Institut für Technische Informatik

Universität Stuttgart
Pfaffenwaldring 47
D–70569 Stuttgart

Bachelorarbeit Nr. 179

Adaptierung an Zeitverhalten-Variationen in rekonfigurierbaren Hardwarestrukturen

Sebastian Brandhofer

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. rer. nat. habil. Hans-Joachim Wunderlich
Betreuer/in:	Dr. rer. nat. Michael Kochte, Dipl.-Inf. Eric Schneider
Beginn am:	20. Oktober 2014
Beendet am:	21. April 2015
CR-Nummer:	B.7.2, B.8.1, B.8.2, C.4, J.6

Kurzfassung

Das Zeitverhalten von Komponenten in rekonfigurierbaren Hardwarestrukturen kann durch Alterungseffekte und zufällige Defekte variieren. Wenn ein System nicht an diese Abweichungen vom nominellen Zeitverhalten adaptiert werden kann, entstehen Verzögerungsfehler während des Betriebs, die zu falschen Ergebnissen oder Systemausfällen führen können. Insbesondere in sicherheitskritischen Anwendungen von rekonfigurierbaren Hardwarestrukturen kann dies zu Gefährdung von Personen führen.

Diese Arbeit stellt einen Algorithmus zur Adaptierung an Zeitverhalten-Variationen in rekonfigurierbaren Hardwarestrukturen vor, der Alterung von Komponenten sowie zufällige Defekte berücksichtigt und Verzögerungsfehler durch eine dem Zeitverhalten angepasste Nutzung der rekonfigurierbaren Hardwarestrukturen vermeidet. Der entworfene Algorithmus wird mit Hilfe von verschiedenen Verzögerungsverteilungen hinsichtlich der Adaptionsfähigkeit, Speicheranforderungen und Laufzeit untersucht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Rekonfigurierbare Hardwarestrukturen werden in vielen sicherheitskritischen Anwendungen eingesetzt. Beispielsweise unterstützen sie Stellwerksanlagen der Deutschen Bahn [Deu13]. Kommt es dabei zu Fehlern, können sie zu Unfällen führen und damit Personen gefährden. Effekte wie Alterung verändern das zeitliche Schaltverhalten in rekonfigurierbaren Strukturen und können so Fehler verursachen. Wenn diese Variationen adaptiert werden können, dann steigt die Zuverlässigkeit solcher Strukturen. Ein längerer fehlerfreier Betrieb wird damit ermöglicht, und die Gefährdung, die sich durch die Nutzung dieser Technik in sicherheitskritischen Anwendungen ergibt, wird verringert.

Rekonfigurierbare Hardwarestrukturen lassen sich zur Laufzeit rekonfigurieren, sodass sich die Konfiguration der Struktur verändert und dadurch eine andere Funktion abgebildet wird. Durch eine Rekonfiguration kann z.B. ein Hardwarebereich, der einen Addierer realisiert, