phase III* ermittelt und in einer Logdatei ("`FPGA_tile.log`") gespeichert. Zur Ermittlung der Kachelinformationen müssen zuvor die Darstellungsebenen der Leitungen durch ein Kommando deaktiviert werden: `wire_layers_disable`

In der *Phase IV* werden die gesammelten Informationen der Logdateien aus Phase II und III ausgewertet und neu organisiert abgespeichert. Für jeden Kacheltyp wird ein eigener Ordner erstellt, in dem für jede Kachelinstanz eine separate Textdatei mit allen dazugehörigen Leitungsinformationen abgelegt ist. Die Textdatei enthält als Namen die vollständige Kachelinformation.

Der Zugriff auf die in Phase IV erstellte textuelle Datenbank kann durch eine optionale *Phase V* beschleunigt werden. Diese Phase kann in der DHHarMa-Software aktiviert werden, um die textuelle Datenbank in eine C++ Datenbank zu überführen. Die Datenbank wird analog zu der DXF komprimiert abgespeichert.

Vergleich der Datenbanken

Ein Vergleich der Leitungsinformationen der *FPGA Editor* Datenbank mit der DXF-V2 Datenbank zeigte, dass in beiden Datenbanken unterschiedliche Leitungen fehlten. Wie bereits erwähnt, ist der Referenzpunkt einer physikalischen Leitung in dem *FPGA Editor* eindeutig, da dieser bei der Auswahl an verschiedenen Teilabschnitten der physikalischen Leitung im Ausgabefenster angezeigt wird. In einem XDL-Report (der dritten Detailstufe) hingegen, werden alle möglichen Teilabschnitte der physikalischen Leitung (Verbindungsdefinition *conn*) in jeder Leitungsdefinition (*wire*) aufgeführt. Eine Analyse ergab, dass in einigen, wenigen Fällen die *FPGA Editor* Datenbank Referenzpunkte verwendet, welche in der DXF-V2 Datenbank aufgrund fehlender physikalischer Anschlussmöglichkeiten nicht berücksichtigt werden. Ein Beispiel für eine solche Verbindung wird im Abschnitt 4.3.1 *Connection* erläutert. Der Grund für die Auswahl dieser Referenzpunkte ist nicht erkennbar und könnte fehlerhafter Natur sein. Um die Kompatibilität zwischen beiden Datenbanken herstellen zu können, musste eine dritte Version der DXF erstellt werden, welche irrelevante Verbindungsinformationen enthält. Die Auswertungen in Abschnitt 4.6.1 zeigen den Einfluss auf den benötigten Speicherplatz dieser neuen Datenbank.

4.5.2 Parser

Neben der FPGA-Repräsentation, werden zwei weitere Eingangsdateien benötigt: Die FPGA-Partitionierung und die XDL-Designbeschreibung des DHHarMa Frontends. Beide Dateien werden durch Parser in die DHHarMa-Software eingelesen und sind detailliert in [103, S. 66 ff.] beschrieben. Zur vollständigen Beschreibung und zum Verständnis des Entwurfsablaufs erfolgt an dieser Stelle eine Zusammenfassung.

Parser der FPGA-Partitionierung

Die Partitionierung des FPGAs wird in einer CSV-Datei mit der Dateiendung FPF (*FPGA Partitioning File*) definiert. Der Eintrag einer Partition, im folgenden auch Region genannt, setzt sich aus vier Angaben zusammen: Typ, Name, Referenzpunkt1 und Referenzpunkt2.

Zur Unterscheidung von Regionstypen wird als erstes ein Schlüsselwort gesetzt. Für inhomogene, statische Regionen wird das Schlüsselwort *Base*, für (homogene) rekonfigurierbare PR-Regionen das Schlüsselwort *Reconf* verwendet.

Der darauf folgende Name der Region setzt sich aus zwei oder vier Teilen zusammen:

1. Typ der Region: statische Region (*Base*) oder PR-Region (*Rec*)
2. Untertyp eines Regionstyps: Ziffer
3. Trennzeichen (engl. delimiter): "_" (nur für PR-Regionen)
4. Nummerierung: Buchstabe (nur für PR-Regionen)

Ein vollständiger Regionsname ist durch diese Konvention stets eindeutig. Die Namen von homogenen PR-Regionen teilen sich die beiden erstgenannten Teile, den Regionstyp und -untertyp (z. B. *Rec0_a* und *Rec0_d*). Diese definierten Regionsnamen müssen zur Erkennung innerhalb der Software ebenfalls in der Top-HDL-Datei des Designs verwendet werden.

Eine Region wird durch zwei Referenzpunkte aufgespannt, so dass stets eine rechteckige Form aufgebaut wird. Als Referenz können die Slice- oder Tile-Koordinaten verwendet werden. Beide Koordinatentypen können mit Hilfe des Xilinx FPGA Editors ermittelt werden. In der Auflistung 4.1 sind exemplarisch elf Regionen definiert. Acht Regionen sind hierbei als rekonfigurierbar und drei als statisch deklariert. Abbildung 4.17 stellt diese Partitionierung mit acht PR-Regionen und vier Regionstypen (*Rec0* - *Rec3*) dar. Die Terminierungen *Base1* und *Base2* sind nicht abgebildet.

```
Reconf , Rec0_a , SLICE_X2Y119 , SLICE_X9Y60
Reconf , Rec1_b , SLICE_X10Y119 , SLICE_X17Y60
Reconf , Rec2_c , SLICE_X18Y119 , SLICE_X25Y60
Reconf , Rec2_d , SLICE_X26Y119 , SLICE_X33Y60
5 Base , Base0 , SLICE_X34Y119 , SLICE_X69Y60
Reconf , Rec2_e , SLICE_X70Y119 , SLICE_X77Y60
Reconf , Rec1_f , SLICE_X78Y119 , SLICE_X85Y60
Reconf , Rec2_g , SLICE_X86Y119 , SLICE_X93Y60
Reconf , Rec3_h , SLICE_X94Y119 , SLICE_X103Y60
```

Auflistung 4.1: "Beispiel einer FPGA-Partitionierungsdatei im CSV-Format"

Am Ende des Einlesevorgangs erfolgt eine Überprüfung der, als homogen gekennzeichneten, Regionen. In dem Fall einer Nichtübereinstimmung, z. B. durch unterschiedliche FPGA-Ressourcen oder abweichender Regionsgröße, wird die Programmausführung terminiert und eine Fehlermeldung ausgegeben.

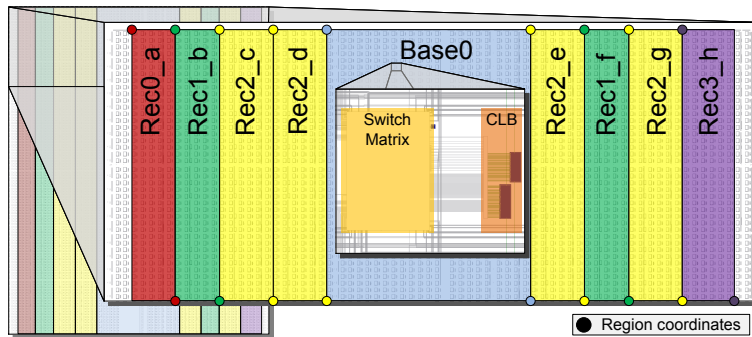


Abbildung 4.17: Partitionierung eines Virtex-6 FPGAs mit acht rekonfigurierbaren Regionen und vier unterschiedlichen Regionstypen [105, S. 3]

Parser des XDL-Designs

Die inhomogene XDL-Design-Beschreibung muss in eine interne Datenstruktur eingelesen werden. Diese Aufgabe übernimmt ein XDL-Parser unter Verwendung eines XDL-Lexers. Die genaue Arbeitsweise beider Komponenten wird detailliert in [103, S. 68 ff.] beschrieben. Zum Verständnis visualisiert Abbildung 4.18 den allgemeinen Einlesevorgang. Grau markierte Parts einer Slices repräsentieren verwendete Parts. Die Zuordnung zu den einzelnen Regionen ist farblich markiert.

4.5.3 Packer

Nachdem das eingangs inhomogene Design in eine interne Datenstruktur überführt wurde, erfolgt eine Homogenisierung der Slice-Konfigurationen in als homogen gekennzeichneten Regionen. Das Ziel des homogenen Packens ist, dass am Ende des Vorgangs in allen Regionen mit gleichem Regionstyp die identischen Parts in identischen Slices existieren. Je nach Slice-Architektur kann die Anfangskonfiguration der Parts stark variieren, sodass die gleiche Funktionalität durch verschiedene Part-Konfigurationen abgebildet sein kann.

Abbildung 4.19 stellt ein Beispiel für die Slice-Architektur ab der Virtex-6-Familie dar. Die Unterschiede zwischen den Konfigurationen zweier Regionen vom gleichen Typ sind rot gekennzeichnet. Während der Funktionsgenerator in der Region *Rec2_c* die LUT5 verwendet, wird in Region *Rec2_d* die LUT6 verwendet. Zusätzlich werden unterschiedliche Eingänge verwendet, sodass die Verdrahtung ebenfalls nicht gleich aufgebaut sein kann. Darüber hinaus verwenden beide Parts unterschiedliche Register und daher abweichende Ausgänge. Abschließend erfolgt die Konfiguration in zwei unterschiedlichen Parts (B und C). Das homogene Packen ist in mehrere Phasen unterteilt, welche detailliert in [103, S. 80 ff.] beschrieben und hier zusammengefasst werden.

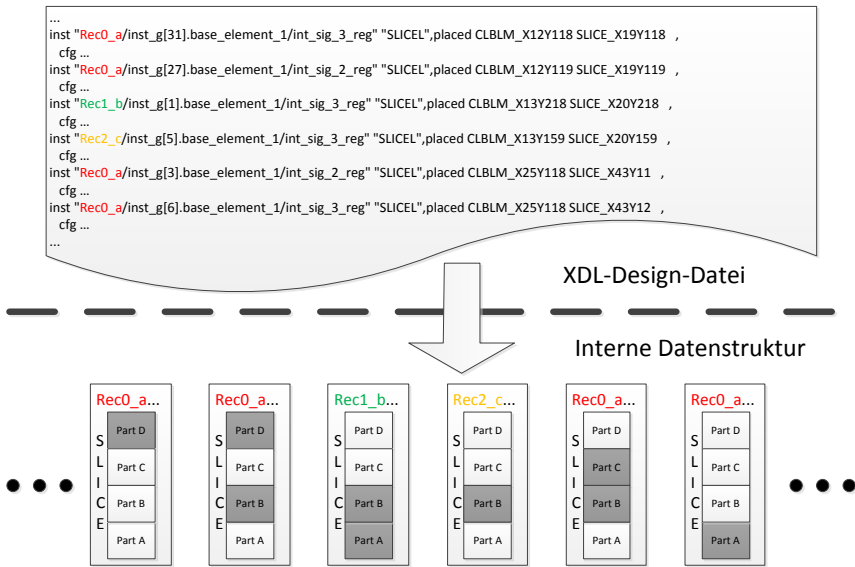


Abbildung 4.18: Einlesen der Slice-Instanzen einer XDL-Design-Datei [103, S. 69]

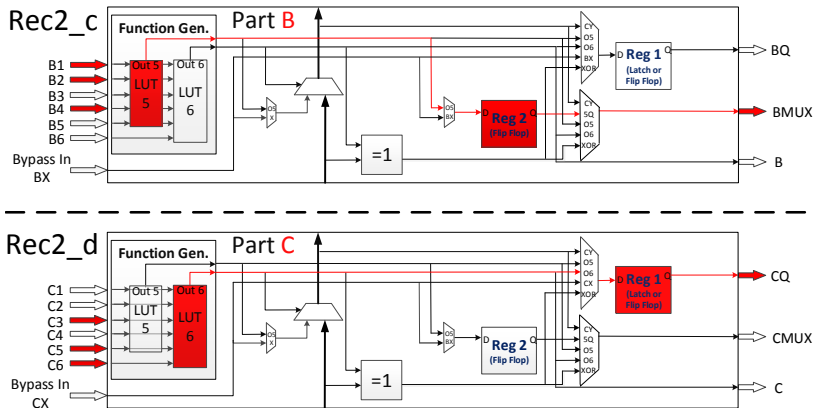


Abbildung 4.19: Unterschiedliche Packung der Slices innerhalb zweier Regionen vom gleichen Regionstyp [105, S. 4]

Analyse der Slice-Instanzen und Part-Kategorisierung

In einem ersten Schritt werden Cluster von Slice-Instanzen gebildet, wobei primär die Verbindungen zwischen Slices einer Region analysiert werden. Anschließend folgt für jedes Cluster eine Analyse aufgrund der initialen Konfiguration der Parts. Ein Part kann einer von drei Kategorien angehören:

1. *voll*: Min. eine LUT des Funktionsgenerators und ein Register
2. *FG*: Min. eine LUT des Funktionsgenerators und kein Register
3. *REG*: Keine LUT des Funktionsgenerators und min. ein Register

Um die Erkennung homogener Part-Konfigurationen zu vereinfachen, werden die Parts in elementare Blöcke aufgeteilt, in denen jeweils nur eine LUT pro Funktionsgenerator und ein Register pro Part verwendet werden. Während der Kategorisierung werden die Parts daher auf ihre derzeitige Konfiguration überprüft und gegebenenfalls Teilungsschritte durchgeführt. Bei diesen Vorgängen kann es dazu kommen, dass Netze modifiziert werden müssen. So kann bei einer Teilung eines Parts ein Eingangspin von beiden Konfigurationen verwendet werden, was eine Replikation des Pins in dem Netz nötig macht. Ein Beispiel für einen Teilungsschritt ist in Abbildung 4.20 dargestellt, in dem ein Part mit beiden verwendeten LUTs und Registern aufgeteilt wird. Weitere Beispiele sind in [103, S. 83] beschrieben.

Homogene Transformationsschritte

Nach der Kategorisierung der Parts und der Aufteilung in elementare Blöcke, werden die Konfigurationen in Regionen vom gleichen Typ überprüft und gegebenenfalls Transformationsschritte zur Erstellung einer homogenen Konfiguration durchgeführt. Zur Durchführung dieses Schritts wird eine Region als Vorbild, auch Master-Region genannt, verwendet und alle weiteren Regionen vom gleichen Typ entsprechend umkonfiguriert.

Im ersten Schritt werden zunächst die vollen Parts verglichen. Fehlt in einer Region ein voller Part, so existieren äquivalente *FG*- und *REG*-Parts, sodass ein neuer, identischer voller Part erstellt werden kann. Ist dies nicht der Fall, so existiert ein Fehler, welches dem Benutzer mitgeteilt wird. Die Konfiguration innerhalb der Register und/oder des Funktionsgenerators kann dabei abweichen, sodass weitere Transformationsschritte in dem *FG*- und/oder *Reg*-Part benötigt werden.

Existiert zum Beispiel in der Master-Region ein *FG*-Part, in dem die logische Funktion durch die LUT5 in einem Part einer anderen Region jedoch durch die LUT6 implementiert ist, so muss die LUT6 in eine LUT5 transformiert werden. Ein weiteres - die zwei Register ab der Virtex-6-Slice-Architektur betreffendes - Beispiel: In der Master-Region existiert ein *REG*-Part, welcher einen Wert in Register1 abspeichert. In einem Part einer anderen Region wird hingegen derselbe Wert in Register2 abgespeichert. Die Konvertierung eines Register1 zu einem Register2 (vice versa) wird somit benötigt. Bei allen Transformationsschritten müssen externe aber auch interne Verbindungen beachtet und gegebenenfalls weitere Transformationsschritte eingeleitet, sowie die verbundenen Netze angepasst werden.

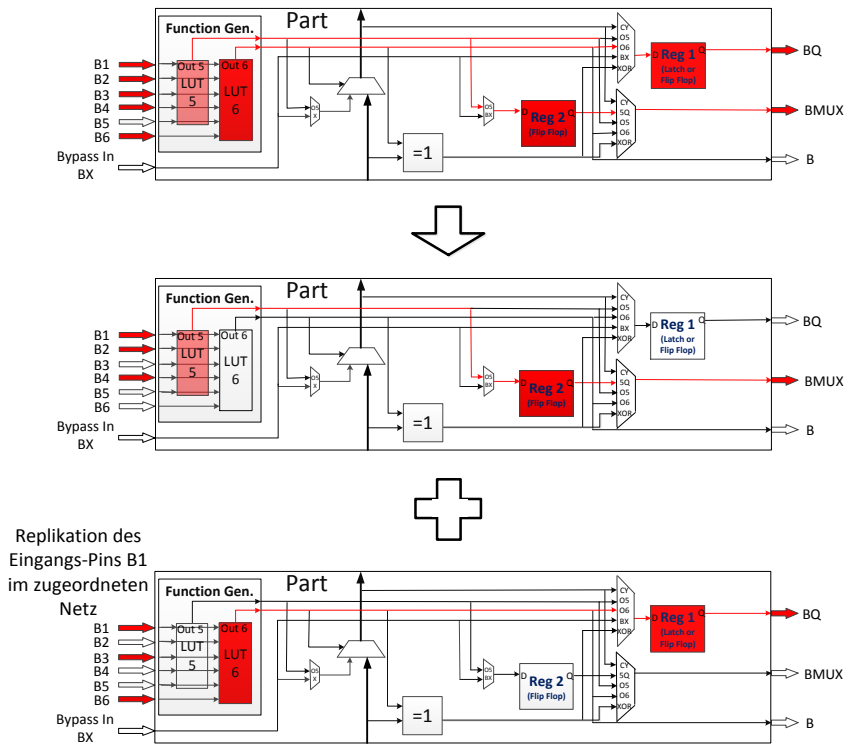


Abbildung 4.20: Aufteilen eines Parts mit zwei belegten Registern [103, S. 86]

Nach der erfolgreichen Überprüfung aller Part-Konfigurationen eines Regionstyps, kann unter Umständen ein weiterer Schritt eingeleitet werden: Die Erzeugung neuer (homogener) voller Parts aus existierenden *FG*- und *REG*-Parts.

Homogene Packung

Anschließend wird der eigentliche Packungsschritt in neue Slices, ebenfalls Cluster für Cluster einer Region, durchgeführt. Zunächst werden die vollen und anschließend die halbvollen Parts in neue Slices eingefügt. Bevor ein Part in eine Slice aufgenommen wird, wird diese auf Kompatibilität zum Part überprüft. Dabei wird unter anderem der Slice-Typ sowie die globalen Konfigurationsschalter (z. B. *CLKINV* oder *SYNC" _ATTR*) überprüft. In dem Fall, dass ein Part als erstes in eine Slice eingefügt wird, werden die globalen Konfigurationsschalter in die neue Slice expandiert. Wird während des Packens die maximale Anzahl an Parts einer Slice erreicht, wird eine neue Slice erstellt.

Die maximale Packungsdichte einer Slice ist, wie in Abschnitt 3.2 beschrieben, abhängig von der FPGA-Familie. So enthält ein Part folgende wesentliche Komponenten:

- 1 LUT und 1 Register : Virtex-4
- 2 LUTs und 1 Register : Virtex-5
- 2 LUTs und 2 Register : ab Virtex-6/Spartan-6

Ferner variiert die Anzahl der Parts einer Slice je nach FPGA-Familie. Eine Slice eines Virtex-4 enthält zwei Parts, die Slices aller anderen FPGA-Familien hingegen vier Parts. Es resultiert die maximale Packungsdichte einer Slice familienbedingt zu:

- 2 LUTs und 2 Register : Virtex-4
- 8 LUTs und 4 Register : Virtex-5
- 8 LUTs und 8 Register : ab Virtex-6/Spartan-6

Dadurch, dass von dem Packer immer nur eine LUT und ein Register eines Parts verwendet werden, wird die maximale Packungsdichte der Slices einzig für die Virtex-4-Slice-Architektur erreicht, sodass an dieser Stelle ein Optimierungspotential der Software existiert:

- 2 LUTs und 2 Register : Virtex-4
- 4 LUTs und 4 Register : Virtex-5
- 4 LUTs und 4 Register : ab Virtex-6/Spartan-6

Ergibt sich durch die Packung eine neue Part-Position innerhalb der Slice, müssen die Netze, genauer gesagt die Ein- und/oder Ausgänge der Netze, angepasst werden. Eine Auswertung des Packers erfolgt anhand der vorgestellten Anwendungsgebiete in Abschnitt 4.6.3 und Abschnitt 4.6.4.

4.5.4 Placer

Die nun homogen gepackten (Cluster-) Slices müssen anschließend homogen in Regionen vom gleichen Typ platziert werden. Neben Algorithmen zur *Cluster-basierten Platzierung* wurde der Algorithmus *Simulated Annealing* (dt. Simulierte Abkühlung) implementiert. Beide Verfahren sind in [103, S. 88 ff.] beschrieben. In diesem Abschnitt werden die Beschreibungen beider Verfahren zusammengefasst und Änderungen in der Cluster-basierten Platzierung vorgestellt.

4.5.4.1 Cluster-basierte Platzierung

Die Cluster-basierte Platzierung kann in vier Schritte unterteilt werden: Der Bestimmung einer Master-Region, der Erstellung von Platzierungsformen, der virtuellen und der finalen Platzierung. Im Rahmen der Dissertation wurde der dritte Schritt, die virtuelle Platzierung, durch die Verwendung der quelloffenen Bibliothek *RectangleBinPack* dem Verfahren der Cluster-basierten Platzierung hinzugefügt.

Bestimmung einer Master-Region In einem ersten Schritt wird für jeden Regionstypen für den mehrere Regionen des gleichen Typs existieren, sogenannte Slave-Regionen, eine Master-Region bestimmt. In dieser werden alle folgenden Schritte der Platzierung durchgeführt und für die Slave-Regionen repliziert.

Erstellung von Platzierungsformen Für jedes Cluster der Master-Region wird eine rechteckige Platzierungsform erstellt, welche auf der Anzahl benötigter Slice-Ressourcen beruht. Die Form kann dabei quadratisch oder rechteckig gewählt werden und enthält ferner die jeweilige Anzahl der benötigten Slice-Typen. Anschließend wird für die Master-Region eine rechteckige Platzierungsform erstellt, welche durch die Grenzen der Partitionierung der Region definiert ist. Diese Form kann durch globale (Eingangs-) Parameter der DHHarMa-Software minimiert werden, um so einen freien Bereich in Ringform um Regionen herum aufbauen zu können.

Virtuelle Platzierung mittels *RectangleBinPack* Der spaltenbasierte Aufbau von Xilinx FPGAs wird von FPGA-Familie zu FPGA-Familie irregulärer. So existieren ab der Virtex-5-Familie z. B. CLB-Spalten vom Typ CLB-LL (zwei Slices vom Typ L) und CLB-LM (jeweils eine Slice vom Typ L und M). Die Funktionalität einer Slice vom Typ L kann generell auch in einer Slice vom Typ M realisiert werden. Diese Slice-Konvertierung kann aktiviert werden, ist jedoch nur bedingt zu empfehlen. Die Funktionalität der Funktionsgeneratoren sollte nicht unnötig deaktiviert werden, da diese von weiteren Designteilen (z. B. Modulen eines DPR-Systems) benötigt sein könnten. Dieses Argument wird durch die begrenzte Anzahl der Slices vom Typ M unterstrichen.

Diese beschriebene Irregularität hat für die Platzierung zur Folge, dass die Cluster-Platzierungsform nicht eindeutig bestimmt werden kann, da diese von der gewählten Position abhängig ist. Durch die Verwendung von virtuellen Platzierungen können kleine Änderungen an den Platzierungsformen vorgenommen werden, um so eine, für ein ausgewähltes Cluster an einer bestimmten Position, optimale Form zu ermitteln. Diese virtuelle Platzierung wurde überarbeitet. Zentraler Bestandteil der Anpassungen ist die quelloffene C++-Bibliothek *RectangleBinPack* [92] von Jylänki. Wie der Name andeutet, enthält die Bibliothek Verfahren für ein kombinatorisches Optimierungsproblem, das sogenannte *Behälterproblem* (engl. bin packing problem). Die Bibliothek bietet drei Hauptverfahren für das zweidimensionale, rechteckige Behälterproblem: Skyline, Guillotine, MaxRect. Für jedes Verfahren kann weiterhin eine heuristische Auswahlregel, wie z. B. *Best/Worst Area Fit*, *Best/Worst Short Side Fit* oder *Best/Worst Long Side Fit*, bestimmt werden. Details sowie ein Vergleich der Verfahren werden ausführlich in der Dokumentation [92] gegeben. Diese Verfahren gehören zu den typischen Packungsverfahren, können aufgrund der Verwendung von Platzierungsformen auf die Platzierung übertragen werden. In diesem Fall existiert ein Behälter (die Master-Region-Platzierungsform) sowie n (Anzahl der Cluster) Objekte, die durch ihre Platzierungsform ebenfalls ein Rechteck aufspannen und in die Master-Region gepackt werden müssen. Beachtet werden muss jedoch, dass die Packungsverfahren von regulären Behältern mit einheitlichen Ressourcen ausgehen. Dies ist bei FPGAs, wie bereits beschrieben, nicht der Fall. Daher muss nach einem Platzierungsvorgang stets überprüft werden, ob die benötigten FPGA-Ressourcen mit den verfügbaren FPGA-Ressourcen übereinstimmen. Allgemein, werden die Platzierungsformen iterativ platziert. Änderungen an den Platzierungsformen können dabei Resultat der Verfahren und/oder der Überprüfung

der FPGA-Ressourcen sein. Ist die Platzierung aller Cluster-Platzierungsformen nicht möglich, kann dies mehrere Gründe haben. In einigen Fällen kann die Auswahl einer anderen Heuristik oder eines anderen Verfahrens zu einer erfolgreichen Platzierung führen. Eine unvollständige Platzierung kann jedoch auch auf eine zu geringe Größe der Platzierungsform der Master-Region und damit für das BinPacking unzureichende Partitionierung hindeuten.

Finale Platzierung Im Anschluss an eine erfolgreiche virtuelle Platzierung folgt die Übernahme der ermittelten Positionen für jedes Cluster. Während dieses Vorgangs werden die konkreten Lokalisierungsinformationen in Form von XY-Koordinatenpaare für den Namen der Kachel und der Primitive Site ermittelt (siehe Abschnitt 4.3.2 und Abbildung 4.7). Nachdem eine Slice eines Cluster in der Master-Region platziert wurde, wird die relative Position zur nördlichen und östlichen Regions-Grenze bestimmt. Diese relative Position wird anschließend zur Platzierung der gleichen Slices aller Slave-Regionen verwendet.

4.5.4.2 Simulated Annealing

Simulated Annealing beschreibt ein heuristisches Optimierungsverfahren, welches typischerweise eingesetzt wird, wenn aufgrund einer hohen Anzahl an Möglichkeiten ein einfaches mathematisches Verfahren nicht benutzt werden kann [188]. Anwendung findet das Verfahren z. B. im Floorplanning im Chipentwurf, welches dem Platzieren von Slices auf einem FPGA ähnelt. Der Name des Verfahrens beruht auf dem Abkühlungsprozess eines erhitzten Metalls, in welchem die Atome ausreichend Zeit haben stabile Kristalle zu bilden und somit ein energiearmer Zustand nahe am Optimum erreicht wird. Im Gegensatz zu anderen Verfahren, darf sich bei dem Simulated Annealing Verfahren ein Zwischenergebnis im Verlauf verschlechtern, wodurch ein lokales Minimum verlassen werden kann.

Der in DHHarMa implementierte Algorithmus basiert auf dem *Versatile Place and Route* (VPR) Algorithmus von Betz et al. [27]. Betz et al. stellen mehrere Formeln vor, welche die Ausführung des Verfahrens beeinflussen und teilweise empirisch ermittelte Parameter enthalten. Diese Formeln werden im einzelnen kurz vorgestellt und Modifikationen in dem *DHHarMa Placer* erläutert.

Kostenfunktion Betz et al. stellen eine Kostenfunktion vor, welche intern für die Berechnung der Endtemperatur verwendet wird, jedoch auch als allgemeines Bewertungsmaß für die Platzierung verwendet werden kann:

$$Cost = \sum_{n=1}^{N_{nets}} q(n) \left[\frac{bb_x(n)}{C_{av,x}(n)} + \frac{bb_y(n)}{C_{av,y}(n)} \right] \quad (4.1)$$

Für die Berechnung der Gesamtkosten werden die Einzelkosten aller Netze (N_{nets}) berechnet und aufsummiert. Der Parameter $q(n)$ in der Formel modifiziert dabei eine zu optimistische Kostenberechnung aufgrund eines stark verzweigten Netzes (engl. fanout).

Für Netze mit bis zu drei Verzweigungen, welches drei Eingängen entspricht, besitzt der Parameter den Zahlenwert 1. Der Parameter steigt anschließend exponentiell auf einen Maximalwert von 2,79 für 50 Verzweigungen. Die Variablen $bb_x(n)$ und $bb_y(n)$ geben für jedes Netz die Anzahl der Switch-Matrizen, in [27] auch **bounding box** (bb) genannt, in X- und Y-Richtung wieder. Beide Variablen bilden die sogenannte Manhattan-Distanz oder Manhattan-Metrik, die Distanz zwischen zwei Punkten in Anlehnung an das orthogonale Straßengitter in Manhattan. Die Variablen $C_{av,y}(n)$ und $C_{av,x}(n)$ geben die durchschnittliche Anzahl von Leitungsressourcen in X- und Y-Richtung an und sind innerhalb einer FPGA-Familie konstant.

Anzahl Umplatzierungen Zur Bestimmung der Anzahl der Umplatzierungen (der Slices) innerhalb eines Temperaturschritts wird in [27] folgende Formel vorgestellt:

$$N_{swaps} = 10 \cdot (N_{slices})^{1,33} \quad (4.2)$$

Die Variable N_{slices} steht dabei für die Gesamtanzahl der Slices. Diese Formel hat den größten Einfluss auf die Ausführungszeit, da die Ausführung der Umplatzierungen der Slices der rechenintensivste Teil des Algorithmus ist. Eine Reduzierung von N_{swaps} führt daher zu einer wesentlich kürzeren Ausführungszeit, wodurch allerdings die Qualität der Platzierung sinkt. Betz et al. dokumentieren hierzu folgende Abschätzung: Wird die Anzahl der Umplatzierungen durch einen Faktor 10 reduziert, ergibt sich eine Platzierungsbeschleunigung um einen Faktor 10 mit einer um 10 % reduzierten Platzierungsqualität.

Aufgrund dieser Aussage wurde die Funktion durch Hinzufügen der Variablen β auf eine allgemeinere Form gebracht:

$$N_{swaps} = \beta \cdot (N_{slices})^{1,33} \quad (4.3)$$

Allgemein kann für beide Formeln festgehalten werden, dass die Anzahl der Umplatzierungen für jeden Temperaturschritt konstant ist.

Akzeptierte Umplatzierungen Die Variable R_{accept} enthält die prozentuale Anzahl akzeptierter Umplatzierungen in dem aktuellen Temperaturschritt. Laut Betz et al. sollte dieser Wert möglichst lange in der Nähe von 44 % bleiben. Dies wird erreicht, indem die Variable R_{accept} Einfluss nimmt auf die maximale Distanz zur ursprünglichen Position in welcher Umplatzierungen durchgeführt werden können. Der Wert der Variablen wird in jedem Temperaturschritt auf Basis der akzeptierten Umplatzierungen aktualisiert:

$$D_{limit} = D_{limit} \cdot (0,56 + R_{accept}) \quad (4.4)$$

Initial wird für die Variable D_{limit} ein großer Wert gewählt. In der ursprünglichen Implementierung wird der Wert auf die maximal mögliche Größe (Breite oder Länge) des FPGAs gesetzt. Bei der Implementierung des *DHHarMa Placers* muss darauf geachtet werden, dass eine Umplatzierung innerhalb der Region passiert und somit die Grenzen

der Region anstatt des gesamten FPGAs eingehalten werden. Die Variable D_{limit} wird daher initial an die Größe der Region angepasst. Im Verlauf nähert sich D_{limit} sukzessive dem Wert 1 an.

Anfangstemperatur Die Temperatur ist der wesentliche Parameter des Simulated Annealing Verfahrens. Die Anfangstemperatur wird in [27] in mehreren Schritten ermittelt. Zunächst erfolgt eine zufallsgenerierte Platzierung des gesamten Designs. Anschließend werden N_{slices} Umplatzierungen mit Berechnung der Standardabweichung der resultierenden Kosten durchgeführt. Die Anfangstemperatur wird auf einen Wert gesetzt der 20-fach höher ist als diese Standardabweichung. Diese Vorgehensweise zur Bestimmung der Anfangstemperatur wird ebenfalls in DHHarMa verwendet. Zusätzlich kann jedoch auch ein fester Wert verwendet werden. Diese Option wurde hinzugefügt, da neben einer zufallsgenerierten Platzierung auch die Cluster-basierte Platzierung verwendet werden kann, welche bereits eine gute Eingangsplatzierung darstellt. Im Fall einer Cluster-basierten Platzierung wird daher ein Simulated Annealing mit einer geringen Temperatur durchgeführt. Zusätzlich besteht die Möglichkeit, negative Umplatzierungen direkt zu verwerfen. Dies birgt allerdings das Risiko in sich, in einem lokalen Minimum zu landen. Da mit der Cluster-basierten Platzierung eine nahezu optimale Platzierung gefunden wurde, stellt dies jedoch kein großes Risiko dar.

Temperaturverlauf Der Temperaturverlauf während der Ausführung wird indirekt durch die Variable R_{accept} bestimmt. In Abhängigkeit dieses Wertes wird ein weiterer Parameter α auf vier empirisch ermittelte Werte gesetzt:

- $\alpha = 0.50 : R_{accept} > 96\%$
- $\alpha = 0.90 : 80\% < R_{accept} \leq 96\%$
- $\alpha = 0.95 : 15\% < R_{accept} \leq 80\%$
- $\alpha = 0.80 : R_{accept} \leq 15\%$

Die Temperatur des nächsten Schritts wird anschließend durch folgende Formel bestimmt:

$$T_{new} = \alpha \cdot T_{old} \quad (4.5)$$

Ausgangstemperatur Die Ausgangstemperatur stellt den Schwellenwert der Ausführung des Algorithmus dar und wird über die folgende Formel berechnet:

$$T_{exit} = 0.005 \cdot \frac{Cost}{N_{nets}} \quad (4.6)$$

Parameter des Algorithmus Zusammengefasst können fünf Parameter zur Modifikation des Simulated Annealing Algorithmus eingestellt werden:

- Anfangstemperatur T_{init}
- Maximal erlaubte Distanz einer Umplatzierung D_{limit}
- Faktor β zur Berechnung der Anzahl der Umplatzierungen
- Verwerfen von Verschlechterungen (Gefahr eines lokalen Minimums)
- Aktivierung der SA in bestimmten Regionstypen (Base, Reconf, Base&Reconf)

Die Abbildung A.3 in Anhang B.3 zeigt wesentliche Ausschnitte des Programmcodes des Algorithmus. Die in diesem Abschnitt aufgeführten Formeln sind durch Kommentare hervorgehoben. Eine dedizierte Auswertung der vorgestellten Platzierungsalgorithmen erfolgt in Abschnitt 4.6.2.

4.5.5 Router

Dieser Abschnitt stellt zur vollständigen Beschreibung des Gesamtentwurfsablauf das generelle Prinzip der homogenen Verdrahtung durch einen Router vor und basiert auf [103, S. 94 ff.]. Der Router wurde von Cozzi entwickelt und wird detailliert in [42] und [43] beschrieben.

Um eine homogene Verdrahtung realisieren zu können, müssen die homogen gepackten und platzierten Slices in Regionen vom gleichen Typ für die Verbindungen die gleichen Leitungsressourcen verwenden. Die Abbildung 4.21 stellt eine inhomogene (links) und homogene (rechts) Verdrahtung gegenüber. In der rechten Abbildung werden innerhalb der Switch-Matrix in zwei Regionen vom gleichen Typ stets die gleichen PIPs verwendet, wodurch die Verschaltung in beiden Regionen identisch ist.

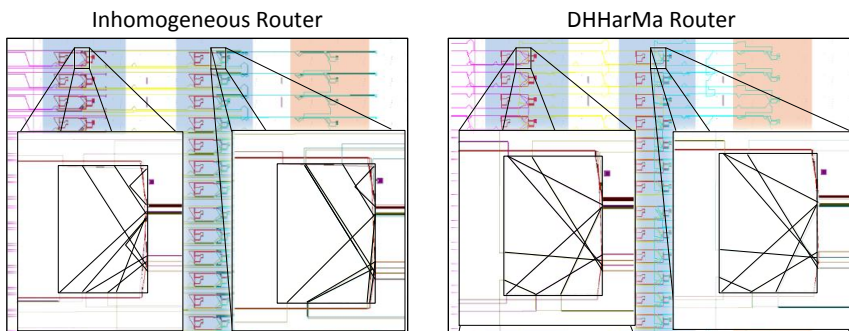


Abbildung 4.21: Gegenüberstellung einer inhomogenen (links) und homogenen (rechts) Verdrahtung [105, S. 6]

Abbildung 4.22 stellt den Entwurfsablauf des homogenen Routers dar. Der Algorithmus kann dabei in fünf Phasen unterteilt werden.

Identifikation von homogenen Inter-Nets und Intra-Nets Ein Netz kann je nach Verbindung einer von zwei möglichen Kategorien angehören: *Inter-Net* oder *Intra-Net*. Zur Kategorisierung werden alle Netze in einzelne Punkt-zu-Punkt Netze umgewandelt. Konkret wird ein Netz mit n Netzeingängen in n separate Verbindungen zu dem Netzausgang aufgeteilt. Die einzelnen Punkt-zu-Punkt Verbindungen werden anschließend kategorisiert: Verbindungen innerhalb der Region erhalten das Attribut *Intra-Net*, Verbindungen zu zwei unterschiedlichen Regionen erhalten das Attribut *Inter-Net*. In jeder

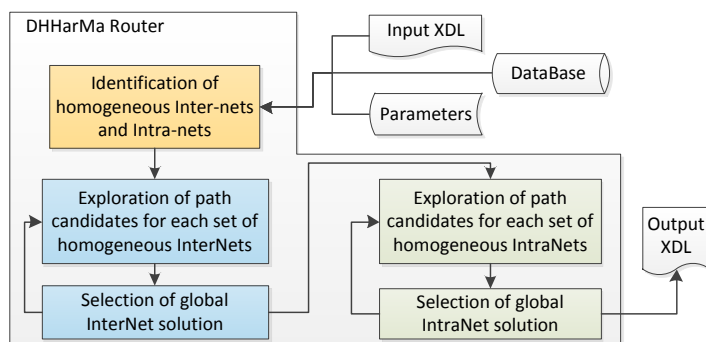


Abbildung 4.22: Entwurfsablauf des homogenen Routers [105, S. 5]

Region des gleichen Typs werden Verbindungen mit identischen Ein- und Ausgängen und identischer Kategorie (Intra-Net oder Inter-Net) zu einer Gruppe zusammengefasst.

Erkundung der Pfade der Gruppen der homogenen Inter-Nets Die Pfade der einzelnen Punkt-zu-Punkt Verbindungen werden über einen Tiefensuche-Algorithmus mittels der physikalischen Leitungen der DXF (siehe Abschnitt 4.5.1) berechnet. Die Tiefe wird dabei durch die Anzahl der verwendeten PIPs zur Realisierung einer physikalischen Verbindung repräsentiert. In dem Fall, dass kein vollständiger Pfad mit der aktuell vorgegebenen Tiefe gefunden werden kann, wird die Tiefe erhöht, das heißt die Anzahl der verwendbaren PIPs inkrementiert. Ein vollständiger Pfad wird anschließend in die Lösungsmenge aufgenommen, wenn er in allen Regionen gleichen Typs existiert.

Ein Beispiel der Vorgehensweise des Algorithmus wird in Abbildung 4.23 dargestellt. Startpunkt ist der Ausgang einer Verbindung aus der homogenen Menge der Inter-Nets

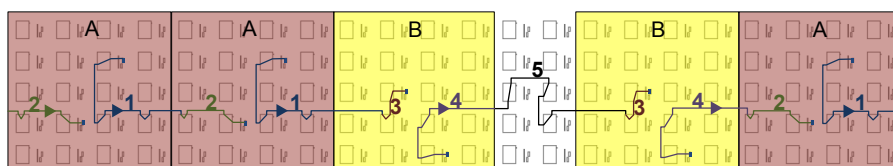


Abbildung 4.23: Vorgehensweise des Routing-Algorithmus[105, S. 6]

in allen Regionen gleichen Typs. Der Pfad wächst in allen Regionen bis eine Grenze zu einer anderen Region erreicht wird (siehe Pfadbeispiele 1 und 4). Die Verbindung ab der Grenze beschreibt einen Pfad, der in Abhängigkeit von dem Regionstyp unterschiedlich gewählt werden kann (siehe Pfad 2 und 3). Einen Sonderfall stellen Verbindungen durch freie oder statische Regionen dar, in der keine Homogenität benötigt wird (siehe

Pfad 5). Der Algorithmus speichert für jeden homogen realisierbaren Pfad die Menge benötigter PIPs und sucht anschließend alternative Lösungen.

Auswahl einer globalen Pfad-Lösung der homogenen Inter-Nets In diesem Abschnitt wird aus jeder Gruppe der homogenen Inter-Nets eine für alle Regionen des gleichen Typs geltende Pfadlösung bestimmt. Primäres Auswahlkriterium ist hierbei die Anzahl benötigter PIPs.

Erkundung der Pfade der Gruppen der homogenen Intra-Nets Der Algorithmus zur Berechnung der Pfade der internen Verbindungen einer Region ähnelt dem der Inter-Nets. Ein wesentlicher Unterschied ist, dass die benachbarten Regionstypen keinen Einfluss auf den Pfad nehmen. Den größten Einfluss auf den Pfad haben die bereits verwendeten Ressourcen der Inter-Nets, welche mögliche Lösungen verhindern.

Auswahl einer globalen Pfad-Lösung der homogenen Intra-Nets Die Auswahl der globalen Lösung eines Pfads für homogene Intra-Nets erfolgt analog zur Auswahl einer globalen Pfad-Lösung der homogenen Inter-Nets.

4.5.6 XDL Writer

Abschließend wird die Datenstruktur des nun homogenen Designs in eine XDL-konforme Datei exportiert. Diese Aufgabe wird von dem *XDL Writer* durchgeführt. Die Abbildung 4.24 visualisiert die einzelnen Abschnitte der XDL-Ausgangsdatei. Die Darstellung ähnelt der Abbildung 4.7 in Abschnitt 4.3.2. An dieser Stelle werden die Unterschiede der Eingangs- und Ausgangsdatei hervorgehoben. Die Ausgangsdatei wird typischerweise im NMC-konformen Format erstellt. Dieses Format repräsentiert ein Hard-Makro, welches einem Design im NCD-Format hinzugefügt werden kann. Zusätzlich bietet der XDL Writer die Möglichkeit, direkt eine NCD-konforme Datei zu erstellen. Die Unterschiede beider Formate sind farblich dargestellt.

Eigenschaften In dem ersten Abschnitt (*Properties*) der Ausgangsdatei werden allgemeine Eigenschaften aufgeführt. In der ersten Zeile wird entweder der Name des Designs angegeben oder die Datei als Hard-Makro durch das Schlüsselwort `__XILINX_NMC_BUS` gekennzeichnet. In beiden Fällen muss zusätzlich der vollständige FPGA-Name angegeben werden. Die zweite Zeile der Datei definiert ein sogenanntes *Modul*, welches ein Makro innerhalb eines Designs definiert und dem Inhalt einer NMC-konformen Datei entspricht. Eine XDL-Design-Datei kann dabei eine Vielzahl von Modulen enthalten.

Ports, Instanzen und Netze Anschließend folgen die bereits in der Abbildung 4.7 in Abschnitt 4.3.2 dargestellten Abschnitte der Ports, Instanzen und Netze. Im Vergleich zu der Eingangsdatei werden die Slice-Konfigurationen sortiert ausgegeben. Die Instanzen werden Cluster für Cluster für jede Region aufgelistet. Die Konfigurationen der einzelnen Parts werden wiederum zeilenweise aufgeführt. Hierdurch wird die Lesbarkeit der Designbeschreibung erhöht, so dass z. B. kleinere Modifikationen an der Datei manuell durchgeführt werden können. Die Netze sind in vier Gruppen unterteilt. In der ersten

```

design "NameTopEntity"|"__XILINX_NMC_BUS" xc7a100tcsq324-3; Properties
module "PathXDLFile" "Base0_[0](Slice_1)", cfg "_SYSTEM_MACRO::FALSE";

# ===== Ports =====
port "CLK_int" "Base0_[11](Slice_2)" "CLK";
...

# ===== 1. Base Region Instances =====
# ===== Instances for Region : Base0 =====
# Group : 0 / 32
inst "Base0_[0](Slice_1)" "SUCEL",placed CLBLM_L_X10Y97 SLICE_X12Y97 ,
cfg "
{Part-A Settings}
{Part-B Settings}
{Part-C Settings}
{Part-D Settings}
...."
# ===== 2. Reconf Region Instances =====
# ===== Instances for Region : Rec0_a =====
# Group : 0 / 33
inst "Rec0_a_[0](Slice_1)" "SUCEL",placed CLBLM_R_X19Y97 SLICE_X29Y97,
...

# ===== Port Nets =====
net "$NET_0",
  outpin "Base0_[11](Slice_2)" "CLK",
...
# ===== Base Region Nets =====
# ===== Nets for Region : Base0 =====
# Nets of instance : Base0_[0](Slice_1) = 8 :
net "Base0/stb_g[0].primitiv_dedi_bin_stb/right_in_reg",
  outpin "Base0_[0](Slice_1)" DMUX,
  inpin "Base0_[0](Slice_8)" B4 ,
  pip CLBLM_L_X10Y97 CLBLM_L_DMUX->CLBLM_LOGIC_OUTS19,
...
# ===== Reconf Region Nets =====
# ===== Nets for Region : Rec0_a =====
# Nets of instance : Rec0_a_[0](Slice_1) = 7 :
...
# ===== Global Nets =====
net "CLK_int",
...

# ===== SUMMARY =====
# SUMMARY
# Number of Primitive Insts: 1039
# Number of LUTs: 3668
# Number of Regs: 2957
# Number of Nets: 4813
# Number of Pips: 36382
# Number of Ports: 2379
# =====
endmodule "pathXDLFile"; Endmodule

```

Abbildung 4.24: Generierte XDL-Ausgangsdatei im NMC-(rot) und NCD-(schwarz) Format (basierend auf [103, S. 97])

Gruppe werden die Netze der Ports, in der zweiten Gruppe die Netze der statischen, in der dritten Gruppe die Netze der rekonfigurierbaren Regionen und in der letzten Gruppe globale definiert. In den ersten drei Gruppen sind die Netze für die jeweilig zugeordnete Instanz ausgegeben. Diese Darstellung wurde gewählt, da die Netze durch

den Ausgang (*outpin*) stets an eine Instanz gebunden sind. In der Abbildung 4.24 sind zum Beispiel acht Netze der Instanz *Base0_[0]_(Slice1)* zugeordnet.

Zusammenfassung Abschließend wird eine Zusammenfassung der verwendeten Logik- und Verdrahtungsressourcen durch das Design ausgegeben. Für die Logikressourcen wird die Anzahl der Instanzen sowie die konkrete Anzahl der LUTs und Register sowie für die Verdrahtungsressourcen die Anzahl der Netze und verwendeten PIPs aufgeführt. Im Falle einer NMC-Datei wird zusätzlich die Anzahl der Ports dargestellt und Moduldefinition abgeschlossen.

Eine Ausgangsdatei kann nach der Generierung mit der Xilinx-Software *xdl* in das entsprechende binäre NMC- oder NCD-Format transformiert werden. Ein Hard-Makro kann anschließend, wie bereits beschrieben, einem Design hinzugefügt werden.

4.6 Auswertung

In diesem Abschnitt erfolgt eine Auswertung der entwickelten Softwarekomponenten. In dem ersten Unterabschnitt werden die, in Abschnitt 4.5.1 vorgestellten, Datenbanken bezüglich des Speicherbedarfs und der Erstellungszeit analysiert. Anschließend werden die Platzierungsalgorithmen (siehe Abschnitt 4.5.4) durch die Kostenfunktion (4.1) verglichen. Als weiteres Bewertungsmaß wird die Ausführungszeit verwendet. In dem dritten und vierten Unterabschnitt erfolgt eine Auswertung der Anwendungsgebiete mit homogenen Designanforderungen bezüglich des Ressourcenbedarfs und der maximal möglichen Taktfrequenz. Software-Werkzeuge für FPGAs haben allgemein die Reduzierung der benötigten Fläche oder Erhöhung des Takts durch Verwendung optimaler Leitungen zum Ziel. Insbesondere Leitungstypen zur Überbrückung längerer Distanzen können aufgrund der Partitionierung in feingranularen DPR-Systemen nicht verwendet werden. In den letzten Unterabschnitten wird daher der Einfluss der Homogenität auf die beiden genannten Bewertungsmaße untersucht. Hierzu wird ein Vergleich von homogenen, durch DHHarMa erzeugten, Hard-Makros und inhomogenen, durch Xilinx Software-Werkzeuge erstellten, Hard-Makros durchgeführt.

4.6.1 FPGA-Datenbanken

Für die Auswertung der in Abschnitt 4.5.1 vorgestellten FPGA-Datenbanken wird der benötigte Speicherplatz (Festplatte und RAM) sowie die benötigte Zeit zur Erstellung betrachtet.

DXF

Die Auswertung der DXF erfolgt zunächst für das jeweils größte FPGA einer jeden Unterfamilie aller unterstützten FPGA-Familien. Somit werden stets die Maxima vorgestellt. Die Auswertungen (außer Phase III für den Virtex-7 2000t FPGA) wurden auf einem Intel Xeon W3565 Prozessor mit einem Systemtakt von 3,2 GHz und einem 24 GB

DDR3-1333 Arbeitsspeicher ausgeführt. Für die Erstellung der physikalischen Leitungen des größten Virtex-7 FPGAs wurde ein System mit mehr Arbeitsspeicher verwendet (Intel E5-1650 v2 mit 3,5 GHz und 128 GB DDR3-1600 RAM). Als Betriebssystem wurde ein 64-Bit Ubuntu in der Version 12.04 verwendet und die DHHarMa-Software mit dem Compiler gcc in Version 4.8.4 kompiliert.

Speicherauswertung

Tabelle 4.8 stellt die Speicherauswertung dar. Anzumerken ist, dass der Name eines Xilinx FPGAs typischerweise einen Hinweis auf die Größe des FPGA (Anzahl an Logic Cells) gibt. Dies gilt jedoch nicht für die Bezeichnung der Zynq-7000 Bauteile. So besteht die *Programmable Logic* (PL), der FPGA-Teil des Z7100, beispielsweise aus einem Kintex-7 FPGA mit etwa 444 k Logic Cells [253].

Für die Auswertung ist jeweils die Größe des XDL-Reports, der (komprimiert) gespeicherten DXF, der in den Arbeitsspeicher geladenen DXF sowie die Größe der in den Arbeitsspeicher geladenen DXF wenn alle physikalischen Leitungen abgebildet werden. Die absoluten Werte werden durch prozentuale Werte im Bezug auf die Größe des XDL-Reports ergänzt.

Tabelle 4.8: Speicherauswertung der DXF-V3 mit Darstellung des größten FPGAs einer jeden Unterfamilie aller unterstützten FPGA-Familien

Fam.	FPGA	XDLRC in GB	Kompr. DXF P II in MB	Kompr. DXF P II in %	Gel. DXF P II in MB	Gel. DXF P II in %	Gel. DXF P III in MB	Gel. DXF P III in %
V4	FX140	8,3	17,1	0,20	781	9,2	4 715	55,5
	SX55	3,6	6,7	0,18	326	8,8	2 113	57,3
	LX200	10,4	8,2	0,08	505	4,7	5 463	51,3
V5	FX200t	9,9	17,4	0,17	881	8,7	5 363	52,9
	SX240t	12,5	14,2	0,11	746	5,8	6 385	49,9
	TX240t	10,5	12,7	0,12	719	6,7	5 634	52,4
	LX330	13,7	9,9	0,07	675	4,8	6 991	49,8
V6	CX240t	8,9	11,5	0,13	702	7,7	5 335	58,5
	HX565t	19,0	15,1	0,08	1 040	5,4	11 117	57,1
	SX475t	18,6	14,6	0,08	999	5,3	10 600	55,7
	LX760	24,0	14,1	0,06	1 070	4,4	13 675	55,6
S6	LX150t	4,1	20,9	0,50	736	17,5	3 165	75,4
A7	350t	14,6	45,3	0,30	2 055	13,8	8 762	58,6
K7	480t	20,1	52,5	0,26	2 529	12,3	12 208	59,3
V7	2000t	77,1	192,1	0,24	10 123	12,8	41 306	52,3
Z7	100	18,5	53,5	0,28	2 568	13,6	11 519	60,8

Der wesentliche Wert für die Speicherauswertung ist die Größe der (komprimiert) gespeicherten DXF, welche typischerweise im ein- bis zweistelligen MB-Bereich liegt. Das größte FPGA (Virtex-7 2000t) sticht hier mit 192,1 MB hervor. Betrachtet man die prozentual dargestellten Werte sind insbesondere das Virtex-6 LX760 und das Spartan-6 LX150t FPGA als auffällige Beispiele zu nennen. Die Datenbank des Virtex-6 LX760 FPGAs erreicht mit 0,06 % das beste Ergebnis in Bezug auf entfernbare Redundanz des XDL-Reports. Die Datenbank des Spartan-6 LX150t FPGA stellt mit 0,50 % die obere Schranke dar. Erkennbar ist weiterhin der Anstieg für die DXF der FPGAs der 7-Serie.

Dies spiegelt sich ebenfalls in den prozentual dargestellten Werten der, in den Arbeitsspeicher, geladenen DXF wieder. Dieser liegt für die FPGAs der 7-Serie bei etwa 13 %, während er für Virtex-4 - Virtex-6 mit Werten zwischen 4,35 % und 9,19 % im einstelligen Bereich liegt. Das Spartan-6 FPGA stellt mit 17,35 % abermals die obere Schranke dar. Werden zusätzlich die Objekte für die physikalischen Leitungen eines FPGAs erstellt, bewegt sich der prozentuale Anteil in einem Bereich zwischen 49,83 und 75,39 %. Im Mittel wird ein Anteil von etwa 57 % erreicht. Wie eingangs erwähnt, musste aufgrund unzureichenden Arbeitsspeichers für die Erstellung der physikalischen Leitungen des Virtex-7 2000t FPGAs auf eine andere Plattform gewechselt werden. Das Virtex-7 2000t FPGAs benötigt eine Arbeitsspeicherkapazität in Höhe von 40,3 GB.

Ein Vergleich mit der DXF-V2 (siehe Tabelle A.6 im Anhang B.2) zeigt den Einfluss der Hinzunahme der eigentlich irrelevanten Leitungs- und Verbindungsdefinitionen auf den Speicherbedarf, welcher in einem Faktor zwischen 3,1 und 5,0 resultiert.

Der Vorteil der Verwendung der DXF-V3 mit zusätzlich erstellten physikalischen Leitungen gegenüber einer direkten Verwendung der XDL-Reports, rein aus Speicherplatzgründen, ist daher durch die benötigte Kompatibilität zur *FPGA Editor* Datenrepräsentation nur noch gering. Die komprimiert abgespeicherte Datenbank ist jedoch deutlich kleiner und der Zugriff auf benötigte Daten durch die DXF um mehrere Größenordnungen schneller.

Zeitauswertung

Die Tabelle 4.9 stellt die benötigte Zeit für die Phasen zur Erstellung der DXF-V3 (Phase I und II), sowie die Zeit zur Erstellung der Objekte zur Repräsentation der physikalischen Leitungen eines FPGAs (Phase III) dar.

Die Gesamtzeit für die Erstellung der Datenbank ist separat in einer Spalte (Σ PI-II) dargestellt und verdeutlicht die Ausmaße des größten Virtex-7 FPGAs. Während die Erstellung der Datenbank in der Regel in wenigen Stunden erfolgt, benötigt diese für das Virtex-7 FPGA nahezu fünf Tage. Betont werden muss, dass diese Zeit jedoch nur einmalig investiert werden muss bzw. musste.

Wie bereits erwähnt, wird die Phase III nur ausgeführt, wenn die physikalischen Leitungen als einzelne Objekte benötigt werden (siehe Router, Abschnitt 4.5.5). Wie dargestellt, ist diese Phase für die größten FPGAs der unterstützten FPGA-Familien innerhalb eines niedrigen dreistelligen Sekundenbereichs abgeschlossen.

Tabelle 4.9: Zeitauswertung der DXF-V3 mit Darstellung des größten FPGAs einer jeden Unterfamilie aller unterstützten FPGA-Familien

Fam.	FPGA	Phase I in mm:ss	Phase II in hh:mm:ss	Σ PI-II in hh:mm:ss	Phase III in mm:ss
V4	FX140	06:35	03:01:47	03:08:22	01:20
	SX55	02:55	01:00:24	01:03:19	00:36
	LX200	08:12	03:19:02	03:27:14	01:39
V5	FX200t	08:05	00:43:50	00:51:55	01:32
	SX240t	09:55	00:56:08	01:06:03	01:55
	TX240t	08:33	00:47:07	00:55:40	01:42
	LX330	10:59	00:56:35	01:07:34	02:05
V6	CX240t	07:53	00:38:57	00:46:50	01:17
	HX565t	13:41	01:20:56	01:34:37	02:50
	SX475t	15:06	01:30:00	01:45:06	02:43
	LX760	17:49	01:59:13	02:17:02	03:29
S6	LX150t	03:35	00:36:51	00:40:26	00:46
A7	350t	10:08	02:22:05	02:32:13	02:11
K7	480t	14:34	04:02:15	04:16:49	03:01
V7	2000t	56:31	118:56:43	119:53:14	02:16
Z7	100	13:37	03:26:24	03:40:01	02:43

Zusammenfassende Speicher- und Zeitauswertung

Tabelle 4.10 stellt eine zusammenfassende Speicher- und Zeitauswertung für die unterstützten FPGA-Familien dar. In der letzten Zeile erfolgt eine Summierung bzw. Mittelung (mit * gekennzeichnet) der dargestellten Werte.

Speicherauswertung Für die Speicherauswertung ist der Speicherbedarf aller XDL-Reports und der erstellten Datenbanken aufgeführt. Zusätzlich ist die Relation zwischen beiden Summen prozentual abgebildet, welche zwischen 0,11 % und 0,69 % variiert. In der Summe werden für die XDL-Reports aller 233 unterstützter FPGAs über 2,24 TB an Speicher benötigt. Alle Datenbanken belegen einen Speicherplatz von 6,32 GB. Als Mittelwert für die Relation zwischen beiden Formaten kann ein Wert von 0,28 % festgehalten werden.

Zeitauswertung Die Zeitauswertung erfolgt anhand der ersten beiden Phasen zur Erstellung und Speicherung der Datenbank. Während sich die Datenbanken einer Familie in den meisten Fällen innerhalb eines Tages erstellen lassen, wird für die sequentielle Erstellung der Virtex-7-Familie mehr als ein Monat (ca. 38 Tage) benötigt. Eine sequentielle Erstellung aller FPGA-Datenbanken benötigt auf der beschriebenen

Plattform ca. 44 Tage. Durch Verwendung mehrerer CPUs/Plattformen kann diese Zeit reduziert werden. Weiterhin ist der Durchschnitt für diesen Vorgang für einen FPGA innerhalb der Familie dargestellt, welcher zwischen 16 Minuten und 28 Stunden variiert.

Tabelle 4.10: Zusammenfassung der Speicher- und Zeitauswertung aller unterstützter FPGA-Familien für die DXF-V3

Fam.	Σ FPGA #	Speicher			Zeit	
		Σ XDLRC in GB	Σ DXF in MB	Relation in %	Σ PI-II in hh:mm:ss	\emptyset in hh:mm:ss
V4	30	98	241	0,24	20:00:27	00:40:01
V5	41	200	389	0,19	17:08:12	00:25:05
V6	37	368	429	0,11	32:02:46	00:51:58
S6	42	83	583	0,69	10:58:21	00:15:40
A7	20	104	425	0,40	14:08:41	00:42:26
K7	18	235	705	0,29	45:50:58	02:32:50
V7	32	1 097	3 325	0,30	896:53:58	28:01:41
Z7	13	106	375	0,35	15:58:30	01:13:44
Σ/\emptyset^*	233	2 291	6 472	0,28*	1053:01:53	04:31:10*

FPGA Editor Datenbank

Die Auswertung der *FPGA Editor* Datenbank erfolgt für Referenz-FPGAs der unterstützten FPGA-Familien, welche alle existierenden Kacheltypen innerhalb einer FPGA-Familie enthalten. Mit Hilfe dieser FPGAs können die benötigten Informationen für die weiteren FPGAs der FPGA-Familien erzeugt werden. Die Auswertungen wurden auf einem Intel i7-2600 Prozessor mit einem Systemtakt von 3,2 GHz und einem 24 GB DDR3-1333 Arbeitsspeicher ausgeführt.

Speicherauswertung

In Tabelle 4.11 wird für jedes Referenz-FPGA die Größe beider Logdateien und Datenbanken jeweils entpackt und komprimiert dargestellt. Innerhalb einer FPGA-Familie ist der lineare Anstieg in allen Dateitypen erkennbar. Der Speicheranstieg entspricht der Zunahme der Leitungen (siehe Tabelle 4.12). Übergreifend wird durch die Komprimierung der Dateitypen im Durchschnitt nur etwa 6 % des Speicherplatzes benötigt. Die absoluten Dateigrößen der komprimiert abgespeicherten Dateien liegen je nach FPGA-Größe im einstelligen bis niedrigen dreistelligen MB-Bereich. Die C++-basierte Datenbank belegt dabei 52,5 % weniger Speicherplatz als die textuelle Datenbank. Analog zur DXF, ist der wesentliche Wert für die Speicherauswertung die Größe der (komprimiert) gespeicherten Datenbank, welche konkret zwischen 3,3 MB und 133,0 MB liegt.

Tabelle 4.11: Speicherauswertung der *FPGA Editor* Datenbank für verschiedene FPGAs unterschiedlicher FPGA-Familien

Fam.	FPGA	Simple Log		Advanced Log		Text. Datenbank		Datenbank	
		entpackt in MB	kompr. in MB	entpackt in MB	kompr. in MB	entpackt in MB	kompr. in MB	entpackt in MB	kompr. in MB
V4	FX12	36,4	2,3	89,9	5,3	193,2	15,3	99	5,7
	FX40	118,3	7,2	288,0	18,2	621,3	38,6	305	18,3
V5	LX20t	41,3	2,3	106,2	6,0	227,5	14,0	128	6,7
	FX70t	165,2	8,9	415,3	25,2	886,8	52,9	473	26,4
V6	CX130t	284,6	15,1	722,6	43,2	1 573,2	92,1	818	45,7
	LX9	20,4	1,2	52,3	3,4	112,8	7,6	47	3,3
S6	LX16	29,7	1,7	75,5	4,8	162,3	10,6	115	4,7
	LX25t	46,5	2,7	118,1	7,5	253,2	16,6	74	7,5
A7	8t	41,5	2,4	106,4	5,9	227,0	14,7	112	7,2
K7	70t	163,2	8,7	411,3	24,2	878,0	54,5	415	27,6
	355t	822,1	42,8	2 034,7	112,2	4 439,4	254,8	2 218	133,0
V7	330t	781,4	40,6	1 933,4	107,8	4 221,4	242,1	2 051	128,2
Z7	10	67,1	3,9	171,4	9,6	367,7	23,4	168	11,3

Zeitauswertung

In der Tabelle 4.12 sind, analog zur DXF-Auswertung, die Zeit pro Phase sowie die kumulierte Zeit aller Phasen dargestellt. Zu beachten sind die unterschiedlichen Zeitbereiche in den einzelnen dargestellten Phasen. *Phase I* wird in Stunden und Minuten, die weiteren Phasen in Minuten und Sekunden festgehalten. Zusätzlich ist die Anzahl der Leitungen in der dritten Spalte dokumentiert. Hierdurch kann die Ausführungszeit der Phasen auf eine Leitung normiert werden. Der Unterschied zwischen der *Phase I* der Extraktion aus der *FPGA Editor* GUI und der *Phase II* unter Verwendung der Kommandozeilenversion des *FPGA Editors* kann somit verdeutlicht werden. So werden in *Phase I* pro Leitung im Mittel 75 ms und in *Phase II* weniger als 0,5 ms benötigt.

Die daher insbesondere von der *Phase I* geprägten Ausführungszeiten summieren sich zu einem Absolutwert, welcher sich im Rahmen von wenigen Stunden bis hin zu mehreren Tagen bewegt. Die obere Grenze stellt für die *FPGA Editor* Datenbank das Kintex-7 355t FPGA mit einer Gesamtausführungszeit von ca. 220 Stunden oder 9 Tagen dar.

Tabelle 4.12: Zeitauswertung der *FPGA Editor* Datenbank für verschiedene FPGAs unterschiedlicher FPGA-Familien

Fam.	FPGA	Leitungen in #	Phase I in hh:mm	Phase II in mm:ss	Phase III in mm:ss	Phase IV in mm:ss	SUM PI-IV in hh:mm:ss
V4	FX12	627 644	9:50	11:18	23:19	1:56	10:27:11
	FX40	1 985 232	31:06	9:18	34:18	4:55	31:54:32
V5	LX20t	711 807	11:12	2:28	24:53	1:41	11:41:12
	FX70t	2 745 277	41:23	13:35	41:11	8:41	42:27:43
V6	CX130t	4 613 058	75:34	25:54	50:13	15:32	77:06:18
	LX9	352 431	4:56	3:29	21:47	1:04	5:23:06
S6	LX16	509 301	5:59	6:14	23:32	1:33	6:30:07
	LX25t	793 011	12:59	11:15	27:03	2:10	13:40:14
A7	8t	702 554	11:36	2:37	25:22	5:58	11:45:12
K7	70t	2 677 838	44:03	10:50	22:01	5:10	44:19:45
	355t	13 132 573	217:32	93:18	95:25	40:06	219:46:40
V7	330t	12 488 818	188:11	91:27	90:42	40:24	190:23:11
Z7	10	1 129 281	18:39	5:45	21:57	3:16	18:48:19

4.6.2 Platzierungsalgorithmen

Für die Auswertung der, in Abschnitt 4.5.4 vorgestellten, Platzierungsalgorithmen wird primär die Kostenfunktion (4.1), jedoch auch die Ausführungszeit verwendet. Durch das Cluster-basierte BinPack ergeben sich vier Platzierungsstrategien:

1. Cluster-basierte Platzierung (C)
2. Cluster-basierte BinPack Platzierung (C-B)
3. Simulated Annealing mit zufallsbasierter Anfangsplatzierung (SA-ZA)
4. Simulated Annealing mit Cluster-basierter Anfangsplatzierung und geringer Anfangstemperatur (SA-CA)

Platzierungsauswertung

Die Analyse dieser vier Platzierungsstrategien wurde anhand der zweidimensionalen Datenbuskommunikationsinfrastrukturen durchgeführt. Weiterhin wurden die vier Strategien für die Virtex-4-Architektur (LX15) und die 7-Serie-Architektur (A100T) angewendet, um die Auswirkung auf unterschiedliche FPGA-Architekturen und variierende Partitionierungen zu untersuchen. Die Auswertung der Platzierungsstrategien mit Verwendung des Simulated Annealing Algorithmus basiert auf jeweils zehn Durchläufen. In der Auswertung wird primär der Durchschnitt jedoch auch das Minimum und das Maximum betrachtet.

Virtex-4-Architektur

Die Abbildung 4.25 stellt die Auswertung für die Virtex-4-Architektur dar. Aufgrund der deutlich höheren Kosten (Kostenfunktion (4.1)) der 64-Bit Varianten ist der Vergleich der 8-Bit Varianten in dieser Darstellung nicht möglich. Für die Betrachtung dieser Varianten wurde daher ein separates Diagramm erstellt (siehe Abbildung A.5 im Anhang B.3). Die konkreten Werte sind ebenfalls tabellarisch in dem Anhang aufgeführt (Tabelle A.20).

Die Auswertung der Virtex-4-Architektur zeigt, dass die Platzierungsstrategie SA-ZA im Durchschnitt die höchsten Kosten verursacht und somit bezüglich der Kostenfunktion (4.1) die schlechteste Platzierung realisiert. Ebenfalls sind hohe Abweichungen der Kosten aufgrund der initialen Zufallsplatzierung erkennbar. In lediglich einem Design (2x2, 8) erzielte diese Platzierungsstrategie im Vergleich zu den Cluster-basierten Verfahren (C und C-B) ein (marginal) besseres Ergebnis. Im Vergleich zu dem jeweils bestem Ergebnis der Cluster-basierten Verfahren sind die Kosten im Durchschnitt um 31,3 % höher.

Weiterhin ist erkennbar, dass in den 64-Bit Varianten die Platzierungsstrategie C-B nicht möglich ist. Dies liegt zum einen an dem hohen, internen Verschnitt der Formen für die einzelnen Cluster und wird zum anderen durch die gewählte Partitionierung mit einer insgesamt geringen Anzahl an verfügbaren Ressourcen verstärkt. Für kleinere Designs oder wenn die Partitionierung eine hohe Ressourcenanzahl zur Verfügung stellt, erstellt die C-B Platzierung jedoch das beste Platzierungsergebnis.

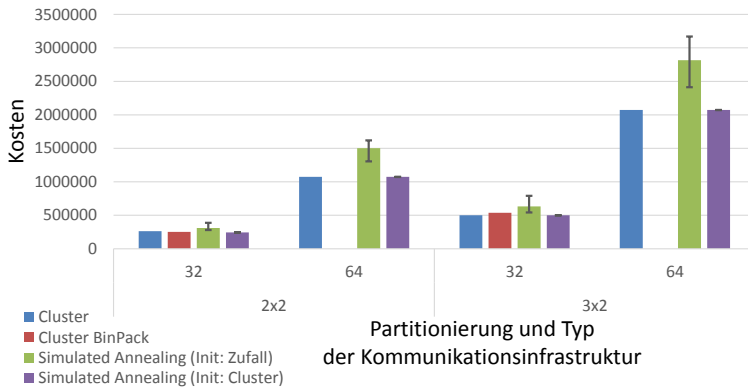


Abbildung 4.25: Auswertung der Platzierungsstrategien für verschiedene Partitionierungen in der Virtex-4-Architektur

Aus diesem Grund wird für die initiale Platzierung der vierten Platzierungsstrategie jeweils beide Cluster-basierten Platzierungen durchgeführt und die bessere Platzierung verwendet. Diese vierte Platzierungsstrategie SA-CA profitiert von dieser guten Eingangsplatzierung und kann das Platzierungsergebnis in drei von sechs Fällen verbessern. Im Durchschnitt liegt die Kosteneinsparung bei 2,2 %, im besten Fall bei bis zu 10,0 % (Design 2x2, 8). Zusammenfassend führt die Platzierungsstrategie SA-CA in allen Designs in der Virtex-4-Architektur zu dem besten Platzierungsergebnis.

Artix-7-Architektur

Die Auswertung der Artix-7-Architektur ist in Abbildung 4.26 dargestellt und führt zum Teil zu anderen Ergebnissen. Die Auswertung der 8-Bit Varianten ist abermals in einem separaten Diagramm (siehe Abbildung A.6 und die konkreten Werte in Tabelle A.21 im Anhang B.3) aufgeführt.

Im Vergleich zu den Cluster-basierten Verfahren führt die Platzierungsstrategie SA-ZA (Durchschnitt) in drei von sechs Designs zu besseren Ergebnissen. Dies liegt primär an den größeren Regionen die durch die Höhe der Taktregionen im Vergleich zu der Virtex-4-Architektur deutlich größer dimensioniert sind (50 CLBs zu 16 CLBs, siehe Abschnitt 3.3.1). Im Durchschnitt sind die ermittelten Kosten um 1,3 % geringer. Es ist weiterhin erkennbar, dass die Platzierungsstrategie bei den mittleren und großen Designs vorteilhaft ist. Die maximale Kosteneinsparung ist in dem Design 2x2, 64 mit 12,9 % erkennbar. Bei zusätzlicher Betrachtung der Kostenminima, sind in dem Design 2x2, 32 bis zu 21,2 % geringere Kosten möglich.

In einem Vergleich aller Platzierungsstrategien, führt die vierte Platzierungsstrategie SA-CA abermals im Durchschnitt zu dem besten Platzierungsergebnis. Die Kosten sind in Durchschnitt um 7,5 % und im besten Fall (Design 2x2, 8) um bis zu 21,2 % geringer.

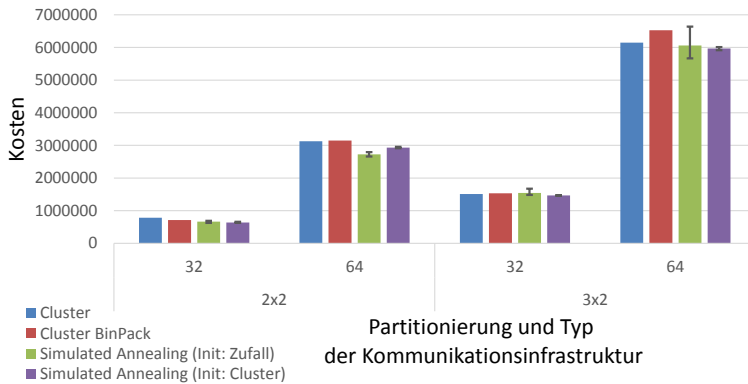


Abbildung 4.26: Auswertung der Platzierungsstrategien für verschiedene Partitionierungen in der Artix-7-Architektur

Ein Gegenbeispiel stellt das Design 2×2 , 64 dar, in welchem die Verwendung des klassischen Simulated Annealing Verfahrens (SA-ZA) zu dem besten Ergebnis führt.

Auswertung der Ausführungszeit

Die Ausführungszeiten in beiden FPGA-Architekturen variieren für die einzelnen Platzierungsstrategien stark. Die beiden Cluster-basierten Verfahren und die Zufallsplatzierung sind für alle Designs in weniger als einer Sekunde durchgeführt. Bei der klassischen Variante des Simulated Annealing Verfahrens variiert die Ausführungszeit zwischen wenigen Minuten und mehreren Stunden. Als obere Grenze für die Ausführungszeit konnte das Design 3×2 , 64 mit ca. 190 Minuten festgehalten werden. Aufgrund der niedrigeren Anfangstemperatur der Simulierten Abkühlung mit Cluster-basierter Anfangsplatzierung ist die Ausführungszeit im Vergleich deutlich geringer und liegt maximal bei ca. 13 Minuten bei dem gleichen Design.

Zusammenfassung

Zusammenfassend kann festgehalten werden, dass für das Anwendungsgebiet der Kommunikationsinfrastrukturen die Simulated Annealing mit Cluster-basierter Anfangsplatzierung im Durchschnitt die besten Platzierungsergebnisse bei einer vergleichsweise geringen Ausführungszeit erzielt. Aufgrund dieser Ergebnisse wird daher diese Platzierungsstrategie für die Platzierung der Komponenten der Kommunikationsinfrastrukturen für DPR-Systeme verwendet.

4.6.3 DPR-Kommunikationsinfrastrukturen

Für die Bewertung der DHHarMa Werkzeuge wurden die, in Abschnitt 4.4 vorgestellten, DPR-Kommunikationsinfrastrukturen mit DHHarMa und Xilinx Werkzeuge erstellt und anschließend gegenübergestellt. Die letztgenannten Kommunikationsinfrastrukturen sind aufgrund der Inhomogenität nicht für ein INDRA 2.0 DPR-System verwendbar. Die Analyse erfolgt für die FPGA-Ressourcen und die Leistungsfähigkeit sowie separat für die eindimensionalen und zweidimensionalen Kommunikationsinfrastrukturen. Um den Einfluss der FPGA-Architektur bewerten zu können, wird die Auswertung für die Virtex-4-, Virtex-6- und 7-Serie-Architektur durchgeführt.

Bewertungsmaße zur Auswertung der FPGA-Ressourcen

Bei den FPGA-Ressourcen ist die wesentliche Kenngröße die Anzahl der verwendeten Slices. Diese beschreibt zum einen die allgemeine Designgröße und zum anderen, durch die Anzahl der Register und LUTs des Designs, die (Slice-) Packungsdichte. Hierbei existieren unterschiedliche (Ober-) Grenzen für die Packungsdichte.

Begrenzung durch die Slice-Architektur Die obere Grenze wird durch die Slice-Architektur definiert (siehe Abschnitt 3.2.6). Für die Virtex-4-Architektur liegt diese Grenze daher bei zwei LUTs und zwei Registern pro Slice, für die Virtex-6- und 7-Serie-Architektur bei 8 LUTs und 8 Registern. Diese maximale Packungsdichte ist nur bei einer vollständigen Technologieabbildung des Designs erreichbar, da nur in diesem Fall beide LUTs des Funktionsgenerators und beide Register der Slice verwendet werden können.

Begrenzung durch die Packer-Werkzeuge Eine weitere Begrenzung für die Packungsdichte kann durch die Packer-Werkzeuge gegeben sein. So ist, wie in Abschnitt 4.5.3 beschrieben, die maximale Packungsdichte des DHHarMa Packers, unabhängig von der Slice-Architektur, auf eine LUT und ein Register pro Slice-Part beschränkt. In den Xilinx Werkzeugen ist die Packungsdichte nur begrenzt einstellbar und teilweise auf bestimmte FPGA-Familien oder durch Einstellungskombinationen beschränkt. Weiterhin sind die meisten Optimierungen erst durch Aktivierung der erweiterten Einstellungen einseh- und editierbar und in der Standardeinstellung deaktiviert. Beispiele für MAP-Einstellungen sind:

- *CLB Pack Factor Percentage* (bis Virtex-4 und nur in Kombination mit deaktivierter Einstellung *Perform Timing-Driven Packing and Placement*): Bestimmung der maximalen CLB-Packungsdichte in Prozent (Standard: 100 %)
- *Optimization Strategy* (bis Virtex-5): Einstellung der *cover* Phase, in der MAP die Logik auf die Funktionsgeneratoren abbildet (Auswahlmöglichkeiten: Area, Speed, Balanced und Off, Standard: Area)
- *Global Optimization* (Virtex-4 bis Virtex-6 und nur in Kombination mit aktivierter Einstellung *Trim Unconnected Signals*): Aktiviert globale Optimierungen an der Netzliste (z. B. entfernen oder replizieren von Logikblöcken und Registern) vor

der Packung (Auswahlmöglichkeiten: Speed, Area (außer Virtex-4), Power (außer Virtex-4) oder Off, Standard: Off)

- *Maximum Compression* (ab Virtex-5): Möglichst dichte Packung des Designs (Checkbox, Standard: Aus)
- *LUT Combining* (ab Virtex-5): Versucht LUT-Paare in LUT6-Komponenten mit zwei Ausgängen zu packen (Auswahlmöglichkeiten: Auto, Area und Off, Standard: Off)
- *Register Ordering* (ab Virtex-6): Bestimmung der Anzahl der Register pro Slice in einer Buskommunikationsinfrastruktur (Auswahlmöglichkeiten: 4 (8 wenn Quelle LUT), 8 oder Off, Standard: 4 (8))

Die Virtex-4-Implementierungen sind bereits durch die Standardeinstellungen *Optimization Strategy* und *CLB Pack Factor Percentage* auf eine Flächenoptimierung eingestellt. Die *MAP*-Einstellung *Global Optimization* kann aufgrund der bedingten Kombination mit der Einstellung *Trim Unconnected Signals* nicht verwendet werden, da diese unverbundene Netze der Kommunikationsinfrastruktur entfernen würde. Für die Implementierungen der FPGAs der Virtex-6- und 7-Serie-Familie werden die Ergebnisse der Standardeinstellungen und der zusätzlichen Verwendung der *MAP*-Einstellungen *Maximum Compression* und *LUT Combining* (Area) dem Vergleich hinzugefügt.

Die Slice-Auslastung der PR-Region durch die Kommunikationsinfrastruktur ist das dritte Bewertungsmaß für den Vergleich der Packer, da hiermit die Anzahl verbleibender FPGA-Ressourcen für eine Anwendung ermittelt werden kann.

Bewertungsmaße zur Auswertung der Leistungsfähigkeit

Die Verbindung zweier Verdrahtungsleitungen erfolgt durch die Aktivierung eines oder mehrerer PIPs (engl. bounce) in einer Switch-Matrix. In der Auswertung wird daher der Zusammenhang zwischen der Anzahl der PIPs zur Realisierung der Netze und der resultierenden Taktfrequenz untersucht.

Im Falle der eindimensionalen Kommunikationsinfrastrukturen werden vollständig durch ISE erstellte Designs den mit DHHarMa gepackt, platziert und verdrahteten Designs gegenübergestellt. Um die Ergebnisse der Verdrahtungswerkzeuge miteinander vergleichen zu können, werden für die zweidimensionalen Kommunikationsinfrastrukturen die homogenen, durch DHHarMa erstellten, Hard-Makros verwendet. Die Anzahl der PIPs und die resultierende Taktfrequenz ist somit für die zweidimensionalen Kommunikationsinfrastrukturen auf das gleiche Eingangsdesign bezogen.

Die Designs für die Datenübertragung von gemeinsam verwendeten Signalen (*DB*) unterscheiden sich lediglich in der (Bit-) Breite der Datenleitungen: 8 bit, 32 bit und 64 bit. In den folgenden Diagrammen sind für die Repräsentation dieses Kommunikationsinfrastrukturtyps jeweils die Werte der breitesten Variante abgebildet. Die dargestellten Mittelwerte enthalten jedoch stets die Werte aller Designs.

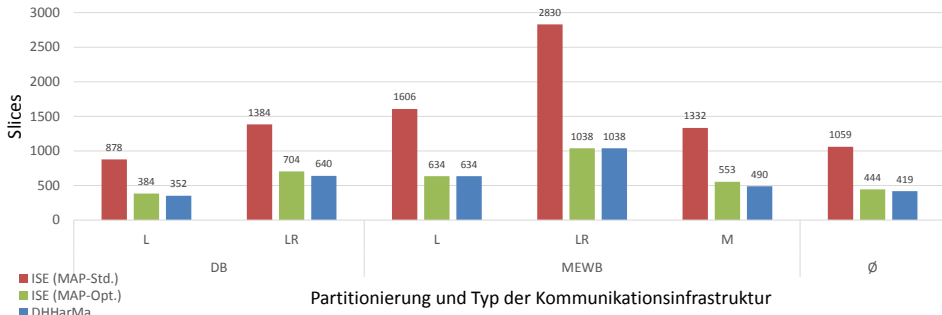


Abbildung 4.27: Gegenüberstellung der Slice-Anzahl für homogene und inhomogene 1D DPR-Kommunikationsinfrastrukturen

Eindimensionale Kommunikationsinfrastrukturen

Für die eindimensionalen Partitionierungen werden die DB- und MEWB-Kommunikationsinfrastrukturen auf die Virtex-6-Architektur abgebildet. Konkret wird der LX75T mit 11 640 Slices und somit maximal 93 120 FFs und LUTs bereitstellt. Durch die vorgestellten *MAP*-Einstellungen ergeben sich somit drei Vergleiche: *MAP* mit Standardeinstellungen (*MAP-Std.*), *MAP* mit optimierten Einstellungen (*MAP-Opt.*) und DHHarMa Packer mit Standardeinstellungen (DHHarMa).

FPGA-Ressourcen

Die eingeführten Bewertungsmaße werden in diesem Abschnitt durch einzelne Diagramme hervorgehoben. Die zugehörigen Daten können in der Tabelle 4.13 im Anhang B.5 nachgeschlagen werden.

Slice-Anzahl In Abbildung 4.27 sind die Anzahl der Slices für die drei Packer dargestellt. Der Vergleich beide Xilinx-Ergebnissen zeigt, dass durch die Verwendung der beiden flächenoptimierten *MAP*-Einstellungen zwischen 49,1 % und 63,3 % an Slices eingespart werden können. Die Ergebnisse der flächenoptimierten *MAP*-Einstellungen und DHHarMa variieren zwischen 0,0 % und 11,4 % zu Gunsten der DHHarMa-Packung. Im Durchschnitt benötigen die homogenen DHHarMa-Implementierungen im Vergleich 7,1 % weniger Slices. Durch diese Abweichungen ergeben sich entsprechende, unterschiedliche Gesamtauslastungen des FPGAs. Für die beiden inhomogenen ISE-Implementierungen werden bei der Verwendung der *MAP*-Standardeinstellungen zwischen 1,0 % und 24,3 % und bei Verwendung der optimierten Einstellungen 0,4 % und 8,9 % der verfügbaren Slices verwendet. Die Ergebnisse der homogenen DHHarMa-Implementierungen stimmen mit dem letztgenannten Bereich überein. Im Durchschnitt werden mit 3,60 % zu 3,81 % insgesamt jedoch weniger Slices in den DHHarMa-Implementierungen verwendet.

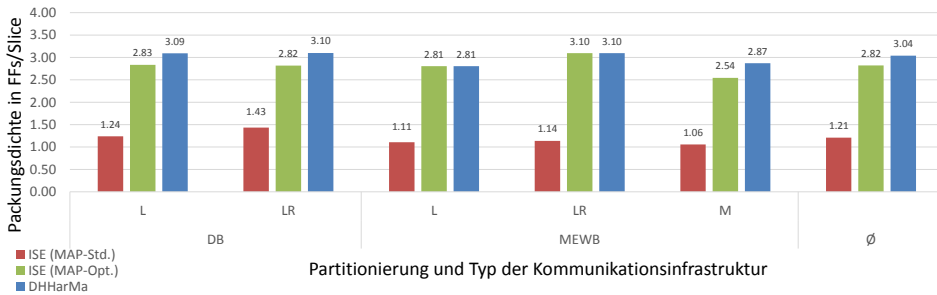


Abbildung 4.28: Gegenüberstellung der FF-Packungsdichte für homogene und inhomogene 1D DPR-Kommunikationsinfrastrukturen

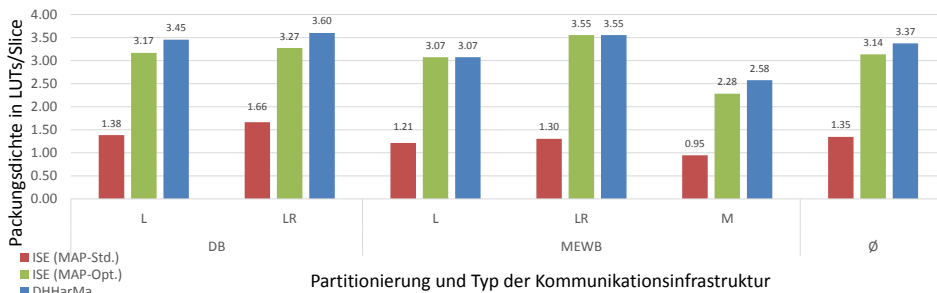


Abbildung 4.29: Gegenüberstellung der LUT-Packungsdichte für homogene und inhomogene 1D DPR-Kommunikationsinfrastrukturen

Packungsdichte Die Abweichungen in der Anzahl an Slices führt zu deutlich unterschiedlichen Packungsdichten der Slices. In Abbildung 4.28 ist die Packungsdichte im Bezug auf die Register dargestellt. Die MAP-Std. Ergebnisse ergeben Packungsdichten von 1,06 Registern–1,44 Registern mit einem Mittelwert von 1,21 Registern. Die deutlich geringe Slice-Anzahl in den MAP-Opt. Implementierungen führt zu Packungsdichten von 2,54 Registern–3,10 Registern und einem Durchschnittswert von 2,82 Registern. Die Packungsdichten der homogenen Hard-Makros des DHHarMa Packers variieren von 2,81 Registern–3,10 Registern und liegen im Mittel mit 3,04 Registern etwas höher als die MAP-Opt. Implementierungen.

Die Analyse der LUT-Packungsdichte führt zu ähnlichen Ergebnissen (siehe Abbildung 4.29). Die homogenen Hard-Makros weisen eine Packungsdichte von 3,58 LUTs–3,60 LUTs auf. Im Durchschnitt ergibt sich ein Wert von 3,37 LUTs. Die Mittelwerte der MAP-Implementierungen liegen bei 1,35 LUTs (MAP-Std.) bzw. 3,14 LUTs (MAP-Opt.).

Auslastung PR-Region Alle PR-Regionstypen besitzen in allen eindimensionalen Designs die gleiche Anzahl an Slice-Ressourcen. Konkret werden pro PR-Region (PRR) 320 Slices mit jeweils 2 560 LUTs und Registern bereitgestellt. In Tabelle 4.13 ist Slice-Auslastung der PR-Region durch die Kommunikationsinfrastruktur (absolut und relativ) dargestellt. Zusätzlich ist die resultierende Anzahl der freien Slices dokumentiert.

Tabelle 4.13: Auslastung der PR-Region für homogene und inhomogene, 1D DPR-Kommunikationsinfrastrukturen in der Virtex-6-Architektur (LX75T)

Typ	Design		Inhom. ISE Komm.-Makros						Hom. DHHarMa Komm.-Makros		
	Konfiguration Part.	Busbr.	Std. MAP-Einstellungen		Opt. MAP-Einstellungen		Std. Einstellungen		Slices/PRR		Freie Slices
			Slices/PRR in #	in %	Freie Slices in #	Slices/PRR in #	in %	Freie Slices in #	in #	in %	in #
DB	L	8	21	6,6	299	8	2,5	312	8	2,5	312
		32	88	27,4	232	32	10,0	288	32	10,0	288
		64	161	50,2	159	64	20,0	256	64	20,0	256
	LR	8	20	6,1	300	8	2,5	312	8	2,5	312
		32	70	22,0	250	32	10,0	288	32	10,0	288
		64	140	43,6	180	64	20,0	256	64	20,0	256
MEWB	L	115	297	92,7	24	101	31,6	219	101	31,6	219
	LR	115	281	87,8	39	101	31,6	219	101	31,6	219
	M	115	299	93,4	21	104	32,5	216	101	31,6	219
∅			153	47,8	167	57	17,8	263	57	17,7	263

Die MAP-Std. Implementierungen führen zu einer prozentualen Auslastung der PR-Region zwischen 6,1 % und 93,4 %. Bei der Verwendung einer MEWB-Kommunikationsinfrastruktur bleibt somit für die Implementierung einer Anwendung nur noch eine geringe zweistellige Anzahl an Slices. Die MAP-Opt. und DHHarMa Implementierungen ergeben nahezu identische PRR-Auslastungen mit einem Wertebereich zwischen 2,5 % und 32,5 %. Der Großteil der FPGA-Ressourcen ist somit für die Anwendung verfügbar.

Leistungsfähigkeit

Zur Auswertung der Leistungsfähigkeit wird in Tabelle 4.14 die Anzahl der PIPs und die Taktfrequenz für die mit ISE und DHHarMa erstellten Designs.

PIPs Mit Ausnahme des Designs *LR 8* ist die Anzahl der PIPs in den DHHarMa-Implementierungen geringer als in den PAR-Implementierungen. Ein relativer Vergleich spannt einen Wertebereich von -24,5 % und 1,8 % auf. Im Durchschnitt werden in DHHarMa 7,2 % weniger PIPs für die Verbindungen der Netze verwendet.

Taktfrequenz Der Vergleich der Taktfrequenz verdeutlicht, dass die Anzahl der PIPs kein eindeutiges Indiz für die Taktfrequenz ist. Die Taktfrequenz der inhomogenen PAR-Implementierungen ist, mit Ausnahme des Designs *MEWB L*, höher als bei den homogen verdrahteten DHHarMa-Implementierungen. Dies wird primär durch die Verwendung von optimalen Verdrahtungsressourcen ermöglicht, welche z. B. die Partitionierungsgrenzen überschreiten dürfen. Insgesamt führt der Vergleich der Taktfrequenz zu einem

Tabelle 4.14: Auswertung der Leistungsfähigkeit für homogene und inhomogene, 1D DPR-Kommunikationsinfrastrukturen in der Virtex-6-Architektur (LX75T)

Typ	Design		Inhom. Komm.-Makro		Hom. Komm.-Makro		Vergleich	
	Konfiguration Part.	Busbr.	PIPs in #	Taktfreq. in MHz	PIPs in #	Taktfreq. in MHz	PIPs in %	Taktfreq. in %
DB	L	8	1 155	294	1 088	257	-5,8	-12,5
		32	4 679	240	4 278	230	-8,6	-3,9
		64	9 566	229	9 000	214	-5,9	-6,5
	LR	8	2 014	113	2 050	108	1,8	-4,1
		32	8 464	113	8 128	108	-4,0	-4,5
		64	16 913	107	16 399	105	-3,0	-1,6
MEWB	L	115	21 629	161	16 337	213	-24,5	32,2
	LR	115	40 857	88	36 382	81	-11,0	-7,6
	M	115	15 583	209	15 033	201	-3,5	-3,8
∅			13 429	173	12 077	169	-7,2	-1,4

Wertebereich zwischen $-12,5\%$ und $32,2\%$ mit einem Mittelwert von $-1,4\%$. Das die Homogenität auch einen positiven Einfluss auf die Taktfrequenz haben kann, wird durch das Design *MEWB L* bewiesen. Dieses erzielt eine um $32,2\%$ höhere Taktfrequenz als die inhomogene ISE Implementierung.

Zweidimensionale Kommunikationsinfrastrukturen

Die Auswertung der zweidimensionalen Kommunikationsinfrastrukturen wird für die drei Kommunikationsinfrastrukturtypen für die Virtex-4-Architektur und die 7-Serie-Architektur durchgeführt und jeweils separat vorgestellt.

Virtex-4-Architektur

Aufgrund der Partitionierung wurde für die Auswertung der DB-Kommunikationsinfrastrukturen ein LX15 FPGA und für die EWB-Kommunikationsinfrastrukturen ein FX100 FPGA gewählt.

FPGA-Ressourcen

Die Tabelle A.24 in Anhang B.5 dokumentiert die FPGA-Ressourcen für homogene und inhomogene, zweidimensionale Kommunikationsinfrastrukturen in der Virtex-4-Architektur. Durch die bereits auf Flächenoptimierung eingestellten *MAP*-Einstellungen ergibt sich für die Auswertung jeweils ein direkter Vergleich zwischen den inhomogenen und den homogenen Designs bzw. den Packern von Xilinx und DHHarMa.

Slice-Anzahl Wie in Abbildung 4.30 dargestellt, ist die Slice-Anzahl in allen homogenen Designs deutlich geringer. Prozentual werden in den DHHarMa Implementierungen zwischen $31,3\%$ und $47,5\%$ weniger Slices verwendet. Der Durchschnittswert liegt bei

41,4 %. Dies spiegelt sich ebenfalls in der Slice-Auslastung der Virtex-4-FPGAs wieder: Die inhomogenen Makros belegen zwischen 2,1 % und 25,0 %, die homogenen Makros hingegen zwischen 1,4 % und 14,6 % der verfügbaren Slice-Ressourcen. Hierbei sei nochmals auf die zwei verwendeten FPGAs für die unterschiedlichen Kommunikationsinfrastrukturtypen hingewiesen.

Den Einfluss des Kommunikationsinfrastrukturtyps (SEWB oder MEWB) führt bei der Master-fähigen Variante zu einem Slice-Anstieg von 22,0 % bis 24,7 %. Der in Abschnitt 4.4.5 prognostizierte Einfluss auf den FPGA-Ressourcenbedarf (Register: 47,3 % bis 50,0 %, LUTs: 39,6 % bis 46,0 %) konnte durch die Packung somit deutlich reduziert werden.

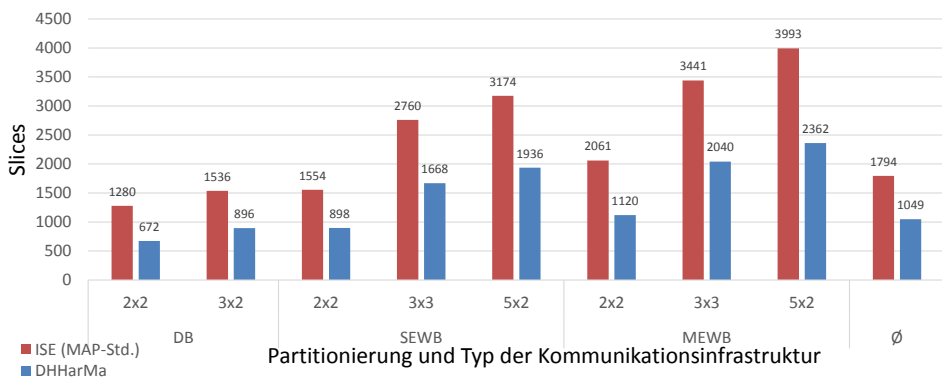


Abbildung 4.30: Gegenüberstellung der Slice-Anzahl für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur

Packungsdichte Die obere Grenze der Packungsdichte der Slices in der Virtex-4-Architektur liegt für beide Packer bei 2 LUTs und 2 Registern (siehe Abschnitt 4.5.3). Bei der Register-Packungsdichte wird dieser Maximalwert von keinem Hardmakro erreicht (siehe Abbildung 4.31). Der Wertebereich der inhomogenen Hardmakros liegt mit 0,74 Registern–1,06 Registern und einem Mittelwert von 0,86 Registern weit von dem möglichen Maximum entfernt. Die homogenen Kommunikationsmakros können mit einem Durchschnittswert von 1,47 Registern einen deutlich besseren Wertebereich (1,22 Registern–1,62 Registern) aufspannen.

Der Maximalwert der Packungsdichte wird jedoch annähernd bei der LUT-Packungsdichte der homogenen Kommunikationsmakros erreicht (siehe Abbildung 4.32). Der durchschnittliche Wert liegt bei 1,80 LUTs bei einem Wertebereich von 1,51 LUTs–1,98 LUTs. Die Packungsdichte der inhomogenen Kommunikationsmakros variiert zwischen 0,88 LUTs und 1,19 LUTs mit einem Mittelwert von 1,05 LUTs.

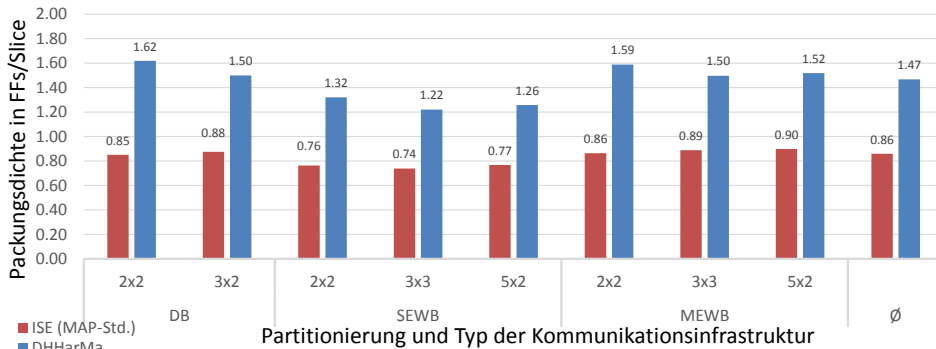


Abbildung 4.31: Gegenüberstellung der FF-Packungsdichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur

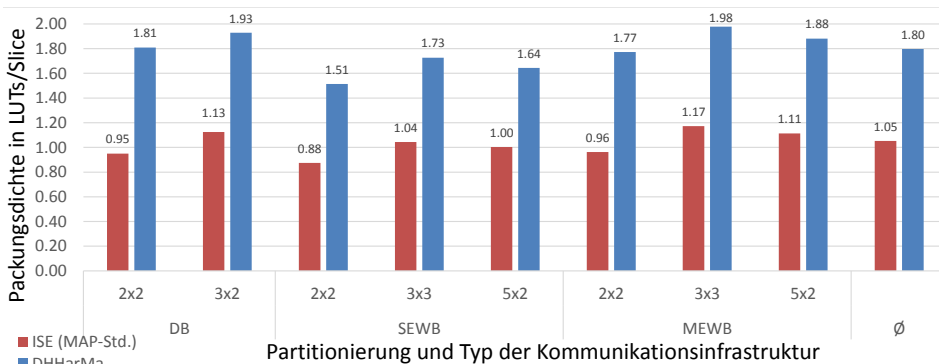


Abbildung 4.32: Gegenüberstellung der LUT-Packungsdichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur

Auslastung PR-Region Die Größe der PR-Regionen wurde für die zweidimensionalen Designs variiert. In den Partitionierungen der DB-Designs stehen pro PR-Region 256 Slices mit jeweils 512 LUTs und Register zur Verfügung. In den Partitionierungen 2x2 und 5x2 der EWB-Designs stehen 480 Slices (960 LUTs und Register) und in der 3x3 Partitionierung 512 Slices (1 024 LUTs und Register) zur Verfügung. In Tabelle 4.15 ist absolute und relative Slice-Auslastung der PR-Region durch die Kommunikationsinfrastruktur sowie die resultierende Anzahl der freien Slices dargestellt.

Die MAP-Std. führen in dem LX15 FPGA zu einer prozentualen Auslastung der PR-Region zwischen 10,4% und 87,5%, in den Partitionierungen des FX100 FPGAs

Tabelle 4.15: Auslastung der PR-Region für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur (DB: LX15, EWB: FX100)

Typ	Design		Inhom. Komm.-Makro (ISE) Standard MAP-Einstellungen			Hom. Komm.-Makro (DHHarMa) Standard Einstellungen		
	Konfiguration		Slices/PRR		Freie Slices	Slices/PRR		Freie Slices
	Part.	Busbr.	in #	in %	in #	in #	in %	in #
DB	2x2	8	28	10,9	228	16	6,3	240
		32	112	43,8	144	64	25,0	192
		64	224	87,5	32	128	50,0	128
	3x2	8	27	10,4	229	16	6,3	240
		32	107	41,7	149	64	25,0	192
		64	213	83,3	43	128	50,0	128
SEWB	2x2	95	290	60,3	191	173	36,0	307
	3x3	95	284	55,4	228	173	33,8	339
	5x2	95	278	57,9	202	173	36,0	307
MEWB	2x2	115	343	71,4	137	207	43,1	273
	3x3	115	331	64,7	181	207	40,4	305
	5x2	115	330	68,8	150	207	43,1	273
∅			214	54,7	159	130	32,9	244

55,4% und 71,4%. In den DHHarMa Implementierungen bleiben deutlich mehr FPGA-Ressourcen für die Anwendungen verfügbar. In den Partitionierungen des LX15 FPGAs ergeben sich relative Auslastungen zwischen 6,3% und 50,0%, in den Partitionierungen des FX100 FPGAs mit den EWB-Kommunikationsinfrastrukturen relative Auslastungen zwischen 33,8% und 43,1%.

Leistungsfähigkeit

Zur Auswertung der Leistungsfähigkeit wird in Tabelle 4.16 die Anzahl der PIPs und die Taktfrequenz dargestellt. Es sei noch mal darauf hingewiesen, dass in den zweidimensionalen Kommunikationsinfrastrukturen jeweils das mit DHHarMa gepackt und platzierte Design verwendet wird, um einen direkten Router-Vergleich zu ermöglichen. Die inhomogenen Kommunikationsmakros wurden anschließend mit der Xilinx Software *FPGA Editor*, die homogenen Kommunikationsmakros mit dem DHHarMa Router verdrahtet.

PIPs Die Gegenüberstellung der Anzahl verwendeter PIPs zwischen den inhomogen und homogenen Kommunikationsmakros zeigt, dass bei einem gleichen Eingangsdesign der DHHarMa Router zur Erhaltung der Homogenität im Durchschnitt 11,6% mehr

Tabelle 4.16: Auswertung der Leistungsfähigkeit für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur (DB: LX15, EWB: FX100)

Typ	Design		Inhom. Komm.-Makro		Hom. Komm.-Makro		Vergleich	
	Konfiguration	Part. Busbr.	PIPs in #	Taktfreq. in MHz	PIPs in #	Taktfreq. in MHz	PIPs in %	Taktfreq. in %
DB	2x2	8	1 158	303	1 258	257	8,6	-15,1
		32	4 947	280	5 279	230	6,7	-17,9
		64	10 476	242	11 399	231	8,8	-4,6
	3x2	8	1 581	259	1 688	215	6,8	-17,0
		32	7 040	238	7 077	198	0,5	-16,5
		64	14 745	213	17 966	182	21,8	-14,6
SEWB	2x2	95	14 442	128	16 032	112	11,0	-12,5
	3x3	95	30 123	121	33 678	56	11,8	-53,7
	5x2	95	34 873	65	41 357	51	18,6	-21,5
MEWB	2x2	115	21 001	186	26 715	148	27,2	-20,4
	3x3	115	46 994	136	53 120	112	13,0	-17,6
	5x2	115	58 013	87	66 171	62	14,1	-28,7
∅			20 449	186	23 478	155	12,6	-20,0

PIPs verwendet. Der Wertebereich liegt hierbei zwischen 0,5 % (*DB 3x2 32*) und 27,2 % (*MEWB 2x2*).

Taktfrequenz Die Taktfrequenz der homogenen Designs ist in allen Kommunikationsmakros geringer. Im Mittel liegt die Taktfrequenz um 20,5 % unter der Taktfrequenz der inhomogenen Lösung des FPGA Editors. Die einzelnen Designs spannen einen Wertebereich von -4,6 % (*DB 2x2 64*) und -53,7 % (*SEWB 3x3*).

7-Serie-Architektur

Durch die gestiegene Anzahl an CLBs innerhalb einer Spalte der Taktregion, ergibt sich eine deutlich größere Einteilung der FPGA-Ressourcen. Das Artix 100T FPGA mit der gleichen Anzahl an Logic Cells wie das Virtex-4 FX100 FPGA, unterteilt sich in der Höhe in lediglich vier Taktregionen. Aus diesem Grund wurde das Kintex 325T FPGA für die 5x2 Partitionierungen verwendet.

FPGA-Ressourcen

Tabelle 4.17 im Anhang B.5 stellt die Ergebnisse der drei Kommunikationsinfrastrukturtypen für die 7-Serie-Architektur anhand der FPGAs A100T und K325T dar.

Slice-Anzahl Der Vergleich der Slice-Anzahl der homogenen und inhomogenen Kommunikationsmakros führt zu ähnlichen Ergebnissen wie in den eindimensionalen Kom-

munikationsinfrastrukturen in der Virtex-6-Architektur (siehe Abbildung 4.33). Absolut werden in den Implementierungen im Mittel für die MAP-Std. 1 289 Slices, für die MAP-Opt. 565 Slices und für DHHarMa 527 Slices verwendet. Der Vergleich der beiden Xilinx Packer-Ergebnissen zeigt, dass die MAP-Opt. Variante zwischen 42,4 % und 66,9 % an Slices einspart. Die Ergebnisse der flächenoptimierten MAP-Einstellungen und DHHarMa variieren zwischen –21,8 % und 12,5 %. Im Durchschnitt benötigen die homogenen DHHarMa-Implementierungen im Vergleich 2,1 % weniger Slices. Aufgrund der deutlich gestiegenen FPGA-Chipfläche in den verwendeten FPGAs fällt die relative Auslastung der Slice-Ressourcen geringer aus.

Die bereitgestellte Master-fähigkeit in der Kommunikationsinfrastruktur resultiert in der 7-Serie-Architektur in einem prozentualen Slice-Anstieg von 28,4 % bis 36,9 % und hat einen größeren Einfluss als in der Virtex-4-Architektur.

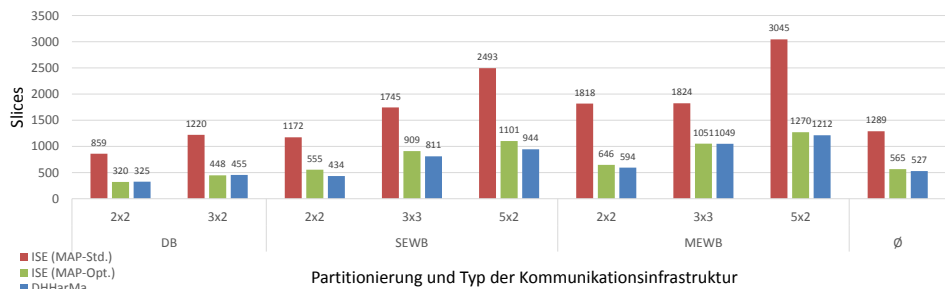


Abbildung 4.33: Gegenüberstellung der Slice-Anzahl für homogene und inhomogene 2D-Kommunikationsinfrastrukturen in der 7-Serie-Architektur

Packungsdichte Die Packungsdichte der Register ist in Abbildung 4.34 dargestellt. Die MAP-Std. Ergebnisse ergeben Packungsdichten von 0,95 Registern–1,67 Registern mit einem Mittelwert von 1,16 Registern. Die MAP-Opt. Implementierungen führen zu Packungsdichten von 2,14 Registern–3,40 Registern und einem Durchschnittswert von 2,86 Registern. Die homogenen Hard-Makros des DHHarMa Packers führen mit Packungsdichten im Bereich von 2,51 Registern–3,35 Registern und einem Mittelwert von 2,91 Registern vergleichbare Ergebnisse.

Die Analyse der LUT-Packungsdichte (siehe Abbildung 4.35) führt zu ähnlichen Ergebnissen. Die homogenen Hard-Makros weisen eine Packungsdichte von 3,13 LUTs–3,85 LUTs mit einem Durchschnitt von 3,56 LUTs. Die Mittelwerte der ISE-Implementierungen liegen bei 1,44 LUTs (MAP-Std.) bzw. 3,49 LUTs (MAP-Opt.).

Auslastung PR-Region Durch die gespiegelte CLB-Anordnung in der 7-Serie ergibt sich eine weitere Inhomogenität (siehe Abschnitt 3.2.5), die eine Partitionierung mit gleichen PR-Regionstypen in der Breite beeinträchtigt. Für die kleineren DB-Designs wurden

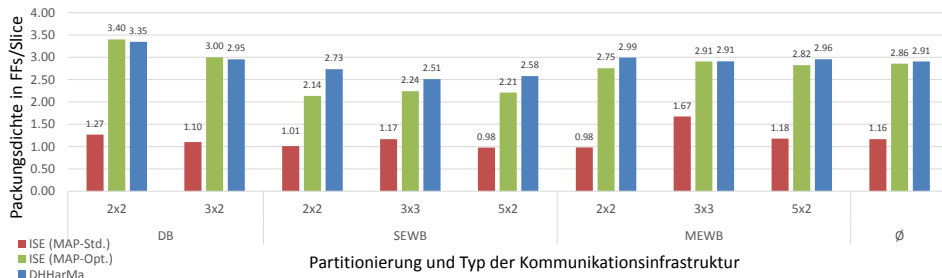


Abbildung 4.34: Gegenüberstellung der FF-Packungsdichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur

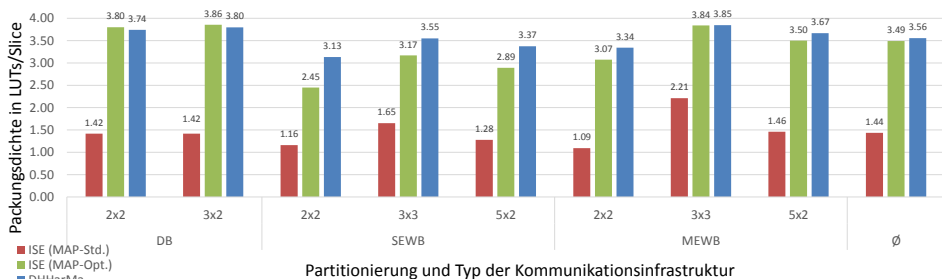


Abbildung 4.35: Gegenüberstellung der LUT-Packungsdichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur

Partitionierungen mit 256 Slices (maximal 2 048 LUTs und Register) pro PR-Region verwendet.

In den größeren EWB-Designs wurde die Slice-Anzahl wie folgt festgelegt:

- 2x2: 600 Slices (4 800 LUTs und Register)
- 3x3: 200 Slices (1 600 LUTs und Register)
- 5x2: 400 Slices (3 200 LUTs und Register)

Die resultierende absolute und relative Slice-Auslastung der PR-Region ist in Tabelle 4.17 dargestellt. Die MAP-Std. Implementierungen führen zu einer prozentualen Auslastung der PR-Region zwischen 8,5 % und 89,6 %. Für die MAP-Opt. und DHHarMa Implementierungen ergeben nahezu identische PRR-Auslastungen. Die DHHarMa Implementierungen sind hierbei in den DB- und MEWB-Designs jeweils um eine Slice höher, in den SEWB-Designs jedoch um fünf bis sechs Slices geringer, sodass die DHHarMa Implementierungen im Durchschnitt eine etwas bessere prozentuale Auslastung erzielen.

Tabelle 4.17: Auslastung der PR-Region für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur (5x2 Partitionierung: Kintex-7 325T, andere Artix-7 100T)

Typ	Design		Inhom. Komm.-Makro (ISE)						Hom. Komm.-Makro (DHHarMa)		
	Konfig. Part.	Busbr.	Std. MAP-Einstellungen		Freie Slices in #	Opt. MAP-Einstellungen		Slices/PRR in #	Std. Einstellungen		Freie Slices in #
			Slices/PRR in #	in %		Slices/PRR in #	in %		in #	in %	
DB	2x2	8	25	9,6	232	8	3,1	248	9	3,5	247
		32	91	35,6	165	32	12,5	224	33	12,9	223
		64	174	67,8	83	64	25,0	192	65	25,4	191
	3x2	8	22	8,5	234	8	3,1	248	9	3,5	247
		32	83	32,2	174	32	12,5	224	33	12,9	223
		64	176	68,9	80	64	25,0	192	65	25,4	191
SEWB	2x2	95	208	34,7	392	91	15,2	509	85	14,2	515
	3x3	95	179	89,6	21	90	45,2	110	85	42,5	115
	5x2	95	215	53,8	185	91	22,8	309	85	21,3	315
MEWB	2x2	115	308	51,3	292	104	17,3	496	105	17,5	495
	3x3	115	167	83,5	33	104	52,0	96	105	52,5	95
	5x2	115	270	67,4	130	104	26,0	296	105	26,3	295
∅			160	50,2	168	66	21,6	262	65	21,5	263

Leistungsfähigkeit

In Tabelle 4.18 sind die Anzahl der PIPs und die Taktfrequenz der Implementierungen für die 7-Serie dargestellt. Analog zu der Auswertung in der Virtex-4-Architektur, wird als Eingangsdesign das mit DHHarMa gepackt und platzierte Design verwendet und anschließend mit der Xilinx Software *FPGA Editor* bzw. dem DHHarMa Router verdrahtet.

PIPs Der Vergleich der Anzahl verwendeter PIPs führt zu dem Ergebnis, dass der DHHarMa Router im Durchschnitt 21,1 % mehr PIPs verwendet als der *FPGA Editor* (Wertebereich: 3,6 % (*DB 2x2 8*) bis 30,6 % (*DB 3x2 32*). Im Vergleich mit der Virtex-4-Architektur werden somit nochmals 9,5 % mehr PIPs verwendet (Virtex-4: 11,6 %).

Taktfrequenz Wie bereits bei der Leistungsauswertung der eindimensionalen Kommunikationsinfrastrukturen angemerkt, gibt diese gestiegene Anzahl der PIPs keinen Aufschluss auf die resultierende Taktfrequenz. So sinkt die durchschnittliche Taktfrequenz aller untersuchter Designs für die 7-Serie-Architektur im Vergleich mit der Virtex-4-Architektur um 8,5 % auf –11,0 %. In einigen DB-Designs ist die Taktfrequenz der homogenen Kommunikationsinfrastrukturen sogar höher als in den inhomogenen Pendants. Das Design *DB 3x2 32* markiert mit einer um 9,0 % höheren Taktfrequenz die obere Schranke des Wertebereichs. Interessanterweise weist gerade dieses Design gleichzeitig mit 30,7 % mehr benötigter PIPs den größten Unterschied in dem Vergleich benötigter PIPs auf. Die untere Schranke des Wertebereichs der Taktfrequenz wird

Tabelle 4.18: Auswertung der Leistungsfähigkeit für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur (5x2 Partitionierung: Kintex-7 325T, andere Artix-7 100T)

Typ	Design		Inhom. Komm.-Makro		Hom. Komm.-Makro		Vergleich	
	Konfiguration Part.	Busbr.	PIPs in #	Taktfreq. in MHz	PIPs in #	Taktfreq. in MHz	PIPs in %	Taktfreq. in %
DB	2x2	8	1 626	240	1 684	259	3,6	7,9
		32	5 156	234	6 672	219	29,4	-6,4
		64	11 791	202	14 408	194	22,2	-4,0
SEWB	3x2	8	1 779	172	2 278	176	28,0	2,4
		32	7 238	155	9 457	169	30,7	9,0
		95	16 282	91	18 972	82	16,5	-9,9
MEWB	3x3	95	34 919	54	42 364	41	21,3	-24,1
	5x2	95	40 906	71	51 197	52	25,2	-26,8
MEWB	5x2	115	24 949	108	29 641	80	18,8	-25,9
		115	52 991	49	61 401	42	15,9	-14,3
		115	61 693	87	74 507	62	20,8	-28,7
Ø			23 575	133	28 416	125	21,1	-11,0

durch das größte Design *MEWB 5x2* mit einer um $-28,7\%$ geringeren Taktfrequenz definiert.

Zusammenfassung

Die Tabelle 4.19 fasst die Auswertung der DPR-Kommunikationsinfrastrukturen der betrachteten Xilinx FPGA-Familien zusammen. Die dargestellten Werte entsprechen den ermittelten Durchschnittswerten der Bewertungsmaße.

FPGA-Ressourcen

Die Gegenüberstellung der inhomogenen, durch Xilinx Werkzeuge erstellten, Implementierungen und der homogenen, durch DHHarMa realisierten, Implementierungen bezüglich der FPGA-Ressourcen resultiert je nach betrachteter FPGA-Architektur in abweichende Ergebnisse. Dies ist primär den variierenden Einstellungsmöglichkeiten des Xilinx Packers *MAP* für die einzelnen FPGA-Familien geschuldet.

Die Virtex-4-Implementierungen sind durch die Standardeinstellungen *Optimization Strategy (Area)* und *CLB Pack Factor Percentage (100 %)* auf eine Flächenoptimierung eingestellt. Der Vergleich mit den DHHarMa Implementierungen zeigt jedoch, dass deutlich bessere Ergebnisse erzielt werden können. So benötigen die DHHarMa Implementierungen durchschnittlich $41,4\%$ weniger Slices als die inhomogenen Xilinx Implementierungen. Die Auswertung der weiteren Bewertungsmaße fällt entsprechend auffallend zu Gunsten der homogenen Implementierungen aus. Exemplarisch seien hier

Tabelle 4.19: Zusammenfassende Darstellung der Auswertung für die DPR-Kommunikationsinfrastrukturen der betrachteten Xilinx FPGA-Familien

Arch.	Impl.	FPGA- Ressourcen					Leistungsfähigkeit	
		Slice Anzahl	FPGA-Ausl. in %	Packungsdichte		PRR-Ausl. in %	PIPs Anzahl	Taktfreq. in MHz
				FF	LUT			
V4	Xilinx	1 794	9,5	0,86	1,47	54,7	20 449	186
	DHHarMa	1 049	5,5	1,05	1,80	32,9	23 478	155
V6	Xilinx	1 059	9,1	1,21	1,35	47,8	13 429	173
	- MAP-Opt. DHHarMa	444 419	3,8 3,6	2,82 3,04	3,14 3,37	17,8 17,7	12 077	169
7S	Xilinx	1 289	6,1	1,16	1,44	50,2	23 757	133
	- MAP-Opt. DHHarMa	565 527	3,1 2,5	2,86 2,91	3,49 3,56	21,6 21,5	28 416	125

die Packungsdichten der DHHarMa Implementierungen erwähnt, welche teils nah an der Begrenzung durch die Slice-Architektur liegen. Im Mittel werden Packungsdichten von 1,47 Registern und 1,80 LUTs gegenüber 0,86 Registern und 1,05 LUTs in den Xilinx Implementierungen erreicht.

Die Standardeinstellungen ab der Virtex-6-Familie sind hingegen auf eine Optimierung der Leistung konfiguriert. Die Verwendung der MAP-Einstellungen *Maximum Compression* und *LUT Combining (Area)* ab der Virtex-6-Architektur in der flächenoptimierten Variante (MAP-Opt.) führt sowohl in den eindimensionalen, als auch in den zweidimensionalen Implementierungen zu einer deutlich geringeren Slice-Anzahl im Vergleich zu den Standardeinstellungen (MAP-Std.). In beiden Fällen verwenden die MAP-Opt. Implementierungen etwa 57 % weniger Slices. Die homogenen DHHarMa Implementierungen benötigen wiederum im Mittel etwas weniger Slices als die MAP-Opt. Implementierungen (1D: 7,1 % und 2D: 2,1 %). Dies hat entsprechende positive Auswirkungen auf die Gesamtauslastung des FPGAs, den (LUT- und Register-) Packungsdichten und der Auslastung der PR-Region durch die Kommunikationsinfrastruktur. Die Auswertung zeigte, dass die DHHarMa und MAP-Opt. Implementierungen auf einem nahezu identischen Niveau liegen. In dem letztgenannten Bewertungsmaß liegen die Abweichungen bei 0,1 % (1D: MAP-Opt. 17,8 % vs. DHHarMa 17,7 % und 2D: MAP-Opt. 21,6 % vs. DHHarMa 21,5 %). Der Großteil der FPGA-Ressourcen ist somit für die Anwendung verfügbar.

Leistungsfähigkeit

Die Auswertung der Leistungsfähigkeit für den *vollständigen Vergleich* der Werkzeugketten erfolgt anhand der Virtex-6-Architektur und resultiert für die Taktfrequenz zu einem Wertebereich zwischen -12,5 % und 32,2 % mit einem Mittelwert von -1,4 % aus Sicht

der DHHarMa-Werkzeugkette. Die Mittlung der Taktfrequenzen ergeben Werte von 173 MHz bzw. 169 MHz.

Durch die Verwendung des gleichen Eingangsdesigns in den zweidimensionalen DPR-Kommunikationsinfrastrukturen kann ein *direkter Vergleich* der Router erfolgen. Auch bei dieser Betrachtung variieren die Ergebnisse in den untersuchten FPGA-Architekturen. Bei der Auswertung der Implementierungen der Virtex-4-Architektur ergeben sich für die Taktfrequenz Mittelwerte von 186 MHz und 155 MHz. Die Taktfrequenz liegt im Mittel um 20,0 % unter der Taktfrequenz der inhomogenen Lösung des FPGA Editors bei einem Wertebereich von $-53,7\%$ und $-4,6\%$. In der Auswertung der 7-Serie beträgt der Unterschied der Xilinx und DHHarMa Implementierungen bzgl. der Taktfrequenz lediglich $-11,0\%$ (Wertebereich: $-28,7\%$ und $9,0\%$) mit Mittelwerten von 133 MHz bzw. 125 MHz. Als Gründe für die unterschiedlichen Ergebnisse der Taktfrequenzen in den FPGA-Architekturen können zum einen die unterschiedlichen Leitungstypen und zum anderen die abweichenden Partitionierungen durch den variierenden FPGA-Aufbau genannt werden.

Die Auswertung zeigt weiterhin, dass die Anzahl verwendeter PIPs keinen entscheidenden Aufschluss auf die resultierende Taktfrequenz gibt. So werden bei den homogenen Implementierungen in der Virtex-4-Familie 21,1 % und in der 7-Serie 12,4 % mehr PIP-Ressourcen verwendet.

4.6.4 Time-to-Digital-Converter

Das Prinzip der Verzögerungsleitung kann, wie in Abschnitt 4.1.2 beschrieben, zur Zeitauswertung zweier Pulse verwendet werden. Für eine hohe Auflösung wird in einer FPGA-Implementierung die Carry Chain einer Slice verwendet. Der Puls wird in die Verzögerungsleitung durch den unteren Bypass-Eingang einer Slice in die Carry Chain geführt. Von dort wird der Puls Slice-intern vertikal in die darüber liegenden Parts geleitet. Anschließend wird der Puls über den Ausgang der Carry Chain (COUT) in die darüberliegende Slice propagiert. Alle folgenden Slices nehmen stets den Puls über den Eingang CIN entgegen, wodurch insgesamt zwei Regionstypen benötigt werden:

1. Erste Slice der Verzögerungsleitung : Pulsempfang über den unteren Bypass-Eingang der Slice
2. Alle folgenden Slices der Verzögerungsleitung : Pulsempfang über den Carry Chain Eingang CIN

Die Abbildungen 4.36 und 4.37 zeigen die Slice-Konfiguration beider Regionstypen für die Slice-Architektur ab der Virtex-6-Familie. Die Abtastung des Pulses wird durch die Verwendung eines internen Registers erreicht. Die Genauigkeit kann weiterhin erhöht werden, indem ein weiteres Register über den MUX-Ausgang verbunden wird. Die gepunkteten roten Linien in den Abbildungen 4.36 und 4.37 visualisieren dieses. Zu beachten ist, dass es aufgrund der Slice-Architektur zu einer Invertierung des Eingangspulses kommt, da der Ausgang O6 das Kontrollsignal des Carry-Multiplexers

realisiert und auf Eins gesetzt sein muss und das XOR-Gatter der Carry Chain mit dem Ausgang der LUT6 (O6) verknüpft ist.

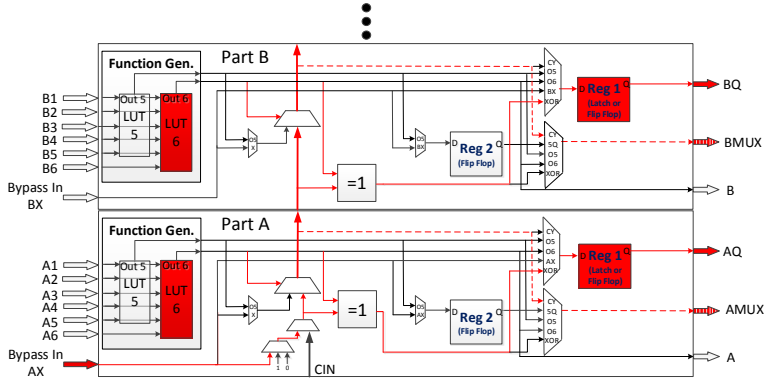


Abbildung 4.36: Slice-Konfiguration der Eingangsbeschaltung der Verzögerungsleitung unter Verwendung der Carry Chain (basierend auf [103, S. 104])

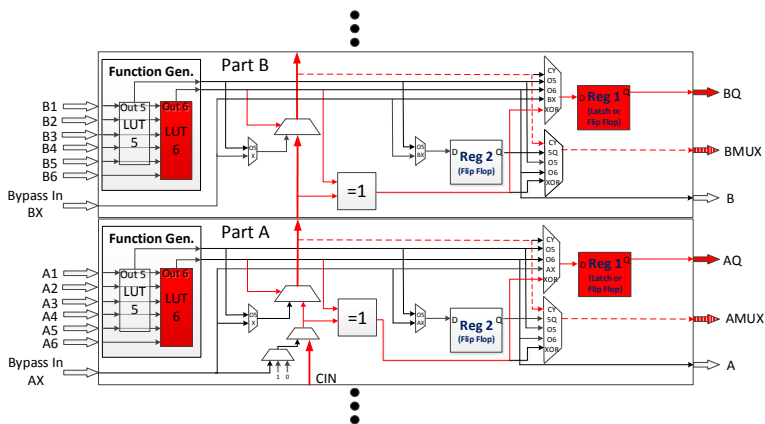


Abbildung 4.37: Slice-Konfiguration der Weiterführung der Verzögerungsleitung unter Verwendung der Carry Chain (basierend auf [103, S. 104])

Für die TDC-Auswertung wurde für beide Varianten homogene und inhomogene Verzögerungsleitungen mit 40 durch die Carry Chain verbundenen Slices erstellt. Die Tabelle 4.20 stellt die Ergebnisse zur Auswertung der FPGA-Ressourcen und der Leistungsfähigkeit zwischen den homogenen und inhomogenen Verzögerungsleitungen

gegenüber. Die dargestellten Ergebnisse basieren auf einem *vollständigen Vergleich* der Entwurfswerkzeuge.

Tabelle 4.20: Auswertung der FPGA-Ressourcen und der Leistungsfähigkeit für homogene und inhomogene Verzögerungsleitungen (basierend auf [105, S. 8])

Design Konfig.(Bit, Part.)	Kacheln	Homogene Verzögerungsleitung					Inhomogene Verzögerungsleitung				
		FFs	LUTs	Slices	PIPs	Takt (MHz)	FFs	LUTs	Slices	PIPs	Takt (MHz)
160	40	320	160	80	863	750	320	240	81	1 731	653
160, XOR	40	160	160	40	223	750	160	160	40	359	750

Bei dem Vergleich der FPGA-Ressourcen der homogenen und inhomogenen Verzögerungsleitungen der ersten Variante fällt insbesondere die Diskrepanz der verwendeten LUTs und PIPs auf. Das inhomogene Hard-Makro verwendet 240 LUTs, welche über 1 731 PIPs verbunden sind und somit zu einer maximalen Taktfrequenz von ca. 652 MHz führt. Die, um 33,3 %, geringere Anzahl an benötigter LUTs des homogenen Makros zeigt, dass der homogene Packer eine kompaktere Packung als der Xilinx Packer durchführt. Dies führt weiterhin zu einer deutlich geringeren Anzahl an PIPs (50,2 %) im homogenen Hard-Makro. Da der DHHarMa-Placer miteinander verbundene Slices so nah wie möglich platziert, werden weniger Verdrahtungsressourcen benötigt, welches sich positiv auf die Taktfrequenz auswirkt. Diese liegt bei der homogenen Implementierung mit 750 Mhz um 14,9 % höher als beim inhomogenen Pendant.

In der zweiten Variante (XOR) wird nur der Ausgang des XOR-Gatters der Carry Chain verwendet, wodurch sich die Genauigkeit des TDCs verringert. Die benötigten FPGA-Ressourcen der homogenen und inhomogenen Implementierung unterscheiden sich nur in der Anzahl benötigter Netze und PIPs, die Anzahl der Register, LUTs und Slices ist identisch. Bei der Betrachtung der Verdrahtungsressourcen ist die homogene Implementierung mit 223 PIPs im Vergleich zu 359 PIPs bei der inhomogenen Implementierung etwas ressourcenschonender. Dies hat in diesem Fall jedoch keinen Einfluss auf die maximale Taktfrequenz. Beide Varianten erreichen eine Taktfrequenz von 750 MHz. Die Abbildung 4.38 zeigt eine homogene, durch die DHHarMa-Software erstellte, und eine inhomogene, durch die Xilinx Software erstellte Verzögerungsleitung.

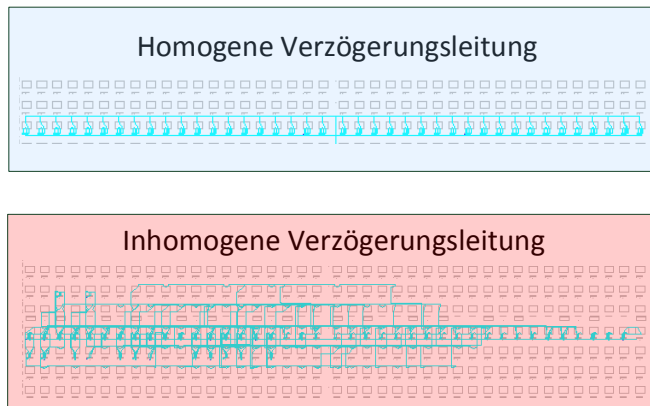


Abbildung 4.38: Implementierung einer homogenen und inhomogenen Verzögerungsleitung mit 40 zusammengeschalteten CLB (um 90° gedrehte Darstellung, basierend auf [105, S. 7])

4.7 Zusammenfassung

Zum Einstieg in die Thematik der Homogenität wurden zunächst zwei typische Anwendungsgebiete vorgestellt, in denen homogen verwendete FPGA-Ressourcen innerhalb eines Designs für die Realisierung der Funktionalität benötigt werden. Anschließend wurde auf den konzipierten und umgesetzten Entwurfsablauf, aufgeteilt in ein Xilinx-basiertes Frontend und ein DHHarMa-basiertes Backend, zur Homogenisierung eines Designs eingegangen.

Die Zwischendatei des Entwurfsablaufs ist ASCII-basiert und folgt einer von Xilinx definierten Sprache, der *Xilinx Design Language* (XDL). Neben der Zwischendatei kann ein sogenannter XDL-Report zur Beschreibung eines Xilinx FPGAs bis zur 7-Serie in dieser Sprache erstellt werden. Beide XDL-Dateitypen sind somit für die Realisierung von DHHarMa essenziell und wurden in einem separaten Abschnitt beschrieben.

Durch die Verwendung einer HDL-Datei als DHHarMa-Eingangsdatei konnte eine generische, hierarchische und daher wartbare Bibliothek für DPR-Kommunikationsinfrastrukturen realisiert werden. Als Kommunikationsprotokoll wurde eine registrierte Version des Wishbone-Protokolls verwendet und jeweils Slave- und Master-Slave-Varianten für ein- und zweidimensionale Kommunikationsinfrastrukturen entwickelt.

Die Implementierung des DHHarMa Backends erfolgt durch verschiedene Softwarekomponenten – insbesondere einen Packer, Placer und Router. Bevor diese einzelnen Homogenisierungswerkzeuge vorgestellt wurden, wurde zunächst auf zwei erstellte FPGA-Datenbanken eingegangen. Diese enthalten Informationen über den konkreten Aufbau eines FPGAs (verfügbare Blöcke und die Verdrahtungsinfrastruktur) und sind essenziell für die aufgeführten DHHarMa Werkzeuge.

Im Anschluss erfolgte eine Auswertung der DHHarMa-Softwarekomponenten. Hierbei wurde zunächst eine Speicher- und Zeitauswertung für die FPGA-Datenbanken vorgestellt. Des Weiteren wurden die vier Platzierungsstrategien für den Anwendungsfall der Kommunikationsinfrastruktur analysiert. Hierbei wurde ebenfalls der Einfluss der FPGA-Architektur durch die Verwendung von Virtex-4- und Artix-7 FPGAs berücksichtigt. Als Ergebnis konnte festgehalten werden, dass die Platzierungsstrategie *Simulated Annealing* mit einer Cluster-basierten Anfangsplatzierung im Durchschnitt die besten Platzierungsergebnisse bei einer vergleichsweise geringen Ausführungszeit erzielt.

Eine Auswertung aller Softwarekomponenten erfolgte durch die Gegenüberstellung von inhomogenen, mit Xilinx Werkzeugen realisierten, Implementierungen und homogenen, mit DHHarMa erstellten, Implementierungen für die beiden zuvor vorgestellten Anwendungsgebiete mit homogenen Designanforderungen. Hierbei war insbesondere die Fragestellung interessant, welchen Einfluss die Homogenität auf den FPGA-Ressourcenbedarf und die Leistungsfähigkeit hat. Um einen Einfluss durch die FPGA-Architektur ermitteln zu können, erfolgte die Auswertung anhand dreier unterschiedlicher FPGA-Architekturen (Virtex-4, Virtex-6 und 7-Serie).

Die Gegenüberstellung bezüglich der FPGA-Ressourcen ergab aufgrund der, je nach FPGA-Familie variierenden, Einstellungsmöglichkeiten des Xilinx Packers *MAP* unterschiedliche Ergebnisse. So benötigen die DHHarMa Implementierungen in der Virtex-4-Architektur durchschnittlich 41,4 % weniger Slices als die inhomogenen Xilinx Implementierungen. Ab der Virtex-6-Architektur führt die Verwendung von optimierten, von den Standardeinstellungen abweichenden, *MAP*-Einstellungen sowohl in den eindimensionalen, als auch in den zweidimensionalen Implementierungen zu durchschnittlichen Slice-Einsparungen von 57 % (im Vergleich mit den *MAP*-Standardeinstellungen). Die homogenen DHHarMa Implementierungen benötigen wiederum weniger Slices als diese optimierten *MAP*-Implementierungen (1D: 7,1 % und 2D: 2,1 %). Die geringeren Slice-Anzahlen haben entsprechend positive Auswirkungen auf die Bewertungsmaße Gesamtauslastung des FPGAs, die (LUT- und Register-) Packungsdichten und die Auslastung der PR-Region durch die DPR-Kommunikationsinfrastruktur. So liegt die Packungsdichte der DHHarMa Implementierungen über denen des FPGA-Herstellers. Die Auswertung des letzten Bewertungsmaßes verdeutlicht, dass die DHHarMa Implementierungen stets die wenigsten FPGA-Ressourcen verwenden, sodass der Großteil der FPGA-Ressourcen innerhalb der PR-Region für die Anwendung zur Verfügung steht und nicht durch die DPR-Kommunikationsinfrastruktur belegt ist.

Die Auswertung der Leistungsfähigkeit für einen *vollständigen Vergleich* der Entwurfswerkzeuge erfolgte anhand der Virtex-6-Architektur und resultierte in durchschnittliche Taktfrequenzen von 173 MHz (Xilinx) und 169 MHz (DHHarMa). Hierbei ergaben sich Abweichungen im Bereich von -12,5 % und 32,2 % aus Sicht der DHHarMa Implementierungen. Durch die Verwendung des gleichen Eingangsdesigns in den zweidimensionalen DPR-Kommunikationsinfrastrukturen erfolgte weiterhin ein *direkter Vergleich* der Router. Für die Virtex-4-Architektur ergaben die Xilinx Implementierungen eine Taktfrequenz, welche gemittelt 20,0 % über der Taktfrequenz der DHHarMa Implementierung

lag (Xilinx: 188 MHz, DHHarMa: 155 MHz). In der 7-Serie beträgt der Unterschied lediglich noch 11,0% mit Mittelwerten von 133 MHz bzw. 125 MHz. Die Auswertung zeigte ferner, dass die Anzahl verwendeter PIPs keinen eindeutigen Aufschluss auf die resultierende Taktfrequenz gibt.

Die Auswertung der TDC-Implementierungen basierte auf einem *vollständigen Vergleich* der Entwurfswerkzeuge und führte zu ähnlichen Ergebnissen. Die mittels DHHarMa erstellten TDCs benötigen weniger FPGA-Ressourcen bei einer Taktfrequenz, welche mindestens der der inhomogenen, durch Xilinx Softwarekomponenten erstellten, Implementierung entspricht. Zusammenfassend zeigte die Analyse aller Designs, dass die benötigte Homogenität im Mittel einen, teils deutlichen, positiven Einfluss auf den FPGA-Ressourcenbedarf und einen geringen negativen Einfluss auf die Leistungsfähigkeit hat. Die verwendete FPGA-Architektur hat hierbei auf beide Größen einen entscheidenden Einfluss.

5 Einsatz von FPGAs in Umgebungen mit Strahlung

Mikroelektronische Bauteile werden in einer Vielzahl unterschiedlicher Anwendungen in verschiedensten Bereichen verwendet. Der Einsatz in missionskritischen Anwendungsbereichen, in denen der Ausfall einer Komponente zu schweren finanziellen Verlusten oder im schlimmsten Fall zum Verlust von Menschenleben führen könnte, stellt dabei besondere Anforderungen an die Systemkomponenten. In diesen Anwendungsbereichen müssen FPGAs entsprechend geschützt werden. Fehler können hierbei auf unterschiedlichen Wegen in ein elektronisches System gelangen. Mechanische Fehler können zum Beispiel durch Vibrationen, Temperaturschwankungen oder Druckveränderungen entstehen. Elektronische Fehler können durch elektromagnetische Störausstrahlung oder elektrostatische Entladung (engl. ElectroStatic Discharge, ESD) in das System gelangen. Eine weitere Fehlerquelle, die insbesondere im Weltraum, aber auch auf der Erde auftritt, ist Strahlung. In diesem Kapitel wird diese Fehlerquelle intensiv betrachtet und ihre Auswirkungen auf FPGAs und die daraus resultierenden Maßnahmen vorgestellt.

Die drei ersten Abschnitte dieses Kapitels gehen auf (physikalische) Grundlagen der Strahlung ein. In dem ersten Abschnitt wird der Ursprung der Strahlung betrachtet. Anschließend werden die physikalischen Eigenschaften von und die daraus resultierenden Effekte durch Strahlung vorgestellt. Die Fehlerklassen, die Auswirkungen auf unterschiedliche FPGA-Technologien, und verschiedene Verfahren zur Behandlung bzw. Linderung/Abschwächung (engl. Radiation Effect Mitigation) der Strahlungseffekte auf unterschiedlichen Hardware- und Software-Ebenen, stehen in den darauffolgenden Abschnitten im Vordergrund. Standard-FPGAs sind besonders anfällig für Strahlung, was zu einer Vielzahl unterschiedlicher Fehler führen kann. In Umgebungen mit hoher Strahlungsbelastung werden daher strahlungstolerante oder -gehärtete FPGAs verwendet. Die aktuell verfügbaren FPGAs dieser Kategorie werden anschließend beschrieben. Abschließend werden Experimente bzw. Projekte aus terrestrischen und kosmischen Anwendungsbereichen vorgestellt, welche den Einsatz und Trend von FPGAs-basierten Systemen veranschaulichen.

5.1 Strahlung in unterschiedlichen Umgebungen

In [15, S. 44] wird Strahlung allgemein beschrieben als Menge von geladenen oder nicht geladenen Partikeln, die mit einem anderen elektronischen System durch einen Energieaustausch interagiert. Die Umgebung mit den, in der Natur vorkommenden,

rauesten Bedingungen ist der Weltraum, in dem eine Vielzahl unterschiedlicher Strahlungsarten existieren. Die Partikel, vorwiegend Elektronen, Protonen und Schwerionen aber auch Neutronen, bewegen sich im Vakuum des Weltraums und interagieren erst beim Auftreffen auf andere Atome und Moleküle, zum Beispiel denen der Erdatmosphäre. Durch die Reaktion beim Aufeinandertreffen, verlieren die Partikel an Energie, sodass die Erdatmosphäre als eine Art Schutzschild fungiert. Elektronische Schaltungen in terrestrischen Umgebungen sind daher sicherer vor kosmischer Strahlung. Dennoch gibt es Anwendungsgebiete auf der Erde, in denen eine ähnliche oder sogar höhere Strahlung vorherrscht. Beispiele hierfür sind in der Hochenergiephysik (engl. **High-Energy Physics Experiment**, HEP) zu finden (siehe Abschnitt 5.9.1). Durch die stetige Miniaturisierung in der Halbleiter-Prozesstechnik und der gleichzeitigen Herabsetzung der Betriebsspannung in elektronischen Bauteilen, werden diese zunehmend anfälliger, auch für terrestrische Strahlung.

5.1.1 Kosmische Strahlung

Strahlung im Weltraum hat ihren Ursprung entweder innerhalb oder außerhalb des Sonnensystems und teilt sich daher in Solar- und galaktische Strahlung auf [58, S. 1 ff.].

Solarstrahlung Strahlung innerhalb des Sonnensystems wird als Solarstrahlung (engl. **Solar Cosmic Ray**, SCR) bezeichnet. Beispiele hierfür sind Sonneneruptionen (engl. solar flares), koronaler Massenauswurf (engl. coronal mass ejection) oder Sonnenwinde (engl. solar wind). Der Sonnenfleckenzyklus (engl. solar cycle) beschreibt die Periodizität der Sonnenflecken von einem zum nächsten Minimum und beträgt im Durchschnitt 11 Jahre. In einem Minimum kommt es oft monatelang zu keinem Sonnenfleck, in einem Maximum hingegen zu Hunderten.

Galaktische Strahlung Strahlung außerhalb des Sonnensystems wird galaktische Strahlung (engl. **Galactic Cosmic Ray**, GCR) genannt und enthält im Vergleich zu der Solarstrahlung deutlich mehr Energie. Der Ursprung ist unter anderem in Pulsaren oder Explosionen von (Super-) Novas zu finden.

Wie bereits in der Einleitung erwähnt, wirken die Magnetfelder der Planeten als Schutzschild. In Abbildung 5.1 ist das Magnetfeld der Erde, Magnetosphäre genannt, dargestellt. Die Linien stellen exemplarisch den Sonnenwind dar, welcher das Übergangsgebiet (engl. *Magnetosheath*), den sonnenzugewandten Teil der Magnetosphäre, staucht und der Magnetoschweif (engl. *Magnetotail*), den sonnenabgewandten Teil, streckt. Die Wechselwirkung mit der Magnetosphäre hat einen entscheidenden Einfluss auf die Strahlung in erdnahen Orbits. Die Form der Magnetosphäre ändert sich je nach einfallender Strahlung stark. Einen besonderen Einfluss hat dabei der erwähnte Sonnenfleckenzyklus. Je höher die Strahlung, um so mehr wird das Übergangsgebiet gestaucht, wodurch die Strahlung tiefer eindringen kann.

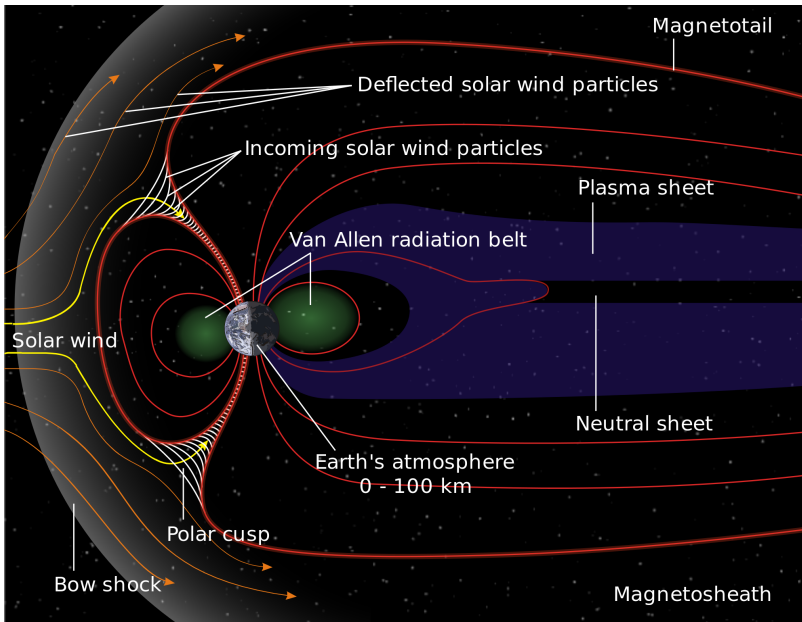


Abbildung 5.1: Magnetosphäre der Erde [49]

Eine zusätzliche Strahlungsquelle ist der, im Jahr 1958 nach seinem Entdecker benannte, van-Allen-Gürtel. Dieser, aus einem inneren und äußeren Ring bestehende, Gürtel fängt energiereiche Teilchen durch Wechselwirkung mit einfallender Strahlung ein. Während der innere Gürtel aus Protonen und Elektronen besteht, sind im äußeren Gürtel vorwiegend Elektronen gefangen. Die Partikel sind aufgrund magnetosphärischer Kräfte eingefangen, bewegen sich jedoch stetig in komplexen Bewegungsmustern. Nukleare Tests auf der Erde haben ebenfalls eine Auswirkung auf die van-Allen-Gürtel. Die einzigen Orbits, die von der van-Allen-Gürtel-Strahlung beeinflusst werden, liegen im sogenannten *Low Earth Orbit* (LEO). Ein Bereich mit besonders hoher Strahlung ist die sogenannte *Südatlantische Anomalie* (engl. *South Atlantic Anomaly*, SAA) vor der Küste Brasiliens, an welcher der innere van-Allen-Gürtel der Erdoberfläche näher kommt. In Tabelle 5.1 sind die typischen Satellitenorbits mit den Kenndaten Höhe, Umlaufzeit und Empfangsfenster zur Kommunikation aufgeführt.

5.1.2 Terrestrische Strahlung

Strahlung auf der Erdoberfläche resultiert hauptsächlich aus der Wechselwirkung von kosmischer Strahlung und der Erdatmosphäre [15, S. 45 f.]. Sie wird daher auch als sekundäre Strahlung bezeichnet und ist im Vergleich zu der primären Strahlung deutlich

Tabelle 5.1: Kenndaten der typischen Satellitenorbits (basierend auf [186])

Orbit	Höhe in km	Umlauf- zeit in h	Empfangsfenster Komm. in h
Low Earth Orbit (LEO)	200 - 2 000	1,5 - 5	0,25
Medium Earth Orbit (MEO)	2 000 - 35 786	5 - 24	2 - 4
Geostationary Earth Orbit (GEO)	≈ 35 786	24	immer
High Earth Orbit	> 36 000	> 24	-
Highly Elliptical Orbit (HEO), auch Molnija-Orbits	400 - 40 000	12	8

schwächer. Der Großteil dieser sekundären Strahlung besteht dabei aus hochenergetischen Neutronen. Die Dosierung auf der Erde ist abhängig von dem magnetischen Feld und der Höhenlage des Ortes.

Zusätzlich existiert Strahlung, die von der Erde selbst emittiert wird [126]. Quellen hierfür sind z. B. Uran oder Thorium im Bauteilgehäuse oder Bor als Dotierungsmaterial im Halbleitersubstrat. Die Strahlungsart besteht hauptsächlich aus niedrigenergetischer Alpha- oder Gammastrahlung und hat in der Regel keinen Einfluss auf ein elektronisches Bauteil. Wenn diese Materialien jedoch in dem Gehäuse oder dem Substrat des Bauteils verwendet werden, kann dies zu einer unerwünschten Reaktion mit dem Bauteil/Halbleitermaterial führen. Hersteller von mikroelektronischen Bauteilen verwenden daher sogenannte *Ultra-Low-Alpha* (ULA) Materialien. Eine Untersuchung des Einflusses dieser terrestrischen Strahlung auf FPGAs folgt in Abschnitt 5.9.1.

5.2 Eigenschaften von Strahlung

Um die Auswirkung von Strahlung auf die Umgebung/Umwelt spezifizieren zu können, werden vier Maße definiert, die zum Beispiel in [15], [177], [153] oder [60] beschrieben sind. Die englischen Fachtermini lauten: *Flux*, *Fluence*, *Cross Section* und *Weibull Curve*. Der folgende Abschnitt fasst die genannten Quellen zusammen.

Abbildung 5.2 visualisiert eine schematische Darstellung für die folgenden Definitionen. Die Gesamtfläche wird mit S (engl. surface) und die Fläche unter Strahlenbeschuss mit a (engl. area) abgekürzt. Der Strahl besteht aus einer Anzahl von Partikeln p , wobei jedes einzelne Partikel einen unterschiedlichen Einfallswinkel α besitzen kann.

Flux/Fluss *Flux*, zu Deutsch Fluss, definiert die Anzahl der Partikel die eine Fläche zu einem Zeitpunkt durchdringen und wird in $\frac{1}{\text{cm}^2 \cdot \text{s}}$ angegeben:

$$f = p/a \cdot t \tag{5.1}$$

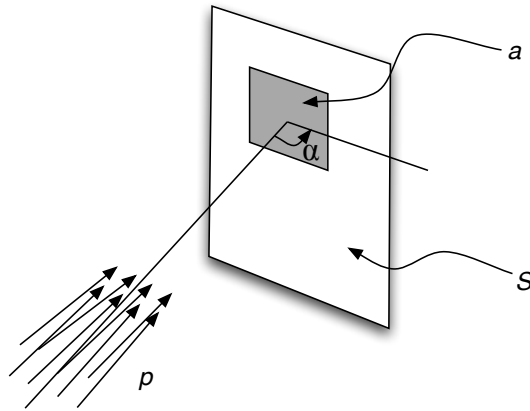


Abbildung 5.2: Darstellung zur Definitionserklärung von Strahlung [15, S. 47]

Fluence/Fluenz *Fluence* (dt. Fluenz) bezeichnet die Flux eines definierten Zeitraums und wird in $\frac{1}{\text{cm}^2}$ angegeben:

$$\Phi = \int_0^t f(t) dt \quad (5.2)$$

Cross Section/Wirkungsquerschnitt Das Maß *Cross Section* σ , im Deutschen Wirkungsquerschnitt, ist ein statisches Flächenmaß, welches die Anfälligkeit eines Bauteils gegenüber Strahlung beschreibt [153]. Die Berechnung erfolgt durch die Division der Anzahl der detektierten Fehler mit der Fluenz und wird in cm^2 gemessen:

$$\sigma = \frac{\#\text{Fehler}}{\text{Fluenz}} = \frac{\#\text{Fehler}}{\#\text{Partikel}/\text{cm}^2} \quad (5.3)$$

Weibull Curve/Weibull Kurve Die sogenannte *Weibull Curve* wird verwendet um die Anfälligkeit eines Bauteils gegenüber Strahlung darzustellen. Abbildung 5.3 zeigt exemplarisch die Weibull Kurve des Konfigurationsspeichers eines Xilinx Virtex-5QV FPGAs [172] bei Protonenbeschuss. Auf der Abszisse des Diagramms ist die Energie der Partikel und auf der Ordinate der Wirkungsquerschnitt (in dem dargestellten Fall pro Bit) dargestellt. Die vier Punkte in dem Diagramm zeigen aufgenommene Werte von Strahlungstests. Der Verlauf der Weibull Kurve wird approximiert.

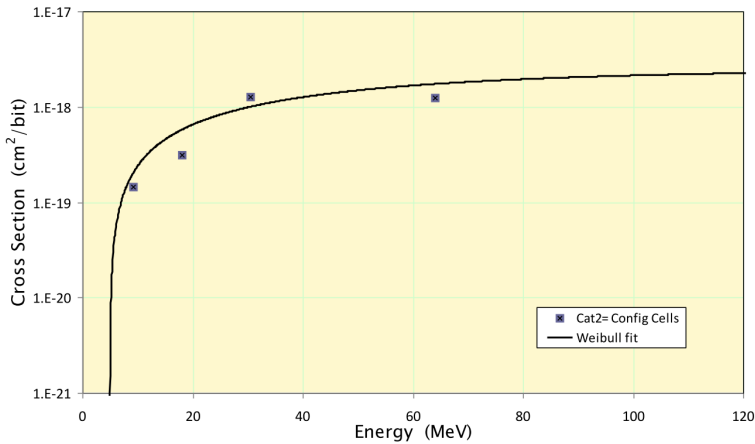


Abbildung 5.3: Weibull Kurve des Konfigurationsspeichers eines Xilinx Virtex-5 QV FPGAs (XQR5VFX130) [172]

5.3 Physikalische Effekte durch Strahlung

Bei der Wechselwirkung von elektronischen Bauteilen und Strahlung kommt es zu verschiedenen physikalischen Effekten. Die Effekte sind in der Quelle [15, S. 47 ff.] detailliert und werden hier zusammengefasst.

In Abbildung 5.4 wird das Eindringen eines Partikels (*primary particle*) in ein Material dargestellt. Der Pfad des Partikels beschreibt durch Kollisionen mit Atomen des Materials einen zufälligen Verlauf. Durch diese Wechselwirkung verliert das primäre Partikel an Energie und sekundäre Partikel entstehen. Wie angedeutet, kann der allgemeine Weg als Zylinder mit einem Radius Δ approximiert werden.

Stopping Power/Bremsvermögen Die *Stopping Power*, zu Deutsch das Bremsvermögen, beschreibt die kinetische Energie, die ein Partikel beim Eindringen in das Material verliert und wird in $\frac{\text{MeV}}{\text{cm}}$ angegeben:

$$S(E) = -\frac{dE}{dx} \tag{5.4}$$

Linear Energy Transfer/Linearer Energietransfer Das Maß *Linear Energy Transfer* (LET), der Lineare Energietransfer, beschreibt hingegen die transferierte Energie an das Material. Dabei wird nur das Material des zuvor approximierten Pfads des Partikels betrachtet. Zu dieser Energie tragen somit nur sekundäre Partikel innerhalb des Zylinders mit dem Radius Δ bei. Als Einheit wird $\frac{\text{MeV cm}^2}{\text{mg}}$ angegeben. Sie wird durch

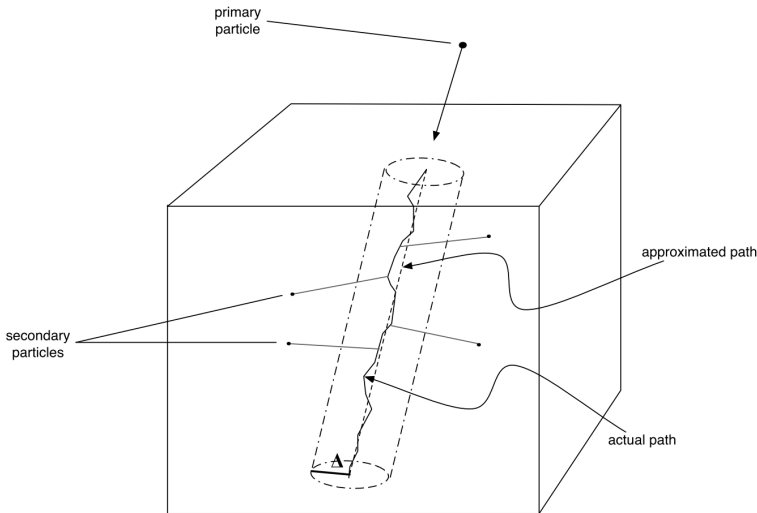


Abbildung 5.4: Eindringen eines Partikels in ein Material und Wechselwirkung mit dem Material [15, S. 48]

die Division der Energie pro Strecke $\frac{\text{MeV}}{\text{cm}}$ und der Dichte des Materials $\frac{\text{mg}}{\text{cm}^3}$ bestimmt.

$$LET_{\Delta} = -\frac{dE_{\Delta}}{dx} \quad (5.5)$$

dE_{Δ} fasst den Energieverlust durch die Kollisionen zusammen, vermindert um die kinetische Energie der sekundären Partikel außerhalb des Zylinders. Geht Δ gegen Unendlich, sind der lineare Energietransfer und das Bremsvermögen identisch.

Funneling Effect/Trichtereffekt In dem Bereich der Mikroelektronik ist der vorherrschende physikalische Effekt durch Strahlung der sogenannte *Funneling Effect* (engl. funnel, Trichter). Ein Partikel, geladen oder ungeladen, durchdringt zunächst die verschiedenen Materiallagen bevor es in das Substrat eindringt. Je nachdem, ob das Partikel geladen oder ungeladen ist, ergeben sich zwei unterschiedliche Verläufe, die in den Abbildungen 5.5 dargestellt sind. Ein geladenes Partikel erzeugt auf dem Pfad durch die Materiallagen einen Ionisationspfad mit freien Elektronen und Löchern, wie in Abbildung 5.5 a dargestellt. In den meisten Lagen stellt sich schnell wieder ein neutraler Zustand durch Rekombination von Löchern und Elektronen ein. Sobald das Partikel jedoch in den Verarmungsbereich (engl. depletion) eines Transistors im Substrat eindringt, wird die vorhandene elektrische Ladung des vordotierten Bereichs temporär trichterförmig verzerrt. Der Anstieg der Ladung passiert in Bruchteilen einer Nano-

sekunde [15, S. 50] und fällt anschließend auf ein Niveau ähnlich der Ausgangslage zurück.

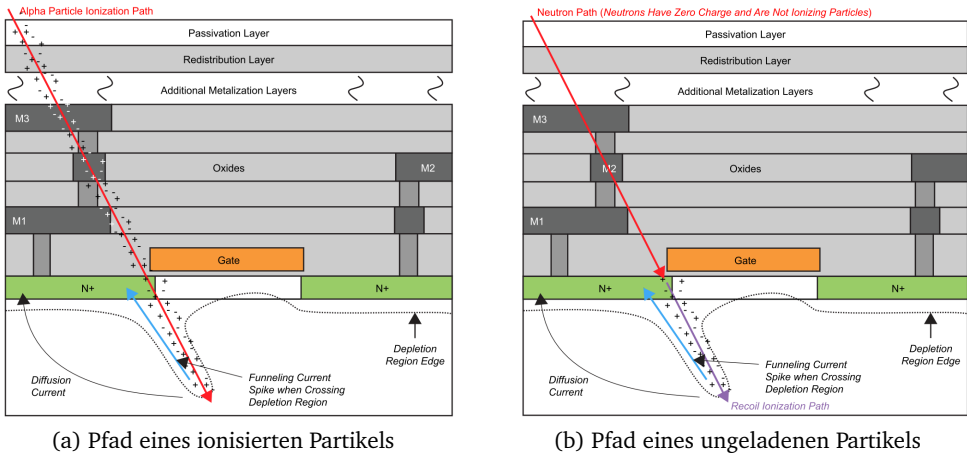


Abbildung 5.5: Funneling Effect in einem CMOS-basierten Bauteil [6]

In dem Fall eines Neutrons, kann es durch die Kollisionen im Substrat zu demselben Effekt der Trichterbildung kommen (Abbildung 5.5 b). Mögliche Auswirkungen auf ein System durch diesen temporären Anstieg der Ladung werden in Abschnitt 5.4.1 vorgestellt.

Displacement Effect/Verschiebungseffekt Ein weiterer Effekt ist der sogenannte *Displacement Effect*, übersetzt werden kann dieser Begriff mit Verschiebungseffekt. Wie der Name andeutet, beschreibt dieser Effekt Verschiebungen innerhalb der kristallinen Gitterstruktur im Material, verursacht durch Kollisionen mit Partikeln. In [154, S. 59] wird erwähnt, dass es zu einer automatischen Reparatur in 90 % der Fälle durch Rekombination innerhalb einer Minute kommt. Zusätzlich kann eine Reparatur der Gitterstruktur durch externe Erwärmung erfolgen. Ist dies nicht möglich, bleibt ein permanenter Defekt im Material.

Charge Accumulation Effect/Ladungskumulationseffekt Durch die bisher genannten Effekte, Funneling und Displacement Effects, kann es zu einer Anhäufung von Ladung kommen. Im englischen wird dieser Effekt durch den Begriff *Charge Accumulation Effect* beschreiben. Durch die zusätzliche Ladung kann die Charakteristik des Transistors maßgeblich verändert werden. Je nachdem wo im Material und wie lange ein Material der Strahlung ausgesetzt ist, führt die Ladungskumulation zunächst zu einem (anwachsenden) Verlust der Leistungsfähigkeit durch Veränderung der Schwellspannung, bis

hin zur Zerstörung des Transistors durch Erhöhung der Kriechströme. Weitere Details sind in [14] aufgeführt.

5.4 Kategorisierung der physikalischen Strahlungseffekte

Die physikalischen Effekte werden in der Literatur (z. B. [60],[15],[190],[57],[52] oder [14]) in zwei Kategorien unterteilt: *Single Event Effects* (SEEs) und *Cumulative Effects* (CE). SEEs modellieren Fehler, welche durch die Auswirkung eines einzelnen Partikels, typischerweise Schwerionen und Protonen, verursacht werden. Kumulative Effekte hingegen modellieren die Langzeit-Effekte durch den Ladungskumulations- und Verschiebungseffekt durch Elektronen und Protonen. Abbildung 5.6 stellt die Aufteilung der physikalischen Effekte dar.

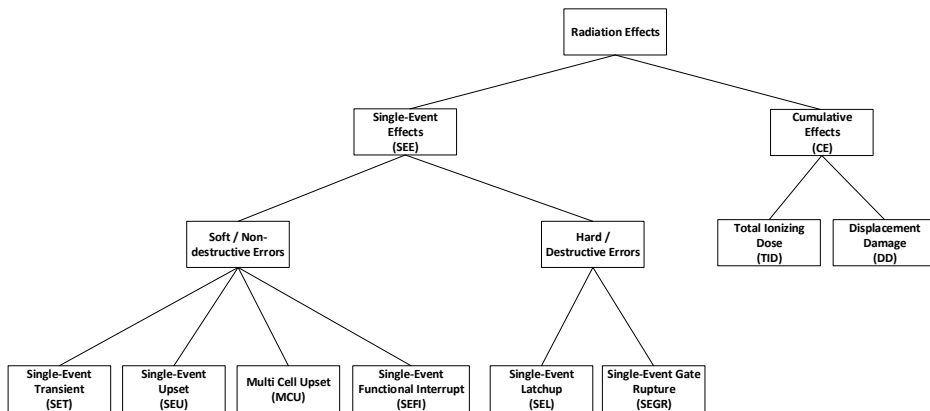


Abbildung 5.6: Kategorisierung der physikalischen Strahlungseffekte

5.4.1 Single-Event Effects (SEEs)

Single-Event Effects modellieren die Auswirkungen durch den Trichtereffekt, also Auswirkungen eines einzelnen, geladenen oder ungeladen, Partikels auf das Material. In Abhängigkeit von Ort, elektrischen Feldern und der Energie des Partikels kann es zu unterschiedlichen Verhalten kommen. Temporäre Fehler, sogenannte *soft errors* oder auch *non-destructive*, zu Deutsch nicht zerstörende SEEs [60, S. 28 ff.], haben nur eine bestimmte Zeitspanne einen Einfluss auf die Schaltung und müssen im schlimmsten Fall durch aus- und wieder einschalten des Bauteils (engl. power cycle) behoben werden. Permanente Fehler, sogenannte *hard errors*, bezeichnen Effekte, die irreparabel sind und somit ein Bauteil dauerhaft beschädigen. Synonym wird der englische Begriff *destructive*, zu Deutsch zerstörende SEEs, verwendet. In den folgenden Abschnitten

werden die einzelnen Effekte nur kurz beschrieben. Eine detaillierte Beschreibung der SEEs kann in [52] nachgeschlagen werden.

Single-Event Upset (SEU)

Ein *Single-Event Upset* (SEU) beschreibt den Effekt der Änderung eines Speicherinhalts und ist umgangssprachlich auch als Bit-flip bekannt. Synonym wird auch der Begriff *Single-Bit Upset* (SBU) verwendet. Durch den Funneling Effekt kann es, wie in Abschnitt 5.3 beschrieben, innerhalb weniger Piko- bis Nanosekunden zu einer zusätzlichen Ladung durch Elektronen-Löcher-Paare an kritischen Bereichen des Transistors kommen. Übersteigt die Ladung einen bestimmten Wert (Q_{crit} , [190]) kann dies zu einer Spannungsänderung in kritischen Knotenpunkten führen.

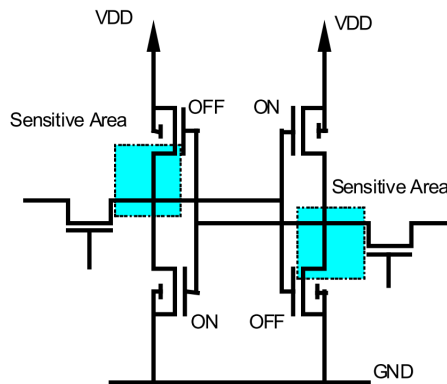


Abbildung 5.7: Kritische Bereiche für eine 6-Transistor-SRAM-Zelle [212, S. 29]

Ein Beispiel für diesen kritischen Bereich einer 6-Transistor-SRAM-Zelle ist in Abbildung 5.7 dargestellt. Durch die Rückkopplung wird ein Logikwechsel dauerhaft übernommen.

Multiple Cell Upset (MCU) und Multiple Bit Upset (MBU)

Ein *Multiple Cell Upset* (MCU) beschreibt den Effekt, dass sich zwei oder mehrere Speicherbereiche durch die Wechselwirkung mit einem Partikel ändern. Typischerweise sind die betroffenen Zellen physikalisch benachbart [60, S. 28 f.]. Abbildung 5.8 zeigt die Auswirkung eines MCUs in zwei unterschiedlichen Wörtern. Liegen zwei Fehler in einem Wort, so wird dieser Sonderfall *Multiple Bit Upset* (MBU) genannt (siehe Abbildung 5.9).

Durch die Miniaturisierung in der Halbleiter-Prozesstechnik wird der Abstand zwischen einzelnen Transistoren geringer, wodurch CMOS-basierte Bauteile anfälliger für MCUs werden.

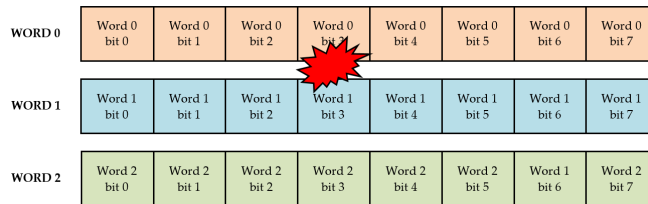


Abbildung 5.8: Zwei Upsets in zwei unterschiedlichen Wörtern (MCU) [60, S. 27]

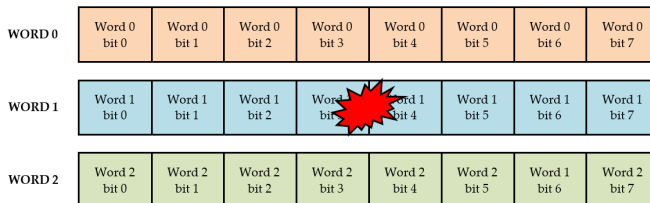


Abbildung 5.9: Zwei Upsets in einem Wort (MBU) [60, S. 28]

Single-Event Transient (SET)

Durch den Funneling Effekt kann es weiterhin zu einem fehlerhaften Spannungspuls (engl. *glitch*) am Ausgang des Transistors kommen, welcher *Single-Event Transient* (SET) genannt wird. Die Form eines SET hängt von verschiedenen Parametern ab: Zum einen von den Eigenschaften des Partikels selbst, dem LET und dem Einfallswinkel α , zum anderen von den Eigenschaften der Materialien in den verschiedenen Lagen und den vorherrschenden elektrischen Feldern.

Single-Event Functional Interrupt (SEFI)

Ein *Single-Event Functional Interrupt* (SEFI) [190] ist ein Begriff, der verschiedene, signifikante Änderungen der funktionalen Arbeitsweise eines Bauteils zusammenfasst. Ein SEFI wird typischerweise durch die Änderung eines wichtigen Kontrollregisters einer Systemkomponente verursacht. Beispiele sind ein Zurücksetzen (engl. *reset*) des Bauteils, die Initialisierung eines bestimmten Arbeitsmodus (z.B. *sleep mode*) oder das Herunterfahren (engl. *shutdown*) des Bauteils. Da die Fläche für solche kritischen Komponenten eines Systems/Bauteils sehr gering ist, ist die Wahrscheinlichkeit eines SEFIs sehr gering. Ein SEFI kann durch einen power cycle behoben werden.

Single-Event Latchup (SEL)

Ein *Single-Event Latchup* (SEL) [190] ist ein potentiell destruktiver Effekt. Durch ein einzelnes Partikel kann es zu parasitären p-n-p-n Strukturen im Halbleitermaterial kommen, wodurch große Ströme entstehen. In vielen Fällen kann dies zur Zerstörung

des Geräts führen, wenn das Gerät nicht frühzeitig ausgeschaltet wird. Ein Bauteil, das für den Einsatz in einer Umgebung mit Strahlung entwickelt wurde, wird explizit auf diesen Effekt getestet um einen potentiell katastrophalen Fehler zu vermeiden.

Single-Event Gate Rupture (SEGR)

Ein *Single-Event Gate Rupture* (SEGR) [15, S. 55 f.] ist ein destruktiver Fehler, der z. B. in nicht flüchtigen Speichertechnologien wie EEPROMs beobachtet wurde. Der Effekt beschreibt die Durchlöcherung der Isolierung des Transistors. Trifft ein Schwerion auf ein Transistor-Gate, während zusätzlich ein hohes elektrisches Feld anliegt (z. B. bei einem Schreibvorgang), erzeugt das Schwerion einen stark leitfähigen Pfad zwischen Gate und Substrat. Auf dem Pfad durch die Isolierungsschicht kommt es zu Entladungen, die das Halbleitermaterial stark aufheizen und zum Schmelzen führen können.

5.4.2 Kumulative Effekte

Im Gegensatz zu den Single-Event Effects, bei denen die Auswirkungen eines einzelnen Partikels betrachtet werden, fassen kumulative Effekte die Langzeiteffekte von Strahlung, Ladungskumulation(5.3) und Verschiebung(5.3), zusammen.

Total Ionizing Dose (TID)

Der Begriff *Total Ionizing Dose* (TID) fasst die Effekte der Ladungskumulation zusammen. In [14] wird der Begriff TID definiert, als Strahlungsdosis, die ein Bauteil toleriert, bis dessen (Hardware-) Spezifikation/Charakteristik nicht mehr eingehalten werden kann. Der Wert wird in der SI Einheit Gray (Gy, $1 \text{ Gy} = 1 \text{ J/kg}$) oder rad ($100 \text{ rad} = 1 \text{ Gy}$) gemessen. In Abhängigkeit von dem Orbit und der Abschirmung des Bauteils, liegt der Wert zwischen wenigen krad und mehreren 100 krad [60, S. 26].

Displacement Damage (DD)

Der *Displacement Damage* (DD) Effekt fasst die physikalischen Effekte der Verschiebung in der kristallinen Gitterstruktur zusammen. Bleiben Defekte in dieser zurück, kann es zu unterschiedlichen Effekten in bipolaren Bauteilen, wie zum Beispiel LEDs oder Laserdioden, kommen [60, S. 26].

5.5 Auswirkung der Strahlungseffekte auf FPGAs

Moderne FPGAs integrieren eine Vielzahl unterschiedlicher Komponenten [190]: Programmierbare LUTs, Register, interner Blockspeicher (BRAM), programmierbare Prozessoren, Ein- und Ausgangsblöcke, Multi-Gigabit-Transceiver, digitale Signalprozessoren (DSPs), PCI-Express-Schnittstellen, Taktmanagementblöcke etc.. In diesem Abschnitt werden konkrete Auswirkungen der zuvor beschriebenen physikalischen Effekte auf diese Teilkomponenten aufgeführt. Insbesondere wird auf die Anfälligkeit des Konfigurationsspeichers, des internen Blockspeichers, der Register und des Konfigurationsprozessors eingegangen.

5.5.1 SEU/MCU/MBU Effekte

SEUs oder MCUs können alle Speicherelemente eines Bauteils betreffen [15, S. 59 f.]. Die Speicherelemente in einem FPGA gehören entweder zum Konfigurationsspeicher oder sind frei für den Benutzer verfügbar. Zu dem Konfigurationsspeicher zählen dabei ebenfalls die verschiedenen Komponenten des Konfigurationsprozessors: Interne Register und Flip-Flops des Zustandsautomaten. Zu dem frei verfügbaren Speicher gehören die Register (z.B. Flip-Flops oder Schieberegister), der eingebettete Speicher (Block RAM), aber auch die Register und Speicherelemente von eingebetteten Hard-IP-Cores (z.B. CPUs oder DSPs), welche für ein Design verwendet werden können.

Konfigurationsspeicher

Ein Upset im Konfigurationsspeicher kann zu mehreren Fehlern in der Schaltung führen, da der Fehler üblicherweise erst bei einem Austausch der Konfiguration wieder überschrieben wird. Ein Upset eines einzelnen Bits kann dabei zu einer Änderung in der Verdrahtung oder der Logikfunktion (realisiert durch eine LUT) in einer Schaltung führen. Abbildung 5.10 visualisiert diese beiden möglichen Einflüsse auf ein Design. Ein Upset in dem Konfigurationsprozessor kann weiterhin zu einer falschen Konfiguration oder zu einem SEFI führen (siehe Abschnitt 5.4.1).

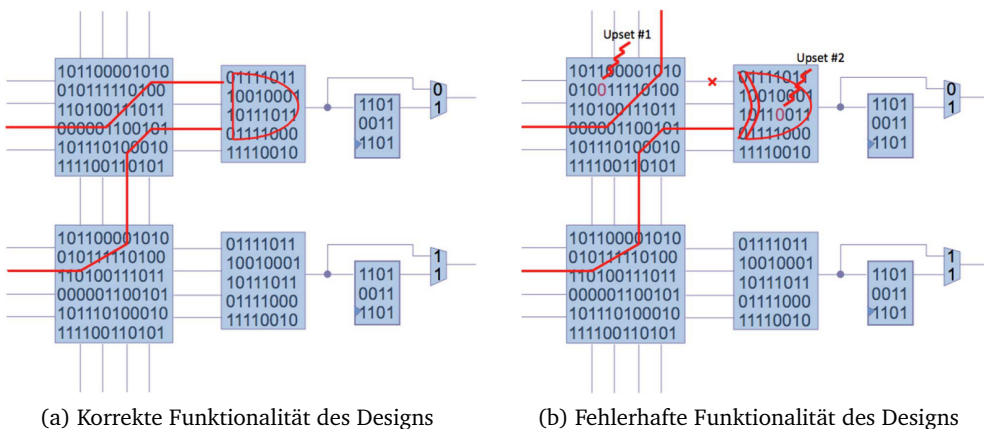


Abbildung 5.10: Einfluss eines Upsets auf die Logik oder Verdrahtung eines Designs [190, S. 384]

Die Art des Konfigurationsspeichers eines FPGAs - SRAM, Flash oder Antifuse (siehe Abschnitt 2.3) - bestimmt die Anfälligkeit für Upsets. Antifuse-basierte FPGAs sind durch die feste Programmierung gegen Upsets immun [120]. Flash-basierte Konfigurationsspeicher sind ebenfalls immun [15, 121]. Der Grund dafür liegt laut Battezzati et al. in der

Dimensionierung der Speicherzellen. Im Vergleich zu reinen Flash-Speicher-Bauteilen, sind die Zellen deutlich größer dimensioniert. Mit fortschreitender Miniaturisierung könnten jedoch auch Flash-basierte Konfigurationsspeicher von Upsets betroffen sein. SRAM-basierte FPGAs sind sehr anfällig für Upsets, wie bereits in Abschnitt 5.4.1 dargestellt. Xilinx erwähnt in mehreren Dokumenten (z. B. [201], [86] oder [114]), dass die SRAM-Zellen des Konfigurationsspeichers, analog zu den Flash-basierten FPGAs, größer dimensioniert und robuster sind als SRAM-Zellen in herkömmlichen Speicherbauteilen, welche auf Geschwindigkeit und Kosten optimiert sind. Zitat aus [114]:

The design criteria for these static latches have historically been and continue to be stability, stability, and stability.

Diese Aussage wird bei der Betrachtung der Entwicklung der SEU-Anfälligkeit von Xilinx FPGAs gestützt (siehe Abschnitt 5.9.1).

Benutzerspeicher

Ein Upset in dem Benutzerspeicher (Blockspeicher oder Register) kann dazu führen, dass ein falscher Wert an die verarbeitende Logik weitergegeben wird. Dies ist nur dann der Fall, wenn die betroffene Speicherzelle in dem aktuellen Design verwendet wird.

Register werden darüber hinaus zur Implementierung von wichtigen und kritischen Teilkomponenten wie Zustandsautomaten, Zählern oder Synchronisierern verwendet [190]. Speichert ein betroffenes Register den Zustand eines Zustandsautomaten, ist dieser nicht mehr kohärent zur restlichen Schaltung, produziert falsche Ausgabewerte oder führt im schlimmsten Fall zu einer (System-) Blockierung (engl. deadlock). Das Fehlverhalten kann nur durch Zurücksetzen des Zustandsautomaten gelöst werden.

Der Blockspeicher ist z. B. bei Xilinx SRAM-basierten FPGAs deutlich anfälliger gegenüber Upsets. In [83] wird ein Faktor 5 im Vergleich mit den Speicherzellen des Konfigurationsspeichers genannt. Begründet ist die höhere Anfälligkeit in den höheren Anforderungen bzgl. der Verarbeitungsgeschwindigkeit. FPGAs, die Flip-Flops oder (nicht Flash-basierten) eingebetteten Speicher verwenden, sind gleichermaßen von Upsets betroffen. Daher sind alle drei FPGA-Typen bezüglich des Benutzerspeichers Upset-anfällig.

Mehrbitfehler

Die Anzahl von Mehrbitfehlern, verursacht durch einen SEE, wird durch die Miniaturisierung in der Fertigungstechnologie, unabhängig von der Speicherkategorie, wahrscheinlicher. Strahlungstests an verschiedenen Xilinx Virtex Familien belegen dies. In [139] wird anhand von Tests an Virtex-, Virtex-II-, Virtex-4- und Virtex-5-FPGAs die Entwicklung von Mehrbitfehlern betrachtet. Abbildung 5.11 zeigt den prozentualen Anteil der Ein- und Mehrbitfehler. Während in Virtex-FPGAs noch über 90 % der Events in Einbitfehler resultieren, sind Ein- und Zweibitfehler in Virtex-4 FPGAs bereits gleichstark vertreten (ca. 45 %). In der Virtex-5-Familie nimmt der Anteil der Einbitfehler

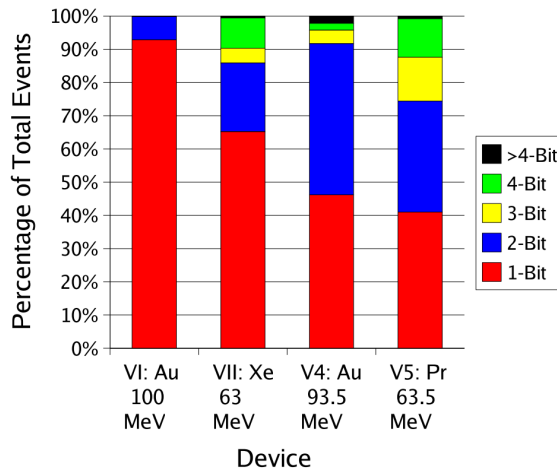


Abbildung 5.11: Entwicklung der Ein- und Mehrbitfehler in unterschiedlichen Xilinx FPGA-Familien [139, S. 7]

weiter ab. Nur noch ca. 40 % der Events bilden sich als Einbitfehler aus. Das heißt auf der anderen Seite, dass der Anteil der Mehrbitfehler bei ca. 60 % liegt. 25 % der Events resultieren dabei in Drei- und Vierbitfehler.

Xilinx wirkt diesem Trend ab der Virtex-6-Familie entgegen und reduziert den Effekt von Mehrbitfehlern durch die Verschachtelung der physikalischen Konfigurationsspeicherbits (engl. bit-interleaving) [86]. Der FPGA-Hersteller Microsemi verwendet dieses Prinzip ebenfalls ab der RTG4-Familie, um den Einfluss von Mehrbitfehlern zu minimieren [123, S. 7].

5.5.2 SET Effekte

Der Spannungspuls durch ein SET kann verschiedene Auswirkungen auf ein System haben. Im schlimmsten Fall kann dieser zu einem Fehlverhalten im System führen, wenn er zum Beispiel von Speicherelementen detektiert und gespeichert wird. In diesem Fall verhält sich ein SET wie ein SEU. In [57] wird dieses Beispiel detailliert. Abbildung 5.12 zeigt exemplarisch ein Design und Bedingungen, die erfüllt sein müssen damit ein SET zu einem fehlerhaften Verhalten des Designs führt. Die Pulsweite eines SETs ist abhängig von mehreren Faktoren, insbesondere der Strukturgröße und der Prozesstechnologie, dem Wirkungsquerschnitt sowie der Partikelart. Allgemein sind SETs Subnanosekundenereignisse. Laut Mavis et al. ist die Pulsweite in EEPROM-basierten Technologien um eine Größenordnung länger, sodass Flash-basierte FPGAs anfälliger gegenüber SETs sind [119].

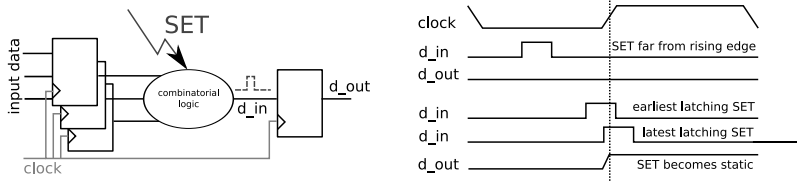


Abbildung 5.12: Bedingungen zur Speicherung eines SETs in einem Register [57, S. 30]

Im linken Teil der Abbildung ist ein Teil einer Schaltung, in dem ein SET auftritt, (abstrahiert) dargestellt. Der rechte Teil zeigt das Wellendiagramm mit drei möglichen Zeitverläufen des SETs auf dem Speichereingang d_{in} , sowie die Auswirkung auf den Ausgang d_{out} . Unkritisch ist ein SET, der nicht synchron zu der steigenden Taktflanke an einem Speichereingang ankommt. Fällt der SET jedoch in die Nähe der steigenden Taktflanke, kann der fehlerhafte Wert übernommen werden. In diesem Fall wird der SET statisch und verhält sich wie ein SEU. Werden die Setup- und Hold-Zeiten nicht eingehalten, kann es zu unvorhersagbaren oder metastabilen Ausgaben am Ausgang kommen. Anhand dieses Beispiels wird ebenfalls der Einfluss der Taktfrequenz auf die SET-Sensibilität eines Systems verdeutlicht: Mit wachsender Taktfrequenz steigt die Wahrscheinlichkeit einen SET zu speichern.

Weitere Beispiele für SET-sensible Systemkomponenten sind zum Beispiel die Takt- oder Reset-Leitungen. Ein SET auf diesen Leitungen kann zu de-synchronisierten oder fälschlich zurückgesetzten Systemkomponenten führen.

5.5.3 SEL und SEGR Effekte

Diese Effekte führen wie in den Abschnitten 5.4.1 und 5.4.1 beschrieben in der Regel zur (teilweisen) Zerstörung des Bauteils. Moderne FPGAs bestehen aus einer Vielzahl unterschiedlicher Komponenten, wie Prozessoren oder Multi-Gigabit-Transceivern. Um eine Immunität gegen diese destruktiven Effekte gewährleisten zu können, müssen all diese Einzelkomponenten immun gegenüber SEL und SEGR sein. Konkrete Angaben sind lediglich für den SEL-Effekt für spezielle strahlungsgehärtete oder -tolerante FPGAs verfügbar (siehe Abschnitt 5.8, und Tabelle 5.6). Die Anfälligkeit ist hierbei für alle FPGA-Typen in einer vergleichbaren Größenordnung.

5.5.4 Kumulative Effekte

Bezüglich der kumulativen Effekte, sind FPGAs und ASICs nicht von *Displacement Damage* Effekten betroffen [60, S. 26]. Durch die TID Effekte hingegen, kann es zu drei Auswirkungen [15, S. 59 f.] kommen: Performanceverlust, Anstieg der Verlustleistung und Verlust der Programmierbarkeit des Bauteils. Eine Ladungskumulation in

der Oxidschicht eines MOS-Bauteils hat zunächst eine Änderung der Schwellspannung zur Folge, welche zu dem genannten Performanceverlust führt. Die höheren Kriechströme führen weiterhin zu einem Anstieg der Verlustleistung. Die dritte Auswirkung beschreibt den Verlust der Programmierbarkeit. In Transistoren, die eine Speicherzelle aufbauen, kann es durch die zusätzliche Ladung und der damit verbundenen Schwellspannungsänderung zu dem Verlust der Rekonfigurierbarkeit kommen.

Flash-basierte FPGAs sind aufgrund ihrer Bauform deutlich anfälliger als SRAM-basierte FPGAs und haben somit einen geringeren TID-Schwellwert [176]. Eine Flash-Speicherzelle basiert auf einem *Floating-Gate Metal-Oxide-Semiconductor Field-Effect-Transistor* (FG-MOSFET), welcher aus einem MOS-Transistor mit zwei überlappenden Gates besteht. Das erste Gate ist dabei vollständig in einer Oxid-Schicht (engl. tunnel oxide) vergraben, während das zweite den eigentlichen Gate-Anschluss bildet. Elektronen können in der Tunneloxidschicht des isolierten Gates eingefangen werden und ändern somit die Charakteristik des Transistors maßgeblich. In der englischsprachigen Literatur wird dieser Effekt *hole trapping* genannt. Neben TID und Herstellungsfehlern kann dieser Effekt auch durch sogenannte *Hot Carrier Injection* verursacht werden. Diese beschreibt die elektronische Beanspruchung des Transistors durch das zyklische Programmieren bzw. Löschen und damit einhergehende Schäden in der Oxidschicht. Laut Urbina-Ortega et al. führt dies in einem Flash-basierten FPGA zu einer kontinuierlichen Herabsetzung der Leistungsfähigkeit (Taktfrequenz). Durch TID steigt zusätzlich die Sensitivität gegenüber SEEs. Ein Beispiel dafür sind sogenannte *stuck bits*: Ein SEU in einer Speicherzelle, der nicht korrigiert werden kann.

Zusammenfassung

Tabelle 5.2 fasst den Einfluss der Strahlungseffekte auf die unterschiedlichen FPGA-Technologien zusammen. Die Antifuse-Technologie bietet hierbei die höchste Robustheit. Flash-basierte FPGAs bieten durch die Dimensionierung der Transistoren für den Konfigurationsspeicher ebenfalls eine hohe Robustheit gegenüber SEU-, MCU- und MBU-Effekte, sind jedoch anfällig gegenüber SET- und TID-Effekte. In der SRAM-Technologie ist insbesondere die Anfälligkeit gegenüber SEU-, MCU- und MBU-Effekte zu nennen.

Tabelle 5.2: Anfälligkeit der FPGA-Technologien gegenüber Strahlungseffekte

FPGA-Technologie	Single-Event Effekte			Kumulative Effekte	
	SEU, MCU, MBU	SET	SEL ¹ , SEGR ¹	DD ¹	TID
Antifuse	++	o	o	+	o
Flash	+	-	o	+	--
SRAM	-	o	o	+	-

¹ Diese Effekte sind mehr von dem Packaging als der Speichertechnologie abhängig.

5.6 Abschwächungsverfahren zur Behandlung der Strahlungseffekte

Im Laufe der Jahre wurde eine Vielzahl unterschiedlicher Techniken zur Minderung der Auswirkungen der Strahlungseffekte entwickelt, welche im Verlauf als Abschwächungsverfahren (engl. mitigation techniques) bezeichnet werden. Etablierte Kategorien dieser Verfahren sind hierbei Redundanzverfahren, Verfahren zur Fehlererkennung und Fehlerkorrektur sowie Speicher-Scrubbing. Bevor auf diese Verfahren näher eingegangen wird, wird zunächst die Fehlerkette, welche zu einem letztlichen Ausfall eines Systems führt, sowie existierende Fehlermodelle zur Kategorisierung der Verfahren vorgestellt.

5.6.1 Fehlermodelle

In der Literatur, z. B. dem *Fault Management Handbook* der NASA (National Aeronautics and Space Administration) [127] oder dem ISO-Standard für sicherheitsrelevante elektrische/elektronische Systeme in Kraftfahrzeugen (ISO 26262) wird die Fehlerkette in drei Glieder unterteilt: Defekt (engl. fault), Fehler (engl. error) und Ausfall (engl. failure). Der ISO-Standard 26262 beschreibt diese drei Glieder in einem Wörterbuch (ISO 26262-1). Zitat aus [184]:

- Fault: Abnormal condition that can cause an element or an item to fail.
- Error: Discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition.
- Failure: Termination of the ability of an element to perform a function as required.

In [15, S. 43 f.] wird diese Definition konkreter für den Einsatz in mikroelektronischen Systemen formuliert: Ein Defekt ist definiert als das Fehlverhalten einer einzelnen Komponente in einem System. Wird diese Komponente in dem System aktiv, so kann der Defekt zum Ausgang der Komponente propagieren, wodurch der Defekt zu einem Fehler wird. Wird dieser Fehler weiter zu dem Ausgang des Systems propagiert und führt somit zu einem Fehlverhalten des Systems, wird der Fehler zu einem Ausfall.

NASA: Fault Management

Ausfälle können laut dem *NASA Fault Management Handbook* [127, S. 145 ff.] entweder verhindert (engl. Failure Prevention) oder toleriert (engl. Failure Tolerance) werden. In [155] werden diese beiden Kategorien weiter ausgeführt. Maßnahmen zur Ausfallverhinderung können entweder zur Systemlaufzeit (engl. Operational Failure Avoidance) oder zur Entwurfszeit (engl. Design Time Fault Avoidance) erfolgen. Bei der Verhinderung zur Entwurfszeit werden Funktionen zur Minimierung der Risiken eines Defekts und daraus resultierende Ausfälle integriert. Eine Ausfallverhinderung zur Laufzeit hingegen prognostiziert, dass ein Ausfall in der Zukunft passieren wird und führt Maßnahmen zur Verhinderung ein. Bei der Fehlertoleranz können Ausfälle entweder akzeptiert und abgeschwächt werden. Das Maskieren von Ausfällen (engl. Failure Masking) erlaubt einen Ausfall, solange die Effekte keine Auswirkungen auf die

Funktion der nächst höheren Komponente haben. Dem gegenüber stehen Verfahren zur Wiederherstellung (engl. Failure Recovery), welche eine temporäre Schädigung der Systemfunktion erlauben, jedoch auf den Ausfall reagieren bevor ein Missionsziel gefährdet ist. Ein letztes Verfahren beschreibt die Anpassung der Missionsziele (engl. Goal Change). Die, folgend vorgestellten Verfahren können entsprechend dieser Kategorisierung, wie in Abbildung 5.13 dargestellt, eingeordnet werden.

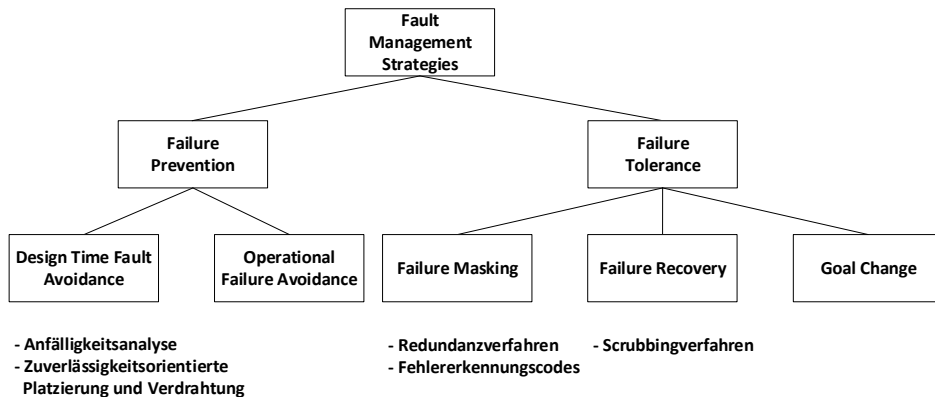


Abbildung 5.13: Kategorisierung der Verfahren nach dem NASA Fault Management Modell (basierend auf [155, S. 8])

ESA: Techniques for Radiation Effects Mitigation

Die Europäische Weltraumorganisation ESA stellt in dem *Technique for radiation effects mitigation in ASICs and FPGAs handbook* [60, S. 31 ff.] ein weiteres Modell vor, in welchem die Verfahren zur Behandlung von Strahlungseffekten auf vier verschiedenen Abstraktionsebenen eingeordnet werden:

1. Elektronische Systemebene (ESE): Techniken auf Bauteil- oder Softwareebene
2. Schaltungstechnische Architekturebene (SAE): Spezifische Techniken in der Schaltung
3. Physikalische Layoutebene (PLE): Techniken zur Optimierung von Lage und Dimension von Transistoren
4. Fertigungstechnikebene (FTE): Techniken die den Fertigungsprozess betreffen

Abbildung 5.14 stellt die genannten Ebenen dar und definiert auf der rechten Seite zwei Akronyme: RHBD (Radiation Hardening By Design) und RHBP (Radiation Hardening By Process). Im Vergleich mit dem NASA-Modell ermöglicht das ESA-Modell durch die letzten beiden Ebenen die Erfassung von Verfahren vor der Fertigung des FPGAs. Auf Verfahren in diesen letzten beiden Ebenen wird jedoch nicht weiter eingegangen. In [60] werden besagte Verfahren in den Einzelabschnitten *Layout, Analoge*

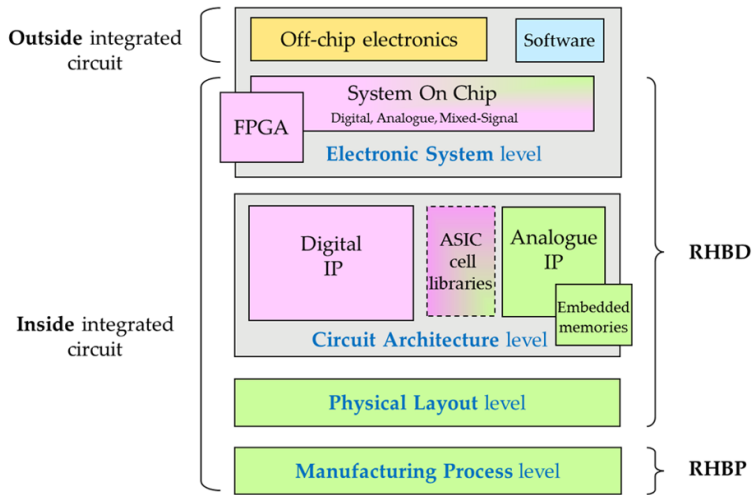


Abbildung 5.14: Vier Abstraktionsebenen der Verfahren zur Erhöhung der Fehlertoleranz [60, S. 33]

Schaltungen, Eingebettete Speicher und Strahlungsgehärtete ASIC Bibliotheken auf etwa 50 Seiten vorgestellt. FPGA-Familien, in denen diese Verfahren angewendet wurden, werden in Abschnitt 5.8 aufgeführt.

Festgehalten wird, dass es keine allgemeingültige Lösungsformel zur Erhöhung der Fehlertoleranz des Systems gibt. Vielmehr müssen bei der Wahl der Verfahren die technischen Anforderungen und Ziele den finanziellen Rahmenbedingungen und der Entwicklungszeit gegenübergestellt werden.

5.6.2 Redundanzverfahren

Verfahren welche auf der schaltungstechnischen Architekturebene, der elektronischen Systemebene und der physikalischen Layoutebene eingesetzt werden, sind sogenannte Redundanzverfahren [60, S. 106 ff.]. Die Verfahren teilen sich in der Regel in räumliche oder zeitliche Redundanzverfahren auf, wobei eine Kombination beider Verfahren ebenfalls möglich ist.

Räumliche Redundanz

Bei der räumlichen Redundanz, auch Hardware Redundanz genannt, werden einzelne Komponenten repliziert und eine parallele Verarbeitung mit den gleichen Eingangsdaten durchgeführt. Die Ausgaben werden anschließend verglichen um eine Fehlererkennung oder Fehlerkorrektur durchführen zu können. Abbildung 5.15 zeigt die allgemeine Architektur einer räumlichen Redundanz. Der Vergleich am Ende ist dabei der kritische

Teil dieser Architektur. Typischerweise werden, basierend auf der Anzahl der replizierten Module, zwei Architekturen verwendet: Duplex- und Triple-Architekturen.

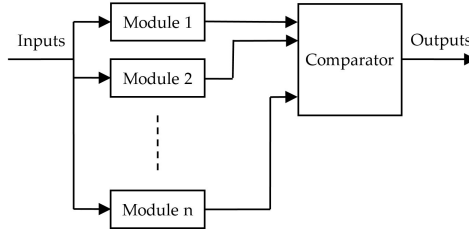


Abbildung 5.15: Blockdiagramm einer Architektur mit räumlichen Redundanz [60, S. 107]

Duplex-Architekturen

Duplex-Architekturen verwenden zwei Module, sogenannte *Dual Modular Redundancy* (DMR) oder auch *Duplicate With Compare* (DWC) genannt, und dienen lediglich der Fehlererkennung. Das Verfahren kann sowohl auf kombinatorischer als auch sequenzieller Logik angewendet werden und ermöglicht die Erkennung von SET- und SEU-Fehlern, wie in Abbildung 5.16 dargestellt.

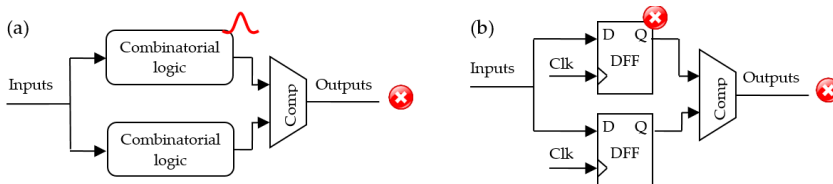


Abbildung 5.16: Duplex-Architekturen in kombinatorischer und sequenzieller Logik [60, S. 107]

In der Regel wird der Komparator, wie in Abbildung 5.17 (a) dargestellt, realisiert. Im Fall eines detektierten Unterschieds (b), wird ein Fehler an die nächsthöhere Komponente weitergegeben. Diese hat dann, je nachdem wie kritisch die Ausgangswerte sind, zwei Möglichkeiten zur Fehlerbehandlung: Verwerfen der fehlerhaften Daten und Fortsetzen der Verarbeitung oder erneute Datenverarbeitung mit den gleichen Eingangsdaten.

Triple-Architekturen

Um eine Fehlerkorrektur durchführen zu können, müssen mindestens drei Module verwendet werden. In der englischsprachigen Literatur wird das Verfahren mit drei

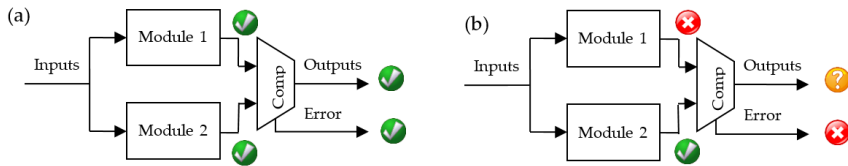


Abbildung 5.17: Fehlererkennung in einer Duplex-Architektur [60, S. 107]

Modulen *Triple Modular Redundancy* genannt. Die grundsätzliche Architektur besteht, wie in Abbildung 5.18 dargestellt, aus drei redundanten Modulen, dessen Ausgänge zu einem Majoritätsvergleich oder Komparator (*Voter*) weitergeleitet werden. Ein fehlerhafter Ausgangswert durch einen SEU oder SET kann dabei im Idealfall durch die zwei fehlerfreien Ausgangswerte erkannt werden.

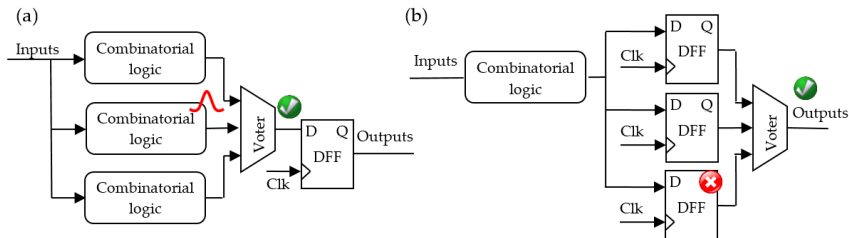


Abbildung 5.18: TMR-Architekturen in kombinatorischer und sequenzieller Logik [60, S. 109]

Diese Variante wird im Falle einer Verdreifachung der Register *Lokales TMR* genannt. Eine konkrete Anwendung findet dieses Verfahren z.B. auf Transistorebene (PLE) bei den Microsemi RTAX-S und RTSXS FPGAs (siehe Abschnitt 5.8.3). Bei diesen FPGAs wird die Fehlertoleranz aller Flip-Flops des FPGAs durch lokales TMR erhöht. Das Verfahren kann auch auf der Schaltungsebene (SAE) zur Erhöhung der Fehlertoleranz der Flip-Flops eines Standard-FPGAs eingesetzt werden.

Die, in Abbildung 5.18, dargestellte Architektur hat wesentliche Schwachstellen. Ein SET in dem Schaltungsstück des Komparators kann in beiden Architekturen (a) und (b) zu einem falschen Ausgangswert führen. Weiterhin kann in der sequenziellen Architektur (b) ein SET in der kombinatorischen Logik nicht erkannt werden, da alle Speicherelemente denselben, falschen Wert übernehmen würden. Zusätzlich könnten die Inhalte der Register durch SEUs zu einem akkumulierten Fehler führen.

Diese Schwachstellen können durch ein *vollständiges TMR* (engl. full TMR) behoben werden. Hierbei erfolgt eine Verdreifachung aller, in Abbildung 5.18 dargestellten, Komponenten: Kombinatorische Logik, Register und Komparatoren. Der letztgenannte Schwachpunkt kann durch eine Rückkopplung des Komparatorausgangs zum Register-

eingang behoben werden. Abbildung 5.19 (a) zeigt den Einfluss eines SEUs in einem Flip-Flop in einer vollständigen TMR-Architektur. Wie dargestellt, erzeugt der SEU einen fehlerhaften Wert am Ausgang des betroffenen Flip-Flops. Durch die zwei fehlerfreien Werte der anderen Flip-Flops können die drei Komparatoren dennoch den korrekten Wert bestimmen. Durch die Rückkopplung wird weiterhin der fehlerhafte Wert in dem Flip-Flop überschrieben und somit automatisch korrigiert.

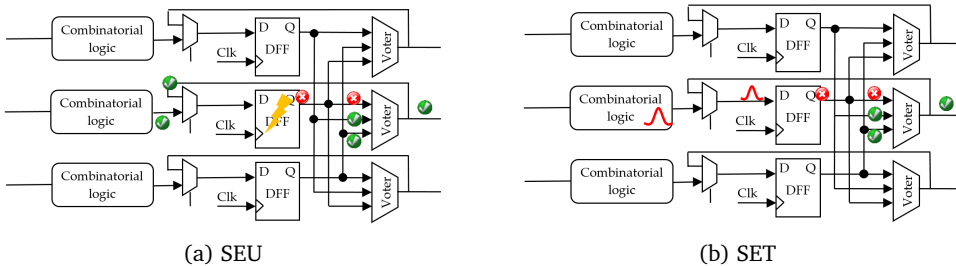


Abbildung 5.19: Fehlererkennung und -korrektur in einer vollständigen TMR-Architektur [60, S. 110]

In Abbildung 5.19 (b) wird Verhalten eines SETs in der kombinatorischen Logik in einer vollständigen TMR-Architektur dargestellt. Der SET propagiert zunächst durch die kombinatorische Logik und wird zuletzt in einem Flip-Flop gespeichert. Eine Fehlererkennung und Reparatur erfolgt analog zu dem SEU-Fehlerfall. Durch die Verdreifung der Komparatoren kann die nachfolgende Komponente ebenfalls einen Fehler auf einem der drei Komparatorausgängen detektieren.

Neben der Anwendung auf der schaltungstechnischen Architekturebene, also in einer Netzliste einer Schaltung, kann die gesamte Schaltung auf Systemebene als Modul gesehen und mit DWC oder TMR erweitert werden [74]. Dies bringt den Vorteil einer einfachen Implementierung, jedoch den Nachteil einer fehlenden, automatischen Synchronisierung der drei Module nach einer Reparatur, wie sie innerhalb einer Schaltung möglich ist.

Alle aufgeführten räumlichen Redundanz-Architekturen teilen die gleichen Nachteile. Der offensichtlichste Nachteil ist der erhöhte Flächenbedarf, der je nach Verfahren einen Faktor von minimal 2 bzw. 3 ausmacht. Mit dem erhöhten Flächenbedarf geht eine entsprechende erhöhte Verlustleistung einher. Zusätzlich kommt es aufgrund der zusätzlichen Komparatorlogik zu Takt-/Geschwindigkeitseinbußen im System.

Zeitliche Redundanz

Bei dem Verfahren der zeitlichen Redundanz werden die Daten zu (geringfügig) unterschiedlichen Zeitpunkten aufgenommen und anschließend verglichen. Eine unter-

schiedliche Datenaufnahme (engl. sampling) kann durch mehrere parallele Instanzen ermöglicht werden und kombiniert somit die zeitliche und räumliche Redundanz. Die Granularität der Instanzen kann dabei von Registern bis zu vollständigen Modulen variieren [60, S. 112 f.].

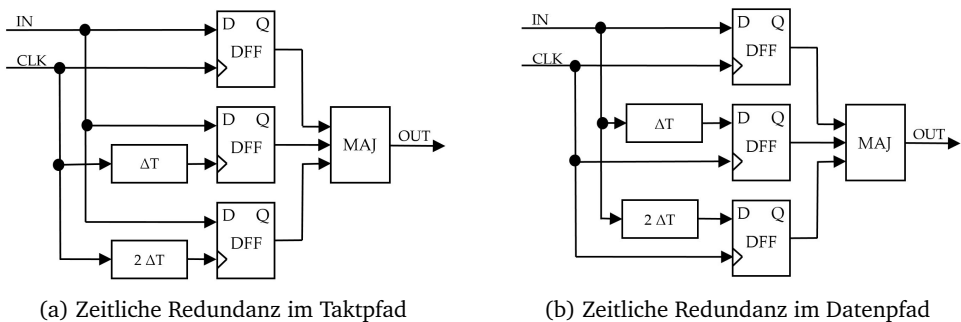


Abbildung 5.20: Architekturen mit zeitlicher Redundanz [60, S. 113 f.]

Auf der Schaltungsebene (SAE) kann eine Verzweigung zu den parallelen Instanzen im Taktpfad oder im Datenpfad erfolgen. Die Abbildungen 5.20 zeigen beide Fälle. Der Versatz (ΔT) sollte dabei so dimensioniert werden, dass dieser größer als die Pulsbreite eines möglichen SETs ist. Nur dann ist eine Abschwächung eines SETs möglich.

Die in den Abbildungen 5.20 dargestellten (internen) Modifikationen haben einen Einfluss auf die Fläche. So wird in [118] von einem Faktor 4 gegenüber konventioneller Flip-Flops gesprochen. Da der Anteil von Flip-Flops in einem Chip weiterhin zwischen 20 % und 40 % liegt, relativiert sich der Einfluss durch das Hinzufügen der zeitlichen Redundanz zu einem Gesamtfaktor zwischen 1,4 und 1,8. Wie bereits in Abschnitt 5.5.2 erwähnt, ist der Einfluss von SETs auf ein Design mit niedriger Taktfrequenz deutlich geringer und die SEU-Effekte dominieren. Das Verhältnis ändert sich jedoch mit steigender Taktfrequenz. Weiterhin muss die Taktfrequenz durch die zusätzlich hinzugefügten Verzögerungspfade verringert werden. Am Beispiel der Taktfrequenzen von 50 MHz und 500 MHz und einer Verzögerung von 200 ps wird der Einfluss deutlich gemacht. Während die Taktfrequenz bei einem mit 50 MHz betriebenen Design um lediglich 2 % gesenkt werden muss, liegt die Herabsetzung der Taktfrequenz bei einem mit 500 MHz betriebenen Design bei 20 %.

Anwendung findet das Verfahren der zeitlichen Redundanz zum Beispiel in den Flip-Flops des LEON2-FT Prozessors und in Bauteilen, die diesen IP-Core integrieren (z. B. Atmel AT697). Die zeitliche Redundanz wurde in dem Prozessor auf dem Taktpfad der TMR-Flip-Flops implementiert. Weiterhin wurde die Fehlertoleranz der DSP-Blöcke in FPGAs der Hersteller Xilinx (Virtex-5QV) und Microsemi (RTAX-4000D) durch dieses Verfahren erhöht.

5.6.3 Fehlererkennung und -korrektur

Neben der Redundanz auf zeitlicher und räumlicher Ebene, kann Redundanz in die zu übermittelnden Daten integriert werden. Kodierungen zur Fehlerkorrektur (engl. Error-Correcting Codes, ECCs) fügen zusätzliche Daten, wie Paritätsdaten oder Prüfsummen, zu den eigentlichen Daten hinzu, um eine fehlerhafte Übertragung der Daten erkennen zu können [60, S. 123 f.]. In dem Fall, dass die fehlerhaften Daten erkannt und korrigiert werden können, wird von Fehlerkorrekturverfahren (engl. Error Detection And Correction, EDAC) gesprochen. Fehlerkorrekturverfahren wurden zum Beispiel zur Reduzierung der Fehlerrate in Speichermedien wie CDs, DVDs oder RAMs eingesetzt. Weiterhin existieren verschiedene Verfahren mit unterschiedlichen Charakteristika im Bezug auf die Fehlererkennung und Fehlerkorrektur. Allgemein ist eine Aufteilung in zwei Kategorien möglich: Blockcode und konvolutioneller Code (auch Faltungscodcode). Dabei werden letztgenannte Kodierungen hauptsächlich zum Datentransfer in Videos, der Mobil- oder auch Satellitenkommunikation verwendet. Blockcodes werden hingegen für die Sicherung von Speichermedien eingesetzt.

Tabelle 5.3: Fehlerkorrekturkodierungen und exemplarische Anwendungen (basierend auf [60, S. 123])

Verfahren	Erkennung	Korrektur	Anwendung
Paritätsbits	x		RS-232-Standard
CRC	x		Netzwerk (Ethernet)
BCH-Code	x	x	Datensicherung bei DVB-T
Hamming-Code	x	x	Datensicherung z.B. Festplatte
Reed-Solomon-Code	x	x	Fehlerkorrektur bei Audio-CD

Tabelle 5.3 zeigt typische Fehlerkorrekturverfahren und exemplarische Anwendungen. Das Akronym CRC steht für eine zyklische Redundanzprüfung (engl. Cyclic Redundancy Check, CRC), das Akronym BCH fasst die Wissenschaftler die diese Kodierung entwickelt haben zusammen (Bose, Chaudhuri und Hocquenghem).

Bei der Auswahl eines Fehlerkorrekturverfahrens für eine konkrete Anwendung müssen die jeweiligen Bedingungen beachtet werden. So gibt es nicht die eine Fehlerkorrekturkodierung die eine allgemeingültige Lösung bietet. Anwendung finden diese Verfahren zur Erhöhung der Fehlertoleranz des Konfigurations- und Benutzerspeichers eines FPGAs. Als konkrete Anwendungen im Bereich der Xilinx FPGAs kann der CRC zur Überprüfung der Konfigurationsdaten (siehe Abschnitt 3.3.4) oder ein Hamming-Code zur Erkennung von Ein- und Zweibitfehlern im Konfigurationsspeicher (Details hierzu folgen im Abschnitt 5.7.3) genannt werden.

5.6.4 Speicher-Scrubbing

Ein Verfahren zum Schutz von Speicherblöcken auf Schaltungs- oder Systemebene ist das Überschreiben von Speicherinhalten [60, S. 98 f.]. In der Literatur hat sich der englische Begriff *Scrubbing* (dt. schrubben, reinigen) etabliert. Scrubbing wirkt einer Akkumulation von Einbitfehlern zur Systemlaufzeit entgegen und reduziert die Zeit einer fehlerhaften Bearbeitung durch eine betroffene Komponente.

Scrubbing-Verfahren können laut Rollins et al. in zwei Kategorien unterteilt werden: *Deterministisch* und *nicht-deterministisch* [147]. Ein nicht-deterministisches Scrubbing-Verfahren überprüft den Speicherinhalt nur dann, wenn explizit gelesen wird, während ein deterministisches Scrubbing-Verfahren regelmäßig den Speicherinhalt überprüft.

Das Verfahren kann auf unterschiedliche Arten von Speicher angewendet werden. Bei der Verwendung eines Speichers als ROM können vor dem Überschreiben des Speicherinhalts einfache Methoden zur Erkennung von Fehlern, wie z. B. ein Vergleich mit einem Ursprungswert, verwendet werden. Bei einem RAM verändert sich der Speicherinhalt hingegen dynamisch, sodass andere Verfahren, wie z. B. Redundanzverfahren benötigt werden um den korrekten Wert zu ermitteln. Anwendung findet dieses Verfahren daher in allen FPGAs, unabhängig von der verwendeten Technologie, zur Erhöhung der Fehlertoleranz des Benutzerspeichers (SRAM: Abschnitt 5.7.3, Antifuse und Flash: Abschnitt 5.7.4). Für SRAM-basierte FPGAs wird dieses Verfahren darüber hinaus verwendet, um die Fehlertoleranz des Konfigurationsspeichers zu erhöhen (Details hierzu folgen in Abschnitt 5.6.4).

5.7 Anwendung und Analyse der Abschwächungsverfahren

In diesem Abschnitt wird die Anwendung der zuvor vorgestellten Abschwächungsverfahren auf die drei FPGA-Technologien analysiert. Je nach FPGA-Typ ergibt sich ein unterschiedlicher Schwerpunkt zur Erhöhung der Fehlertoleranz, basierend auf den Ergebnissen des Abschnitts 5.5 (vgl. Tabelle 5.2). Der Fokus dieser Dissertation liegt auf SRAM-basierten FPGAs, sodass die Analyse dieser Speichertechnologie deutlich detaillierter ausfällt als die der Flash- und Antifuse-basierten FPGAs. Vor der Analyse der Anwendung auf die einzelnen FPGA-Technologien, werden zunächst typische Zuverlässigkeitsmetriken und Softwarekomponenten zur Strahlungsabschätzung vorgestellt.

5.7.1 Zuverlässigkeitsmetriken

Zur Analyse eines Gesamtsystems werden häufig Modelle und Simulationen zur Berechnung der Zuverlässigkeit verwendet. In der Literatur (z. B. [31, S. 37 ff.], [131] oder [82]) werden dafür unterschiedliche Metriken verwendet, welche im folgenden, basierend auf der erstgenannten Quelle, beschrieben werden.

Verteilungsfunktion Allgemein kann die Lebensdauer einer Komponente über eine Verteilungsfunktion (engl. distribution function) beschrieben werden. Eine Wahrscheinlichkeitsdichtefunktion (engl. probability density function) beschreibt dabei die Einzelwahrscheinlichkeit $f(x)$, die besagt, dass eine stochastische Variable X einen Wert kleiner oder gleich Variable x haben wird. Die Verteilungsfunktion $F(x)$ integriert über alle Einzelwahrscheinlichkeiten und wird daher auch als kumulierte Verteilungsfunktion bezeichnet:

$$F(x) = \int_{-\infty}^x f(x)dx = P(X \leq x) \quad (5.6)$$

Überlebensfunktion Basierend auf der Verteilungsfunktion kann die Überlebensfunktion (engl. survival function, auch reliability function) berechnet werden, Diese beschreibt die Wahrscheinlichkeit, dass die stochastische Variable X größer als die Variable x ist:

$$R(x) = 1 - F(x) = P(X > x) \quad (5.7)$$

Bezogen auf die Lebensdauer einer Komponente, beschreibt die Formel die Wahrscheinlichkeit, dass kein Ausfall bis zu einem bestimmten Zeitpunkt stattgefunden hat.

Ausfallrate Die Ausfallrate (engl. failure rate) beschreibt die Häufigkeit von Ausfällen pro Zeit und wird aus dem Verhältnis der Wahrscheinlichkeitsdichtefunktion $f(x)$ und der Überlebensfunktion $R(X)$ gebildet:

$$\lambda(x) = \frac{f(x)}{R(x)} \quad (5.8)$$

Mean Time To Failure (MTTF) Die *Mean Time To Failure* (MTTF) wird oft als Metrik zur Beschreibung von fehlertoleranten Systemen verwendet. Sie wird in Stunden, Tagen oder Jahren angegeben und kann über die Verteilungsfunktion berechnet werden:

$$MTTF = \int_{-\infty}^{\infty} x f(x)dx \quad (5.9)$$

Mean Time To Repair (MTTR) Die *Mean Time To Repair* (MTTR) beschreibt hingegen die Zeit, in der eine Komponente repariert, ersetzt oder zurückgesetzt wird.

Mean Time Between Failure (MTBF) Die *Mean Time Between Failure* (MTBF) beschreibt die Zeitspanne zwischen zwei Ausfällen und kann als invertierte Ausfallrate oder der Summe aus MTTF und MTTR beschrieben werden:

$$MTBF = \frac{1}{\lambda(x)} = MTTF + MTTR \quad (5.10)$$

Verfügbarkeit Das Verhältnis zwischen Ausfallszeit und der Betriebszeit eines Systems ist die Definition der Verfügbarkeit (engl. availability). Die Verfügbarkeit kann dabei durch die drei zuletzt genannten Metriken definiert werden:

$$\text{Verfügbarkeit} = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF} \quad (5.11)$$

In kritischen Systeme wird oft von *Hochverfügbarkeit* gesprochen und es existieren mehrere Verfügbarkeitsklassen [183]. Dabei kann ein System als hochverfügbar eingestuft werden, wenn seine jährliche Ausfallzeit im Bereich weniger Minuten liegt. Dies ist der Fall, wenn die Verfügbarkeit etwa einen Wert größer als 99,999 % besitzt:

99,999 % \equiv 26,3 Sekunden pro Monat oder 5 Minuten und 16 Sekunden pro Jahr

Failures In Time (FIT) Eine weitere Metrik beschreibt die Anzahl der Fehler pro eine Milliarde Stunden ($\approx 114\,155$ Jahre), die sogenannte *Failure In Time* (FIT), welche zum Beispiel von Xilinx verwendet wird:

$$FIT = \frac{N_{\text{Fehler}}}{10^9 h} \quad (5.12)$$

5.7.2 Strahlungsabschätzung

Um eine Abschätzung der Strahlung erhalten zu können existiert Software, die basierend auf dem Wirkungsquerschnitt eines Bauteils für unterschiedliche Szenarien in verschiedenen Umgebungen (z. B. Orbits) eine Strahlungsabschätzung und Fehlerraten berechnet.

CREME96 Die Software *Cosmic Ray Effects on MicroElectronics* (CREME) wurde 1981 von dem *Naval Research Laboratory* (Washington D.C., USA) entwickelt und hat sich als De-facto-Standard etabliert. Die letzte Revision der Software stammt aus dem Jahr 1996, sodass sich der Name CREME96 etabliert hat. CREME96 erstellt numerische Modelle von verschiedenen Weltraumumgebungen und evaluiert die zu erwartenden Fehlerraten durch Protonen- oder Schwerionenstrahlung [153, S. 49 f.]. Bei der Berechnung können sieben unterschiedliche Sonnenzustände (engl. solar conditions) für den Flux der Partikel berücksichtigt werden: Solar minimum, solar maximum, solar minimum trapped proton peak, solar maximum trapped proton peak, worst week, worst day und worst 5 minute peak. Eine detaillierte Dokumentation zur Parametrierung von und Berechnung der Fehlerraten mittels CREME ist in [58] zu finden.

Xilinx SEU FIT Rate Calculator Bei der Verwendung von Xilinx FPGAs kann zusätzlich der *SEU FIT Rate Calculator* verwendet werden. Die Software bietet feingranulare Einstellungsmöglichkeiten um eine möglichst genaue Vorhersage über die Anfälligkeit eines Designs geben zu können. Nachdem eine Familie und ein konkretes FPGA ausgewählt wurde, können designspezifische Parameter wie die Anzahl benutzter BRAM-Blöcke

oder die prozentuale Ausnutzung der Logikressourcen eingegeben werden. Bei den Umgebungseinstellungen kann die Höhe, der Längen- und Breitengrad sowie die Sonnenaktivität eingestellt werden.

5.7.3 SRAM-basierte FPGAs

Wie in Abschnitt 5.5 zusammengefasst wurde, existiert bei SRAM-basierten FPGAs eine primäre Strahlungsanfälligkeit gegenüber SEU-, MCU- und MBU-Effekten für den Konfigurations- und Benutzerspeicher. Die primäre Anfälligkeit gegenüber diesen Strahlungseffekten ist insbesondere der stetigen Miniaturisierung der Strukturgröße in der CMOS-Technologie begründet. In diesem Abschnitt wird daher die konkrete Anwendung der zuvor beschriebenen Abschwächungsverfahren zur Erhöhung der Fehlertoleranz gegenüber SEEs beschrieben. Die Miniaturisierung der Strukturgröße führt jedoch auch gleichzeitig zu einer Skalierung der Versorgungsspannungen. Durch die zusätzlich anwachsende Anzahl an Transistoren pro Bauteil steigt auch die Anfälligkeit gegenüber TID. Verfahren zur Behandlung dieses Effekts sind nur im geringen Maße vorhanden. Eine Übersicht dieser Verfahren erfolgt in dem Abschnitt Fehlererkennung und -korrektur.

Redundanzverfahren

Die Integration der, in Abschnitt 5.6.2 vorgestellten Redundanzverfahren, kann teilweise automatisiert durch Software erfolgen. So bieten bekannte Hersteller von Synthesewerkzeugen Software an, welche die TMR-Redundanzverfahren in ein Design automatisch integrieren: *Precision Hi-Rel* von Mentor Graphics, *Synplify Premier* von Synopsys oder *TMRTool* vom FPGA-Hersteller Xilinx. Aus dem wissenschaftlichen Bereich stammen zum Beispiel die frei-verfügbare Software BYU-LANL TMR (BL-TMR) der *Brigham Young University* (BYU) und der *Los Alamos National Laboratory* (LANL) oder die Software *Reliability-oriented place and Route Algorithm* (RoRA) der Politecnico di Torino.

Die in Abschnitt 5.6.2 vorgestellten, räumlichen Redundanz-Architekturen, DWC und TMR, teilen die gleichen Nachteile. Der offensichtlichste Nachteil ist der erhöhte Flächenbedarf, der je nach Verfahren einen Faktor von minimal 2 bzw. 3 ausmacht. In [90] wird DWC detailliert vorgestellt und mittels Fehlerinjektion und Strahlungstests getestet. Johnson et al. schreiben, dass ca. 99.85 % aller Fehler von dem DWC-Design erkannt werden können, bei einem gleichzeitigen Hardwareoverhead von 100 %. In [212] nennt Xilinx einen Faktor von 3,2 für die Software TMRTool. Mit dem erhöhten Flächenbedarf geht eine entsprechende erhöhte Verlustleistung einher. Zusätzlich kommt es aufgrund der zusätzlichen Komparatorlogik zu Takt-/Geschwindigkeitseinbußen im System. Bei dem TMRTool liegt dieser laut Xilinx bei ca. 10 %. In [12] wird dieser Wert anhand einer konkreten Schaltung auf ca. 14,6 % beziffert (Ausgangsschaltung: 156,59 MHz, TMRTool: 136,61 MHz).

Redundanzverfahren sind weiterhin anfällig für Ansammlungen von SEUs, die sich zur Verarbeitungszeit in einem System ergeben können [131]. Mehrere SEUs können

den Komparatormechanismus aushebeln und somit falsche Ausgaben produzieren. Detaillierte Analysen der FPGA Ressourcen sowie Fehlerinjektionstests haben ferner gezeigt, dass Designs/Schaltungen trotz Verwendung von TMR anfällig für einzelne SEUs im Konfigurationsspeicher sind [143]. Aufgrund dieser Tatsache entstand Software zur Anfälligkeitsanalyse eines Designs.

Anfälligkeitsanalyse

Mit der Anfälligkeitsanalyse kann ein Design auf die beschriebenen, kritischen Fehlerfälle hin untersucht werden. Die Analyse eines Designs kann z. B. mit der Software *Static AnalyzeR* (STAR) der Politecnico di Torino durchgeführt werden und bestimmt kritische/sensitive Konfigurationsbits [143]. Im Jahr 2004 wurde die erste Version der Software vorgestellt, welche Xilinx Spartan-II- und Virtex-FPGAs unterstützt. In einigen Fällen können einzelne SEUs zu mehreren Fehlern im Design führen. Insbesondere wenn Verdrahtungsressourcen betroffen sind, kann es zu drei unterschiedlichen Modifikationen an ursprünglich separaten Netzen kommen:

- *Short*: Kurzschluss zweier separater Netze durch eine neue Verbindung
- *Open*: Deaktivierung von Teilverbindungen eines Netzes
- *Open/Bridge*: Kombination von Short und Open in zwei Netzen

Ist z. B. eine Verbindung zweier Module eines TMR-Systems betroffen, führt dies zu einem falschen Ausgangswert. Die Komparatoren übernehmen dann den zweifach vorliegenden, fehlerhaften Wert. Darüber hinaus kann es durch MBUs zu einem ähnlichen Verhalten in der Schaltung kommen. Dieses Phänomen ist in der englischsprachigen Literatur als *Domain Crossing Event* (DCE) bekannt. Die Anfälligkeit für dieses Phänomen ist abhängig von mehreren Faktoren. So ist die FPGA-Architektur, die Organisation der Konfigurationsspeicherbits, die konkrete Schaltung auf FPGA und das betroffene Speicherbit von entscheidender Bedeutung. In [26] wird beschrieben, dass etwa 10 % der Fehler in den Verdrahtungsressourcen eines FPGAs (als Referenz wird ein Xilinx Spartan-II FPGA verwendet) mehrere Fehler verursachen können und somit nicht von TMR geschützt werden können.

In [144] und [166] wird eine Weiterentwicklung von STAR vorgestellt: *Reliability-oriented place and Route Algorithm* (RoRA). Die Software erweitert zunächst ein Design durch TMR und detektiert anschließend die DCEs in einem Design und eliminiert diese durch eine erneute Platzierung und Verdrahtung. Die ermittelten Daten der Anfälligkeitsanalyse werden somit verwendet um ein Design robuster zu gestalten. Initial wurden nur Xilinx FPGAs unterstützt, später folgte die Unterstützung für weitere SRAM- (Altera) und Flash-basierte (Microsemi) FPGAs. Die Wirksamkeit von RoRA wurde durch Fehlerinjektionstests bewiesen und ergab eine um Faktor 85 höhere Fehlertoleranz im Vergleich zu einem unbehandelten/normalen TMR-Design. Durch die erneute Platzierung und insbesondere durch die Verdrahtung mit Hinsicht auf DCEs, wird als Nachteil eine um etwa 22 % geringere Verarbeitungsgeschwindigkeit berichtet. Eine Weiterentwicklung der Software mit dem Namen *V-Place* (Versatile Placement), welche diesem Nachteil entgegenwirkt, wird in [164] vorgestellt. Sterpone

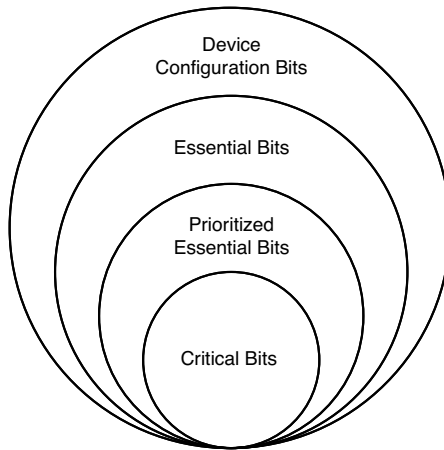


Abbildung 5.21: Klassifizierung der Xilinx-Konfigurationsspeicherbits [111, S. 2]

et al. schreiben von einer Erhöhung des Takts um bis zu 44 % bei einem Design auf einem Xilinx Virtex-II XC2V1000 FPGA. Mit einer weiteren Software STAR-MCU [167] wird ein Design auf Mehrbitfehler untersucht.

Im Jahr 2014 wurde die Software VERI-Place in Version 6.0 erstmals online zur Verfügung gestellt. Die Software erlaubt unter anderem die Erstellung von Anfälligkeitsanalysereports und Empfehlungen in Form von Randbedingungen für die FPGA-interne Platzierung von Designs mit Xilinx FPGAs zur Erhöhung der Zuverlässigkeit. Die aktuelle Version (8.0) unterstützt ebenfalls Xilinx FPGAs der 7-Serie.

Xilinx veröffentlichte erstmals ein Dokument und Software zur Anfälligkeitsanalyse Ende 2009 für die Virtex-5-Architektur. Das Dokument und die Software ist zur Zeit nur mit einem Geheimhaltungsvertrag (engl. Non-Disclosure Agreement, NDA) einsehbar. Im April 2012 stellte Xilinx mit den sogenannten *Essential Bits* [111] einen Algorithmus zur Anfälligkeitsanalyse für FPGAs der Familien Virtex-6, Spartan-6 und 7-Serie vor. Im Vordergrund steht hierbei die Klassifizierung von Konfigurationsbits in vier Klassen, wie in Abbildung 5.21 dargestellt. Die Oberklasse *Device Configuration Bits* enthält alle Konfigurationsspeicherbits. *Essential Bits* sind jene Speicherbits, die in einer spezifischen Schaltung verwendet werden. Die *Essential Bits* können durch einen benutzerspezifischen Prioritätsfilter auf bestimmte Teilregionen einer Schaltung beschränkt werden. Als Beispiel für diese *Prioritized Essential Bits* wird ein Zustandsautomat einer Kernkomponente eines Designs genannt. Die letzte Klasse *Critical Bits* umfasst die Speicherbits, welche bei einer Zustandsänderung zu einem funktionalen Defekt führen.

Während die Software der Politecnico di Torino die erstellte Anfälligkeitsanalyse verwendet um ein Design durch Änderungen an der Platzierungen oder der Verdrahtungen robuster zu gestalten, kann bei Xilinx die Anfälligkeitsanalyse proprietär von einem Xilinx SEU-Controller (ab Virtex-6) verwendet werden, um das Scrubbing auf sensible Bereiche des Designs anpassen zu können.

Fehlererkennung und -korrektur

Eine Fehlererkennung und -korrektur zur Erhöhung der Fehlertoleranz erfolgt in verschiedenen Bereichen eines SRAM-basierten FPGAs. In diesem Abschnitt werden Verfahren für den Konfigurationsspeicher und den Blockspeicher sowie Verfahren zur Erkennung permanenter Fehler vorgestellt.

Fehlererkennung und -korrektur im Konfigurationsspeicher

Eine Anwendung von Fehlerkorrekturverfahren ist in Komponenten zur Überwachung des Konfigurationsspeichers eines SRAM-basierten FPGA zu finden. Xilinx integriert ab der Virtex-4-Familie einen Hard-IP-Block (*Frame ECC*) zur Einzelfehlerkorrektur und Doppelfehlererkennung (engl. **S**ingle **E**rror **C**orrection **D**ouble **E**rror **D**etection, SECDED) im Konfigurationsspeicher [86]. Der Konfigurationsspeicher in einem Xilinx FPGA ist in sogenannte *Configuration Frames* aufgeteilt (siehe Abschnitt 3.3.1). In vielen FPGA-Familien sind die Daten eines *Configuration Frames* durch einen inkludierten Fehlerkorrekturcode geschützt [111]. In allen Xilinx FPGAs wird das Array aller *Configuration Frames*, welches der wesentliche Teil einer Konfigurationsdatei ist, weiterhin durch einen 32-bit CRC geschützt. Die Verwendung des CRCs erlaubt eine robuste Fehlererkennung, während der Fehlerkorrekturcode eine feingranulare Fehlerlokalisierung ermöglicht. So berechnet der *Frame ECC* IP-Block eines Virtex-4-FPGAs automatisch während eines Zurücklesens des Konfigurationsspeichers einen sogenannten *Syndromwert* (Hamming-Code mit 11 Bits) mit welchem eine konkrete Lokalisierung eines Einbitfehlers und eine Detektion eines Zweibitfehlers erfolgen kann. Die Korrekturfunktionalität für einen Virtex-4 muss jedoch vom Benutzer implementiert werden und wird in Abschnitt 6.2.2 beschrieben. In aktuelleren FPGA-Familien ist die Fehlerkorrektur durch einen Hard-IP-Core integriert. Details hierzu werden im folgenden Abschnitt beschrieben.

Fehlererkennung und -korrektur im Blockspeicher

Fehlerkorrekturverfahren werden weiterhin zur Erhöhung der Fehlertoleranz von BRAM-Blöcken eingesetzt. Eine ECC-Integration kann allgemein in Form von zusätzlicher Logik implementiert werden. Einige Hersteller bieten weiterhin Hardware-Lösungen an. So integriert z.B. Xilinx seit der Virtex-4-Familie einen 8 Bit Hamming-Code (SECDED) für 64 Bit Datenwörter [198]. Einbitfehler in den Daten werden während des Lesens automatisch korrigiert.

Dies ist zugleich der wesentliche Kritikpunkt an den eingebauten SECDED-Mechanismen. Eine Korrektur findet nur dann statt, wenn die Daten explizit gelesen wer-

den. Um einer Ansammlung von Einbitfehlern in BRAM-Blöcken entgegenwirken zu können, muss der Inhalt periodisch geprüft (gelesen) und korrigiert (geschrieben), also ein Scrubbing verwendet werden. Die Verwendung des ECC hat weiterhin einen deutlichen Einfluss auf die maximal erreichbare Taktfrequenz. Im Fall einer Xilinx Virtex-5-Architektur sinkt diese um bis zu 27,8 % (450 MHz vs. 325 MHz, [233, S. 48].

Fehlererkennung und -korrektur zur Erkennung permanenter Fehler

Testverfahren zur Erkennung permanenter Defekte auf FPGAs lassen sich allgemein in zwei Kategorien unterteilen: *anwendungsunabhängig* (engl. application-independent) und *anwendungsabhängig* (engl. application-dependent). Die anwendungsunabhängigen Verfahren werden oftmals zur Erkennung von Fehlern durch den Herstellungsprozess des FPGAs verwendet [142, 156, 168]. Die Verfahren werden weiterhin in der Regel offline und nur durch den FPGA-Hersteller durchgeführt und testen auf alle möglichen Fehler in der zugrundeliegenden FPGA-Struktur. Die Fehlererkennung erfolgt durch mehrere Testkonfigurationen mit entsprechenden Test-Pattern, wobei eine Testkonfiguration eine Untermenge der möglichen Fehlerfälle abdeckt. Testverfahren der zweitgenannten Kategorie werden für das Testen von konkreten, durch ein Design belegten, FPGA-Ressourcen verwendet und sind somit abhängig von der Anwendung [13, 25, 41, 148]. Im Gegensatz zu den anwendungsunabhängigen Verfahren werden diese Testverfahren somit durch den Benutzer des FPGAs durchgeführt. Die Ausführung kann dabei entweder offline oder online erfolgen. Das Grundprinzip dieser Verfahren gründet darauf, dass in den meisten FPGA-Designs nur ein Bruchteil der verfügbaren FPGA-Ressourcen verwendet werden (siehe Abschnitt 5.7.3). Das Testen dieser Ressourcen ist für einige Anwendungen ausreichend, da eine fehlerfreie Ausführung des Systems gegeben ist. Anwendung finden diese Verfahren neben der Erkennung von Defekten in der FPGA-Struktur in der Erkennung von SEUs [25].

Die Effekte von, durch Strahlung induzierten, permanenten Fehlern ähneln denen der Fehler, welche durch den Herstellungsprozesse erzeugt werden. Konkret können in beiden Fällen Fehler aufgrund von unterbrochenen Leitungen (*Open-Fehler*) sowie kurzgeschlossenen Leitungen (*Short-Fehler*) beobachtet werden. Die Fehlererkennung erfolgt, wie eingangs beschrieben, über mehrere Testkonfigurationen. Diese Testkonfigurationen bestehen aus zwei wesentlichen Komponenten. Der *Test Pattern Generator* (TPG) erzeugt Teststimuli für die zu testende Menge an FPGA-Ressourcen. Der, mit diesen FPGA-Ressourcen verbundene, *Output Response Analyzer* (ORA) überwacht den Ausgang dieser Ressourcen und bestimmt ob diese fehlerhaft oder fehlerfrei sind. Eine abstrakte Darstellung der Testschaltung ist in Abbildung 5.22 zu sehen.

Die Techniken dieser Analyse lassen sich in zwei Kategorien unterteilen: vergleichsbasiert (engl. comparison-based) und paritätsbasiert (engl. parity-based). In den vergleichsbasierten Verfahren, kennt der ORA sowohl die erwarteten, vom TPG erzeugten, als auch die über die zu testenden FPGA-Ressourcen versendeten Ausgangswerte. Durch einen Vergleich wird daher entschieden, ob die Daten übereinstimmen und somit eine fehlerfreie Datenübertragung erfolgt ist [149]. Der wesentliche Nachteil hierbei ist,

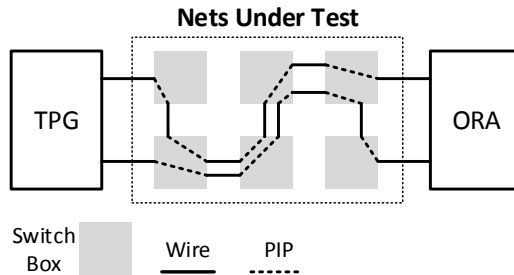


Abbildung 5.22: Abstrakte Darstellung der Testschaltung zur permanenten Fehlererkennung [161, S. 3]

dass Fehler in dem TPG nicht erkannt werden können. Aus diesem Grund wurden paritätsbasierte Verfahren entwickelt. Der TPG berechnet, basierend auf den Ausgangswerten ein Paritätsbit. Auf der Gegenseite berechnet ebenfalls der ORA ein Paritätsbit der empfangenen Eingangswerte. Durch den Vergleich dieser zusätzlichen Paritätsbits ist es nun möglich, Fehler in dem TPG zu erkennen.

Die paritätsbasierten Verfahren zur Analyse lassen sich wiederum in zwei Kategorien unterteilen: *single parity* oder *cross-coupled parity*. In der erstgenannten Technik besteht der TPG aus einem n-bit großen Zähler und produziert zusammen mit dem Paritätsbit n+1 Ausgangswerte [169]. Der ORA empfängt diese Ausgangswerte, berechnet wie bereits beschrieben ein eigenes Paritätsbit und vergleicht dieses mit dem empfangenen Paritätsbit. Diese Umsetzung hat den Nachteil, dass einige Fehler nicht erkannt werden können. Zudem muss sichergestellt sein, dass das Paritätsbit fehlerfrei übertragen wurde. Das zweitgenannte Verfahren löst diese Problematik durch die Verwendung zweier unabhängiger Zähler in beiden Testkomponenten (TPG und ORA) [254]. Die Verbindung erfolgt anschließend, wie der Name des Verfahrens andeutet, teils Überkreuz: Der erste ORA wird mit den n-bits des ersten TPG und dem Paritätsbit des zweiten TPG, der zweite ORA mit den n-bits des zweiten TPG und dem Paritätsbit des ersten TPGs verbunden. Somit ist es möglich eindeutig zwischen Übertragungsfehlern und Fehlern in der Erzeugung durch den TPG zu unterscheiden.

Die Fehlererkennung von, durch langanhaltender Strahlung induzierter, permanenter Fehler auf FPGAs wird einzig in [16] untersucht. Bauer et al. untersuchen die Auswirkung auf die konfigurierbaren Logikblöcke (CLBs). Die Fehlererkennung erfolgt weiterhin einzig für die verwendeten FPGA-Ressourcen eines Designs und ist somit anwendungsabhängig.

Scrubbing des Konfigurationsspeichers

Das Scrubbing-Verfahren kann insbesondere zur Erhöhung der Fehlertoleranz in SRAM-basierten FPGAs eingesetzt werden, da Fehler im Konfigurationsspeicher, welche einen

Einfluss auf die Verdrahtungs- und Logikressourcen haben, ohne Beeinträchtigung des laufenden Systems behoben werden können. Da sich der Inhalt des Benutzerspeichers in der Regel ändert, kann das Verfahren nicht verwendet werden, um den Speicherinhalt zu schützen. Das Scrubbing-Verfahren des Konfigurationsspeichers kann in zwei Kategorien unterteilt werden: *Blind Scrubbing* und *Readback Scrubbing*.

Blind Scrubbing

Bei dieser Variante wird der Speicher periodisch mit einer fehlerfreien Kopie des Speicherinhalts überschrieben. Da keine Fehlerauswertung erfolgt, wird dieses Verfahren als präventiv oder auch *blind* bezeichnet. Das Verfahren realisiert ein deterministisches Scrubbing nach der Kategorisierung aus Abschnitt 5.6.4. Xilinx verwendet weiterhin den Begriff *Scheduled Maintenance* [83]. Die Kopie des Speicherinhalts wird in einem externen, strahlungsgehärteten Speicher abgelegt und in der englischsprachigen Literatur *golden copy* genannt. Kritikpunkte bei diesem Verfahren sind, dass der Kontroller selbst durch den Strahlungseinfluss beschädigt sein kann, wodurch der Speicherinhalt fehlerhaft würde und dass unnötige Schreibvorgänge erfolgen. Der Rekonfigurationskontroller wird aus dem erstgenannten Grund in der Regel in eine strahlungsgehärtete Komponente ausgelagert.

Readback Scrubbing

Den Kritikpunkten des Blind Scrubbings wird mit dem sogenannten Readback Scrubbing Verfahren entgegengewirkt. Durch ein vorheriges Zurücklesen des aktuellen Speicherinhalts über den Kontroller kann festgestellt werden ob dieser fehlerhaft ist. Es erfolgt also eine Art *Monitoring* des Konfigurationsspeichers. Nur in einem Fehlerfall erfolgt anschließend eine Reparatur des Speicherinhalts, wodurch unnötige Schreibvorgänge vermieden werden. Das Verfahren entspricht daher, nach der Kategorisierung aus Abschnitt 5.6.4, einem nicht-deterministischem Scrubbing. Xilinx führt für diese Art des Scrubbings den Begriff *Running Repair* ein und das *Xilinx Radiation Test Consortium* (XRTC) empfiehlt die Verwendung dieses Verfahrens gegenüber einem Blind Scrubbing [173]. Neben einem Vergleich mit der fehlerfreien Konfigurationsdatei (*golden copy*), kann durch Verwendung eines Fehlerkorrekturcodes (ECCs) auf den Konfigurationsspeicherinhalt eine zweite Methode bei Xilinx FPGAs angewendet werden. Wie bereits beschrieben, integriert Xilinx seit der Virtex-4-Familie einen Hard-IP-Block (*Frame ECC*) zur Einzelfehlerkorrektur und Doppelfehlererkennung (SECDED). Durch die Lokalisierung eines Einbitfehler (SEU) kann ein Einbitfehler gezielt korrigiert werden. Details zu dieser Komponente sowie einer Implementierung des Readback Scrubbings sind in Abschnitt 6.2.2 beschrieben. Eine Fehlerkorrektur von Mehrbitfehlern kann für FPGAs allgemein über zusätzlich implementierte Fehlerkorrekturverfahren erfolgen [109].

Parameter des Scrubbings

Unabhängig von dem gewählten Scrubbingverfahren, sind verschiedene Parameter für beide Verfahren auszuwählen, welche z. B. in [82] analysiert und gegenübergestellt werden:

- Feste oder variable Scrubbing-Rate
- Interner oder externer Rekonfigurationskontroller
- Vollständige oder partielle Überprüfung
- 1D oder 2D Scrubbing
- Soft- oder Hardware-Implementierung

Scrubbing-Rate Die Scrubbing-Rate kann auf einen festen Wert oder einen variablen Wert gesetzt werden. In den ersten Beiträgen (aus dem Jahr 2008 und in Zusammenarbeit mit Xilinx) wird eine fixe Rate mit einem, um Faktor 10 höheren Wert als die maximal berechnete SEU-Rate empfohlen [2]. Diese hohe Rate führt zu einer konstanten, erhöhten Verlustleistung des Gesamtsystems. Für eine variable Rate sprechen hingegen, dass ein FPGA in vielen Anwendungen nicht zu jedem Zeitpunkt aktiv ist und dass die Strahlung in vielen Anwendungsgebieten schwankt. Eine Anpassung an den aktuellen Betriebsmodus oder die aktuelle Betriebsposition ist somit vorteilhaft.

Extern vs. Intern Die Konfiguration kann über einen externen oder internen Rekonfigurationskontroller erfolgen. In [23] wird ein Vergleich zwischen einem externen und einem internen Rekonfigurationskontroller vorgestellt. Externe Rekonfigurationskontroller sind typischerweise in strahlungsgehärteten Prozessoren, FPGAs oder PLDs (engl. **Programmable Logic Devices**) realisiert. Es wird zusammenfassend festgehalten, dass aufgrund der Anfälligkeit für SEEs eines FPGAs interne Rekonfigurationskontroller weniger sicher sind, da die Logik zur Ansteuerung selbst durch Upsets beeinflusst werden kann. Strahlungstests haben gezeigt, dass insbesondere MBUs dazu führen, dass ein interner Rekonfigurationskontroller nicht mehr fehlerfrei arbeitet. Durch die Verwendung von TMR innerhalb des internen Rekonfigurationskontrollers kann die Robustheit jedoch deutlich gesteigert werden.

Vollständig vs. Partiiell Die Konfiguration kann weiterhin vollständig, für ein gesamtes FPGA oder partiell, das heißt ausschnittsweise für ein FPGA durch Scrubbing überschrieben werden. Xilinx verwendet für die beiden Konfigurationsarten die Termini *device-based* und *frame-based* [35, S. 15 ff.]. Die Vorteile eines vollständigen Scrubbings sind die simple Implementierung ohne großen Verwaltungsaufwand und die etwas höhere Scrubbing-Geschwindigkeit. Der letztgenannte Vorteil ergibt sich, da nur einmalig der Header der Konfigurationsdatei verarbeitet, die Kommandos an den Paketprozessor gesendet sowie ein SEFI-Test der Konfigurationsschnittstelle erfolgen muss. Ein deutlicher Nachteil ist die Auswirkung auf das System, in dem Fall, dass der Konfigurationsprozessor selbst durch einen SEFI fehlerhaft arbeitet. In dem Fall hat eine vermeintliche Korrektur weitreichende Konsequenzen. In [36, S. 19 ff.] beschreibt Xilinx dieses Fehlverhalten als *Scrub/High-Current SEFI*. Upsets in der internen Logik

des Konfigurationsprozessors führen zu einem stetigen Anstieg der Stromaufnahme und können zur Beschädigung des FPGAs führen. Xilinx empfiehlt daher zur Minimierung des Einflusses dieses SEFIs ein Scrubbing nur dann auszuführen, wenn ein Fehler detektiert wurde. Dies ist einer der Vorteile eines partiellen Scrubbings, da hier nur Teile der Konfiguration betroffen sind. Nachteilig an einem partiellen Scrubbing ist der erhöhte Implementierungsaufwand zur Realisierung des Rekonfigurationskontrollers und eine insgesamt geringere Scrubbing-Rate durch den jeweiligen, zusätzlichen Konfigurationsaufwand (Header, Konfigurationskommandos und SEFI-Test).

1D vs. 2D Scrubbing Die Überprüfung des Konfigurationsspeichers erfolgt typischerweise zeilenweise pro *Configuration Frame* und somit auf einer vertikalen, eindimensionalen Achse von der Startspalte bis zur letzten FPGA-Spalte (siehe Abschnitt 3.3.1). In der Regel werden jedoch nicht alle FPGA-Blöcke von einem Design verwendet. Eine Überprüfung eines solchen Bereichs kann somit unnötig sein. Zu Bedenken hierbei ist jedoch, dass die Verdrahtung durch einen ungenutzten (Logik-) Bereich führen kann. Zusätzlich kann ein Bereich wichtige/kritische Systemkomponenten enthalten. Durch die Implementierung eines zweidimensionalen Scrubbings, kann ein Bereich gezielt und häufiger überprüft werden. Hierbei erfolgt die Überprüfung des FPGAs also nicht bis zum Zeilenende, der letzten Spalte des FPGAs, sondern nur bis zur letzten Spalte der Systemkomponente. Durch diese feingranulare Einteilung ist es ferner möglich *Configuration Frame* basierte Scrubbing Profile/Schedules anzulegen.

Soft- vs. Hardware-Implementierung Ein Scrubber kann weiterhin durch eine Software auf einem Prozessor oder in Hardware realisiert sein. Der Vorteil einer Software-Lösung ist die Flexibilität. So kann beispielsweise schnell zwischen verschiedenen Algorithmen gewechselt werden. Im Vergleich mit einer Hardware-Lösung ist diese jedoch deutlich langsamer in der Fehlererkennung und -korrektur. Nachteilig an der Hardware-Lösung ist die vergleichsweise geringe Flexibilität, z. B. um komplexe Scrubbing-Strategien implementieren zu können. Eine hybride Lösung mit einer Aufteilung in Hard- und Software ist ebenfalls möglich.

Leistungsaufnahme Das Scrubbing des Konfigurationsspeichers führt, zum einen durch die zusätzliche Logik und zum anderen durch die Verwendung der Konfigurationsschnittstelle und des Paketprozessors, unweigerlich zu einer erhöhten Leistungsaufnahme des Gesamtsystems, unabhängig von der Art der Implementierung [82]. Der Anstieg der Leistungsaufnahme kann jedoch durch die Berücksichtigung einzelner Faktoren minimiert werden. Der erste Faktor ist die Readback- oder Scrubbing-Rate. Die Verwendung einer festen Rate, die üblicherweise an die höchste Strahlungsrate der Mission angepasst ist, führt zu einer hohen Leistungsaufnahme, welche für einen großen Zeitraum unnötig ist. Durch die Verwendung einer variablen, an die Strahlungssituation angepassten, Scrubbing-Rate kann die Leistungsaufnahme auf ein Minimum reduziert werden. Durch die Variation der Konfigurationstaktfrequenz kann die Verlustleistung weiter minimiert werden. Die Verwendung eines Konfigurationsspeicher-Scrubbing ist

in Bereichen mit hoher Strahlungsrate auch aus Sicht der Verlustleistung sinnvoll. Der Einfluss von Strahlung kann zu einer Aktivierung von eigentlich unbenutzten Blöcken in einem FPGA führen. Dies führt zu einer erhöhten Leistungsaufnahme, welche die Leistungsaufnahme des Scrubbing übersteigen kann. Ein konkretes Beispiel hierzu ist in Abschnitt 5.9.1 beschrieben, wo die Stromaufnahme des Systems durch das Scrubbing konstant um 10 % ansteigt. In einem Strahlungstest stieg die Stromaufnahme bei deaktiviertem Scrubbing hingegen kontinuierlich an, sodass diese nach acht Stunden bereits doppelt so hoch wie eingangs war.

Scrubber-Implementierungen

Xilinx bietet ab der Virtex-4-Familie eine hybride ScrubberImplementierung, *Soft Error Mitigation (SEM) Controller*), basierend auf einem PicoBlaze und FPGA-Logik, an [213]. Durch die Verwendung des PicoBlaze ergeben sich jedoch einige Nachteile (z. B. bei der Fehlererkennungszeit), sodass im akademischen Bereich Alternativen entwickelt wurden. Eine Übersicht und ein Vergleich für die Virtex-4-Familie folgt in Abschnitt 6.4.

FPGAs ab der Virtex-5-Familie integrieren einen Hard-IP-Core zur kontinuierlichen Überprüfung durch ein Zurücklesen des Konfigurationsspeichers und anschließendem Vergleich des daraus berechneten CRC-Werts und eines intern gespeicherten CRC-Werts [83]. Der gespeicherte CRC-Wert wird hierbei während des ersten Durchlaufs berechnet und anschließend als *golden value* verwendet. Im Falle einer Nichtübereinstimmung signalisiert die Komponente einen CRC-Fehler und die, zur Virtex-4-Familie identische, FRAME_ECC-Komponente kann zur genauen Lokalisierung eines Einbitfehler verwendet werden.

Für die Virtex-6-Familie wurde der CRC-IP-Core hardwareseitig um die SECEDED-Funktionalität erweitert, sodass die bis dato zusätzlich benötigte Logik zur Fehlerkorrektur der Einbitfehler nicht mehr vom Anwender bereitgestellt werden muss. Im Falle einer Fehlererkennung sind verschiedene Modi verfügbar, wie *correct and continue* (nach der Korrektur wird mit dem nächsten Configuration Frame fortgefahren) oder *correct and halt* (nach der Korrektur wird das Zurücklesen gestoppt). Weitere Details zu dem CRC-IP-Core sind in [239, S. 155 ff.] beschrieben.

Die bereitgestellte Funktionalität kann für FPGAs der 7-Serie durch eine neue Version des Logik-IP-Cores SEM erweitert werden, sodass auch eine Fehlerinjektion und Fehlerklassifikation (basierend auf der bereits beschriebenen Anfälligkeitsanalyse *Essential Bits*) durchgeführt werden kann [214]. Durch diese Klassifikation kann die Korrektur auf nur für das Design relevante Fehler eingegrenzt werden.

Bei der Verwendung des Scrubbing-Verfahrens müssen einige Einschränkungen berücksichtigt werden. Während Fehler in der Verdrahtung und in Logikressourcen (CLBs, DSP etc.) behoben werden können, existieren Komponenten, die nicht überprüft werden können. Bei Xilinx FPGAs sind dies z. B. der Konfigurationsprozessor selbst, digitale Taktverteiler (DCM), aber auch Schieberegister oder LUTRAM sowie BRAM. Weiter-

hin muss berücksichtigt werden, dass selbst bei hohen Scrubbing-Raten eine endliche Zeitspanne existiert, in der sich ein Fehler in einem System ausbreiten kann [190]. Aus diesen Gründen wird je nach benötigter Zuverlässigkeit die Kombination mehrerer Abschwächungsverfahren zur Erhöhung der Fehlertoleranz eines Systems empfohlen.

Kombination der Verfahren

In [29] stellt Xilinx eine Entscheidungsmatrix zur Erhöhung der Fehlertoleranz eines Xilinx-basierten Systems vor (siehe Abbildung 5.23). Als Parameter sind die SEU-Rate und das Zeitfenster der Benutzung sowie die Sensitivität/Wichtigkeit der Daten und die Langlebigkeit eines Fehlers aufgeführt.

Data Criticality		Low → High				
Error Persistence		No	Yes			
SEU Rate	Operating Window	Low	Minutes	No Mitigation	XTMR	
		High	Days	Scrubbing	Scrubbing XTMR	Redundant Devices
		Months				
		Continuous				

Abbildung 5.23: Entscheidungsmatrix zur Wahl der Abschwächungsverfahren [29, S. 3]

Durch die Matrix kann eine grobe Abschätzung für die Verwendung von (Kombinationen von) Abschwächungsverfahren für den Einsatz eines Systems in einer Mission erfolgen. Die Empfehlung variiert dabei von dem Einsatz keines Verfahrens für unkritische Daten bei gleichzeitig geringer SEU-Rate und kurzem Zeitfenster über Scrubbing, XTMR, eine Kombination von XTMR und Scrubbing bis hinzu der Verwendung von redundanten Bauteilen in dem Fall, dass die Daten kritisch, die SEU-Rate hoch und das Zeitfenster der Benutzung groß ist. Xilinx weist darauf hin, dass in der Matrix zwei wesentliche Parameter nicht betrachtet werden: Verlustleistung und Verarbeitungsgeschwindigkeit. Die Entscheidungsmatrix von Xilinx gibt eine erste Übersicht über die Kombination von verschiedenen Verfahren zur Erhöhung der Fehlertoleranz.

5.7.4 Flash- und Antifuse-basierte FPGAs

Flash-basierte FPGAs sind, wie in Abschnitt 5.6 dargestellt, primär anfällig für zwei Strahlungseffekte: SET und TID. Der Benutzerspeicher (Register nur im Fall von Antifuse-basierten FPGAs) ist von beiden FPGA-Technologien durch die Verwendung

von SRAM-Technologie ebenfalls anfällig gegenüber SEUs. In [176] werden Verfahren zur Behandlung von Strahlungseffekten für Flash- und Antifuse-basierte FPGAs vorgestellt. Als Referenzen werden Microsemis Flash-basierter RT ProASIC3 FPGA und Microsemis Antifuse-basierter RTAX FPGA verwendet. Die RT ProASIC3-Familie basiert dabei auf den gleichen Masken und Silizium wie die kommerzielle ProASIC3-Familie. Weitere Details zu beiden Familien sind in Abschnitt 5.8.3 beschrieben.

SEE-Effekte

Der SRAM-basierte Benutzerspeicher in beiden FPGA-Technologien ist anfällig für SETs oder SEUs. Durch die Verwendung von Flash-Zellen als Register in einem Flash-basierten FPGA sind diese ebenfalls anfällig gegenüber SETs und entsprechende Verfahren zur Abschwächung müssen eingesetzt werden.

Register

Auf Netzlistenebene werden in [176] unterschiedlichen TMR-Varianten (siehe Abschnitt 5.6.2) aufgeführt und die Auswirkungen auf die benötigte Fläche und Taktfrequenz diskutiert. Besonders erwähnenswert ist hierbei, dass im Falle der Verwendung des vollständigen TMR-Verfahrens in dem ProASIC3-FPGA, die theoretische Auswirkung von 200 % auf die Fläche deutlich übertroffen wird. Bedingt durch die ProASIC3-Architektur kommt es bei der Fläche zu einem Faktor zwischen 4,5 und 7,5. Dies führt weiterhin zu einer Herabsetzung des Takts um etwa 50 %. Aufgrund dieses starken Einflusses wird die Verwendung von lokalem TMR, also der Verdreifachung der Register und einem zusätzlichem Komparator, empfohlen. Die Verwendung von lokalem TMR hat bereits gezeigt, dass die SEU-Raten im Vergleich mit durch TMR geschützten Antifuse-Register, um eine Größenordnung besser sind. Durch die Verwendung von lokalem TMR ist die SET-Empfindlichkeit verbessert, die wesentlichen Schwachstellen (siehe Abschnitt 5.6.2) sind jedoch nicht behoben. Durch die Verwendung von zusätzlichem *Guard Gate Filtering* kann ein SET-Filter aufgebaut werden. Abbildung 5.24 stellt exemplarisch das auf zeitlicher Redundanz beruhende Verfahren dar.

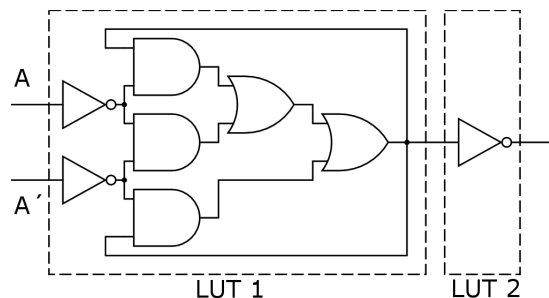


Abbildung 5.24: Guard Gate zur SET-Filterung [176, S. 5]

Ein Guard Gate besteht aus zwei Eingangsinvertern, welche den Ausgang nur dann treiben, wenn beide Eingänge den gleichen Wert besitzen. Es wird weiter festgehalten, das eine SET-Immunität von 4 ns durch Inverterketten mit acht Gliedern erreicht werden kann. Dies führt in der ProASIC3-Architektur wiederum zu einem maximalen Takt von 60 MHz. Eine Software zur SET-Analyse und automatischen Integration solcher Inverterketten wird z. B. in [165] vorgestellt.

Blockspeicher

Der Blockspeicher ist, wie eingangs erwähnt, der empfindlichste Teil beider FPGA-Technologien. Die Empfehlung von Urbina-Ortega et al. ist daher, analog zu SRAM-basierten FPGAs, die Kombination von Scrubbing und EDAC-Verfahren [176]. Als EDAC wird mindestens die Verwendung von SECDED angeraten. Die Scrubbing-Rate sollte weiterhin entsprechend der Worst-Day-Abschätzung gewählt werden.

TID-Effekte

Die einzige Möglichkeit die TID-Effekte in Flash-basierten FPGAs abzuschwächen, ist von vornherein die Herabsetzung der Signallaufzeitverzögerung (engl. propagation delay) durch Timing Constraints zu berücksichtigen [176]. Mit Hilfe von Formeln kann eine zusätzliche, definierte Toleranz bzgl. der TID-Anfälligkeit berechnet werden. In dem Fall des RT ProASIC3-FPGAs wird die Verwendung folgender Formel empfohlen, wobei $\Delta P_d(TID)$ die prozentuale Abweichung der Verzögerung der Signallaufzeit beschreibt:

$$\Delta P_d(TID) \approx \begin{cases} \frac{TID}{2}, & \text{wenn } TID < 20 \\ 10 + 8 \cdot \frac{(TID - 20)}{15}, & \text{wenn: } 20 \leq TID \leq 35 \end{cases} \quad (5.13)$$

Urbina-Ortega et al. resümieren, dass die geringe TID ein entscheidender Schwachpunkt ist, welcher dazu führt, dass die Mehrheit Flash-basierter FPGAs in der Regel nicht in Langzeitmissionen (Durchschnittsdauer von 15 Jahren mit wenigstens 70 krad) im GEO eingesetzt werden können [176]. Einzig die RTG4-Familie von Microsemi (siehe Abschnitt 5.8.3) bietet Flash-basierte FPGAs mit einer TID von bis zu 100 krad an.

5.8 Strahlungstolerante und -gehärtete FPGAs

Neben den Standard-FPGAs, oftmals auch als *Commercial Off-The-Shelf* (COTS-) FPGAs bezeichnet, existieren FPGA-Familien, die durch RHBD-Schritte zur Designzeit des FPGAs und / oder RHBP-Schritte in der Fertigung zuverlässiger bzw. immun gegenüber bestimmten Strahlungseffekten sind. Der Begriff COTS wird in der Literatur nicht eindeutig verwendet, da auch die strahlungstoleranten (engl. radiation-tolerant, rad-tolerant) oder -gehärteten (engl. radiation-hardened, oftmals mit rad-hard abgekürzt) FPGAs kommerziell verfügbar sind. Da strahlungstolerante oder -gehärtete FPGAs nicht

in großer Stückzahl produziert werden, schließt der Begriff diese Klasse von FPGAs nicht ein. In dieser Dissertation umfasst der Begriff COTS daher ausschließlich Standard-FPGAs. Analoge Beispiele aus der Literatur mit dieser Verwendung des Begriffs sind z. B. in [31, 70, 88, 107, 171] zu finden. Die Definitionen von strahlungstolerant und strahlungsgehärtet variieren ebenfalls in der Literatur. In [15, S. 202] werden diese beiden Kategorien wie folgt definiert:

- Strahlungstolerant: FPGAs, die gegen bestimmte Strahlungseffekte durch Verwendung von RHBD-Methoden immun sind und gegen andere Strahlungseffekte durch Verfahren auf Anwendungsebene geschützt werden können.
- Strahlungsgehärtet: FPGAs, die eine Immunität gegenüber Strahlungseffekte durch RHBD- und RHBP-Verfahren bieten.

Der Vorteil der Verwendung dieser FPGAs liegt an dem, durch den Hersteller, verifizierten Strahlungsschutz, wodurch der Anwender in der Regel keine weiteren Verfahren zur Behandlung von Strahlungseffekten im Design implementieren muss und sich direkt mit der zu entwickelnden Anwendung befassen kann. Die Kosten eines solchen FPGAs liegen laut Battezzati et al. um mindestens eine Größenordnung höher, als vergleichbare kommerzielle Versionen [15]. Dieser Punkt, die vergleichsweise hohen Kosten, wird oftmals als Hauptargument der Verwendung von COTS-FPGAs aufgeführt. Laut Swift ist dies ein falscher Grund, da die Kosten für anschließende Tests berücksichtigt werden müssen [171]. Akzeptable Gründe seien hingegen die Verwendung von aktueller anstelle mehrerer Jahre bzw. jahrzehntealter Technik. Am Beispiel von Xilinx kann dies verdeutlicht werden: Die kommerzielle Virtex-4-Familie wurde bereits im Jahr 2004, die Virtex-5-Familie im Jahr 2006 produziert. Die strahlungstolerante Virtex-4QV-Familie und die strahlungsgehärtete Virtex-5QV-Familie sind die aktuellsten, verfügbaren SRAM-basierten FPGAs für den Einsatz im Weltraum. Die Vorteile durch neue Prozessschritte, wie sinkende Verlustleistung oder eine höhere Performance können somit ausschlaggebende Gründe zur Verwendung von COTS-FPGAs sein. Weiterhin wird die Anzahl von FPGA-Ressourcen mit jeder FPGA-Generation höher. Ein limitierender Faktor bei strahlungsgehärteten bzw. -toleranten FPGAs kann darüber hinaus die Anzahl an verfügbaren IO-Blöcken sein.

5.8.1 Xilinx

Xilinx bietet sogenannte *space-grade* FPGAs für die Verwendung im Weltraum an. Während FPGAs der Xilinx Virtex-4QV-Familie und vorherige Familien auf denselben Fertigungsmasken und somit gleichen Transistor-Verschaltungen wie die kommerziell verfügbaren FPGAs basieren, ist das Xilinx Virtex-5QV-FPGA das erste FPGA mit implementierten RHBD- und RHBP-Methoden.

Virtex-4QV In der Virtex-4QV-Familie existieren die gleichen Unterfamilien wie in der kommerziellen Virtex-4-Familie. Im Unterschied zu dieser werden jedoch nur vier FPGAs insgesamt hergestellt [216]: XQR4VLX200, XQR4VSX55 und XQR4VFX60/140. Der einzige Unterschied in der Fertigung gegenüber den kommerziellen Gegenstücken ist die

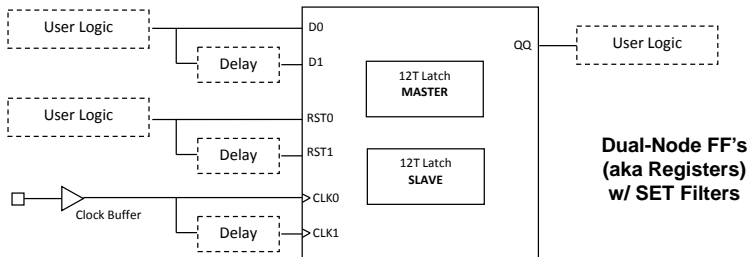


Abbildung 5.25: Dual-Node Flip-Flops mit SET-Filter (Xilinx Virtex-5QV) [170, S. 10]

Verwendung einer dünnen Epitaxieschicht um eine Latchup-Immunität gewährleisten zu können. Strahlungstests haben dabei eine SEL-Immunität bis zu $100 \frac{\text{MeV cm}^2}{\text{mg}}$ ergeben. Somit gehören diese FPGAs zu der Klasse der strahlungstoleranten FPGAs. Weiterhin wird ein Keramikgehäuse verwendet und jeder Wafer Tests nach den *Defense Supply Center Columbus* (DSCC) Standards für Zuverlässigkeit und Strahlungscharakterisierung unterzogen. Xilinx garantiert für die FPGAs eine TID von bis zu 300 krad. Da keinerlei RHBD-Methoden für die einzelnen Speicher, des Konfigurationsprozessors oder der Konfigurationsschnittstelle realisiert wurden, müssen die zuvor aufgeführten Verfahren zur Behandlung von Strahlungseffekten angewendet werden, um ein Design robuster zu gestalten.

Virtex-5QV Während bei den vorherigen *space-grade* Versionen verschiedene FPGAs auswählbar waren, existiert in der Xilinx Virtex-5QV-Familie lediglich ein FPGA, das XQR5VFX130 [236]. Im Vergleich zu dem Pin-kompatiblen, kommerziellen Gegenstück XC5VFX130T wurden der *System Monitor*, die zwei PowerPC-Blöcke und vier IO-Blöcke an den Ecken sowie zwei GTX-Transceiver entfernt um die Zuverlässigkeit des Gehäuses zu erhöhen [172]. Das Virtex-5QV-FPGA ist durch RHBD- und RHBP-Verfahren das erste strahlungstolerante FPGA von Xilinx. Zur Erhöhung der SEU-Robustheit in dem Konfigurationsspeicher, wurde die SRAM-Zelle verdoppelt (engl. dual-node configuration cell), sodass 12 Transistoren anstelle von 6 Transistoren ein Konfigurationsbit abbilden. Zusätzlich wurde ein definierter Abstand zwischen diesen beiden SRAM-Zellen festgelegt, um den Einfluss eines SEE und die Auswirkung auf die kritische Ladung beider Speicherzellen (Q_{crit1} bzw. Q_{crit2}) zu minimieren. Durch diese Maßnahme ist gleichzeitig die Robustheit des LUTRAM erhöht worden. Im Vergleich mit der kommerziellen Virtex-5-Architektur konnte der Wirkungsquerschnitt um ca. drei Größenordnungen verringert werden. Laut Huey hat diese Veränderung einen Flächenanstieg von 25 % im Silizium (bei einer Strukturgröße von 65 nm) zur Folge [84]. Ähnlich zu den Konfigurationsszellen, wurden die Anzahl der Transistoren der Flip-Flops verdoppelt und SET-Filter an den Daten-, Takt- und Kontrollleitungen der Flip-Flops eingefügt (siehe Abbildung 5.25). Um den Zustand des Flip-Flops zu

ändern müssen beide Dateneingänge übereinstimmen. In [170] wird berichtet, dass durch dieses Verfahren eine um bis zu zwei Größenordnungen bessere Upset-Rate erzielt werden kann. Weiterhin haben SETs mit einem Glitch von bis zu 800 ps [236] keinen Einfluss auf die Flip-Flops. Der Konfigurationsprozessor des FPGAs und der JTAG-Kontroller sowie deren Kontrollregister wurden weiterhin gegenüber SEUs und SETs durch TMR und EDAC geschützt. Durch diese Maßnahmen konnte die SEFI-Rate verbessert werden. Xilinx garantiert in [236] eine TID von bis zu 1 000 krad und eine SEL-Immunität bis zu $100 \frac{\text{MeV cm}^2}{\text{mg}}$. In [172] wurde in Strahlungstests eine Immunität bis zu $130 \frac{\text{MeV cm}^2}{\text{mg}}$ und eine vollständige Immunität der Konfigurationsspeicherzellen durch Protonenstrahlung festgehalten.

Vergleich In [153, S. 68] wird auf die Entwicklung der Xilinx *space-grade* FPGAs für die abgeschätzten Upset-Raten in einem geostationären Orbit (GEO) eingegangen. Die Tabelle 5.4 stellt exemplarisch drei FPGAs der Virtex-II bis zur Virtex-5QV Familie gegenüber. Der Einfluss der RHBD- und RHBP-Methoden der Virtex-5QV-Familie auf die Upset-Raten wird hierbei besonders deutlich.

Tabelle 5.4: Abgeschätzte Upset-Raten für drei FPGAs der aktuellsten Xilinx *space-grade* FPGA-Familien [153, S. 68]

FPGA	CRAM in $\frac{\text{Upset}}{\text{Bauteil-Tag}}$	BRAM in $\frac{\text{Upset}}{\text{Bauteil-Tag}}$	SEFI in $\frac{\text{Bauteil-Jahre}}{\text{Event}}$
XQR2V6000	13,3	2,03	181
XQR4VFX60	4,4	3,26	103
XQR5VFX130T	0,014	0,002 3(ECC)	9 930

Zynq RT UltraScale+ In [85] wird die nächste Generation der Xilinx *space-grade* Familie vorgestellt, welche auf der Zynq UltraScale+ Familie basiert und analog zu der Virtex-4QV-Familie strahlungstolerant sein. Die FPGAs dieser Familie werden in einer 3D FinFET-Technologie und einer Strukturgröße von 16 nm gefertigt. Das Package entspricht dem des Virtex-5QV FPGAs (45 mm, 1 mm Pitch und 1752 Pins). Laut Huey ist die Auslieferung der ersten COTS-MPSoCs für das erste Halbjahr 2017 und die RT-Variante für das Jahr 2018 geplant. Das erste RT-MPSoC wird das RT-ZU19EG, das größte Zynq UltraScale+ MPSoC. In der Tabelle 5.5 sind die Kenngrößen dieses MPSoCs und zusätzlich ein Vergleich mit dem Virtex-5QV-FPGA dargestellt.

Der Vergleich der Kenngrößen des FPGA-Teils führt für die LCs zu einem Faktor von 8,7, für den On-Chip-Speicher zu einem Faktor von 6,5 und für die DSPs zu einem Faktor von 6,2. Weiterhin stehen dreimal so viele MGTs mit deutlich höherer Bandbreite zur Übertragung von Daten zur Verfügung. Laut Huey soll diese Familie daher mindestens für die nächsten 15 Jahre eine ausreichende Leistung bereitstellen.

Tabelle 5.5: Kenngrößen des RT-ZU19EG MPSoCs und zusätzlicher Vergleich mit dem Virtex-5QV-FPGA (basierend auf [85, S. 13])

	XQR5VFX130	RT-ZU19EG
Rollout	2011	2018
Strukturgröße	65 nm	16 nm
Prozesstechn.	Planar CMOS	3D FinFET
Strahlungsklass.	RH-BD	RT
Logic Cell (LC)	131 k	1 143 k
On-Chip-Speicher	12,3 Mbit	80,4 Mbit
DSPs	320	1 968
MGTs	18	52
(Bandbreite)	(4,5 Gbit/s)	(6/16/33 Gbit/s)
Prozessoren	Soft-IP-Cores	Quad-A53 Dual-R5 Mali-400 GPU Soft-IP-Cores
Gehäuse	45 mm, 1 mm Pitch, 1 752 Pins CCGA	

5.8.2 Atmel

Der Hersteller Atmel fertigt aktuell drei SRAM-basierte, strahlungsgehärtete FPGAs [30]: AT40KEL040, ATF280 und ATFS450. Der Konfigurationsspeicher, Flip-Flops und Benutzerspeicher (FreeRAM) wird in allen FPGAs durch *Dual Interlocked storage CELL* (DICE) geschützt. Analog zum Xilinx Virtex-5QV-FPGA, werden zwei Speicherzellen verwendet um ein Bit zu speichern. Somit kann ein Zustandswechsel nur dann erfolgen, wenn an beiden Dateneingängen der gleiche Wert anliegt. Weiterhin wird der Rekonfigurationskontroller durch TMR und die Takt- und Reset-Bäume durch DMR gegen SETs geschützt [15, S. 208].

AT40KEL040 Das AT40KEL040-FPGA basiert auf der kommerziellen AT40K-Architektur und wird in einem 350 nm Prozessverfahren gefertigt. Durch die Verwendung dieser älteren Fertigungstechnik wird ein Systemtakt von maximal 20 MHz erreicht [10]. Der konfigurierbare Logikblock wird bei Atmel (*core cell*) genannt und besteht aus zwei LUTs mit je drei Eingängen (LUT3), welche zu einer LUT mit vier Eingängen (LUT4) kombiniert werden können, einem 2-zu-1 Multiplexer sowie einem Flip-Flop. Das AT40KEL040 FPGA besitzt insgesamt 2 304 core cells. Zusätzlich stehen 18 kbit SRAM-Benutzerspeicher zur Verfügung, welche in 32x4 Blöcken organisiert sind. In Tests wurde für den FPGA eine TID von bis zu 300 krad und eine SEL-Immunität von mindestens $70 \frac{\text{MeVcm}^2}{\text{mg}}$ ermittelt [30].

ATF280 Das ATF280-FPGA wird in einem 180 nm Prozessverfahren gefertigt, wodurch der maximale Systemtakt auf bis zu 50 MHz gesteigert werden konnte [9]. In dem FPGA stehen 14 400 core cells zur Verfügung, wobei der Aufbau der *core cell* identisch zu dem Aufbau im AT40KEL040-FPGA ist. Ebenfalls ist die Anzahl des Benutzerspeichers auf 114 kbit erhöht worden. Eine Neuerung ist der sogenannte *Configuration Self Internal Checker* (CSIC), welcher die Datenintegrität des Konfigurationsspeichers zur Laufzeit sicherstellt. Durch die Verwendung eines Fehlererkennungscode (*Recurrent Checksum*) ist die Erkennung von SEUs im Konfigurationsspeicher möglich. Strahlungstests haben, zu dem AT40KEL040-FPGA, identische Ergebnisse bzgl. der TID (300 krad) sowie SEL-Immunität ($> 70 \frac{\text{MeV cm}^2}{\text{mg}}$) ergeben [30]. Im offiziellen Datenblatt [9] wird die SEL-Immunität mit $95 \frac{\text{MeV cm}^2}{\text{mg}}$ angegeben.

ATFS450 Das ATFS450-FPGA wird in einem 150 nm SOI-Prozessverfahren gefertigt. In [30] wurde das FPGA im September 2014 für das zweite Halbjahr 2015 angekündigt. Derzeit ist das FPGA weder auf der Herstellerseite aufgeführt noch ein offizielles Datenblatt verfügbar. Die Eckdaten werden in [30] mit ca. 21 000 core cells, 180 kbit und einem maximalen Takt von 70 MHz beschrieben. Erwähnenswert ist die angegebene TID von vergleichsweise geringen 100 krad.

Atmel bietet mit dem ATFee560 eine Kombination zweier ATF280-FPGAs sowie zweier EEPROM AT69170 zur Speicherung der Konfigurationsdaten in einem Gehäuse an. Mit dem ATF697 wird weiterhin eine Kombination eines ATF280-FPGA und eines AT697 LEON2-FT-Prozessors mit 100 MHz in einem Gehäuse gefertigt.

5.8.3 Microsemi

Microsemi bietet vier unterschiedliche strahlungstolerante FPGA-Familien in zwei Speichertechnologien (Flash und Antifuse) an [124]: Antifuse: RTSX-SU und RTAX sowie Flash: RT ProASIC3 und RTG4.

RTSX-SU Die RTSX-SU-Familie basiert auf der kommerziellen, in einem 250 nm CMOS-Prozessverfahren gefertigte, SX-A-Familie, wodurch pinkompatible FPGAs zum Prototyping verwendet werden können [128]. Im Gegensatz zu den kommerziellen Gegenstücken wurden die, bis zu 2 012 verfügbaren, Flip-Flops durch TMR gegenüber SEUs geschützt. Weiterhin existieren bis zu 4,024 Logikblöcke (*C-cells*) mit bis zu fünf Eingängen (LUT5) zur Realisierung kombinatorischer Logik. Die RTSX-SU-FPGAs haben laut [125] eine SEL-Immunität von $104 \frac{\text{MeV cm}^2}{\text{mg}}$ und eine TID 100 krad.

RTAX Die RTAX-Familie wird in einem 150 nm CMOS-Prozess gefertigt und ist in zwei Unterfamilien aufgeteilt: RTAX-S/SL und RTAX-DSP. In der RTAX-S/SL-Familie kann zwischen vier verschiedenen FPGAs ausgewählt werden. Die Anzahl der verfügbaren Logikblöcke (*C-cells*) variiert von 2 816 C-cells bis 40 320 C-cells mit je bis zu fünf Eingängen (LUT5). Die Anzahl der Register entspricht der Hälfte der C-cells. Zusätzlich

existiert eingebetteter Benutzerspeicher (*Embedded RAM*) in einer Größe von 54 kbit bis 540 kbit, welcher optional durch ein EDAC geschützt werden kann. Die RTAX-DSP-Familie enthält zwei FPGAs, welche die RTAX-S/SL um zusätzliche strahlungstolerante DSP-Blöcke (18x18 Multiply-Accumulate) erweitert. Die SET-Anfälligkeit der Register in der RTAX-S/SL-Familie konnte durch Verdreifachung der Ausgangspuffer um Faktor 16 verringert werden [128]. Die SEL-Immunität liegt bei $117 \frac{\text{MeV cm}^2}{\text{mg}}$, die TID bleibt bei mehr als 200 krad innerhalb der Spezifikationen. Bis 300 krad wurden keine funktionalen Ausfälle detektiert.

RT ProASIC3 Die RT ProASIC3-Familie basiert auf der kommerziellen ProASIC3-Familie und wird mit dem gleichen 130 nm Prozessverfahren gefertigt. Lediglich das Gehäuse ist ein hermetisch versiegeltes Keramikgehäuse [124]. Sogenannte *VersaTiles* können eine kombinatorische Funktion mit drei Eingängen (LUT3), ein Latch oder ein Flip-Flop realisieren. Zusätzlich existiert bis zu 504 kbit SRAM-basierter Benutzerspeicher (*RAM Blocks*) und 1 kbit Flash-basierter ROM [128]. Durch die sogenannte *Flash Freeze* Funktion kann der aktuelle Zustand eines FPGAs gespeichert werden (laut [128] in unter 1 μs) um in einen Stromsparmmodus versetzt zu werden. Durch die Verwendung von Flash als Konfigurationsspeicher, ist das FPGA vor SEUs im Konfigurationsspeicher geschützt. Dennoch können SEUs im VersaTile, bei einer Konfiguration als Flip-Flop, und in den RAM Blocks beobachtet werden [15, S. 204 ff.]. Weiterhin ist das FPGA anfällig für SETs. In den VersaTiles, den Taktleitungen und den IO-Blöcken wurden SETs detektiert. Die TID wird in [124] mit 25 krad bis 30 krad angegeben.

RTG4 Die RTG4-Familie wird in einem 65 nm Prozessverfahren gefertigt. Wie die RT ProASIC3-Familie bietet der Flash-basierte Konfigurationsspeicher einen Schutz vor SEUs [124]. Die drei FPGAs der RTG4-Familie verfügen über verschiedene Block-Typen [129]. Die *Logic Elements* umfassen eine LUT mit vier Eingängen (LUT4) und ein durch TMR-geschütztes Flip-Flop mit optionalem SET-Filter. Die Anzahl an LUT4 und Flip-Flops variiert zwischen 77,616 und 203,400. Weiterhin stehen zwei SRAM-basierte Benutzerspeicherblöcke mit optionalen EDAC (SECDED) in den Größen 18 kbit (LSRAM) und 1 kbit (uSRAM) zur Verfügung. Beide Speicherarten sind MBU-resistent [123]. Insgesamt stehen je nach FPGA zwischen 2,8 Mb und 6,7 Mb an SRAM zur Verfügung. Ein Flash-basierter *uPROM* mit einer Größe von 254 kbit bis 417 kbit kann zum Beispiel zur Speicherung von DSP-Koeffizienten verwendet werden und kann internen RAM oder Register initialisieren [128]. Die *Mathblocks* ähneln den DSP-Blöcken der RTAX-DSP-Familie und können mathematische Berechnung (18x18 MAC) durchführen. Optional können die Ein- und Ausgangsregister eines Mathblocks durch TMR vor SEUs geschützt werden. Ähnlich zu den Flip-Flops in den Logic Elements können SET-Filter aktiviert werden. Bei Verwendung des SET-Filters verringert sich der Takt von 300 MHz auf 250 MHz. Die SET-Filter eliminieren hierbei Glitches von bis zu 600 ps [123]. Die TID wird in [124] mit 100 krad angegeben. Dies wurde durch die Verwendung von sogenannten *Indirectly Coupled* Verbindungen erreicht [123]. Während den vorherigen

Familien das Datensignal direkt durch den Flash-Transistor geführt wurde (*Directly Coupled Interconnection*), wird dieses in der RTG4-Familie durch einen zusätzlichen Pass-Transistor geführt. Dieser Pass-Transistor wird durch zwei Flash-Transistoren in Push-Pull-Konfiguration geschaltet. Durch dieses Verfahren konnte die TID auf 100 krad erhöht werden.

Zusammenfassung

Tabelle 5.6 fasst die wesentlichen Kenngrößen der vorgestellten strahlungstoleranten bzw. -gehärteten FPGAs zusammen. Neben den fertigungsspezifischen Kenndaten, Speichertechnologie und Strukturgröße, werden die Auswirkungen von Strahlung durch TID und SEL aufgeführt. Um einen Vergleich bzgl. der Komplexität und Größe der FPGAs zu ermöglichen, wird die Anzahl der LUTs, der internen Speicherblöcke und die maximale Taktfrequenz dargestellt. Bei dem Vergleich der LUT muss die variierende Anzahl an LUT-Eingängen berücksichtigt werden (Xilinx: LUT4 oder LUT6, Atmel: LUT3, Microsemi: LUT3, LUT4 oder LUT5).

Tabelle 5.6: Kenngrößen der strahlungstoleranten und -gehärteten FPGAs

Hersteller	Familie	Speicher- techn.	Struktur- größe in nm	TID in krad	SEL-Immunität in $\frac{\text{MeVcm}^2}{\text{mg}}$	LUTs in k	BRAM in kbit	Taktfreq. in MHz
Xilinx	Virtex-4QV	SRAM	90	300	100	178 ¹	9 936	200
	Virtex-5QV	SRAM	65	1 000	100–130	82 ²	10 728	200
	Zynq RT US+	SRAM	16	-	-	523	34 600	-
Atmel	AT40KELO40	SRAM	350	300	70	2 ³	18	20
	ATF280	SRAM	180	300	95	14 ³	115	50
	ATFS450	SRAM	150	100	70	21 ³	180	70
Microsemi	RT ProASIC3	Flash	130	25–30	96	75 ³	504	250
	RTG4	Flash	65	100	110	203 ¹	6 700	300
	RTSX-SU	Antifuse	250	100	104	4 ⁴		230
	RTAX	Antifuse	150	100–300	117	40 ⁴	540	350

Legende:

- ¹ LUT4
- ² LUT6
- ³ LUT3
- ⁴ LUT5

5.9 Anwendungsbereiche

FPGAs sind zu einer weit verbreiteten Implementierungsplattform für digitale Schaltungen in vielen verschiedenen Anwendungsgebieten geworden. Die Hardware-Flexibilität von FPGAs bei gleichzeitig hoher Verarbeitungsgeschwindigkeit erschließt immer weitere Anwendungsgebiete, auch in rauen Umgebungen, in denen hoch-energetische

Strahlung herrscht. In diesem Abschnitt werden Anwendungsgebiete auf der Erde und dem Weltraum vorgestellt.

5.9.1 Terrestrische Anwendungsbereiche

Wie bereits in Abschnitt 5.1.2 erwähnt, ist die Strahlung auf der Erde in den meisten Umgebungen sehr gering. Dennoch kommt es auf der Erde durch die Wechselwirkungen der Sekundärstrahlung mit elektronischen Bauteilen zu unerwünschten Fehlverhalten. Der Einfluss auf besonders kritische Bereiche ist daher nicht zu vernachlässigen. Insbesondere wenn in einem System eine Vielzahl von FPGAs verwendet werden, steigt die Fehlerwahrscheinlichkeit.

Xilinx Rosetta Experiment

Im Jahr 2005 hat Xilinx das *Rosetta* Experiment initiiert [114, 203]. In dem Experiment werden die Ergebnisse von Strahlungstests durch Beschuss mit Neutronen oder Protonen im *Los Alamos Neutron Science Center* (LANSCE) und dem realen Einfluss der Sekundärstrahlung durch atmosphärische Neutronen kombiniert. Zur Beschleunigung des atmosphärischen Tests, werden Cluster SRAM-basierter FPGAs aus mehreren FPGA-Familien an zehn unterschiedlichen Orten mit variierender Höhenlage (-488 bis 3962 m) auf der Erde deponiert. Die Vorhersage der atmosphärischen Strahlung von Neutronen ist stark abhängig vom Standort (Höhen- und Breitengrad). In dem Standard JEDEC89A wurden daher Modelle festgelegt. Als Referenzort wird die Stadt New York und die Höhe 0 ü.NN festgelegt. Die Ergebnisse des Rosetta Experiments werden viertel- bis halbjährlich in dem *Device Reliability Report* [204] veröffentlicht. Als Maßeinheit wird von Xilinx FIT verwendet (siehe Abschnitt 5.7.1). Die FIT-Rate wird in dem Report in FIT/Mbit (Mbit = 10^6 bit) dokumentiert.

$$FIT/Mbit = \frac{N_{Fehler}}{10^9 h 10^6 bit} \quad (5.14)$$

Echt-/Langzeitmessung an unterschiedlichen Orten

Zur Bestimmung des Einflusses von atmosphärischer Strahlung wurden FPGAs in Hundertergruppen an unterschiedlichen Orten auf der Erde positioniert. in Tabelle 5.7 sind die Orte und deren Höhenlage aufgeführt.

Für den Test wird der Konfigurations- und Benutzerspeicher mit einem ausgewählten Muster beschrieben und anschließend durch periodisches Zurücklesen überwacht. Die Anzahl der FPGAs und die höheren Höhenlagen dienen als Beschleuniger des Experiments. Weitere Details zu dem genauen Testvorgang sind in [114] beschrieben.

Tabelle 5.8 stellt die FIT-Raten aktueller Xilinx FPGA-Familien aus [204] gegenüber. Für jede FPGA-Familie ist die Prozessstrukturgröße (*Strukt.*), sowie die FIT-Raten für den Konfigurationsspeicher (*CRAM*) und den Blockspeicher (*BRAM*) für die atmosphärische Strahlung und Alphastrahlung angegeben. Bis auf wenige Ausnahmen sind sinkende FIT-Raten bei der Verringerung der Strukturgröße erkennbar. Laut Xilinx liegt dies daran, dass Erkenntnisse aus dem Bereich der fehlertoleranten FPGAs in die Entwicklung neuer

Tabelle 5.7: Orte mit Höhenlage des Xilinx Rosetta Experiments [203]

Ort	Höhenlage	
	in ft	in m
San Jose, CA	257	78
Marseilles, France	359	109
Longmont, CO	4 958	1 511
Albuquerque, NM	5 145	1 568
Pic du Bure, France	8 196	2 498
Pic du Midi, France	9 298	2 834
Aiguille du Midi, France	11 289	3 441
White Mountain, CA	12 442	3 792
Mauna Kea, HI	13 000	3 962
Rustrel, France	-1 600	-488

COTS-FPGA-Familien eingeflossen sind [86]. Auffällig ist der deutliche Unterschied der FIT-Rate für Konfigurations- und Blockspeicher (bis zur 7-Serie).

Die FIT-Raten sind ferner als pessimistisch einzustufen, da nicht jeder Upset eines Bits zu einem funktionalen Fehler führt. Die Anzahl der ungenutzten und unkritischen Bits reduziert die Rate um den sogenannten *Device Vulnerability Factor* (DVF). Dieser Faktor liegt im Falle eines typischen Designs bei ca. 5 %, im schlimmsten Fall ist er niemals größer als 10 % [204]. Diese Werte sind darauf zurückzuführen, dass typischerweise weniger als 30 % der Konfigurationsbits in einem Design verwendet werden [86]. Der Wert wurde laut Xilinx durch unabhängige Kunden ermittelt.

Alphastrahlung Zur Erkennung von Alphastrahlung durch Kontaminationen in dem Gehäuse oder Substrat (siehe Abschnitt 5.1.2), wird der Standort der FPGA-Gruppen regelmäßig geändert. Eine konstante Upset-Rate durch Alphapartikel kann durch diese Höhenunabhängigkeit erkannt werden. Der Standort Rustrel in Frankreich befindet sich ca. 500 m u.NN unter einem Berg und ist daher vollständig vor kosmischer Strahlung geschützt. Somit liefert diese Einrichtung direkte Angaben zu Kontaminationen im Gehäuse. Die ermittelten Daten aus diesen Messungen werden, beginnend mit FPGAs der Virtex-6 und Spartan-6-Familie, ebenfalls in Tabelle 5.8 festgehalten. Xilinx verwendet für die Herstellung sogenannte *ultra-low alpha* Materialien, wodurch die FIT-Rate für den Konfigurationsspeicher immer unter 100 FIT/Mb bleibt. Die Rate liegt laut Xilinx um Faktor 100 bis 1000 höher, wenn diese Materialien nicht verwendet werden [86].

Strahlungstests in LANSCE Die Strahlungstests in LANSCE werden auf Basis des Standards JESD89A/89-3A durchgeführt. Mit den Tests wird der Wirkungsquerschnitt (siehe Abschnitt 5.2) der jeweiligen FPGAs gemessen. Trotz Miniaturisierung ist in den dokumentierten Werten aus [204] ein Abwärtstrend des Wirkungsquerschnitts (Cross

Tabelle 5.8: Ergebnisse zur atmosphärischen Strahlung und Alphastrahlung des Rosetta Experiments [204]

Familie	Strukt. in nm	Atmosphärische Strahlung			Alphastrahlung		
		in FIT/Mbit GRAM	BRAM	Fehler in %	in FIT/Mbit GRAM	BRAM	Fehler in %
VII	150	405	478	8			
VII Pro	130	437	770	8			
S3	90	190	373	-50, +80			
V4	90	263	484	11			
V5	65	165	692	-13, +15			
S6	45	179	375	-10, +11	88	172	-50, +100
V6	40	105	303	-11, +12	7	120	-47, +97
7S	28	85	77	-11, +12	33	30	-51, +129
US	20	38	66	-33, +54	16	30	-50, +100

Section) von Generation zu Generation erkennbar. Um einen konkreten Vergleich mit den Ergebnissen der Echtzeitmessung vornehmen zu können, muss eine Umrechnung des Wirkungsquerschnitts in FIT/Mbit erfolgen. Die folgende Berechnung aus [114] wird hierfür verwendet:

$$FIT/Mbit = Cross\ Section \cdot \frac{14}{cm^2 h} \cdot 10^6 \cdot 10^9 \quad (5.15)$$

Der Wert $\frac{14}{cm^2 h}$ beschreibt den Neutronen-Flux pro Quadratcentimeter pro Stunde für den Referenzstandort New York. Der Faktor 10^6 normiert den Wert auf ein Megabit, der Faktor 10^9 normiert den Wert auf eine Milliarde Stunden (siehe Abschnitt 5.7.1).

Tabelle 5.9 zeigt die Werte aus [204] zusammen mit den umgerechneten FIT/Mbit-Raten. Dabei haben bereits die ersten Experimente gezeigt, dass der Beschuss mit 10 MeV akkurate Ergebnisse für die Simulation von atmosphärischer Strahlung produziert. Vergleicht man die FIT-Raten der Echtzeitmessungen mit denen der Strahlungstests ergeben sich maximale Abweichungen von +75 % (Konfigurationsspeicher Virtex-5) und -43 % (Konfigurationsspeicher Spartan-3). Im Mittel sind die Werte der Echtzeitmessung im Falle des Konfigurationsspeichers um 6 %, die Werte des Blockspeichers um 23 % höher.

Vergleich der Strahlungstoleranz von ASICs und FPGAs

Im Vergleich zu ASIC-Komponenten in der gleichen Fertigungstechnologie sind die FIT-Raten von Xilinx FPGAs deutlich geringer. Dies liegt insbesondere daran, dass die SRAM-Zellen des Konfigurationsspeichers größer dimensioniert und robuster sind als

Tabelle 5.9: Wirkungsquerschnitt (nach [204]) und FIT/Mbit Ergebnisse der Strahlungstests in LANSCE

Fam.	Strukturgr. in nm	Wirkungsquerschnitt in $10^{-14} \text{cm}^2/\text{bit}$		FIT-Rate in FIT/Mbit	
		GRAM	BRAM	GRAM	BRAM
VII	150	2,56	2,64	358	370
VII Pro	130	2,74	3,91	384	547
S3	90	2,40	3,48	336	487
V4	90	1,55	2,74	217	384
V5	65	0,67	3,96	94	554
S6	45	1,00	2,20	140	308
V6	40	1,26	1,14	176	160
7S	28	0,69	0,63	97	88
US	20	0,25	0,44	36	62

herkömmliche SRAM-Zellen in Speicherbauteilen, welche auf Geschwindigkeit und Kosten optimiert sind [86].

In [249] erfolgt eine Gegenüberstellung für die 40 nm Strukturgröße. Für ASICs liegt die FIT-Rate pro Millionen Transistoren oder Speicherbits bei 5 000 FIT/Mbit, während sie bei der Virtex-6-Familie bei 105 FIT/Mbit im Falle des Konfigurationsspeichers, 303 FIT/Mbit bei dem Blockspeicher und weniger als 1 FIT/Mbit bei den Flip-Flops liegt. Die Gesamtfehlerrate eines Bauteils wird als *absolute FIT-Rate* oder Bauteil-FIT-Rate bezeichnet. Um die absolute FIT-Rate des Bauteils auszurechnen, wird die Anzahl der Transistoren und Speicherbits benötigt. Im Falle eines ASICs werden 5 bis 10 Millionen Transistoren und 10 bis 40 Millionen Speicherbits veranschlagt, wodurch sich eine obere Abschätzung der FIT-Rate von 250 000 FIT/Mbit ergibt. Statistisch bedeutet dies, dass pro Bauteil, ein Fehler alle 0,46 Jahre vorkommt. Der konkrete Vergleich endet in dem Dokument an dieser Stelle. Um einen fairen Vergleich durchführen zu können, muss die absolute FIT-Rate von Virtex-6 FPGAs berechnet werden. Die Gesamtfehlerrate eines FPGAs wird allgemein wie folgt berechnet:

$$FIT_{Total} = FIT_{GRAM} + FIT_{BRAM} + FIT_{FF} \quad (5.16)$$

Die Gleichung kann für eine Approximation vereinfacht werden, da die FIT-Rate für die Flip-Flops zum einen gering ist und zum anderen die Anzahl der Flip-Flops selbst bei dem größten FPGA bei etwa fünf Millionen liegt. Somit würde sich die absolute FIT-Rate maximal im einstelligen Bereich ändern.

$$FIT_{Total} = FIT_{GRAM} + FIT_{BRAM} \quad (5.17)$$

Die Anzahl der Jahre, bis es zu einem Upset kommt, lässt sich durch die Umkehrfunktion und Umrechnung der Stunden zu Jahre ermitteln und entspricht der in Abschnitt 5.7.1 definierten *Mean Time Between Failures* (MTBF).

$$MTBF = \frac{1}{FIT_{Total}/24 \cdot 365 h} \quad (5.18)$$

In Tabelle 5.10 sind die minimale und maximale Anzahl an Bits für den Konfigurations- und Blockspeicher für die jeweils kleinsten und größten Virtex-6 FPGAs dargestellt. Zusätzlich ist in der dritten Spalte die FIT-Rate aus [204] angegeben. In der vierten Spalte ist das Ergebnis der minimalen und maximalen FIT-Raten aufgeführt. Die kumulierten Werte aus Tabelle 5.10 ergeben eine FIT_{Total} von 4 447 FIT/Mbit bzw. 31 008 FIT/Mbit. Dies entspricht einer MTBF von 25,67 Jahren bzw. 3,68 Jahren .

Tabelle 5.10: Daten zur Berechnung der absoluten FIT-Rate für Xilinx Virtex-6 FPGAs

	min. Anzahl		max. Anzahl		FIT-Rate in FIT/Mbit	$FIT_{CRAM/BRAM}$	
	in Mb	FPGA	in Mb	FPGA		min	max
CRAM	26,2	LX75T	184,8	LX760	105	2751	19404
BRAM	5,6	LX75T	38,3	SX475T	303	1696	11604

Die Ergebnisse aus Tabelle 5.10 ergeben bei einem Vergleich für die obere Grenze, dass der ASIC um Faktor 8,1 anfälliger ist (250 000 FIT/Mbit aus [249] gegenüber 31 008 FIT/Mbit bei einem Virtex-6-FPGA). Ein Vergleich der unteren Grenzen ergibt einen Faktor von 16,9 (ASIC: 75 000 FIT/Mbit vs. FPGA: 4 447 FIT/Mbit). Diese Ergebnisse relativieren somit die stark abweichenden FIT-Raten zwischen ASICs und Xilinx FPGAs in einer vergleichbaren Fertigungstechnologie.

Entwicklung der Strahlungstoleranz der Xilinx COTS-FPGA-Familien

Die wachsende FPGA-Größe von Generation zu Generation (siehe Abschnitt 3.2) resultiert in einem entsprechenden Anstieg an Konfigurationsspeicherbits. Eine interessante Fragestellung ist daher, wie dieser Anstieg der Logikgröße gegenüber den sinkenden FIT-Raten skaliert. In [249] wird diese Fragestellung als Ansporn verwendet. Die Gesamtfehlerrate eines FPGAs soll trotz Anstieg an Konfigurations- und Blockspeicher durch eine jeweils geringe FIT-Rate von Generation zu Generation sinken.

Tabelle 5.11 gibt Antworten auf diese Fragestellung und zeigt, dass das Vorhaben nicht eingehalten werden konnte. Für die einzelnen Familien sind jeweils das kleinste und größte FPGA dargestellt. Betrachtet man die Entwicklung der MTBF von der Virtex-4-Familie an, ist mit Ausnahme der Spartan-6-Familie, ein stetiger Abwärtstrend erkennbar. Bei den größten FPGAs der jeweiligen Familien fällt die MTBF von 6,2 Jahre auf 2,3 Jahre. Bei den kleinsten FPGAs ist ein ähnlicher Trend erkennbar, wenn die kleineren Artix-7-FPGAs nicht berücksichtigt werden. Die Abbildung 5.26 stellt den

Tabelle 5.11: Absolute FIT-Rate und MTBF für die jeweils kleinsten und größten FPGAs der aktuellen Xilinx FPGA-Familien

FPGA-Familie	CRAM in Mbit		BRAM in Mbit		FIT_{Total} in FIT/Mbit		MTBF in Jahre	
	min	max	min	max	min	max	max	min
V4	4,7	51,3	0,6	10,0	1 526	18 332	74,8	6,2
V5	8,3	82,7	0,9	18,6	1 992	26 517	57,3	4,3
S6	2,7	33,9	0,2	4,8	558	7 868	204,4	14,5
V6	26,2	184,8	5,6	38,3	4 448	31 009	25,6	3,6
7S	17,5	447,3	0,9	67,7	1 557	43 233	73,3	2,6
US	128,1	1 031,7	12,7	132,9	5 706	47 976	20,0	2,3

gegenläufigen Trend der absoluten FIT-Rate und der MTBF für die größten FPGAs der untersuchten FPGA-Familien dar. Die dargestellten Linien deuten den jeweiligen Trend mit größtem Bestimmtheitsmaß an.

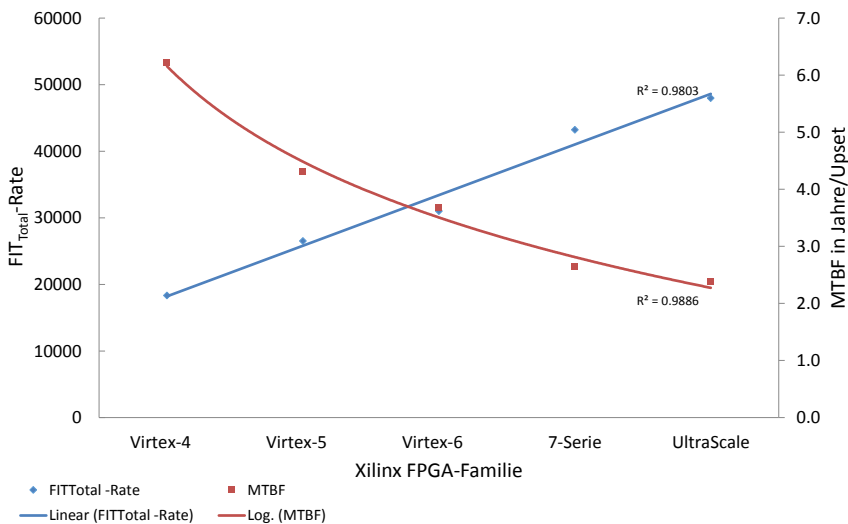


Abbildung 5.26: Entwicklung der absoluten FIT-Raten und der MTBF für aktuelle Xilinx FPGA-Familien

Die Reduzierung der FIT-Raten trotz Miniaturisierung sind auf der einen Seite beachtlich auf der anderen Seite jedoch notwendig. Die Berechnung der MTBF der UltraScale FPGAs mit den FIT-Raten älterer Xilinx Architekturen verdeutlicht dies. Werden die

FIT-Raten eines Virtex-4 für die UltraScale FPGAs verwendet, liegt die MTBF eines FPGAs nur noch zwischen 2,9 Jahren und 0,3 Jahren.

Weitere Einflüsse auf die Fehleranfälligkeit eines Systems

In [201] werden zwei weitere Szenarien betrachtet. Das erste Szenario betrachtet ein System, das aus einer Vielzahl der Bauteile besteht, wodurch die Fehlerwahrscheinlichkeit linear mit der Anzahl der Bauteile steigt. In dem Szenario wird ein Prozessor mit einer vergleichsweise geringen, absoluten FIT-Rate von 600 FIT/Mbit (vgl. Werte aus Tabelle 5.11) verwendet. Als Standort wird die Stadt New York und die Höhe 0 ü.NN betrachtet. Die MTBF von tausend solcher Systeme fällt auf 70 Tage, was bedeutet, dass es im Durchschnitt alle 70 Tage zu einem Upset kommt. Das zweite Szenario variiert den Einsatzort und somit die Höhenlage desselben Prozessors. Als Standort wird ein Orbit in 40 kft ($\approx 12,2$ km) über den Polen gewählt, wodurch sich die absolute FIT-Rate auf 367 200 FIT/Mbit erhöht. Dies entspricht einer MTBF von 110 Tagen. Xilinx fasst nach der Berechnung zusammen, dass die MTBF beider Szenarien in der gleichen Größenordnung liegen und somit ein System in der Luft ähnlich anfällig ist wie tausend Systeme auf der Erde. Da die Strahlung in dem Weltraum sehr stark variiert (Standort und kosmische Strahlung), ist diese Schlussfolgerung an sich etwas fragwürdig. Dennoch verdeutlichen die Szenarien den Einfluss der beiden Variablen: Anzahl der Bauteile und Standort eines Systems.

Die absoluten FIT-Raten bzw. die MTBF für FPGAs oder Prozessoren deuten bereits an, dass diese in einigen Anwendungsgebieten auf der Erde nicht hinnehmbar sind. Somit wird je nach Anzahl, Typ und Standort der System-Bauteile die Verwendung von Abschwächungstechniken (siehe Abschnitt 5.6) auf unterschiedlichen Ebenen benötigt, um die Fehlertoleranz des Gesamtsystems zu erhöhen.

Hochenergiephysik

Ein terrestrischer Anwendungsbereich mit teils höherer Strahlung als im kosmischen Anwendungsbereich ist in der Hochenergiephysik (HEP) zu finden. Dieses Anwendungsfeld besteht aus Experimenten zur Untersuchung der Eigenschaften und Wechselwirkungen von Elementarteilchen, wie z. B. Quarks oder Leptonen [190]. Die Experimente verwenden Teilchenbeschleuniger um geladene Partikel zunächst zu beschleunigen und anschließend kollidieren zu lassen. Die subatomare Struktur der Nebenprodukte einer solchen Kollision werden anschließend durch Partikeldetektoren analysiert. Diese Detektoren messen die Energie, Richtung, Effekt oder Spin der verschiedenen Partikel. In den Detektoren müssen die Daten mit hoher Geschwindigkeit aufgenommen und zur Weiterverarbeitung an einem externen Computer schnellstmöglich transferiert werden. FPGAs eignen sich aufgrund ihrer Struktur hervorragend für diese Aufgaben, sodass sie eine breite Verwendung in den Detektoren gefunden haben. Konkret, werden FPGAs an Sensoren, Analog-Digital-Umsetzer (engl. Analog-to Digital Converter, ADC) angeschlossen, messen Zeitdifferenzen im Nanosekundenbereich und transferieren die Daten über Hochgeschwindigkeitsschnittstellen an einen Computer. Beispiele für

den Einsatz von FPGAs in Teilchendetektoren sind die Experimente ATLAS (A Toroidal LHC ApparatuS), CMS (Compact Muon Solenoid) und ALICE (A Large Ion Collider Experiment) des Teilchenbeschleunigers LHC (*Large Hadron Collider*) im europäischen Kernforschungszentrum CERN bei Genf.

Durch die Kollisionen beschleunigter Partikel wird ein starkes Strahlungsfeld generiert. Die Strahlung ist dabei abhängig von dem jeweiligen Experiment und dem Standort des Detektors. Da die Strahlung in der Nähe der Kollision sehr hoch ist, werden FPGAs typischerweise an Stellen mit geringerer Strahlung eingesetzt. Die Strahlung in HEP Experimenten ist relativ konstant, so dass die zu erwartenden Upsets leichter vorhergesagt und entsprechende Abschwächungsmethoden verwendet werden können. Zusätzlich können defekte Bauteile bei einer Wartung ersetzt werden. Abbildung 5.27 zeigt die Strahlungsdosen einiger Experimente des LHCs und markiert (mit roten Pfeilen) ebenfalls die typischen Strahlungsdosen auf der Erde und im Weltraum. Zusätzlich grenzt die Darstellung die Bereiche der Verwendung von COTS- und strahlungsgehärteten Komponenten ein.

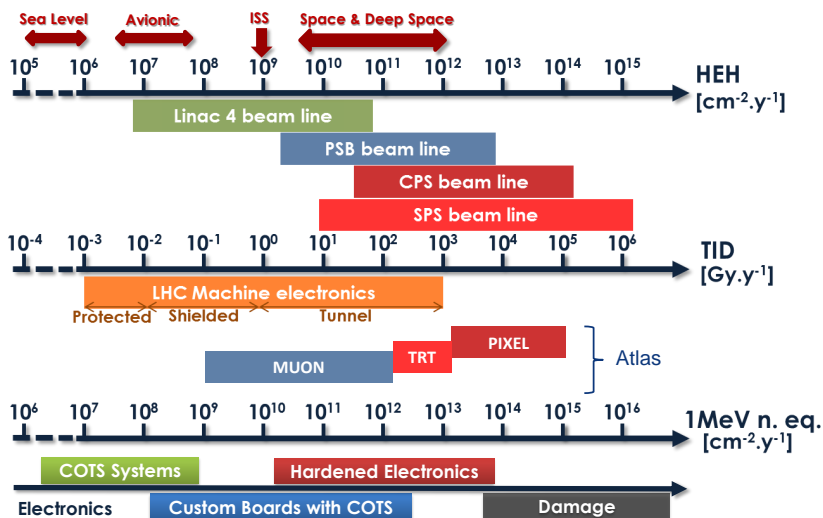


Abbildung 5.27: Übersicht der Strahlung im LHC im europäischen Kernforschungszentrum CERN [50]

In [50] werden die größeren Experimente des LHCs vorgestellt, in denen jeweils Tausende COTS-FPGAs eingesetzt werden. Mithilfe der Experimente ATLAS und CMS wurde im Juli 2012 eine der Hauptaufgaben des LHCs erreicht: Der Nachweis der Existenz des Higgs-Boson, dem letzten Teilchen des Standardmodells der Teilchenphysik.

ALICE

Ein besonders erfolgreiches Beispiel für die Verwendung von FPGAs in einem HEP Experiment ist der Teilchendetektor in dem ALICE des LHCs [146]. Dieser Detektor analysiert sogenanntes Quark-Gluon-Plasma, welches unmittelbar nach dem Urknall (engl. Big Bang) existiert haben soll. Der wesentliche Teil des Detektors ist die Spurendriftkammer (engl. Time Projection Chamber, TPC). Die FPGA-basierte Datenverarbeitungseinheit hat die Aufgabe, die Sensorwerte auszunehmen und anschließend zu transferieren. Die Datenmenge liegt laut Wirthlin bei ca. 150 GB [190].

Die Verarbeitungseinheit besteht aus 216 einzelnen Kontrolleinheiten, welche jeweils ein Xilinx Virtex-II PRO FPGA verwenden. Die Kontrolleinheiten übernehmen die Aufgabe der Datenkompression, digitale Datenverarbeitung und Datenpufferung. Durch diese Aufgabenbereiche sind die FPGAs vollständig ausgenutzt, so dass keinerlei Abschwächungsstrategien implementiert werden konnten. Das FPGA wird während des Betriebs auf Upsets untersucht. Wird ein Upset erkannt, wird dieser vermerkt. Führt ein Upset zu einem Ausfall der Kontrolleinheit, wird diese vollständig rekonfiguriert. Bevor das System in den LHC integriert wurde, wurden Fehlerinjektionstests durchgeführt, um die Robustheit der Kontrolleinheit zu ermitteln. Dabei wurde festgestellt, dass lediglich einer von 93 Upsets im Konfigurationsspeicher zu einem Ausfall führt. Zur Erkennung eines Upsets wird der Konfigurationsspeicher zurückgelesen, was in dem System 150 ms dauert. Die FPGAs werden im Betrieb als Upset-Sensoren benutzt und nehmen den konkreten Ort des Upsets in einer Logdatei auf. In [146] wurde im Vorfeld eine worst-case Abschätzung von 75 Upsets in einem Testdurchlauf (4 Stunden) aller 216 FPGAs berechnet. In [190] wird nach der Installation im LHC für das gesamte Jahr 2011 ein Wert größer 1 600 Upsets dokumentiert.

ATLAS

Weitere Beispiele sind in dem ATLAS Experiment des LHCs zu finden. Die steigenden Datenraten in den Detektoren können nur mit aktuellen Transceivern bewältigt werden. Der Einsatz von GTX Transceiver der Xilinx 7-Serien FPGAs wird in [34] anhand des Telexperiment *Liquid Argon* (LAr) des ATLAS evaluiert. Die 16 GTX eines Kintex-7 325T FPGAs werden dazu in der Einrichtung *The Svedberg Laboratory* (TSL) in Uppsala, Schweden mit einem 180 MeV Protonenstrahl getestet. Pro Transceiver ist eine Datenrate von bis zu 12,5 GB/s möglich, so dass sich pro FPGA eine maximale Datenrate von 200 Gb/s ergibt. Die Strahlung in ATLAS variiert laut Cannon et al. je nach Abstand zum Kollisionspunkt zwischen 0.1 und 100 Mrad und 1×10^{15} Neutronen/cm². Das FPGA wurde in TSL einer Strahlung von $1,55 \times 10^{13}$ Protonen/cm² in einer Zeitspanne von 24 Stunden, drei Tage mit je acht Stunden, ausgesetzt. Die Auswertung des Strahlungstests ergab, dass die größte Anzahl an Fehlern durch Upsets in dem Konfigurationsspeicher verursacht wurde. Dies bedeutet, dass der Wirkungsquerschnitt der Transceiver gering ist und die Fehler durch Upsets in der angeschlossenen, programmierbaren Logik erfolgten. Diese könnten durch Abschwächungsverfahren auf Designebene

deutlich reduziert werden. Während der Tests kam es weiterhin zu elf SEFI-Fehlern, welche durch einen *power cycle* behoben werden konnten (siehe Abschnitt 5.4.1).

Besonders interessante Ergebnisse ergaben die Auswertung der Strommessung. Als einziges Abschwächungsverfahren wurde *Blind Scrubbing* und *Readback Scrubbing* angewendet (siehe Abschnitt 5.6.4). Die Hälfte der Testzeit wurde das Scrubbing deaktiviert, um so den Mehraufwand dieses Verfahrens beurteilen zu können. Bei aktiviertem Scrubbing stieg die Stromaufnahme des FPGAs konstant um 10 %. Bei deaktiviertem Scrubbing hingegen stieg die Stromaufnahme kontinuierlich an. Am Ende des Testlaufs (acht Stunden) war die Stromaufnahme doppelt so hoch. Laut Cannon et al. liegt dies daran, dass durch Upsets inaktive Teile des FPGAs aktiviert werden.

Ein weiteres Ergebnis liefert die Betrachtung der Strahlungsdosis. Das FPGA wurde während des Tests einer Strahlungsdosis von etwa 1 Mrad ausgesetzt. Es wurden in dem Test dieses COTS-FPGAs keine permanenten Fehler in der programmierbaren Logik festgestellt. Es wurde lediglich ein geringer Anstieg in der Stromaufnahme an der Kernspannungsversorgung festgestellt, welcher jedoch innerhalb der Spezifikation lag. Mit den Auswertungen der Strahlungstests wurde exemplarisch die Fehlerrate für den Einsatz in dem ATLAS LAr berechnet. Die MTBF einer GTX-Leitung liegt bei 234 Tage. Für den Detektor in dem LAr werden ca. 20 000 GTX-Leitungen benötigt. Bei Verwendung des gleichen FPGA-Typs würden 1 250 FPGAs benötigt, wodurch die MTBF einer Leitung auf ca. 17 Minuten sinkt. Diese Berechnung ist aus zwei Gründen pessimistisch: Zum einen werden in dem FPGA-Design, bis auf das Scrubbing, keine Abschwächungsmethoden verwendet. Zum anderen könnte die Einzelfehlerrate durch die Verwendung eines größeren FPGAs mit mehr GTX-Transceivern deutlich reduziert werden.

Aufgrund der steigenden Anforderung bzgl. der Auflösung des LHCs ab 2015, wird eine neue Komponente, der sogenannte *Level-1 Trigger Topological Processor* (L1Topo), dem ATLAS Experiment hinzugefügt [17]. Die Aufgabe des L1Topo ist die Weiterverarbeitung von vorverarbeiteten Daten existierender Detektoren mit topologischen Algorithmen. Eine weitere besondere Anforderung ist die hohe Eingangsbandbreite von ca. 1 Tb/s. Diese beiden Aufgaben werden in einem ersten Prototyp von zwei Xilinx Virtex-7 VX485T COTS-FPGAs übernommen.

5.9.2 Kosmische Anwendungsbereiche

FPGAs wurden im Weltraum ursprünglich als *Glue Logic* zur Verbindung zweier zunächst inkompatibler Hardwarekomponenten, eingesetzt. Durch das stetig steigende Datenvolumen in einem Satellitensystem wird heutzutage primär die Verarbeitungsgeschwindigkeit von FPGAs benötigt [190]. Dies wird insbesondere dadurch verstärkt, dass die Verbindungsgeschwindigkeit für die Übertragung zur Erde sehr gering ist. Neben der eigentlichen Datenverarbeitung werden die Daten daher im Vorfeld im Satelliten komprimiert. Im Gegensatz zu Anwendungen im HEP Bereich, ist die Strah-

lung im Weltraum durch Sonnenaktivitäten stark schwankend und abhängig von der Betriebsposition bzw. dem Orbit (siehe Abschnitt 5.1).

Übersicht aktueller ESA-Missionen

Gardenyes analysiert in [66] den Trend der Hardwarekomponenten in aktuellen Weltraummissionen der ESA. Tabelle 5.12 führt die Missionen chronologisch auf. Für jede Mission wird der Start, die Laufzeit, die Kosten, das Gewicht und der Orbit der Mission dargestellt. Abbildung 5.28 zeigt die absolute Anzahl der Komponenten der einzel-

Tabelle 5.12: Aktuelle Weltraummissionen der ESA [66, S. 32]

Mission	Start im Jahr	Laufzeit in a	Kosten in M€	Gewicht in kg	Orbit
Ariane 5	1997	-	8 000	746 000	GEO
Rosetta	2004	12	1 000	3 000	Interpl.
Venus Express	2005	9	220	1 240	Interpl.
GOCE	2009	1,7	350	1 050	LEO
Immarsat 4	2009	13	1 200	5 960	GEO
Hylas	2010	15	120	2 242	GEO
Galileo IOV	2011	12	1 512	700	MEO
Vega	2012	-	710	138 000	LEO
Proba V	2012	2,5	60	160	LEO
Sentinel 2	2013	7	435	1 200	LEO
Bepicolombo	2014	7,5	970	1 140	Interpl.

nen Missionen. Anzumerken ist hierbei, dass die Missionen dabei nach der Anzahl der Komponenten und nicht chronologisch angeordnet sind. Im Mittel bestehen die IC-Komponenten der Weltraummissionen zu 50 % aus FPGAs. Werden die Galileo IOV und Immarsat 4 Missionen nicht betrachtet, liegt der Anteil sogar bei 60 %.

Der Trend der verwendeten IC-Komponenten ist in Abbildung 5.29 dargestellt und zeigt, dass die Verwendung von FPGAs stetig anwächst, während der Anteil der (Standard-) ASICs immer weiter zurückgeht.

Eine repräsentative Mission ist die *Sentinel 2* Mission. In dieser Mission wird deutlich, dass FPGAs eine wesentliche Rolle in Datenverarbeitungs- und Platformanwendungen eingenommen haben. Tabelle 5.13 führt die konkrete Anzahl der Komponenten für die beiden genannten Bereiche auf. Gesamt betrachtet, liegt der FPGA-Anteil bei 58 %, der Custom-ASIC-Anteil bei 29 %, der Mikroprozessoranteil bei 9 % und der Standard-ASIC-Anteil bei 4 %. Im Datenverarbeitungsbereich werden FPGAs mit einem Anteil von 86 % eingesetzt.

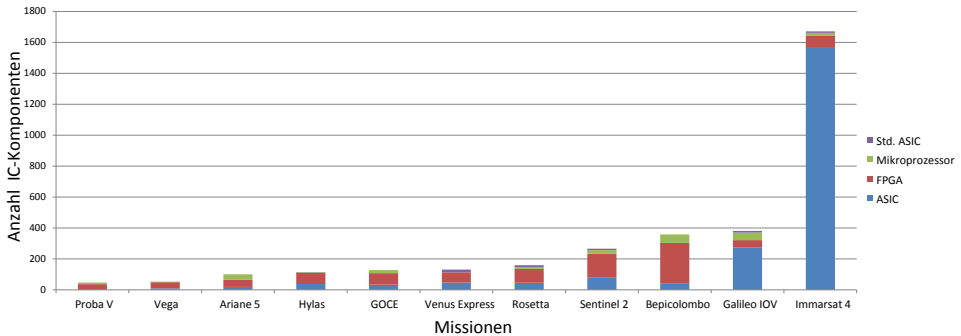


Abbildung 5.28: Absolute Anzahl der verwendeten IC-Komponenten in aktuellen ESA Missionen (basierend auf [66, S. 64])

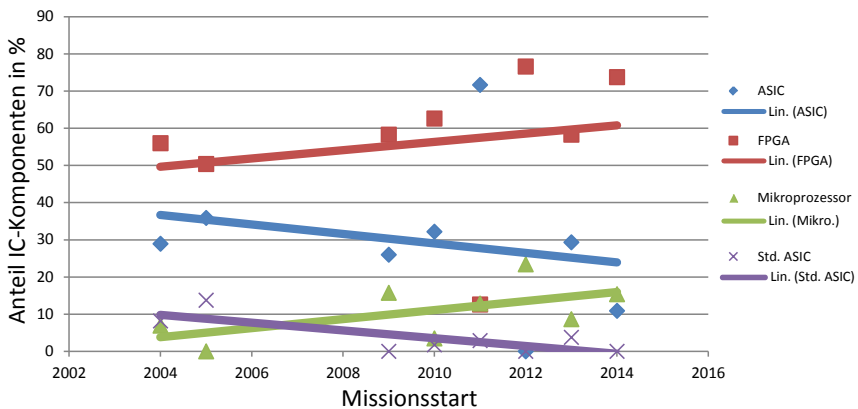


Abbildung 5.29: Trend der verwendeten IC-Komponenten in aktuellen ESA Missionen (modifizierte Darstellung aus [66, S. 76])

Übersicht FPGA-basierter Weltraummissionen

Viele Weltraummissionen in denen Steuerungsaufgaben von dem FPGA übernommen werden, sind proprietärer Natur und Informationen daher nicht publiziert. Im folgenden werden chronologisch Missionen aufgeführt, die öffentliche Ergebnisse produziert haben bzw. produzieren werden und somit Meilensteine in der Geschichte FPGA-basierter Weltraummissionen darstellen.

FedSat

Der australische Wissenschaftssatellit FedSat war der erste Satellit, in dem ein FPGA zur Berechnung von Standardaufgaben verwendet wurde [24]. Am 14. Dezember 2002

Tabelle 5.13: Anzahl der Komponenten der Sentinel 2 Mission [66]

IC-Typ	Plattform (in #)	Datenverarbeitung (in #)
FPGA	118	37
Custom-ASIC	72	6
Mikroprozessor	23	0
Standard-ASIC	10	0

wurde der Satellit auf einem Orbit in 800 km Höhe ausgesetzt [63]. Der Satellit verfügte über fünf Datenverarbeitungseinheiten. Mit dem HPC-I (*High Performance Computing*) Experiment übernahm erstmals eine rekonfigurierbare Verarbeitungseinheit, ein Xilinx XQR4062XL FPGA, Standardaufgaben in einem Satelliten. Das FPGA verfügte über einen Readback Controller, wodurch Upsets durch Strahlung erkannt werden konnten [178]. Im April 2007 fiel die Batterie des Satelliten nach über einer Milliarde geflogener Kilometer aus. Die von dem Satelliten ermittelten Daten führten zu 37 Dissertationen und über 750 wissenschaftlichen Veröffentlichungen.

Mars 2003 Landeeinheit und Rover

Der erste Einsatz in einem missionskritischen System war die Mars 2003 Landeeinheit und der Mars Rover des *Jet Propulsion Laboratory* (JPL) [178, 248]. In der Landeeinheit wurde unter anderem die Pyrotechnik, der Fallschirm und die Sprengbolzen über vier Xilinx XQR4062XL FPGAs gesteuert. In den Rovern *Spirit* und *Opportunity* erfolgt die Ansteuerung der Motoren ebenfalls über ein XQVR1000 FPGA. Als Abschwächungstechniken werden TMR, Readback zur SEU-Erkennung und (vollständiges) Blind Scrubbing zur SEU-Korrektur angewendet. Weiterhin waren die FPGAs der Landeeinheit während des siebenmonatigen Flugs zum Mars aktiv, sodass erstmals wichtige Daten wie SEU-Raten ermittelt werden konnten [248].

Cibola Flight Experiment

Das *Cibola Flight Experiment* (CFE) der Einrichtung *Los Alamos National Laboratory* (LANL) wird in einem Satelliten in 560 km Höhe (LEO) betrieben [33]. Das im März im Jahr 2007 gestartete Experiment besteht aus drei rekonfigurierbaren Verarbeitungseinheiten (engl. *ReConfigurable Computer modules*, RCC). Jedes RCC enthält jeweils einen strahlungsgehärteten Prozessor (BAE Systems R6000 mit 30 MHz), ein strahlungstolerantes Antifuse-FPGA (Actel, nun Microsemi RT54SX32S) und drei SRAM-basierte FPGAs (Xilinx Virtex XQVR1000) mit je 288 MB SDRAM (Hyundai, 3 Bänke mit jeweils vier 64 Mb). Der Satellit ist ein Hochfrequenzinstrument (engl. *Radio Frequency*, RF) mit zwei Antennen zur Verarbeitung von Zwischenfrequenzen (engl. *Intermediate Frequency*, IF) der Ionosphäre.

Das Antifuse-FPGA integriert einen Watchdog und dient als Rekonfigurationskontrolller für die drei Xilinx FPGAs. In dem Experiment wurden erstmalig vier unterschiedliche

Methoden zur Echtzeit SEU-Erkennung verwendet und vorab geschätzte (CREME96) und in dem Satelliten gemessene SEU-Raten gegenübergestellt. Die erste Methode, SEU1, verwendet ausschließlich Readback und Blind Scrubbing durch das externe Actel FPGA. In SEU2, der zweiten Methode, wird das Verfahren *Duplicate With Compare* (DWC) oder auch *Dual Modular Redundancy* (DMR) (siehe Abschnitt 5.6.2) verwendet um SEUs zu detektieren. Somit wurde erstmals eine SEU-Erkennung innerhalb des FPGAs realisiert. SEU3 erweitert die SEU2-Methode um die SEU-Erkennung in BRAM-Blöcken durch schreiben und zurücklesen eines festen Musters. SEU4 wiederum erweitert SEU3 um die Fehlererkennung und -korrektur des angeschlossenen SDRAMs.

In [33] wird die SEU-Rate auf 0,5 bis 26 Upsets pro FPGA pro Tag geschätzt. Die starke Schwankung ist den unterschiedlichen Bedingungen und Standorten auf dem Orbit geschuldet. Während des Betriebs wurden tatsächlich (bis zur Veröffentlichung 2009) 759 SEUs in 2 830 Gerätetagen detektiert. Dies führt zu einer deutlich geringeren SEU-Rate von 0,27 pro FPGA und Tag. Die abgeschätzte SEU-Rate weicht damit stark von der realen SEU-Rate ab. Die durchschnittliche SEU-Rate des gesamten RRC-Moduls liegt durch die neun FPGAs bei 2,4 SEUs pro Tag. Das Experiment wurde für eine Laufzeit von vier bis fünf Jahren angesetzt.

In [138] werden aktuellere Werte (Stand Mai 2014) des Satelliten vorgestellt. Der Satellit hat somit die geplante Laufzeit bereits nahezu verdoppelt. Die Anzahl der registrierten SEUs bis Mai 2014 wird mit 2 816 beziffert, wodurch sich die SEU-Rate des Satelliten je nach Strahlung zwischen 1,90 und 3,51 SEUs pro Tag bewegt. Die erste mittlere Abschätzung von 2,4 SEUs wurde somit nach weiteren fünf Betriebsjahren bestätigt. Die Abweichung der realen zur abgeschätzten SEU-Rate wird ebenfalls diskutiert. Laut Quinn et al. liegt die starke Abweichung an der letztendlichen Verwendung der FPGAs in dem Satelliten [138]. Zum einen wurden die FPGAs nur an 1 064 von möglichen 2 330 Tagen verwendet, sodass SEUs nur in 46 % der Zeit detektiert werden konnten. Zum anderen wurde in der Zeit von 2007 bis 2009 die Verwendung der FPGAs in der SAA verhindert, sodass gerade der strahlungsintensive Bereich des Orbits ausgespart wurde. Durch Änderungen in der Magnetosphäre hat der CFE-Satellit den ursprünglichen Orbit verlassen und befindet sich in einem leichten Sinkflug. Quinn et al. nehmen an, dass sich der Satellit 2015 bereits auf einem Orbit in Höhe von 500 km befindet, was eine Senkung der SEU-Rate von 30 % Prozent zur Folge hat. Im Jahr 2023 wird der Satellit voraussichtlich in die Erdatmosphäre eintreten.

Erwähnenswert ist, dass LANL die Fähigkeit der Rekonfiguration stark verwendet. So wird erwähnt, dass zweimal im Monat neue Konfigurationsdateien für die FPGAs auf den Satelliten geladen werden, um neue Erkennungsalgorithmen oder Abschwächungsverfahren zu erproben.

SpaceCube

SpaceCube ist ein FPGA-basiertes Datenverarbeitungssystem des NASA *Goddard Space Flight Centers* (GSFCs) [141]. Das Ziel des Programms ist die Erhöhung der Datenverarbeitungsgeschwindigkeit in Satelliten um ein bis zwei Größenordnungen. Tabelle 5.14

zeigt die unterschiedlichen Versionen des SpaceCubes, beginnend mit Version 1.0a im Mai 2009, welche 60 Stunden in der *Hubble Servicing Mission 4* im Betrieb war. Dabei werden erstmalig COTS-FPGAs der Virtex-4-Familie (FX60) verwendet.

Das zweite SpaceCube-Experiment (1.0b) diente der Validierung von neuen Abschwächungstechniken *Radiation Hardened By Software* (RHBS) auf Basis von COTS-FPGAs und wurde im November 2009 im Rahmen des *Materials on the International Space Station Experiment 7* (MISSE7) an der Internationalen Raumstation ISS montiert. In [64] wird die Anzahl der SEUs bis Juni 2012 auf 175 beziffert, in [133] wird eine SEU-Rate von 0.09 SEUs pro Tag und FPGA genannt. Mit Version 2.0 werden zwei Xilinx Virtex-5 FPGAs (FX130T) auf das Basisboard integriert. Durch Pinkompatibilität kann die kommerzielle oder die RHBD-Variante des FPGAs eingesetzt werden. Zusätzlich werden die Schnittstellen/Kommunikationsprotokolle SpaceWire und cPCI in das eingebettete System integriert. Die Version 2.0 wurde 2014 ebenfalls auf der ISS im Rahmen des *ISS SpaceCube Experiment 2.0* (ISE 2.0) in Betrieb genommen.

Tabelle 5.14: SpaceCube Versionen und Missionen [64]

Version	Mission	Start Jahr	FPGA(s)	Leistung in W
1.0a	Hubble Servicing M4	2009	2 x V4 FX60	37
1.0b	MISSE-7 (ISS)	2009	2 x V4 FX60	32
1.5	SMART (DoD/ORS)	2011	V5(Q) FX130T	< 20
1.0c	Argon Ground Demo.	2011	2 x V4 FX60	40
1.0d	STP-H4 (ISS)	2011	2 x V4 FX60	40
2.0	ISE 2.0 (ISS)	2013	2 x V5(Q) FX130T	15 - 20

Mission Response Module

Das *Mission Response Module* (MRM) ist ein weiteres Experiment von den LANL, welches im Oktober 2011 in einem Satelliten des amerikanischen *Department of Defense* (DoD) auf dem LEO in Betrieb genommen wurde [137]. Das Vorhaben des Experiments ist die Überprüfung der Verwendung kommerzieller Technologien in kritischen Missionen wie z. B. dem Programm *Space-based Nuclear Detonation Detection* (SNDD) des *Department of Energy* (DoE). Das MRM ist ein Hochfrequenzinstrument, welches ein *Software Defined Radio* (SDF) System in FPGAs realisiert. Das MRM besteht aus zwei Einheiten, den sogenannten *Los Alamos Experimental Units* (LEUs), welche je zwei Virtex-4 FPGAs (XQR4VLX200 und XQR4VSX55) und ein Microsemi RTAX FPGA enthalten.

Das Experiment verwendet die Abschwächungstechnik TMR (siehe Abschnitt 5.6.2) mittels einer eigenentwickelten Software, *BYU-LANL Triple Modular Redundancy* (BL-TMR), und vergleicht Ergebnisse von Strahlungstests mit Echtzeitmessungen des Satellits im LEO für Xilinx Virtex-4 FPGAs. Das Antifuse-FPGA übernimmt wie bei Cibola

die Aufgabe des externen Rekonfigurationskontrollers und führt Readback Scrubbing durch. Die LEU, insbesondere die Scrubbing Komponente wurde im Vorfeld in einem Protonen-Strahlungstest mit 63,3 MeV validiert. Die Auswertung des Strahlungstests zeigt, dass nur 3,25 % der induzierten Fehler sich als MBUs manifestieren. Dieser Wert ist wichtig, da MBUs, wie in Abschnitt 5.6.2 beschrieben, den Einsatz von TMR limitieren. CREME96 hat für den Bestimmungsort eine SEU-Rate von 68 Upsets–89 Upsets pro FPGA und Tag ermittelt. Mit Fehlerinjektionstests wurde das konkrete FPGA-Design auf *sensitive* Bits überprüft (siehe Abschnitt 5.7.3). Die Fehlerinjektion hat hierbei für die jeweiligen Designs einen Wert von 0,4 % (SX55) und 0,1 % (LX200) ergeben. Im Schnitt würde es somit pro FPGA alle 15 bis 25 Tage zu einem beobachtbaren Ausgabefehler kommen. Auf der anderen Seite bedeutet dieses Ergebnis, dass 99 % der Upsets keine beobachtbaren Ausgabefehler zur Folge haben.

Die zwischen Oktober 2011 und Juni 2012 gemessenen Echtzeitdaten von 3 170 SEUs beider LEUs ergeben im Mittel eine SEU-Rate von 14,4 (LX200) bzw. 5,2 (SX55) SEUs pro FPGA und Tag. Der Vergleich mit den, vorab durch CREME96, ermittelten Werten ergibt eine Abweichung um (mindestens) Faktor 4,7. Quinn et al. nehmen an, dass die Dicke der Schutzhülle des Satelliten und der *Duty Cycle* innerhalb der SAA zu der geringeren Rate führen [137]. Die Ergebnisse des Fehlerinjektionstest können ebenfalls an die Echtzeitmessdaten angepasst werden: Alle 75 bis 125 Tage kommt es zu einem beobachtbaren Ausgabefehler. Die Echtzeitmessung hat den SEU-Immunitätswert der Fehlerinjektion (99 %) bestätigt.

Eine Analyse der Upsets zeigt weiterhin, dass 78 % der Upsets in CLBs, 15 % in den Switch-Matrizen des BRAM und 6 % in den IOBs erfolgte. Der Inhalt der BRAM-Blöcke wurde nicht auf Upsets überwacht und fällt somit aus der Bewertung. Bei der Betrachtung, ob die betroffene Speicherzelle die programmierbare Verdrahtung oder die Logik beschreibt, ergibt sich ein Verhältnis von 89 % (± 5 %) für die Verdrahtung zu 6 % (± 1 %) für die Logik.

Ein Novum während der Messung sind fünf detektierte SEFIs (siehe Abschnitt 5.4.1) in der SelectMAP-Schnittstelle [138]. Dies ist bemerkenswert, da selbst die worst-case SEFI-Fehlerrate im Bereich von Jahren bis Jahrzehnten liegt. Nach einer vollständigen Rekonfiguration waren die SEFIs, wie zu erwarten war, behoben. Die Auswertung der Echtzeitdaten hat ergeben, dass 8,42 % der SEEs in MBUs resultieren, wobei sich der Großteil als Zweitbitfehler (6,59 %) manifestiert hat. Dies entspricht annähernd dem Ergebnis der Strahlungstests.

RadSat

Besonders interessant ist der Einsatz von COTS-FPGAs in sogenannten Kleinsatelliten, da in diesen die Kosten einen limitierenden Faktor darstellen. Kleinsatelliten werden nach Gewicht in fünf unterschiedliche Kategorien eingeteilt [185]:

- Minisatelliten: 100 - 500 kg
- Mikrosatelliten: 10 - 100 kg
- Nanosatelliten: 1 - 10 kg

- Picosatelliten: 0,1 - 1 kg
- Femtosatelliten: < 0,1 kg

Zu den Nanosatelliten gehört die im Jahr 1999 von der Stanford University und California Polytechnic State University definierte *CubeSat*-Spezifikation mit einer Abmessung von 11,35 cm x 10 cm x 10 cm und einem Maximalgewicht von 1,33 kg.

Der *RadSat* der Montana State University ist ein Beispiel für ein solches *CubeSat*-Experiment [107]. Neben acht Ballonflügen bis zu einer Höhe von 30 km, wurde bereits ein Raketenflug mit einer Höhe von 120 km durchgeführt. Die NASA unterstützt die Entwicklung von *CubeSat*-Experimenten durch mehrere Initiativen. *RadSat* hat im Rahmen des NASA *SmallSat Technology Partnership* Programms im Jahr 2013 einen weiteren Raketenflug mit einer Höhe von 300 km sowie eine sechsmonatige Integration auf der ISS erhalten. Beide Flüge sind für das Jahr 2016 angesetzt. 2015 wurde *RadSat* darüber hinaus für eine eigenständige Langzeitmission im LEO im Jahr 2017 durch die NASA *CubeSat Launch Initiative* ausgewählt.

Besonders innovativ ist die Verwendung des COTS-FPGAs in dem *RadSat*. In der aktuellen Version wird ein Xilinx Virtex-6 LX75 FPGA durch Verwendung von dynamisch partieller Rekonfiguration in neun Kacheln unterteilt. In jeder Kachel kann ein eigenständiges Prozessorsystem, basierend auf einer MicroBlaze Soft-CPU, aufgebaut werden. Zu jedem Zeitpunkt kann durch Kombination dreier Kacheln ein TMR-Design aufgebaut werden. Zusätzlich werden alle Kacheln mittels Readback Scrubbing überwacht. In dem Fall, dass ein Fehler in einer aktiven Kachel detektiert wird, wird die defekte Kachel durch eine neue, fehlerfreie, inaktive (engl. spare) Kachel ersetzt. Der Vorteil dieses Verfahrens ist, dass im Falle eines SEUs das Gesamtsystem weiterarbeiten kann und nicht auf die Reparatur eines Teilsystems gewartet werden muss.

Die Verwendung von COTS-FPGAs bietet einen enormen Anstieg an Verarbeitungsleistung bei annähernd gleicher Verlustleistung im Vergleich mit strahlungsgehärteten Prozessoren. Zusätzlich sind die Kosten dieser speziellen Prozessoren laut LaMeres et al. um Faktor 40 höher als vergleichbare kommerzielle Versionen. In einem direkten Vergleich erzielt das Virtex-6 FPGA eine Verarbeitungsgeschwindigkeit von 234 MIPS (engl. Million Instructions Per Second) bei 2 W. Im Vergleich mit dem RAD750 Prozessor wird eine um Faktor 2 bessere Energieeffizienz festgestellt. Ein weiterer Vergleich mit anderen Prozessoren, wie dem RAD6000 mit 35 MIPS, ergibt einen Faktor von 7 in der Verarbeitungsgeschwindigkeit. LaMeres et al. beschreiben weiterhin, dass die Gesamtkosten des Systems im Vergleich mit strahlungsgehärteten Komponenten um einen Faktor 100 geringer sind. Für die finale Version wird derzeit die Verwendung eines Xilinx Artix-7 FPGAs mit 4 Kacheln bevorzugt.

Solar Orbiter

Der Solar Orbiter ist eine Raumsonde mit mehreren Fernerkundungsinstrumenten zur Untersuchung der Oberfläche und Atmosphäre der Sonne. Die Sonde soll im Oktober 2018 den Weg auf einen Orbit der Sonne über mehrere Gravitationsmanöver (engl. Swing-By) mit der Erde und Venus starten [56]. Der geplante Orbit hat einen Abstand

von 45 Millionen Kilometer. In [108] wird eines der Erkundungsinstrumente genauer beschrieben, der *Polarimetric and Helioseismic Imager* (PHI). Das Instrument hat die Aufgabe der Vermessung des Magnetfelds in der Photosphäre.

Die Datenverarbeitungseinheit (**Data Processing Unit, DPU**) muss zum einen die Datenrate für die Telemetrie von 3,2 Gbit/s auf 100 Mbit/s reduzieren und zum anderen eine interne Verarbeitungsgeschwindigkeit von 40 GFlops bereitstellen. Die Verarbeitungsgeschwindigkeit von strahlungstoleranten oder -gehärteten CPUs reicht mit maximal 10 bis 100 MFlops bei weitem nicht aus. Die DPU besteht aus einem Systemcontroller (strahlungsgehärteter Aeroflex Gaisler GR712 LEON3-FT Prozessor), einem Konfigurationscontroller (strahlungsgehärtetes Microsemi RTAX2000 FPGA), 4 Tb NAND-Flash-Speicher, 8 Gb SDRAM und zwei rekonfigurierbaren Verarbeitungseinheiten (SRAM-basierte, strahlungstolerante Xilinx Virtex-4QV55 FPGAs).

Die FPGA-Eigenschaft der (vollständigen) Rekonfiguration wird ausgenutzt um zwischen unterschiedlichen Verarbeitungsmodi während der Mission umschalten zu können. In dem ersten Modus, der Datenerfassung (engl. Data Acquisition Mode) kommuniziert das eine FPGA mit dem Bildstabilisierungssystem, das andere FPGA übernimmt die eigentliche Datenerfassung. Dieser Modus wird ca. 9 h aktiv sein, bevor auf den zweiten Modus, der Datenverarbeitung, für ca. 30 Tage umgeschaltet wird. In diesem Modus übernimmt das eine FPGA die Vorverarbeitung der Daten, während das zweite FPGA diese anschließend weiterverarbeitet (Inversion der Strahlungstransportgleichung, engl. Radiative Transfer Equation, RTE). Als Abschwächungstechnik wird Readback Scrubbing über die JTAG-Schnittstelle verwendet. Die Vorteile sind hierbei, dass nur vier Ein- bzw. Ausgänge benötigt werden und dass eine vollständig externe Lösung in strahlungstoleranten Komponenten realisiert werden kann. Der wesentliche Nachteil ist die im Vergleich sehr langsame Verarbeitungsgeschwindigkeit von 66 Mbit/s. Die Verwendung der externen SelectMAP- oder internen ICAP-Schnittstelle erlaubt eine um Faktor 48 höhere Datenrate (3,2 Gbit/s, siehe Abschnitt 3.3.2).

Heinrich Hertz Satellit

Der Heinrich Hertz Satellit ist ein, von dem *Deutschen Zentrum für Luft- und Raumfahrt* (DLR) finanzierter, Kommunikationssatellit. Geplanter Missionsstart ist das Jahr 2020, Missionsstandort ist GEO und die geplante Laufzeit 15 Jahre [69]. Die Kapazitäten der Datenverarbeitung sollen zu jeweils 50 % für wissenschaftliche und militärische Zwecke verwendet werden. Der *Fraunhofer On-Board Processor* (FOBP) ist ein Teil des Satelliten und implementiert regenerative Transponder durch zwei rekonfigurierbare, strahlungstolerante Xilinx Virtex-5 QVFX130 FPGAs. In [150] wird der Systemaufbau und die Systemkonfiguration des FOBPs beschrieben. Das System teilt sich in zwei identische, FPGA-basierte Teilsysteme zur digitalen Signalverarbeitung (DSP1 und DSP2) auf. Jedes DSP verfügt neben den erwähnten FPGAs über ADCs und DACs sowie verschiedene Speicherblöcke in unterschiedlichen Technologien. In den DSPs wird auf einen externen Rekonfigurationscontroller verzichtet und die initiale Rekonfiguration durch den FPGA selbst vorgenommen. Die initialen Konfigurationsdateien sind in einem

magnetoelektronischen Speicher (MRAM) abgelegt. Zur Systeminitialisierung werden beide FPGA mit einer identischen Konfiguration, einem DPR-System mit einem LEON3-FT als Kontrolleinheit, beschrieben. Anschließend erfolgt eine Aushandlung, welche zu einer Teilung in ein Master- und ein Slave-FPGA resultiert. Das Master-FPGA ist anschließend für die (interne und externe) Rekonfiguration, die Fehlerbehandlung (Fault Detection, Isolation and Recovery, FDIR) und die Konfiguration der Telemetrieinheit (realisiert durch das Slave-FPGA) zuständig.

Ein Ziel des Projekts ist die adaptive Abschwächung, das heißt, dass Komponenten-Redundanz als Abschwächungsverfahren nur dann verwendet werden soll, wenn es wirklich notwendig ist. Laut Glein kann hierdurch in 90 % der Anwendungszeit der Datendurchsatz des FOBPs, im Vergleich zu einem reinen TMR-Design, verdreifacht werden. Die SEU-Raten wurden im Vorfeld mit CREME96 für das Virtex-5-FPGA im GEO mit einer 4,5 mm dicken Aluminiumabschirmung berechnet. Für den Konfigurationsspeicher variiert die SEU-Rate pro Tag und Gerät je nach Sonnenaktivität zwischen $3,0 \times 10^{-4}$ und 1,19. Die Rate des BRAMs ist mit $1,25$ und $6,74 \times 10^3$ um mindestens drei Größenordnungen höher. Die Idee von Glein ist daher diese Anfälligkeit des BRAMs auszunutzen und einen BRAM-basierten Sensor innerhalb des FPGAs aufzubauen. Als Beispiel wird ein, aus 64 von 298 verfügbaren BRAM-Blöcken aufgebauter Sensor vorgestellt, der eine Auflösung von einer Minute (im worst-case Szenario) bereitstellt. Weiterhin werden Schwellwerte für die Umschaltung von drei Betriebsmodi genannt. Diese bestimmen ob der Modus 1 *noR*, also keine Redundanz, Modus 2 *DMR* mit doppelter Redundanz oder Modus *TMR* mit dreifacher Redundanz aktiviert werden soll. Durch die Verwendung von dynamisch partieller Rekonfiguration können, dem aktuellen Modus entsprechend, vorgefertigte Module auf das FPGA konfiguriert werden. Weiterhin ist die Verwendung von neuen Konfigurationsdateien, z. B. für die Implementierung neuer Kommunikationsprotokolle, festes Ziel der langjährigen Mission. In [163] werden konkrete Zeiten für dieses Szenario genannt. Das aktuelle System ermöglicht eine Bandbreite von 2 MHz, bei einer Verfügbarkeit von 99,9 % und einem (TCP-basierten) Datendurchsatz von etwa 0,85 Mbit/s. Die Übertragung von vollständigen (6 MB) und typischen partiellen (3,5 MB) Konfigurationsdateien resultiert zu 57 s bzw. 33 s. Die Zeit der anschließenden (dynamisch partiellen) Rekonfiguration variiert je nach verwendeter interner oder externer Schnittstelle, liegt jedoch immer unterhalb einer Sekunde.

5.10 Zusammenfassung

In diesem Kapitel wurde die Verwendung von FPGAs in Systemen unter Strahlungseinfluss beschrieben. Zunächst wurde auf die physikalischen Eigenschaften und Effekte sowie den Ursprung von Strahlung eingegangen. Anschließend erfolgte eine Kategorisierung der Effekte in Single-Event Effects (SEU, MCU und MBU, SET, SEFI, SEL sowie SEGR) und kumulative Effekte (TID und DD).

Die Auswirkungen dieser Effekte auf die FPGA-Technologien SRAM, Flash und Antifuse wurde darauffolgend analysiert und die jeweiligen Anfälligkeiten festgehalten. Die Antifuse-Technologie bietet hierbei die höchste Robustheit. Flash-basierte FPGAs bieten eine hohe Robustheit gegeben über SEU-, MCU- und MBU-Effekte, sind jedoch anfällig gegenüber SET- und TID-Effekten. Die, in dieser Dissertation insbesondere betrachteten, SRAM-basierten FPGAs weisen eine deutliche Anfälligkeit gegenüber SEU-, MCU- und MBU-Effekten auf. Die Anfälligkeit gegenüber diesen Strahlungseffekten ist insbesondere in der stetigen Miniaturisierung der Strukturgröße begründet. Durch die gleichzeitige Skalierung der Versorgungsspannungen sowie der steigenden Transistoranzahl pro Bauteil steigt jedoch auch die Anfälligkeit gegenüber TID.

Zur Abschwächung der Strahlungseffekte wurden Verfahren mit Hilfe zweier Fehlermodelle der NASA und ESA vorgestellt und kategorisiert. Insbesondere wurde auf Redundanzverfahren, Verfahren zur Fehlererkennung und -korrektur sowie Scrubbing-Verfahren eingegangen. Die Anwendung und Analyse dieser Verfahren erfolgte detailliert für SRAM-basierten FPGA-Technologie sowie kurz für die Antifuse- und Flash-basierten FPGA-Technologien. Als zentrales Ergebnis wurde festgehalten, dass in vielen Fällen der Einsatz von Kombinationen verschiedener Verfahren zur Erhöhung der Fehlertoleranz der Einzelkomponenten sinnvoll ist.

Für den Einsatz in Umgebungen mit hoher Strahlungsdosis wurden aktuelle strahlungstolerante und -gehärtete FPGAs vorgestellt. Diese FPGAs sind durch spezielle Fertigungsschritte oder Designänderungen am Bauteil selbst (z. B. Layout-Änderungen) robuster gegenüber Strahlung als kommerzielle Standard-FPGAs (COTS-FPGAs).

Abschließend wurden Experimente und Projekte in Anwendungsbereichen mit (erhöhter) Strahlung vorgestellt, in denen FPGAs eingesetzt werden. Hier wurde zunächst auf die Entwicklung der Strahlungstoleranz aktueller Xilinx COTS-FPGAs gegenüber terrestrischer Strahlung (Experiment Rosetta) eingegangen. Zusätzlich wurde der Einfluss der Parameter Anzahl und Typ der FPGAs sowie des Standorts auf die Anfälligkeit des Gesamtsystems analysiert. Als konkretes terrestrisches Anwendungsgebiet wurden Hochenergiephysikexperimente des Teilchenbeschleunigers (*Large Hadron Collider*) im europäischen Kernforschungszentrum CERN vorgestellt. Die Strahlungsdosis in den Experimenten übersteigt dabei in einigen Experimenten die Strahlungsdosis in kosmischen Bereichen. Viele der Hochenergiephysikexperimente, aber auch viele kosmische Missionen, benötigen die Verarbeitungsgeschwindigkeit von FPGAs um die anfallenden Daten verarbeiten zu können. Strahlungsgehärtete CPUs sind hierbei um zwei bis drei Größenordnungen langsamer. Die geringe Kommunikationsbandbreite in kosmischen Missionen forciert den Einsatz von FPGAs, da Datenkomprimierungsalgorithmen dann deutlich schneller ausgeführt werden können.

Die erhöhte Systemflexibilität ist ein weiterer Grund für den Einsatz von FPGAs. In vielen der genannten kosmischen Missionen konnten nachträglich Systemänderungen durchgeführt werden (z. B. im *Cibola Flight Experiment* zweimal pro Monat). Die verfügbare Hardwareflexibilität von FPGAs wird hierbei immer weiter ausgenutzt. So werden beispielsweise die FPGAs in dem *Solar Orbiter* im Zeitmultiplex-Betrieb verwendet um

Daten zunächst aufnehmen und anschließend verarbeiten zu können. Während bei dem *Solar Orbiter* die FPGAs vollständig (von extern) rekonfiguriert werden, können die FPGAs des *Heinrich Hertz* Satelliten durch dynamisch partielle Rekonfiguration an die aktuelle Strahlungssituation angepasst werden. Durch einen selbstentwickelten, BRAM-basierten Strahlungssensor/-detektor können Verarbeitungsmodul mit entsprechenden Abschwächungsverfahren im System konfiguriert und somit ein optimaler Betriebsmodus bezüglich der Leistungsfähigkeit und der Fehlertoleranz eingestellt werden. In *CubeSat*-Experimenten werden oftmals COTS-Komponenten eingesetzt. Die Verwendung von COTS-FPGAs resultiert in einem Anstieg der Verarbeitungsleistung bei gleichzeitiger Reduzierung der Gesamtkosten des Systems. In dem vorgestellten *Rad-Sat*-Experiment ergibt sich durch die Verwendung eines Virtex-6-FPGAs eine Erhöhung der Verarbeitungsleistung um Faktor 7, während die Produktionskosten im Vergleich zu strahlungsgehärteten Komponenten um zwei Größenordnungen sinken. Hierbei realisiert das FPGA ein aus mehreren Kacheln aufgebautes DPR-System. Durch Kombination dreier Kacheln kann ein TMR-Design mit drei separaten Soft-CPU's aufgebaut werden.

Wie dargestellt, steigt das Interesse von DPR als wichtige Systemeigenschaft kontinuierlich. Mit steigender Anzahl an Verarbeitungsmodulen und -algorithmen wird bei der Verwendung des Standardansatzes des FPGA-Herstellers Xilinx die Wartbarkeit des Systems deutlich eingeschränkt und die verfügbare Flexibilität eines FPGAs nicht ausgeschöpft (siehe Abschnitt 3.5.8). Die Erstellung eines flexiblen, wartbaren und fehlertoleranten DPR-Systems mit der Unterstützung von Modulplatzierung ist daher wünschenswert und wird im folgenden Kapitel beschrieben.

6 Evaluation der Ziele

Der Einsatz rekonfigurierbarer Hardware in Weltraummissionen wächst, wie in Abschnitt 5.9.2 dokumentiert, stetig. Insbesondere für die Verarbeitung der ermittelten Daten reicht die Leistungsfähigkeit von Prozessoren oftmals nicht aus. Aktuelle FPGAs hingegen bieten eine hohe Leistungsfähigkeit bei der Datenverarbeitungsschritte und ermöglichen durch dynamisch partielle Rekonfiguration den Aufbau eines flexiblen, energieeffizienten Systems, welches sich an die wechselhaften, rauen Bedingungen im Weltraum anpassen kann. Müssen einzelne Datenverarbeitungen nicht dauerhaft ausgeführt werden, kann durch einen Zeitmultiplex-Betrieb ein weiterer Vorteil der DPR genutzt werden.

Im Rahmen des, von der Europäischen Weltraumorganisation (engl. European Space Agency, ESA) geförderten, Projekts *FPGA Based Generic Module & Dynamic Reconfigurator* wurde ein Demonstrator für dynamisch rekonfigurierbare Verarbeitungsmodule (*Dynamically Reconfigurable Processing Module demonstrator*, DRPM) entwickelt. Die Systemarchitektur dieses Demonstrators erlaubt nachträgliche Änderungen am System oder eine Adaptierung des Systems, z. B. durch einen dynamischen Austausch von Datenverarbeitungsalgorithmen. Diese Funktionalität wird durch Rekonfigurations- und Systemkontrolleneinheiten realisiert. Zusätzlich werden Schnittstellen und Protokolle aus dem Bereich der Avionik (Luft- und Raumfahrt) in das System integriert. Beispiele hierfür sind *SpaceWire* (mit *Remote Access Control Protocol*, RMAP), *SpaceFibre*, *Wizard-Link* oder *MIL-STD-1553B*. Der Demonstrator stellt aufgrund der Systemarchitektur eine ideale Evaluationsplattform für die aufgeführten Ziele dieser Dissertation dar.

Im ersten Abschnitt wird der DRPM Demonstrator beschrieben. Dabei wird zunächst auf die Hardware, das Rapid Prototyping System RAPTOR und die beiden Erweiterungsmodule DB-V4 und DB-SPACE, eingegangen. Im Anschluss wird die skalierbare DRPM-Systemarchitektur beschrieben. Hierbei werden die Realisierungen des Verarbeitungs- und des Kommunikationsmoduls und verschiedene Systemvarianten des DRPMs vorgestellt. Im zweiten Abschnitt werden die Kernkomponenten zur Realisierung des INDRA 2.0 DPR-Systems beschrieben. Hierzu zählen ein externer und ein interner Rekonfigurationskontroller, ein Speicherkontroller sowie die Komponenten zur Realisierung der Kommunikation im DPR-System. Für die homogene DPR-Kommunikationsinfrastruktur werden ein- und zweidimensionale Varianten mit variierender Kachelanzahl vorgestellt, welche durch DHHarMa erstellt wurden. Im anschließenden Kapitel werden drei verschiedene Anwendungstypen – *Finite Impulse Response* (FIR), *Fast Fourier Transform* (FFT) und Bildfilter – für die dynamisch rekonfigurierbaren Verarbeitungsmodule des DPR-Systems vorgestellt. Im letzten Abschnitt wird auf das dritte

Ziel dieser Dissertation, die entwickelten Abschwächungsverfahren gegenüber SEEs im Konfigurationsspeicher und permanente Fehlern in den Verdrahtungsressourcen eines FPGAs, eingegangen.

6.1 Demonstrator für dynamisch rekonfigurierbare Verarbeitungsmodule

An der Entwicklung des Demonstrators für dynamisch rekonfigurierbare Verarbeitungsmodule waren vier Partner aus der Wirtschaft und Wissenschaft beteiligt: Die Unternehmen TWT GmbH Science & Innovation und Swiss Space Technology sowie das Politecnico di Torino und die Universität Bielefeld. Der Demonstrator wird ausführlich in dem Handbuch des DRPMs beschrieben [45]. In Abbildung 6.1 ist eine allgemeine Systemübersicht des DRPMs dargestellt. Durch die genannten Aufgabenbereiche ergibt sich die Aufteilung in Kommunikations- und Verarbeitungsmodule. Die Skalierbarkeit des Demonstrators ist durch die gestrichelten Umrandungen der beiden Modultypen angedeutet.

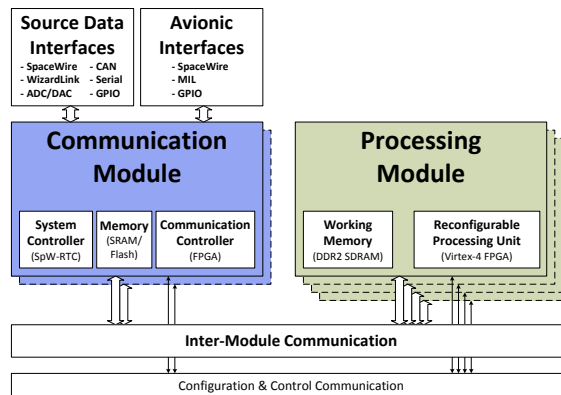


Abbildung 6.1: Systemübersicht des DRPM-Systems [77, S. 1]

Der Demonstrator basiert auf dem modularen Rapid Prototyping System RAPTOR [135], welches bis zu sechs physikalische Module (engl. DaughterBoard, DB) integrieren kann. Die Datenverarbeitung erfolgt auf FPGA-basierten Verarbeitungsmodule. In diesem Projekt wird ein Verarbeitungsmodule mit einem Xilinx Virtex-4 FX100 (DB-V4) und einem DDR2-SDRAM verwendet. Zur Integration der Kommunikationsschnittstellen aus dem Bereich der Avionik wurde ein neues Kommunikationsmodul (DB-SPACE) entwickelt. Die Systemkontrollkomponente besteht aus einem strahlungsgehärtetem ASSP (Atmel AT7913E) mit einem LEON2-FT-Prozessor. Die FPGAs auf beiden Modultypen sind COTS-FPGAs, um die Kosten des Prototypen des DRPMs gering zu halten. Für

die COTS-FPGAs gibt es äquivalente strahlungstolerante oder -gehärtete Komponenten (siehe Abschnitt 5.8), sodass die Entwicklung eines zukünftigen, für den Weltraum qualifizierten, DRPMs möglich ist.

Mögliche Anwendungsszenarien für das DRPM sind in Anwendungen mit hohen Anforderungen an die Datenverarbeitung zu finden. Konkrete Beispiele hierfür sind Erd- oder Wetterbeobachtungssatelliten sowie Kommunikationssatelliten [59, S. 13 f.].

6.1.1 Übersicht über die DRPM-Hardware

Das Ziel des DRPM Demonstrators ist die Entwicklung einer skalierbaren Plattform, in der die Anzahl der rekonfigurierbaren Verarbeitungskomponenten und/oder Kommunikationsschnittstellen variiert werden kann und somit verschiedene Einsatzszenarien unterstützt werden können. Dies wird primär durch die Verwendung des RAPTOR-Systems als Basis und zusätzliche Erweiterungsmodule ermöglicht.

RAPTOR - Rapid Prototyping Platform

Die Rapid Prototyping Platform RAPTOR ermöglicht den Aufbau modularer Systeme [135]. In dem DRPM wird die zweite Generation RAPTOR-X64 verwendet, welches eine Bestückung von bis zu sechs Modulen ermöglicht [45, S. 19 f.]. Die Architektur des RAPTOR-X64 ist in Abbildung 6.2 dargestellt. Dieses Basissystem besteht aus Kommunikations- (*Local-Bus*, *Broadcast-Bus*) und Managementinfrastrukturen (z. B. *System Monitor*, *CTRL+Config Logic*), an die sich gesteckte (Erweiterungs-) Module anschließen. Dabei können neue FPGA-Familien durch die Implementierung eines neuen Moduls (DBs) in die RAPTOR-Plattform integriert werden. Eine Hardware Emulation, die frühzeitige Überprüfung einer Hardware-Komponente, kann über FPGA-basierte Verarbeitungsmodule erfolgen.

Das in dem DRPM verwendete RAPTOR-X64-Basisboard kann über PCI-X oder USB-2.0 mit einem Host-PC kommunizieren. Beide Schnittstellen sind direkt an den, in dem Basisboard eingebetteten, 32 bit breiten Local-Bus angeschlossen. Diese Kommunikationsschnittstelle ist insbesondere für Debugging- und Monitoring-Aufgaben vorteilhaft. Mit dem Broadcast-Bus existiert ein weiterer, RAPTOR-interner Kommunikationsbus für die Module. Benachbarte Module können ferner über direkte Verbindungsleitungen mit einer maximalen Bandbreite von mehr als 20 Gbit/s miteinander kommunizieren. Die Konfigurations- und Anwendungsdaten können entweder über ein direkt gekoppelten Host-PC oder in einer *Compact Flash* Karte abgelegt werden. Im letztgenannten Fall kann das Basisboard daher auch im eigenständigen (engl. *stand-alone*) Betrieb verwendet werden.

DB-V4

Das *DB-V4* ist ein FPGA-basiertes Datenverarbeitungsmodul für das RAPTOR-X64 [45, S. 22 f.]. Die Kernkomponenten des Moduls sind ein Xilinx Virtex-4 FX100 FPGA und ein DDR2-SODIMM-Modul mit bis zu 4 GB. Das FPGA integriert zwei eingebettete

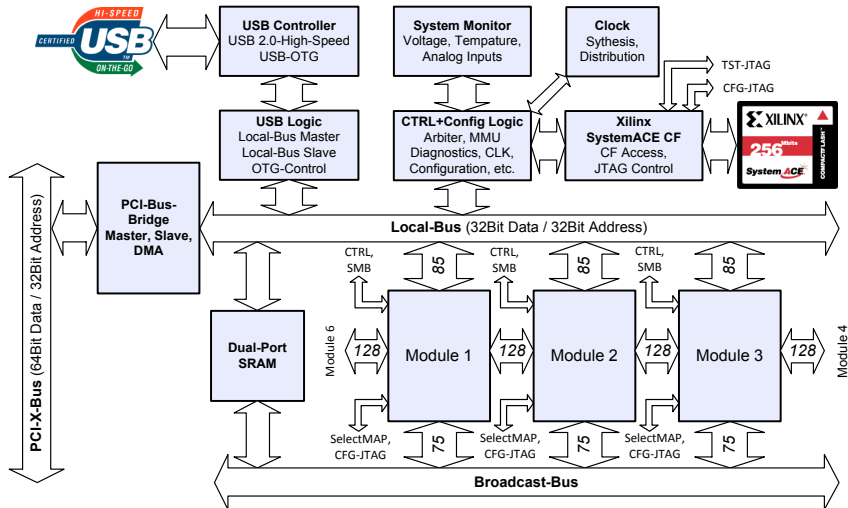


Abbildung 6.2: Übersicht der RAPTOR-X64-Architektur [45, S. 20]

PowerPC-Prozessoren (PPC405) mit bis zu 450 MHz. Die Abbildung 6.3 zeigt das Blockdiagramm des DB-V4s.

Die Kommunikation mit externen Komponenten kann über 20 serielle MGTs mit jeweils bis zu 6,5 Gbit/s (Voll duplex) erfolgen. Insgesamt ist eine Datenübertragungsrate von bis zu 32,5 Gbit/s über vier kupferbasierte Verbindungen möglich.

DB-SPACE

Das *DB-SPACE* bildet das Kommunikationsmodul. Die Hauptsteuerkomponente ist ein strahlungsgehärteter *SpaceWire Remote Terminal Controller* von Atmel (*SpaceWire-RTC*, AT7913E), welcher als Systemkontroller die wesentlichen Steuerungsaufgaben übernimmt. Der *SpaceWire-RTC* integriert einen LEON2-FT Sparc V8 Prozessor (FT steht für *fault tolerant*), welcher aus einer *Integer Unit* (IU) und einer *MEIKO Floating Point Unit* (FPU) besteht und an mehrere Kommunikationsschnittstellen (z. B. *SpaceWire* und *CAN*) angeschlossen ist. Zusätzlich existiert eine Speicherschnittstelle zur Verbindung eines externen Speichers und eine FIFO. Abbildung 6.4 stellt das Blockdiagramm des *DB-SPACE* dar.

Zwei Spartan-6 FPGAs (LX100 und LX150) erweitern die Kommunikationsinfrastruktur des Moduls bzw. des Gesamtsystems. Durch die Verwendung von FPGAs kann ein flexibles Kommunikationssystem auf dem *DB-SPACE* realisiert und verschiedene Varianten/Prototypen des DRPMs erstellt werden. Das *Int-Comm-FPGA* verbindet das Modul mit der RAPTOR-X64-Kommunikationsinfrastruktur (Local-Bus) und den Modulverbindungen (LVDS-basierte Direktverbinder), um mit den benachbarten Modulen

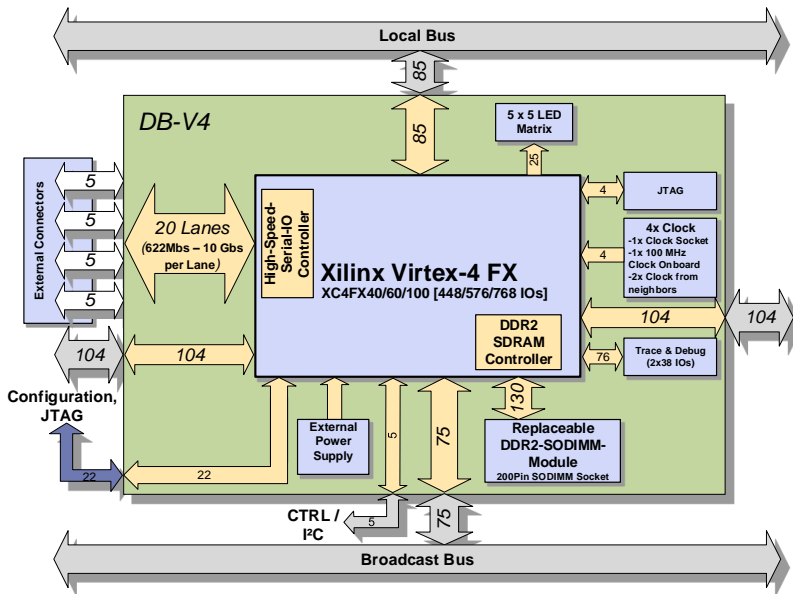


Abbildung 6.3: Blockdiagramm des DB-V4s [45, S. 22]

kommunizieren zu können. Das *Ext-Comm-FPGA* integriert weitere Kommunikationsschnittstellen aus dem Bereich der Avionik in das eingebettete System.

Als externer Rekonfigurationskontrolller für alle FPGAs in dem System wird ein Xilinx CoolRunner-II CPLD (XC2C384) verwendet. Dieser ist direkt an einen 1 Gbit großen Flash-Speicher von Micron (PC28F00AP33EFA) angeschlossen, in dem die initialen Konfigurationsdateien der FPGAs gespeichert sind. Die Konfiguration der FPGAs auf den benachbarten Verarbeitungsmodulen erfolgt über dedizierte Verbindungen der Modulverbindungen. Details zu diesem externen Rekonfigurationskontrolller werden in Abschnitt 6.2.1 beschrieben. Alle aufgeführten Kommunikationsschnittstellen für den Bereich der Avionik sowie die GPIO- und DEBUG-Schnittstellen sind an das *DB-SPACE Front Panel* angeschlossen (siehe Abbildung 6.5).

DRPM-Hardware-Setup

Durch die Modularität des RAPTOR-Basisboards ergeben sich verschiedene Konfigurationsmöglichkeiten für das DRPM. Als Basiskonfiguration wird ein Setup mit einem Kommunikationsmodul, basierend auf dem *DB-SPACE*, und zwei, auf dem *DB-V4* basierenden, Verarbeitungsmodulen, gewählt. Der Aufbau der Hardware dieser Konfiguration ist in Abbildung 6.5 visualisiert. Weitere Systemkonfigurationen werden in Abschnitt 6.1.6 vorgestellt.

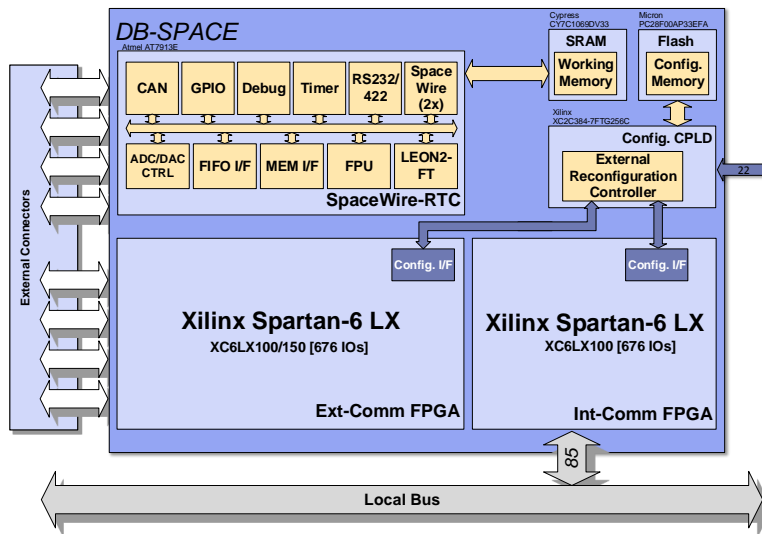


Abbildung 6.4: Blockdiagramm des DB-SPACE [45, S. 24]

6.1.2 Systemarchitektur des DRPMs

Die Systemarchitektur der Basiskonfiguration des DRPMs mit zwei Verarbeitungsmodulen und einem Kommunikationsmodul ist in Abbildung 6.6 dargestellt. Der Entwurf aller FPGA-Teilsysteme erfolgt mit Hilfe des Xilinx *Embedded Development Kits* (EDK). Diese Software ermöglichen den Aufbau eines prozessorbasierten *System-on-a-Chip* (SoC). Bei diesem gemeinsamen Entwurf von Hardware und Software (Hardware/Software Codesign), stellt die Software *Xilinx Platform Studio* (XPS) die Entwicklungsumgebung für die Hardware zur Verfügung. Dabei wird ein IP-Core-basierter Ansatz durch eine Bibliothek von (primär HDL-basierten) IP-Cores für Peripherie und Prozessoren unterstützt. Eigene Komponenten können in Form eines IP-Cores die Bibliothek erweitern. Das Eclipse-basierte *Xilinx Software Development Kit* (XSDK) bildet die Entwicklungsumgebung für die Software und besteht aus einer *GNU Compiler Collection* (GCC) für den Hard-IP-Core-Prozessor PowerPC® und den Soft-IP-Core-Prozessor MicroBlaze. Der Compiler unterstützt die Programmiersprachen C und C++ und übersetzt den Programmcode in das *Executable and Linking Format* (ELF). Die einzelnen Teilkomponenten bzw. IP-Cores der beiden Modultypen werden in den folgenden Abschnitten beschrieben.

6.1 Demonstrator für dynamisch rekonfigurierbare Verarbeitungsmodule

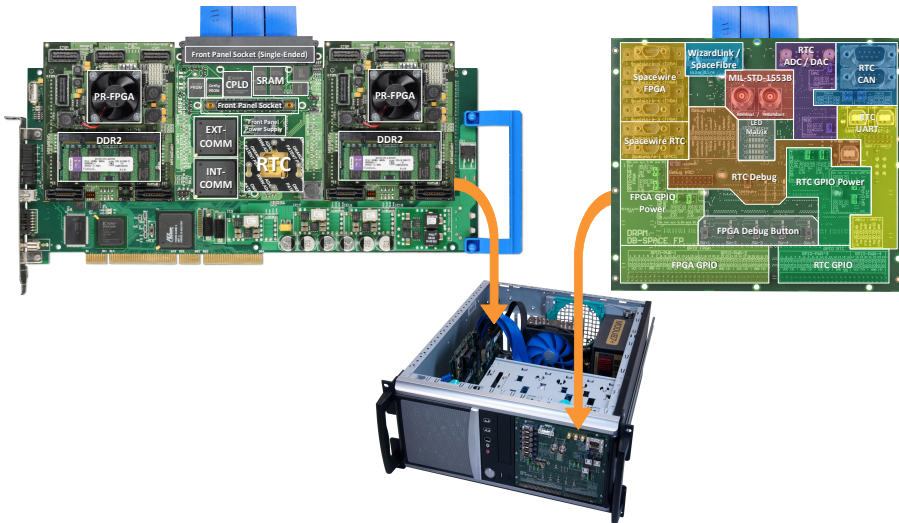


Abbildung 6.5: Basiskonfiguration des DRPMs mit zwei Verarbeitungsmodulen und einem Kommunikationsmodul

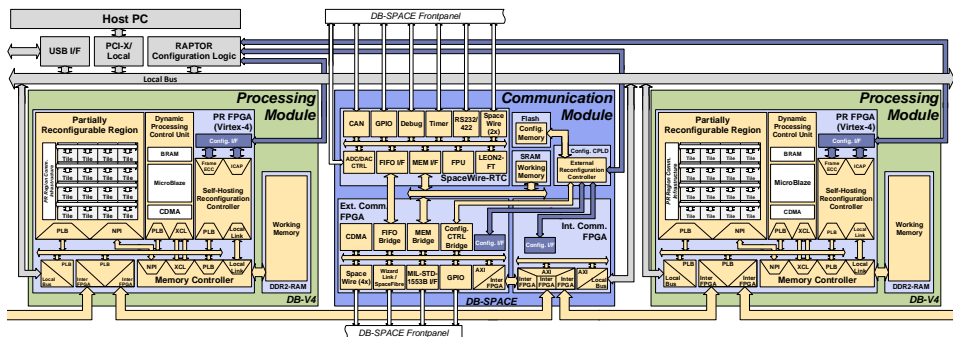


Abbildung 6.6: Systemarchitektur der Basiskonfiguration des DRPMs mit zwei Verarbeitungsmodulen und einem Kommunikationsmodul (basierend auf [77, S. 3])

6.1.3 Verarbeitungsmodul PR-FPGA

Das Verarbeitungsmodul besteht primär aus zwei Komponenten: Einem Xilinx Virtex-4 FX100 FPGA, welches die benötigten rekonfigurierbaren Ressourcen bereitstellt, und einem 4 GB großen DDR2-SODIMM (engl. Small Outline Dual In-line Memory Module). Die FPGA-Ressourcen teilen sich zur Realisierung eines DPR-Systems in statische und

dynamische Komponenten auf. Die Abbildung 6.7 stellt die Aufteilung des eingebetteten SoCs in Hauptkomponenten detaillierter vor.

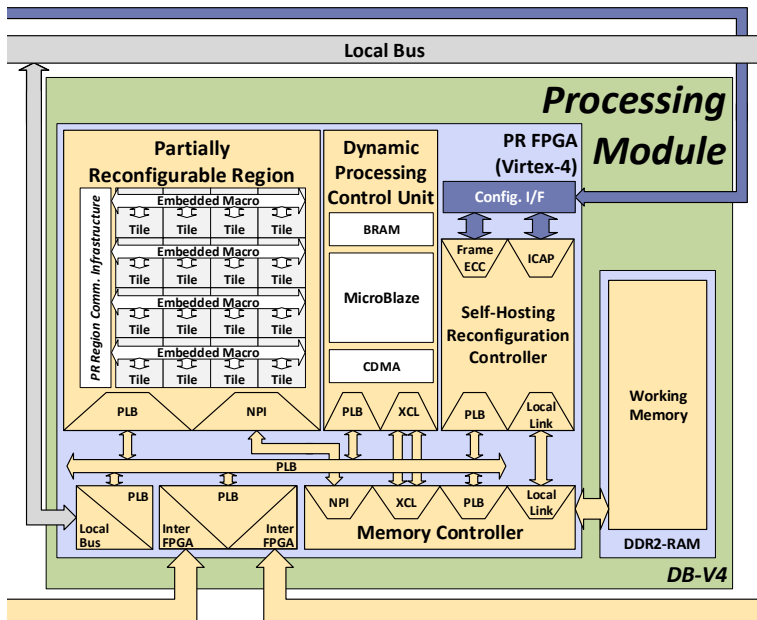


Abbildung 6.7: Übersicht der Hauptkomponenten des Verarbeitungsmoduls (basierend auf [77, S. 4])

Statische Komponenten

Die sogenannte *Dynamic Processing Control Unit* (DPCU) bildet die Kontrolleinheit des Verarbeitungsmoduls und besteht aus einem MicroBlaze-Prozessor, einer *Central Direct Memory Access* (CDMA) Einheit zur performanten, CPU-unabhängigen, internen oder externen Datenübertragung, einem Interruptkontroller und lokalem Blockspeicher (BRAM) für den Bootloader. Der MicroBlaze-Prozessor konfiguriert die statischen Komponenten zur Systeminitialisierung und Laufzeit, steuert den Datenfluss innerhalb des Verarbeitungsmoduls und verwaltet und wählt Positionen für die PR-Module.

FPGA-intern sind die statischen Komponenten über einen IBM *Processor Local Bus* (Version 4.6) verbunden. Die Datenbreite beträgt dabei 64 bit, die Addressbreite 32 bit. Die Kommunikation mit anderen Kommunikations- oder Verarbeitungsmodulen erfolgt über die *Inter FPGA Bridge*. Eine Anbindung an den Local-Bus des RAPTOR-X64 wird

ebenfalls über eine dedizierte Brücke hergestellt. Beide Kommunikationsbrücken sind sowohl als Master als auch Slave an den PLB angeschlossen.

Der DDR2-Speicher wird über einen *Multi-Port Memory Controller* (MPMC) in das SoC integriert und stellt den Speicher über verschiedene Ports zur Verfügung. Dies hat zum Vorteil, dass der PLB entlastet werden kann und einzelne Komponenten eine performante, auf sie zugeschnittene, Anbindung erhalten. Der Speicherkontroller basiert auf dem MPMC von Xilinx und wurde um zwei Funktionen erweitert: Fehlerinjektion und ECC-Fehlerstatistik. Weitere Details zu dem Speicherkontroller und den Schnittstellen folgen in Abschnitt 6.2.3.

Der *Self-Hosting Reconfiguration Controller* (SHRC) implementiert einen internen Rekonfigurationskontroller, welcher mehrere Funktionen realisiert. Zum einen ermöglicht der SHRC die dynamisch partielle Rekonfiguration auf dem FPGA über die FPGA-interne Konfigurationsschnittstelle ICAP (siehe Abschnitt 3.3.2). Zum anderen kann zur Erhöhung der Fehlertoleranz zwischen zwei verschiedenen Scrubbing Verfahren gewählt werden (siehe Abschnitt 5.7.3). Beim Blind Scrubbing wird das FPGA in regelmäßigen Abständen mit der original Konfigurationsdatei überschrieben, welches einer reinen dynamisch partiellen Rekonfiguration entspricht. Weiterhin, integriert der SHRC den Hard-IP-Block *Frame ECC*. Diese Komponente dient der Realisierung eines Readback Scrubbings zur Detektion von, durch SEEs induzierten, Ein- und Zweibitfehlern im Konfigurationsspeicher (siehe Abschnitt 5.7.3). Durch das Readback Scrubbing kann auch ein permanenter Fehler erkannt und das System entsprechend umkonfiguriert werden. Um die Konfigurationszeit möglichst gering zu halten, existiert eine direkte Anbindung zu dem DDR2-Speicher über dedizierte *Local Link* Verbindungen zum MPMC. Der DDR2-Speicher ist zur Laufzeit der primäre Speicherort der (partiellen) Konfigurationsdateien. Diese performante Anbindung ist, wie in Abschnitt 3.1 beschrieben, Grundvoraussetzung für die Realisierung eines energieeffizienten DPR-Systems. Details zu dieser Komponente werden in Abschnitt 6.2.2 beschrieben.

Dynamische Komponenten

Die dynamischen Komponenten (PR-Module) können in dem partiell rekonfigurierbaren Bereich (PR-Region) in rekonfigurierbare Kacheln (PR-Tiles) platziert werden. Ein DPR-System kann aus verschiedenen PR-Tile-Typen bestehen, die sich in den verfügbaren FPGA-Ressourcen unterscheiden. PR-Tiles des gleichen Typs stellen identische Ressourcen bereit.

Die PR-Tiles sind über ein eingebettetes Kommunikationsmakro mit dem statischen Bereich und den anderen Modulen verbunden (siehe Abschnitt 3.4). Das Kommunikationsmakro wird, wie in Abschnitt 4.6 beschrieben, im Vorfeld automatisch generiert und implementiert zusätzliche Funktionen, wie z. B. eine PR-Tile-Isolation oder ein initiales PR-Tile-Reset nach einer Rekonfiguration. Die bereitgestellte Homogenität der PR-Tiles und des Kommunikationsmakros erlaubt eine Umplatzierung von Modulen. Weitere Details folgen in Abschnitt 6.2.4. Die PR-Region ist über eine PLB-Bridge an

das MicroBlaze-System und über dedizierte *Native Port Interface* (NPI) Kanäle an den MPMC angeschlossen.

6.1.4 Kommunikationsmodul DB-SPACE

Das Kommunikationsmodul DB-SPACE integriert, wie in Abbildung 6.4 dargestellt, bereits durch den SpaceWire-RTC eine Vielzahl von Kommunikationsschnittstellen in das eingebettete System. Abbildung 6.8 detailliert das Gesamtsystem des Kommunikationsmoduls.

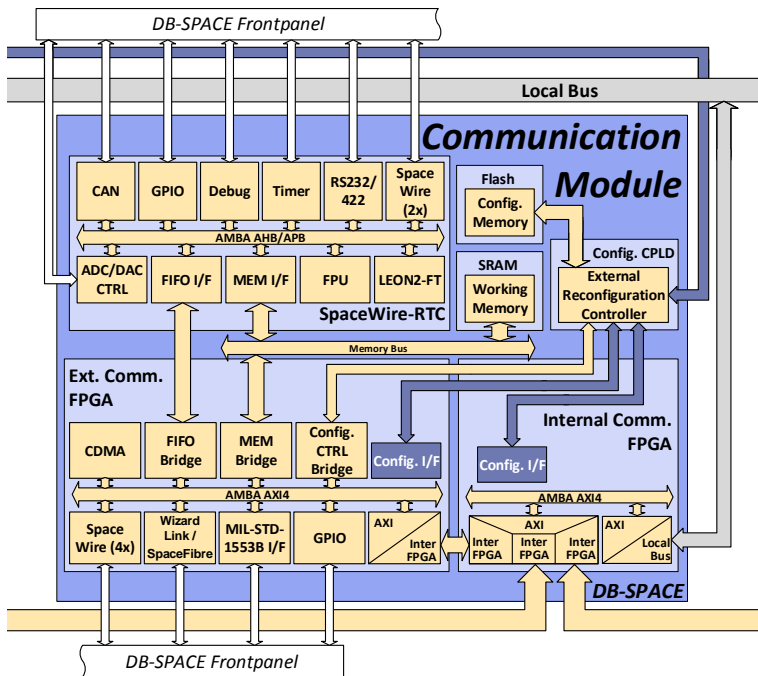


Abbildung 6.8: Übersicht der Hauptkomponenten des Kommunikationsmoduls (basierend auf [77, S. 4])

Das Ext-Comm-FPGA erweitert das System um weitere vier SpaceWire-, eine SpaceFibre- oder WizardLink- sowie eine MIL-STD-1553B-Schnittstelle(n). Für Debugging-Aufgaben werden weitere 32 GPIOs über das Frontpanel angebunden. Der Transport größerer Datenmengen wird, analog zum PR-FPGA, über eine CDMA-Einheit gesteuert. Über die *Config. CTRL Bridge* kann mit dem externen Rekonfigurationskontrolller kommuniziert werden. Dies ist zum Beispiel bei der Behebung eines Zweibitfehlers

in einem PR-FPGA durch eine vollständige Rekonfiguration nötig. Der SRAM-basierte Arbeitsspeicher wird vom Ext-Comm-FPGA und dem SpaceWire-RTC eingebunden. Eine direkte Kommunikation zwischen dem Ext-Comm-FPGA und dem SpaceWire-RTC wird über die *FIFO-Bridge* hergestellt. Die Kommunikationskette aller DB-SPACE-Subsysteme wird über eine Inter FPGA Bridge zu dem Int-Comm-FPGA geschlossen.

Das Int-Comm-FPGA realisiert die Kommunikation mit den benachbarten Verarbeitungsmodulen über eine Inter FPGA Bridge (über die physikalischen Links-Rechts-Verbinder) und verbindet das Modul über eine Local-Bus Bridge mit der RAPTOR-X64-Kommunikationsinfrastruktur.

Die interne Kommunikation aller Einzelsysteme erfolgt über flexible ARM *Advanced Microcontroller Bus Architecture* (AMBA) Busse. Auf dem SpaceWire-RTC wird die Kommunikationsinfrastruktur durch einen *Advanced High-performance Bus* (AHB) und einen *Advanced Peripheral Bus* (APB) realisiert. Xilinx EDK unterstützt AMBA *Advanced eXtensible Interface* der vierten Generation (AXI4) ab FPGAs der Virtex-6- und der Spartan-6-Familie, sodass die Kommunikationsinfrastrukturen auf dem Int-Comm-FPGA und dem Ext-Comm-FPGA durch Busse dieses Typs realisiert werden.

6.1.5 Systeminitialisierung

Die Systeminitialisierung beinhaltet die Initialisierung des Kommunikationsmoduls und der Verarbeitungsmodule und ist die wesentliche Aufgabe des externen Rekonfigurationskontrollers (siehe Abschnitt 6.2.1). Zur Realisierung dieser Aufgabe wird ein 1 Gbit großer NOR-Flash-Speicher integriert (siehe Abbildung 6.4). Dieser dient als Speicherort der (partiellen) Konfigurationsdateien aller FPGAs und des initialen Programmcodes (engl. boot code) des SpaceWire-RTCs.

Die Initialisierung des Kommunikationsmoduls besteht aus der Konfiguration des Ext-Comm-FPGAs und des Int-Comm-FPGAs. Nach der FPGA-Initialisierung übernimmt der Ext-Comm-FPGA die Initialisierung des SpaceWire-RTCs, welches aus dem Laden des initialen Programmcodes besteht. Der LEON2-FT des SpaceWire-RTCs übernimmt anschließend die Funktion der primären Steuerkomponente im DRPM-System.

Die Initialisierung eines Verarbeitungsmoduls besteht aus der Konfiguration des PR-FPGAs. Um diese Konfiguration zu ermöglichen, ist der externe Rekonfigurationskontroller zusätzlich an die Konfigurationsinfrastruktur des RAPTOR-X64 angeschlossen. Hierdurch ist es möglich, dynamisch zwischen der DB-SPACE-internen und der RAPTOR-weiten Konfigurationsinfrastruktur umzuschalten. Nach der Konfiguration des PR-FPGAs übernimmt die DPCU die Aufgabe als sekundäre Steuerkomponente im DRPM-System. Die Kommunikation zwischen den beiden DRPM-Steuereinheiten erfolgt über das Protokoll *Heterogeneous Multi Processor Communication Interface* (HMPCI), welches von Cozzi und Cassano realisiert wurde und in [43, 45] beschrieben ist.

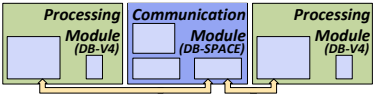
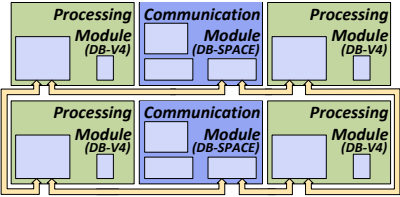
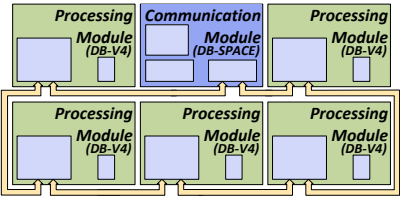
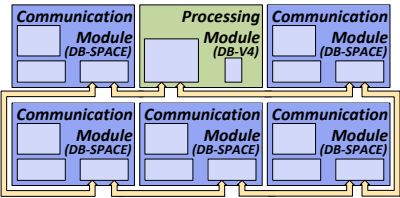
6.1.6 Systemvarianten des DRPMs

Neben der gewählten Basiskonfiguration existieren aufgrund der Modularität des RAPTOR-X64 Basisboards weitere Konfigurationsvarianten. Drei weitere Systemvarianten mit einer Vollbestückung mit jeweils sechs Modulen werden in [77, S. 5] vorgestellt. Die erste Konfiguration entspricht einer Verdopplung der zuvor beschriebenen Basiskonfiguration und besteht daher aus zwei Kommunikationsmodulen und vier Verarbeitungsmodulen. Die beiden anderen Konfiguration entsprechen der maximal möglichen Konfiguration unter Berücksichtigung beider Modultypen. Die erste Variante besteht aus einem Kommunikationsmodul und fünf Verarbeitungsmodulen, wodurch ein System mit dem Fokus auf die Verarbeitungsgeschwindigkeit aufgebaut werden kann. Die inverse Konfiguration, bestehend aus fünf Kommunikationsmodulen und einem Verarbeitungsmodul, realisiert ein System mit einer Vielzahl von Kommunikationsschnittstellen.

Alle aufgeführten Konfigurationen sind in Tabelle 6.1 dargestellt. In der zweiten Spalte sind die integrierten Schnittstellen des Systems aufgeführt. In der dritten Spalte sind die Speicher- und FPGA-Ressourcen erfasst. Die Angaben der FPGA-Ressourcen beinhalten nur die Ressourcen der Verarbeitungsmodule und sind daher als konservativ einzustufen, da in der Regel die Ressourcen der FPGAs der Kommunikationsmodule ebenfalls zur Datenverarbeitung verwendet werden können. Die Anzahl der FPGA-Ressourcen sind durch Slices, *Logic Cells*, DSP-Slices und BRAM-Blöcke angegeben. Da die Slice-Konfiguration FPGA-Familien-spezifisch ist (siehe Abschnitt 3.2), wird der von Xilinx definierte Terminus *Logic Cells* (eine LUT4 und ein FF) verwendet, um einen FPGA-Familienübergreifenden Vergleich zu ermöglichen.

Die Basiskonfiguration integriert insgesamt sechs SpaceWire-, eine WizardLink- oder SpaceFibre-, eine MIL-, eine CAN- und zwei UART-Schnittstellen sowie 54 GPIOs. Als externer Speicher stehen 8 GB DDR2- und 1 GB Flash-Speicher zur Verfügung. Die zwei Verarbeitungsmodule besitzen insgesamt 84 352 Slices, wodurch 168 704 LUTs und FFs (*Logic Cells*) für Datenverarbeitungsaufgaben bereitstehen. Weiterhin können 320 DSP-Slices und 1 692 kB FPGA-interner Speicher (BRAM) verwendet werden. Diese Ressourcen und Schnittstellen skalieren entsprechend der jeweiligen Anzahl der Module in den einzelnen Konfigurationen. Die dritte Konfiguration stellt z. B. ein leistungsfähiges Datenverarbeitungssystem mit 20 GB DDR2-Speicher und 210 800 Slices bereit. Die vierte Konfiguration integriert 30 SpaceWire- und fünf WizardLink- oder SpaceFibre-Schnittstellen und kann als Kommunikationsrouter verwendet werden.

Tabelle 6.1: Konfigurationsvarianten des DRPMs (basierend auf [77, S. 5])

Konfiguration	Schnittstellen	Ressourcen		
	6 x SpaceWire	8 GB	DDR2	
	1 x WizardLink	1 Gbit	Flash	
	oder SpaceFibre	84 352	Slices	
	1 x MIL	168 704	Logic Cells	
	1 x ADC/DAC	320	DSP-Slices	
	1 x CAN	1 692 kB	BRAM	
	2 x UART			
	54 x GPIO			
		12 x SpaceWire	16 GB	DDR2
		2 x WizardLink	2 Gbit	Flash
oder SpaceFibre		168 704	Slices	
2 x MIL		379 584	Logic Cells	
2 x ADC/DAC		640	DSP-Slices	
2 x CAN		3 384 kB	BRAM	
4 x UART				
108 x GPIO				
		6 x SpaceWire	20 GB	DDR2
		1 x WizardLink	1 Gbit	Flash
	oder SpaceFibre	210 880	Slices	
	1 x MIL	474 480	Logic Cells	
	1 x ADC/DAC	800	DSP-Slices	
	1 x CAN	4 230 kB	BRAM	
	2 x UART			
	54 x GPIO			
		30 x SpaceWire	4 GB	DDR2
		5 x WizardLink	5 Gbit	Flash
oder SpaceFibre		42 176	Slices	
5 x MIL		94 896	Logic Cells	
5 x ADC/DAC		160	DSP-Slices	
5 x CAN		846 kB	BRAM	
10 x UART				
270 x GPIO				

6.2 Realisierung des INDRA 2.0 DPR-Systems

Für die Realisierung eines performanten und flexiblen DPR-Systems werden die wesentlichen Komponenten des INDRA 2.0 DPR-Systems auf dem PR-FPGA beschrieben. Die (Re-) Konfiguration des DRPMs erfolgt über einen externen und einen internen Rekonfigurationskontroller. Durch die Implementierung und Verwendung eines internen und externen Rekonfigurationskontrollers können die Vorteile beider Typen kombiniert werden, ohne dass deren Nachteile zum Tragen kommen. Beide unterstützen

hierbei DPR. Neben den Rekonfigurationskontrollern wird eine performante Anbindung des FPGAs an den DDR2-Speicher des Verarbeitungsmoduls benötigt. Dies geschieht, wie eingangs beschrieben, über dedizierte Ein- und Ausgänge des *Multi-Port Memory Controllers* (MPMCs), sodass der PLB entlastet werden kann und einzelne Komponenten eine performante, auf sie zugeschnittene, Anbindung erhalten. So sind der interne Rekonfigurationskontroller und eine Kommunikationsbrücke zwischen dem statischen und dem dynamischen Bereich über dedizierte Kanäle angeschlossen. Für die, an die Kommunikationsbrücke angeschlossene, DPR-Kommunikationsinfrastruktur wurden verschiedene ein- und zweidimensionale Versionen durch DHHarMa erstellt.

6.2.1 Externer Rekonfigurationskontroller

Der externe Rekonfigurationskontroller wurde von Schlüssler implementiert und übernimmt mehrere wesentliche Aufgaben im DRPM-System, welche detailliert in [152] und zusammengefasst in [44, S. 82 ff.] beschrieben sind. Durch die Verwendung eines Flash-basierten CoolRunner-II CPLDs (XC2C384) ist die Komponente direkt nach dem Einschalten der Versorgungsspannung betriebsbereit. Zusätzlich ist eine Portierung auf eine strahlungsgehärtete Komponente möglich.

Aufbau

Abbildung 6.9 zeigt den internen Aufbau des externen Rekonfigurationskontrollers. Der *Controller* steuert den exklusiven Zugriff auf den Flash-Speicher über einen Zustandsautomaten (*Flash FSM*) und schaltet zwischen zwei Konfigurationsinfrastrukturen, DB-SPACE-intern oder RAPTOR-weit, um. Durch die Anbindung an die RAPTOR-Konfigurationsinfrastruktur ist es möglich, FPGAs auf anderen Modulen zu konfigurieren, wodurch eine sehr hohe Systemflexibilität erreicht wird.

Funktionen

Die wesentliche Aufgabe des externen Rekonfigurationskontrollers ist, wie bereits in Abschnitt 6.1.5 beschrieben, die Systeminitialisierung. Diese beinhaltet die Initialisierung des Kommunikationsmoduls und der Verarbeitungsmodule. Die Initialisierung des Kommunikationsmoduls besteht aus der Konfiguration der FPGAs (Ext-Comm-FPGA und Int-Comm-FPGA) und dem anschließenden Laden des initialen Programmcodes des SpaceWire-RTCs. Die Initialisierung des Verarbeitungsmoduls besteht aus der Konfiguration des PR-FPGAs. Nach der Initialisierung des Gesamtsystems übernimmt der SpaceWire-RTC die Kontrolle über den externen Rekonfigurationskontroller und steuert die externe (vollständige oder dynamisch partielle) Rekonfiguration der PR-FPGAs.

Auswertung

In diesem Abschnitt erfolgt eine Auswertung der einzelnen Funktionen anhand der erreichbaren Datenübertragungsraten. Diese Auswertung fasst die Ergebnisse aus [152] und [44, S. 82 ff.] zusammen.

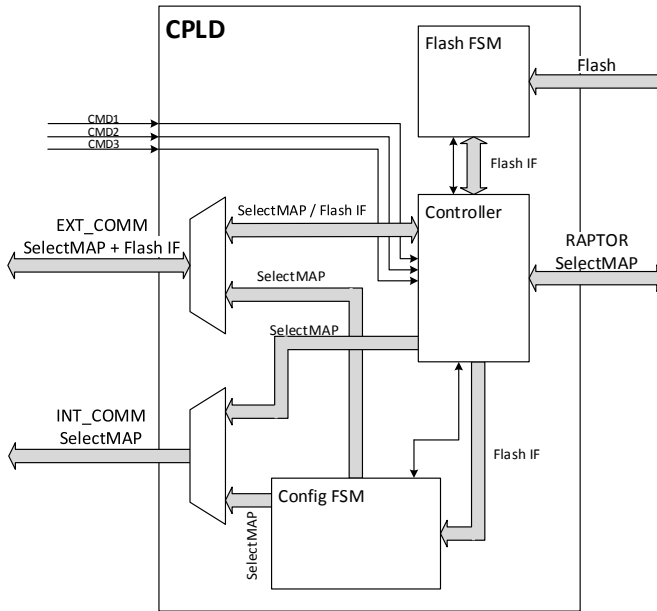


Abbildung 6.9: Übersicht des externen Rekonfigurationskontrollers [152, S. 14]

Flash-Anbindung

In Tabelle 6.2 sind die Zugriffszeiten und die resultierenden Datenübertragungsraten für Schreib- und Lesevorgänge dargestellt. Schreibtransfers können nur durch Einzelzugriffe, Lesetransfers zusätzlich im Burst-Verfahren ausgeführt werden. Somit können nach einer initialen Verzögerung von 160 ns, zu jeder Taktflanke Daten aus dem Flash-Speicher ausgelesen werden.

Tabelle 6.2: Verzögerungen und Datenübertragungsraten für Schreib- und Lesetransfers auf den Flash-Speicher (basierend auf [44, S. 82])

Transfertyp	Verzögerungen in ns	Datenübertragungsraten in MB/s
Schreibtransfer (Einzel)	120	16,7
Lesetransfer (Einzel)	180	11,1
Lesetransfer (Burst)	20	100,0

Konfiguration der FPGAs auf dem Kommunikationsmodul

Die Konfiguration des Int- und Ext-Comm-FPGAs entspricht einem Lesevorgang des Flash-Speicherinhalts. Durch die 8 bit breite Anbindung der SelectMAP-Schnittstelle und einem Takt von 50 MHz ergibt sich eine maximale Konfigurationsdatenrate von 50 MB/s (siehe Abschnitt 3.3.2). Wie in Tabelle 6.3 dargestellt, kann diese Rate nahezu erreicht werden. Die Differenz zum theoretischen Maximum ergibt sich durch die CPLD-interne Aufteilung der Daten in Blöcke zu je 128 kB. Durch diese Aufteilung kann es am Ende der Konfiguration zu einem hohen internen Verschnitt im letzten zu übertragenden Block kommen. Da dieser letzte Block vollständig übertragen wird, unabhängig von der Speicherbelegung, ergibt sich, je nach Größe der Konfigurationsdatei, ein abweichender Datenüberhang/Overhead.

Tabelle 6.3: Konfigurationsgeschwindigkeit und -zeit für die FPGAs des Kommunikationsmoduls (basierend auf [44, S. 84])

FPGA	Konfigurations- geschwindigkeit in MB/s	Konfigurationszeit in ms	Konfigurations- dateigröße in MB
Int-Comm	47,9	69,2	3,31
Ext-Comm	47,8	88,2	4,23

Konfiguration des FPGAs auf dem Verarbeitungsmodul

Für die Realisierung der Konfiguration des PR-FPGAs sind zwei Schritte nötig, welche beide innerhalb des CPLDs realisiert wurden. Der erste Schritt besteht aus dem Lesen der Konfigurationsdatei aus dem Flash. Anschließend müssen diese Daten über den RAPTOR-X64-Konfigurationsbus versendet werden. Aufgrund dieser beiden Schritte reduziert sich die maximale Konfigurationsdatenrate, wie in Tabelle 6.4 dargestellt, auf 18,2 MB/s. Da der RAPTOR-Konfigurationsbus mit 25 MHz betrieben wird, sind theoretisch bis zu 25 MB/s möglich. Die maximale Konfigurationsrate erreicht somit 72,8 % der möglichen Konfigurationsrate. Bei einer vollständigen Rekonfiguration im DRPM-Betrieb muss darüber hinaus die Dauer des Löschvorgangs berücksichtigt werden, welche mit 0,7 ms vermessen wurde.

SpaceWire-RTC Bootloader

Der initiale Programmcode des SpaceWire-RTCs ist in dem Flash-Speicher abgelegt. Damit dieser in den SpaceWire-RTC transportiert werden kann, wurde der Ext-Comm-FPGA, wie in Abbildung 6.10 dargestellt, erweitert. Nach Beendigung der FPGA-Konfigurationen, beginnt der *EXTCONF-Controller* auf dem Ext-Comm-FPGA mit dem Transfer des Programmcodes für den SpaceWire-RTC in 400 Byte Blöcken in den SRAM-Speicher, welcher über die *Mem-Bridge* (siehe Abbildung 6.8) angeschlossenen ist.

Die Datenübertragungsrate ist aufgrund der hohen Initialisierungskosten abhängig von der Größe des Programmcodes. Wie in Abbildung 6.11 dargestellt, kann eine

Tabelle 6.4: Konfigurationsdatenrate und -zeit für die vollständige und partielle Rekonfiguration des PR-FPGAs des Verarbeitungsmoduls (basierend auf [44, S. 85])

Rekonfiguration	Konfigurationsdatenrate in MB/s	Konfigurationszeit in ms	Konfigurationsdateigröße in MB
vollständig	18,1	227,7	4,13
partiell	18,2	0,0054 - 219,9	0,001 - 4

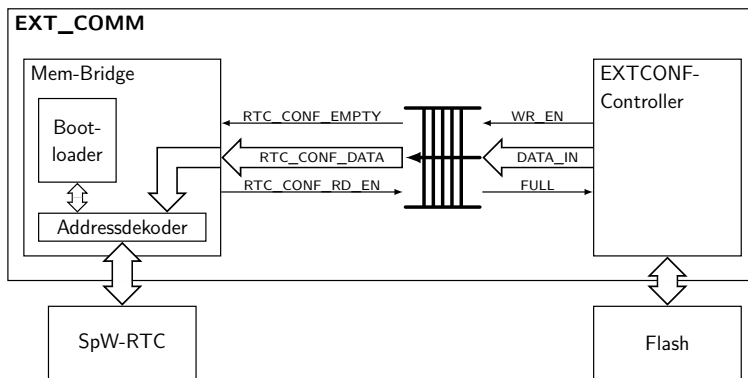


Abbildung 6.10: Verbindung zwischen dem SpaceWire-RTC (über die Mem-Bridge) und dem Flash-Speicher [152, S. 28]

Rate von bis zu 19,07 MiB/s bzw. 20 MB/s erreicht werden. Der Bootloader für den SpaceWire-RTC besteht aus 74 Instruktionen und ist im Anhang (siehe Abschnitt C.2, Abbildung A.12) dargestellt. Details zu den einzelnen Schritten sind in [152, S. 24 ff.] beschrieben.

6.2.2 Interner Rekonfigurationskontrolller

Der *Self-Hosting Reconfiguration Controller* (SHRC) realisiert durch die Anbindung an die interne Konfigurationsschnittstelle ICAP (siehe Abschnitt 3.3.2) und den damit verbundenen schreibenden und lesenden Zugriff auf den Konfigurationsspeicher, mehrere Funktionen [45, S. 181]. So steuert der SHRC als interner Rekonfigurationskontrolller die FPGA-interne, dynamisch partielle Rekonfiguration durch einen Schreibvorgang. Der lesende Zugriff auf den Konfigurationsspeicher ermöglicht weiterhin die Extraktion der aktuellen Konfiguration eines FPGAs zur Laufzeit.

Durch die Anbindung an die interne Konfigurationsschnittstelle ICAP kann der interne Rekonfigurationskontrolller erweitert werden und Verfahren zur Erhöhung der

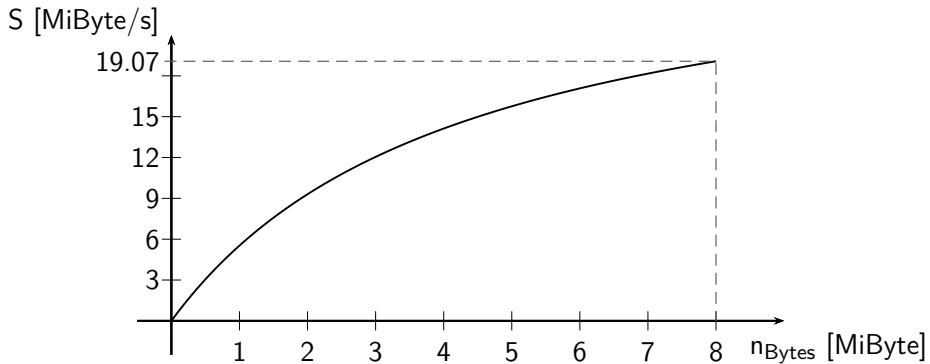


Abbildung 6.11: Datenübertragungsrate für die Übertragung des Programmcodes für den SpaceWire-RTC [152, S. 30]

Fehlertoleranz gegenüber SEEs implementieren. Konkret wurde der SHRC um zwei Scrubbing-Verfahren (siehe Abschnitt 5.7.3) erweitert. Genaue Details erfolgen in dem Abschnitt 6.4.1. Aufgrund der Implementierung des SHRCs werden einige Komponenten zur Realisierung bereits in diesem Abschnitt genannt. So integriert der SHRC den Hard-IP-Block *Frame ECC* zur Realisierung eines *Readback Scrubbings* zur Detektion von Ein- und Zweibitfehlern und zur Korrektur von Einbitfehlern (SECEDED) im Konfigurationsspeicher des FPGAs (siehe Abschnitt 5.7.3). Bevor detaillierter auf die Funktionen eingegangen wird, erfolgt zunächst eine Übersicht über den Aufbau des SHRCs.

Aufbau

Abbildung 6.12 stellt die Einzelkomponenten des SHRCs hierarchisch dar. Der IP-Core besteht aus vier Komponenten bzw. Entities: *Top*, *Self-Hosting Reconfiguration Controller*, *Vertex Configuration Manager* und *Readback Scrubbing Unit*.

Top Auf der obersten Ebene *Top* integriert der IP-Core die externen Schnittstellen, einen *PLB Slave* (mit *Burst*-Funktionalität) und zwei *Local Link* Kanäle zur Streaming-basierten, direkten Kommunikation mit dem DDR2-Speichercontroller, einen Interruptcontroller und die Komponente *Self-Hosting Reconfiguration Controller*.

Self-Hosting Reconfiguration Controller Die Komponente *Self-Hosting Reconfiguration Controller* (SHRC) ist die Hauptkomponente und Namensgeber des IP-Cores. Zur Steuerung und Überwachung des IP-Cores existieren mehrere, per PLB zugreifbare, Kontroll- und Statusregister. Zusätzlich zum internen Status des IP-Cores, kann der Status des Paket-Prozessors angefordert werden. Die Implementierung eines autonomen *Readback Scrubbings* erfolgt über den Zustandsautomaten *Readback Scheduling* sowie einen BRAM mit 256 Einträgen (*Readback Scheduling BRAM*). Der Zustandsautomat

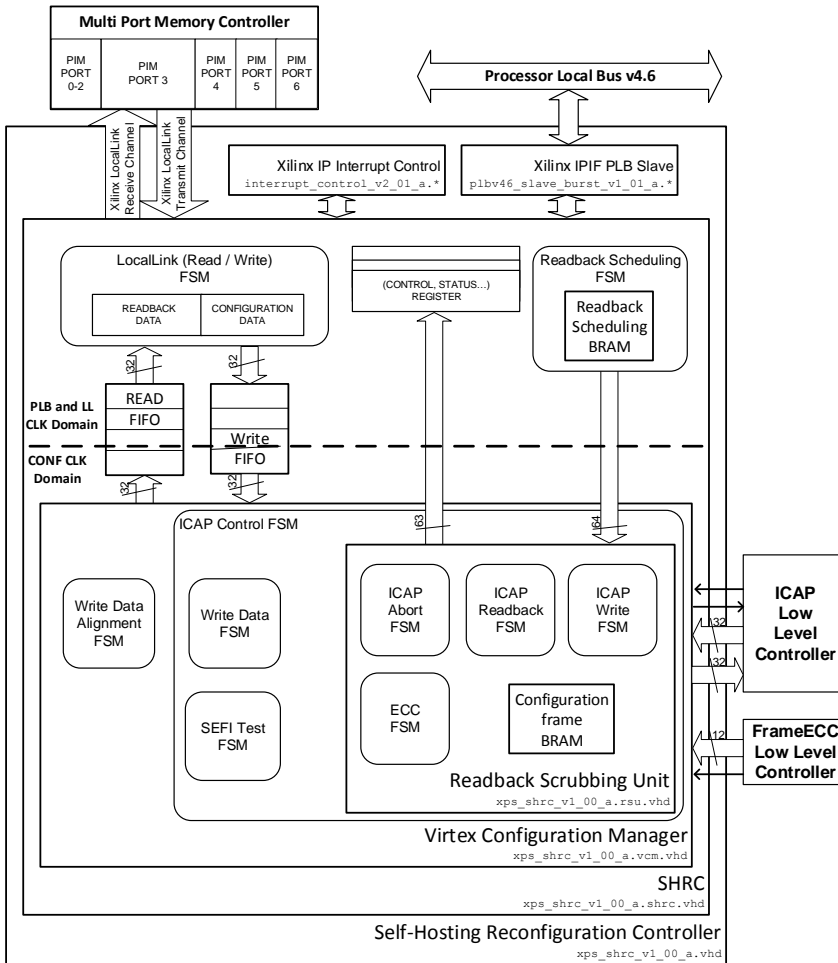


Abbildung 6.12: Übersicht über den Self-Hosting Reconfiguration Controller [45, S. 181]

Local Link (Read/Write) steuert den Datenfluss zwischen dem IP-Core und dem Speichercontroller (MPMC) im Falle eines Schreib- oder Lesevorgangs. Die Kommunikation mit dem MPMC wird über einen *Soft Direct Memory Access (SDMA)* Kontroller realisiert [209, S. 128 ff.]. Dieser ermöglicht, durch die Erstellung und Verwendung von verketteten Deskriptoren, einen kontinuierlichen Datenfluss, selbst wenn die Daten in dem Speicher verteilt abgelegt sind. Weitere Details hierzu sind in Abschnitt 6.2.3 beschrieben.

Virtex Configuration Manager Der Virtex Configuration Manager (VCM) steuert die Kommunikation mit dem Paket-Prozessor über die angebundene ICAP-Schnittstelle. Der Zustandsautomat *Write Data Alignment* sorgt dafür, dass die, über Local Link empfangenen, Konfigurationsdaten Wort-ausgerichtet sind, bevor diese an die ICAP-Schnittstelle weitergeleitet werden. Die Datenflusssteuerung für die Durchführung der dynamisch partiellen Rekonfiguration wird über den Zustandsautomaten *ICAP Write*, welcher die in Tabelle A.2 im Anhang A.2 dargestellte Konfigurationssequenz implementiert. Der Zustandsautomat *SEFI Test* führt vor jedem Schreiben oder Lesen über die ICAP-Schnittstelle Testschritte durch, um den internen Zustand des Paket-Prozessors und der ICAP-Verbindung vorab zu prüfen (siehe Abschnitt 5.4.1). Weiterhin ist der VCM mit dem Hard-IP-Block *Frame ECC* zur Erkennung und Korrektur von Einbitfehlern und der Erkennung von Zweibitfehlern (SECEDED) verbunden. Da der *Frame ECC* Block nur einmal in der Hardware existiert, erfolgt die Instanziierung in dieser Entity, sodass Redundanzverfahren (siehe Abschnitt 5.6.2 und 5.7.3) auf die Steuereinheit *Readback Scrubbing Unit* angewendet werden können, um die Zuverlässigkeit dieser kritischen Komponente zu erhöhen. Fehlerinformationen werden in einer FIFO des VCMs gespeichert. Der Zustandsautomat *ICAP Control* steuert den Zugriff auf die ICAP-Schnittstelle, da mehrere Zustandsautomaten zur Implementierung ihrer Funktion einen exklusiven Zugriff auf die ICAP-Schnittstelle benötigen. Der Datentransfer zwischen den unterschiedlichen Taktdomänen erfolgt über asynchrone FIFOs (im Falle von Konfigurations-, oder Fehlerinformationsdaten) oder Dual-Port-BRAMs (im Falle der Scrubbing Tasks).

Readback Scrubbing Unit Die *Readback Scrubbing Unit* (RSU) realisiert die Konfigurationsschritte für einen Lesevorgang (Readback) oder für das Readback Scrubbing. Die Konfigurationsschritte basieren auf den von Xilinx in [36] und [35] dokumentierten Empfehlungen und sind durch vier Zustandsautomaten realisiert. Der Zustandsautomat *ICAP Abort* generiert eine Abbruchsequenz für die ICAP-Schnittstelle, um diese in einen definierten Zustand zu versetzen, bevor ein Lesen oder Schreiben initialisiert wird. Der Zustandsautomat *ICAP Readback* sendet die Kommandosequenz zum Auslesen des Konfigurationsspeichers. Bei einem *Readback Scrubbing* werden die gelesenen Daten eines *Configuration Frames* in einem BRAM (*Configuration Frame BRAM*) zwischengespeichert, um im Falle eines erkannten Einbitfehlers eine Korrektur vornehmen zu können. Bei einem Lesevorgang werden die gelesenen Daten an den VCM übertragen, in einer FIFO zwischengespeichert und über den Zustandsautomaten *Local Link (Read/Write)* des SHRCs an den MPMC gesendet und in dem daran angeschlossenen DDR2-Speicher abgelegt. Die Korrektur eines Einbitfehlers erfolgt durch den Zustandsautomaten *ECC*. Dieser wertet die Statussignale der *Frame ECC* Komponente aus und korrigiert gezielt das fehlerhafte Bit in dem zwischengespeicherten *Configuration Frame* (Details folgen in Abschnitt 6.4.1). Anschließend wird der korrigierte Inhalt durch den RSU-internen Zustandsautomaten *ICAP Write* zurück in den Konfigurationsspeicher geschrieben.

Die RSU wurde als dedizierte Komponente mit registrierten Ein- und Ausgängen implementiert, sodass Redundanzverfahren implementiert werden können, um die Zuverlässigkeit dieser kritischen Komponente zu erhöhen.

Funktionen

In diesem Abschnitt werden die Rekonfigurationsfunktionen des SHRCs beschrieben. Im Vordergrund stehen dabei die Realisierung der DPR und des Zurücklesens.

Dynamisch partielle Rekonfiguration mit Modulumplatzierung

Eine dynamisch partielle Rekonfiguration in dem SHRC erfolgt im Wesentlichen durch den Zustandsautomaten *ICAP Write* im VCM. Der Zustandsautomat steuert den Datenfluss der, über Local Link empfangenen, Daten und setzt die Steuersignale der ICAP-Schnittstelle. Die Wortausrichtung der Konfigurationsdaten durch den Zustandsautomat *Write Data Alignment* wurde implementiert, um eine einfache Manipulation an der FAR in dem Schreibprozess durchführen zu können. Diese Manipulation ermöglicht es, dass die Konfigurationsdatei an eine andere, vom Original abweichende, Konfigurationsspeicheradresse des FPGAs geschrieben wird. Aus Modulsicht bedeutet dies, dass ein Modul an eine andere Position auf dem FPGA platziert wird. Diese Funktionsweise zur Umplatzierung von Modulen wurde erstmalig durch den Filter *REPLICA (RE)location Per onLINE Configuration Alternation*) in [93] für Virtex-FPGAs beschrieben.

Die Voraussetzungen für die Modulumplatzierung sind eine homogene Kommunikationsinfrastruktur zwischen dem statischen und dem dynamischen Bereich und identische FPGA-Ressourcen an dem neuen Zielort (siehe Abschnitt 3.5). Weitere Details zur PR-Kommunikationsinfrastruktur im DRPM sind in Abschnitt 6.2.4 beschrieben.

Zurücklesen des Konfigurationsspeicherinhalts

Das Lesen des Konfigurationsspeicherinhalts kann in zwei Kategorien unterteilt werden [228, S. 99 ff.]:

- *Readback Verify*: Lesen aller Konfigurationsspeicherzellen inklusive aktueller Werte der LUTRAM-, SRL16- und BRAM-Komponenten
- *Readback Capture*: Lesen der in *Readback Verify* aufgeführten Speicherkomponenten sowie der aktuellen Werte der CLB- und IOB-Register

Das *Readback Verify* beschreibt somit den Standard beim Zurücklesen des Konfigurationsspeicherinhalts und wird ebenfalls in der aktuellen Implementierung verwendet. Ein *Readback Capture* kann durch ein Kommando an den Paket-Prozessor initialisiert werden. Durch eine Erweiterung des RSU-Zustandsautomats *ICAP Readback* könnte diese Lesekategorie zusätzlich integriert werden.

Auswertung

In diesem Abschnitt erfolgt eine Auswertung anhand der FPGA-Ressourcen, der Datenübertragungsraten sowie der Leistung.

FPGA-Ressourcen

Die benötigten FPGA-Ressourcen des IP-Cores SHRCs sind in Tabelle 6.5 dargestellt. Da die Anzahl an Slices von Synthese zu Synthese variiert, wurden die feingranulareren Bewertungsmaße Register und LUTs hinzugefügt. Weiterhin sind die Anzahl der BRAMs und DSPs aufgeführt. Das Zeichen '|' dient als Hierarchieseparator. Eine übergeordnete Komponente enthält sowohl die eigenen als auch die FPGA-Ressourcen der integrierten Komponente(n). So enthalten beispielsweise die Angaben zum SHRC|VCMs die FPGA-Ressourcen der RSU.

Tabelle 6.5: FPGA-Ressourcen des SHRCs (basierend auf [45, S. 186])

xps_shrc	Slices	Register	LUTs	BRAMs	DSPs
SHRC	1 076	1 178	2 093	3	0
SHRC VCM	851	869	1 184	0	0
SHRC VCM RSU	225	309	311	1	0
INTERRUPT_CONTROLLER	11	15	13	0	0
PLBV46_SLAVE_BURST	172	199	198	0	0
Gesamt	1 259	1 392	2 304	3	0

Leistungsfähigkeit

Der SHRC ermöglicht das Schreiben und Lesen des Konfigurationsspeichers über die ICAP. Die verfügbare Bandbreite von 3,2 Gbit/s bzw. 400 MB/s (siehe Abschnitt 3.3.2) kann aufgrund von Wartezyklen der Local Link Anbindung des MPMCs nicht erreicht werden. Abbildungen im Anhang C.1 zeigen die Wartezyklen durch die Simulation mit ModelSim und auf dem FPGA mit ChipScope. Die daraus resultierenden Datentransferraten sind in Abbildung 6.13 für unterschiedliche Transferlängen bei fester, größtmöglicher Burst-Größe von 32 Wörtern dargestellt. Auf eine Variation der Burst-Größen wurde verzichtet, da bei der Anwendung als Rekonfigurationskontroller stets größere Datenmengen zu erwarten sind.

Da der Local Link TX Kanal die Datenrichtung vom DDR2-Speicher zum SHRC beschreibt, entsprechen die Datentransferraten der maximalen Konfigurationsgeschwindigkeit. Die Raten variieren von 142,2 MB/s bis maximal 216,89 MB/s und erreichen zwischen 35,6 % und 54,2 % des theoretischen Maximums. Der RX Kanal, welcher für die Readback Funktion verwendet wird, erzielt in der Regel höhere Datentransferraten. Sie liegen zwischen 139,1 MB/s und maximal 278,1 MB/s und erreichen damit 34,8 % und 69,5 % des theoretischen Maximums. Ein Readback erfolgt daher bei größeren Datenmengen etwa 28 % schneller als eine Rekonfiguration.

Die maximal mögliche Datenübertragungsrate kann hingegen ohne die externe Anbindung an den MPMC erreicht werden. Dies wird durch das Readback Scrubbing

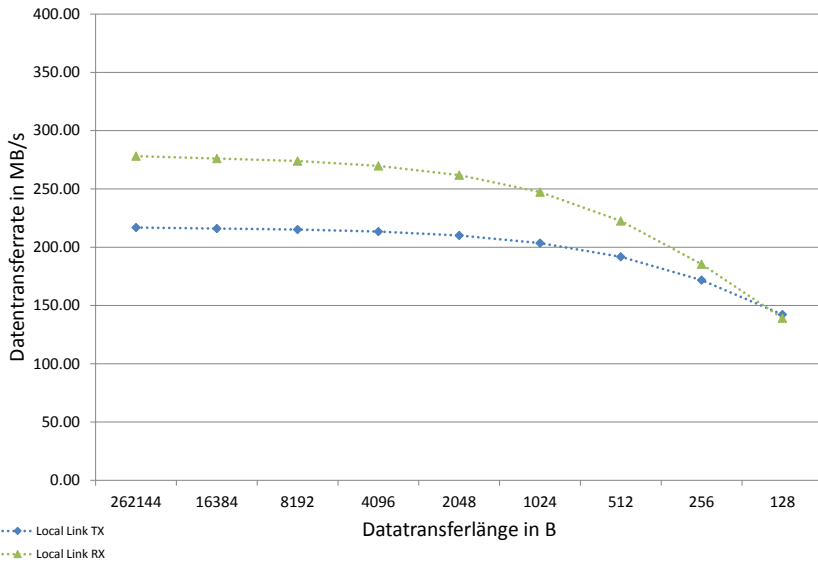


Abbildung 6.13: Datenübertragungsraten für Local Link

mit einer internen Anbindung an einen BRAM-Block verdeutlicht. Tabelle 6.6 fasst die maximalen Datenübertragungsraten für die drei Funktionen des SHRCs zusammen.

Tabelle 6.6: Datenübertragungsraten des SHRCs für die unterstützten Funktionen

Funktion	Max. Datenübertragungsrate (in MB/s)
DPR	217
Readback	278
Readback Scrubbing	400

In Tabelle 6.7 sind typische Zeiten für DPR- und Readback-Vorgänge mit den, in Tabelle 6.6 aufgeführten Datenübertragungsraten dargestellt.

Leistung

Die Leistung des SHRCs wurde mittels des Xilinx *Power Estimators* (XPE, Version 11.1) ermittelt. Die Anteile teilen sich auf die Takt-, Logik- und BRAM-Komponenten auf und ergeben insgesamt eine Leistung von 54 mW. Der Großteil der Leistung wird für die Takterzeugung (32 mW) aufgewendet. Die Leistung aller Logikelemente des SHRCs beträgt 17 mW.

Tabelle 6.7: DPR- und Readback-Zeiten für typische Konfigurationsdateigrößen (basierend auf [44, S. 86])

Konfigurationsdaten (in B)	Zeit für DPR (in μ s)	Zeit für Readback (in μ s)
69 273	320	249
103 910	481	373
173 183	801	662
311 221	1 440	1 119

Zusammenfassung

In Tabelle 6.8 sind die Kenngrößen, Merkmale und Funktionen des internen und externen Rekonfigurationskontrollers gegenübergestellt. Die Ergebnisse der Scrubbings des SHRCs sind hierbei vorgegriffen und werden im Abschnitt 6.4.1 detailliert beschrieben. Der externe Rekonfigurationskontroller ist an die FPGA-externe Konfigurationsschnitt-

Tabelle 6.8: Gegenüberstellung des externen und internen Rekonfigurationskontrollers im DRPM (basierend auf [77])

Funktionen/Merkmale	Ext. Rekonf.	Int. Rekonf. (SHRC)
Konfigurations-schnittstelle	SelectMAP (8 bit)	ICAP (32 bit)
Speicherart für Konfigurationsdateien	Flash (bis zu 128 MB)	DDR2-SDRAM (bis zu 1 GB)
Initiale/Vollständige Konfiguration	✓ (DB-SPACE: 47,9 MB/s extern: 18,1 MB/s)	✗
Dynamisch partielle Rekonfiguration /Blind Scrubbing	✓ (extern: 18,2 MB/s)	✓ (217 MB/s)
Readback	✗	✓ (278 MB/s)
Readback Scrubbing	✗	✓ (400 MB/s)
Fehlerstatistik und -Monitoring	✗	✓ (SECEDED)

stelle SelectMAP angeschlossen, welche bei einer 8 bit breiten Anbindung je nach Konfigurationstakt eine maximale Konfigurationsdatenrate von 50 MB/s (DB-SPACE-intern) bzw. 25 MB/s (RAPTOR-weit) ermöglicht (siehe Abschnitt 3.3.2). Als Speicherort für die Konfigurationsdateien stehen bis zu 1 Gbit Flash-Speicher zur Verfügung. Der externe Rekonfigurationskontroller implementiert die Funktionen der initialen/vollständigen und dynamisch partiellen Rekonfiguration. Durch die letztgenannte Funktion wird das Abschwächungsverfahren Blind Scrubbing realisiert. DB-SPACE-intern kann mit einer Konfigurationsdatenrate von 47,9 MB/s nahezu das mögliche Maximum erreicht werden. Extern ist sie jedoch auf 18,1 MB/s für eine vollständige Rekonfiguration, respektive 18,2 MB/s für eine DPR begrenzt. Das Lesen des Konfigurationsspeichers ist nicht implementiert. Das Abschwächungsverfahrens Readback Scrubbing oder Fehlerstatistikmechanismen bzw. ein Fehler-Monitoring können somit nicht realisiert werden.

Der Self-Hosting Reconfiguration Controller realisiert einen internen Rekonfigurationskontroller auf dem PR-FPGA und ist an die FPGA-interne Konfigurationsschnittstelle ICAP angeschlossen. Diese ermöglicht durch die 32 bit breite Anbindung und einen Konfigurationstakt von 100 MHz eine maximale Konfigurationsdatenrate von 400 MB/s. Eine initiale Konfiguration ist jedoch nicht möglich. Durch die Anbindung an den DDR2-SDRAM, welcher den Speicherort der Konfigurationsdateien auf dem Verarbeitungsmodule mit bis zu 1 GB bildet, kann diese Datenrate in den Implementierungen nur in einer der aufgeführten Funktion erreicht werden. Bei einer DPR kann eine Datenrate von 217 MB/s erreicht werden, welches 54,2 % der maximalen Datenrate entspricht. Ein Vergleich mit der Datenrate des externen Rekonfigurationskontrollers resultiert in einem Faktor von ca. 12. Die Datenrate beim Lesen ist mit 278 MB/s höher und entspricht 69,5 % des theoretischen Maximums. Durch die Integration der FPGA-internen *Frame ECC* Komponente wird ein Readback Scrubbing mit SECDED-Funktionalität realisiert, welches durch die Verwendung eines Hamming-Codes ohne einen Vergleich mit den (golden copy) Konfigurationsdaten arbeitet. Hierdurch entfällt die zusätzliche Kommunikation mit einem Speicher, sodass die maximale Konfigurationsdatenrate von 400 MB/s für das Lesen und die Korrektur im Einbitfehlerfall erreicht wird. Basierend auf den Auswertungen der Frame ECC Komponente erfolgt ein internes Fehler-Monitoring und die Erstellung einer Fehlerstatistik für Ein- und Mehrbitfehler. Ferner ist es möglich, auf eine ansteigende Fehlerrate durch Rekonfigurationen mit robusteren System- oder PR-Modulvarianten zu reagieren und permanente Fehler auf einem FPGA zu erkennen.

Durch die Implementierung und Verwendung eines internen und externen Rekonfigurationskontrollers können die Vorteile beider Typen kombiniert werden, ohne deren Nachteile teilen zu müssen. Für die initiale Konfiguration und die Korrektur im detektierten Mehrbit- oder SEFI-Fehlerfall wird der externe Rekonfigurationskontroller verwendet. Im Einbitfehlerfall erfolgt die Korrektur durch ein gezieltes, Configuration Frame-basiertes Scrubbing mit Hilfe der Fehlerstatistik des Readback Scrubbings. Die DPR für die Ausführung einer neuen Datenverarbeitung/Anwendung wird über den SpaceWire-RTC gesteuert und von dem internen Rekonfigurationskontroller ausgeführt.

6.2.3 Multi-Port Memory Controller

Der *Multi-Port Memory Controller* (MPMC) integriert das 4 GB große DDR2-SODIMM in das EDK-System und basiert auf dem IP-Core (MPMC v6.05.a, [209]) von Xilinx. In diesem Abschnitt wird auf die einzelnen Kommunikationsschnittstellen eingegangen. Die eingangs erwähnten erweiterten Funktionen des Speicherkontrollers – Fehlerinjektion und ECC-Fehlerstatistik – wurden von Lallet implementiert und werden in Abschnitt 6.4.1 beschrieben.

Aufbau

Abbildung 6.14 stellt die Einzelkomponenten des MPMCs dar. Je nach Konfiguration besteht der IP-Core aus einer Vielzahl von HDL-Dateien (Verilog und VHDL), sodass eine abstrahierte Darstellung, basierend auf [209, S. 56] und [209, S. 67], gewählt wurde. Bei der Darstellung liegt der Fokus auf die zuvor genannten Erweiterungen.

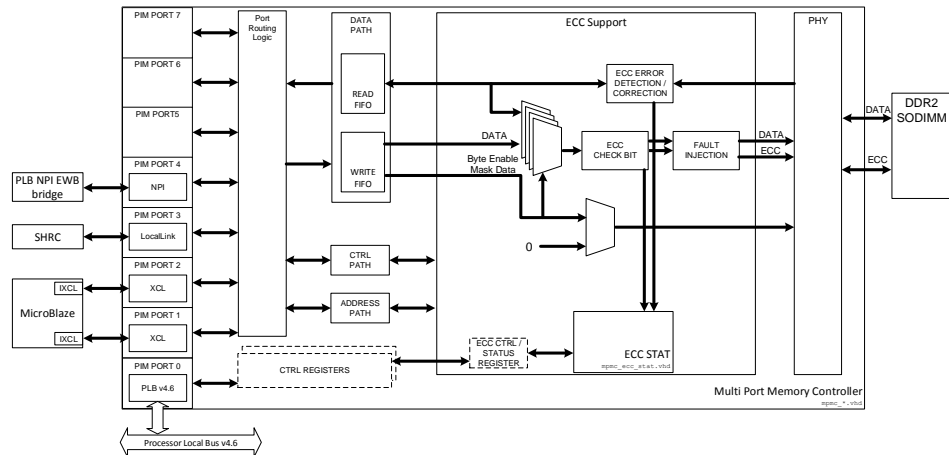


Abbildung 6.14: Übersicht über den Multi-Port Memory Controller [45, S. 187] (basierend auf [209, S. 56] und [209, S. 67])

Auf der linken Seite sind die einzelnen Ports und Verbindungen zu den IP-Cores dargestellt. Xilinx verwendet für die einzelnen Schnittstellen den Terminus *Personality Interface Module* (PIM). Der Datentransfer zwischen den angeschlossenen IP-Cores und dem Speicher erfolgt in dem DRPM über mehrere Komponenten. Die Komponente *CTRL Path* steuert den Datenfluss. *Data Path* besteht aus Schreib- und Lese-FIFOs, in denen die Daten zwischengespeichert werden. *Address Path* spezifiziert die Adresse von der gelesen bzw. an die geschrieben wird. Die *Port Routing Logic* realisiert einen Multiplexer, um den Datenfluss zwischen dem DDR2-Speicher und den einzelnen PIM Ports zu ermöglichen. Die Komponente *ECC Support* zeigt insbesondere den Datenfluss durch

die Erweiterung *Fault Injection* und die Erweiterung *ECC Stat*, in der die Ereignisse der SECDED-Komponente und der *ECC-Check-Bit*-Komponente protokolliert werden. Die *PHY* beschreibt die physische Schnittstelle zum Speicher und führt eine Umwandlung des *Double Data Rate* (DDR) Datenbusses des Speichers in einen *Single Data Rate* (SDR) Datenbus mit doppelter Datenbusbreite durch [209, S. 66]. Der DDR2-Datenbus im Verarbeitungsmodul ist 40 bit breit, wovon 8 bit ECC-Daten enthalten, und arbeitet mit einer Taktfrequenz von 400 MHz.

Kommunikationsschnittstellen

Der MPMC stellte unterschiedliche Kommunikationsschnittstellen (PIM) zur Verfügung [209, S. 123 ff.]. Im DRPM werden fünf der möglichen acht Ports mit vier PIM-Typen verwendet (siehe Abbildung 6.14). Die Eigenschaften dieser PIM-Typen sind in [45, S. 188 ff.] dokumentiert und werden in diesem Abschnitt zusammengefasst.

Xilinx CacheLink PIM Der XCL PIM unterstützt Lese- und Schreibtransfers mit einer Länge von 1, 4, 8 oder 16 Wörtern und wird allgemein für die Anbindung des Prozessors an den DDR2-Speicher verwendet. Im DRPM wird der MicroBlaze der DPCU über zwei PIM-Ports angeschlossen. Konkrete Details zu XCL sind in [209, S. 124 ff.] dokumentiert.

Soft Direct Memory Access PIM Durch den SDMA Controller wird eine Schnittstelle für einen Streaming-basierten Datenaustausch bereitgestellt. SDMA verfügt über zwei *Local Link* Kanäle: Einen für die Sende- und einen für die Empfangsrichtung. Das *Local Link* Protokoll kapselt den Zugriff auf die MPMC-internen Schreib- und Lese-FIFOs (siehe Abbildung 6.14) und erweitert somit den nativen Zugriff um weitere Funktionen. So wird ein kontinuierlicher Datenfluss unter Verwendung von Deskriptoren realisiert. Die Verwendung der verketteten Deskriptoren (*Scatter Gather*) ermöglicht selbst dann einen kontinuierlichen Datenfluss, wenn die Daten im Speicher verteilt abgelegt sind. Eine Anpassung dieser Deskriptorkette ist zur Laufzeit möglich. Bei der Übertragung können weiterhin zu übertragende, anwendungsspezifische Daten in dem *Local Link* Header oder Footer übermittelt werden. Die Beendigung oder ein Fehlerfall auf einem *Local Link* Kanal wird über dedizierte Interrupts signalisiert. Der Aufbau der Deskriptoren und weitere Details zu SDMA sind in [209, S. 128 ff.] beschrieben.

Native Port Interface PIM NPI ermöglicht über Adress-, Daten- und Kontrollsignale einen direkten Zugriff auf die MPMC-internen Schreib- und Lese-FIFOs (siehe Abbildung 6.14). Daten können dabei simultan über zwei separate Kanäle transferiert werden. Die maximal unterstützte Burst-Größe ist abhängig von der Datenbreite der NPI-Anbindung. Die Anbindung an die PLB-NPI-EWB-Bridge erlaubt durch die Datenbreite von 64 bit eine maximale Burst-Größe von 64 Wörtern. Details zu NPI sind in [209, S. 173 ff.] dokumentiert.

Auswertung

In diesem Abschnitt erfolgt eine Auswertung der einzelnen Schnittstellen anhand der benötigten FPGA-Ressourcen, der erreichbaren Datenübertragungsraten und der Leistung.

FPGA-Ressourcen

Die benötigten FPGA-Ressourcen des MPMCs sind in Tabelle 6.9 dargestellt. Die Kernkomponente *mpmc_core* belegt den Großteil der FPGA-Ressourcen. Der Anteil zur Realisierung der Local Link Kommunikationsschnittstelle ist ebenfalls hoch. Die FPGA-Ressourcen der NPI-Schnittstelle sind im Report nicht separat ausgewiesen und somit in der Kernkomponente *mpmc_core* enthalten.

Tabelle 6.9: FPGA-Ressourcen des MPMCs (basierend auf [45, S. 197])

mpmc	Slices	Register	LUTs	BRAMs	DSPs
mpmc_ecc_control	142	198	161	0	0
mpmc_ecc_stat	518	758	1 045	0	0
mpmc_core	3 280	4 454	3 992	17	0
XCL	57	73	84	0	0
PLB	469	598	731	0	0
Local Link (SDMA)	1 202	1 174	1 942	0	0
Gesamt	5 668	7 255	7 955	17	0

Leistungsfähigkeit

Die Datenübertragungsraten wurden in den Abschnitten der einzelnen, an den MPMC angeschlossenen, IP-Cores bereits dokumentiert und sind in Tabelle 6.10 für die relevanten Protokolle zusammengefasst dargestellt. Auffällig ist insbesondere der Einfluss des Local Link Protokolls auf die Datenübertragungsraten. Hier ist ein deutlicher Protokoll-Overhead im Vergleich zur selbstimplementierten, Deskriptor-basierten NPI-Kommunikation der PLB-NPI-EWB-Bridge erkennbar. Die NPI-Kommunikation bieten in Leserichtung eine um 24,1 % und in Schreibrichtung sogar um 55,0 % höhere Datenübertragungsrate.

Tabelle 6.10: Datenübertragungsraten des MPMCs

Protokoll/PIM	Max. Datenübertragungsrate (in MB/s)	
	Lesetransfer	Schreibtransfer
Local Link (SDMA)	278,1	216,9
NPI	345,1	336,3

Leistung

Die Leistung des MPMCs ist aufgrund der benötigten Taktversorgung und der Anzahl verwendeten FPGA-Ressourcen mit etwa 325 mW die höchste Teilleistung des Gesamtsystems. Die Leistung der Taktversorgung und der Logikelemente liegen beide jeweils bei etwa 137 mW.

6.2.4 Kommunikation im DPR-System des PR-FPGAs

Die Kommunikation zwischen dem statischen und dem dynamischen Bereich im DPR-System wird, wie in Abbildung 6.15 dargestellt, im Wesentlichen aus drei Komponenten aufgebaut: Eine Y-Brücke mit Anbindung für den PLB und NPI-Kanäle an den EWB, eine PR-Tile-Management-Komponente und der eigentlichen DPR-Kommunikationsinfrastruktur. Beide Komponenten werden dem EDK-System des PR-FPGAs hinzugefügt und erweitern dieses zur Realisierung des flexiblen INDRA 2.0 DPR-Systems Abschnitt 3.5.7.

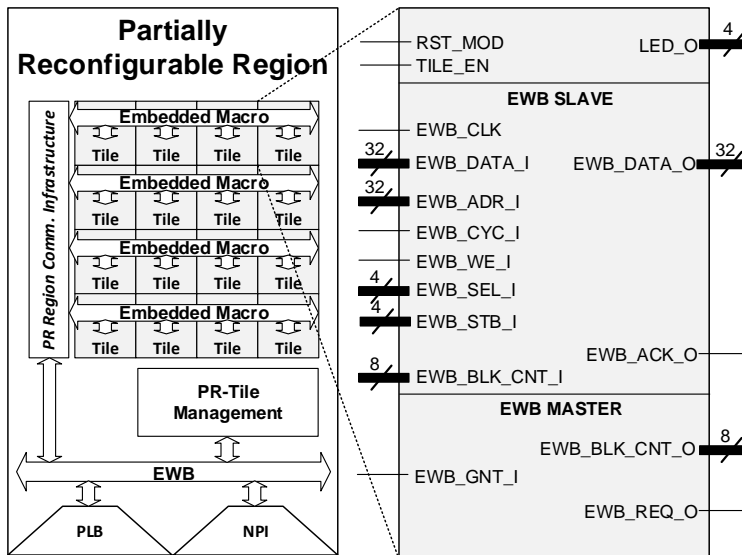


Abbildung 6.15: Übersicht der Komponenten in der PR-Region
(basierend auf [45, S. 219])

6.2.4.1 PLB-NPI-EWB-Brücke

Die PLB-NPI-EWB-Brücke kontrolliert den Datenfluss von und zum dynamischen Bereich (PR-Region) des PR-FPGAs und ist somit ein wesentlicher Bestandteil des DPR-Systems des PR-FPGAs [45, S. 201]. Die PR-Region ist, wie in Abbildung 6.15 dargestellt, in PR-Tiles aufgeteilt. Die PR-Tiles über eine eingebettete Kommunikationsinfrastruktur

tur an einen registrierten Wishbone Bus angeschlossen (siehe Abschnitt 4.4.1). Über die PLB-Anbindung kann jede Systemkomponente direkt eine Kommunikation mit den PR-Modulen aufbauen. Zusätzlich existiert eine dedizierte Verbindung des MPMC's über eine NPI-Anbindung.

Aufbau

Abbildung 6.16 zeigt den Aufteilung des IP-Cores in die zwei VHDL Entities *Top* und *PLBv46 NPI EWB Interface*.

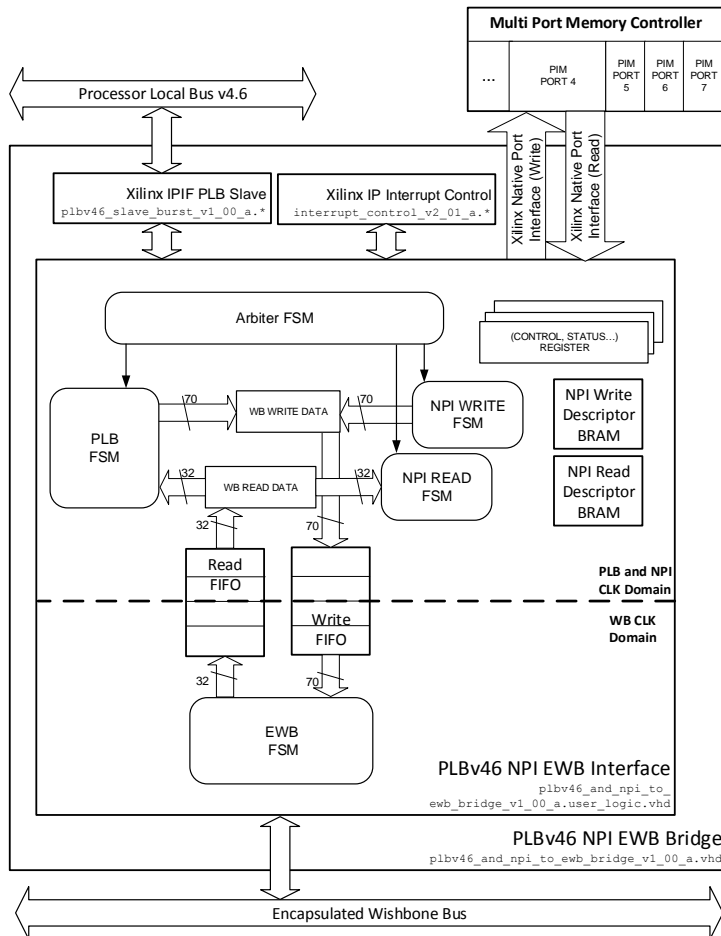


Abbildung 6.16: Übersicht über die PLB-NPI-EWB-Bridge (basierend auf [45, S. 201])

Top

Auf der Top Ebene bindet der IP-Core einen *Burst-fähigen PLBv4.6 Slave* und einen Interruptkontrolller ein. Die PLB-Signale und NPI-Kanäle werden mit dem integrierten *PLBv46 NPI EWB Interface* verbunden.

PLBv46 NPI EWB Interface

Das PLBv46 NPI EWB Interface implementiert die Ansteuerung des PLBs über den Zustandsautomaten *PLB FSM*. Die Kommunikation über die NPI-Kanäle erfolgt durch die Zustandsautomaten *NPI READ FSM* und *NPI WRITE FSM*. Da über diese beiden Schnittstellen (NPI und PLB) Daten transferiert werden können, steuert ein Arbitrer (*Arbiter FSM*) den Datenfluss. Der IP-Core verfügt über mehrere interne Register, die von der DPCU über den PLB Slave gesetzt bzw. gelesen werden können. Neben Kontroll- und Statusregistern, werden Register verwendet, um einen Deskriptor für einen Transfer zu erstellen. Ein vollständiger Deskriptor für einen NPI-Schreib- oder Lesetransfer wird anschließend in dedizierten BRAMs mit bis zu 256 Einträgen abgespeichert.

Zu transferierende Daten werden, unabhängig von der Quelle oder dem Zielort, in asynchronen FIFOs zwischengespeichert, sodass die unterschiedlichen Taktdomänen (PLB/NPI und WB) voneinander entkoppelt werden. Der Zustandsautomat *EWB FSM* realisiert die Ansteuerung des EWBs zur Datenübertragung.

Funktionen

Die Funktionen des IP-Cores sind detailliert in [45, S. 202] beschrieben und sind in den folgenden Abschnitten für die einzelnen Schnittstellen zusammengefasst.

Processor Local Bus Slave

Der PLBv46 Slave wird über das Schnittstellenprotokoll IPIF (*IP InterFace*) angesteuert. Die Implementierung unterstützt Einzel- und Burst-Zugriffe. Die letztgenannten Burst-Zugriffe werden auf einem PLB in Blöcken zu jeweils maximal 16 Transfers aufgeteilt. Die internen Register sind über den Slave durch einen dedizierten Adressbereich stets erreichbar.

Native Port Interface

Die Schnittstelle NPI wird für die Kommunikation mit dem MPMC verwendet und ermöglicht einen direkten Zugriff auf die MPMC-internen Schreib- und Lese-FIFOs (siehe Abschnitt 6.2.3). Die beiden NPI-Kanäle erlauben simultane Transfers, sodass intern zwei dedizierte Zustandsautomaten die Kommunikationsansteuerung realisieren.

Um einen kontinuierlichen Datentransfer zu ermöglichen, ist in der Bridge eine deskriptorbasierte NPI-Kommunikation, ähnlich zu dem Local Link Protokoll (siehe Abschnitt 6.2.2 oder 6.2.3), implementiert. Ein Deskriptor besteht aus einer Quell- und Zieladresse (32 bit) und der Größe (engl. size) des Transfers (4 bit). Die Größe kann im DRPM auf 16, 32 oder 64 Wörter gesetzt werden (siehe Abschnitt 6.2.3). Zur Maximierung der Gesamtdatenübertragungsrates innerhalb des Systems können neue Deskriptoren dynamisch, sogar während eines NPI-Transfers, hinzugefügt werden. Eine

stetige Aktualisierung der Deskriptoren wird daher angestrebt. Da die Deskriptoren über interne Register per PLB Slave definiert und anschließend in einem BRAM gespeichert werden, erfolgt die Aktualisierung idealer Weise parallel zu einer aktiven NPI-Übertragung.

Encapsulated Wishbone Bus

Das EWB-Protokoll wurde in Abschnitt 4.4.1 eingeführt. Abbildung 6.17 zeigt die wesentlichen Signale eines Schreib- und Lesevorgangs über PLB, EWB und WB. Da die Busse asymmetrisch zueinander sind, der PLB ist 64 bit und der EWB sowie WB 32 bit breit, müssen die Daten im Falle eines Schreibvorgangs (#1) aufgeteilt bzw. im Falle eines Lesevorgangs (#2) zusammengefügt werden. Den Einfluss des Zwischenspeicherns in FIFOs im Falle eines Schreibvorgangs wird in dem Signalverlauf durch die späte Bestätigung des Transfers (WB_ACK, #3) durch die, an den WB angeschlossene, Komponente deutlich gemacht. Ein ähnliches Verhalten lässt sich auch beim Lesevorgang erkennen (#4). Hier werden die Daten nach einer Leseanforderung ebenfalls mit einer Verzögerung von einem Takt bestätigt.

Auswertung

Die Auswertung der Kommunikationsbrücke zwischen dem statischen und dem dynamischen Bereich erfolgt anhand der benötigten FPGA-Ressourcen, der Datenübertragungsraten der MPMC-Anbindung über die NPI-Kanäle und der Leistung.

FPGA-Ressourcen

Die benötigten FPGA-Ressourcen der PLB-NPI-EWB-Bridge sind in Tabelle 6.11 dargestellt. Der Großteil der FPGA-Ressourcen wird in dem IP-Core für die Ansteuerung der NPI- und EWB-Kommunikation benötigt.

Tabelle 6.11: FPGA-Ressourcen der PLB-NPI-EWB-Bridge
(basierend auf [45, S. 211])

plbv46_npi_ewb_bridge	Slices	Register	LUTs	BRAMs	DSPs
PLBV46_NPI_EWB_Interface	913	1 108	1 826	4	0
INTERRUPT_CONTROLLER	9	10	15	0	0
PLBV46_SLAVE_BURST	227	230	295	0	0
Gesamt	1 149	1 348	2 136	4	0

Leistungsfähigkeit

Zur Untersuchung der Leistungsfähigkeit des IP-Cores wird die Kommunikationsstrecke zwischen dem DDR2-Speicher und den PR-Modulen analysiert. Für die Analyse der NPI-Kommunikation werden die Parameter Transfergröße und Burst-Größe variiert.

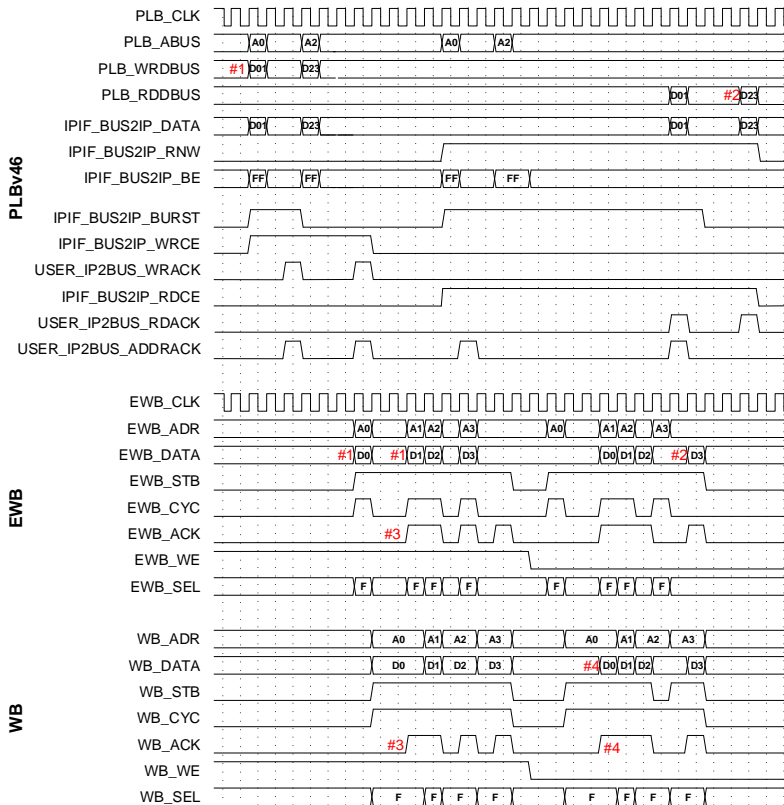


Abbildung 6.17: Signalverlauf eines Schreib- und Lesetransfers über PLB, EWB und WB (basierend auf [45, S. 205])

Die Abbildung 6.18 zeigt den Einfluss beider Parameter auf die Datenübertragungsrates für Lese- und Schreibzugriffe.

Tabelle 6.12 fasst die maximalen Datenübertragungsrates für Lese- und Schreibvorgänge zusammen und zeigt den deutlichen Einfluss des Parameters NPI-Burst-Größe. So erzielt die größtmögliche Burst-Größe eine um bis zu 47,3 % (Schreiben) bzw. 40,4 % (Lesen) höhere Transferrate im Vergleich mit der kleinstmöglichen NPI-Burst-Größe. Die maximalen Datenübertragungsrates erreichen dabei 84,1 % (Schreiben) bzw. 86,3 % (Lesen) der theoretisch möglichen Datenübertragungsrates von 400 MB/s.

Leistung

Die Leistung der Kommunikationsbrücke teilt sich auf die Takt-, Logik- und BRAM-Komponenten auf, welche insgesamt eine Leistung von 54 mW haben. Der Großteil

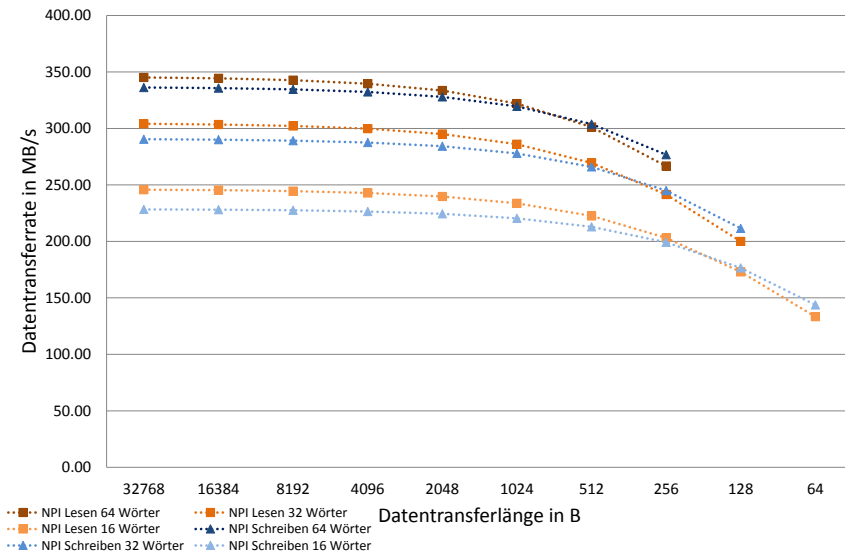


Abbildung 6.18: Datenübertragungsraten für NPI-Transfers

Tabelle 6.12: Maximale NPI-Datenübertragungsraten für verschiedene NPI-Burst-Größen (basierend auf [44, S. 58])

NPI-Burst-Größe	Datenübertragungsrate (in MB/s)	
	Lesen	Schreiben
16	245,7	228,3
32	304,1	290,5
64	345,1	336,3

der Leistung wird für die Takterzeugung (32 mW) aufgewendet. Die Leistung der Logikelemente beträgt 16 mW, die der BRAM-Komponenten 6 mW.

PR-Tile-Management

Die PR-Tile-Management-Komponente besteht primär aus einem Arbiter und Adressdekoder des Wishbone Busses. Zusätzlich werden weitere Funktionen zur Konfiguration der PR-Tiles integriert. Diese Komponente und ihre Funktionen sind in [45, S. 212 ff.] beschrieben und basiert auf der Implementierung aus [76, S. 76 ff.].

Aufbau

Der Aufbau der Komponente ist in Abbildung 6.19 dargestellt und deutet die zusätzlichen Funktionen der PR-Tile-Management-Komponente an. Die Anbindung erfolgt

dabei, analog zu den PR-Modulen, nach einer Umsetzung von EWB zu WB (siehe Abschnitt 6.2.4.1).

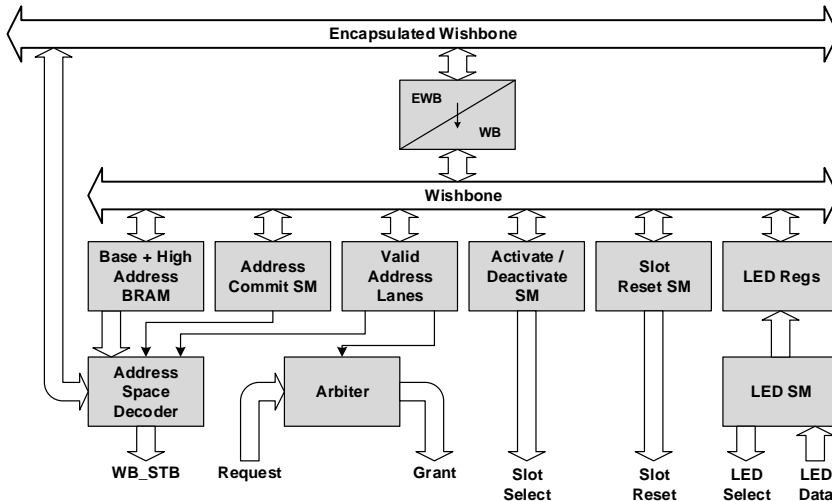


Abbildung 6.19: Übersicht der PR-Tile-Management-Komponenten (basierend auf [76, S. 76])

Funktionen

Die erste Funktion der PR-Tile-Management-Komponente ist ein dynamischer Adressdekoder, welcher in der DRPM-Version bis zu 32 (Master-fähige) PR-Tiles unterstützt. Durch den generischen Aufbau der Komponente kann die Anzahl durch wenige Änderungen weiter erhöht werden.

Der Adressdekoder (*Address Space Decoder*) wird durch einen BRAM-Block mit individuellen PR-Tile-Adressen und -Adressräumen (*Base + High Address BRAM*) erweitert. Die Adresskonfiguration eines PR-Tiles kann somit dynamisch angepasst werden. Darüber hinaus können PR-Tiles vom Bus abgekoppelt (*Activate / Deactivate SM*) und zurückgesetzt (*Slot Reset SM*) werden. Die Entkopplung verhindert einen ungewollten Zugriff auf den Bus während einer DPR. Nach erfolgreicher DPR, wird ein dediziertes Reset an das platzierte PR-Modul gesendet. Beide aufgeführten Funktionen werden von Xilinx explizit empfohlen (siehe Abschnitt 3.3.4 oder [211, S. 109 f.]).

Ein Arbiter kontrolliert den Zugriff auf den Wishbone Bus. Damit keine ungewollte Arbitrierung durch ungetriebene Signale (z. B. während einer DPR) erfolgt, ist diese an die Adresskodierung gekoppelt. Ist eine PR-Tile deaktiviert, findet keine Arbitrierung statt. Durch die sogenannte LED-Funktionalität (*LED Regs* und *LED SM*) können von jeder PR-Tile vier dedizierte Signale zum statischen Bereich übertragen werden, ohne expliziten Buszugriff. Die Signale werden im Round-Robin-Verfahren ausgelesen, sodass

automatisch alle 32 Takte neue Daten von einem Modul übertragen werden. Die LED-Funktionalität kann z. B. zur Übermittlung des aktuellen Status verwendet werden.

DPR-Kommunikationsinfrastruktur

Ein flexibles DPR-System, das mehrere Rekonfigurationsarten unterstützt, benötigt, wie in Abschnitt 3.5 ausgeführt, eine homogene Kommunikationsinfrastruktur zur Verbindung der statischen und dynamischen Systemkomponenten. Die Homogenität muss dabei für beide Signalarten (gemeinsam verwendet und dediziert) eingehalten werden.

Die DPR-Kommunikationsinfrastruktur ist als eingebettetes Makro realisiert (siehe Abschnitt 3.4), gehört somit zum statischen Bereich und stellt in jeder PR-Tile fest positionierte und verdrahtete Ein- und Ausgänge (engl. ports) bereit, mit denen sich die PR-Module verbinden. Die Erstellung dieser homogenen Kommunikationsinfrastruktur erfolgt durch die bereits vorgestellte DHHarMa-Werkzeugkette (siehe Kapitel 4). Zur Kommunikation mit dem dynamischen Bereich des DRPMs wird, wie in Abschnitt 6.2.4.1 beschrieben, das Protokoll *Encapsulated Wishbone Bus* verwendet.

Varianten der Kommunikationsinfrastruktur

Die PR-Region des PR-FPGAs besteht, wie in Abbildung 6.20 dargestellt, aus nahezu fünf vollständigen Taktregionen, sodass etwa ein Viertel der Fläche des Virtex-4 FX100 FPGAs für PR-Module verfügbar ist.

Die verfügbare dynamische Fläche kann durch unterschiedliche Partitionierungen aufgeteilt und anschließend durch eine entsprechende DPR-Kommunikationsinfrastruktur angeschlossen werden. In Tabelle 6.13 ist eine Übersicht der FPGA-Ressourcen in Slices für verschiedene Partitionierungen für beide EWB-Protokolltypen dargestellt. Die zweidimensionalen Varianten basieren auf den, bereits in Abschnitt 4.6.3 vorgestellten, Partitionierungen. Die Wahl der Partitionierung hat, wie dargestellt, einen entscheidenden Einfluss auf die verfügbaren FPGA-Ressourcen.

Tabelle 6.13: FPGA-Ressourcenübersicht bei Verwendung verschiedener 1D und 2D DPR-Kommunikationsinfrastrukturen für beide EWB-Protokolltypen

Protokoll	Partitionierung	In PR-Tile verfügbar in Slices	Ressourcen in Slices	Komm-Inf. in %	Für PR-Modul verfügbar in Slices	in %
SEWB	4x1	1 728	118	6,8	1 610	93,2
	2x2	480	173	36,0	307	64,0
	3x3	512	173	33,8	339	66,2
	5x2	480	173	36,0	307	64,0
MEWB	4x1	1 728	203	11,7	1 525	88,3
	2x2	480	207	43,1	273	56,9
	3x3	512	207	40,4	305	59,6
	5x2	480	207	43,1	273	56,9

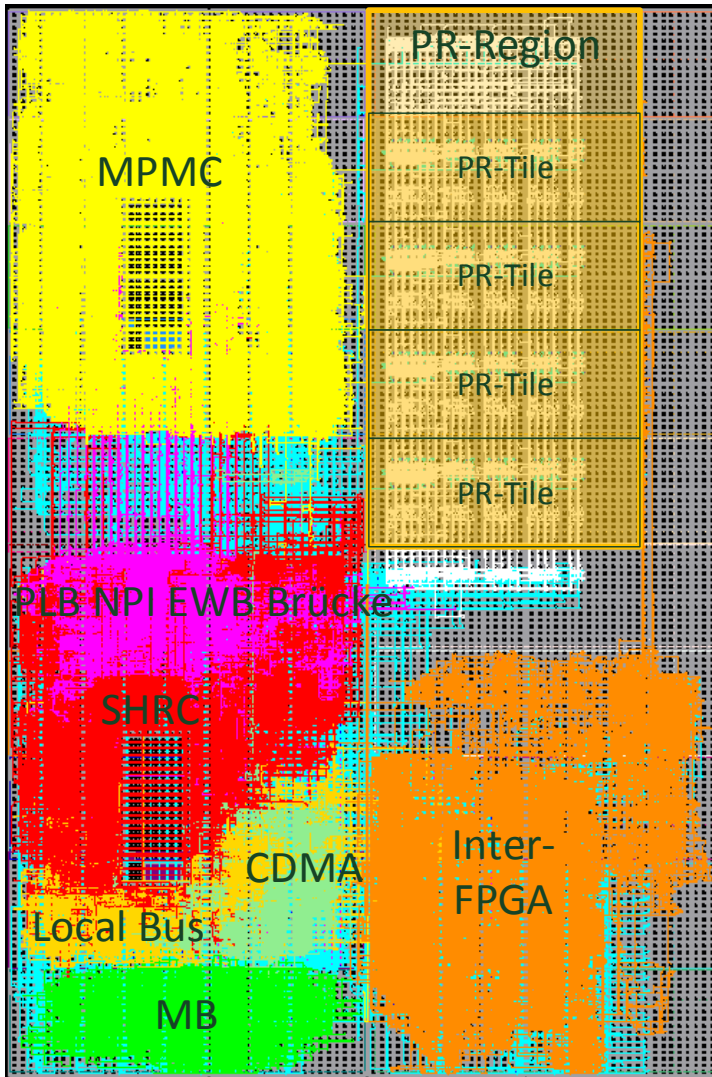


Abbildung 6.20: Übersicht der PR-FPGA-Komponenten

Während in Abbildung 6.20 eine 4x1 SEWB DPR-Kommunikationsinfrastruktur dargestellt ist, sind in Abbildung 6.21 zwei weitere Realisierungen abgebildet. In Abbildung 6.21 (a) ist eine Partitionierung der PR-Region in 10 Kacheln inklusive einer Master-Slave-fähige EWB-Kommunikationsinfrastruktur visualisiert. Abbildung 6.21 (b)

stellt eine 3x3 Partitionierung mit einer Slave-fähigen EWB-Kommunikationsinfrastruktur dar.

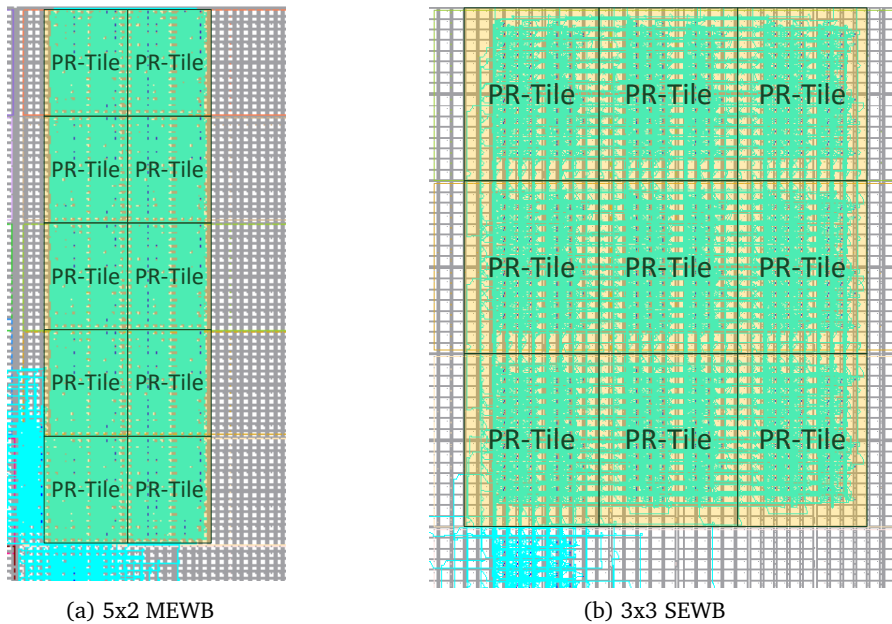


Abbildung 6.21: Alternative Partitionierung der PR-Region inklusive DPR-Kommunikationsinfrastruktur

Kreuzende Signalleitungen

Wie in Abschnitt 3.5 beschrieben, führen statische Signalleitungen, welche die PR-Region kreuzen, zu einer Inhomogenität im DPR-System, sodass eine Umplatzierung von PR-Modulen verhindert wird. Kreuzende Signale treten dabei z. B. durch vorgegebene IOB-Belegungen auf, welche die Partitionierung des FPGAs einschränken. Diese genannte Problematik ergab sich in nahezu jedem Synthesedurchlauf des PR-FPGAs aufgrund der Anbindung der Local Bus und Inter FPGA Schnittstelle. In Abbildung 6.22 sind zwei konkrete Beispiele der Problematik kreuzender Signalleitungen dargestellt. In dem INDRA 2.0 DPR-System wird diese Problematik durch einen Nachbearbeitungsschritt mittels des *PSReRouters* gelöst (siehe Abschnitt 3.5.7).

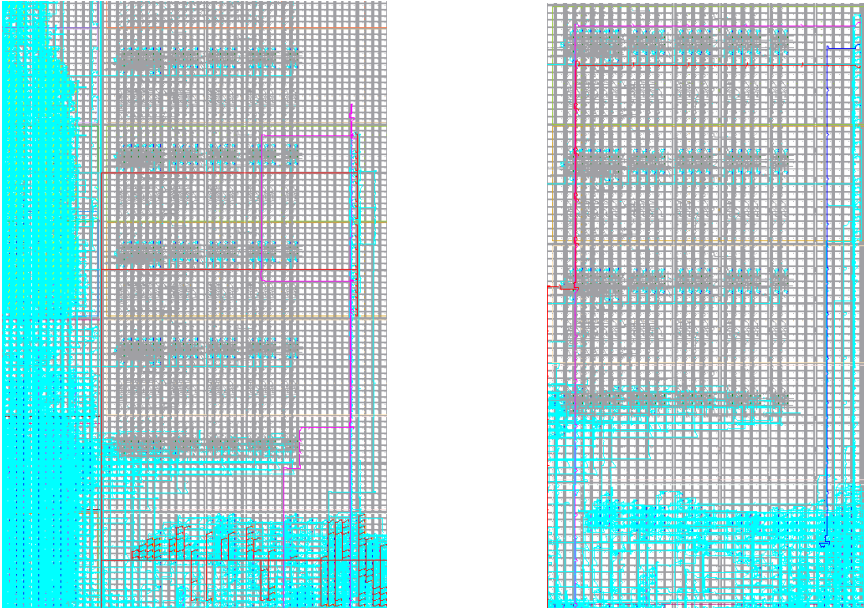


Abbildung 6.22: Problematik von kreuzenden statischen Signalleitungen

6.3 PR-Module

Im Rahmen des *DRPM* Projekts wurden drei Anwendungstypen untersucht: *Fast Fourier Transform* (FFT), *Finite Impulse Response* (FIR) und Bildfilter. Die beiden erstgenannten Anwendungstypen wurden mit dem *Xilinx System Generator* [220] realisiert und bezüglich der FPGA-Ressourcen, der Leistungsfähigkeit und der Leistung bzw. Energie analysiert. Wie in Abschnitt 3.1 beschrieben, werden durch Parametervariationen optimale Implementierungsvarianten für eine Verarbeitungsaufgabe erstellt. Der Datenfluss innerhalb des DRPMs ist in Abbildung 6.23 dargestellt. Als Datenquellen und Datensenken werden primär die hervorgehobenen Schnittstellen, SpaceWire oder Ext-Comm-FPGA sowie Wizard Link bzw. SpaceFibre, des DB-SPACE verwendet. Die Daten werden anschließend über die Inter FPGA Verbindung zunächst zu einem benachbarten Verarbeitungsmodul und anschließend über den PLB in den DDR2-Speicher transferiert. Die *DPCU* (siehe Abschnitt 6.1.3) initialisiert die weiteren Schritte zur Datenverarbeitung. So wird zunächst, basierend auf dem aktuellen Ablaufplan, ein PR-Modul mit der entsprechenden Anwendung zur Datenverarbeitung mittels des SHRCs in der PR-Tile platziert. Anschließend werden die Datentransfers zwischen dem DDR2-Speicher und dem PR-Modul initialisiert. Aufgrund der höheren Übertragungsrates erfolgen die Datentransfers in der Regel über die NPI-Kanäle.

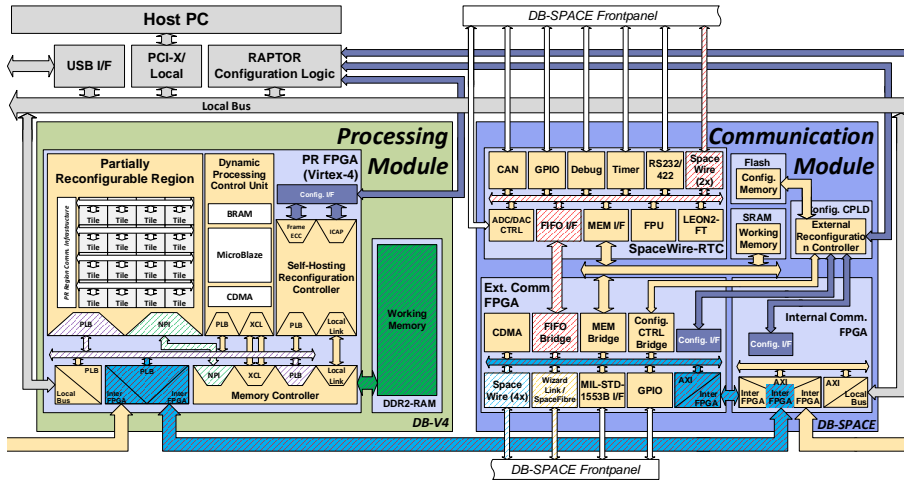


Abbildung 6.23: Datenfluss innerhalb des DRPMs

Für die Implementierung der PR-Module wird eine eindimensionale Partitionierung der PR-Tile mit einer SEWB-basierten DPR-Kommunikationsinfrastruktur und vier PR-Tiles vom gleichen Typ betrachtet. Die FPGA-Ressourcen einer PR-Tile sind in Tabelle 6.14 detailliert. Hierbei sind die Anzahl verfügbarer Ressourcen, die Ressourcen der PR-Anschlussstelle der Kommunikationsinfrastruktur und die Anzahl der verbleibenden FPGA-Ressourcen für die Implementierung einer Anwendung (PR-Modul) dokumentiert. Es ist zu beachten, dass die Anzahl verfügbarer Slice-interner Komponenten (FFs und LUTs) einzig durch die Anzahl verfügbarer Slices bestimmt wird. Unvollständig verwendete Slice-Ressourcen sind durch die Verwendung von Hard-Makros für die Kommunikationsinfrastruktur für weitere Implementierungen geblockt. Prozentual stehen somit 93,2 % der Slice-Ressourcen und 100 % der DSP- und BRAM-Ressourcen einer PR-Tile für die Implementierung eines PR-Moduls zur Verfügung.

DRPM-Doppelpufferspeicher Zur Entkopplung der Anwendungen von der Kommunikation und zur Unterstützung von Blocktransfers wurden Doppelpufferspeicher für die erwähnten Anwendungstypen realisiert. Jeweils am Ein- und Ausgang werden die Daten in acht Speicherblockabschnitte mit je 128 Einträgen zwischengespeichert. Die Adressierung der Abschnitte erfolgt intern und autonom. Die DPCU überprüft über ein internes Statusregister des Doppelpufferspeichers den aktuellen Füllstand des Ein- und Ausgangsspeichers und initialisiert entsprechende Datentransfers. Da der Ausgabewert einer FFT-Berechnung doppelt so breit ist wie der Eingangswert, wurden zwei Typen des Doppelspeichers realisiert: Ein symmetrischer Doppelpufferspeicher mit 4 kB Ein- und Ausgangsspeicher und ein asymmetrischer Doppelpufferspeicher mit 4 kB Ein- und

Tabelle 6.14: FPGA-Ressourcenübersicht bei Verwendung der eindimensionalen Kommunikationsinfrastruktur

	FPGA-Ressourcen				
	FFs	LUTs	Slices	DSP48s	RAMB16s
PR-Tile	3 456	3 456	1 728	8	16
PR-Anschlussstelle	172	210	118	0	0
Auslastung in %	5,0	6,1	6,8	0,0	0,0
Freie Ressourcen	3 220	3 220	1 610	8	16
in %	93,2	93,2	93,2	100,0	100,0

8 kB Ausgangsspeicher. Die in Tabelle 6.15 dargestellten FPGA-Ressourcen enthalten neben den beiden Doppelpufferspeichertypen die Logik zur Ansteuerung der Filter und Abhandlung der EWB-Kommunikation. Die komplexere Ansteuerung des FIR-Filters führt in dem (kleineren) symmetrischen Doppelpufferspeicher zu einem höheren Ressourcenbedarf.

Tabelle 6.15: FPGA-Ressourcen der Doppelpuffertypen der DRPM-Anwendungen

Puffertyp	FPGA-Ressourcen				
	FFs	LUTs	Slices	DSP48s	RAMB16s
Sym. Puffer (4 kB)	336	195	260	0	4
Auslastung in %	11,0	6,4	17,1	0,0	25,0
Asym. Puffer (4/8 kB)	203	207	211	0	6
Auslastung in %	6,7	6,8	13,9	0,0	37,5

6.3.1 Fast Fourier Transform

Die schnelle Fourier-Transformation (engl. Fast Fourier Transform, FFT) beschreibt einen Algorithmus zur effizienten Berechnung der diskreten Fourier-Transformation (engl. Discrete Fourier Transform, DFT) [187]. Mit der DFT kann ein digitales Signal in seine Frequenzanteile zerlegt und anschließend analysiert werden.

Mit dem *LogiCORE IP Fast Fourier Transform* von Xilinx wird die DFT durch den Cooley-Tukey Algorithmus realisiert und auf FPGA-Blöcke abgebildet [207]. Durch die Verwendung dieses Algorithmus müssen die Abtastpunkte einer Zweierpotenz entsprechen. Der IP-Core unterstützt Abtastpunkte im Bereich von 8 (bzw. Radix-4-Architektur: 64) bis 65536 und Datenbreiten von 8 bis 34 bit. Mathematisch wird die DFT $X(k)$, $k = 0, \dots, N - 1$ durch eine Sequenz $x(n)$, $n = 0, \dots, N - 1$ wie folgt

definiert [207]:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{jnk2\pi}{N}} \quad k = 0, \dots, N-1 \text{ und } j = \sqrt{-1} \quad (6.1)$$

Weiterhin werden vier unterschiedliche Architekturen unterstützt, welche Verarbeitungseinheiten, sogenannte Radix-2 *butterfly* oder Radix-4 *dragonfly*, verwenden:

1. *Pipelined, Streaming I/O*: Kontinuierliche Datenverarbeitung durch mehrere Radix-2 *butterfly* Verarbeitungseinheiten und zusätzlichen Speicher
2. *Radix-4 Burst I/O*: Sukzessives Laden und Verarbeiten von Daten mit einer Radix-4 *dragonfly* Verarbeitungseinheit
3. *Radix-2 Burst I/O*: Sukzessives Laden und Verarbeiten von Daten mit einer Radix-2 *butterfly* Verarbeitungseinheit
4. *Radix-2 Lite Burst I/O*: Sukzessives Laden und Verarbeiten von Daten mit einer, um FPGA-Ressourcen begrenzten, Version einer Radix-2 *butterfly* Verarbeitungseinheit

Pipelined, Streaming I/O

Abbildung 6.24 stellt die *Pipelined, Streaming I/O* Architektur mit mehreren Radix-2 *butterfly* Verarbeitungseinheiten sowie zusätzlichen Speicherelementen dar. Die Architektur erlaubt somit, auf Kosten zusätzlicher FPGA-Ressourcen, das Laden neuer Eingangsdaten parallel zur Verarbeitung aktueller Daten, sodass der höchstmögliche Datendurchsatz erzielt werden kann.

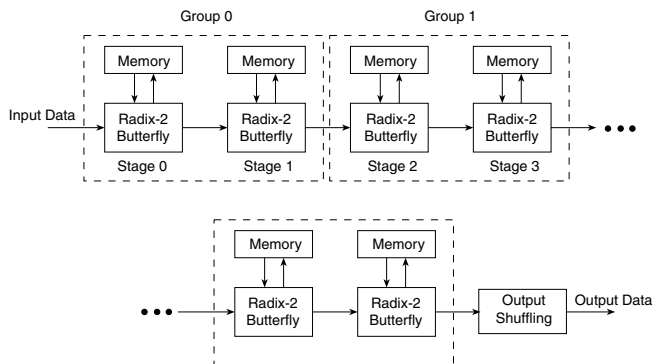


Abbildung 6.24: Aufbau der Pipelined, Streaming I/O FFT-Architektur [207, S. 7]

Radix-4 Burst I/O

In allen Radix-basierten Architekturen erfolgt das Laden und Verarbeiten der Eingangsdaten sukzessive. Abbildung 6.25 visualisiert den allgemeinen Aufbau der Radix-

Architektur. Vor der Verarbeitung muss ein vollständiger Satz von Eingangsdaten in die Speicherelemente geladen werden. Die verarbeiteten Daten werden anschließend aus der Verarbeitungseinheit ausgelesen und in dieselben Speicherelemente geschrieben. Bevor neue Eingangsdaten in die Speicherelemente geschrieben werden, müssen daher die verarbeiteten Daten ausgelesen werden. Die Verarbeitung durch den Cooley-Tukey Algorithmus führt zu einer bitweisen Invertierung der verarbeiteten Daten. Durch zusätzliche FPGA-Ressourcen kann die natürliche Anordnung der Daten wiederhergestellt werden.

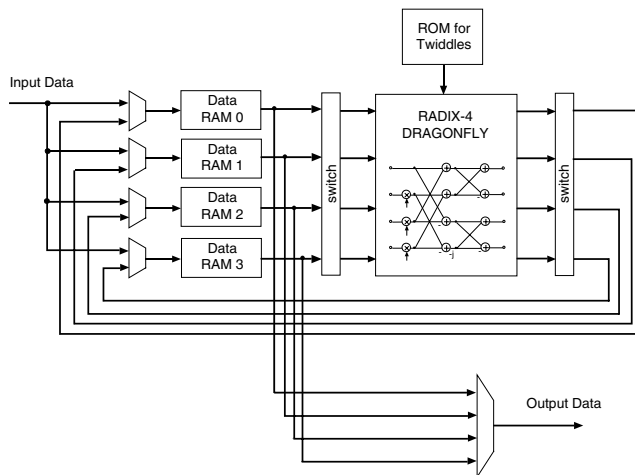


Abbildung 6.25: Aufbau der Radix-4-Architektur [207, S. 8]

Radix-2 Burst I/O und Radix-2 Lite Burst I/O

Der Aufbau einer Radix-2-Architektur ähnelt dem der Radix-4-Architektur. Anstelle einer *dragonfly* Verarbeitungseinheit wird eine *butterfly* Verarbeitungseinheit sowie weniger Speicherelemente verwendet. In Abbildung 6.26 sind die beiden Radix-2-Architekturen dargestellt. Die Ressourcenbegrenzungen in der Lite Version sind durch Anmerkungen und Kreuze hervorgehoben.

Spezifizierte FFT Eigenschaften

Als Eingangsdatum $x(n)$ wurde für den Real- und Imaginärteil jeweils eine 16 bit breite, vorzeichenbehaftete Festkommazahl mit einem Nachkommaanteil von 13 bit gewählt. Der FFT-IP-Core erlaubt die Einstellung der Verarbeitungsfrequenz. Da eine Änderung dieser zu keinen Ressourceneinsparungen führte, wurde die Verarbeitungsfrequenz fest auf 100 MHz gesetzt. Der Einfluss der Abtastpunkte wurde durch drei Varianten untersucht ($N=1\ 024$, $2\ 048$ und $4\ 096$).

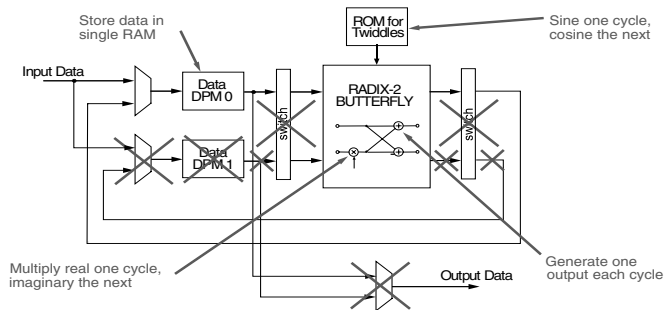


Abbildung 6.26: Aufbau der Radix-2-Architektur [207, S. 9]

Auswertung der FFT-Implementierungen

Die Auswertung der FFT-Implementierungen erfolgt jeweils für alle vier vorgestellten FFT-Architekturen und jeweils separat für eine Abtastpunktvariante. Die folgenden Diagramme stellen die FPGA-Ressourcen durch die (relative) Auslastung einer PR-Tile, die Verarbeitungszeit und die benötigte Leistung dar. Eine zusammenfassende Übersicht dieser Werte folgt in Tabelle 6.16. Die textuell aufgeführten Werte basieren auf der detaillierteren Tabelle A.25 im Anhang C.3. Die Verarbeitungszeit wird durch zwei Zeitwerte festgehalten: Die reine Verarbeitungszeit der Verarbeitungseinheit und die vollständige Verarbeitungszeit inklusive der Datentransfers der Ein- und Ausgangsdaten. Beide Werte sind in Tabelle A.25 dokumentiert. In den Diagrammen ist jeweils die vollständige Verarbeitungszeit dargestellt. Die Pipelined-Architektur erlaubt als einzige Architektur ein paralleles Laden und Verarbeiten von Daten, sodass die dargestellte Verarbeitungszeit der vollständigen Verarbeitungszeit entspricht. Darüber hinaus existieren für die Pipelined-Architektur aufgrund der Pipeline zwei Verarbeitungszeiten. Die initiale Verarbeitungszeit enthält zusätzlich die Zeit zum Befüllen der Pipeline. Kann die Pipeline anschließend kontinuierlich mit Daten versorgt werden, ergibt sich eine zweite Verarbeitungszeit, welche lediglich von der Datenmenge abhängt. Da die FFT-Implementierungen im DRPM über den DRPM-Doppelpufferspeicher an eine Wishbone Businfrastruktur angeschlossen ist, kann dies jedoch nicht gewährleistet werden. Aus diesem Grund entspricht die, für die Pipeline-Architektur dargestellte, vollständige Verarbeitungszeit der initialen Verarbeitungszeit. In den anderen Architekturen muss zur Ermittlung der vollständigen Verarbeitungszeit das Laden und Entladen der Daten berücksichtigt werden. Die Zeit für die Datentransfers skaliert hierbei mit der Datenmenge/Abtastpunkten.

1024 Abtastpunkte In Abbildung 6.27 sind die Ergebnisse der prozentualen Auslastung einer PR-Tile bezüglich der kritischen FPGA-Ressource, welche die Anzahl der benötigten PR-Tiles bestimmt, der Leistung und der Verarbeitungszeit der verschiede-

nen FFT-Architekturen dargestellt. Die primäre kritische Ressource bezüglich der FPGA-Ressourcen ist in drei der vier Implementierungen der DSP-Block. In der Radix-2 Lite Architektur ist aufgrund der geteilt verwendeten DSP-Blöcke die kritische Ressource der BRAM-Block, sodass für FFT-Implementierungen eine sekundäre kritische FPGA-Ressource existiert. Die Pipelined- und Radix-4-Architektur belegen aufgrund der primären kritischen Ressource drei PR-Tiles. In der Pipelined-Architektur übersteigt zusätzlich die Anzahl benötigter Slices (234 %) die Ressourcen zweier PR-Tiles. Ein Vergleich der FPGA-Ressourcen der Radix-4 mit der Pipelined-Architektur führt zu 36,9 % weniger Slices (43,1 % weniger Registern und 39 % weniger LUTs), 18,2 % weniger DSPs, jedoch 70,0 % mehr BRAM-Blöcken. Für die beiden Radix-2-Implementierungen reichen hingegen die FPGA-Ressourcen einer PR-Tile aus. Die Radix-2-Implementierung benötigt hierbei 52,4 % weniger Slices, 66,7 % weniger DSPs und 35,3 % weniger BRAMs als die Radix-4-Implementierung. Ein Vergleich der Lite Version mit der Radix-2-Implementierung führt zu weiteren Ressourceneinsparungen bezüglich der Slices (23,8 %) und DSPs (33,3 %). Dieser Unterschied im Resourcebedarf spiegelt sich in der Leistungsbetrachtung wieder. So liegt etwa die Leistung der Pipeline-Architektur mit 220 mW um 33,9 % über der Leistung der Radix-4-Architektur (165 mW). Die Betrachtung der Verarbeitungszeit fällt invers zu den vorherigen Betrachtungen aus. Die Radix-4-Implementierung benötigt im Vergleich mit der Pipelined-Architektur mit 34,3 μs zu 16,4 μs mehr als die doppelte Ausführungszeit für eine vollständige Berechnung. Ein annähernd gleiches Verhältnis ergibt sich ebenfalls bei dem Vergleich der Radix-4- mit der Radix-2-Implementierung (73,4 μs). Die Verarbeitungszeit der Lite Variante ist mit 123,1 μs nochmals um 67,8 % höher als die der Radix-2-Implementierung.

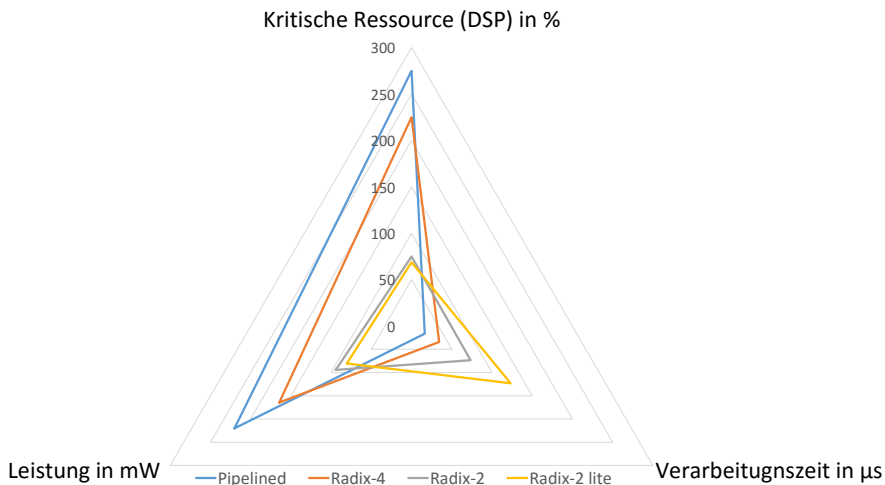


Abbildung 6.27: Analyse der untersuchten FFT-Architekturen mit 1024 Abtastpunkten

2048 Abtastpunkte Die Ergebnisse der FFT-Implementierungen mit 2 048 Abtastpunkten ist in Abbildung 6.28 dargestellt. Im Vergleich zu den Implementierungen mit 1 024 Abtastpunkten ergeben sich in der FPGA-Ressourcenauswertung Änderungen in dem Verhältnis der Pipelined- zur Radix-4-Architektur. So werden mehr DSP-Blöcke in der Pipelined-Architektur verwendet, sodass die Radix-4-Architektur nun 35,7 % weniger Ressourcen benötigt. Die Verdopplung der Abtastpunkte führt, außer in der Radix-4-Architektur, zu einem Anstieg der BRAM-Blöcke, sodass sich das Verhältnis dieser FPGA-Ressource zu Gunsten der Radix-4-Architektur verändert. Die primäre kritische Ressource in der untersuchten Partitionierung bleibt der DSP-Block. In der Radix-2-Architektur belegt nun, analog zur Lite Version, der Anteil der BRAM-Blöcke die höchste PR-Tile-Auslastung. Die Anzahl der benötigten PR-Tiles für die einzelnen Architekturen bleibt dennoch identisch zu denen in der Auswertung der Implementierungen mit 1 024 Abtastpunkten. Beim Vergleich der Leistungen der FFT-Implementierungen ergeben sich nahezu identische relative Unterschiede. Bei der Betrachtung der Ausführungszeit erhöht sich das Verhältnis zwischen der Pipelined- und der Radix-4-Architektur auf 128,5 % (N=1 024: 109,0 %).

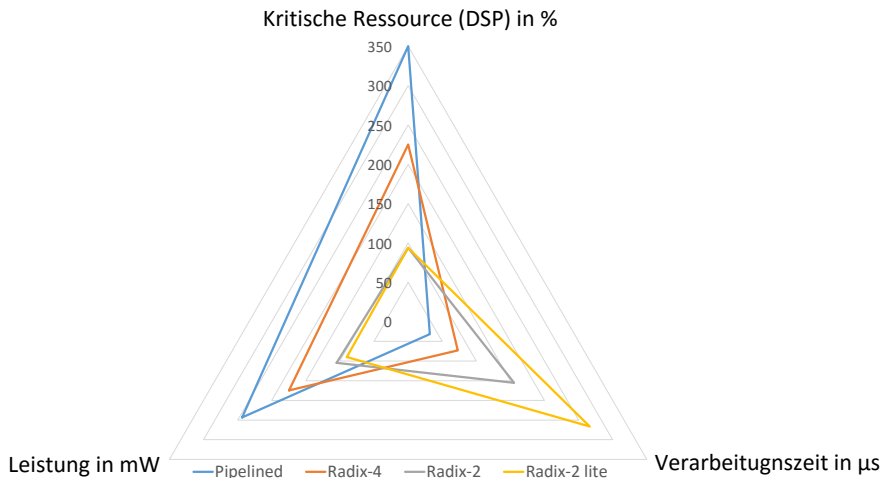


Abbildung 6.28: Analyse der untersuchten FFT-Architekturen mit 2 048 Abtastpunkten

4 096 Abtastpunkte Wie in Abbildung 6.29 dargestellt, führt einzig die Auslastung der BRAM-Blöcke in den Radix-2-Implementierungen dazu, dass zwei PR-Tiles verwendet werden müssen. Darüber hinaus werden in der Pipelined-Architektur nun mehr BRAM-Blöcke als in der Radix-4-Architektur verwendet. Eine weitere Abweichung ergibt sich in dem Verhältnis der Ausführungszeit zwischen der Radix-4- und Radix-2-Architektur, welche mit 144,6 µs zu 329,8 µs auf 128,1 % ansteigt (N=2 048: 113,3 %).

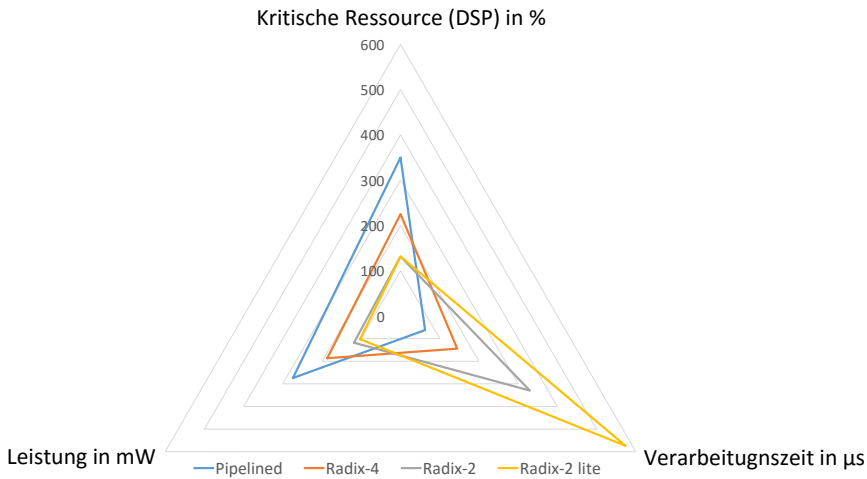


Abbildung 6.29: Analyse der untersuchten FFT-Architekturen mit 4096 Abtastpunkten

Zusammenfassung Tabelle 6.16 fasst die beschriebene Auswertung zusammen. Als Bewertungsmaße sind für die FPGA-Ressourcen die Auslastung und Anzahl der PR-Tiles, für die Leistungsfähigkeit die Verarbeitungszeit und für die Leistungsbewertung Leistung und Energie angegeben.

Von den vier gegenübergestellten Architekturen kristallisieren sich für die untersuchte Systempartitionierung drei relevante Varianten heraus. Für eine schnelle Verarbeitung bietet sich die Pipelined-Architektur an, welche die geringste Verarbeitungszeit auf Kosten benötigter FPGA-Ressourcen garantiert. Betrachtet man die Energie für die Datenverarbeitung, führt diese Architektur stets zu den geringsten Kosten. Dem gegenüber steht mit der Radix-2-Architektur eine auf FPGA-Ressourcen optimierte Variante, jedoch mit einer deutlich längeren Verarbeitungszeit (Faktor 4,5 bis 5,3). Ein Kompromiss stellt die Radix-4-Architektur dar. Die Verarbeitungszeit ist um Faktor 2,1 bis 2,3 größer als in der Pipelined-Architektur. Der Vergleich mit der Radix-2-Architektur führt zu dem gleichen Verhältnis zu Gunsten der Radix-4-Architektur. Die Lite Version bringt in der untersuchten Partitionierung keine entscheidenden Vorteile in der FPGA-Ressourcenauslastung mit sich und weist eine deutlich längere Verarbeitungszeit als die reine Radix-2-Architektur (Faktor 1,7) auf.

Das Diagramm in Abbildung 6.30 stellt die Auslastung der PR-Tile durch die kritische Ressource der Verarbeitungszeit grafisch gegenüber. Der Durchmesser eines Punkts deutet die benötigte Energie zur Datenverarbeitung an. Hierbei ist ein eindeutiger Trend bei steigender Anzahl an Abtastpunkten für die untersuchten Metriken erkennbar.

Bei den FFT-Implementierungen ist insbesondere der Datentransfer von Bedeutung. Da stets nur vollständige Datensätze verarbeitet werden können, müssen entsprechende

Tabelle 6.16: Ergebnisse der Auswertung der Verarbeitungszeit, der Ressourcen-Auslastung und der Leistung/Energie der FFT-Implementierungen

Abtastpunkte	Architektur	Auslastung PR-Tile Ø in %	PR-Tiles krit. Ress. in %	PR-Tiles in #	Verarbeitungszeit in µs	Leistung in mW	Energie in µJ
1024	Pipelined	183	275	3	16,4	220,2	3,6
	Radix-4	136	225	3	34,3	164,5	5,6
	Radix-2	63	75	1	73,4	94,3	6,9
	Radix-2 lite	48	69	1	123,1	80,5	9,9
2048	Pipelined	220	350	4	31,9	243,4	7,8
	Radix-4	144	225	3	72,9	174,7	12,7
	Radix-2	70	94	1	155,5	104,8	16,3
	Radix-2 lite	55	94	1	266,5	89,9	23,9
4096	Pipelined	237	350	4	62,7	274,4	17,2
	Radix-4	155	225	3	144,6	186,9	27,0
	Radix-2	79	131	2	329,8	118,6	39,1
	Radix-2 lite	63	131	2	573,7	103,0	59,1

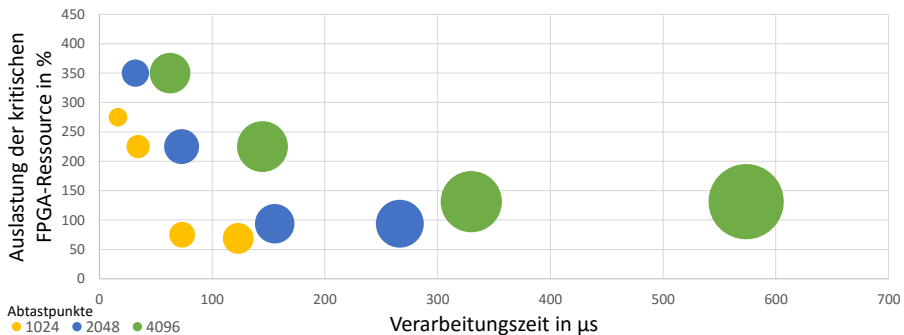


Abbildung 6.30: Gegenüberstellung der Auslastung einer PR-Tile, der Verarbeitungszeit und der Energie der untersuchten FFT-Implementierungen

NPI-Datentransfers initialisiert werden, bevor mit einer Verarbeitung der Daten begonnen werden kann. Wie bereits erwähnt, muss des Weiteren beachtet werden, dass die Ausgangsdaten doppelt so breit sind wie die Eingangsdaten. Ohne die Verwendung eines asymmetrischen Doppelpufferspeichers kann es daher bei FFT-Implementierungen mit einer großen Anzahl an Abtastpunkten zu Einbrüchen im Datendurchsatz kommen. Bezüglich der FPGA-Ressourcen ist die primäre kritische Ressource der DSP-Block.

Bei steigender Anzahl an Abtastpunkten wird der BRAM-Block zu einer sekundären kritischen Ressource.

6.3.2 Finite Impulse Response

Die wesentliche Eigenschaft von FIR-Filtern ist, dass sie eine Impulsantwort mit endlicher Länge garantieren, sodass sie immer stabil bleiben und niemals zu selbstständigen Schwingungen angeregt werden können [182]. Die Filterantwort kann mittels der diskreten Summenfaltung ermittelt werden, wobei N die Anzahl der Filterkoeffizienten beschreibt:

$$y(k) = \sum_{n=0}^{N-1} a(n)x(k-n) \quad k = 0, 1, \dots \quad (6.2)$$

Der *LogiCORE IP FIR Compiler* [208] von Xilinx ermöglicht die Erstellung von FIR-Filtern mit bis zu 256 Koeffizientensätzen, welche während der Laufzeit ausgetauscht werden können. Die Datenbreite der Eingangs- und Koeffizientendatensätze liegt bei maximal 49 bit. Durch die Verwendung von DSP-Blöcken, können sogenannte *Multiply-Accumulate* (MAC) Architekturen realisiert werden, um die in Formel 6.2 dargestellte Filterantwort zu berechnen. Eine Realisierung mit einer MAC-Einheit ist in Abbildung 6.31 dargestellt.

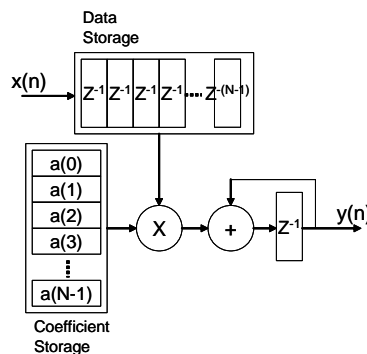


Abbildung 6.31: Single Multiply-Accumulate Architektur [208, S. 16]

Durch die Verwendung mehrerer, auf dem FPGA verfügbarer, DSP-Einheiten werden sogenannte *Multi-MAC*-Architekturen aufgebaut. Der *FIR Compiler* von Xilinx unterstützt hierbei zwei unterschiedliche MAC-Architekturen:

- *Systolic-MAC* (SMAC, S.16): Architektur mit kaskadierten DSP-Blöcken, die Koeffizienten-Symmetrie ausnutzen kann
- *Transpose-MAC* (TMAC, S.17): Architektur, welche die transponierte Normalform implementiert

Systolic-MAC

Die SMAC-Architektur führt zu flächenoptimierten Filterimplementierungen mit einer allgemein hohen Leistungsfähigkeit. Abbildung 6.32 stellt die SMAC-Architektur als Pipeline-Implementierung dar.

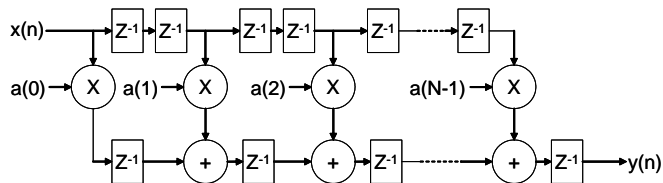


Abbildung 6.32: Systolic-MAC als Pipeline-Implementierung [208, S. 17]

Im Falle einer Koeffizienten-Symmetrie können in der SMAC-Architektur Ressourcen eingespart werden (siehe Abbildung 6.33). Wie angedeutet, werden die Daten und Koeffizienten nacheinander in die DSP-Einheiten geladen, miteinander verrechnet und anschließend abgespeichert. Hierdurch können insbesondere DSP- aber auch Speicherelemente eingespart werden.

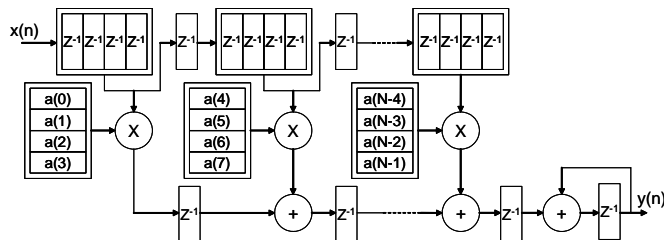


Abbildung 6.33: Systolic-MAC als Multi-MAC-Implementierung [208, S. 17]

Transpose-MAC

Die TMAC-Architektur ermöglicht im Allgemeinen eine Implementierung mit geringer Latenz. Im Vergleich zur SMAC-Architektur können in der TMAC-Architektur weitere Ressourcen (insbesondere Speicherelemente) eingespart werden. Die Implementierung als transponierte Direktform ist in Abbildung 6.34, die Multi-MAC-Implementierung in Abbildung 6.35 dargestellt.

Spezifizierte FIR-Filtereigenschaften

Konkret wurde ein Tiefpassfilter für Frequenzen im Bereich von 0 kHz und 2 kHz mit 16 und 64 Taps realisiert. Ein Tap beschreibt hierbei die Anzahl der verwendeten Speicherelemente (in den Abbildungen durch Z dargestellt) und wird zur Realisierung zeitlicher

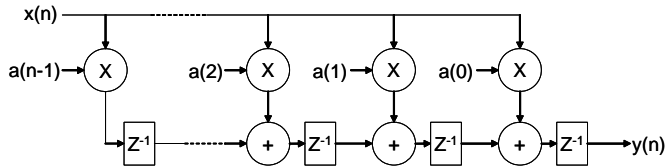


Abbildung 6.34: Transpose-MAC als Direktform [208, S. 18]

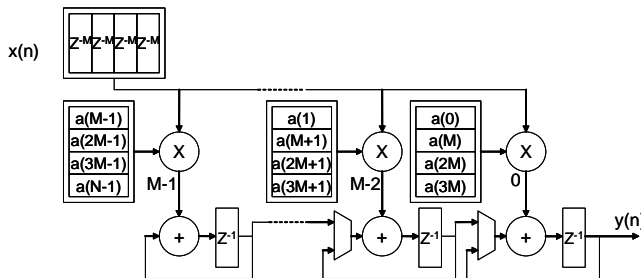


Abbildung 6.35: Transpose-MAC als Multi-MAC-Implementierung [208, S. 18]

Verzögerungen in der Übertragungsfunktion eingesetzt. Als Filtereingangsdatum wurde eine 16 bit breite, vorzeichenbehaftete Festkommazahl mit einem Nachkommaanteil von 13 bit gewählt. Das Filterausgangsdatum resultiert in einer Bitbreite von 32 bit mit einem 28 bit breiten Nachkommaanteil. Die Samplerate wurde auf 10 kHz festgesetzt. Die Sampleperiode, das Verhältnis von Eingangs- zu Ausgangsdatenrate, wurde hingegen zu Gunsten der FPGA-Ressourcen variiert. Als Implementierungsart wurde stets die Multi-MAC-Variante gewählt.

Auswertung FIR-Filter-Implementierungen

In Abbildung 6.36 sind die Auswirkungen auf die FIR-Filter-Implementierungen der Tap-Größe 16 auf die Bewertungsmaße FPGA-Ressourcen, Leistung und Datendurchsatz dargestellt. Die Darstellung basiert auf den Daten der Tabelle 6.17. Eine, bezüglich der FPGA-Ressourcen, detailliertere Tabelle A.26 ist im Anhang C.3 enthalten. In dieser Tabelle ist bei den Varianten der Tap-Größe 16 zu erkennen, dass die TMAC-Architektur eine deutlich geringere Anzahl an Registern und LUTs aufweist, jedoch wesentlich mehr DSP-Blöcke benötigt. Die kritische Ressource in den PR-Tiles ist, analog zu den FFT-Implementierungen, der DSP-Block. In allen Fällen bestimmt diese kritische Ressource die Anzahl an benötigten PR-Tiles für die Implementierung. Ein Vergleich der FPGA-Ressourcen beider Architekturen bei einer Tap-Größe von 16 und einer Sampleperiode von 2 (Datendurchsatz von 50 MSamples/s) zeigt, dass die FPGA-

Ressourcen einer PR-Tile lediglich bei Verwendung der SMAC-Architektur ausreichend sind. Die Auswirkung auf die Leistung ist beim Vergleich der Architekturen gering.

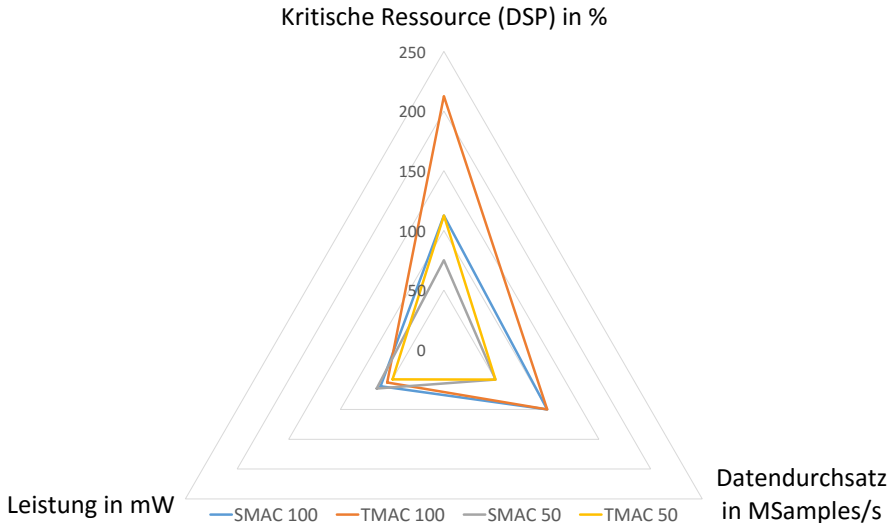


Abbildung 6.36: Analyse der FIR-Filter-Implementierungen mit Tap-Größe 16

Aus diesem Grund wurde für die Varianten der TAP-Größe 64 nur die SMAC-Architektur untersucht, welche in Implementierungen für ein bis vier PR-Tiles resultieren (siehe Abbildung 6.37).

Zusammenfassung In Tabelle 6.17 sind die beschriebenen Ergebnisse der verschiedenen FIR-Filter-Implementierungen für die untersuchte Systempartitionierung tabellarisch dargestellt. Neben dem Datendurchsatz ist die Auslastung der PR-Tile jeweils durch zwei Werte dargestellt. In der dritten Spalte ist die durchschnittliche Auslastung aller FPGA-Ressourcen und in der vierten Spalte die Auslastung durch die kritische Ressource dokumentiert. Der letztgenannte Wert bestimmt die Anzahl der benötigten PR-Tiles. Die Verwendung der TMAC-Architektur bietet bei den gewählten Filtereigenschaften aufgrund der kritischen Ressource (DSP-Block) keine Vorteile gegenüber der SMAC-Architektur, sodass die SMAC-Architektur in allen Fällen zu bevorzugen ist. Die Auswirkung auf die Leistung ist in der letzten Spalte dokumentiert.

6.3.3 Bildfilter

Für den letzten Anwendungstyp, wurden simple Bildfilter (Invertierung und Rotanteilfilter) zur Live-Demonstration des DRPMs implementiert. Die Anwendungen der DPR wurde hierbei zum einen durch Filterwechsel zur Systemlaufzeit und zum anderen durch (Einbit-) Fehlerinjektion in dem verarbeitenden Bildfilter sowie der anschließen-

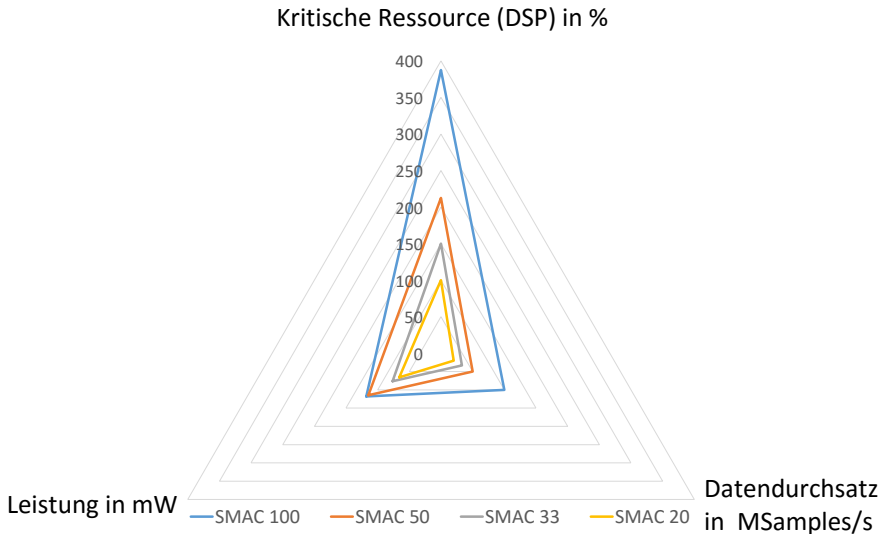


Abbildung 6.37: Analyse der FIR-Filter-Implementierungen mit Tap-Größe 64

Tabelle 6.17: Ergebnisse der Auswertung des Datendurchsatzes, der Ressourcenauslastung der PR-Tile und der Anzahl benötigter PR-Tiles der FIR-Filter-Implementierungen

Tap	Architektur	Datendurchsatz in MSamples/s	Auslastung PR-Tile		PR-Tiles in #	Leistung in mW
			Ø in %	Krit. Res. in %		
16	SMAC	100	45	112	2	61,2
	TMAC	100	56	212	3	54,8
	SMAC	50	41	75	1	65,0
	TMAC	50	37	112	2	49,6
64	SMAC	100	132	387	4	118,0
	SMAC	50	95	212	3	114,7
	SMAC	33	61	150	2	76,5
	SMAC	20	46	100	1	65,9

den Selbstheilung durch den SHRC veranschaulicht. Zur Demonstration wurde die in Abbildung 6.38 dargestellte Benutzeroberfläche entwickelt, mit der das DRPM vor und zur Laufzeit konfiguriert werden kann.

Die Vorkonfiguration beinhaltet die Auswahl der Eingangsdaten (Bilder, Video oder Live-Kameradaten), der Dateneingangs- und Datenausgangsschnittstelle, des Filters und ob eine Fehlerinjektion stattfinden soll. Bei aktivierter Fehlerinjektion wird auto-

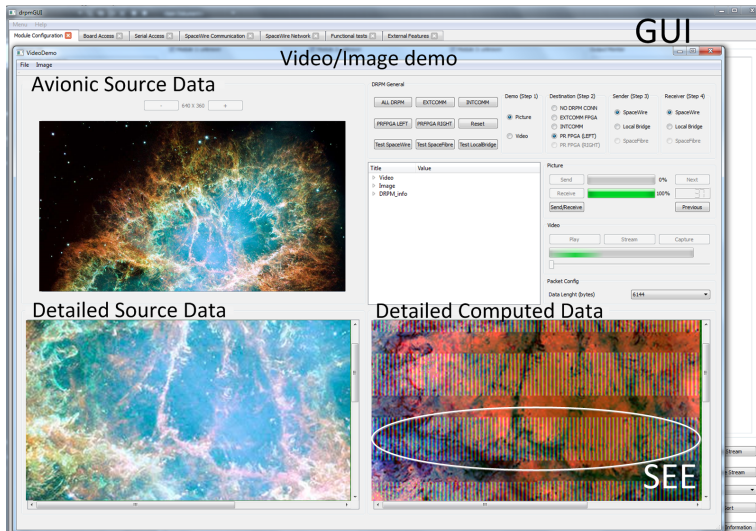


Abbildung 6.38: Benutzeroberfläche zur Konfiguration des DRPMs

matisch das Readback Scrubbing aktiviert. Hierbei kann zwischen drei vorgegebenen *Readback Scrubbing Schedules* (siehe Abschnitt 6.2.2) gewählt werden: Gesamter PR-FPGA, PR-Modul und FPGA-Spalte in welcher der Einbitfehler injiziert wurde. Entsprechend des *Readback Scrubbing Schedules* ergeben sich unterschiedliche Zeitspannen der Fehlererkennung und -korrektur. Zur Laufzeit kann anschließend der Filtertyp, die Fehlerinjektion sowie das *Readback Scrubbing Schedule* verändert werden.

Eine Live-Demonstration des DRPMs erfolgte unter anderem auf der Konferenz *Design, Automation and Test in Europe* (DATE 2014, [104]) und dem *Space FPGA Users Workshop* (SEFUW 2014).

6.4 Verfahren zur Erhöhung der Fehlertoleranz

Durch die Verwendung der SRAM-Speichertechnologie sind SRAM-basierte FPGAs insbesondere gegenüber *Single-Event Effects* (SEE) anfällig (siehe Abschnitt 5.7.3). Die stetige Miniaturisierung der Strukturgröße in der CMOS-Technologie hat hierbei allgemein einen großen Einfluss auf die Anfälligkeit dieser Speichertechnologie. Durch die gleichzeitige Skalierung der Versorgungsspannungen sowie der steigenden Anzahl an Transistoren steigt auch die Anfälligkeit gegenüber TID [145].

Der DRPM Demonstrator – insbesondere die Systemarchitektur des PR-FPGAs – bietet sich als Evaluationsplattform für Verfahren zur Erhöhung der Fehlertoleranz an. Für die Behandlung von SEEs wurde der SHRC des DRPMs um zwei Scrubbing Verfahren erwei-

tert. Details hierzu werden in dem ersten Abschnitt beschrieben. Die DPR-Funktionalität wird darüber hinaus zur Erkennung von hardwareseitigen, permanenten Fehlern (*Hard Errors*) zur Laufzeit eingesetzt. So wurden kleine autonome Testschaltungen erstellt, welche durch DPR platziert und das Testergebnis durch einen Lesevorgang ermitteln werden. Die Testauswertung erfolgt durch die DPCU auf dem PR-FPGA, sodass der aktuelle Zustand des FPGAs zur Laufzeit festgestellt und somit die Fehlertoleranz des Gesamtsystems weiter erhöht werden kann.

6.4.1 Behandlung von Single-Event Effects

Scrubbing-Verfahren sind, wie in Abschnitt 5.6.4 beschrieben, für die Fehlererkennung und -korrektur des SRAM-Konfigurationsspeichers von essenzieller Bedeutung. Bevor die realisierten Scrubbing-Verfahren vorgestellt werden, erfolgt zunächst eine Berechnung der Fehlerrate für den PR-FPGA.

Fehlerrate

Basierend auf den Ergebnissen des Rosetta Experiments (siehe Abschnitt 5.9.1), kann die Fehlerrate für den PR-FPGA ermittelt werden. Der Konfigurationsspeicher des Virtex-4 FX100 umfasst 33,0 Mbit und beinhaltet 6,8 Mbit an BRAM. Bei Verwendung der ermittelten FIT-Raten des Rosetta Experiments für die Virtex-4-Architektur ergibt sich eine Anfälligkeit von 263 FIT/Mbit (CRAM) bzw. 484 FIT/Mbit (BRAM). Unter Verwendung der approximierenden Formel 5.17 ergibt sich die absolute FIT-Rate des FPGAs von:

$$FIT_{Total} = 33,0 \text{ Mbit} \cdot 263 \text{ FIT/Mbit} + 6,8 \text{ Mbit} \cdot 484 \text{ FIT/Mbit} = 11\,970$$

Mit Hilfe der Formel 5.18 berechnet sich die MTBF zu 9,54 Jahren. Angemerkt sei hierbei der feste Ortsbezug (New York, USA) durch die verwendeten FIT-Raten.

Blind Scrubbing

Ein FPGA-internes *Blind Scrubbing* kann periodisch durch die DPCU initialisiert werden. Hierbei sind jedoch folgende, bereits in Abschnitt 5.7.3 genannten, Nachteile zu beachten: Fehlerhafter Konfigurationsspeicherinhalt durch einen beschädigten Controller und / oder Konfigurationsdatenspeicherort sowie unnötige Schreibvorgänge des Konfigurationsspeichers. Wie, außerdem in Abschnitt 5.7.3 bereits festgehalten, ist ein externes, partielles Blind Scrubbing einem FPGA internem Blind Scrubbing vorzuziehen. Dem erstgenannten Nachteil wird im DRPM durch die Realisierung des externen Rekonfigurationscontrollers in einem CPLD und einem angeschlossenen Flash-Speicher entgegengewirkt (siehe Abschnitt 6.2.1). Anzumerken ist hierbei, dass diese Komponenten in dem DRPM-Prototyp durch COTS-Komponenten realisiert sind, jedoch durch äquivalente, strahlungsgehärtete Komponenten ausgetauscht werden können.

Readback Scrubbing

Wie in Abschnitt 5.7.3 beschrieben, empfiehlt Xilinx aufgrund der genannten Nachteile die Verwendung eines Readback Scrubbings. Die Implementierung des *Configuration*

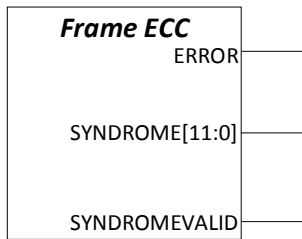


Abbildung 6.39: Frame ECC Portbelegung

Tabelle 6.18: Auswertung des Frame ECC Syndromwerts (SW)

SW[11]	SW[10:0]	Resultat
0	0	Kein Fehler
0	$\neq 0$	Zweibitfehler
1	-	Einbitfehler

Frame basierten *Readback Scrubbings* wurde unter Beachtung der Empfehlungen von Xilinx [35, 36] sowie [54, 55] realisiert.

Der Hard-IP-Block *Frame ECC* ist die wesentliche Komponente zur Detektion von Ein- und Zweibitfehlern im Konfigurationsspeicher. Abbildung 6.39 zeigt den *Frame ECC* Block für Xilinx FPGAs der Virtex-4-Familie. Die Funktionsweise des Blocks wird für die Virtex-4-Familie in [228, S. 75 f.] beschrieben. Der IP-Core berechnet zur Realisierung der SECDED-Funktionalität während jedes Lesevorgangs eines (vollständigen) *Configuration Frames* einen sogenannten Syndromwert (*SYNDROME*), basierend auf einem Hamming-Code. Die Gültigkeit des berechneten Syndromwerts wird über das Statussignal *SYNDROMEVALID* kenntlich gemacht. Das Statussignal *ERROR* signalisiert, dass ein Einbit- oder Zweibitfehler detektiert wurde. Ein gültiger Syndromwert (*SW*) enthält die Information über den Typ des Fehlers (Ein- oder Zweibit), sowie indirekte Informationen zur Lokalisierung eines Einbitfehlers. Die Auswertung des Syndromwerts kann zu drei Resultaten führen, welche in Tabelle 6.18 aufgeführt sind. Das höchstwertige Bit (engl. **M**ost **S**ignificant **B**it, **MSB**) des Syndroms gibt somit Aufschluss über den Fehlertyp, jedoch nur unter zusätzlicher Auswertung der niederwertigen Bits. Im Falle eines Einbitfehlers, können die unteren elf Syndrombits verwendet werden, um eine Berechnung der konkreten (Bit-) Position innerhalb des *Configuration Frames* durchzuführen. Eine direkte Verwendung dieser Bits ist nicht möglich, da diese auf den Adressraum 704 bis 2047 verweisen. Um die Syndrombits in den Adressraum des *Configuration Frames* zu übertragen, muss, je nach Syndromwert, ein fester Offset von dem Syndromwert subtrahiert werden:

- $SW[10:0] < 1024$: Offset 704
- $SW[10:0] \geq 1024$: Offset 736

Die Subtraktion der Offsets kann vereinfacht werden, indem 22 oder 23 von $SW[10:5]$ abgezogen wird, sodass Xilinx die Verwendung folgender Formel empfiehlt [228, S. 76]:

$$bit_index = S[10:5] - 6'd22 - S[10], S[5:0] \quad (6.3)$$

Zusätzlich existieren Sonderfälle im Falle eines detektierten Einbitfehlers. Ist $SW[10:0]$ gleich Null oder ein Wert zur Basis Zwei, liegt ein Einbitfehler in einem der zwölf

Hamming-Paritätsbits vor. Die Zuordnung der konkreten Bitposition innerhalb des *Configuration Frames* zum Syndromwert ist in Tabelle 6.19 dargestellt. Die Erkennung und Korrektur eines Einbitfehlers auf dem PR-FPGA zur Laufzeit ist im Anhang C.1 in Abbildung A.11 anhand eines Ausschnitts der Xilinx Software ChipScope dargestellt.

Tabelle 6.19: Syndromwertauswertung eines Einbitfehlers in einem Paritätsbit (basierend auf [228, S. 75])

SW[10:0]	Bitposition
100000000001	640
100000000010	641
1000000000100	642
1000000001000	643
1000000010000	644
1000000100000	645
1000010000000	646
1000100000000	647
1001000000000	648
1010000000000	649
1100000000000	650
1000000000000	651

Auf das potentielle Fehlverhalten der *Frame ECC* Komponente bei Mehrbitfehlern wurde bereits in Abschnitt 5.7.3 hingewiesen. Bei einer ungeraden Anzahl an Mehrbitfehlern führt die Berechnung der Bitposition zu einem fehlerhaften Wert, sodass durch die vermeintliche Korrektur ein weiterer Fehler injiziert wird. Laut Berg et al. kann es weiterhin dazu kommen, dass eine gerade Anzahl an Fehlern größer zwei von der *Frame ECC* Komponente nicht erkannt wird [23]. Konkret wurde dies für Vier- und Achtbitfehler durch Fehlerinjektion in einem Virtex-4 LX25 FPGA nachgewiesen.

Bitfehlerinformationsdaten

Im Falle eines, durch die *Readback Scrubbing Unit* (RSU) detektierten, Einbit- oder Zweibitfehlers, werden nähere Informationen zum Fehler (Typ des Fehlers, Configuration Frame address, die berechnete Bitposition und der Zeitpunkt des Auftretens) von der RSU an den VCM übertragen und dort in einer FIFO gespeichert. Diese Fehlerinformationsdaten werden zweimal (invertiert und normal) übertragen, um Fehler auf dem Übertragungsweg zwischen der RSU und dem VCM erkennen zu können. Der Inhalt der FIFO ist wiederum über ein dediziertes Register auslesbar. Durch die Implementierung dieses Fehlerstatistikmechanismus kann das System adaptiv auf Fehler reagieren. So kann bei einer steigenden Fehlerrate die Zuverlässigkeit des Systems durch Verwendung von robusteren, z. B. durch Redundanzverfahren erweiterten, (statischen und

dynamischen) Konfigurationen erhöht werden. Diese Vorgehensweise ist beispielsweise im Heinrich Hertz Satelliten (siehe Abschnitt 5.9.2) implementiert. So können, je nach Strahlungssituation, Verarbeitungsmodul mit entsprechenden Abschwächungsverfahren verwendet werden.

Statusinformationen

Neben einem Statusregister mit detaillierten Statusinformationen des SHRCs, informieren Interrupts die DPCU über Änderungen am internen Zustand des SHRCs. Das Interrupt *VCM-Done* kündigt an, dass der aktuelle Schreib- oder Lesevorgang beendet wurde. Ein *Heartbeat* von der RSU signalisiert, wie von Xilinx in [35, S. 21 f.] empfohlen, den aktuellen Status des IP-Cores. Das Auftreten eines Ein- oder Zweibitfehlers während des Scrubbings wird über das Interrupt *Readback Scrubbing Error* signalisiert. Das Interrupt *Irreparable Error* informiert die DPCU, dass ein SEFI-Fehler oder ein Fehler in der VCM-RSU-Kommunikation aufgetreten ist. Ein Fehler dieser Kategorie führt zwangsläufig zu einer vollständigen Rekonfiguration des PR-FPGAs und ermöglicht der DPCU zuvor wichtige Daten mit dem SpaceWire-RTC auszutauschen.

Single-Event Functional Interrupt

Die rechtzeitige Erkennung eines SEFIs zur Laufzeit ist essentiell, um die Zuverlässigkeit des Gesamtsystems gewährleisten zu können. Laut Xilinx ([4, S. 8 ff.], [35, S. 9 ff.] und [36, S. 19 ff.]) wurden in Strahlungstests folgende SEFIs in Virtex-4-FPGAs festgestellt:

- Power-On-Reset (POR): Verhalten ähnlich eines FPGA-Neustarts
- Global Signals: Verlust einzelner FPGA-Funktionen oder der Konfigurationslogik durch Fehler in einer Vielzahl weiterer Konfigurationskontrollsignale
- SMAP (ICAP) und JCFG: Verlust der Konfigurierbarkeit über eine Konfigurationsschnittstelle (SelectMAP, ICAP oder JTAG)
- FAR: Verlust des Lesens oder Schreibens des FARs oder fehlerhafte Aktivierung der Autoinkrementierfunktion der *Configuration Frame* Adresse
- Scrub/High-Current : Stetiger Anstieg der Stromaufnahme, der zur Beschädigung des FPGAs führen kann

Beim Auftreten eines SEFIs der beiden erstgenannten SEFI-Typen empfiehlt Xilinx eine sofortige Reparatur durch eine vollständige Rekonfiguration. Bei den darauffolgenden SEFI-Typen, SMAP, ICAP und JCFG sowie FAR, kann die vollständige Rekonfiguration hingegen im Anschluss an die Ausführung einer Anwendung erfolgen.

Ein High-Current-SEFI ist dabei Resultat von Upsets innerhalb der Logik des Konfigurationsprozessors, wie bereits in Abschnitt 5.7.3 beschrieben. Das Auftreten eines Scrub-SEFIs [4, 35], welcher auch High-Current-SEFI [36, S. 19 ff.] genannt wird, ist sehr unwahrscheinlich. Laut letztgenannter Quelle ist diese um bis zu eine Größenordnung geringer als alle anderen SEFI-Typen, wenn ein konstantes Scrubbing erfolgt. Dennoch sind die potenziellen Auswirkungen des SEFIs nicht zu vernachlässigen, da die Stromaufnahme zu einer Beschädigung des FPGAs führen kann. Eine Korrektur durch Scrubbing erfolgt daher, wie von Xilinx empfohlen, nur wenn durch ein Readback

ein Fehler detektiert wurde und nur für den betroffenen Configuration Frame, um den Einfluss dieses Fehlers weiter zu minimieren.

Der SHRC implementiert weitere Tests zur Detektion von SEFIs. Dabei werden von Xilinx in [35, S. 10 ff.] und [36, S. 10 ff.] empfohlene Schrittfolgen durchgeführt, wie beispielsweise das Schreiben und Lesen des FAR-Registers oder das Überprüfen des Status- und Kontrollregisters zur Erkennung von internen Fehlern des Konfigurations-Prozessors.

Readback Scrubbing Scheduling

Durch die Erstellung von *Readback Scrubbing Scheduling Tasks* kann für einzelne Regionen eine eigene Scrubbing-Rate festgelegt werden, sodass beispielsweise kritische Regionen mit einer höheren Scrubbing-Rate überwacht werden können. Dieses Verfahren entspricht dem, in Abschnitt 5.7.3 vorgestellten, *Scrubbing Profile*. Analog zu den Fehlerinformationsdaten, werden die Daten eines Readback Scheduling Tasks invertiert und normal vom VCM zur RSU übertragen, um Fehler auf dem Übertragungsweg erkennen zu können. Ein Readback Scheduling Task ähnelt einem Deskriptor und besteht aus folgenden Teilen:

- Adresse des *Configuration Frames*
- Länge (in Spalten innerhalb einer clock region)
- Anzahl der *Minor Frames* (Bestimmung des Blocktyps, siehe Abschnitt 3.3.1)
- Sprungadresse zum nächsten Scheduling Task

Für die Erstellung eines Readback Scrubbing Scheduling Tasks ist ein Wissen über den FPGA-Aufbau unerlässlich. Die Granularität liegt, wie gerade ausgeführt, auf Blocktyp-Ebene. Tabelle 6.20 stellt eine Übersicht über den Ressourcenbedarf der wesentlichen Komponenten des PR-FPGAs und daraus resultierende Scrubbing-Eigenschaften dieser bereit. Konkret wird die Dimensionierung der Bereiche in Spalten und Zeilen (Höhe einer Taktregion) sowie der Anzahl an Configuration Frames und der daraus resultierenden Bitanzahl angegeben. Hieraus lassen sich die Ausführungszeit des Scrubbings sowie die Scrubbing-Rate berechnen.

Zur Veranschaulichung der dargestellten Spaltenanzahlen der einzelnen Blocktypen sei auf Abbildung 3.10 auf Seite 45 verwiesen. Die genaue Anzahl an Configuration Frames der relevanten Blocktypen ist in der letzten Zeile aufgeführt (siehe Abschnitt 3.3.1). Die weiteren Blocktyp 0 Blöcke (MGT- und CLK) werden jedoch in der Berechnung des gesamten PR-FPGAs berücksichtigt. Bei den BRAM-Blöcken wurde lediglich der Blocktyp 1 (*BRAM Int*) einbezogen, welcher die Verdrahtung der Switch Matrizen enthält. Die weiteren Blocktypen beziehen sich auf den dynamischen BRAM-Inhalt und können somit nicht durch Scrubbing geschützt werden (siehe Abschnitt 5.7.3).

Die Dimensionierung der wesentlichen Komponenten des PR-FPGAs ist bereits in Abbildung 6.20 erkennbar. Der MPMC stellt mit 3 716 Configuration Frames die größte statische Komponente dar. Eine Überprüfung des Konfigurationsspeichers für diese Komponente benötigt 1,5 ms und kann theoretisch 656 mal pro Sekunde durchgeführt werden. Die Bereiche der Komponenten überlappen sich teilweise und wurden

Tabelle 6.20: Übersicht über den Ressourcenbedarf der wesentlichen Komponenten des PR-FPGAs und die daraus resultierenden Scrubbing-Eigenschaften

Komponente	Zu überprüfender FPGA-Bereich						Ausführung Scrubbing		Scrubbing-Rate in #/s
	CLB	Spalten			Zeilen	Summe		in ms	
		IOB	DSP	BRAM		in CF	in bit		
MPMC	34	2	1	5	4	3 716	4 875 392	1,524	656
SHRC	34	2	1	5	3	2 787	3 656 544	1,143	875
Local Bus Brücke	34	2	1	5	3	2 787	3 656 544	1,143	875
PLB NPI EWB Brücke	34	2	1	5	3	2 787	3 656 544	1,143	875
MB	34	2	1	5	1	929	1 218 848	0,381	2 625
DPCU	34	2	1	5	2	1 858	2 437 696	0,762	1 313
Inter-FPGA Brücke	34	1	1	5	4	3 596	4 717 952	1,474	678
PR-Region	28	0	1	4	5	3 600	4 723 200	1,476	678
Partitionierung									
4x1	28	0	1	4	1	717	940 704	0,294	3 402
PR-Tile 2x2, 5x2	8	0	0	1	1	196	257 152	0,080	12 444
3x3	9	0	0	1	1	218	286 016	0,089	11 188
PR-FPGA	68	3	2	10	10	18 710	24 547 520	7,671	130
Anzahl CF pro Blocktyp	22	30	21	20					

großzügig (zur Einhaltung von Zeitbedingungen) gewählt. So ergibt sich etwa eine gemeinsame FPGA-Fläche für den SHRC, die Local Bus und die PLB-NPI-EWB-Brücke. Das Readback Scrubbing für diesen Bereich dauert etwa 1,1 ms. Die MicroBlaze CPU (MB) ist in der unteren linken Taktregion des FPGAs platziert. Die gesamte DPCU belegt hingegen zwei Taktregionen. Beide Werte sind separat aufgeführt, da in einem Schedule die MB CPU in der Regel öfter überprüft wird als die gesamte DPCU. Die Inter-FPGA Schnittstelle ist die einzige statische Komponente auf der rechten FPGA-Seite und kann in 1,4 ms auf Ein- und Zweibitfehler überprüft werden. Für den dynamischen Bereich wurde die Gesamtfläche sowie die PR-Tiles der einzelnen, in Abschnitt 6.2.4 vorgestellten, Partitionierungen aufgeführt. Die vollständige Fläche kann in etwa 1,5 ms untersucht werden. Werden nur einzelne PR-Tiles verwendet oder wird eine besonders wichtige Anwendung in einer oder mehreren PR-Tiles ausgeführt, können diese in einer deutlich geringeren Zeit überprüft werden bzw. die Scrubbing-Rate für diese durch ein Schedule erhöht werden. Sollen nur einzelne Spalten auf Fehler überprüft werden, kann dies für eine CLB-Spalte in 90 μ s, eine IOB-Spalte in 123 μ s und eine MGT-Spalte in 82 μ s geschehen. Die Überprüfung des gesamten PR-FPGAs benötigt ca. 7,7 ms und könnte entsprechend 130 mal pro Sekunde durchgeführt werden.

Nach einer initialen Erstellung eines konkreten Readback Schedules durch die DPCU wird dieses an den IP-Core übertragen und intern in einem BRAM gespeichert. Anschließend wird das Readback Scrubbing autonom durchgeführt. Durch die Aufteilung in bis zu 256 Tasks, können komplexe Schedules erstellt werden, welche zusätzlich dynamisch, z. B. an die Strahlungssituation, angepasst werden können. Es werden daher die Vorteile einer Software- und Hardware-Lösung kombiniert. Die Verwendung

von variablen Scrubbing-Raten führt weiterhin dazu, dass die zusätzlich benötigte Leistungsaufnahme durch das Scrubbing auf ein Minimum reduziert werden kann (siehe Abschnitt 5.7.3).

In Tabelle 6.21 sind zusammenfassend die charakteristischen Parameter für die Scrubbing-Funktionalität des SHRCs aufgeführt. Neben der Scrubbing-Rate und der Datenübertragungsrate, ist die Korrekturzeit eines defekten *Configuration Frames*, die Ausführungszeit des SEFI-Tests und die Zeit für einen Wechsel des Scrubbing-Tasks dargestellt.

Tabelle 6.21: Charakteristische Parameter für die Scrubbing-Funktionalität des SHRCs (basierend auf [44, S. 87])

Max. Scrubbing-Rate	2,44 MZyklen/s
Datenübertragungsrate	400 MB/s
SEC in Configuration Frame	1 250 ns
SEFI-Test	390 ns
Wechsel Scrubbing-Task	110 ns

Vergleich von Scrubber-Implementierungen

In [55] und [112] erfolgen Vergleiche von Virtex-4 Scrubber-Implementierungen. Tabelle 6.22 fasst die wesentlichen Charakteristika zusammen und erweitert diese um die Werte der, im Rahmen dieser Dissertation entstandenen, *Readback Scrubbing Unit* (RSU).

Tabelle 6.22: Vergleich von Virtex-4 Scrubber-Implementierungen (basierend auf [55] und [112])

Scrubber	FPGA-Ressourcen			Fehlerkorrekturzeit (Grob/in ns)
	Slices	Register	BRAMs (18 kB)	
Xilinx [213]	149	-	2	hoch/(ca. 40 000 [112])
Heiner et al. [80]	1 308	1 082	6	hoch/(ca. 40 000 [112])
Dutton et al. [55]	182	-	1	gering/-
Legat et al. [112]	176	118	2 (ECC)	gering/2100
SHRC (RSU)	225	309	1	gering/1250

Die erste Implementierung stammt von Xilinx und ist eine hybride Lösung, basierend auf einem PicoBlaze und FPGA-Logik [213]. Der PicoBlaze realisiert den Controller und benötigt einen 18 kbit BRAM als Programmspeicher sowie LUTRAMs für einen internen Scratchpad-Speicher. Diese Lösung hat mehrere Nachteile. Zum einen ist sie durch die Verwendung der genannten Speicher anfällig für Strahlungseffekte und zum anderen deutlich langsamer in der Fehlerkorrektur als andere Implementierungen. Die zweite

Implementierung wird in [80] beschrieben, basiert auf dem Scrubber von Xilinx und wurde durch TMR-Verfahren erweitert. Die letzten drei Scrubber haben eine ähnliche Designarchitektur. Die Implementierungen von Dutton et al. und Legat et al. gehen jedoch nicht auf die Detektion von SEFs ein.

Die vorgestellten Implementierungen realisieren reine Readback Scrubber, während der Funktionsumfang des vorgestellten SHRCs wesentlich größer ist. Daher werden für einen Vergleich der FPGA-Ressourcen lediglich die ermittelten Werte für die RSU herangezogen. Bei der Implementierungen fällt insbesondere die deutlich höhere Anzahl an Slices bei der TMR-Implementierung von Heiner et al. auf. Der Ressourcenbedarf der anderen, insbesondere der letzten drei Scrubber, ähneln sich. Durch die Registrierung der Ein- und Ausgangssignale kommt es bei der RSU zu einer etwas höheren Slice-Anzahl. Dies ist ebenfalls durch den Vergleich der benötigten Register erkennbar.

Ein Vergleich der Scrubbing-Charakteristika kann nicht direkt erfolgen, da die Angaben in den Vergleichen zum einen unterschiedliche Parameter aufführen und zum anderen auf unterschiedlichen FPGAs beruhen (Legat et al.: Virtex-4 LX15, Dutton et al.: alle Virtex-4-FPGAs, RSU: Virtex-4 FX100). Die maximale Fehlererkennungszeit ist abhängig von dem konkreten FPGA, da sie einzig von der Gesamtanzahl der *Configuration Frames* abhängt. Die Vergleiche aus [112] und [55] zeigen, dass die maximale Fehlererkennungszeit für die unterschiedlichen Implementierungen nahezu identisch ist. Aufgrund dieser Tatsache kann die Fehlererkennungszeit als annähernd konstant angesehen werden und ist daher nicht separat in der Tabelle aufgeführt. Dennoch ist diese Zeit ein wichtiger Parameter, da sie die maximale Zeit beschreibt, in der ein SEU den Konfigurationsspeicher und damit das Design beeinflusst. Bei einer vollständigen Prüfung eines Virtex-4 FX100 liegt diese bei etwa 7,67 ms. Eine konkrete Korrekturzeit ist nur für den Scrubber aus [112] angegeben und liegt in der gleichen Größenordnung wie die der RSU. Die Korrekturzeit des Scrubbers aus [55] ist nicht angegeben, wird aufgrund der ähnlichen Designarchitektur jedoch ebenfalls in dieser Größenordnung liegen. Die Korrekturzeiten der Scrubber von Xilinx und Heiner et al. sind deutlich höher. In [55] wird der Faktor 20 genannt. Dies liegt laut Legat et al. daran, dass erstens ein vollständiges Zurücklesen durchgeführt wird, welches nicht unterbrochen werden kann, und zweitens im Anschluss an eine Fehlererkennung eine zusätzliche Frame-basierte Erkennung erfolgen muss, um die konkrete Position des SEUs zu bestimmen.

ECC-Statistik

Die von Xilinx implementierte ECC-Komponente innerhalb des MPMCs speichert grobgranulare ECC-Informationen zur Laufzeit [209, S. 70 ff.]. Dazu gehören allgemeine Statusinformationen, (12 bit breite) Zähler für Einbit-, Zweibit- und Paritätsfehler sowie ein Register für die ECC-Adresse, in der ein Fehler detektiert wurde. Diese Statistikeinheit wurde um Registern erweitert, um weitere relevante Informationen zur Laufzeit speichern zu können. Drei Register werden zur Speicherung der Anzahl der Einbit-, Zweibit- und Paritätsfehler verwendet. Im Gegensatz zu den existierenden Zählern, können diese durch ein Kommando zur Laufzeit zurückgesetzt werden und

sind mit 32 bit deutlich größer ausgelegt. Weiterhin werden drei Register mit Durchschnittswerten für die drei Fehlerkategorien bereitgestellt. Die Zeit zur Berechnung dieser Durchschnittswerte wird durch ein separates Register festgelegt. Eine weitere feingranulare Statistik wird durch die Speicherung der am häufigsten auftretenden, fehlerhaften Adressen festgehalten. Die Adresse wird dabei separat für einen Schreib- und Lesevorgang gespeichert.

Fehlerinjektion

Die von Xilinx bereitgestellte ECC-Komponente des MPMCs implementiert eine grobgranulare Fehlerinjektion, in der die Injektion von Einbit-, Zweibit- und Paritätsfehlern erzwungen werden kann [209, S. 75 ff.]. Diese Fehlerinjektionsfunktionalität wird in dem DRPM so erweitert, dass eine Fehlerinjektion in vordefinierten Adressbereichen erfolgen kann. Durch diese Erweiterung ist es somit gezielt möglich, den Einfluss von durch Strahlung induzierten SEUs, MBUs oder MCUs auf verschiedene Adressräume des DDR2-Speichers zu emulieren. Im DRPM kann dies primär zur Manipulation von im DDR2-Speicher abgelegten Konfigurationsdateien verwendet werden.

6.4.2 Behandlung von permanenten Fehlern

Die Auswirkungen von TID Effekten wurden bereits in Abschnitt 5.5.4 beschrieben. Neben einem Performanceverlust und einem Anstieg der Verlustleistung ist bei SRAM-basierten FPGAs insbesondere der Verlust der Programmierbarkeit zu nennen. Die Verdrahtungsstruktur moderner FPGAs belegt mit bis zu 90 % den Großteil der Bauteilfläche [22]. Ein permanenter Fehler in einem FPGA kann eine Verdrahtungsressource *direkt* oder durch die Konfiguration der damit verbundenen Verdrahtungsressourcen (PIPs) *indirekt* beeinflussen [168]. Der erstgenannte Fall führt zu dem funktionalen Effekt eines *Stuck-at-0/1*, sodass die physikalische Leitung auf einen logischen Wert von Null oder Eins festgesetzt ist. Der zweite Fall führt zum funktionalen Effekt eines *Stuck-on/off*, sodass ein PIP entweder programmiert oder unprogrammiert bleibt.

Das von der ESA geförderte Projekt *On-Line on-demand Testing approach for permanent Radiation Effects in REconfigurable systems* (OLT(RE)²) realisiert eine anwendungsunabhängige Fehlererkennung der Verdrahtungsressourcen eines FPGAs zur Laufzeit. Das Projekt wurde in Zusammenarbeit mit der Università di Pisa und dem Politecnico di Torino durchgeführt und in dem *Technique for radiation effects mitigation in ASICs and FPGAs handbook* der *European Cooperation for Space Standardization* (ECSS) aufgenommen.

Die Grundidee von OLT(RE)² ist eine Art Selbsttest für ein FPGA bereitzustellen, sodass eine FPGA-Fläche vor der Verwendung auf permanente Fehler überprüft wird. Hierfür werden *couple-parity*-basierte Testschaltungen mit einem *Test Pattern Generator* (TPG) zur Testerzeugung sowie einem *Output Response Analyzer* (ORA) zur Analyse erstellt. Die Verdrahtungsressourcen zwischen dem TPG und ORA sind die überprüften Ressourcen und werden zu sogenannten *Nets Under Test* (NUT) zusammengefasst. Zur

Abdeckung möglichst vieler Verdrahtungsressourcen werden mehrere Konfigurationen der Testschaltung mit variierenden Verdrahtungsressourcen erzeugt. Diese Testschaltungen werden mittels DPR in der zu testenden Region (*Area Under Test*, AUT) auf dem FPGA platziert. Die autonom arbeitenden Testschaltungen führen anschließend die Fehlererkennung durch. Das Ergebnis wird in LUTRAM-Komponenten des ORAs gespeichert, sodass dieses abschließend zurückgelesen und ausgewertet werden kann.

Der beschriebene Selbsttest des FPGAs ist aus zwei Gründen interessant. So steigt, wie in den Anwendungsbereichen (insbesondere HEP) in Abschnitt 5.9 ausgeführt, der Bedarf nach performanter, rekonfigurierbarer Hardware kontinuierlich, welcher oftmals in der Verwendung von COTS-FPGAs resultiert. Zusätzlich existieren Projekte, in denen strahlungsgehärtete oder -tolerante FPGAs aus Kostengründen nicht verwendet werden können (z. B. CubeSat-Projekte). OLT(RE)² unterstützt, aufgrund der Verwendung der DHHarMa-Softwarekomponenten (siehe Abschnitt 4.5), FPGAs bis zur 7-Serie. In [43] untersucht Cozzi die FPGA-Architekturen Virtex-5, Spartan-6 und Virtex-6 und stellt die Ergebnisse des Entwurfsablaufs vor.

In diesem Abschnitt werden zunächst die existierenden Fehlerkategorien und das Vorgehen zur Fehlererkennung vorgestellt und hierdurch eine Terminologie festgelegt. Anschließend wird die daraus resultierende Testschaltung zur Fehlererkennung dargestellt. Nach diesen wichtigen Grundlagen zur Fehlerkategorisierung und -erkennung wird der Gesamttestablauf, bestehend aus Entwurf und Ausführung, beschrieben. Abschließend erfolgt eine Auswertung der einzelnen Teile von OLT(RE)².

Fehlerkategorien beim Test der Verdrahtungsressourcen

In der Literatur werden die bereits genannten Fehlerkategorien für den Test der Verdrahtungsressourcen von FPGAs definiert [168]. Die Fehler werden zum einen durch den Typ der Verdrahtungsressource und zum anderen durch die Komplexität zur Detektion des Fehlers kategorisiert. Durch einen permanenten Fehler an einer physikalischen Leitung (engl. *Physical Wires*, PW) kann diese auf einen dauerhaften Logikpegel festgesetzt sein (*Stuck-at-0/1*). Ein permanenter Fehler in dem Konfigurationsspeicher oder an einem Transistor zur Realisierung des PIPs kann zu einer dauerhaften, fehlerhaften Programmierung eines PIPs führen. Wie eingangs beschrieben, kann dies zwei unterschiedlichen Auswirkungen haben: einen dauerhaft unprogrammierten PIP (*Stuck-off*) und einen dauerhaft programmierten PIP (*Stuck-on*). Die aufgeführten Fehlerkategorien sind in Abbildung 6.40 dargestellt.

Fehlerkategorie 1: Stuck-at-0/1 Die Überprüfung einer physikalischen Leitung erfolgt, indem beide möglichen Logikpegel übertragen werden. Ist dies möglich, kann die Verdrahtungsressource als Stuck-at-fehlerfrei angesehen werden.

Fehlerkategorie 2: Stuck-off PIP Im Fall eines Stuck-off PIPs wird in dem Netz, welches diese Verdrahtungsressource verwendet, ein *Open* Effekt erzeugt, welches das Netz in einen unbekanntem Zustand überführt. Die Überprüfung dieser Fehlerkategorie erfolgt

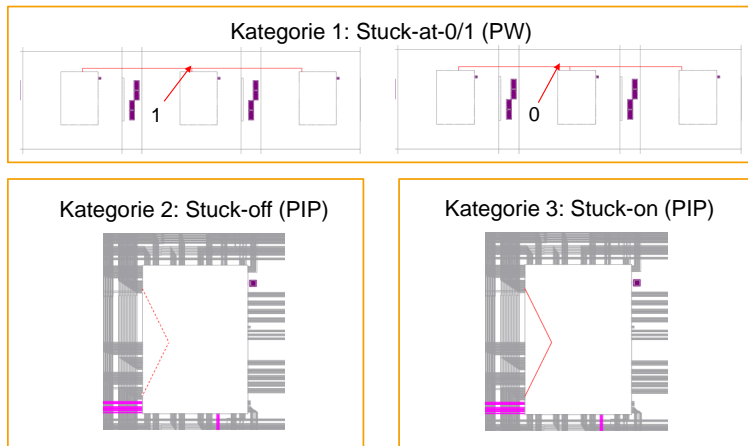


Abbildung 6.40: Fehlerkategorien der Verdrahtungsressourcen [48, S. 18]

analog zu der Überprüfung einen Stuck-at-Fehlers: Beide Logikpegel werden über den zu überprüfenden PIP versendet.

Fehlerkategorie 3: Stuck-on PIP Ein Stuck-on PIP kann zu unerwünschten Kurzschlüssen oder Brücken mit entsprechenden Seiteneffekten führen. Zur Überprüfung dieser Fehlerkategorie werden zwei Netze benötigt, da nur dann ein Konflikt zwischen diesen beiden Netzen ermittelt werden kann.

Vorgehen zur Fehlererkennung

Das konkrete Vorgehen zur Fehlererkennung der zuvor definierten Fehlerkategorien wird anhand der, in Abbildung 6.41 visualisierten, Switch Matrix zwischen einem TPG und ORA mit 7 physikalischen Leitungen (a-g) und 6 PIPs (α , β , γ , δ , ϵ und ζ) beschrieben. Zu testende Verdrahtungsressourcen werden für den weiteren Verlauf mit PIP-UT (engl. under test) respektive PW-UT abgekürzt.

Die Verschaltung zur Erkennung von Stuck-at Fehlern von PWs-UT (a und d) sowie eines Stuck-off Fehlers eines PIP-UT (β) ist in Abbildung 6.42 dargestellt. Für die Erkennung beider Fehlerkategorien werden beide Logikpegel zwischen TPG und ORA übertragen. Wichtig zu vermerken ist hierbei, dass alle enthaltenen Verdrahtungsressourcen in der Verbindung zwischen TPG und ORA, dem NUT, auf die beiden Fehlerkategorien überprüft werden.

Eine mögliche Verschaltung zur Erkennung eines Stuck-on Fehlers eines PIPs (β) ist in Abbildung 6.43 dargestellt. Wie im vorherigen Abschnitt beschrieben, werden zur Detektion eines Fehlers dieser Fehlerkategorie zwei Netze benötigt. In der Abbildung wird zudem verdeutlicht, dass durch dieses Vorgehen auf einen eventuellen Kurzschluss

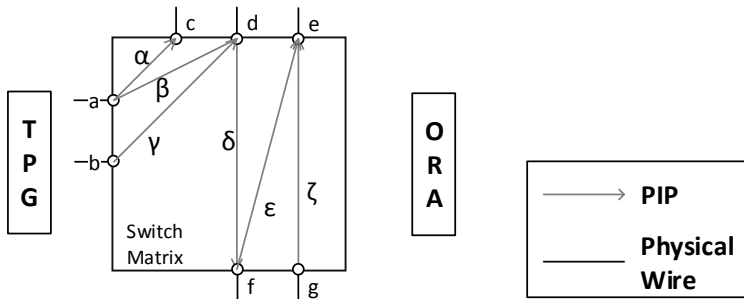


Abbildung 6.41: Switch-Matrix mit Hervorhebung von 6 PIPs und 7 physikalischen Leitungen

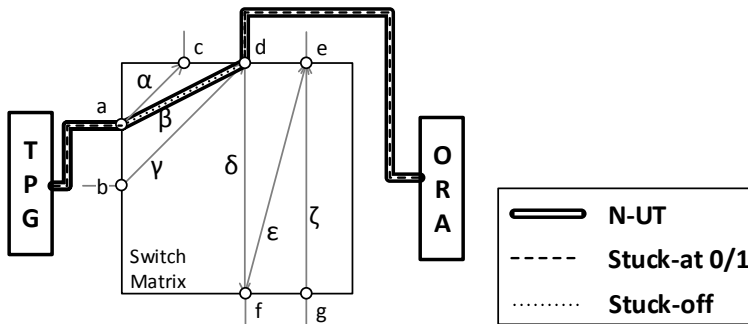


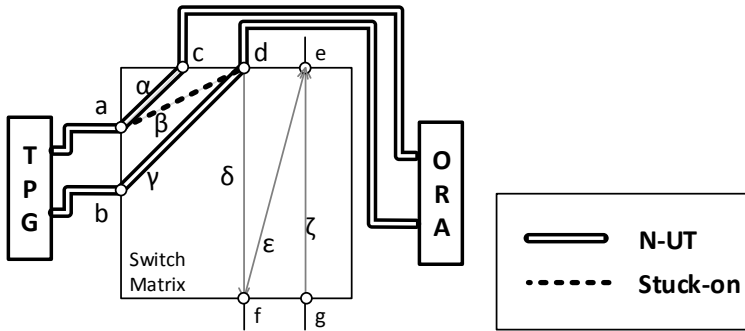
Abbildung 6.42: Test der Stuck-at Fehler der PW-UT a und d sowie des Stuck-off Fehlers des PIP-UT β

getestet wird. Anzumerken ist ebenfalls, dass durch diesen Testfall die beiden dargestellten NUTs die Stuck-at Fehler der PWs a , b , c und d sowie die Stuck-off Fehler der PIPs α und γ abgedeckt werden.

Testschaltungen

Die Schaltungen zur Überprüfung der Verdrahtungsressourcen bestehen, wie in Abschnitt 5.7.3 beschrieben, aus einem *Test Pattern Generator* (TPG) zur Generierung der Teststimuli und einem *Output Response Analyzer* (ORA) zur Fehlerdiagnose. Beide Komponenten werden über die zu testenden Verdrahtungsressourcen, den *Nets Under Test* (NUT), verbunden.

Bei der Dimensionierung der NUT muss ein Kompromiss zwischen der Anzahl verwendeter Verdrahtungsressourcen und der Granularität der Testauflösung gefunden werden.

Abbildung 6.43: Test des Stuck-on Fehlers des PIP-UT β

Die Testschaltung wurde initial für die Virtex-4-Architektur entworfen [160]. Die Anzahl der NUT ist daher zur optimalen Ausnutzung einer Slice auf den Wert 8 gesetzt, welcher der Anzahl der verfügbaren Eingänge beider LUTs einer Slice entspricht.

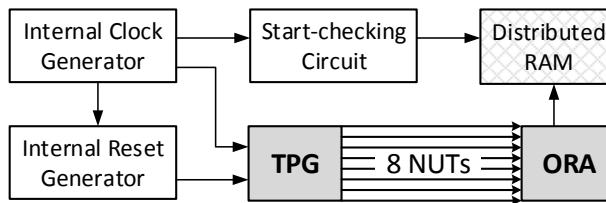


Abbildung 6.44: Blockschaltbild der Testschaltung zur permanenten Fehlererkennung

In Abbildung 6.44 ist das Blockschaltbild der Testschaltung dargestellt. Die interne Takterzeugung (*Internal Clock Generator*) erfolgt durch Ringoszillatoren. Anzumerken ist, dass die Taktversorgung über lokale Verdrahtungsressourcen realisiert wird. Das Reset-Signal, zur Sicherstellung eines definierten Anfangszustands, wird ebenfalls intern, durch ein 16-bit Schieberegister, erzeugt (*Internal Reset Generator*). Die Speicherung des Ergebnisses erfolgt in einer LUTRAM-Komponente (*Distributed RAM*), sodass dieses durch Zurücklesen ermittelt und anschließend von einer externen Komponente ausgewertet werden kann, ohne über IOBs kommunizieren zu müssen. Aufgrund dieser Eigenschaften kann die Testschaltung ihre Funktion völlig autonom ausführen. Zusätzlich wurde zur Testschaltung eine Komponente zur Erkennung eines korrekten Testdurchlaufs hinzugefügt (*Start-checking Circuit*). Diese Komponente erkennt Initialisierungsfehler, welche zu einem ungültigen Test führen würden [160]. Das Ergeb-

nis wird ebenfalls in einer LUTRAM-Komponente gespeichert und kann entsprechend ausgewertet werden.

Die wesentlichen Komponenten der Testschaltung sind die Komponenten TPG und ORA. Die Testschaltung verwendet, basierend auf den Ergebnissen aus Abschnitt 5.7.3, das *cross-coupled parity* Verfahren. Die konkrete Implementierung des Verfahrens ist in Abbildung 6.45 dargestellt. Der TPG besteht aus zwei 2-bit Zählern, welche jeweils durch zwei LUT4 und zwei FFs realisiert sind. Insgesamt wird somit eine Virtex-4-Slice pro Zähler benötigt. Der erste Zähler (*UP 2 bit counter*) zählt hierbei inkrementell von

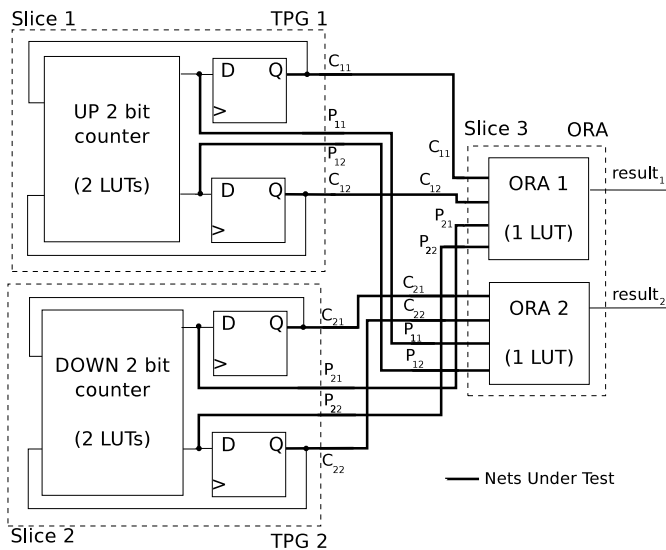


Abbildung 6.45: Komponenten der Testschaltungen zur permanenten Fehlererkennung [161, S. 4]

0 bis 3 und überträgt die Werte als Teststimuli (C_{11} und C_{12}) über FFs an den ersten ORA. Die Paritätsbits (P_{11} und P_{12}) werden durch die rückgekoppelten LUTs berechnet und sind mit dem zweiten ORA verbunden. Der zweite Zähler (*DOWN 2 bit counter*) zählt absteigend von 3 bis 0. Die Paritätsbits sind ebenfalls überkreuzt angeschlossen. Die Teststimuli der beiden TPG bestehen aufgrund der gewählten Größe der Zähler, wie in Tabelle 6.23 dargestellt, aus vier Testkonfigurationen. Der ORA überprüft, ob die einzelnen empfangenen Daten einer Testkonfiguration entsprechen und können eindeutig bestimmen, ob ein Fehler in den übertragenden Eingangsdaten oder dem verbundenen Zähler existiert. Die Zähler werden für die Adressierung der LUTRAM-Komponente zur Speicherung des aktuellen Testergebnisses verwendet.

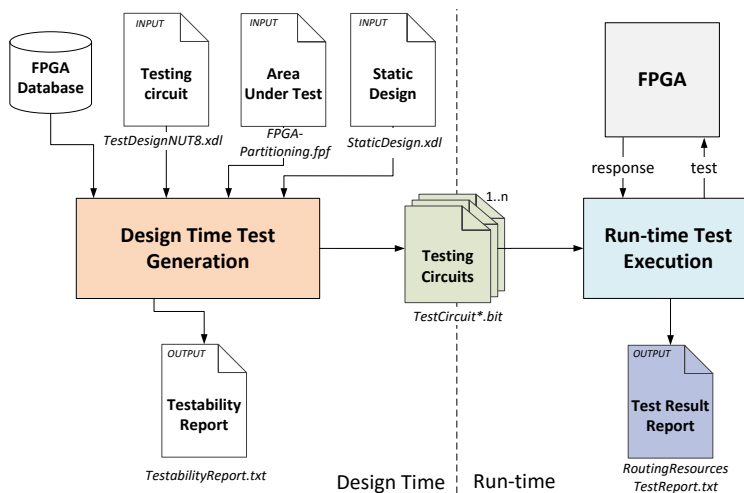
Tabelle 6.23: Teststimuli des TPG für die 8-NUT Testschaltung [161, S. 4]

Stimuli Nr.	C_{11}	C_{12}	P_{11}	P_{12}	C_{21}	C_{22}	P_{21}	P_{22}
1	0	0	0	1	1	1	1	0
2	0	1	1	0	1	0	0	1
3	1	0	1	1	0	1	0	0
4	1	1	0	0	0	0	1	1

Ein vollständiger Test einer konkreten Testschaltung wird in 20 Taktzyklen durchgeführt (16 Takte für den Reset und 4 Takte für den eigentlichen Test). Die Testschaltung wurde durch Fehlerinjektion auf Robustheit überprüft.

Testablauf

Der Testablauf von OLT(RE)² unterteilt sich, in Abhängigkeit des Ausführungszeitpunkts, in zwei Teile: den Entwurf (*Design Time*) und die Ausführung (*Run-time*). Eine Gesamtübersicht des Testablaufs ist in Abbildung 6.46 dargestellt. Die Schnittstelle beider Teile sind die erstellten Testschaltungen in Form partieller Konfigurationsdateien (*Testing Circuits*.bit*).

Abbildung 6.46: Gesamtübersicht des OLT(RE)² Testverfahrens [48, S. 7]

Entwurfsablauf

Wie der Name andeutet, ist die primäre Aufgabe des Entwurfsablaufs *Design Time Test Generation* die Erstellung von Testschaltungen zur Überprüfung der FPGA-Ressourcen. Bevor dies jedoch erfolgen kann, muss eine Kategorisierung der verfügbaren Verdrahtungsressourcen durchgeführt werden. Der Entwurfsablauf ist detaillierter in Abbildung 6.47 dargestellt. Der Entwurfsablauf ist durch C++ Softwarekomponenten

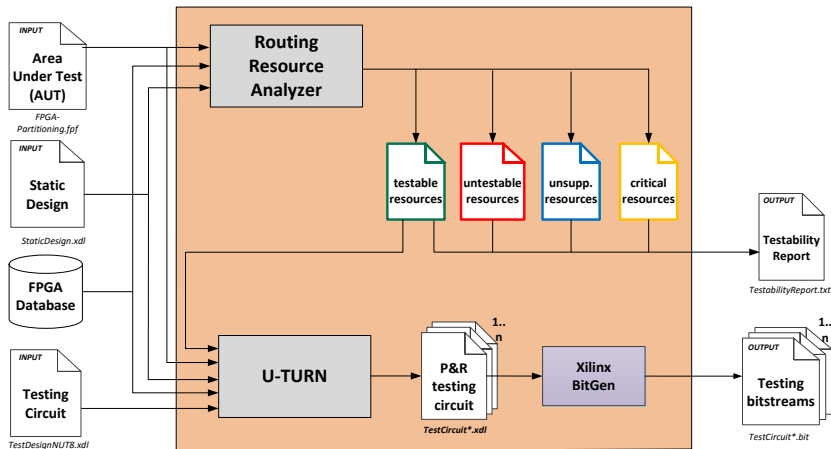


Abbildung 6.47: Entwurfsablauf von OLT(RE)² (basierend auf [48, S. 8])

realisiert und verwendet DHHarMa-Komponenten (siehe Abschnitt 4.5). Hierdurch werden aktuell die FPGAs der Familien Virtex-4 bis 7-Serie unterstützt.

Zur Ausführung werden vier Eingangsdateien benötigt. Die Partitionierung des Systems, in der die zu testende FPGA-Fläche (*Area Under Test*, AUT) aufgeführt ist, wird durch eine FPF-Datei festgelegt. Zusätzlich muss das statische Design sowie die Testschaltung in Form von XDL-Dateien (siehe Abschnitt 4.3) bereitgestellt werden. Die vierte Eingangsdatei ist die FPGA-Datenbank DXF von DHHarMa (siehe Abschnitt 4.5.1), in welcher der Aufbau des FPGAs gespeichert ist.

Kategorisierung der Verdrahtungsressourcen Die erste Softwarekomponente, der *Routing Resource Analyzer* (RRA), führt eine Kategorisierung der FPGA-Verdrahtungsressourcen durch: nicht-testbar, kritisch, testbar und nicht-unterstützt. Die ersten drei Kategorien ergeben sich durch die vorgegebene Systempartitionierung, wie in Abbildung 6.48 dargestellt. Verdrahtungsressourcen, die außerhalb der PR-Region und damit auch außerhalb der AUT liegen, werden je nach Verbindung als nicht-testbar (*untestable*) oder kritisch (*critical*) eingestuft. Zusätzlich werden Verdrahtungsressourcen der eingebetteten IP-Cores, der DSP- und BRAM-Komponenten sowie die Carry Chain

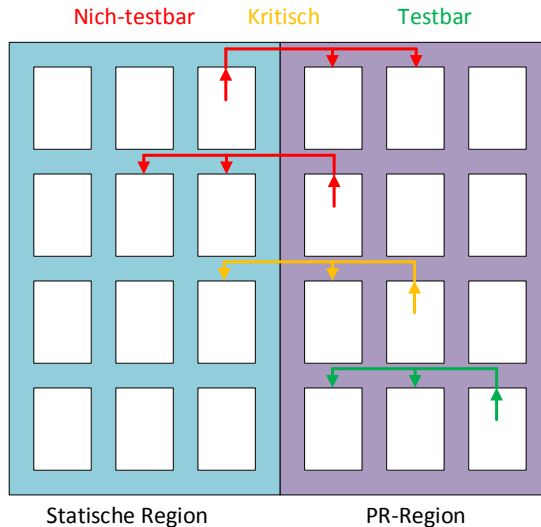


Abbildung 6.48: Kategorisierung der FPGA-Verdrahtungsressourcen (basierend auf [48, S. 11])

und der Taktverteilung aktuell nicht-unterstützt (*unsupported*). Die verbleibenden Verdrahtungsressourcen gehören der Kategorie testbar (*testable*) an. Das Ergebnis der Kategorisierung wird zusätzlich in einer Datei, dem *Testability Report*, dokumentiert.

Erstellung der Testschaltungen Die testbaren Verdrahtungsressourcen werden anschließend, zur Erzeugung der Testschaltungen, an die Softwarekomponente *U-TURN* übergeben. Der Name der Komponente beruht auf der Vorgehensweise des, von Mascolo realisierten Verdrahtungsalgorithmus [117]. Dieser rekursive Algorithmus bestimmt eine Switch Matrix als Ausgangspunkt des Tests und wählt für die Realisierung eines NUTs einen Pfad, welcher stets zu dieser zurückkehrt. Eine vollständige Beschreibung des Algorithmus erfolgt in [117]. Die vorherige Platzierung der Testkomponenten wurde durch einen Algorithmus von Cozzi realisiert und ist ausführlich in [43] beschrieben. Dieser Algorithmus garantiert eine klare Abgrenzung des Bereichs der Testkomponenten der Testschaltung und der zu testenden FPGA-Ressourcen, indem die AUT in weitere Unterbereiche (*Sub-Area Under Test*, SAUT) eingeteilt wird. Eine platzierte Testschaltung wird anschließend mehrere Male durch den U-TURN Algorithmus verdrahtet, bis alle testbaren Verdrahtungsressourcen einer SAUT abgedeckt sind. Jede vollständig verdrahtete Version der Testschaltung wird in Form einer XDL-Datei abgespeichert und anschließend durch das Xilinx Werkzeug *XDL* konvertiert und mittels *BitGen* eine partielle Konfigurationsdatei erstellt. Wie eingangs erwähnt, muss bei der

Dimensionierung der NUT allgemein ein Kompromiss zwischen der Anzahl verwendeter Verdrahtungsressourcen und der Granularität der Testauflösung gefunden werden. Tests in der Virtex-4-Architektur haben hierbei gezeigt, dass auf dieser eine physikalische Einschränkung bezüglich der Anzahl der verwendeten FPGA-Verdrahtungsressourcen gibt. Die Verwendung zu vieler PIPs (ca. 150 PIPs) in einem NUT führt aufgrund des kumulierten Jitters und abnehmender Flankensteilheit zu einer fehlerhaften Ausführung der Testschaltung. Um eine stabile Ausführung zu gewährleisten, wurde daher eine Obergrenze von 100 PIPs eingeführt.

Ausführung der Testschaltungen

Nach Erstellung kann die Ausführung der Testschaltungen *Run-time Test Execution* erfolgen. Der in Abbildung 6.49 dargestellte Ablauf, kann bei Bedarf zur Systemlaufzeit ausgeführt werden. Die Testschaltungen werden zunächst auf das Zielsystem übertragen, welches diese in einem internen Speicher ablegt. Anschließend können die Testschaltungen einzeln mittels DPR geladen und das Ergebnis der Überprüfung durch den ORA aus den LUTRAM-Komponenten der Testschaltung zurückgelesen werden. Nach dem Zurücklesen kann die nächste Testschaltung geladen und parallel dazu eine Auswertung des Testdurchlaufs durch den *Test Response Analyzer* erfolgen. Das Gesamtergebnis wird in Form eines Reports dokumentiert.

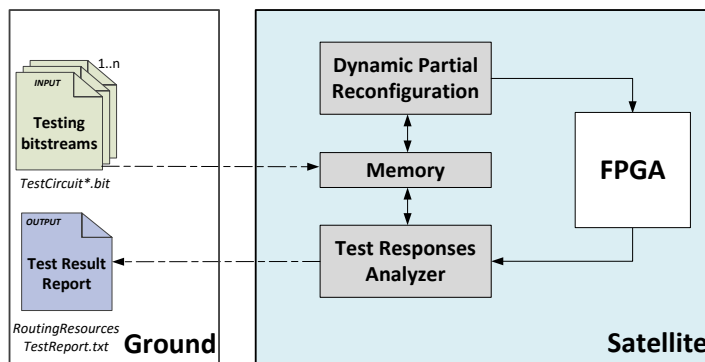


Abbildung 6.49: Ablauf der Testausführung von OLT(RE)² [48, S. 9]

Auswertung der einzelnen OLT(RE)²-Teile

Die Auswertung der Erkennung permanenter Fehler in den Verdrahtungsressourcen des PR-FPGAs erfolgt für die einzelnen OLT(RE)²-Teile. Zunächst erfolgt die Kategorisierung der Verdrahtungsressourcen des PR-FPGAs (Virtex-4 FX100). Im Anschluss werden die Testschaltungen hinsichtlich des FPGA-Ressourcenbedarfs, der Fehlerabdeckung

durch die Verwendung der testbaren Verdrahtungsressourcen und der benötigten Erstellungszeit sowie des Speicherbedarfs analysiert. Abschließend werden Ergebnisse von Hardware-Tests auf dem PR-FPGA des DPRMs vorgestellt. Hierbei werden die Ergebnisse des normalen Modus zur Fehlererkennung sowie zweier Fehlerinjektionstests zur Funktionsüberprüfung präsentiert.

Auswertung der Kategorisierung der Verdrahtungsressourcen

Die Kategorisierung der Verdrahtungsressourcen auf dem PR-FPGA erfolgt für eine Taktregion innerhalb der PR-Region durch den *Routing Resource Analyzer* (RRA). Eine Taktregion besteht aus 672 Switch Matrizen und 416 CLBs sowie 222 240 physikalischen Leitungen, die durch 2 323 415 PIPs verschaltet werden können.

Die Kategorisierung kann in zwei Teile aufgegliedert werden: Die Testschaltungs-unabhängige (engl. *Test Circuit Independent*, TCI) und die Testschaltungs-abhängigen (engl. *Test Circuit Dependent*, TCD) Analyse. Die erstgenannte Analyse betrachtet, wie in Abschnitt 6.4.2 beschrieben, die Auswirkung der Systempartitionierung. Die zweite Analyse wertet die verbleibenden testbaren Verdrahtungsressourcen bezüglich der Unterstützung durch die realisierten Testschaltungen aus. Wie bereits erwähnt, werden ein Großteil der Verdrahtungsressourcen der eingebetteten IP-Cores, der DSP- und BRAM-Komponenten sowie die Carry Chain und der Taktverteilung nicht berücksichtigt. In Abbildung 6.50 ist exemplarisch eine Switch Matrix mit, nach Test-Kategorie eingefärbten, Verdrahtungsressourcen dargestellt.

Tabelle 6.24: Kategorisierung der Verdrahtungsressourcen einer PR-FPGA-Taktregion

Typ der Verdrahtungsressource	Verfügbare Ressourcen in #	Test-Kategorie	Testschaltungs-unabhängige Analyse		Testschaltungs-abhängige Analyse	
			in #	in %	in #	in %
Physikalische Leitungen	222 240	nicht-testbar	16 811	7,5		
		kritisch	7 735	3,5		
		testbar	197 694	89,0	111 179	56,2
		nicht-unterstützt			86 515	43,8
PIPs	2 323 415	nicht-testbar	258 242	11,2		
		kritisch	191 436	8,2		
		testbar	1 873 737	80,6	1 317 736	70,3
		nicht-unterstützt			556 001	29,7

Tabelle 6.24 stellt die Ergebnisse der Test-Kategorisierung dar. Durch die Systempartitionierung können 89,0 % der physikalischen Leitungen und 80,6 % der PIPs auf permanente Fehler überprüft werden. Die verbleibenden 11,0 % der physikalischen Leitungen und 19,4 % der PIPs könnten durch zusätzliche, Taktregions-überlappende Schaltungen weiter reduziert werden. Durch die realisierten Testschaltungen wird die Anzahl testbarer Verdrahtungsressourcen weiter eingeschränkt. Insgesamt kön-

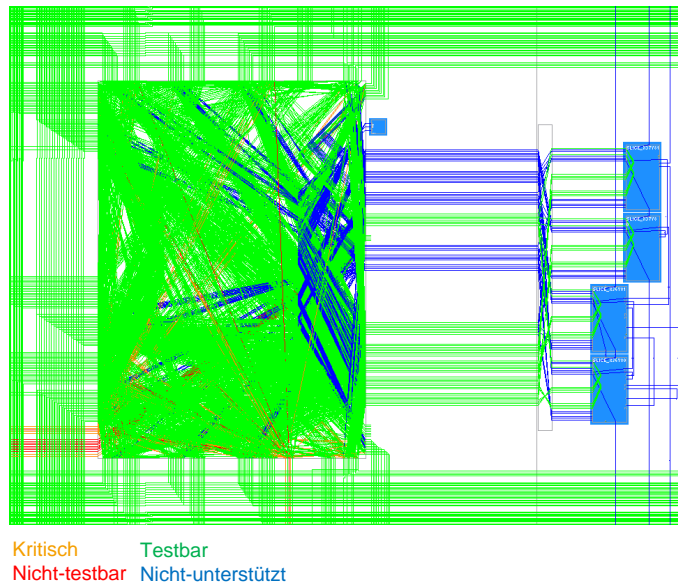


Abbildung 6.50: Switch Matrix mit, nach Test-Kategorie eingefärbten, Verdrahtungsressourcen (basierend auf [48, S. 13])

nen 111 179 physikalische Leitungen auf permanente Fehler getestet werden, welches 56,2 % der testbaren Leitungen entspricht. Die Anzahl der unterstützten PIPs ist mit 70,3 % höher. Die *Heatmaps* in Anhang C.4 stellen beide Auswertungen der testbaren PIPs für die untersuchte Taktregion grafisch dar. Die dargestellte Einfärbung entspricht der prozentualen Abdeckung der Switch Matrizen und verdeutlicht insbesondere die Auswirkung der Partitionierung an Randgebieten.

Auswertung der Testschaltungen

Nach der Kategorisierung erfolgt im nächsten Schritt des OLT(RE)² Entwurfsablauf die Erstellung der Testschaltungen. Das grundlegende Ziel hierbei ist eine möglichst hohe Abdeckung der testbaren Verdrahtungsressourcen.

Ressourcenauslastung der Testschaltungen Die autonom arbeitenden Testschaltungen haben zum Ziel, eine möglichst geringe Anzahl von FPGA-Ressourcen zu verwenden, sodass deren Einflussradius klein ist. In Tabelle 6.25 sind die benötigten FPGA-Ressourcen aufgeführt. Wie dargestellt, werden insgesamt lediglich 39 LUTs und 11 FFs für die Implementierung auf einem Virtex-4 FPGA benötigt. Die gesamte Testschaltung belegt 24 Slices und ist durch 43 Netze intern verschaltet. Durch die Verwendung lokaler

Verdrahtungsressourcen für die interne Verschaltung sind die Testschaltungen dabei flexibel auf dem FPGA platzierbar.

Tabelle 6.25: Ressourcenbedarf der OLT(RE)² Testschaltung

Testschaltungs- komponenten	Logik	LUTs		FFs
		LUTRAM	Schieberegister	
Clock Generator	6	0	0	1
Reset Generator	0	0	1	0
Start Checker	1	1	0	0
TPG	14	0	0	8
ORA	14	2	0	2
Σ	35	3	1	11

Abdeckung der testbaren Verdrahtungsressourcen Der in Abschnitt 6.4.2 beschriebene Verdrahtungsalgorithmus *U-TURN* hat zum Ziel, möglichst viele testbare Verdrahtungsressourcen einer AUT abzudecken. Zusätzlich ist es möglich, den Algorithmus auf die Überprüfung eines Verdrahtungsressourcentyps zu parametrieren. Die Tabelle 6.26 stellt die ermittelten testbaren Verdrahtungsressourcen, die Fehlerabdeckung zur Überprüfung beider Verdrahtungsressourcentypen, sowie die daraus resultierende Anzahl benötigter Testschaltungen dar. Hervorzuheben ist hierbei, die unterschiedliche Anzahl testbarer PIPs aufgrund der Fehlerkategorie. Wie dargestellt, ist bereits mit einer Anzahl von 846 Testschaltungen eine Abdeckung der Stuck-at Fehler der physikalischen Leitungen (PW) möglich. Gleichzeitig erfolgt eine Überprüfung der PIP-Fehlerkategorien, jedoch mit moderater Abdeckung (Stuck-off: 24,0 %, Stuck-on: 14,3 %). Eine nahezu vollständige Abdeckung beider PIP-Fehlerkategorien für die testbaren PIP-Ressourcen führt zu einem deutlichen Anstieg benötigter Testschaltungen. Insbesondere durch die Stuck-on Fehlerabdeckung werden 38 245 Testschaltungen benötigt, welches in einen Faktor 45 im Vergleich zur Überprüfung der physikalischen Leitungen resultiert.

Tabelle 6.26: Testabdeckung der Verdrahtungsressourcen einer Taktregion des PR-FPGAs (basierend auf [48, S. 23])

Testbare Verdrahtungsressourcen			Testabdeckung der Verdrahtungsressourcen						Test- Designs
PWs		PIPs	PWs		PIPs				
Stuck-at-0/1 in #	Stuck-off in #	Stuck-on in #	Stuck-at-0/1 in #	in %	Stuck-off in #	in %	Stuck-on in #	in %	
111 179	1 317 736	1 270 955	110 167	99,1	315 883	24,0	181 952	14,3	846
			111 179	100,0	1 305 778	99,1	1 248 119	98,2	38 245

Speicher- und Zeitauswertung Um eine Vorstellung von den Laufzeiten der beiden Softwarekomponenten, sowie den Speicherbedarf der Testschaltungen zu bekommen, wurde eine Auswertung beider Maße durchgeführt. Die Ergebnisse sind in Tabelle 6.27 aufgeführt. Die Kategorisierung der Verdrahtungsressourcen der Taktregion erfolgt innerhalb von 75 Sekunden. Die dargestellte Zeit von *U-TURN* enthält neben der Laufzeit des eigentlichen Algorithmus die Konvertierungszeiten zur Erstellung der partiellen Konfigurationsdateien. Der Einfluss der überprüften Fehlerkategorie ist hierbei deutlich erkennbar. Anzumerken ist hierbei, dass zwei unterschiedliche Hardware-Setups zur Erstellung verwendet wurden. Die erste *U-TURN* Laufzeit wurde mit einem Intel i7-2600K Prozessor (3,4 GHz) mit 12 GB DDR3 (1 333 MHz), die zweite Laufzeit mit einem Intel Xeon W3565 Prozessor (3,2 GHz) mit 24 GB DDR3 (1 333 MHz) ermittelt.

Tabelle 6.27: Speicher- und Zeitauswertung der Schritte zur Erstellung der OLT(RE)² Testschaltungen

Software- komponente	Testschal- tungen in #	Speicherbedarf		Laufzeit in hh:mm
		unkompr. in MB	kompr. in MB	
RRA	-	-	-	00:01
U-TURN	846	36	3,3	04:11
	38 245	1 623	149,2	78:25

Hardware-Tests

Die Funktion der Fehlererkennung der erstellten Testschaltungen wurde anschließend auf dem PR-FPGA des DPRMs getestet. Zusätzlich wurden zwei Fehlerinjektionstests durchgeführt. So wurden zur Funktionsüberprüfung der Testschaltung *Open*-Fehler in sämtliche Netze der Testschaltung injiziert. Zusätzlich wurde die Fehlererkennung der Testschaltungen durch Injektion von *Stuck-at*-Fehlern in den physikalischen Leitungen überprüft.

Testablauf Der Ablauf der Hardware-Tests ist hierbei identisch. Zunächst werden die Testschaltungen sowie die jeweiligen Positionen (Konfigurationsspeicheradressen) der LUTRAM-Komponenten zur Speicherung der Testergebnisse von einem Host-PC über die Local Bus Schnittstelle in den DDR2-Speicher des PR-FPGAs transferiert. Anschließend steuert die DPCU die Testausführung. Die partiellen Konfigurationsdateien mit den implementierten Testschaltungen werden durch den SHRC mittel DPR in der PR-Region platziert. Das Zurücklesen des Testergebnisses wird direkt im Anschluss initialisiert, da die Ausführungszeit der Testschaltung deutlich geringer ist, als die Parametrierungszeit des SHRCs zum Zurücklesen. Die DPCU wertet anschließend den Inhalt der Testergebnisse (Start-Checker und ORA) aus und initialisiert den nächsten

Testlauf. Am Testende sendet die DPCU die Testergebnisse zu dem Host-PC, welcher einen Test-Report erstellt.

Testergebnisse Durch die Verwendung zweier Komponenten zur Testauswertung (Start-Checker und ORA) kann das Testergebnis in drei mögliche Kategorien fallen:

- *TNS*: Der Test wurde nicht durchgeführt (engl. Test Not Started)
- *TSFD*: Der Test wurde durchgeführt und ein Fehler erkannt (engl. Test Started Fault Detected)
- *TSFND*: Der Test wurde durchgeführt und kein Fehler erkannt (engl. Test Started Fault Not Detected)

Je nach Test lassen sich diese Kategorien zu positiven und negativen Ergebnissen gruppieren. Bei dem Test zur Fehlererkennung sind die negativen Ergebnisse *TNS* und *TSFND*. Für die Fehlerinjektionstests gelten diese Ergebnisse hingegen als positiv.

Funktionsüberprüfung der Testschaltung Zur Erkennung möglicher Schwachstellen wurde eine Funktionsüberprüfung der Testschaltung durch Injektion von *Open*-Fehlern durchgeführt. Die Injektion erfolgte konkret durch Entfernen eines PIPs in einem Netz. Da jeder PIP ein notwendiger Bestandteil des Netzes ist, reicht das Entfernen eines PIPs um die Verbindung zu unterbrechen. Da einige Netze gegabelt sind, wurde jeweils der erste PIP in allen 43 Netzen entfernt. Die Ergebnisse dieser 43 Testschaltungen sind in Tabelle 6.28 dargestellt. Insgesamt werden 83,7 % aller injizierten Fehler von der Testschaltung erkannt. Die Erkennung von injizierten *Open*-Fehlern in den NUTs erfolgt zu 100 %. Wie dargestellt, werden jedoch auch 16,3 % der injizierten Fehler von der Testschaltung nicht erkannt. Durch die Kenntnis, welche Netze betroffen können Rückschlüsse gezogen werden: Fehler in internen Kontrollsignalen des TPGs und ORAs können zur Blockade des Tests führen. Zusätzlich können Fehler dazu führen, dass Ergebnisse nicht in dem LUTRAM geschrieben werden. Der letztgenannte Fehler resultiert daher, dass sich ein Netz mit treibender logischer Null, bei einer Unterbrechung in ein Netz mit treibender logischer Eins verwandelt.

Funktionsüberprüfung der Fehlererkennung Zur Funktionsüberprüfung der Fehlererkennung der Testschaltungen wurde eine Injektion von *Stuck-at*-Fehlern in den NUT durchgeführt. Dabei wurden beide Fälle (*Stuck-at-0* und *Stuck-at-1*) für die 846 Testschaltungen zur Überprüfung der physikalischen Leitungen betrachtet. Die Fehlerinjektion erfolgte durch Manipulation der LUT-Gleichungen von Quellen der NUT (TPG-Ausgänge) auf einen festen Wert. Die Testergebnisse dieser Fehlerinjektionen ergaben in beiden Fällen eine Fehlererkennung von 100 % (*TSFD*).

Fehlererkennung Abschließend erfolgte ein Testdurchlauf mit den Testschaltungen ohne jegliche injizierte Fehler auf dem PR-FPGA. Die Testergebnisse waren für alle Testschaltungen *TSFND*, sodass in 100 % der testbaren Verdrahtungsressourcen keine permanenten Fehler erkannt wurden.

Tabelle 6.28: Ergebnisse der *Open*-Fehlerinjektion auf dem PR-FPGA

Testschaltungs- komponenten	Injizierte Fehler in #	Positive Ergebnisse				Negative Ergebnisse	
		TNS in #	TSFD in #	Σ in #	Σ in %	TSFND in #	TSFND in %
Clock Generator	6	6	0	6	100,0	0	0,0
Reset Generator	1	1	0	1	100,0	0	0,0
Start Checker	4	2	0	2	50,0	2	50,0
TPG	12	0	8	8	66,7	4	33,3
ORA	12	1	10	11	91,7	1	8,3
NUTs	8	0	8	8	100,0	0	0,0
Σ	43	10	26	36	83,7	7	16,3

Testausführungszeit Die Ausführungszeit der beiden vorgestellten Testsätze zur Fehlererkennung von permanenten Fehlern variiert durch die Anzahl der Testschaltungen stark. Die Überprüfung der testbaren physikalischen Leitungen durch die 846 Testschaltungen dauert 26 Sekunden, die Überprüfung der testbaren PIPs durch 38 245 Testschaltungen hingegen mehr als 19 Minuten.

6.5 Zusammenfassung

In diesem Kapitel wurden die, in der Einleitung dieser Dissertation aufgeführten, Ziele anhand des, von der Europäischen Weltraumorganisation geförderten, DRPM Demonstrators evaluiert. Zum Verständnis wurde dieser Demonstrator zunächst näher beschrieben. Hierbei wurde die zugrunde liegende Hardware, das RAPTOR-Board und die Erweiterungsmodule DB-V4 und DB-SPACE, vorgestellt. Anschließend wurde die skalierbare DRPM-Systemarchitektur vorgestellt und separat auf die beiden Modultypen des DPRMs, das Verarbeitungsmodul und das Kommunikationsmodul, eingegangen. Weiterhin wurden Systemvarianten des DPRMs erläutert, welche die Skalierbarkeit und die Anpassung der Systemarchitektur an unterschiedliche Anwendungsszenarien unterstreichen.

Im Anschluss wurde auf die wesentlichen Komponenten des DPRMs zur Realisierung eines flexiblen INDRA 2.0 DPR-Systems eingegangen. Konkret wurden die Realisierungen eines externen und eines internen Rekonfigurationskontrollers vorgestellt. Hierbei wurde insbesondere der *Self-Hosting Reconfiguration Controller* beschrieben. Dieser implementiert den internen Rekonfigurationskontroller und ermöglicht die dynamisch partielle Rekonfiguration mit Modulumplatzierung durch die Modifikation der Konfigurationsdatei und das Zurücklesen des Konfigurationsspeichers. Zusätzlich wurde eine Kommunikationsbrücke für den dynamischen Bereich zur Kommunikation mit dem Prozessorsubsystem sowie eine Streaming-basierte, dedizierte, performante Anbindung

an den DDR2-Speicher des Verarbeitungsmoduls entwickelt. Für die daran angeschlossene DPR-Kommunikationsinfrastruktur wurden, mittels DHHarMa erstellte, ein- und zweidimensionale Kommunikationsinfrastrukturen mit variierender Kachelanzahl vorgestellt. Im Rahmen des DPRM-Projekts wurden unterschiedliche Anwendungen in Form von dynamischen Modulen (PR-Modulen) entwickelt. Konkret wurden FIR-Filter, FFT-Implementierungen und simple Bildfilter vorgestellt. Die beiden erstgenannten PR-Modul-Typen wurden bezüglich der Leistungsfähigkeit, des FPGA-Ressourcenbedarfs und der Leistung bzw. des Energiebedarfs analysiert. Basierend auf diesen Ergebnissen ist die Wahl eines optimalen PR-Modules möglich.

Der DRPM Demonstrator bietet sich aufgrund der beschriebenen Eigenschaften als ideale Testplattform für die Entwicklung von Verfahren zur Erhöhung der Fehlertoleranz an. So wurde der SHRC für die Behandlung von Auswirkungen durch SEE im Konfigurationsspeicher des FPGAs um die *Readback Scrubbing Unit (RSU)* erweitert. Diese Komponente realisiert ein (*Configuration Frame* basiertes) Readback Scrubbing mit Einbitfehlerkorrektur und Zweibitfehlererkennung. Durch individuelle Scrubbing Tasks können Readback Scrubbing Schedules erstellt und das Readback Scrubbing anschließend autonom ausgeführt werden. Hierdurch ist es weiterhin möglich, adaptive Scrubbing-Raten für einzelne FPGA-Bereiche zu realisieren, sodass kritische Komponenten öfter überprüft werden können und die zusätzlich benötigte Leistungsaufnahme durch das Scrubbing minimiert werden kann. Weiterhin wurde durch integrierte Fehlerstatistikmechanismen, der Speicherung und Analyse der Bitfehlerinformationen (Typ des Fehlers, Configuration Frame, Bitposition innerhalb des Configuration Frames und Zeitpunkt des Auftretens), eine adaptive Anpassung des Systems ermöglicht. So kann bei einer steigenden Fehlerrate die Zuverlässigkeit des Systems, z. B. durch Verwendung von robusteren, durch Redundanzverfahren erweiterten, (statischen und dynamischen) Konfigurationen, erhöht werden. In einem Vergleich mit anderen Scrubber-Implementierungen für die Virtex-4-Architektur wurde festgehalten, dass die RSU die schnellste Fehlerkorrekturzeit bietet.

Mit OLT(RE)² wurde ein neues Verfahren zur Erkennung permanenter Fehler in den Verdrahtungsressourcen eines SRAM-basierten FPGAs vorgestellt. Die Entwicklung des Verfahrens wurde ebenfalls von der ESA gefördert und OLT(RE)² in dem *Techniques for radiation effects mitigation in ASICs and FPGAs handbook* der *European Cooperation for Space Standardization (ECSS)* aufgenommen. Die Grundidee von OLT(RE)² ist eine Art Selbsttest für ein FPGA bereitzustellen, sodass eine FPGA-Fläche vor der Verwendung auf permanente Fehler in den Verdrahtungsressourcen (physikalische Leitungen: Stuck-at-0/1, PIPs: Stuck-off/on) überprüft werden kann. Zur Diagnose der Verdrahtungsressourcen wurde eine autonome Testschaltung mit kleinem FPGA-Footprint entwickelt, die aus einem *Test Pattern Generator (TPG)* zur Testerzeugung sowie einem *Output Response Analyzer (ORA)* zur Analyse besteht. Die Verdrahtungsressourcen zwischen dem TPG und ORA sind die überprüften Ressourcen und werden zu sogenannten *Nets Under Test (NUT)* zusammengefasst. Zur Erhöhung der Fehlerabdeckung wurden stets mehrere Konfigurationen der Testschaltung mit variierenden Verdrahtungsressourcen

erzeugt. Hierbei muss festgehalten werden, dass aufgrund der Partitionierungen und der verwendeten Testschaltungen nicht alle Verdrahtungsressourcen überprüft werden können (physikalische Leitungen: 56,2 %, PIPs: 70,3 %). Die Erhöhung der Fehlerabdeckung wäre daher wünschenswert und könnte durch überlappende Testregionen und weitere Testschaltungen mit Unterstützung weiterer Hard-IP-Cores realisiert werden. Diese Testschaltungen werden mittels DPR durch den SHRC in der zu testenden Region (*Area Under Test*, AUT) auf dem FPGA platziert. Die autonom arbeitenden Testschaltungen führen anschließend automatisch die Fehlererkennung durch. Das Ergebnis wird in LUTRAM-Komponenten gespeichert, zurückgelesen und zur Laufzeit auf dem FPGA ausgewertet. Die Funktionsweise der Testschaltungen wurde anschließend auf dem PR-FPGA des DRPMs evaluiert. Hierbei wurde die Funktion der Fehlererkennung durch Injektion von Stuck-at-0/1 Fehlern überprüft. Zusätzlich wurde die Anfälligkeit aller Komponenten der Testschaltung durch Fehlerinjektion (Open Fehler) analysiert. Bei der Diagnose der Verdrahtungsressourcen ergab sich eine, je nach Fehlerart variierende Komplexität. So wurden zur Überprüfung auf Stuck-at-0/1 Fehler der 111 179 testbaren physikalischen Leitungen einer Taktregion auf dem PR-FPGA 846 Testschaltungen benötigt. Zur Überprüfung der Stuck-off/on Fehler der 1 305 778 bzw. 1 248 119 testbaren PIPs wurden hingegen 38 245 Testschaltungen erstellt. Es ergibt sich somit ein Faktor von 45 zwischen der Überprüfung der beiden Verdrahtungsressourcetypen. Dieser Faktor zeichnete sich in dem Speicherbedarf (149,2 MB zu 3,3 MB) sowie der Ausführungszeit (19 min zu 26 s) ab.

Durch die Kombination aller in der Einleitung aufgeführten Ziele dieser Dissertation konnte der DRPM Demonstrator zu einem flexiblen, adaptiven und insbesondere fehlertoleranten DPR-System erweitert werden. Die Fehlertoleranz wurde durch die Fehlererkennung und -korrektur von SEEs im Konfigurationsspeicher sowie die Fehlererkennung von permanenten Fehlern in den Verdrahtungsressourcen des SRAM-basierten PR-FPGAs erhöht.

7 Zusammenfassung und Ausblick

Rekonfigurierbare Architekturen bieten einen Kompromiss zwischen der Leistungsfähigkeit anwendungsspezifischer Schaltungen (ASICs) und der Flexibilität heutiger Prozessoren. FPGAs haben sich aufgrund dieser Maße zu der am häufigsten eingesetzten rekonfigurierbaren Architektur entwickelt. Insbesondere die Konfigurationsart dynamisch partielle Rekonfiguration (DPR) moderner SRAM-basierter FPGAs verdeutlicht die gewonnene System-Flexibilität. DPR wird in verschiedensten Anwendungsgebieten aus unterschiedlichsten Motivationen heraus eingesetzt. Die Hauptanwendung der DPR ist die Erstellung eines Systems, welches Veränderungen an bestimmten FPGA-Bereichen zur Laufzeit erlaubt. Zusätzlich wird in Anwendungsgebieten mit Strahlungseinfluss die DPR-Funktionalität in Verfahren zur Abschwächung von durch Strahlung induzierten Fehlern eingesetzt. In dieser Dissertation wurde diese Konfigurationsart mehrfach zur Realisierung eines flexiblen und gegenüber durch Strahlung induzierten Fehler toleranten DPR-Systems eingesetzt.

Hierbei wurde gezeigt, dass die Eigenschaften eines realisierten DPR-Systems durch den verwendeten Entwurfsablauf bestimmt werden. Im Vergleich zum bereits bei kleinen DPR-Systemen hohen Speicher- und Wartungsaufwand eines Xilinx DPR-Systems, ermöglicht der realisierte Entwurfsablauf von INDRA 2.0 den Entwurf von flexiblen und wartbaren DPR-Systemen, welche die verfügbare Hardware-Flexibilität vollständig erschließen. Die Einschränkungen in Xilinx DPR-Systemen resultieren aus einer inhomogenen Kommunikationsinfrastruktur zwischen statischen und dynamischen Bereichen. Die DPR-Kommunikationsinfrastruktur ist in dynamischen Bereichen in der Regel unterschiedlich verdrahtet, sodass sich nach der Synthese eine feste Position für die dynamischen Komponenten ergibt. Darüber hinaus ergibt sich die Schwierigkeit, dass Änderungen am statischen Design größtenteils auch Änderungen an der Platzierung und Verdrahtung der DPR-Kommunikationsinfrastruktur zur Folge haben. Dies führt zu einer Inkompatibilität zwischen der veränderten DPR-Kommunikationsinfrastruktur und den dynamischen Komponenten, sodass diese ebenfalls neu implementiert werden müssen. Beide aufgeführten Eigenschaften, eine homogene DPR-Kommunikationsinfrastruktur und eine Entkopplung der Entwurfsabläufe der statischen und dynamischen Komponenten, sind die primären Ziele der vorgestellten Entwurfsabläufe aus dem akademischen Bereich. Ein Vergleich dieser durch verschiedene Bewertungsmaße zeigte, dass insbesondere der im Rahmen der Dissertation von Cozzi und dieser Dissertation konzipierte und implementierte Entwurfsablauf von INDRA 2.0 die Realisierung von flexiblen und wartbaren DPR-Systemen mit einer Unterstützung der Xilinx

FPGA-Familien ab Virtex-4 bis zur 7-Serie ermöglicht. Die Umsetzung des ersten Ziels dieser Dissertation baut auf den aktuellen Xilinx *PR* Entwurfsablauf auf und erweitert diesen durch die Erstellung einer homogenen DPR-Kommunikationsinfrastruktur und der Entkopplung des Entwurfs von statischen und dynamischen Komponenten. Für die Entkopplung wurde mit dem Nachbearbeitungsschritt durch den *PS ReRouter* unter Verwendung der *FED (FPGA Editor Datenbank)* eine Lösung gefunden, welche garantiert, dass die verwendeten Xilinx Werkzeuge nicht eingeschränkt werden.

Für Anwendungen mit Homogenitätsanforderungen, wie TDCs oder die bereits aufgeführte homogene DPR-Kommunikationsinfrastruktur, wurde mit *DHHarMa* die Transformation eines zunächst inhomogenen Designs in ein homogenes Design ermöglicht und somit das zweite Ziel der Dissertation realisiert. Die Entwicklung eigener Softwarekomponenten war notwendig, da die Werkzeuge von Xilinx bezüglich der Homogenität nicht parametrierbar sind. Die Erstellung eines *homogenen* Hard-Makros wurde durch drei softwarebasierte Homogenisierungsschritte realisiert. Ein *Packer* garantiert die homogene Verwendung der internen FPGA-Ressourcen, konkret der (CLB-) internen Komponenten, ein *Placer* die homogene Platzierung der Designkomponenten und ein *Router* die Verwendung gleicher Verdrahtungsressourcen bei der Verdrahtung des Designs. Die entwickelten Softwarekomponenten werden durch zwei eigene FPGA-Datenbanken (*DXF* und *FED*) unterstützt. Beide Datenbanken enthalten Informationen über den konkreten Aufbau von Xilinx FPGAs der Familien Virtex-4 bis hin zur 7-Serie.

Die einzelnen *DHHarMa*-Softwarekomponenten wurden detailliert durch verschiedene Bewertungsmaße analysiert. Der Vergleich der Speicher- und Zeitauswertung für die FPGA Datenbanken verdeutlichte im Fall der *DXF* das Einsparungspotenzial gegenüber einer direkten Verwendung der Quelldatei (XDL-Report). Darüber hinaus wurde eine Analyse der vier realisierten Platzierungsstrategien für den Anwendungsfall der DPR-Kommunikationsinfrastruktur vorgestellt. Zusammenfassend ließ sich festhalten, dass die Platzierungsstrategie *Simulated Annealing* mit einer Cluster-basierten Anfangsplatzierung die besten Platzierungsergebnisse bei vergleichsweise geringer Ausführungszeit erzielt. Bei der Analyse wurde der Einfluss der FPGA-Architektur durch die Verwendung von Virtex-4 und Artix-7 FPGAs berücksichtigt.

Anhand der Gegenüberstellung von inhomogenen Xilinx- und homogenen *DHHarMa*-Implementierungen für die zwei vorgestellten Anwendungsgebiete mit homogenen Designanforderungen, erfolgte eine Auswertung der Softwarekomponenten für die drei Homogenisierungsschritte. Um einen Einfluss der FPGA-Architektur auf die Homogenisierungsschritte ermitteln zu können, erfolgte bei dieser Auswertung die Betrachtung dreier FPGA-Architekturen (Virtex-4, Virtex-6 und 7-Serie). Hierbei wurde die, bereits in der Einleitung aufgeführte, Fragestellung bezüglich des Einflusses der Homogenität auf den FPGA-Ressourcenbedarf und die Leistungsfähigkeit beantwortet. Der ausführliche Vergleich zeigte, dass die benötigte Homogenität auf den FPGA-Ressourcenbedarf keinen negativen Einfluss hat. Vielmehr wiesen die durch den *DHHarMa*-Packer erstellten

Implementierungen eine geringere Slice-Anzahl auf. Die betrachteten Bewertungsmaße des Packervergleichs in dem Anwendungsgebiet der DPR-Kommunikationsinfrastrukturen (Gesamtauslastung des FPGAs, (LUT- und Register-) Packungsdichten und Auslastung der PR-Region durch die Kommunikationsinfrastruktur) fielen somit in allen FPGA-Architekturen zugunsten des DHHarMa-Packers aus. Anzumerken ist hierbei, dass es aufgrund von, je nach FPGA-Familie variierenden, Einstellungsmöglichkeiten des Xilinx Packers (*MAP*) zu deutlichen Unterschieden bei den betrachteten FPGA-Architekturen kam. Die Parameter wurden hierbei stets so gewählt, dass eine möglichst dichte Packung erzielt wurde. Die Auswertung der Leistungsfähigkeit zeigte hingegen einen geringen negativen Einfluss der DHHarMa-Implementierungen gegenüber den Xilinx-Implementierungen auf. Hierbei war ebenfalls ein Einfluss der FPGA-Architektur erkennbar.

Mit dem, von der Europäischen Weltraumorganisation (ESA) geförderten Projekt DRPM, wurde in Zusammenarbeit mit internationalen Partnern ein Demonstrator für dynamisch rekonfigurierbare Verarbeitungsmodulare mit einer skalierbaren Systemarchitektur entwickelt. Der Demonstrator stellte aufgrund dieser Systemarchitektur eine ideale Evaluationsplattform dar, in der Einleitung der Dissertation aufgeführten, Ziele dar. So konnte ein flexibles DPR-System durch die Anwendung von INDRA 2.0 realisiert werden. Durch die Verwendung unterschiedlicher, mittels DHHarMa erzeugter, DPR-Kommunikationsinfrastrukturen, wurden Varianten für die Partitionierung des dynamischen Bereichs vorgestellt.

Eine wesentliche Komponente des DPR-Systems stellte der *Self-Hosting Reconfiguration Controller* dar. Dieser realisierte einerseits einen internen Rekonfigurationskontrollierer mit Unterstützung von DPR inklusive Modulplatzierung durch Modifikation der Konfigurationsdatei und dem Zurücklesen des Konfigurationsspeichers. Andererseits wurde die Komponente durch die *Readback Scrubbing Unit* erweitert, welche eine Einbitfehlerkorrektur und Zweibitfehlererkennung im Konfigurationsspeicher implementiert und somit die Fehlertoleranz des DPR-Systems gegenüber SEE erhöht. Durch integrierte Fehlerstatistikmechanismen wurde eine Analyse des Systems zur Laufzeit ermöglicht. Zusätzlich wurde die Erstellung von Readback Scrubbing Schedules geschaffen, sodass die Fehlererkennung und -korrektur zum einen automatisch und zum anderen zur Laufzeit angepasst werden kann. Ein Vergleich mit existierenden Scrubber-Implementierungen zeigte, dass die RSU die geringste Fehlerkorrekturzeit besitzt.

Mit OLT(RE)², einem ebenfalls von der ESA geförderten Projekt, wurde ein Selbsttest für ein SRAM-basiertes FPGA bereitgestellt. Dieser Selbsttest ermöglicht eine Überprüfung einer FPGA-Fläche vor der Verwendung auf permanente Fehler in den Verdrahtungsressourcen (physikalische Leitungen: Stuck-at-0/1, PIPs: Stuck-off/on). Das beschriebene Verfahren wurde in dem *Techniques for radiation effects mitigation in ASICs and FPGAs handbook* der *European Cooperation for Space Standardization (ECSS)* aufgenommen. Zur Diagnose der Verdrahtungsressourcen wurde eine kleine, autonome Testschaltung entwickelt, welche mittels DPR platziert und das Testergebnis

durch Zurücklesen zur Laufzeit ermittelt wird. Zur bestmöglichen Ressourcenabdeckung einer Testregion wurden stets mehrere Konfigurationen der Testschaltung mit variierenden Verdrahtungsressourcen erzeugt. Die Funktionsweise der Testschaltung wurde auf dem PR-FPGA des DRPMs evaluiert. So wurde die Fehlererkennung durch Injektion von Stuck-at-0/1 Fehlern überprüft und die Anfälligkeit aller Testschaltungskomponenten durch Fehlerinjektion (Open Fehler) analysiert. Bei der Diagnose der Verdrahtungsressourcen ergab sich eine, je nach Fehlerart variierende Komplexität. Die Anzahl benötigter Testschaltungen zur Abdeckung der Stuck-at-0/1 Fehler der physikalischen Leitungen war um einen Faktor 45 geringer als die Anzahl benötigter Testschaltungen zur Abdeckung der Stuck-off/on Fehler der PIPs.

Durch die Kombination aller, in der Einleitung aufgeführten, Ziele der Dissertation wurde der DRPM Demonstrator zu einem flexiblen, adaptiven und insbesondere fehlertoleranten DPR-System erweitert, welches die Auswirkungen von SEEs auf den Konfigurationsspeicher durch Fehlererkennung und -korrektur behandelt sowie die Fehlererkennung von permanenten Fehlern in den Verdrahtungsressourcen des SRAM-basierten PR-FPGAs ermöglicht.

7.1 Ausblick

Die beiden vorgestellten Entwurfsabläufe basieren auf den Werkzeugen der Xilinx ISE, welche FPGAs bis zur 7-Serie unterstützen. Die aktuellen FPGA-Familien UltraScale und UltraScale+ werden nur von der neueren Xilinx Software Vivado unterstützt. Für XDL-basierte CAD-Werkzeuge, wie die Softwarekomponenten von DHHarMa und INDRA 2.0, hat die Software-Umstellung eine weitreichende Konsequenz, da XDL von der Xilinx Software Vivado nicht mehr unterstützt wird und somit die Schnittstelle, in Form einer XDL-Design-Datei, zur Xilinx Software entfernt wurde. Eine Interaktion mit der neuen IDE für externe CAD-Werkzeuge ist nur über eine in Vivado eingebaute Schnittstelle mittels Tcl möglich. Für die Unterstützung aktueller FPGA-Familien ist somit eine Adaptierung der Softwarekomponenten nötig. Eine Übersicht über die Xilinx Vivado Software sowie das Framework *Tincr* mit quelloffenen Tcl-Scriptsammlungen wird im Anhang B.1 gegeben. Das letztgenannte Framework bietet hierbei die Möglichkeit einer Portierung XDL-basierter CAD-Werkzeuge. Die Erstellung eines homogenen Hard-Makros mittels DHHarMa wäre anschließend ebenfalls für die neuen Xilinx FPGA-Familien möglich.

Für eine vollständige Portierung der INDRA 2.0 Softwarekomponenten müsste der zweite wichtige Schritt zur Erstellung eines flexiblen DPR-Systems, die Entkopplung von statischen und dynamischen Komponenten mittels eines Nachbearbeitungsschritts, ebenfalls angepasst werden. Wie beschrieben, erfolgt der Nachbearbeitungsschritt in Form einer Modifikation des statischen Designs durch ein Skript für das ISE Softwarewerkzeug FPGA Editor. Da sich diese Art der Manipulation an einem Design in Vivado über Tcl-Kommandos durchführen lassen, müsste die existierende Software um die Erstellung eines Tcl-Skript-Generator erweitert werden. Neben diesen Portierungsschrit-

ten ist eine weitere Automatisierung des INDRA 2.0 Ablaufs vorstellbar. Hierbei wäre insbesondere eine GUI-unterstützte Erstellung einer Systempartitionierung sowie eine automatische Parametrierung des *PSReRouters* wünschenswert.

Bezüglich der Verfahren zur Erhöhung der Fehlertoleranz sind ebenfalls Portierungen auf neue FPGA-Familien denkbar. Für die SEE-Fehlererkennung bedeutet dies eine Anpassung des implementierten, adaptiven Readback Scrubbings. Hierbei sollte jedoch auch der, je nach FPGA-Familie, deutlich gewachsene Funktionsumfang der Xilinx Implementierungen mit einer nun vollständig im FPGA-Bauteil realisierten Hardwarelösung beachtet werden. Feingranulare und für FPGA-Bereiche individuell einstellbare Scrubbingraten wie bei dem SHRC werden hierbei allerdings nicht unterstützt.

Bei der Erkennung permanenter Fehler in der Verdrahtungsinfrastruktur ist insbesondere eine höhere Ressourcenabdeckung wünschenswert. Dies könnte zum einen durch die Verwendung (Taktregions-) überlappender Testschaltungen und zum anderen durch neuartige Testschaltungsrealisierungen für die aktuell nicht unterstützten Hard-IP-Cores geschehen. Weiterhin ist eine Minimierung der Anzahl benötigter Testschaltungen erstrebenswert, da sich dies sowohl auf den Speicherbedarf als auch auf die Ausführungszeit auswirkt. Hierbei könnte die Homogenität in der FPGA-Struktur abermals ausgenutzt werden. Durch die Erstellung von Testschaltungen, welche an verschiedene FPGA-Positionen platziert werden können, könnte durch das DPR-Konzept der Modulumpplatzierung die Gesamtanzahl benötigter Testschaltungen reduziert werden.

Abbildungsverzeichnis

2.1	Kategorisierung der Bauteilgruppen basierend auf der Einteilung der Fachgruppe Kognitronik und Sensorik	8
2.2	Anordnung der Basisblöcke eines Xilinx FPGAs	12
2.3	Allgemeine Übersicht über die CLB-internen Komponenten	13
2.4	Anbindung der Basisblöcke an die Switch Matrix eines Xilinx FPGAs	15
2.5	Klassifizierung von FPGAs auf Basis der Konfigurierbarkeit	18
3.1	Dynamisch partielle Rekonfiguration durch einen internen (links) und externen (rechts) Rekonfigurationskontroller	22
3.2	Visualisierung der Vorteile des Zeitmultiplex-Betriebs und der Verwendung eines kleineren FPGAs	23
3.3	Gegenüberstellung des Ressourcenbedarfs eines statischen Systems und DPR-Systemen	26
3.4	Darstellung des Aufbaus eines CLBs der Virtex-4-Familie	32
3.5	Darstellung des Aufbaus eines CLBs der Virtex-5-Familie	34
3.6	Interner Aufbau der LUT ab der Virtex-5-Familie	34
3.7	Darstellung des Aufbaus eines CLBs der Spartan-6-Familie	37
3.8	Entwicklung der FPGA-Ressourcen bei den Generationswechseln	41
3.9	Aufteilung eines Xilinx Virtex-4 Configuration Frames	43
3.10	Darstellung der <i>Frame Address</i> anhand des Xilinx Virtex-4 FX100	45
3.11	Signalbelegung der SelectMAP- und ICAP-Schnittstelle	47
3.12	Konfigurationsablauf eines Xilinx FPGAs	49
3.13	Unterschied in der Stromaufnahme zwischen einem direkten Überschreiben und einem vorherigen Löschen der Konfiguration	54
3.14	Link Makros zur Anbindung von PR-Modulen	58
3.15	Weitere Link Makros zur Anbindung von PR-Modulen	59
3.16	Eingebettetes Makro mit definierten Anschlusspunkten an den Kommunikationskanal	60
3.17	1-Bit breite, bidirektionale, gemeinsam verwendete Kommunikationsinfrastruktur als eingebettetes Makros	60
3.18	Dedizierte, unidirektionale Signale im eingebetteten Makro	61
3.19	Xilinx EA PR Entwurfsablauf	65
3.20	Xilinx PR Entwurfsablauf	66
3.21	Entwurfsablauf eines ReCoBus DPR-System	68
3.22	Entwurfsablauf eines GoAhead DPR-Systems	70

3.23 Entwurfsablauf eines OpenPR DPR-Systems	72
3.24 Entwurfsablauf eines DREAMS DPR-Systems	74
3.25 INDRA – I ntegrated D esign F low for R econfigurable A rchitectures . .	76
3.26 INDRA 2.0 – I ntegrated D esign F low for R econfigurable A rchitectures	78
3.27 Erstellung eines Bus-Makros mit der Software <i>X-CMG</i>	79
3.28 Konfigurationsdatei des Xilinx <i>PR</i> Entwurfsablaufs	82
3.29 Ordnerstruktur des Xilinx <i>PR</i> Entwurfsablaufs	83
3.30 Ausschnitte eines <i>FPGA Editor</i> Skripts zur Neuverdrahtung von statischen Netzen	86
4.1 FPGA-basierte TDC-Implementierung durch Verwendung einer Carry Chain	93
4.2 Entwurfsablauf von DHHarMa zur Erzeugung von homogenen Hard- Makros	94
4.3 Xilinx-basiertes Frontend von DHHarMa zur Erzeugung von homogenen Hard-Makro	94
4.4 DHHarMa-Backend zur Erzeugung von homogenen Hard-Makros . .	96
4.5 Ausschnitte des XDL-Reports des Virtex-7 2000T FPGAs	98
4.6 Visualisierung der Komponenten des XDL-Reports einer Virtex-7 Switch- Matrix mit benachbartem CLB	99
4.7 Ausschnitt einer XDL-Design-Datei mit farblicher Kennzeichnung der Einflüsse der DHHarMa Werkzeuge	102
4.8 Architekturabbildung für ein bidirektionales, geteilt verwendetes Signal in einem eingebetteten Makro	106
4.9 Architekturabbildung für ein eindimensionales, dediziertes Signal in einem eingebetteten Makro	107
4.10 Architekturabbildung für ein zweidimensionales, dediziertes Signal in einem eingebetteten Makro	107
4.11 Partitionierung <i>LR</i> für 1D DPR-Kommunikationsinfrastrukturen	112
4.12 Partitionierung <i>L</i> für 1D DPR-Kommunikationsinfrastrukturen	112
4.13 Partitionierung <i>M</i> für 1D DPR-Kommunikationsinfrastrukturen	112
4.14 Partitionierungen für 2D DPR-Kommunikationsinfrastrukturen	113
4.15 Weitere Partitionierungen für 2D DPR-Kommunikationsinfrastrukturen	114
4.16 Datenstruktur der DXF des Kacheltyps CLB	119
4.17 Partitionierung eines Virtex-6 FPGAs mit acht rekonfigurierbaren Regio- nen und vier unterschiedlichen Regionstypen	125
4.18 Einlesen der Slice-Instanzen einer XDL-Design-Datei	126
4.19 Unterschiedliche Packung der Slices innerhalb zweier Regionen vom gleichen Regionstyp	126
4.20 Aufteilen eines Parts mit zwei belegten Registern	128
4.21 Gegenüberstellung einer inhomogenen (links) und homogenen (rechts) Verdrahtung	134

4.22 Entwurfsablauf des homogenen Routers	135
4.23 Vorgehensweise des Routing-Algorithmus	135
4.24 Generierte XDL-Ausgangsdatei im NMC-(rot) und NCD-(schwarz) Format	137
4.25 Auswertung der Platzierungsstrategien für verschiedene Partitionierungen in der Virtex-4-Architektur	146
4.26 Auswertung der Platzierungsstrategien für verschiedene Partitionierungen in der Artix-7-Architektur	147
4.27 Gegenüberstellung der Slice-Anzahl für homogene und inhomogene 1D DPR-Kommunikationsinfrastrukturen	150
4.28 Gegenüberstellung der FF-Packungsdichte für homogene und inhomogene 1D DPR-Kommunikationsinfrastrukturen	151
4.29 Gegenüberstellung der LUT-Packungsdichte für homogene und inhomogene 1D DPR-Kommunikationsinfrastrukturen	151
4.30 Gegenüberstellung der Slice-Anzahl für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur	154
4.31 Gegenüberstellung der FF-Dichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur	155
4.32 Gegenüberstellung der LUT-Dichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur	155
4.33 Gegenüberstellung der Slice-Anzahl für homogene und inhomogene 2D-Kommunikationsinfrastrukturen in der 7-Serie-Architektur	158
4.34 Gegenüberstellung der FF-Packungsdichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur	159
4.35 Gegenüberstellung der LUT-Packungsdichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur	159
4.36 Slice-Konfiguration der Eingangsbeschaltung der Verzögerungsleitung mit Verwendung der Carry Chain	164
4.37 Slice-Konfiguration der Weiterführung der Verzögerungsleitung mit Verwendung des Eingangs der Carry Chain	164
4.38 Implementierung einer homogenen und inhomogenen Verzögerungsleitung mit 40 zusammenschalteten CLBs (um 90° gedrehte Darstellung)	166
5.1 Magnetosphäre der Erde	171
5.2 Darstellung zur Definitionserklärung von Strahlung	173
5.3 Weibull Kurve des Konfigurationsspeichers eines Xilinx Virtex-5 QV FPGAs (XQR5VFX130)	174
5.4 Eindringen eines Partikels in ein Material und Wechselwirkung mit dem Material	175
5.5 Funneling Effect in einem CMOS-basierten Bauteil	176
5.6 Kategorisierung der physikalischen Strahlungseffekte	177
5.7 Kritische Bereiche für eine 6-Transistor-SRAM-Zelle	178

5.8	Zwei Upsets in zwei unterschiedlichen Wörtern (MCU)	179
5.9	Zwei Upsets in einem Wort (MBU)	179
5.10	Einfluss eines Upsets auf die Logik oder Verdrahtung eines Designs .	181
5.11	Entwicklung der Ein- und Mehrbitfehler in unterschiedlichen Xilinx FPGA-Familien	183
5.12	Bedingungen zur Speicherung eines SETs in einem Register	184
5.13	Kategorisierung der Verfahren nach dem NASA Fault Management Mo- dell	187
5.14	Vier Abstraktionsebenen der Verfahren zur Erhöhung der Fehlertoleranz	188
5.15	Blockdiagramm einer Architektur mit räumlichen Redundanz	189
5.16	Duplex-Architekturen in kombinatorischer und sequenzieller Logik .	189
5.17	Fehlererkennung in einer Duplex-Architektur	190
5.18	TMR-Architekturen in kombinatorischer und sequenzieller Logik . . .	190
5.19	Fehlererkennung und -korrektur in einer vollständigen TMR-Architektur	191
5.20	Architekturen mit zeitlicher Redundanz	192
5.21	Klassifizierung der Xilinx-Konfigurationsspeicherbits	199
5.22	Abstrakte Darstellung der Testschaltung zur permanenten Fehlererken- nung	202
5.23	Entscheidungsmatrix zur Wahl der Abschwächungsverfahren	207
5.24	Guard Gate zur SET-Filterung	208
5.25	Dual-Node Flip-Flops mit SET-Filter (Xilinx Virtex-5QV)	211
5.26	Entwicklung der absoluten FIT-Raten und der MTBF für aktuelle Xilinx FPGA-Familien	222
5.27	Übersicht der Strahlung im LHC im europäischen Kernforschungszen- trum CERN	224
5.28	Absolute Anzahl der verwendeten IC-Komponenten in aktuellen ESA Missionen	228
5.29	Trend der verwendeten IC-Komponenten in aktuellen ESA Missionen	228
6.1	Systemübersicht des DRPM-Systems	240
6.2	Übersicht der RAPTOR-X64-Architektur	242
6.3	Blockdiagramm des DB-V4s	243
6.4	Blockdiagramm des DB-SPACE	244
6.5	Basiskonfiguration des DRPMs mit zwei Verarbeitungsmodulen und ei- nem Kommunikationsmodul	245
6.6	Systemarchitektur der Basiskonfiguration des DRPMs mit zwei Verarbei- tungsmodulen und einem Kommunikationsmodul	245
6.7	Übersicht der Hauptkomponenten des Verarbeitungsmoduls	246
6.8	Übersicht der Hauptkomponenten des Kommunikationsmoduls	248
6.9	Übersicht des externen Rekonfigurationskontrollers	253
6.10	Verbindung zwischen dem SpaceWire-RTC (über die Mem-Bridge) und dem Flash-Speicher	255

6.11 Datenübertragungsrates für die Übertragung des Programmcodes für den SpaceWire-RTC	256
6.12 Übersicht über den Self-Hosting Reconfiguration Controller	257
6.13 Datenübertragungsrates für Local Link	261
6.14 Übersicht über den Multi-Port Memory Controller	264
6.15 Übersicht der Komponenten in der PR-Region	267
6.16 Übersicht über die PLB-NPI-EWB-Bridge	268
6.17 Signalverlauf eines Schreib- und Lesetransfers über PLB, EWB und WB	271
6.18 Datenübertragungsrates für NPI-Transfers	272
6.19 Übersicht der PR-Tile-Management-Komponenten	273
6.20 Übersicht der PR-FPGA-Komponenten	275
6.21 Alternative Partitionierung der PR-Region inklusive DPR-Kommunikationsinfrastruktur	276
6.22 Problematik von kreuzenden statischen Signalleitungen	277
6.23 Datenfluss innerhalb des DRPMs	278
6.24 Aufbau der Pipelined, Streaming I/O FFT-Architektur	280
6.25 Aufbau der Radix-4 FFT-Architektur	281
6.26 Aufbau der Radix-2 FFT-Architektur	282
6.27 Analyse der untersuchten FFT-Architekturen mit 1 024 Abtastpunkten	283
6.28 Analyse der untersuchten FFT-Architekturen mit 2 048 Abtastpunkten	284
6.29 Analyse der untersuchten FFT-Architekturen mit 4 096 Abtastpunkten	285
6.30 Gegenüberstellung der Auslastung einer PR-Tile, der Verarbeitungszeit und der Energie der untersuchten FFT-Implementierungen	286
6.31 Single Multiply-Accumulate Architektur	287
6.32 Systolic-MAC als Pipeline-Implementierung	288
6.33 Systolic-MAC als Multi-MAC-Implementierung	288
6.34 Transpose-MAC als Direktform	289
6.35 Transpose-MAC als Multi-MAC-Implementierung	289
6.36 Analyse der FIR-Filter-Implementierungen mit Tap-Größe 16	290
6.37 Analyse der FIR-Filter-Implementierungen mit Tap-Größe 64	291
6.38 Benutzeroberfläche zur Konfiguration des DRPMs	292
6.39 Frame ECC Portbelegung	294
6.40 Fehlerkategorien der Verdrahtungsressourcen	303
6.41 Switch-Matrix mit Hervorhebung von 6 PIPs und 7 physikalischen Leitungen	304
6.42 Test der Stuck-at Fehler der PW-UT a und d sowie des Stuck-off Fehlers des PIP-UT β	304
6.43 Test des Stuck-on Fehlers des PIP-UT β	305
6.44 Blockschaltbild der Testschaltung zur permanenten Fehlererkennung	305
6.45 Komponenten der Testschaltungen zur permanenten Fehlererkennung	306
6.46 Gesamtübersicht des OLT(RE) ² Testverfahrens	307
6.47 Entwurfsablauf von OLT(RE) ²	308

6.48 Kategorisierung der FPGA-Verdrahtungsressourcen	309
6.49 Ablauf der Testausführung von OLT(RE) ²	310
6.50 Switch Matrix mit, nach Test-Kategorie eingefärbten, Verdrahtungsressourcen	312
A.1 Übersicht einer Xilinx Konfigurationsdatei	366
A.2 Exportier- und Importierzeiten für Vivado (DCP) und Tincr (TCP) checkpoints	374
A.3 Programmcode der simulierten Abkühlung	376
A.4 Umsetzung Encapsulated Wishbone zu Wishbone	377
A.5 Auswertung der Platzierungsstrategien für verschiedene Partitionierungen mit 8 bit Kommunikationsinfrastrukturen in der Virtex-4-Architektur	386
A.6 Auswertung der Platzierungsstrategien für verschiedene Partitionierungen mit 8 bit Kommunikationsinfrastrukturen in der Artix-7-Architektur	387
A.7 Messergebnisse der Datenübertragung des SHRC über den Local Link TX Kanal (ModelSim)	390
A.8 Messergebnisse der Datenübertragung des SHRC über Local Link TX (ChipScope)	390
A.9 Messergebnisse der Datenübertragung des SHRC über Local Link RX (ModelSim)	391
A.10 Messergebnisse der Datenübertragung des SHRC über den Local Link RX Kanal (ChipScope)	391
A.11 Erkennung und Korrektur eines Einbitfehlers auf dem PR-FPGA	392
A.12 Ablauf des Bootloaders	393
A.13 Heatmap der Testschaltungs-unabhängigen Analyse	395
A.14 Heatmap der Testschaltungs-abhängigen Analyse	395

Tabellenverzeichnis

2.1	Übersicht über aktuelle FPGA-Familien	17
3.1	Gegenüberstellung der Slice-Architekturen der FPGA-Familien	39
3.2	Gegenüberstellung der aktuellen Xilinx FPGA-Familien unter Betrachtung der maximal verfügbaren FPGA-Ressourcen	41
3.3	Entwicklung des <i>Configuration Frames</i> für aktuelle Xilinx FPGA-Familien	42
3.4	Anzahl der <i>Configuration Frames</i> pro Basisblock für einige Xilinx FPGA-Familien	44
3.5	(Bitweise) Kodierung der <i>Configuration Frame Address</i> für aktuelle Xilinx FPGA-Familien	44
3.6	Ein- und Ausgänge der Konfigurationsschnittstelle SelectMAP/ICAP .	48
3.7	Maximale Konfigurationsdatenraten für die Konfigurationsschnittstellen eines Xilinx FPGAs	48
3.8	Approximierte Konfigurationszeiten für den jeweils kleinsten und größten FPGA der untersuchten FPGA-Familien (mit Ausnahme UltraScale+)	49
3.9	Mögliche Startup-Phasen während der Konfiguration	51
3.10	Vergleich der Funktionen unterschiedlicher Kommunikationsmakrotypen	61
3.11	Gegenüberstellung grob- und feingranularer DPR-Systeme	89
4.1	Signalbelegung der Master Encapsulated Wishbone Bus (MEWB) Kommunikationsinfrastruktur	105
4.2	Signalbelegung der Slave Encapsulated Wishbone Bus (SEWB) Kommunikationsinfrastruktur	105
4.3	Komponenten der HDL-Bibliothek	109
4.4	Zusammensetzung der FPGA-Ressourcen für die 2D MEWB-Anschlussstelle in der statischen Region (<i>tile_base_2d</i>)	110
4.5	Zusammensetzung der FPGA-Ressourcen für die 2D MEWB-Anschlussstelle in der PR-Region (<i>tile_pr_2d</i>)	110
4.6	FPGA-Ressourcen der 1D DPR-Kommunikationsinfrastrukturen	116
4.7	FPGA-Ressourcen der 2D DPR-Kommunikationsinfrastrukturen	117
4.8	Speicherauswertung der DXF-V3 mit Darstellung des größten FPGAs einer jeden Unterfamilie aller unterstützten FPGA-Familien	139
4.9	Zeitauswertung der DXF-V3 mit Darstellung des größten FPGAs einer jeden Unterfamilie aller unterstützten FPGA-Familien	141

4.10 Zusammenfassung der Speicher- und Zeitauswertung aller unterstützter FPGA-Familien für die DXF-V3	142
4.11 Speicherauswertung der <i>FPGA Editor</i> Datenbank für verschiedene FPGAs unterschiedlicher FPGA-Familien	143
4.12 Zeitauswertung der <i>FPGA Editor</i> Datenbank für verschiedene FPGAs unterschiedlicher FPGA-Familien	144
4.13 Auslastung der PR-Region für homogene und inhomogene, 1D DPR-Kommunikationsinfrastrukturen in der Virtex-6-Architektur (LX75T)	152
4.14 Auswertung der Leistungsfähigkeit für homogene und inhomogene, 1D DPR-Kommunikationsinfrastrukturen in der Virtex-6-Architektur (LX75T)	153
4.15 Auslastung der PR-Region für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur (DB: LX15, EWB: FX100)	156
4.16 Auswertung der Leistungsfähigkeit für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur (DB: LX15, EWB: FX100)	157
4.17 Auslastung der PR-Region für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur (5x2 Partitionierung: Kintex-7 325T, andere Artix-7 100T)	160
4.18 Auswertung der Leistungsfähigkeit für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur (5x2 Partitionierung: Kintex-7 325T, andere Artix-7 100T)	161
4.19 Zusammenfassende Darstellung der Auswertung für die DPR-Kommunikationsinfrastrukturen der betrachteten Xilinx FPGA-Familien	162
4.20 Auswertung der FPGA-Ressourcen und der Leistungsfähigkeit für homogene und inhomogene Verzögerungsleitungen	165
5.1 Kenndaten der typischen Satellitenorbits	172
5.2 Anfälligkeit der FPGA-Technologien gegenüber Strahlungseffekte . .	185
5.3 Fehlerkorrekturkodierungen und exemplarische Anwendungen	193
5.4 Abgeschätzte Upset-Raten für drei FPGAs der aktuellsten Xilinx space-grade FPGA-Familien	212
5.5 Kenngrößen des RT-ZU19EG MPSoCs und zusätzlicher Vergleich mit dem Virtex-5QV-FPGA	213
5.6 Kenngrößen der strahlungstoleranten und -gehärteten FPGAs	216
5.7 Orte mit Höhenlage des Xilinx Rosetta Experiments	218
5.8 Ergebnisse zur atmosphärischen Strahlung und Alphastrahlung des Rosetta Experiments	219
5.9 Wirkungsquerschnitt und FIT/Mbit Ergebnisse der Strahlungstests in LANSCE	220
5.10 Daten zur Berechnung der absoluten FIT-Rate für Xilinx Virtex-6 FPGAs	221

5.11 Absolute FIT-Rate und MTBF für die jeweils kleinsten und größten FPGAs der aktuellen Xilinx FPGA-Familien	222
5.12 Aktuelle Weltraummissionen der ESA	227
5.13 Anzahl der Komponenten der Sentinel 2 Mission	229
5.14 SpaceCube Versionen und Missionen	231
6.1 Konfigurationsvarianten des DRPMs	251
6.2 Verzögerungen und Datenübertragungsraten für Schreib- und Lesetransfers auf den Flash-Speicher	253
6.3 Konfigurationsgeschwindigkeit und -zeit für die FPGAs des Kommunikationsmoduls	254
6.4 Konfigurationsdatenrate und -zeit für die vollständige und partielle Rekonfiguration des PR-FPGAs des Verarbeitungsmoduls	255
6.5 FPGA-Ressourcen des SHRCs	260
6.6 Datenübertragungsraten des SHRCs für die unterstützten Funktionen	261
6.7 DPR- und Readback-Zeiten für typische Konfigurationsdateigrößen .	262
6.8 Gegenüberstellung des externen und internen Rekonfigurationskontrollers im DRPM	262
6.9 FPGA-Ressourcen des MPMCs	266
6.10 Datenübertragungsraten des MPMCs	266
6.11 FPGA-Ressourcen der PLB-NPI-EWB-Bridge	270
6.12 Maximale NPI-Datenübertragungsraten für verschiedene NPI-Burst-Größen	272
6.13 FPGA-Ressourcenübersicht bei Verwendung verschiedener 1D und 2D DPR-Kommunikationsinfrastrukturen für beide EWB-Protokolltypen	274
6.14 FPGA-Ressourcenübersicht bei Verwendung der eindimensionalen Kommunikationsinfrastruktur	279
6.15 FPGA-Ressourcen der Doppelpuffertypen der DRPM-Anwendungen .	279
6.16 Ergebnisse der Auswertung der Verarbeitungszeit, der Ressourcen-Auslastung und der Leistung/Energie der FFT-Implementierungen	286
6.17 Ergebnisse der Auswertung des Datendurchsatzes, der Ressourcenauslastung der PR-Tile und der Anzahl benötigter PR-Tiles der FIR-Filter-Implementierungen	291
6.18 Auswertung des Frame ECC Syndromwerts (SW)	294
6.19 Syndromwertauswertung eines Einbitfehlers in einem Paritätbit . . .	295
6.20 Übersicht über den Ressourcenbedarf der wesentlichen Komponenten des PR-FPGAs und die daraus resultierenden Scrubbing-Eigenschaften	298
6.21 Charakteristische Parameter für die Scrubbing-Funktionalität des SHRCs	299
6.22 Vergleich von Virtex-4 Scrubber-Implementierungen	299
6.23 Teststimuli des TPG für die 8-NUT Testschaltung	307
6.24 Kategorisierung der Verdrahtungsressourcen einer PR-FPGA-Taktregion	311

6.25 Ressourcenbedarf der OLT(RE) ² Testschaltung	313
6.26 Testabdeckung der Verdrahtungsressourcen einer Taktregion des PR-FPGAs	313
6.27 Speicher- und Zeitauswertung der Schritte zur Erstellung der OLT(RE) ² Testschaltungen	314
6.28 Ergebnisse der <i>Open</i> -Fehlerinjektion auf dem PR-FPGA	316
A.1 Konfigurationssequenz eines Virtex-4 FPGAs	367
A.2 Konfigurationssequenz für die dynamisch partielle Rekonfiguration eines Virtex-4 FPGAs	368
A.3 Gegenüberstellung der Konfigurationssequenzen für eine vollständige und eine partielle Konfiguration eines Virtex-4 FPGAs	369
A.4 Kodierung des Konfigurationsmodus	370
A.5 Laufzeit der Extraktion durch <code>write_xdlrc</code> für die größten FPGAs der 7-Serie-Unterfamilien	373
A.6 Speicherauswertung der zweiten DXF mit Darstellung des größten FPGAs einer jeden Unterfamilie aller unterstützten FPGA-Familien	375
A.7 Zeitauswertung der zweiten DXF mit Darstellung des größten FPGAs einer jeden Unterfamilie aller unterstützten FPGA-Familien	375
A.8 Zusammensetzung der FPGA-Ressourcen für die eindimensionale SEWB-Anschlussstelle in der statischen Region (<i>tile_base</i>)	378
A.9 Zusammensetzung der FPGA-Ressourcen für die eindimensionale SEWB-Anschlussstelle in der PR-Region (<i>tile_pr</i>)	378
A.10 Zusammensetzung der FPGA-Ressourcen für die eindimensionale MEWB-Anschlussstelle in der statischen Region (<i>tile_base</i>)	379
A.11 Zusammensetzung der FPGA-Ressourcen für die eindimensionale MEWB-Anschlussstelle in der PR-Region (<i>tile_pr</i>)	379
A.12 Zusammensetzung der FPGA-Ressourcen für die zweidimensionale SEWB-Anschlussstelle in der statischen Region (<i>tile_base_2d</i>)	380
A.13 Zusammensetzung der FPGA-Ressourcen für die zweidimensionale SEWB-Anschlussstelle in der PR-Region (<i>tile_pr_2d</i>)	380
A.14 FPGA-Ressourcen der 1D DB-Kommunikationsinfrastrukturen	381
A.15 FPGA-Ressourcen der 1D SEWB-Kommunikationsinfrastrukturen	382
A.16 FPGA-Ressourcen der 1D MEWB-Kommunikationsinfrastrukturen	383
A.17 FPGA-Ressourcen der 2D DB-Kommunikationsinfrastrukturen	384
A.18 FPGA-Ressourcen der 2D SEWB-Kommunikationsinfrastrukturen	385
A.19 FPGA-Ressourcen der 2D MEWB-Kommunikationsinfrastrukturen	385
A.20 Auswertung der Platzierungsstrategien für verschiedene Partitionierungen in der Virtex-4-Architektur	386
A.21 Auswertung der Platzierungsstrategien für verschiedene Partitionierungen in der Artix-7-Architektur	387

A.22 Auswertung der Packungsdichte für homogene und inhomogene 1D DPR-Kommunikationsinfrastrukturen in der Virtex-6-Architektur (LX75T)	388
A.23 Auswertung der Packungsdichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur (5x2 Partitionierung: Kintex-7 325T, andere Artix-7 100T)	388
A.24 Auswertung der Packungsdichte für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur (DB: LX15, EWB: FX100)	389
A.25 Verarbeitungszeit, FPGA-Ressourcenbedarf und Leistung der FFT-Implementierungen	394
A.26 Datendurchsatz, FPGA-Ressourcenbedarf und Leistung der FIR-Filter-Implementierungen	394

Abkürzungsverzeichnis

ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
AHB	Advanced High-performance Bus
ALICE	A Large Ion Collider Experiment
ALM	Adaptive Logic Module
AMBA	Advanced Microcontroller Bus Architecture
AMiRo	Autonomous Mini Robot
APB	Advanced Peripheral Bus
API	Application Programming Interface
ARM	Acorn RISC Machine
ASIC	Application-Specific Integrated Circuits
ASMBL	Advanced Silicon Modular Block
ATLAS	A Toroidal LHC Apparatus
AXI	Advanced eXtensible Interface
BPI	Byte Peripheral Interface
BRAM	Block Random-Access Memory
CDMA	Central Direct Memory Access
CE	Cumulative Effect
CFE	Cibola Flight Experiment
CLB	Configurable Logic Block
CMS	Compact Muon Solenoid
COTS	Commercial Off-The-Shelf
CPLD	Complex Programmable Logic Device
CRC	Cyclic Redundancy Check
CREME	Cosmic Ray Effects on MicroElectronics
DB	Daughter Board
DCE	Domain Crossing Event
DCM	Digital Clock Manager
DD	Displacement Damage
DHHarMa	Design Flow for Homogeneous Hard Macros
DICE	Dual Interlocked storage Cell

DiRT	D irected RouT ing
DLL	D elay- L ocked L oop
DLR	D eutschen Z entrum für L uft- und R aumfahrt
DMR	D ual M odular R edundancy
DPCU	D ynamic P rocessing C ontrol U nit
DPR	D ynamisch P artielle R ekonfiguration
DREAMS	D ynamically R econfigurable E mdeded P latforms for N etworked C ontext- A ware M ultimedia S ystems
DRP	D ynamic R econfiguration P ort
DRPM	D ynamically R econfigurable P rocessing M odule
DSP	D igital S ignal P rocessor
DVF	D evice V ulnerability F actor
DWC	D uplicate W ith C ompare
DXF	D atenbank für X ilinx F PGAs
EAPR	E arly A ccess P artial R econfiguration
ECC	E rror- C orrecting C odes
ECU	E lectronic C ontrol U nit
EDAC	E rror D etection A nd C orrection
EDK	E mbded D evelopment K it
EEPROM	E lectrically E rasable P rogrammable R ead- O nly M emory
ELF	E xecutable and L inking F ormat
ESA	E uropean S pace A gency
ESD	E lectro S tatic D ischarge
EWB	E ncapsulated W ishbone B us
FDIR	F ault D etection, I solation and R ecovery
FF	F lip F lop
FFT	F ast F ourier T ransform
FG-MOSFET	F loating- G ate M etal- O xide- S emiconductor F ield- E ffect- T ransistor
FIR	F inite I mpulse R esponse
FIT	F ailure I n T ime
FOBP	F raunhofer O n- B oard P rocessor
FPGA	F ield P rogrammable G ate A rray
GCC	G NU C ompiler C ollection
GCR	G alactic C osmic R ay
GEO	G eostationary E arth O rbital
GUI	G raphical U ser I nterface
HDL	H ardware D escription L anguage
HEO	H ighly E lliptical O rbital

HEP	High-Energy Physics
HMAC	Hashed Message Authentication Code
HPC	High-Performance Computing
HPRC	High-Performance Reconfigurable Computing
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
INDRA	INtegrated Design Flow for Reconfigurable Architectures
IOB	Input Output Block
IP	Intellectual Property
ISE	Integrated Software Environment
JTAG	Joint Test Action Group
LAB	Logic Array Block
LCA	Logic Cell Array
LEO	Low Earth Orbit
LET	Linear Energy Transfer
LHC	Large Hadron Collider
LUT	Look-Up Table
MAC	Multiply-ACcumulate
MBU	Multiple Bit Upset
MCU	Multiple Cell Upset
MEO	Medium Earth Orbit
MGT	Multi-Gigabit Transceiver
MIPS	Million Instructions Per Second
MPMC	Multi-Port Memory Controller
MRM	Mission Response Module
MSB	Most Significant Bit
MTBF	Mean Time Between Failure
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
NASA	National Aeronautics and Space Administration
NCD	Native Generic Database
NGC	Native Generic Circuit
NUT	Net Under Test
OLT(RE) ²	On-Line on-demand Testing approach for permanent Radiation Effects in REconfigurable systems

ORA	Output Response Analyzer
OTN	Optical Transport Network
OTP	One-Time Programmable
PCAP	Processor Configuration Access Port
PIP	Programmable Interconnect Point
PL	Programmable Logic
PLB	Processor Local Bus
PLD	Programmable Logic Devices
PLL	Phase-Locked Loop
PS	Processing System
R3TOS	Reliable Reconfigurable Real-time Operating System
RAM	Random-Access Memory
ReconOS	Reconfigurable Operating System
RH	Radiation Hardend
RHBD	Radiation Hardening By Design
RHBP	Radiation Hardening By Process
RHBS	Radiation Hardened By Software
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RoRA	Reliability-oriented place and Route Algorithm
RT	Radiation Tolerant
RTL	Register Transfer Level
SAA	South Atlantic Anomaly
SBU	Single-Bit Upset
SCR	Solar Cosmic Ray
SDMA	Soft Direct Memory Access
SDR	Software-Defined Radio
SECEDED	Single Error Correction Double Error Detection
SEE	Single Event Effect
SEFI	Single-Event Functional Interrupt
SEGR	Single-Event Gate Ruptur
SEL	Single-Event Latchup
SEM	Soft Error Mitigation
SET	Single-Event Transient
SEU	Single-Event Upset
SHRC	Self-Hosting Reconfiguration Controller
SM	Switch Matrix
SoC	System-on-a-Chip
SODIMM	Small Outline Dual In-line Memory Module

SPI	Serial Peripheral Interface
SPLD	Simple Programmable Logic Device
SRAM	Static Random-Access Memory
SSI	Stacked Silicon Interconnect
STAR	STatic AnalyseR
STL	Standard Template Library
STT	Sub-Tile Type
TCD	Test Circuit Dependent
TCI	Test Circuit Independent
Tcl	Tool command language
TDC	Time-to-Digital-Converter
TDM	Time Division Multiplexing
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
Torc	Tools for open reconfigurable computing
TPG	Test Pattern Generator
TRACE	Timing Reporter And Circuit Evaluator
TT	Tile Type
UCF	User Constraint File
ULA	Ultra-Low-Alpha
VPR	Versatile Place and Route
XCF	Xilinx Constraint File
XDL	Xilinx Design Language
XPS	Xilinx Platform Studio
XST	Xilinx Synthesis Technology

Referenzen

- [1] 1-CORE Technologies. *FPGA Architectures Overview*. URL: <https://www.pdx.edu/nanogroup/sites/www.pdx.edu.nanogroup/files/FPGA-architecture.pdf> (siehe S. 16).
- [2] P. Adell, G. Allen, G. Swift und S. McClure. „Assessing and mitigating radiation effects in Xilinx SRAM FPGAs“. In: *European Conference on Radiation and its Effects on Components and Systems, RADECS (2008)*, S. 418–424. DOI: 10.1109/RADECS.2008.5782755 (siehe S. 204).
- [3] A. Agne, M. Happe, A. Keller, E. Lubbers, B. Plattner, M. Platzner und C. Plessl. „ReconOS: An operating system approach for reconfigurable computing“. In: *IEEE Micro* 34.1 (2014), S. 60–71. DOI: 10.1109/MM.2013.110 (siehe S. 62).
- [4] G. Allen, G. Swift und C. Carmichael. „Virtex-4 VQ static SEU characterization summary“. In: *Jet Propulsion Laboratory Publication (2008)* (siehe S. 296).
- [5] Altera. *Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs (WP-01137-1.0)*. 2010 (siehe S. 22–23).
- [6] Altera. *Introduction to Single-Event Upsets (WP-01206-1.0)*. 2013 (siehe S. 176).
- [7] Altera. *White Paper FPGA Run-Time Reconfiguration : Two Approaches (v1.0)*. 2008 (siehe S. 29).
- [8] S. Asano, T. Maruyama und Y. Yamaguchi. „Performance comparison of FPGA, GPU and CPU in image processing“. In: *FPL 09: 19th International Conference on Field Programmable Logic and Applications* October (2009), S. 126–131. DOI: 10.1109/FPL.2009.5272532 (siehe S. 11).
- [9] Atmel. *ATF280F*. 2012 (siehe S. 214).
- [10] Atmel. *Rad Hard Reprogrammable FPGAs with FreeRAM AT40KELO40 AT40KFL040 AT40KELO40*. 2006 (siehe S. 213).
- [11] *AutoIt*. URL: <https://www.autoitscript.com/site/autoit/> (siehe S. 122).
- [12] J. R. Azambuja, C. Pilotto und F. L. Kastensmidt. „Mitigating soft errors in SRAM-based FPGAs by using large grain TMR with selective partial reconfiguration“. In: *European Conference on Radiation and its Effects on Components and Systems (RADECS) (2008)*, S. 288–293. DOI: 10.1109/RADECS.2008.5782729 (siehe S. 197).

- [13] M. Baradaran Tahoori. „Application-dependent diagnosis of FPGAs“. In: *International Conference on Test* (2004), S. 645–654. DOI: 10.1109/TEST.2004.1387002 (siehe S. 201).
- [14] H. J. Barnaby. „Total-Ionizing-Dose Effects in Modern CMOS Technologies“. In: *IEEE Transactions on Nuclear Science* 53.6 (2006), S. 3103–3121. DOI: 10.1109/TNS.2006.885952 (siehe S. 177, 180).
- [15] N. Battezzati, L. Sterpone und M. Violante. *Reconfigurable field programmable gate arrays for mission-critical applications*. Springer, 2011, S. 220. DOI: 10.1007/978-1-4419-7595-9 (siehe S. 15–16, 27, 169, 171–177, 180–181, 184, 186, 210, 213, 215).
- [16] L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, E. Schneider, H. Zhang, J. Henkel und H. J. Wunderlich. „Test strategies for reliable runtime reconfigurable architectures“. In: *IEEE Transactions on Computers* 62.8 (2013), S. 1494–1507. DOI: 10.1109/TC.2013.53 (siehe S. 202).
- [17] B. Bauss, V. Büscher, R. Degele, W. Ji, S. Moritz, A. Reiss, U. Schäfer, E. Simioni, S. Tapprogge und V. Wenzel. „An FPGA based Topological Processor prototype for the ATLAS Level-1 Trigger upgrade“. In: *Journal of Instrumentation* 7.12 (2012), S. C12007. DOI: 10.1088/1748-0221/7/01/C01067 (siehe S. 226).
- [18] C. Beckhoff, D. Koch und J. I. M. Torresen. „Design Tools for Implementing Self-Aware and Fault-Tolerant Systems on FPGAs“. In: *ACM Trans. Reconfigurable Technol. Syst.* 7.2 (2014), 14:1–14:23. DOI: 10.1145/2617597 (siehe S. 87–88).
- [19] C. Beckhoff, D. Koch und J. Torresen. „GoAhead: A Partial Reconfiguration Framework“. In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2012. DOI: 10.1109/FCCM.2012.17 (siehe S. 67, 69–71).
- [20] C. Beckhoff, D. Koch und J. Torresen. „Short-Circuits on FPGAs caused by Partial Runtime Reconfiguration“. In: *Field Programmable Logic and Applications (FPL)*. 2010, S. 596–601. DOI: 10.1109/FPL.2010.117 (siehe S. 54).
- [21] C. Beckhoff, D. Koch und J. Torresen. „The Xilinx Design Language (XDL): Tutorial and Use Cases“. In: *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. 2011, S. 1–8 (siehe S. 97).
- [22] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, A. Paccagnella, M. Rebaudengo, M. S. Reorda, M. Violante und P. Zambolin. „Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA“. In: *Proceedings - Design, Automation and Test in Europe Conference and Exhibition*. Bd. 1. 2004, S. 584–589. DOI: 10.1109/DATE.2004.1268908 (siehe S. 301).

- [23] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. A. LaBel, M. Friendlich, H. Kim und A. Phan. „Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis“. In: *IEEE Transactions on Nuclear Science* 55.4 (2008), S. 2259–2266. DOI: 10.1109/TNS.2008.2001422 (siehe S. 204, 295).
- [24] P. Bergsman. „Xilinx FPGA Blasted into Orbit“. In: *Xcell journal* 00 (2003) (siehe S. 228).
- [25] C. Bernardeschi, L. Cassano, M. G. C. A. Cimino und A. Domenici. „GABES: A genetic algorithm based environment for SEU testing in SRAM-FPGAs“. In: *Journal of Systems Architecture* 59.10 PART D (2013), S. 1243–1254. DOI: 10.1016/j.sysarc.2013.10.006 (siehe S. 201).
- [26] P. Bernardi, M. Sonza Reorda, L. Sterpone und M. Violante. „On the evaluation of SEU sensitiveness in SRAM-based FPGAs“. In: *International On-Line Testing Symposium (IOLTS)* (2004), S. 115–120. DOI: 10.1109/OLT.2004.1319668 (siehe S. 198).
- [27] V. Betz und J. Rose. „VPR: A new packing, placement and routing tool for FPGA research“. In: *Field Programmable Logic and Applications (FPL)*. 1997 (siehe S. 131–133).
- [28] *Boost C++ Libraries*. 2016. URL: www.boost.org (siehe S. 117).
- [29] B. Bridgford, C. Carmichael und C. W. Tseng. *Single-Event Upset Mitigation Selection Guide (XAPP987 v1.0)*. 2008 (siehe S. 207).
- [30] V. Briot. „AT40K FPGAs Platform For SPACE APPLICATIONS AT40K FPGA PLATFORM“. In: 2. *Space FPGA Users Workshop (SEFUW) - Präsentation*. 2014, S. 1–18 (siehe S. 213–214).
- [31] F. Brosser und E. Milh. „SEU Mitigation Techniques for Advanced Reprogrammable FPGA in Space“. Master Thesis. 2014 (siehe S. 194, 210).
- [32] *bzip2*. URL: <http://www.bzip.org/> (siehe S. 118).
- [33] M. Caffrey, K. Morgan, D. Roussel-Dupre, S. Robinson, A. Nelson, A. Salazar, M. Wirthlin, W. Howes und D. Richins. „On-orbit flight results from the reconfigurable cibola flight experiment satellite (CFESat)“. In: *Symposium on Field Programmable Custom Computing Machines (FCCM)* (2009), S. 3–10. DOI: 10.1109/FCCM.2009.22 (siehe S. 229–230).
- [34] M. Cannon, M. Wirthlin, A. Camplani, M. Citterio und C. Meroni. „Evaluating Xilinx 7 Series GTX Transceivers for Use in High Energy Physics Experiments Through Proton Irradiation“. In: *IEEE Transactions on Nuclear Science* 62.6 (2015), S. 2695–2702 (siehe S. 225–226).
- [35] C. Carmichael und C. Tseng. *Correcting single-event upsets in Virtex-4 FPGA configuration memory (XAPP1088 v1.0)*. 2009 (siehe S. 43, 45, 204, 258, 294, 296–297).

- [36] C. Carmichael und C. W. Tseng. *Correcting Single-Event Upsets in Virtex-4 Platform FPGA Configuration Memory (XAPP988 v1.0)*. 2008 (siehe S. 204, 258, 294, 296–297).
- [40] R. Cicalese, A. Aloisio, P. Branchini und R. Giordano. *FPGA implementation of a high-resolution time-to-digital converter*. 2007. DOI: 10.1109/NSSMIC.2007.4436379 (siehe S. 93).
- [41] A. Cilaro. „New techniques and tools for application-dependent testing of FPGA-based components“. In: *IEEE Transactions on Industrial Informatics* 11.1 (2015), S. 94–103. DOI: 10.1109/TII.2014.2370532 (siehe S. 201).
- [42] D. Cozzi. „Homogeneous communication router for Xilinx FPGAs“. Master thesis. 2010 (siehe S. 134).
- [43] D. Cozzi. „Run-time reconfigurable, fault-tolerant FPGA systems for space applications“. Diss. Universität Bielefeld, 2016 (siehe S. 30, 77, 85, 134, 249, 302, 309).
- [44] D. Cozzi, J. Hagemeyer, S. Korf, D. Jungewelter und M. Porrmann. *Benchmark Development and Performance Report v1.3 "FPGA Based Generic Module & Dynamic Reconfigurator"*. 2014 (siehe S. 252–255, 262, 272, 299, 390–391).
- [45] D. Cozzi, J. Hagemeyer, S. Korf, D. Jungewelter und M. Porrmann. *User Manual v6.1 "FPGA Based Generic Module & Dynamic Reconfigurator"*. 2014 (siehe S. 240–244, 249, 255, 257, 260, 264–272, 377).
- [49] W. Crochot. *Magnetosphere*. 2016. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/5/50/Structure_of_the_magnetosphere-en.svg/2000px-Structure_of_the_magnetosphere-en.png (siehe S. 171).
- [50] S. Danzeca. „An overview of FPGA use in the LHC accelerator and the CERN experiments“. In: 3. *SpacE FPGA Users Workshop (SEFUW) - Präsentation*. 2016. URL: <https://indico.esa.int/indico/event/130/session/14/contribution/73/material/slides/0.pptx> (siehe S. 224).
- [51] A. DeHon. „Reconfigurable Architectures for General-Purpose Computing“. Ph.D. Thesis. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1996 (siehe S. 15).
- [52] P. E. Dodd und L. W. Massengill. „Basic mechanisms and modeling of single-event upset in digital microelectronics“. In: *IEEE Transactions on Nuclear Science* 50 III.3 (2003), S. 583–602. DOI: 10.1109/TNS.2003.813129 (siehe S. 177–178).
- [54] B. F. Dutton und C. E. Stroud. „On-Line Single Event Upset Detection and Correction in Field Programmable Gate Array Configuration Memories“. In: *International Journal of Computer Applications* 17.2 (2010), S. 1–11 (siehe S. 294).

- [55] B. F. Dutton und C. E. Stroud. „Single Event Upset Detection and Correction in Virtex-4 and Virtex-5 FPGAs“. In: *International Conf. on Computers and Their Applications (ISCA)*. 2009, S. 57–62 (siehe S. 294, 299–300).
- [56] EADS-Astrium. „Solar Orbiter“. In: *Journal of Physics: Conference Series* 271. July (2011), S. 011004. DOI: 10.1088/1742-6596/271/1/011004 (siehe S. 233).
- [57] H. Engel. „Development of a radiation tolerant softcore CPU for SRAM based FPGAs“. Diploma Thesis. Universität Heidelberg, 2009 (siehe S. 177, 183–184).
- [58] J. Engel, K. S. Morgan, M. J. Wirthlin und P. S. Graham. „Predicting On-Orbit Static Single Event Upset Rates in Xilinx Virtex FPGAs“. In: *Military and Aerospace Programmable Logic Devices Conference (MAPLD)*. 2006 (siehe S. 170, 196).
- [59] *ESA STUDY CONTRACT REPORT ” FPGA Based Generic Module & Dynamic Reconfigurator ” DRPM Architectural Review and Design Assessment Report – DD-21* (siehe S. 241).
- [60] European Cooperation for Space Standardization. *Technique for radiation effects mitigation in ASICs and FPGAs handbook*. ESA Requirements und Standards Division, 2015 (siehe S. 172, 177–180, 184, 187–194).
- [61] U. Farooq, Z. Marrakchi und H. Mehrez. *Tree-based Heterogeneous FPGA Architectures*. 2012. DOI: 10.1007/978-1-4614-3594-5 (siehe S. 15–16).
- [62] C. Favi und E. Charbon. „A 17ps time-to-digital converter implemented in 65nm FPGA technology“. In: *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2009, S. 113–120 (siehe S. 93).
- [63] *FedSat - historical highlights*. URL: <https://www.itr.unisa.edu.au/projects/fedsat> (siehe S. 229).
- [64] T. Flatley. *SpaceCube : A Family Of Science Data Processors*. 2012 (siehe S. 231).
- [65] J. Fowers, G. Brown, P. Cooke und G. Stitt. „A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications“. In: *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays - FPGA ’12* (2012), S. 47. DOI: 10.1145/2145694.2145704 (siehe S. 11).
- [66] R. B. Gardenyes. „Trends and patterns in ASIC and FPGA use in space missions and impact in technology roadmaps of the European Space Agency“. Master Thesis. Delft University of Technology, 2012 (siehe S. 227–229).
- [68] E. Gerald. „Organization of Computer Systems-the Fixed Plus Variable Structure Computer“. In: *Western Joint IRE-AIEE-ACM Computer Conference*. 1960, S. 8. DOI: 10.1145/1460361.1460365 (siehe S. 11).

- [69] R. Glein. „BRAM Radiation Sensor for a Self-Adaptative SEU Mitigation“. In: 2. *SpacE FPGA Users Workshop (SEFUW) - Präsentation* (2014) (siehe S. 234–235).
- [70] R. Glein, F. Rittner, A. Becher, D. Ziener und A. Heuberger. „Reliability of Space-Grade vs . COTS SRAM-Based FPGA in N-Modular Redundancy“. In: *Adaptive Hardware and Systems (AHS)*. 2015. ISBN: 9781467375016 (siehe S. 210).
- [71] *GoAhead - A Partial Design Tool*. 2012. URL: <http://www.mn.uio.no/ifi/english/research/projects/cosrecos/goahead/> (siehe S. 71).
- [72] Grandviewresearch.com. *FPGA (Field-Programmable Gate Array) Market Analysis By Application (Automotive, Consumer Electronics, Data Processing, Industrial, Military And Aerospace, Telecom) And Segment Forecasts To 2020*. Techn. Ber. 2015. URL: <http://www.grandviewresearch.com/industry-analysis/fpga-market> (siehe S. 1).
- [73] R. Griessl, M. Peykanu, J. Hagemeyer, M. Porrman, S. Krupop, M. dem Berge, T. Kiesel und W. Christmann. „A Scalable Server Architecture for Next-Generation Heterogeneous Compute Clusters“. In: *IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2014*. IEEE, 2014 (siehe S. 20).
- [74] S. Habinc. *Suitability of reprogrammable FPGAs in space applications - Feasibility Report*. 2002 (siehe S. 191).
- [75] J. Hagemeyer, B. Kettelhoit, M. Koester und M. Porrman. „A Design Methodology for Communication Infrastructures on Partially Reconfigurable FPGAs“. In: *Field Programmable Logic and Applications (FPL)*. 2007, S. 331–338 (siehe S. 75–76).
- [76] J. Hagemeyer. „Homogene On-Chip Kommunikationsstrukturen für dynamisch rekonfigurierbare Systeme“. Diplomarbeit. Fachgruppe Schaltungstechnik, Universität Paderborn, 2005 (siehe S. 79, 104, 272–273).
- [79] J. Hagemeyer, B. Kettelhoit, M. Koester und M. Porrman. „Design of Homogeneous Communication Infrastructures for Partially Reconfigurable FPGAs“. In: *Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. CSREA Press, 2007 (siehe S. 59).
- [80] J. Heiner, N. Collins und M. Wirthlin. „Fault tolerant ICAP controller for high-reliable internal scrubbing“. In: *IEEE Aerospace Conference*. December 2007. 2008. DOI: 10.1109/AERO.2008.4526471 (siehe S. 299–300).
- [81] S. Herbrechtsmeier, U. Rückert und J. Sitte. „AMiRo – Autonomous Mini Robot for Research and Education“. In: *Advances in Autonomous Mini Robots*. Hrsg. von U. Rückert, J. Sitte und F. Werner. Springer, 2012. DOI: 10.1007/978-3-642-27482-4_12 (siehe S. 20).

- [82] I. Herrera-Alzu und M. López-Vallejo. „Design techniques for Xilinx Virtex FPGA configuration memory scrubbers“. In: *IEEE Transactions on Nuclear Science* 60.1 (2013), S. 376–385. DOI: 10.1109/TNS.2012.2231881 (siehe S. 194, 204–205).
- [83] C. Hu und S. Zain. *NSEU Mitigation in Avionics Applications (XAPP1073 v1.0)*. 2010 (siehe S. 182, 203, 206).
- [84] K. Huey. „Xilinx CN Package Qualification Updates for MRQW 2015“. In: *Microelectronics Reliability & Qualification Working Meeting (MRQW)*. 2015 (siehe S. 211).
- [85] K. Huey. „Xilinx Virtex-5QV Update and Space Roadmap“. In: *3. Space FPGA Users Workshop (SEFUW) - Präsentation*. 2016. URL: <https://indico.esa.int/indico/event/130/session/11/contribution/54/material/slides/0.pdf> (siehe S. 212–213).
- [86] B. J. Hussein und G. Swift. *Mitigation Single-Event Upsets (WP395 v1.1)*. 2015 (siehe S. 182–183, 200, 218, 220).
- [87] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, R. Torrego, I. Martinez, T. Arslan und J. Perez. „R3TOS: A novel reliable reconfigurable real-time operating system for highly adaptive, efficient, and dependable computing on FPGAs“. In: *IEEE Transactions on Computers* 62.8 (2013), S. 1542–1556. DOI: 10.1109/TC.2013.79 (siehe S. 62).
- [88] A. Jacobs, G. Cieslewski, A. D. George und A. N. N. Gordon-ross. „Reconfigurable Fault Tolerance : A Comprehensive Framework for Reliable and Adaptive FPGA-Based Space Computing“. In: *ACM Trans. Reconfigurable Technol. Syst.* 5.4 (2012), 21:1–21:30. DOI: 10.1145/2392616.2392619 (siehe S. 210).
- [89] B. Jeffres. *Intel Begins Shipping Xeon Chips With FPGA Accelerators*. 2016. URL: <http://www.eweek.com/servers/intel-begins-shipping-xeon-chips-with-fpga-accelerators.html> (siehe S. 11).
- [90] J. Johnson, W. Howes, M. Wirthlin, D. L. McMurtrey, M. Caffrey, P. Graham und K. Morgan. „Using duplication with compare for on-line error detection in FPGA-based designs“. In: *IEEE Aerospace Conference*. 2008. DOI: 10.1109/AERO.2008.4526470 (siehe S. 197).
- [92] J. Jylänki. *RectangleBinPack*. URL: <https://github.com/juj/RectangleBinPack> (siehe S. 130).
- [93] H. Kalte, G. Lee, M. Porrmann und U. Rückert. „REPLICA : A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems“. In: *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2005 (siehe S. 259).
- [94] C. Kao. „Benefits of partial reconfiguration“. In: *Xcell journal* 55 (2005), S. 65–67 (siehe S. 29).

- [95] F. Kesel und R. Bartholomä. *Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs: Einführung mit VHDL und SystemC*. Oldenbourg, 2009. ISBN: 9783486589764 (siehe S. 8–10, 15–16, 19).
- [96] B. Kettelhoit. „Architektur und Entwurf dynamisch rekonfigurierbarer FPGA-Systeme“. Dissertation. Fachgruppe Schaltungstechnik, Universität Paderborn, 2009 (siehe S. 25–27).
- [97] H. Klar, T. Noll, H. Henke und U. Rückert. *Integrierte Digitale Schaltungen*. 2015. ISBN: 9783540406006 (siehe S. 8).
- [98] D. Koch. *GoAhead Tutorial Version 1.0*. 2012 (siehe S. 63, 69).
- [99] D. Koch. *ReCoBus-Builder Documentation & User Guide*. 2009 (siehe S. 68–69).
- [100] D. Koch, C. Beckhoff und J. Teich. „ReCoBus-Builder - A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs“. In: *Field Programmable Logic and Applications (FPL)*. 2008, S. 119–124 (siehe S. 67).
- [101] M. Koester, W. Luk, J. Hagemeyer, M. Pormann und U. Ruckert. „Design Optimizations for Tiled Partially Reconfigurable Systems“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19.6 (2011), S. 1048–1061. DOI: 10.1109/TVLSI.2010.2044902 (siehe S. 56–61).
- [102] S. Korf. *Effiziente Datenstruktur für Xilinx FPGAs*. Studienarbeit. 2010 (siehe S. 97, 101, 118–120).
- [103] S. Korf. „VHDL-basierte Erzeugung homogener Strukturen für Xilinx FPGAs“. Diplomarbeit. Fachgruppe Schaltungstechnik, Universität Paderborn, 2011 (siehe S. 31, 35, 39, 94, 97, 118–120, 123, 125–129, 134, 137, 164, 375–376).
- [107] B. J. LaMeres, S. Harkness, M. Handley, P. Moholt, C. Julien, T. Kaiser, D. Klumpar, K. Mashburn, L. Springer und G. A. Crum. „RadSat - Radiation Tolerant SmallSat Computer System“. In: *Annual AIAA/USU Conference on Small Satellites*. 2015 (siehe S. 210, 233).
- [108] T. Lange, B. Fiethe, H. Michel und H. Michalik. „From Rosetta to Current Developments Using FPGAs for Scientific Space Missions“. In: *2. Space FPGA Users Workshop (SEFUW) - Präsentation*. 2014 (siehe S. 234).
- [109] M. Lanuzza, P. Zicari, F. Frustaci, S. Perri und P. Corsonello. „Exploiting Self-Reconfiguration Capability to Improve SRAM-based FPGA Robustness in Space and Avionics Applications“. In: *ACM Trans. Reconfigurable Technol. Syst.* 4.1 (2010), S. 1–22. DOI: 10.1145/1857927.1857935 (siehe S. 203).
- [110] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, B. Hutchings und M. Wirthlin. *RapidSmith: A Library for Low-level Manipulation of Partially Placed-and-Routed FPGA Designs*. 2014. URL: <http://rapidsmith.sourceforge.net/doc/TechReportAndDocumentation.pdf> (siehe S. 43).

- [111] R. Le. „Soft Error Mitigation Using Prioritized Essential Bits (XAPP538 v1.0)“. In: (2012), S. 1–11 (siehe S. 199–200).
- [112] U. Legat, A. Biasizzo und F. Novak. „SEU recovery mechanism for SRAM-Based FPGAs“. In: *IEEE Transactions on Nuclear Science* 59.5 PART 3 (2012), S. 2562–2571. DOI: 10.1109/TNS.2012.2211617 (siehe S. 299–300).
- [113] P. Leray, Y. Louet und J. Palicot. „Moving a Processing Element from Hot to Cool Spots : Is This an Efficient Method to Decrease Leakage Power Consumption in FPGAs“. In: *Green Communications - Theoretical Fundamentals, Algorithms and Applications*. 2012. Kap. 8, S. 197–220 (siehe S. 27).
- [114] A. Lesea, S. Drimer, J. J. Fabula, C. Carmichael und P. Alfke. „The rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs“. In: *IEEE Transactions on Device and Materials Reliability* 5.3 (Sep. 2005), S. 317–328. DOI: 10.1109/TDMR.2005.854207 (siehe S. 182, 217, 219).
- [115] S. Liu, R. N. Pittman, A. Forin und J.-L. Gaudiot. „Achieving energy efficiency through runtime partial reconfiguration on reconfigurable systems“. In: *ACM Transactions on Embedded Computing Systems* 12.3 (2013), S. 1–21. DOI: 10.1145/2442116.2442122 (siehe S. 27–29).
- [116] P. Lysaght und J. Dunlop. „Dynamic Reconfiguration of FPGAs“. In: *International Workshop on Field Programmable Logic and Applications on More FPGAs*. 1994, S. 82–94. ISBN: 0-9518453-1-4 (siehe S. 18).
- [118] D. G. Mavis und P. H. Eaton. „Soft error rate mitigation techniques for modern microcircuits“. In: *IEEE International Reliability Physics Symposium Proceedings*. 2002, S. 216–225. DOI: 10.1109/RELPHY.2002.996639 (siehe S. 192).
- [119] D. G. Mavis und P. H. Eaton. „SEU and SET Mitigation Techniques for FPGA Circuit and Configuration Bit Storage Design“. In: *Aerospace. Military and Aerospace Programmable Logic Devices (MAPLD)*, 2000 505. 2000, S. 1–15 (siehe S. 183).
- [120] J. McCollum. „ASIC versus antifuse FPGA reliability“. In: *Aerospace conference, 2009 IEEE*. 2009, S. 1–11. DOI: 10.1109/AERO.2009.4839526 (siehe S. 181).
- [121] J. McCollum. *Radiation tolerant flash FPGA*. 2001. URL: <http://www.google.com/patents/US6324102> (siehe S. 181).
- [122] J. Meyer, J. Noguera, M. Hübner, L. Braun, O. Sander, R. M. Gil, R. Stewart und J. Becker. „Fast Start-up for Spartan-6 FPGAs using Dynamic Partial Reconfiguration“. In: *Design, Automation Test in Europe (DATE)*. 2011, S. 1–6. DOI: 10.1109/DATE.2011.5763244 (siehe S. 29).
- [123] Microsemi. *Mitigation of Radiation Effects in RTG4 Radiation- Tolerant Flash FPGAs*. 2015 (siehe S. 183, 215).

- [124] Microsemi. *Radiation-Tolerant FPGAs*. 2015. URL: http://www.microsemi.com/document-portal/doc_download/131352-radiation-tolerant-fpgas-catalog (siehe S. 214–215).
- [125] Microsemi. *RTSX-SU Radiation-Tolerant FPGAs (UMC) Designed for Space RTSX-SU Device Status*. 2012 (siehe S. 214).
- [126] Microsemi. *Understanding Single Event Effects (SEEs) in FPGAs*. 2011 (siehe S. 172).
- [127] National Aeronautics and Space Administration (NASA). *Fault Management Handbook*. 2012, S. 203 (siehe S. 186).
- [128] K. O'Neill. „FPGAs for Space Applications“. In: *1. SpacE FPGA Users Workshop (SEFUW) - Präsentation*. 2012. URL: https://amstel.estec.esa.int/tecedm/website/conferences/sefuw/d2_p1_ESA_Microsemi_RT_FPGA.pdf (siehe S. 214–215).
- [129] K. O'Neill. „Radiation Tolerant FPGAs and Space System Managers“. In: *2. SpacE FPGA Users Workshop (SEFUW) - Präsentation (2014)*. URL: <https://indico.esa.int/indico/event/59/session/1/contribution/5/material/slides/0.pdf> (siehe S. 215).
- [130] *OpenPR*. URL: <http://openpr-vt.sourceforge.net/OpenPR/OpenPR.html> (siehe S. 71).
- [131] P. S. Ostler, M. P. Caffrey, D. S. Gibelyou, P. S. Graham, K. S. Morgan, B. H. Pratt, H. Quinn und M. J. Wirthlin. „SRAM FPGA reliability analysis for harsh radiation environments“. In: *IEEE Transactions on Nuclear Science* 56.6 (2009), S. 3519–3526. DOI: 10.1109/TNS.2009.2033381 (siehe S. 194, 197).
- [132] A. Otero, E. D. Torre und T. Riesgo. „Dreams : A Tool for the design of Dynamically Reconfigurable Embedded and Modular Systems“. In: *International Conference on ReConfigurable Computing and FPGA (ReConFig)*. 2012, S. 8 (siehe S. 73–74).
- [133] D. Petrick, D. Espinosa, R. Ripley, G. Crum, A. Geist und T. Flatley. „Adapting the reconfigurable spacecube processing system for multiple mission applications“. In: *IEEE Aerospace Conference*. 2014, S. 1–20. DOI: 10.1109/AERO.2014.6836227 (siehe S. 231).
- [134] M. Pormann. *Zielarchitekturen in eingebetteten Systemen*. 2014 (siehe S. 8).
- [135] M. Pormann, J. Hagemeyer, C. Pohl, J. Romoth und M. Strugholtz. „RAPTOR - A Scalable Platform for Rapid Prototyping and FPGA-based Cluster Computing“. In: *Parallel Computing: From Multicores and GPUs to Petascale*. Hrsg. von Chapman, B. et al. IOS Press, 2010, S. 592–599 (siehe S. 19, 240–241).

- [136] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, H. Demme, John and Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck und S. Heil. „A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services“. In: *Proceeding of the 41st Annual International Symposium on Computer Architecture (ISCA)*. IEEE Press, 2014, S. 13–24. ISBN: 978-1-4799-4394-4 (siehe S. 20).
- [137] H. Quinn, P. Graham, K. Morgan, Z. Baker, M. Caffrey, D. Smith und R. Bell. „On-Orbit Results for the Xilinx Virtex-4 FPGA“. In: *Radiation Effects Data Workshop (REDW)*. 2012, S. 1–8. DOI: 10.1109/REDW.2012.6353715 (siehe S. 231–232).
- [138] H. Quinn, E. Johnson, J. Johnson, B. Pratt, N. Rollins, J. Krone, D. Roussel-Dupre, M. Caffrey, P. Graham, M. Wirthlin, K. Morgan, A. Salazar, T. Nelson und W. Howes. „The Cibola Flight Experiment“. In: *ACM Transactions on Reconfigurable Technology and Systems* 8.1 (2015), S. 1–22. DOI: 10.1145/2629556 (siehe S. 230, 232).
- [139] H. Quinn, K. Morgan, J. Krone und M. Caffrey. „Eight Years of MBU Data : What Does It All Mean ?“ In: *Single Event Effect Symposium (SEE)*. 2007 (siehe S. 182–183).
- [140] *RapidSmith*. 2010. URL: <http://rapidsmith.sourceforge.net/> (siehe S. 62, 73–74, 118).
- [141] G. Reenbelt. „Advanced Hybrid On-Board Science Data Processor - SpaceCube 2.0“. In: *Earth Science Technology Forum* (2010), S. 2–5 (siehe S. 230).
- [142] M. Renovell, J. M. Portal, J. Figuras und Y. Zorian. „Minimizing the number of test configurations for different FPGA families“. In: *Test Symposium, 1999. (ATS '99) Proceedings. Eighth Asian*. 1999, S. 363–368. DOI: 10.1109/ATS.1999.810776 (siehe S. 201).
- [143] M. S. Reorda, L. Sterpone und M. Violante. „Efficient estimation of SEU effects in SRAM-based FPGAs“. In: *IEEE International On-Line Testing Symposium (IOLTS) 2005* (2005), S. 54–59. DOI: 10.1109/IOLTS.2005.26 (siehe S. 198).
- [144] M. S. Reorda, L. S. Terpone, M. V. Iolante und P. Torino. „Multiple errors produced by single upsets in FPGA configuration memory : a possible solution“. In: *European Test Symposium (ETS)*. Prin 2004. 2005 (siehe S. 198).
- [145] P. Roche, J. L. Autran, G. Gasiot und D. Munteanu. „Technology downscaling worsening radiation effects in bulk: SOI to the rescue“. In: *Technical Digest - International Electron Devices Meeting, IEDM*. 2013. DOI: 10.1109/IEDM.2013.6724728 (siehe S. 292).
- [146] K. Roed. „Single Event Upsets in SRAM FPGA based readout electronics for the Time Projection Chamber in the ALICE experiment“. Dissertation. University of Bergen, 2009 (siehe S. 225).

- [147] N. Rollins, M. Fuller und M. J. Wirthlin. „A comparison of fault-tolerant memories in SRAM-based FPGAs“. In: *IEEE Aerospace Conference*. 2010. DOI: 10.1109/AERO.2010.5446661 (siehe S. 194).
- [148] M. Rozkovec, J. Jeníček und O. Novák. „Application dependent FPGA testing method“. In: *Proceedings - 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2010*. 2010, S. 525–530. DOI: 10.1109/DSD.2010.65 (siehe S. 201).
- [149] J. S und V. K. Agrawal. „Detection and Diagnosis of Faults in the Routing Resources of a SRAM based FPGAs“. In: *International Journal of Computer Applications* 53.13 (2012), S. 18–22. DOI: 10.5120/8481-2421 (siehe S. 201).
- [150] D. Sabena, L. Sterpone, M. Schölzel, T. Koal, H. T. Vierhaus, S. Wong, R. Glein, F. Rittner, C. Stender, M. Porrmann und J. Hagemeyer. „Reconfigurable high performance architectures: How much are they ready for safety-critical applications?“ In: *19th IEEE European Test Symposium, ETS 2014* (2014). DOI: 10.1109/ETS.2014.6847820 (siehe S. 234).
- [153] F. H. Schmidt. „Fault Tolerant Design Implementation on Radiation Hardened By Design SRAM-Based FPGAs by“. Master Thesis. 2013 (siehe S. 172–173, 196, 212).
- [154] J. R. Schwank. *Space and Military Radiation Effects in Silicon-on- Insulator Devices*. 1996 (siehe S. 176).
- [155] F. Siegle, T. Vladimirova, J. Ilstad und O. Emam. „Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications“. In: *ACM Computing Surveys* 47.2 (2015), 37:1–37:34. DOI: 10.1145/2671181 (siehe S. 186–187).
- [156] J. Smith, T. Xia und C. Stroud. „An Automated BIST Architecture for Testing and Diagnosing FPGA Interconnect Faults“. In: *Journal of Electronic Testing* 22.3 (2006), S. 239–253. DOI: 10.1007/s10836-006-9319-7 (siehe S. 201).
- [157] A. A. Sohanghpurwala. „OpenPR : An Open-Source Partial Reconfiguration Tool-Kit for Xilinx FPGAs“. Master thesis. Virginia Polytechnic Institute und State University, Blacksburg, 2010 (siehe S. 71, 73).
- [158] A. A. Sohanghpurwala, P. Athanas, T. Frangieh und A. Wood. „OpenPR : An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs“. In: *IEEE International Parallel and Distributed Processing Symposium Workshops and Phd Forum (IPDPSW)*. 2011. DOI: 10.1109/IPDPS.2011.146 (siehe S. 71–72).
- [159] J. Song, Q. An und S. Liu. „A high-resolution time-to-digital converter implemented in field-programmable-gate-arrays“. In: *IEEE Transactions on Nuclear Science* 53 (2006), S. 236–241 (siehe S. 92).
- [162] SourceTech411. *Top FPGA Companies For 2013*. 2013. URL: <http://sourcetech411.com/2013/04/top-fpga-companies-for-2013/> (siehe S. 1).

- [163] C. Stender. „Fast In-Orbit FPGA Reconfiguration via In-Band TM / TC“. In: 3. *Space FPGA Users Workshop (SEFUW) - Präsentation*. 2016 (siehe S. 235).
- [164] L. Sterpone und N. Battezzati. „A Novel Design Flow for the Performance Optimization of Fault Tolerant Circuits on SRAM-based FPGAs“. In: *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)* (2008), S. 157–163. DOI: 10.1109/AHS.2008.59 (siehe S. 198).
- [165] L. Sterpone und B. Du. „Analysis and Mitigation of Single Event Effects on Flash-based FPGAs“. In: *IEEE European Test Symposium (ETS)* (2014) (siehe S. 209).
- [166] L. Sterpone, M. S. Reorda, M. Violante, P. Torino, I. Dauin und C. Duca. „RoRA : A reliability-oriented place and route algorithm for SRAM-based FPGAs“. In: *Research in Microelectronics and Electronics* (2005). DOI: 10.1109/RME.2005.1543031 (siehe S. 198).
- [167] L. Sterpone und M. Violante. „A new algorithm for the analysis of the MCUs sensitiveness of TMR architectures in SRAM-based FPGAs“. In: *IEEE Transactions on Nuclear Science* 55.4 (2008), S. 2019–2027. DOI: 10.1109/TNS.2008.2001858 (siehe S. 199).
- [168] C. Stroud, J. Nall, M. Lashinsky, N. Carolina, A. Systems und M. Hill. „BIST-Based Diagnosis of FPGA Interconnect“. In: (2002) (siehe S. 201, 301–302).
- [169] X. Sun, J. Xu, B. Chan und P. Trouborst. „Novel technique for built-in self-test of FPGA interconnects“. In: *Proceedings International Test Conference 2000 (IEEE Cat. No.00CH37159)*. 2000, S. 795–803. DOI: 10.1109/TEST.2000.894276 (siehe S. 202).
- [170] G. Swift. „Compendium of XRTC Radiation Results on All Single-Event Effects Observed in the Virtex-5QV“. In: *Military and Aerospace Programmable Logic Devices Conference (MAPLD)*. 2011 (siehe S. 211–212).
- [171] G. M. Swift. „Overview of the XRTC Single-Event Test Results on the Xilinx 7-Series FPGAs“. In: 2. *Space FPGA Users Workshop (SEFUW) - Präsentation*. 2014 (siehe S. 210).
- [172] G. Swift und G. Allen. *Virtex-5Qv Static SEU Characterization Summary*. 2012 (siehe S. 173–174, 211–212).
- [173] G. Swift, G. Allen, C. Tseng, C. Carmichael, G. Miller und J. George. „Static upset characteristics of the 90nm virtex-4QV FPGAs“. In: *IEEE Radiation Effects Data Workshop 4* (2008), S. 98–105. DOI: 10.1109/REDW.2008.25 (siehe S. 203).
- [174] TOPIC. *DYPLO - Technical benefits*. 2016. URL: <https://topicembeddedproducts.com/products/dyplo/technical-benefits/> (siehe S. 63).

- [175] *Torc - Tools for Open Reconfigurable Computing*. URL: <http://torc-isi.sourceforge.net/> (siehe S. 62, 71, 73, 118).
- [176] C. Urbina-Ortega, G. Furano, G. Magistrati, K. Marinis, A. Menicucci und D. Merodio-Codinachs. „Flash-based FPGAs in Space , design guidelines and trade-off for critical applications“. In: *Conference on Radiation Effects on Components and Systems*. 2013. ISBN: 9781467350570 (siehe S. 185, 208–209).
- [177] R. Velazco, P. Fouillat und R. Reis. *Radiation Effects on Embedded Systems*. Springer, Apr. 2007, S. 269 (siehe S. 172).
- [178] J. Wetch. *Xilinx Space Heritage*. URL: <http://slideplayer.com/slide/3602020/> (siehe S. 229).
- [179] B. S. White. „Tincr : Integrating Custom CAD Tool Frameworks with the Xilinx Vivado Design Suite“. Master thesis. Brigham Young University, Provo Utah, 2014 (siehe S. 371–374).
- [180] B. White und B. Nelson. *Tincr*. 2016. URL: <https://sourceforge.net/projects/tincr/> (besucht am 31.05.2016) (siehe S. 371).
- [181] B. White und B. Nelson. „Tincr- A custom CAD tool framework for Vivado“. In: *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. 2014. ISBN: 9781479959440 (siehe S. 371–372).
- [182] Wikipedia. *Filter mit endlicher Impulsantwort*. 2016. URL: https://de.wikipedia.org/wiki/Filter_mit_endlicher_Impulsantwort (siehe S. 287).
- [183] Wikipedia. *Hochverfügbarkeit*. 2016. URL: <https://de.wikipedia.org/wiki/Hochverf%C3%BCgbarkeit> (siehe S. 196).
- [184] Wikipedia. *ISO 26262*. URL: https://en.wikipedia.org/wiki/ISO_26262#Part_1:_Vocabulary (siehe S. 186).
- [185] Wikipedia. *Kleinsatellit*. 2016. URL: <https://de.wikipedia.org/wiki/Kleinsatellit> (siehe S. 232).
- [186] Wikipedia. *Satellitenorbit*. 2016. URL: <https://de.wikipedia.org/wiki/Satellitenorbit> (siehe S. 172).
- [187] Wikipedia. *Schnelle Fourier-Transformation*. 2016. URL: https://de.wikipedia.org/wiki/Schnelle_Fourier-Transformation (siehe S. 279).
- [188] Wikipedia. *Simulierte Abkühlung*. URL: http://de.wikipedia.org/wiki/Simulierte_Abk%C3%BChlung (siehe S. 131).
- [189] Wikipedia. *Time-To-Digital Converter*. 2010. URL: <http://de.wikipedia.org/wiki/Time-to-Digital-Converter> (siehe S. 92).
- [190] M. Wirthlin. „High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond“. In: *Proceedings of the IEEE* 103.3 (2015), S. 379–389. DOI: 10.1109/JPROC.2015.2404212 (siehe S. 177–182, 207, 223, 225–226).

- [191] J. Wu und Z. Shi. „The 10-ps wave union TDC: Improving FPGA TDC resolution beyond its cell delay“. In: *IEEE Nuclear Science Symposium* (2008), S. 3440–3446 (siehe S. 93).
- [192] Xilinx. *7 Series FPGAs Configurable Logic Block (UG474 v1.7)*. 2014 (siehe S. 38).
- [193] Xilinx. *7 Series FPGAs Configuration User Guide (UG470 v1.9)*. 2014 (siehe S. 42).
- [194] Xilinx. *7 Series FPGAs Overview (DS180 v1.16.1)*. 2013 (siehe S. 37).
- [195] Xilinx. *Annual Report Form 10-K United States Securities and Exchange Commission (2011)*. Techn. Ber. URL: <https://www.sec.gov/Archives/edgar/data/743988/000095012311055454/0000950123-11-055454-index.htm> (siehe S. 40).
- [196] Xilinx. *Annual Report Form 10-K United States Securities and Exchange Commission (2016)*. Techn. Ber. URL: <https://www.sec.gov/Archives/edgar/data/743988/000074398816000063/xlnx42201610k.htm> (siehe S. 40).
- [197] Xilinx. *Artix-7 Product Table*. 2016. URL: <http://www.xilinx.com/support/documentation/selection-guides/artix7-product-table.pdf> (siehe S. 38).
- [198] Xilinx. *Block Memory Generator (DS512 v3.3)*. 2009 (siehe S. 200).
- [199] Xilinx. *BPI Fast Configuration and iMPACT Flash Programming with 7 Series FPGAs BPI Configuration Basics (XAPP587 v1.2)*. 2015 (siehe S. 47).
- [200] Xilinx. *Command Line Tools (UG628 v14.7)*. 2013 (siehe S. 54, 81, 83, 94–95).
- [201] Xilinx. „Considerations Surrounding Single Event Effects in FPGAs, ASICs, and Processors (WP402 v1.0.1)“. In: (2012) (siehe S. 182, 223).
- [202] Xilinx. *Constraints Guide (UG625 v. 14.5)*. 2013 (siehe S. 55, 80, 84, 108).
- [203] Xilinx. *Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits (WP286 v1.1)*. 2011 (siehe S. 217–218).
- [204] Xilinx. „Device Reliability Report (UG116 v10.3.1)“. In: (2015) (siehe S. 217–221).
- [205] Xilinx. *Early Access Partial Reconfiguration User Guide (UG208 v1.2)*. 2008 (siehe S. 63–65).
- [206] Xilinx. *Kintex-7 Product Table*. 2016. URL: <http://www.xilinx.com/support/documentation/selection-guides/kintex7-product-table.pdf> (siehe S. 38).
- [207] Xilinx. *LogiCORE IP Fast Fourier Transform (DS260 v7.1)*. 2010 (siehe S. 279–282).

- [208] Xilinx. *LogiCORE IP FIR Compiler (DS534 v5.0)*. 2011 (siehe S. 287–289).
- [209] Xilinx. *LogiCORE IP Multi-Port Memory Controller (DS643 v6.05.a)*. 2011 (siehe S. 257, 264–265, 300–301).
- [210] Xilinx. *Partial Reconfiguration of Virtex FPGAs Using ISE Design Suite (WP374 v1.2)*. 2012 (siehe S. 22–24, 29, 52, 67).
- [211] Xilinx. *Partial Reconfiguration User Guide (UG702 v14.5)*. 2013 (siehe S. 51–53, 55–56, 63, 65–67, 80, 273).
- [212] Xilinx. *Radiation Effects & Mitigation Overview*. URL: <http://www.xilinx.com/esp/mil%5faero/collateral/presentations/radiation%5feffects.pdf> (siehe S. 178, 197).
- [213] Xilinx. *Single Event Upset (SEU) Detection and Correction Using Virtex-4 Devices (XAPP714, v1.5)*. 2007 (siehe S. 206, 299).
- [214] Xilinx. *Soft Error Mitigation Controller (PG036 v3.4.1)*. 2015 (siehe S. 206).
- [215] Xilinx. *Solving Today's Design Security Concerns (WP365 v1.2)*. 2012 (siehe S. 38).
- [216] Xilinx. *Space-Grade Virtex-4QV Family Overview (DS653 v2.1)*. 2014 (siehe S. 210).
- [217] Xilinx. *Spartan-6 Family Overview (DS160 v2.0)*. 2011 (siehe S. 36).
- [218] Xilinx. *Spartan-6 FPGA Configurable Logic Blocks User Guide (UG384 v1.1)*. 2010 (siehe S. 34, 36–37).
- [219] Xilinx. *Spartan-6 Product Table*. 2011. URL: http://www.xilinx.com/publications/prod_mktg/Spartan6_Product_Table.pdf (siehe S. 36, 41).
- [220] Xilinx. *System Generator for DSP User Guide (UG640 v14.3)*. 2012 (siehe S. 277).
- [221] Xilinx. *The Xilinx Design Language (HTML documentation supplied with ISE Version 6.3)*. 2000 (siehe S. 97).
- [222] Xilinx. *UltraScale Architecture and Product Overview (UG890 v2.9)*. 2016 (siehe S. 39, 41).
- [223] Xilinx. *UltraScale Architecture Configurable Logic Block (UG574 v1.4)*. 2015 (siehe S. 40).
- [224] Xilinx. *UltraScale Architecture Configuration User Guide (UG570)*. 2015 (siehe S. 42).
- [225] Xilinx. *Using SPI Flash with 7 Series FPGAs (XAPP586 v1.2)*. 2014 (siehe S. 47).
- [226] Xilinx. *Virtex Architectures Difference-Based Partial Reconfiguration (XAPP290 v2.0)*. 2007 (siehe S. 63).
- [227] Xilinx. *Virtex-4 Family Overview (DS112 v3.1)*. 2010 (siehe S. 31).

-
- [228] Xilinx. *Virtex-4 FPGA Configuration User Guide (UG071 v1.11)*. 2009 (siehe S. 42, 49, 259, 294–295, 367).
- [229] Xilinx. *Virtex-4 FPGA User Guide (UG070 v2.6)*. 2008 (siehe S. 31–32).
- [230] Xilinx. *Virtex-4 Product Table*. 2011 (siehe S. 32, 41).
- [231] Xilinx. *Virtex-5 Family Overview (DS100 v5.0)*. 2009 (siehe S. 33).
- [232] Xilinx. *Virtex-5 FPGA Configuration User Guide (UG191 v3.11)*. 2012 (siehe S. 42).
- [233] Xilinx. *Virtex-5 FPGA Data Sheet : DC and Switching Characteristics Virtex-5 FPGA Electrical Characteristics (DS202 v4.5)*. 2014 (siehe S. 201).
- [234] Xilinx. *Virtex-5 FPGA User Guide (UG190 v5.4)*. 2012 (siehe S. 33–34).
- [235] Xilinx. *Virtex-5 Product Table*. 2011. URL: http://www.xilinx.com/publications/prod_mktg/V5_LX__TXT_psm_table.pdf (siehe S. 35, 41).
- [236] Xilinx. *Virtex-5QV Family Overview (DS192 v1.4)*. 2014 (siehe S. 211–212).
- [237] Xilinx. *Virtex-6 Family Overview (DS150 v2.4)*. 2012 (siehe S. 35).
- [238] Xilinx. *Virtex-6 FPGA CLB User Guide (UG364 v1.2)*. 2012 (siehe S. 35).
- [239] Xilinx. *Virtex-6 FPGA Configuration User Guide (UG360 v3.8)*. 2014 (siehe S. 42, 206).
- [240] Xilinx. *Virtex-6 Product Table*. 2011. URL: <http://www.xilinx.com/support/documentation/selection-guides/virtex6-product-table.pdf> (siehe S. 36, 41).
- [241] Xilinx. *Virtex-7 Product Table*. 2016. URL: <http://www.xilinx.com/support/documentation/selection-guides/virtex7-product-table.pdf> (siehe S. 38, 41).
- [242] Xilinx. *Virtex-II Platform FPGAs: Complete Data Sheet (DS031 v4.0)*. 2014 (siehe S. 12–13, 15).
- [243] Xilinx. *Vivado Design Suite Tcl Command Reference Guide (UG835 v2016.1)*. 2016 (siehe S. 371).
- [244] Xilinx. *Vivado Design Suite User Guide Partial Reconfiguration (UG909 v2016.1)*. 2016 (siehe S. 51, 62).
- [245] Xilinx. *Vivado Design Suite User Guide (UG893 v2016.1)*. 2016 (siehe S. 371).
- [246] Xilinx. *Vivado Design Suite User Guide: Using Tcl Scripting (UG894 v2016.1)*. 2016 (siehe S. 371).
- [247] Xilinx. *XC6200 Field Programmable Gate Arrays (v1.10)*. 1997 (siehe S. 51).
- [248] Xilinx. „Xcell50: FPGAs on Mars“. In: *Xcell journal* 50 (2004) (siehe S. 229).

- [249] Xilinx. *Xilinx FPGAs Overcome the Side Effects of Sub-40 nm Technology (WP256 v1.2)*. 2011 (siehe S. 220–221).
- [250] Xilinx. „Xilinx unveils the vivado design suite for the next decade of 'all programmable' devices“. In: *Xcell journal* 79 (2012), S. 7 (siehe S. 371).
- [251] Xilinx. *XST User Guide for Virtex-6 , Spartan-6 , and 7 Series Devices (UG687 v14.5)*. 2013 (siehe S. 95).
- [252] Xilinx. *XST User Guide (UG627 v11.3)*. 2009 (siehe S. 95).
- [253] Xilinx. *Zynq-7000 Product Table*. 2016. URL: http://www.xilinx.com/publications/prod_mktg/low-end-portfolio-product-selection-guide.pdf (siehe S. 38, 139).
- [254] J. Yao, B. Dixon, C. Stroud und V. Nelson. „System-level built-in self-test of global routing resources in virtex-4 FPGAs“. In: *2009 IEEE International Symposium on Sustainable Systems and Technology, ISSST 2009*. 2009, S. 29–33. DOI: 10.1109/SSST.2009.4806782 (siehe S. 202).

Eigene Veröffentlichungen

- [95] S. Korf, D. Cozzi, M. Koester, J. Hagemeyer, M. Porrman, U. Rückert und M. D. Santambrogio. „Automatic HDL-Based Generation of Homogeneous Hard Macros for FPGAs“. In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2011), S. 125–132. DOI: 10.1109/FCCM.2011.36 (siehe S. 94, 96, 102, 120, 125–126, 134–135, 165–166).
- [117] J. Hagemeyer, A. Hilgenstein, D. Jungewelter, D. Cozzi, C. Felicetti, U. Rückert, S. Korf, M. Koester, F. Margaglia, M. Porrman, F. Dittmann, M. Ditze, J. Harris, L. Sterpone und J. Ilstad. „A Scalable Platform for Run-time Reconfigurable Satellite Payload Processing“. In: *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2012, S. 8. DOI: 10.1109/AHS.2012.6268642 (siehe S. 240, 245–246, 248, 250–251, 262).
- [118] J. Hagemeyer, D. Jungewelter, D. Cozzi, S. Korf und M. Porrman. „DRPM architecture overview“. In: *1. Space FPGA Users Workshop (SEFUW) - Präsentation*. 2012. URL: https://amstel.estec.esa.int/tecedm/website/conferences/sefuw/d2_p6_SEFUW_2012_drpm_hagemeyer_final.pdf.
- [144] L. Cassano, D. Cozzi, S. Korf, J. Hagemeyer, M. Porrman und L. Sterpone. „A CAD Flow for On-Line Testing and Patching Permanent Radiation Effects in Reconfigurable Systems“. In: *Military and Aerospace Programmable Logic Devices (MAPLD)*. 2013.
- [145] L. Cassano, D. Cozzi, S. Korf, J. Hagemeyer, M. Porrman und L. Sterpone. „On-Line Testing of Permanent Radiation Effects in Reconfigurable Systems“. In: *Design, Automation and Test in Europe (DATE)*. 2013, S. 717–720. DOI: 10.7873/date.2013.154.
- [148] S. Korf, G. Sievers, J. Ax, D. Cozzi, T. Jungeblut, J. Hagemeyer, M. Porrman und U. Rückert. „Dynamisch rekonfigurierbare Hardware als Basistechnologie für intelligente technische Systeme“. In: *Wissenschaftsforum Intelligente Technische Systeme (WinTeSys)*. HNI-Verlagsschriftenreihe. 2013, S. 79–90. ISBN: 978-3-942647-29-8.
- [164] L. Cassano, D. Cozzi, D. Jungewelter, S. Korf, J. Hagemeyer, M. Porrman und C. Bernardeschi. „An inter-processor communication interface for data-flow centric heterogeneous embedded multiprocessor systems“. In: *Design Technology of Integrated Systems (DTIS)*. 2014, S. 1–6. DOI: 10.1109/DTIS.2014.6850669.

- [167] D. Cozzi, D. Jungewelter, S. Korf, J. Hagemeyer und M. Porrman. „Dynamically Reconfigurable Hardware for Resource Efficiency and Fault Tolerance in Space Applications“. In: 2. *SpacE FPGA Users Workshop (SEFUW) - Präsentation*. 2014. URL: <https://indico.esa.int/indico/event/59/session/5/contribution/8>.
- [168] R. Dorociak, J. Gausemeier, S. Korf und M. Porrman. „Methods of Improving the Dependability of Self-optimizing Systems“. In: *Dependability of Self-Optimizing Mechatronic Systems*. Hrsg. von J. Gausemeier, F. J. Rammig, W. Schäfer und W. Sextro. Springer Berlin Heidelberg, 2014, S. 37–171. DOI: 10.1007/978-3-642-53742-4_3.
- [169] J. Gausemeier, M. Vaßholz, S. Korf und M. Porrman. „Development of Self-optimizing Systems“. In: *Dependability of Self-Optimizing Mechatronic Systems*. Hrsg. von J. Gausemeier, F. J. Rammig, W. Schäfer und W. Sextro. Springer Berlin Heidelberg, 2014, S. 25–36. DOI: 10.1007/978-3-642-53742-4_2.
- [172] D. Jungewelter, D. Cozzi, D. Kleibrink, S. Korf, J. Hagemeyer, M. Porrman und J. Ilstad. „AXI-based SpaceFibre IP CORE Implementation“. In: *SpaceWire Conference (SpaceWire)*. 2014, S. 196–201. DOI: 10.1109/SpaceWire.2014.6936258.
- [173] S. Korf, D. Cozzi, D. Jungewelter, J. Hagemeyer, M. Porrman und J. Ilstad. „Leveraging dynamic reconfiguration to increase fault-tolerance in FPGA-based satellite systems“. In: *Design, Automation and Test in Europe (DATE) University Booth*. 2014. URL: <https://www.date-conference.com/date14/files/file/date14/ubooth/2602.pdf> (siehe S. 292).
- [183] D. Sorrenti, D. Cozzi, S. Korf, L. Cassano, J. Hagemeyer, M. Porrman und C. Bernardeschi. „Exploiting Dynamic Partial Reconfiguration for On-Line On-Demand Testing of Permanent Faults in Reconfigurable Systems“. In: *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2014. DOI: 10.1109/DFT.2014.6962065 (siehe S. 202, 306–307).
- [218] D. Cozzi, S. Korf, J. Hagemeyer, L. Cassano, M. Porrman, C. Bernardeschi, A. Domenici und L. Sterpone. „OLT(RE)²: A Tool Flow for Mitigation of Permanent Faults in Reconfigurable Systems“. In: 3. *SpacE FPGA Users Workshop (SEFUW) - Präsentation*. 2016. URL: <https://indico.esa.int/indico/event/130/session/7/contribution/43> (siehe S. 303, 307–310, 312–313).
- [243] D. Cozzi, S. Korf, L. Cassano, J. Hagemeyer, C. Bernardeschi, A. Domenici, M. Porrman und L. Sterpone. „OLT(RE)²: an On-Line on-demand Testing approach for permanent Radiation Effects in REconfigurable systems“. In: *ESA TETC 2015 - Präsentation*. DOI: 10.1109/TETC.2016.2586195.

Betreute Arbeiten

- [151] T. Schlüssler. „Realisierung einer Konfigurationsinfrastruktur für fehlertolerante FPGA-basierte Datenverarbeitung in Satelliten“. Bachelorarbeit. Universität Paderborn, 2013 (siehe S. 252–253, 255–256, 393).
- [181] L. Santangelo. „Viv2XDL: a bridge between Vivado and XDL based software“. Master thesis. Università di Pisa, 2014. URL: <https://etd.adm.unipi.it/t/etd-09012014-224107/>.
- [182] D. Sorrenti. „Exploiting Partial Dynamic Reconfiguration for On-Line On-Demand Detection of Permanent Faults in SRAM-based FPGAs“. Master thesis. Università di Pisa, 2014. URL: <https://etd.adm.unipi.it/t/etd-03282014-113705/> (siehe S. 305).
- [205] F. Mascolo. „Design and implementation of a routing algorithm to maximize test coverage of permanent faults in FPGAs“. Master thesis. Università di Pisa, 2015. URL: <https://etd.adm.unipi.it/t/etd-09062015-235709/> (siehe S. 309).

Anhang

A DPR

A.1 Xilinx Konfigurationsdatei

Eine Xilinx Konfigurationsdatei, auch Bitstrom genannt, besteht aus einem Datei-Header und nachfolgenden Konfigurationsdaten. Die Konfigurationsdaten sind entweder Anweisungen an den Paket-Prozessor oder die darauffolgenden binären Daten-Pakete, die eigentlichen Konfigurationsinformationen. Abbildung A.1 zeigt Ausschnitte (den Anfang und das Ende) einer vollständigen Konfigurationsdatei für ein Virtex-4 FX100 FPGA. Der Datei-Header (in der Abbildung in Rottönen dargestellt) ist in sechs Felder unterteilt:

1. Pre-Header: Einheitlich definierte Bitfolge
2. Design-Name nach ASCII-Zeichen a (0x61)
3. FPGA-Typbezeichnung nach ASCII-Zeichen b (0x62)
4. Datum der Erstellung nach ASCII-Zeichen c (0x63)
5. Zeitpunkt der Erstellung nach ASCII-Zeichen d (0x64)
6. Länge der Konfigurationsdaten (in 32-Bit Wörtern) nach ASCII-Zeichen e (0x65)

Der Konfigurationsteil teilt sich durch leere Anweisungen (engl. no operation, NO-OP) und die Konfigurationsdaten in vier Kategorien und ist in Blau-Tönen dargestellt. Dabei erfolgt eine Kategorisierung in Dummy- und Synchronisationswort, Konfigurationsanweisungen und Konfigurationsdaten. Eine Aufschlüsselung zu dem Konfigurationsteil ist in dem Abschnitt *Bitstream Composition* in jedem *Configuration User Guide* gegeben.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump		
00000000	00	09	0f	f0	0f	f0	0f	f0	0f	f0	00	00	01	61	00	28	...	8.8.8.8...	a.
00000010	43	66	67	49	6e	69	74	2e	6e	63	64	3b	53	54	45	50	CfgInit.ncd;STEE		
00000020	50	49	4e	47	3d	30	3b	48	57	5f	54	49	4d	45	4f	55	PING=0;HW TIMEOU		
00000030	54	3d	46	41	4c	53	45	00	62	00	0e	34	76	66	78	31	T=FALSE;b..4vfxl		
00000040	30	30	66	66	31	31	35	32	00	63	00	0b	32	30	31	34	00ff1152;c..2014		
00000050	2f	30	35	2f	30	39	00	64	00	09	31	33	3a	32	36	3a	/05/09;d..13:26:		
00000060	35	39	00	65	00	3f	11	38	ff	ff	ff	ff	aa	99	55	66	59.e.?.8vvv?mUf		
00000070	20	00	00	00	30	00	80	01	00	00	00	07	20	00	00	00	...0.e.......		
00000080	20	00	00	00	30	01	20	01	12	04	3f	e5	30	01	80	01	...0....?@0.e.		
00000090	01	ee	40	93	30	00	80	01	00	00	00	09	20	00	00	00	.i0.e.......		
000000a0	30	00	c0	01	00	00	06	00	30	00	a0	01	00	00	06	00	0.Ä....0.....		
000000b0	20	00	00	00	20	00	00	00	20	00	00	00	20	00	00	00		
000012a0	20	00	00	00	20	00	00	00	30	00	c0	01	00	00	06	000.Ä....		
000012b0	30	00	a0	01	00	00	00	00	30	00	80	01	00	00	00	00	0....0.e.....		
000012c0	20	00	00	00	30	00	20	01	00	00	00	00	30	00	80	01	...0....0.e.		
000012d0	00	00	00	01	20	00	00	00	30	00	40	00	50	0f	bf	220.@P.z'		
003f0f60	00	00	00	00	00	00	00	00	30	00	00	01	cd	60	7f	3e0.e. f>		
003f0f70	30	00	80	01	00	00	00	0a	20	00	00	00	30	00	80	01	0.e.... ..0.e.		
003f0f80	00	00	00	03	20	00	00	00	20	00	00	00	20	00	00	00		
003f1110	20	00	00	00	30	00	80	01	00	00	00	0a	20	00	00	00	...0.e.......		
003f1120	30	00	80	01	00	00	00	00	20	00	00	00	30	00	20	01	0.e.... ..0..		
003f1130	00	01	53	00	30	00	80	01	00	00	00	05	20	00	00	00	..s.0.e.......		
003f1140	30	00	c0	01	00	00	00	00	30	00	a0	01	00	00	00	00	0.Ä....0.....		
003f1150	30	00	00	01	0c	01	1d	96	30	00	80	01	00	00	00	0d	0....0.e.....		
003f1160	20	00	00	00	20	00	00	00	20	00	00	00	20	00	00	00		
003f1170	20	00	00	00	20	00	00	00	20	00	00	00	20	00	00	00		
003f1180	20	00	00	00	20	00	00	00	20	00	00	00	20	00	00	00		
003f1190	20	00	00	00	20	00	00	00	20	00	00	00	20	00	00	00		

Pre-Header	Design-Name	FPGA-Typ	Datum	Zeit	Länge (Wörter)
Dummy + Sync	Konfig. Anweisung	Konfig. Daten			

Abbildung A.1: Übersicht einer Xilinx Konfigurationsdatei

A.2 Xilinx Konfigurationssequenzen

Tabelle A.1: Konfigurationssequenz eines Virtex-4 FPGAs [69, S. 96 ff.]

Konfigurationsdaten	Erläuterung
FFFFFFF	Dummy Wort
AA995566	Sync Wort
20000000	NO-OP
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000007	RCRC Anweisung
20000000	NO-OP
20000000	NO-OP
30012001	Typ 1 Schreib 1 Wort an Register COR
XXXXXXXX	Konfiguration
30018001	Typ 1 Schreib 1 Wort an Register ID
0167C093	Device_ID
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000009	SWITCH Anweisung
20000000	NO-OP
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000000	NULL Anweisung
20000000	NO-OP
3000C001	Typ 1 Schreib 1 Wort an Register MASK
XXXXXXXX	Konfiguration
3000A001	Typ 1 Schreib 1 Wort an Register CTL
XXXXXXXX	Konfiguration
20000000	NO-OP
...	1149 weitere NO-OPs
3000C001	Typ 1 Schreib 1 Wort an Register MASK
XXXXXXXX	Konfiguration
3000A001	Typ 1 Schreib 1 Wort an Register CTL
XXXXXXXX	Konfiguration
30002001	Typ 1 Schreib 1 Wort an Register FAR
00000000	Frame Adresse
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000001	WCFG Anweisung
30004000	Type 1 Schreibe 0 Wörter an FDRI
5003B568	Type 2 Schreibe 243048 Wörter an FDRI
XXXXXXXX	Konfiguration (Wort 0)
...	...
XXXXXXXX	Konfiguration (Wort 243047)
30000001	Typ 1 Schreib 1 Wort an Register CRC
XXXXXXXX	Konfiguration
30008001	Typ 1 Schreib 1 Wort an Register CMD
0000000A	GRESTORE Anweisung
20000000	NO-OP
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000003	LFRM Anweisung
20000000	NO-OP
...	99 weitere NO-OPs
30008001	Typ 1 Schreib 1 Wort an Register CMD
0000000A	GRESTORE Anweisung
20000000	NO-OP
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000000	NULL Anweisung
20000000	NO-OP
30002001	Typ 1 Schreib 1 Wort an Register FAR
00000000	Frame Adresse
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000005	START Anweisung
20000000	NO-OP
3000C001	Typ 1 Schreib 1 Wort an Register MASK
XXXXXXXX	Konfiguration
3000A001	Typ 1 Schreib 1 Wort an Register CTL
XXXXXXXX	Konfiguration
30000001	Typ 1 Schreib 1 Wort an Register CRC
XXXXXXXX	CRC Wort
30008001	Typ 1 Schreib 1 Wort an Register CMD
0000000D	DESYNC Anweisung
20000000	NO-OP
20000000	15 weitere NO-OPs

Tabelle A.2: Konfigurationssequenz für die dynamisch partielle Rekonfiguration eines Virtex-4 FPGAs

Konfigurationsdaten	Erläuterung
FFFFFFF	Dummy Wort
AA995566	Sync Wort
20000000	NO-OP
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000007	RCRC Anweisung
20000000	NO-OP
20000000	NO-OP
30018001	Typ 1 Schreib 1 Wort an Register ID
01EE4093	Device_ID
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000001	WCFG Anweisung
20000000	NO-OP
30002001	Typ 1 Schreib 1 Wort an Register FAR
000009C0	Frame Adresse
20000000	NO-OP
30004000	Typ 1 Schreib 0 Wörter an Register FDRI
5000662E	Typ 2 Schreib 26158 Wörter an Register FDRI
XXXXXXXX	Konfiguration (Wort 0)
...	...
XXXXXXXX	Konfiguration (Wort 26157)
30008001	Typ 1 Schreib 1 Wort an Register CMD
00000003	LFRM Anweisung
20000000	NO-OP
20000000	99 weitere NO-OPs
30000001	Typ 1 Schreib 1 Wort an Register CRC
0000DEFB	CRC word
30008001	Typ 1 Schreib 1 Wort an Register CMD
0000000D	DESYNC Anweisung
20000000	NO-OP

Tabelle A.3: Gegenüberstellung der Konfigurationssequenzen für eine vollständige und eine partielle Konfiguration eines Virtex-4 FPGAs

Konfigurationsdaten (voll)	Erläuterung	Konfigurationsdaten (partiell)
FFFFFFF	Dummy Wort	FFFFFFF
AA995566	Syne Wort	AA995566
20000000	NO-OP	20000000
30008001	Typ 1 Schreib 1 Wort an Register CMD	30008001
00000007	RCRC Anweisung	00000007
20000000	NO-OP	20000000
20000000	NO-OP	20000000
30012001	Typ 1 Schreib 1 Wort an Register COR	-
XXXXXXXX	Konfiguration	-
30018001	Typ 1 Schreib 1 Wort an Register ID	30018001
0167C093	Device_ID	01EE4093
30008001	Typ 1 Schreib 1 Wort an Register CMD	-
00000009	SWTCH Anweisung	-
20000000	NO-OP	-
30008001	Typ 1 Schreib 1 Wort an Register CMD	-
00000000	NULL Anweisung	-
20000000	NO-OP	-
3000C001	Typ 1 Schreib 1 Wort an Register MASK	-
XXXXXXXX	Konfiguration	-
3000A001	Typ 1 Schreib 1 Wort an Register CTL	-
XXXXXXXX	Konfiguration	-
20000000	NO-OP	-
...	1149 weitere NO-OPs	-
3000C001	Typ 1 Schreib 1 Wort an Register MASK	-
XXXXXXXX	Konfiguration	-
3000A001	Typ 1 Schreib 1 Wort an Register CTL	-
XXXXXXXX	Konfiguration	-
30002001	Typ 1 Schreib 1 Wort an Register FAR	30002001 ^d
00000000	Frame Adresse	000099C0
30008001	Typ 1 Schreib 1 Wort an Register CMD	30008001
00000001	WCFG Anweisung	00000001
30004000	Type 1 Schreibe 0 Wörter an FDRI	30004000
5003B568	Type 2 Schreibe 243048 Wörter an FDRI	5000662E
XXXXXXXX	Konfiguration (Wort 0)	XXXXXXXX
...
XXXXXXXX	Konfiguration (Wort 243047)	XXXXXXXX
30000001	Typ 1 Schreib 1 Wort an Register CRC	-
XXXXXXXX	Konfiguration	-
30008001	Typ 1 Schreib 1 Wort an Register CMD	-
0000000A	GRESTORE Anweisung	-
20000000	NO-OP	-
30008001	Typ 1 Schreib 1 Wort an Register CMD	30008001
00000003	LFM Anweisung	00000003
20000000	NO-OP	20000000
...	99 weitere NO-OPs	...
30008001	Typ 1 Schreib 1 Wort an Register CMD	-
0000000A	GRESTORE Anweisung	-
20000000	NO-OP	-
30008001	Typ 1 Schreib 1 Wort an Register CMD	-
00000000	NULL Anweisung	-
20000000	NO-OP	-
30002001	Typ 1 Schreib 1 Wort an Register FAR	-
00000000	Frame Adresse	-
30008001	Typ 1 Schreib 1 Wort an Register CMD	-
00000005	START Anweisung	-
20000000	NO-OP	-
3000C001	Typ 1 Schreib 1 Wort an Register MASK	-
XXXXXXXX	Konfiguration	-
3000A001	Typ 1 Schreib 1 Wort an Register CTL	-
XXXXXXXX	Konfiguration	-
30000001	Typ 1 Schreib 1 Wort an Register CRC	30000001
XXXXXXXX	CRC Wort	XXXXXXXX
30008001	Typ 1 Schreib 1 Wort an Register CMD	30008001
0000000D	DESYNC Anweisung	0000000D
20000000	NO-OP	20000000
20000000	15 weitere NO-OPs	-

^dReihenfolge der beiden Anweisungen sind beim partiellen Bitstrom vertauscht (erst wird an das CMD Register geschrieben, dann an das FAR)

A.3 Kodierung des Konfigurationsmodus

Die Kodierung wurde von der Virtex-4- zur Virtex-5-Familie geändert, um die SPI- und BPI-Konfigurationsmodi aufnehmen zu können. Dafür wurden die unterschiedlichen Slave SelectMAP Modi zu einem Modus vereint.

Tabelle A.4: Kodierung des Konfigurationsmodus

Konfigurationsmodus	M[2:0]	M[2:0]	Datenbreite in bit	Konfigurationstakt
	Virtex-4	ab Virtex-5		
Master Serial	000	000	1	CCLK als Ausgang
Slave SelectMAP32	001	-	32	CCLK als Eingang
Master SPI	-	001	1, 2 ¹ , 4 ¹	CCLK als Ausgang
Master BPI-Up	-	010	8, 16	CCLK als Ausgang
Master BPI-Down	-	011	8, 16	CCLK als Ausgang
Master SelectMAP	011	100	8, 16 ²	CCLK als Ausgang
JTAG	101	101	1	TCK als Eingang
Slave SelectMAP	-	110	8, 16, 32	CCLK als Eingang
Slave SelectMAP8	110	-	8	CCLK als Eingang
Slave Serial	111	111	1	CCLK als Eingang

Legende:

¹ ab 7-Serie

² ab Virtex-5

B DHHarMa

B.1 Xilinx Vivado

Im Jahr 2012 wurde von Xilinx die Software Vivado angekündigt, welche die bisherige Softwareumgebung ISE ablösen sollte [140]. Hauptgrund für die Entwicklung der neuen Software ist laut Xilinx das steigende Wachstum der verfügbaren FPGA-internen Blöcke von Generationssprung zu Generationssprung. Der Generationssprung von der Virtex-6-Familie zur 7-Serie-Familie führt zu einem Anstieg von 758 784 Logic Cells zu 1 954 560 Logic Cells (siehe Tabelle 3.2 in Abschnitt 3.2). Dieser Trend setzt sich auch in der UltraScale-Familie, mit bis zu 5 541 000 Logic Cells, fort.

Die Xilinx Software Vivado steht seit dem Jahr 2013 zur Verfügung und unterstützt FPGAs ab der 7-Serie. Für XDL-basierte CAD-Werkzeuge hat die Software-Umstellung eine weitreichende Konsequenz, da XDL von der Xilinx Software Vivado nicht mehr unterstützt wird und somit die Schnittstelle, in Form einer XDL-Design-Datei, zu der Xilinx Software entfernt wurde. Vivado bietet drei Möglichkeiten/Modi zur Interaktion mit der IDE [237, S. 8 ff.]: GUI, Tcl (engl. **T**ool **c**ommand **l**anguage) oder Batch.

Der GUI-Modus ist der Standardmodus und öffnet die neue Benutzeroberfläche, welche an die Benutzeroberfläche von Xilinx PlanAhead angelehnt ist. Die Tcl-Kommandos von durchgeführte Operationen in dem GUI-Modus werden in der *Tcl Console* im unteren Bereich der Vivado GUI protokolliert. Der Tcl-Modus öffnet eine kommandozeilenbasierte Schnittstelle (engl. **C**ommand **L**ine **I**nterface, CLI) mit der mit dem Vivado Tcl-Interpreter kommuniziert werden kann. Diese Schnittstelle bietet die Möglichkeit der Extraktion von Informationen über den internen Aufbau eines FPGAs oder der Manipulation an einem Design. Details hierzu werden in dem folgenden Abschnitt beschrieben. Laut [186, S. 32] ist der Tcl-Modus um einen Faktor von bis zu fünf schneller als der GUI-Modus. Der Batch-Modus ermöglicht die Ausführung eines Tcl-Skripts und beendet sich nach Abschluss der Ausführung.

Tcl-Interpreter und Tcl-Schnittstelle

Die Tcl-Schnittstelle von Vivado ist der einzige Weg, um über den Tcl-Interpreter auf die Vivado-interne Datenstruktur zugreifen zu können. Hierdurch ist es möglich, Informationen über den internen Aufbau eines neueren FPGAs zu erhalten oder Modifikationen an einem geöffneten Design durchzuführen. Eine Übersicht über die Benutzung der Tcl-Schnittstelle erfolgt in [238], eine Referenz für Vivado Tcl-Kommandos mit über 1 550 Seiten in [235]. Der in Vivado integrierte *Tcl-Store* [238, S. 69 ff.] bietet weiterhin die Möglichkeit existierende Tcl-Skripte von anderen Benutzern zu integrieren oder eigene Tcl-Skripte bereitzustellen.

Tincr

Das Framework *Tincr* [225] von White et al. bietet quelloffene Tcl-Skriptsammlungen für Vivado an. Die *TincrCAD*-Skriptsammlung erweitert die existierenden Tcl-Kommandos um zusätzliche, abstraktere Funktionen und fügt damit ein weiteres Abstraktionsniveau

ein. So werden mehrere Xilinx-basierte, Tcl-Kommandos zu einem neuen, abstrakteren Tcl-Kommando zusammengefasst. *TincrIO* ermöglicht das Importieren und Exportieren von Vivado Designs und die Extraktion von Informationen über den FPGA-internen Aufbau. Durch diese Funktionen ist es prinzipiell möglich, existierende XDL-basierte, externe CAD Werkzeuge weiterzuverwenden. In [187], der ersten Veröffentlichung zu *Tincr*, wird ein Umfang von 131 zusätzlichen Funktionen (5 000 Tcl-Zeilen) erwähnt. In [186, S. 58] wird der Umfang auf 170 Funktionen (9 000 Tcl-Zeilen) beziffert.

TincrCAD

Die *TincrCAD*-Skriptsammlung ist in verschiedene Bibliotheken unterteilt. Die wesentlichen Bibliotheken sind *Design* und *Device*. In der Bibliothek *Design* sind Funktionen zur Modifikation des Designs hinterlegt. Als Beispiel wird das Tcl-Kommando `cells insert` für das Hinzufügen einer neuen Komponente in ein bestehendes Netz aufgeführt. Das Kommando wird durch ein Tcl-Skript mit 30 Zeilen realisiert und kann zum Beispiel für das nachträgliche Hinzufügen einer Pipelinestufe verwendet werden. Die Bibliothek *Device* erweitert den Zugriff auf die internen Komponenten eines FPGAs. Ein Beispiel hierfür ist das Tcl-Kommando `pips between_nodes`, welches überprüft, ob ein PIP zwischen zwei Leitungen existiert. Die Unterbibliotheken ermöglichen, entsprechend des gewählten Bibliotheksnamens, den Entwurf eigener CAD-Werkzeuge. Dies wird durch eine Softwarekomponente zur zufallsbasierten Platzierung veranschaulicht [186, S. 71 ff.].

TincrIO

TincrIO realisiert eine dateibasierte Schnittstelle zu Vivado [186, S. 79 ff.]. Die Hauptintention der Entwicklung von *TincrIO* war, existierende XDL-basierte CAD-Werkzeuge weiterverwenden zu können, sodass eine Neuimplementierung als Tcl-basiertes CAD-Werkzeug entfällt. Weiterhin bieten externe Werkzeuge durch die Implementierung in einer hardwarenahen Programmiersprache wie C++ eine in Tcl nicht zu erreichende Ausführungsgeschwindigkeit. Damit eine externe Bearbeitung realisiert werden kann müssen folgende Funktionen bereitgestellt werden:

1. Extraktion und Speicherung von Informationen über den FPGA-internen Aufbau, analog zu der Funktion `xd1 -report` in XDL (siehe Abschnitt 4.3.1)
2. Exportieren und Importieren von Vivado Designs, analog zu den Funktionen `xd1 -ncd2xd1` und `xd1 -xd12ncd` in XDL (siehe Abschnitt 4.3.2)

Extraktion des FPGA-internen Aufbaus

Um eine externe Verarbeitung durch ein CAD-Werkzeug durchführen zu können, werden Informationen über die Verschaltungs- und Konfigurationsmöglichkeiten von den Basisblöcken eines FPGAs benötigt. In [186, S. 90 ff.] wird die Vorgehensweise der Extraktion dieser Informationen aus Vivado heraus vorgestellt. Als Zielformat wird, nach einer Gegenüberstellung der Vor- und Nachteile, das XDLRC-Format verwendet. Hierdurch können existierende CAD-Werkzeuge ohne weitere Modifikationen eine eigene Datenstruktur oder Datenbank erstellen.

Das Tcl-Kommando `write_xdlrc` aus der TincrIO-Unterbibliothek *Device* erzeugt einen XDL-Report der dritten Detailstufe. Durch die Unterstützung der 7-Serie-FPGAs von ISE und Vivado konnte eine Verifikation der erstellten Reports durchgeführt werden, welche als Ergebnis die funktionale Gleichheit hervorbrachte [186, S. 95 ff.]. Ebenfalls wurde eine Messung bzgl. der Laufzeit zur Extraktion der Informationen durchgeführt. Exemplarisch wird die Laufzeit für die größten FPGAs der 7-Serie-Unterfamilien unter Verwendung von acht Prozessen aufgeführt (siehe Tabelle A.5. Erwähnenswert ist hierbei die Laufzeit für den größten 7-Serie-FPGA, den Virtex-7 2000T, mit ca. 26 Stunden. Die Erstellung von XDL-Reports für die UltraScale-Familie wird erwähnt, allerdings keine konkreten Ergebnisse präsentiert. Da der Extraktionsprozess (annähernd) linear mit der Anzahl an Komponenten anwächst, ist (bei gleichbleibender Hardware- und Prozesskonfiguration) mit maximalen Laufzeiten von drei bis vier Tagen zu rechnen. Hervorzuheben ist, dass die Extraktion nur einmalig durchgeführt werden muss.

Tabelle A.5: Laufzeit der Extraktion durch `write_xdlrc` für die größten FPGAs der 7-Serie-Unterfamilien [186, S. 96]

FPGA	#Sites	#Nodes	Laufzeit in hh:mm:ss
xc7a200tffg1156	60 693	7 857 396	02:44:07
xc7k480tffg1156	131 651	17 495 020	06:26:50
xc7v2000tffg1925	489 060	57 693 368	25:52:28
xc7z100ffg1156	122 450	16 174 017	05:52:13

Exportieren und Importieren von Vivado Designs

Vivado verwendet sogenannte *checkpoints* um Designs in externe Dateien abzuspeichern. Während die Netzliste in einem solchen checkpoint in dem ASCII-basierten EDIF (Electronic Design Interchange Format) abgespeichert ist, sind die Platzierungs- und Verdrahtungsinformationen in einer verschlüsselten Datei im Format XDEF (Xilinx Design Exchange Files) abgelegt. Als Schnittstelle scheidet ein Vivado checkpoint daher aus. Vivado bietet jedoch Tcl-Kommandos an, wodurch die Designinformationen (mit Einschränkungen) in unverschlüsselten, ASCII-basierten Dateien exportiert und anschließend importiert werden können. Hierdurch ist es möglich einen eigenen checkpoint eines Designs anzulegen. Die erwähnten Einschränkungen beziehen sich auf besondere Verdrahtungsstrukturen, den sogenannten *route-through* (Verbindungen durch Komponenten hindurch), welche nicht wiederhergestellt werden können [186, S. 82 ff.].

Die Netzliste kann über das Tcl-Kommando `write_edif` exportiert und über `read_edif` importiert werden. Die Platzierungs- und Verdrahtungsbedingungen können in Form von XDC (Xilinx Design Constraint)-Dateien über das Tcl-Kommando `write_xdc` exportiert und über `read_xdc` importiert werden.

Entscheidend hierbei ist, dass die XDC-Dateien die Platzierungs- und Verdrahtungsbedingungen enthalten und nicht die konkreten Platzierungs- und Verdrahtungsformationen. Der Unterschied wird im Vergleich der Exportier- und Importierzeiten von Vivado checkpoints (DCP) und Tincr checkpoints (TCP) für verschiedene Designs, variierend von einigen Hundert LUTs bis zu 250 000 LUTs, deutlich [186, S. 87 ff.]. Wie in der Abbildung A.2 (a) dargestellt, steigen die Exportierzeiten linear mit der Anzahl an Elementen in dem Design. Der Unterschied zwischen den Exportierzeiten eines DCPs und TCPs ist deutlich zu erkennen, liegt jedoch im Zeitrahmen von Minuten (für das größte Design bei ca. 17 Minuten). Bei der Importierzeit, dargestellt in Abbildung A.2 (b), ist jedoch ein exponentieller Anstieg für das Importieren eines anwachsenden TCPs erkennbar und resultiert in einer Importierzeit im Zeitrahmen von Stunden bis hin zu Tagen. Die Importierzeit eines Xilinx DCPs steigt hingegen linear. Der Grund hierfür ist laut White, dass die Wiederherstellung der Platzierung und Verdrahtung durch die gespeicherten Bedingungen in Form von Tcl-Kommandos mit einigen internen Überprüfungen einhergehen. Hierbei wurde festgestellt, dass das Hinzufügen der letzten Elemente den Großteil der benötigten Zeit veranschlagt.

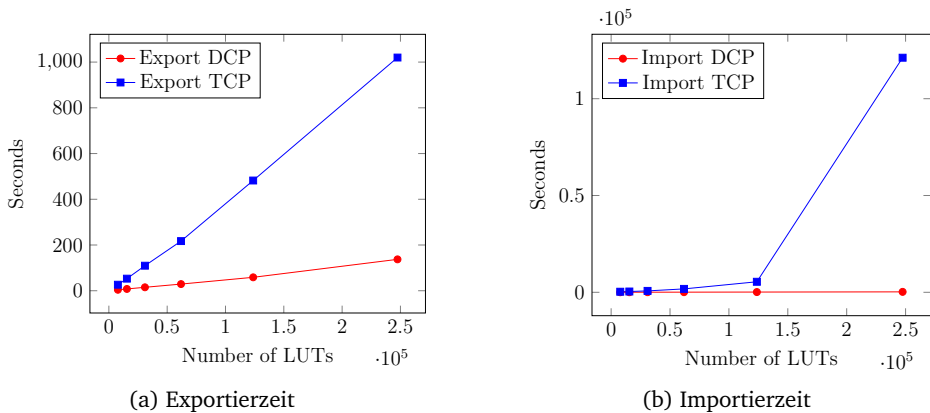


Abbildung A.2: Exportier- und Importierzeiten für Vivado (DCP) und Tincr (TCP) checkpoints [186, S. 88 ff.]

Anpassungen eines XDL-basierten Werkzeugs

Die einzelnen Schritte zur Anpassung eines XDL-basierten Werkzeugs an TincrIO sind in [186, S. 82 f.] beschrieben. Sie umfassen die Integration eines EDIF- und XDC-Parsers für das Importieren eines Tincr checkpoints sowie der Erstellung der genannten Formate für das Exportieren nach erfolgreicher Manipulation an dem Design.

B.2 DXF

Die folgenden Auswertungen wurden auf einem i7-920 Prozessor (2,66 GHz) mit 6 GB Arbeitsspeicher (1066 MHz) in einem 64-Bit Linux-Betriebssystem (Ubuntu 10.10) mit dem Compiler gcc in Version 4.4.1 durchgeführt.

Tabelle A.6: Speicherauswertung der zweiten DXF mit Darstellung des größten FPGAs einer jeden Unterfamilie aller unterstützten FPGA-Familien [94, S. 57]

Familie	FPGA	Größe Report auf HDD in GB	geladene DXF		kompr. DXF		geladene DXF	
			Phase II im RAM in MB	in %	Phase II auf HDD in MB	in %	Phase III im RAM in GB	in %
Virtex-4	fx140ff1517	8,3	220	2,6	5,5	0,06	1,3	15,6
	sx55ff1148	3,6	75	2,1	1,6	0,04	0,5	13,8
	lx200ff1513	10,4	120	1,1	2,1	0,02	2,1	20,1
Virtex-5	fx100tff1738	5,4	130	2,4	2,8	0,05	1,1	20,3
	sx95tff1136	5,6	95	1,6	1,9	0,03	0,7	12,5
	tx150tff1759	6,8	110	1,6	2,1	0,03	1,4	20,5
	lx330ff1760	12,4	105	0,8	2,0	0,02	2,7	21,7
Virtex-6	cx240tff1156	9,0	130	1,4	2,5	0,03	2,2	24,4
	sx475tff1156	18,8	195	1,0	3,4	0,02	4,5	23,9
	lx760ff1760	24,2	210	0,8	3,5	0,01	7,0	28,9
Spartan-6	lx150tfgg484	4,1	215	5,2	5,6	0,13	0,9	21,9

Tabelle A.7: Zeitauswertung der zweiten DXF mit Darstellung des größten FPGAs einer jeden Unterfamilie aller unterstützten FPGA-Familien [94, S. 57]

Familie	FPGA	Phase I	Phase II	Phase III	Total
		Zeit in hh:mm:ss	Zeit in hh:mm:ss	Zeit in hh:mm:ss	Zeit in hh:mm:ss
Virtex-4	fx140ff1517	00:13:57	00:58:07	00:00:15	01:12:19
	sx55ff1148	00:06:14	00:22:10	00:00:11	00:28:35
	lx200ff1513	00:15:45	00:59:27	00:00:20	01:15:22
Virtex-5	fx100tff1738	00:08:11	00:32:40	00:00:08	00:40:59
	sx95tff1136	00:09:17	00:31:15	00:00:08	00:40:30
	tx150tff1759	00:10:23	00:49:02	00:00:10	00:59:35
	lx330ff1760	00:28:01	01:35:05	00:00:20	01:53:26
Virtex-6	cx240tff1156	00:14:18	01:07:48	00:00:15	01:22:21
	sx475tff1156	00:45:43	02:35:08	00:00:29	03:21:20
	lx760ff1760	01:02:23	03:31:22	00:00:39	04:34:24
Spartan-6	lx150tfgg484	00:13:31	00:27:10	00:00:07	00:40:48

B.3 Simulated Annealing

```

simulatedAnnealing(double temperatur, int dLimit = 8, int beta = 10, bool badCostMoves = true
, REGIONS activatedRegions = BASE_RECONF, INITIAL_PLACEMENT initPlacement = RANDOM_PLACEMENT )
{
    //1. Count # Slices and devide them in static and reconf
    int nBlockStatic = getNumberOfStaticSlices();
    int nBlockReconf = getNumberOfReconfSlices();

    //2. Calculate movements per temperaturerature with formula 4.3
    int nSwaps = (beta*(nBlockStatic +nBlockReconf))^(1.33);
    //2.1 devide the movements in the type of regions
    int swapsPerRegion = nSwaps/(nBlockStatic + nBlockReconf);
    int swapsInStaticRegions = swapsPerRegion * nBlockStatic;
    int swapsInReconfRegions = swapsPerRegion * nBlockReconf;
    //2.2 take into accout, that each movement is done not only in one but in all regions:
    int swapsInStaticRegions /= nBaseRegions;
    int swapsInReconfRegions /= nReconfRegions;

    //3. initial settings
    //3.1 parameter
    double alpha = 1;
    int rAccept = 1;
    int nNets = getNumberOfNets();
    //3.2 Select initial Placement
    initialPlacement(activatedRegions, initPlacement);
    double cost = calcCostFunction();

    //4. Simulated Annealing
    while (checkExittemperatur(temperatur,cost,nNets) == false) // formula 4.6
    {
        int acceptedMoves = 0;
        if ((activatedRegions == BASE) || (activatedRegions == BASE_RECONF))
        {
            for (i = 0; i < swapsInStaticRegions; i++)
            {
                randomMove(groupedBaseRegions,acceptedMoves,dLimit,cost,temperatur,
                badCostMoves);
            }
        }
        if ((activatedRegions == RECONF) || (activatedRegions == BASE_RECONF))
        {
            for ( i = 0; i < swapsInReconfRegions; i++)
            {
                randomMove(groupedReconfRegions,acceptedMoves,dLimit,cost,temperatur,
                badCostMoves);
            }
        }
        int rAccept = acceptedMoves / nSwaps;
        temperatur = alpha * temperatur; // formula 4.5
        alpha = calcAlpha(rAccept);
        dLimit = dLimit * (0.56 + rAccept); // formula 4.4
        if (dLimit < 1)
            dLimit = 1;
        cost = calcCostFunction(); // formula 4.1
    }
}

```

Abbildung A.3: Programmcode der simulierten Abkühlung [94, S. 93]

B.4 HDL-Bibliothek für Kommunikationsinfrastrukturen

Umsetzung EWB WB

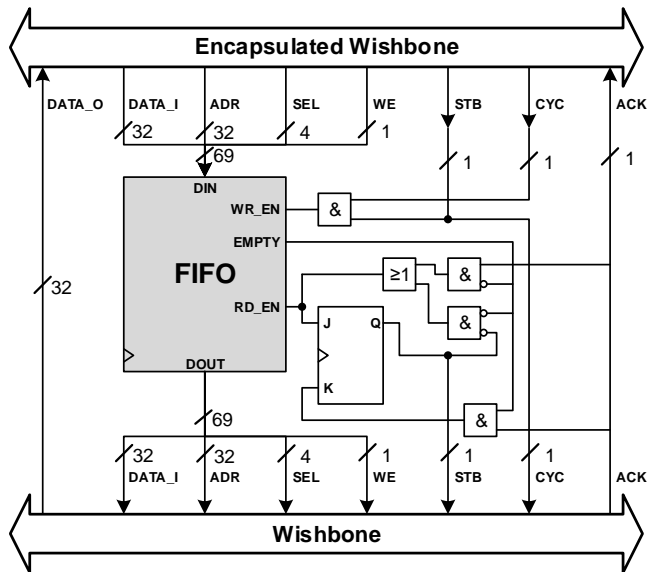


Abbildung A.4: Umsetzung Encapsulated Wishbone zu Wishbone
(basierend auf [166, S. 204])

FPGA-Ressourcen der Anschlussstellen

Tabelle A.8: Zusammensetzung der FPGA-Ressourcen für die eindimensionale SEWB-Anschlussstelle in der statischen Region (*tile_base*)

Bit	Signal Name	VHDL-Komponente	Breite in bit	FPGA-Ressourcen	
				LUTs	FFs
31-0	EWB_DATA	primitiv_base	32	128	96
32	EWB_ACK	primitiv_base	1	4	3
36-33	LED	primitiv_base	4	16	12
40-37	RST_MOD	primitiv_base	4	16	12
44-41	EWB_STB	primitiv_base	4	16	12
48-45	TILE_EN	primitiv_base	4	16	12
80-49	EWB_ADR	primitiv_base	32	128	96
88-81	EWB_BLK_CNT	primitiv_base	8	32	24
89	EWB_CYC	primitiv_base	1	4	3
93-90	EWB_SEL	primitiv_base	4	16	12
94	EWB_WE	primitiv_base	1	4	3
Summe			95	380	285
\emptyset_{bit}				4,00	3,00

Tabelle A.9: Zusammensetzung der FPGA-Ressourcen für die eindimensionale SEWB-Anschlussstelle in der PR-Region (*tile_pr*)

Bit	Signal Name	VHDL-Komponente	Breite in bit	FPGA-Ressourcen	
				LUTs	FFs
31-0	EWB_DATA	primitiv_shared	32	128	96
32	EWB_ACK	primitiv_shared	1	4	3
36-33	LED	primitiv_shared	4	16	12
40-37	RST_MOD	primitiv_dedi_bin	4	5	5
44-41	EWB_STB	primitiv_dedi_bin	4	5	5
48-45	TILE_EN	primitiv_dedi_bin_tog	4	6	5
80-49	EWB_ADR	primitiv_shared_local_in	32	32	32
88-81	EWB_BLK_CNT	primitiv_shared_local_in	8	8	8
89	EWB_CYC	primitiv_shared_local_in	1	1	1
93-90	EWB_SEL	primitiv_shared_local_in	4	4	4
94	EWB_WE	primitiv_shared_local_in	1	1	1
Summe			95	210	172
\emptyset_{bit}				2,21	1,81

Tabelle A.10: Zusammensetzung der FPGA-Ressourcen für die eindimensionale MEWB-Anschlussstelle in der statischen Region (*tile_base*)

Bit	Signal Name	VHDL-Komponente	Breite in bit	FPGA-Ressourcen LUTs	FFs
31-0	EWB_DATA	primitiv_shared	32	128	96
63-32	EWB_ADR	primitiv_shared	32	128	96
67-64	EWB_SEL	primitiv_shared	4	16	12
75-68	EWB_BLK_CNT	primitiv_shared	8	32	24
79-76	LED	primitiv_shared	4	16	12
80	EWB_CYC	primitiv_shared	1	4	3
81	EWB_WE	primitiv_shared	1	4	3
82	EWB_ACK	primitiv_shared	1	4	3
86-83	EWB_GNT	primitiv_shared	4	16	12
90-87	EWB_STB	primitiv_shared	4	16	12
94-91	RST_MOD	primitiv_shared	4	16	12
98-95	TILE_EN	primitiv_shared	4	16	12
114-99	EWB_REQ	primitiv_shared	16	64	48
Summe			115	460	345
\emptyset_{bit}				4,00	3,00

Tabelle A.11: Zusammensetzung der FPGA-Ressourcen für die eindimensionale MEWB-Anschlussstelle in der PR-Region (*tile_pr*)

Bit	Signal Name	VHDL-Komponente	Breite in bit	FPGA-Ressourcen LUTs	FFs
31-0	EWB_DATA	primitiv_shared	32	128	96
63-32	EWB_ADR	primitiv_shared	32	128	96
67-64	EWB_SEL	primitiv_shared	4	16	12
75-68	EWB_BLK_CNT	primitiv_shared	8	32	24
79-76	LED	primitiv_shared	4	16	12
80	EWB_CYC	primitiv_shared	1	4	3
81	EWB_WE	primitiv_shared	1	4	3
82	EWB_ACK	primitiv_shared	1	4	3
86-83	EWB_GNT	primitiv_dedi_bin	4	5	5
90-87	EWB_STB	primitiv_dedi_bin	4	5	5
94-91	RST_MOD	primitiv_dedi_bin	4	5	5
98-95	TILE_EN	primitiv_dedi_bin_tog	4	6	5
114-99	EWB_REQ	primitiv_dedi_master	16	48	32
Summe			115	401	301
\emptyset_{bit}				3,49	2,62

Tabelle A.12: Zusammensetzung der FPGA-Ressourcen für die zweidimensionale SEWB-Anschlussstelle in der statischen Region (*tile_base_2d*)

Bit	Signal Name	VHDL-Komponente	Breite in bit	FPGA-Ressourcen	
				LUTs	FFs
31-0	EWB_DATA	primitiv_base_2d	32	128	96
32	EWB_ACK	primitiv_base_2d	1	4	3
36-33	LED	primitiv_base_2d	4	16	12
40-37	RST_MOD	primitiv_base_2d	4	16	12
44-41	EWB_STB	primitiv_base_2d	4	16	12
48-45	TILE_EN	primitiv_base_2d	4	16	12
80-49	EWB_ADR	primitiv_base_2d	32	128	96
88-81	EWB_BLK_CNT	primitiv_base_2d	8	32	24
89	EWB_CYC	primitiv_base_2d	1	4	3
93-90	EWB_SEL	primitiv_base_2d	4	16	12
94	EWB_WE	primitiv_base_2d	1	4	3
Summe			95	380	285
\emptyset_{bit}				4,00	3,00

Tabelle A.13: Zusammensetzung der FPGA-Ressourcen für die zweidimensionale SEWB-Anschlussstelle in der PR-Region (*tile_pr_2d*)

Bit	Signal Name	VHDL-Komponente	Breite in bit	FPGA-Ressourcen	
				LUTs	FFs
31-0	EWB_DATA	primitiv_shared_2d	32	128	96
32	EWB_ACK	primitiv_shared_2d	1	4	3
36-33	LED	primitiv_shared_2d	4	16	12
40-37	RST_MOD	primitiv_dedi_bin_2d	4	7	5
44-41	EWB_STB	primitiv_dedi_bin_2d	4	7	5
48-45	TILE_EN	primitiv_dedi_bin_tog_2d	4	8	5
80-49	EWB_ADR	primitiv_shared_local_in_2d	32	96	64
88-81	EWB_BLK_CNT	primitiv_shared_local_in_2d	8	24	16
89	EWB_CYC	primitiv_shared_local_in_2d	1	3	2
93-90	EWB_SEL	primitiv_shared_local_in_2d	4	12	8
94	EWB_WE	primitiv_shared_local_in_2d	1	3	2
Summe			95	308	218
\emptyset_{bit}				3,24	2,29

FPGA-Ressourcen der DPR-Kommunikationsinfrastrukturen

Tabelle A.14: FPGA-Ressourcen der 1D DB-Kommunikationsinfrastrukturen

Kommunikationsinfrastruktur Part.	Busbreite in bit	Name	Region VHDL-Komp.	Anzahl	FPGA-Ressourcen LUTs		FFs
					Orig.	Opt.	
L	8	Base0	tile_base	1	32	24	24
		Rec0_a - Rec2_d	tile_pr	4	128	128	96
		Base1	tile_term	1	0	0	16
		Summe		6	160	152	136
		\emptyset_{Reg}			26,67	25,33	22,67
		\emptyset_{Reg_bit}			3,33	3,17	2,83
	32	Base0	tile_base	1	128	96	96
		Rec0_a - Rec2_d	tile_pr	4	512	512	384
		Base1	tile_term	1	0	0	64
		Summe		6	640	608	544
		\emptyset_{Reg}			106,67	101,33	90,67
		\emptyset_{Reg_bit}			3,33	3,17	2,83
	64	Base0	tile_base	1	256	192	192
		Rec0_a - Rec2_d	tile_pr	4	1024	1024	768
		Base1	tile_term	1	0	0	128
		Summe		6	1280	1216	1088
		\emptyset_{Reg}			213,33	202,67	181,33
		\emptyset_{Reg_bit}			3,33	3,17	2,83
LR	8	Base0	tile_base	1	32		24
		Rec0_a - Rec3_h	tile_pr	8	256		192
		Base1	tile_term	1	0		16
		Base2	tile_term	1	0		16
		Summe		11	288		248
		\emptyset_{Reg}			26,18		22,55
	32	Base0	tile_base	1	128		96
		Rec0_a - Rec3_h	tile_pr	8	1024		768
		Base1	tile_term	1	0		64
		Base2	tile_term	1	0		64
		Summe		11	1152		992
		\emptyset_{Reg}			104,73		90,18
	64	Base0	tile_base	1	256		192
		Rec0_a - Rec3_h	tile_pr	8	2048		1536
		Base1	tile_term	1	0		128
		Base2	tile_term	1	0		128
		Summe		11	2304		1984
		\emptyset_{Reg}			209,45		180,36
	\emptyset_{Reg_bit}		3,27		2,82		

Tabelle A.15: FPGA-Ressourcen der 1D SEWB-Kommunikationsinfrastrukturen

Part.	Name	Region VHDL-Komp.	Anzahl	FPGA-Ressourcen			
				LUTs		FFs	
				Orig.	Opt.	Orig.	Opt.
L	Base0	tile_base	1	380	148	285	164
	Rec0_a - Rec2_d	tile_pr	4	840	820	688	648
	Base1	tile_term	1	0	0	190	190
	Summe		6	1 220	968	1 163	1 002
	\emptyset_{Reg}			203,33	161,33	193,83	167,00
	\emptyset_{Reg_bit}			2,14	1,70	2,04	1,76
LR	Base0	tile_base	1	380	148	285	164
	Rec0_a - Rec3_h	tile_pr	8	1 680	1 640	1 296	1 296
	Base1	tile_term	1	0	0	190	190
	Base2	tile_term	1	0	0	190	190
	Summe		11	2 060	1 788	1 961	1 840
	\emptyset_{Reg}			187,27	162,55	178,27	167,27
	\emptyset_{Reg_bit}		1,97	1,71	1,88	1,76	
M	Base0	tile_base	1	380	148	285	164
	Rec0_a - Rec0_b	tile_pr	2	420	410	324	324
	Base1	tile_term	1	0	0	190	190
	Base2	tile_term	1	0	0	190	190
	Summe		5	800	558	989	868
	\emptyset_{Reg}			160,00	111,60	197,80	173,60
	\emptyset_{Reg_bit}		1,68	1,17	2,08	1,83	

Tabelle A.16: FPGA-Ressourcen der 1D MEWB-Kommunikationsinfrastrukturen

Part.	Name	Region VHDL-Komp.	Anzahl	FPGA-Ressourcen		FFs
				Orig.	Opt.	
L	Base0	tile_base	1	460	345	345
	Rec0_a - Rec2_d	tile_pr	4	1604	1604	1204
	Base1	tile_term	1	0	0	230
	Summe		6	2064	1949	1779
	\emptyset_{Reg}			344,00	324,83	296,50
	\emptyset_{Reg_bit}			2,99	2,82	2,58
	LR	Base0	tile_base	1	460	345
Rec0_a - Rec3_h		tile_pr	8	3272	3272	2408
Base1		tile_term	1	0	0	230
Base2		tile_term	1	0	0	230
Summe			10	3732	3617	3213
\emptyset_{Reg}				373,20	361,70	321,30
\emptyset_{Reg_bit}				3,25	3,15	2,79
M	Base0	tile_base	1	460	345	345
	Rec0_a - Rec0_b	tile_pr	2	818	818	602
	Base1	tile_term	1	0	0	230
	Base2	tile_term	1	0	0	230
	Summe		4	1278	1163	1407
	\emptyset_{Reg}			319,50	290,75	351,75
	\emptyset_{Reg_bit}			2,78	2,53	3,06

Tabelle A.17: FPGA-Ressourcen der 2D DB-Kommunikationsinfrastrukturen

Kommunikationsinfrastruktur Part.	Busbreite in bit	Name	Region	VHDL-Komp.	Anzahl	FPGA-Ressourcen		FFs	
						Orig.	Opt.		
2x2	8	Base0		tile_base_2d	1	32	24	24	
		Rec0_a - Rec0_d		tile_pr_2d	4	128	128	96	
		Base1		tile_term	1	0	0	16	
		Summe			6	160	152	136	
		Ø (FPGA-Ressourcen/Region)				26,67	25,33	22,67	
		Ø (FPGA-Ressourcen/bit)				3,33	3,17	2,83	
		32	Base0		tile_base_2d	1	128	96	96
	Rec0_a - Rec0_d			tile_pr_2d	4	512	512	384	
	Base1			tile_term	1	0	0	64	
	Summe				6	640	608	544	
	Ø (FPGA-Ressourcen/Region)					106,67	101,33	90,67	
	Ø (FPGA-Ressourcen/bit)					3,33	3,17	2,83	
	64	Base0		tile_base_2d	1	256	192	192	
Rec0_a - Rec0_d			tile_pr_2d	4	1024	1024	768		
Base1			tile_term	1	0	0	128		
Summe				6	1280	1216	1088		
Ø (FPGA-Ressourcen/Region)					213,33	202,67	181,33		
Ø (FPGA-Ressourcen/bit)					3,33	3,17	2,83		
	8	Base0		tile_base_2d	1	32	24	24	
Rec0_a - Rec0_f			tile_pr_2d	6	192	192	144		
Summe				7	224	216	168		
Ø (FPGA-Ressourcen/Region)					32,00	30,86	24,00		
Ø (FPGA-Ressourcen/bit)					4,00	3,86	3,00		
		32	Base0		tile_base_2d	1	128	96	96
Rec0_a - Rec0_f			tile_pr_2d	6	768	768	576		
Summe				7	896	864	672		
Ø (FPGA-Ressourcen/Region)					128,00	123,43	96,00		
Ø (FPGA-Ressourcen/bit)					4,00	3,86	3,00		
	64		Base0		tile_base_2d	1	256	192	192
Rec0_a - Rec0_f			tile_pr_2d	6	1536	1536	1152		
Summe				7	1792	1728	1344		
Ø (FPGA-Ressourcen/Region)					256,00	246,86	192,00		
Ø (FPGA-Ressourcen/bit)					4,00	3,86	3,00		

Tabelle A.18: FPGA-Ressourcen der 2D SEWB-Kommunikationsinfrastrukturen

Part.	Name	Region VHDL-Komp.	Anzahl	FPGA-Ressourcen			
				LUTs		FFs	
				Orig.	Opt.	Orig.	Opt.
2x2	Base0	tile_base_2d	1	380	148	285	164
	Rec0_a - Rec0_d	tile_pr_2d	4	1 232	1 212	872	832
	Base1	tile_term	1	0	0	190	190
	Summe		6	1 612	1 360	1 347	1 186
	\emptyset_{Reg}			268,67	226,67	224,50	197,67
	\emptyset_{Reg_bit}			2,83	2,39	2,36	2,08
3x3	Base0	tile_base_2d	1	380	148	285	164
	Rec0_a - Rec0_i	tile_pr_2d	9	2 772	2 727	1 962	1 872
	Summe		10	3 152	2 875	2 247	2 036
	\emptyset_{Reg}			315,20	287,50	224,70	203,60
	\emptyset_{Reg_bit}			3,32	3,03	2,37	2,14
	5x2	Base0	tile_base_2d	1	380	148	285
Rec0_a - Rec0_j		tile_pr_2d	10	3 080	3 030	2 180	2 080
Summe			11	3 460	3 178	2 465	2 244
\emptyset_{Reg}				314,55	288,91	224,09	204,00
\emptyset_{Reg_bit}				3,31	3,04	2,36	2,15

Tabelle A.19: FPGA-Ressourcen der 2D MEWB-Kommunikationsinfrastrukturen

Part.	Name	Region VHDL-Komp.	Anzahl	FPGA-Ressourcen		
				LUTs Orig.	Opt.	FFs
2x2	Base0	tile_base_2d	1	460	345	345
	Rec0_a - Rec0_d	tile_pr_2d	4	1 636	1 636	1 204
	Base1	tile_term	1	0	0	230
	Summe		6	2 096	1 981	1 779
	\emptyset_{Reg}			349,33	330,17	296,50
	\emptyset_{Reg_bit}			3,04	2,87	2,58
3x3	Base0	tile_base_2d	1	460	345	345
	Rec0_a - Rec0_i	tile_pr_2d	9	3 681	3 681	2 709
	Summe		10	4 141	4 026	3 054
	\emptyset_{Reg}			414,10	402,60	305,40
	\emptyset_{Reg_bit}			3,60	3,50	2,66
	5x2	Base0	tile_base_2d	1	460	345
Rec0_a - Rec0_j		tile_pr_2d	10	4 090	4 090	3 010
Base1		tile_term	1	0	0	230
Summe			12	4 550	4 435	3 585
\emptyset_{Reg}				379,17	369,58	298,75
\emptyset_{Reg_bit}				3,30	3,21	2,60

B.5 Auswertung

Platzierungsstrategien

Tabelle A.20: Auswertung der Platzierungsstrategien für verschiedene Partitionierungen in der Virtex-4-Architektur

Part.	Busbreite in bit	Cluster	Cluster BinPack	Simulated Annealing (Init: Zufall)		Simulated Annealing (Init: Cluster)			
				min	Ø	min	Ø	max	
2x2	8	17 978	15 296	15 171	19 880	26 994	13 590	13 774	14 004
	32	262 184	251 318	280 140	311 004	387 849	242 121	244 273	246 697
	64	1 075 232	-	1 305 131	1 503 611	1 618 816	1 072 404	1 073 045	1 074 050
3x2	8	31 744	32 776	33 512	41 747	49 977	31 744	31 744	31 744
	32	500 224	537 430	542 363	633 733	790 871	500 224	500 224	500 224
	64	2 072 370	-	2 412 732	2 817 016	3 169 827	2 072 370	2 072 370	2 072 370

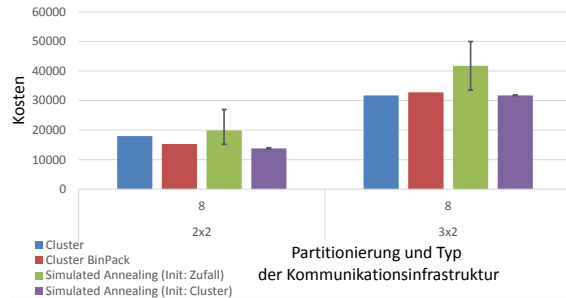


Abbildung A.5: Auswertung der Platzierungsstrategien für verschiedene Partitionierungen mit 8 bit Kommunikationsinfrastrukturen in der Virtex-4-Architektur

Tabelle A.21: Auswertung der Platzierungsstrategien für verschiedene Partitionierungen in der Artix-7-Architektur

Part.	Busbreite in bit	Cluster	Cluster BinPack	Simulated Annealing (Init: Zufall)			Simulated Annealing (Init: Cluster)		
				min	Ø	max	min	Ø	max
2x2	8	55 560	46 572	44 289	47 176	50 837	36 713	37 157	37 556
	32	784 112	713 433	625 684	663 302	690 683	626 745	638 823	660 468
	64	3 128 327	3 148 725	2 659 377	2 724 864	2 793 840	2 912 967	2 931 609	2 956 576
3x2	8	101 720	102 035	104 853	112 056	120 733	99 371	99 605	99 908
	32	1 509 714	1 533 268	1 482 719	1 545 570	1 673 834	1 455 853	1 464 842	1 474 960
	64	6 148 161	6 526 634	5 665 268	6 057 857	6 638 288	5 915 094	5 967 860	6 011 709

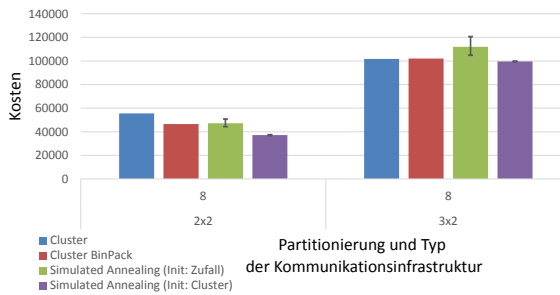


Abbildung A.6: Auswertung der Platzierungsstrategien für verschiedene Partitionierungen mit 8 bit Platzierungsstrategien in der Artix-7-Architektur

DPR-Kommunikationsinfrastrukturen

Tabelle A.22: Auswertung der Packungsdichte für homogene und inhomogene 1D DPR-Kommunikationsinfrastrukturen in der Virtex-6-Architektur (LX75T)

Typ	Design					Inhom. Komm.-Makro (ISE)						Hom. Komm.-Makro (DHHarMa)				Vergleich Slices			
	Konfiguration Part.	Busbr.	PRR in #	FFs in #	LUTs in #	Standard MAP-Einstellungen			Optimierte MAP-Einstellungen			Standard Einstellungen				ISE in %	DHHarMa vs. MAP-Opt. in %		
						Slices in #	Ausl. in %	Packungsdichte in FFs/Sl.	Slices in #	Ausl. in %	Packungsdichte in LUTs/Sl.	Slices in #	Ausl. in %	Packungsdichte in FFs/Sl.	Slices in #			Ausl. in %	Packungsdichte in LUTs/Sl.
DB	L	8	4	136	152	119	1,0	1,14	1,28	48	0,4	2,83	3,17	44	0,4	3,09	3,45	-59,7	-8,3
		32	4	544	608	489	4,2	1,11	1,24	192	1,6	2,83	3,17	176	1,5	3,09	3,45	-60,7	-8,3
		64	4	1088	1216	878	7,5	1,24	1,38	384	3,3	2,83	3,17	352	3,0	3,09	3,45	-56,3	-8,3
	LR	8	8	248	288	204	1,8	1,22	1,41	88	0,8	2,82	3,27	80	0,7	3,10	3,60	-56,9	-9,1
		32	8	992	1152	691	5,9	1,44	1,67	352	3,0	2,82	3,27	320	2,7	3,10	3,60	-49,1	-9,1
		64	8	1984	2304	1384	11,9	1,43	1,66	704	6,0	2,82	3,27	640	5,5	3,10	3,60	-49,1	-9,1
MEWB	L	115	4	1779	1949	1606	13,8	1,11	1,21	634	5,4	2,81	3,07	634	5,4	2,81	3,07	-60,5	0,0
	LR	115	8	3213	3688	2830	24,3	1,14	1,30	1038	8,9	3,10	3,55	1038	8,9	3,10	3,55	-63,3	0,0
	M	115	2	1407	1262	1332	11,4	1,06	0,95	553	4,8	2,54	2,28	490	4,2	2,87	2,58	-58,5	-11,4
∅					1059	9,1	1,21	1,35	444	3,81	2,82	3,14	419	3,60	3,04	3,37	-57,1	-7,1	

Tabelle A.23: Auswertung der Packungsdichte für homogene und inhomogene 2D DPR-Kommunikationsinfrastrukturen in der 7-Serie-Architektur (5x2 Partitionierung: Kintex-7 325T, andere Artix-7 100T)

Typ	Design					Inhom. Komm.-Makro (ISE)						Hom. Komm.-Makro (DHHarMa)				Vergleich Slices			
	Konfiguration Part.	Busbr.	PRR in #	FFs in #	LUTs in #	Standard MAP-Einstellungen			Optimierte MAP-Einstellungen			Standard Einstellungen				ISE in %	DHHarMa vs. MAP-Opt. in %		
						Slices in #	Ausl. in %	Packungsdichte in FFs/Sl.	Slices in #	Ausl. in %	Packungsdichte in LUTs/Sl.	Slices in #	Ausl. in %	Packungsdichte in FFs/Sl.	Slices in #			Ausl. in %	Packungsdichte in LUTs/Sl.
DB	2x2	8	4	136	152	121	0,8	1,12	1,26	40	0,3	3,40	3,80	45	0,3	3,02	3,38	-66,9	12,5
		32	4	544	608	440	2,8	1,24	1,38	160	1,0	3,40	3,80	165	1,0	3,30	3,68	-63,6	3,1
		64	4	1088	1216	859	5,4	1,27	1,42	320	2,0	3,40	3,80	325	2,1	3,35	3,74	-62,7	1,6
	3x2	8	6	168	216	151	1,0	1,11	1,43	56	0,4	3,00	3,86	63	0,4	2,67	3,43	-62,9	12,5
		32	6	672	864	585	3,7	1,15	1,48	224	1,4	3,00	3,86	231	1,5	2,91	3,74	-61,7	3,1
		64	6	1344	1728	1220	7,7	1,10	1,42	448	2,8	3,00	3,86	455	2,9	2,95	3,80	-63,3	1,6
SEWB	2x2	95	4	1186	1360	1172	7,4	1,01	1,16	555	3,5	2,14	2,45	434	2,7	2,73	3,13	-52,6	-21,8
	3x3	95	9	2036	2880	1745	11,0	1,17	1,65	909	5,7	2,24	3,17	811	5,1	2,51	3,55	-47,9	-10,8
	5x2	95	10	2434	3184	2493	4,9	0,98	1,28	1101	6,9	2,21	2,89	944	1,9	2,58	3,37	-55,8	-14,3
MEWB	2x2	115	4	1779	1985	1818	11,5	0,98	1,09	646	4,1	2,75	3,07	594	3,7	2,99	3,34	-64,5	-8,0
	3x3	115	9	3054	4035	1824	11,5	1,67	2,21	1051	6,6	2,91	3,84	1049	6,6	2,91	3,85	-42,4	-0,2
	5x2	115	10	3585	4445	3045	6,0	1,18	1,46	1270	2,5	2,82	3,50	1212	2,4	2,96	3,67	-58,3	-4,6
∅					1289	6,1	1,16	1,44	565	3,1	2,86	3,49	527	2,5	2,91	3,56	-58,6	-2,1	

Tabelle A.24: Auswertung der Packungsdichte für homogene und inhomogene, 2D DPR-Kommunikationsinfrastrukturen in der Virtex-4-Architektur (DB: LX15, EWB: FX100)

Typ	Design			Inhom. Komm.-Makro (ISE)						Hom. Komm.-Makro (DHHarMa)				Vergleich Slices in %
	Konfiguration Part.	Busbr.	PRR in #	FFs in #	LUTs in #	Slices in #	Ausl. in %	Packungsdichte		Slices in #	Ausl. in %	Packungsdichte		
							in FFs/Sl.	in LUTs/Sl.			in FFs/Sl.	in LUTs/Sl.		
DB	2x2	8	4	136	152	128	2,1	1,06	1,19	88	1,4	1,55	1,73	-31,3
		32	4	544	608	640	10,4	0,85	0,95	352	5,7	1,55	1,73	-45,0
		64	4	1088	1216	1280	20,8	0,85	0,95	672	10,9	1,62	1,81	-47,5
	3x2	8	6	168	216	192	3,1	0,88	1,13	112	1,8	1,50	1,93	-41,7
		32	6	672	864	768	12,5	0,88	1,13	448	7,3	1,50	1,93	-41,7
		64	6	1344	1728	1536	25,0	0,88	1,13	896	14,6	1,50	1,93	-41,7
SEWB	2x2	95	4	1186	1360	1554	3,7	0,76	0,88	898	2,1	1,32	1,51	-42,2
	3x3	95	9	2036	2880	2760	6,5	0,74	1,04	1668	4,0	1,22	1,73	-39,6
	5x2	95	10	2434	3184	3174	7,5	0,77	1,00	1936	4,6	1,26	1,64	-39,0
MEWB	2x2	115	4	1779	1985	2061	4,9	0,86	0,96	1120	2,7	1,59	1,77	-45,7
	3x3	115	9	3054	4035	3441	8,2	0,89	1,17	2040	4,8	1,50	1,98	-40,7
	5x2	115	10	3585	4445	3993	9,5	0,90	1,11	2362	5,6	1,52	1,88	-40,8
Ø						1794	9,5	0,86	1,05	1049	5,5	1,47	1,80	-41,4

C Evaluation

C.1 Datenübertragungsraten SHRC

Dynamisch partielle Rekonfiguration

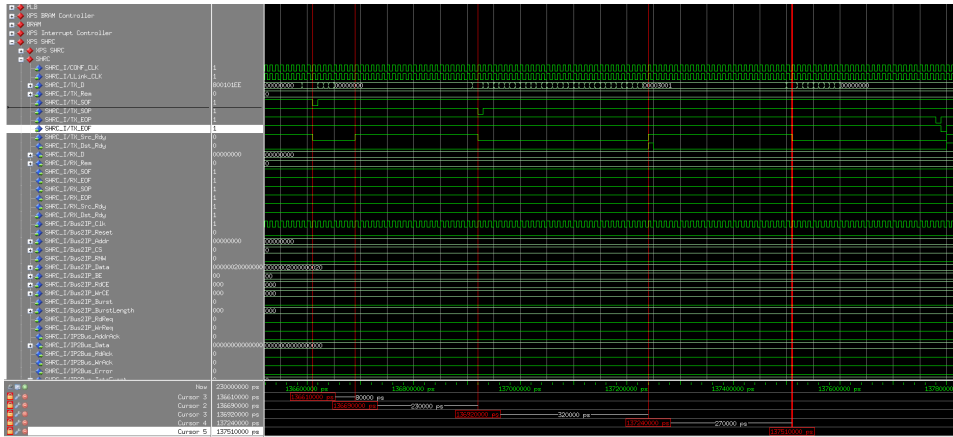


Abbildung A.7: Messergebnisse der Datenübertragung des SHRC über den Local Link TX Kanal (ModelSim) [165, S. 86]

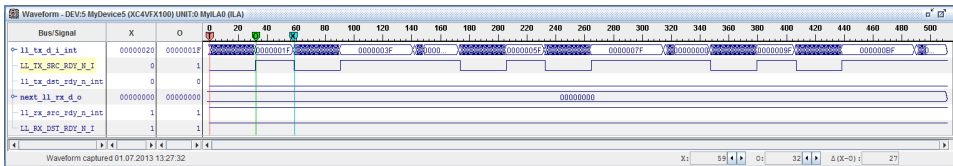


Abbildung A.8: Messergebnisse der Datenübertragung des SHRC über den Local Link TX (ChipScope)

Readback

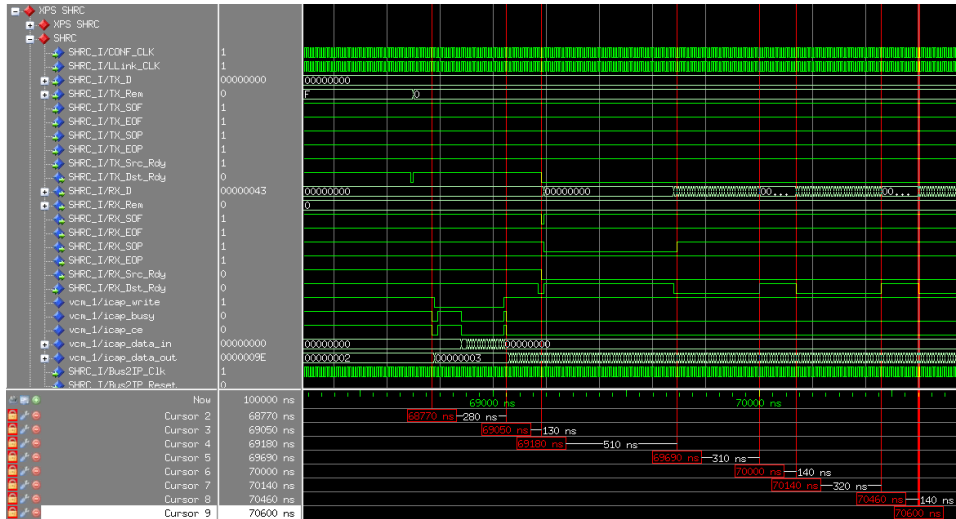


Abbildung A.9: Messergebnisse der Datenübertragung des SHRC über den Local Link RX Kanal (ModelSim) [165, S. 87]

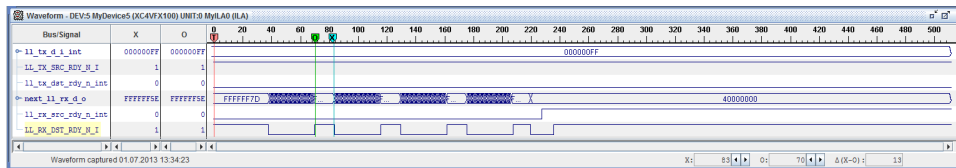


Abbildung A.10: Messergebnisse der Datenübertragung des SHRC über den Local Link RX Kanal (ChipScope)

C.2 Initiale Konfiguration des SpaceWire-RTCs

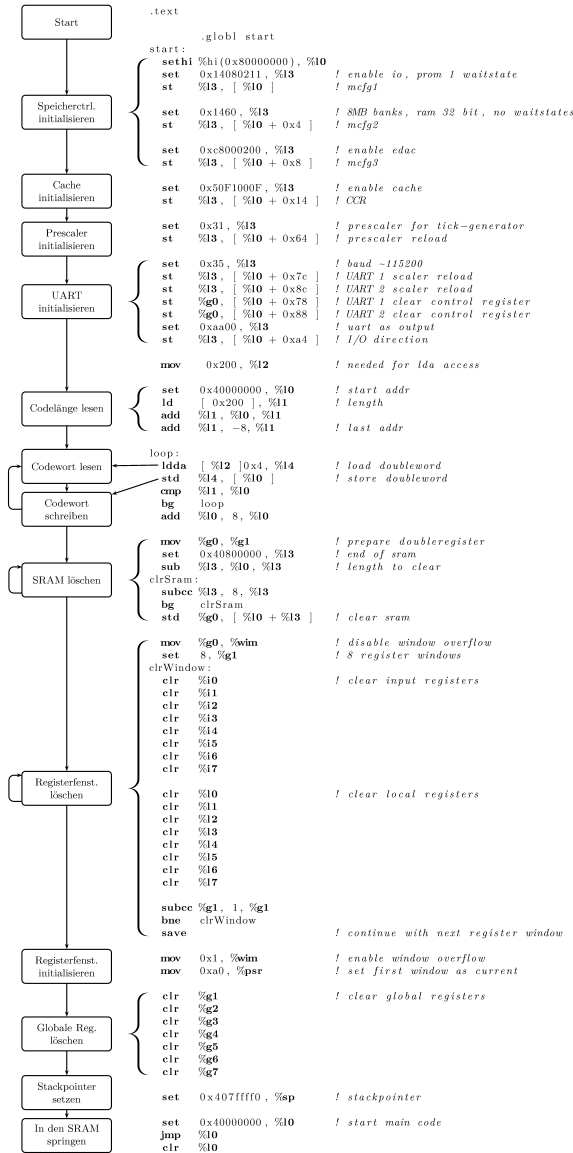


Abbildung A.12: Ablauf des Bootloaders [151, S. 25]

C.3 PR-Module

FFT

Tabelle A.25: Verarbeitungszeit, FPGA-Ressourcenbedarf und Leistung der FFT-Implementierungen

Abtastpunkte	Architektur	Verarbeitungszeit (VZ)		FFs		LUTs		FPGA-Ressourcen Slices		DSPs		BRAMs		PR-Tiles in #	Leistung in mW
		reine VZ in μ s	vollst. VZ in μ s	in #	Ausl. in %	in #	Ausl. in %	in #	Ausl. in %	in #	Ausl. in %	in #	Ausl. in %		
1024	Pipelined	16,4	16,4	5799	191	4685	154	3563	234	22	275	10	63	3	220,2
	Radix-4	13,8	34,3	3300	108	2857	94	2249	148	18	225	17	106	3	164,5
	Radix-2	52,9	73,4	1524	50	1485	49	1071	70	6	75	11	69	1	94,3
	Radix-2 lite	102,6	123,1	1167	38	953	31	816	54	4	50	11	69	1	80,5
2048	Pipelined	31,9	31,9	5491	181	6208	204	3953	260	28	350	17	106	4	243,4
	Radix-4	31,9	72,9	3565	117	3443	113	2447	161	18	225	17	106	3	174,7
	Radix-2	114,5	155,5	1620	53	1588	52	1143	75	6	75	15	94	1	104,8
	Radix-2 lite	225,5	266,5	1234	41	987	32	863	57	4	50	15	94	1	89,9
4096	Pipelined	62,7	62,7	5833	192	5907	194	3885	255	28	350	31	194	4	274,4
	Radix-4	62,7	144,6	3329	109	3729	123	2492	164	18	225	25	156	3	186,9
	Radix-2	247,9	329,8	1684	55	1721	57	1180	78	6	75	21	131	2	118,6
	Radix-2 lite	491,7	573,7	1279	42	1084	36	884	58	4	50	21	131	2	103,0

FIR

Tabelle A.26: Datendurchsatz, FPGA-Ressourcenbedarf und Leistung der FIR-Filter-Implementierungen

TAP	FIR-Arch.	Datendurchsatz in MSamples/s	FFs		LUTs		FPGA-Ressourcen Slices		DSPs		BRAMs		PR-Tiles in #	Leistung in mW
			in #	Ausl. in %	in #	Ausl. in %	in #	Ausl. in %	in #	Ausl. in %	in #	Ausl. in %		
16	SMAC	100	841	28	665	22	546	36	9	113	4	25	2	61,2
	TMAC	100	448	15	261	9	326	21	17	213	4	25	3	54,8
	SMAC	50	1113	37	622	20	696	46	6	75	4	25	1	65,0
	TMAC	50	465	15	316	10	365	24	9	113	4	25	2	49,6
64	SMAC	100	2002	66	2445	80	1565	103	31	388	4	25	4	118,0
	SMAC	50	2687	88	1403	46	1559	102	17	213	4	25	3	114,7
	SMAC	33	1308	43	953	31	812	53	12	150	4	25	2	76,5
	SMAC	20	1044	34	759	25	668	44	8	100	4	25	1	65,9

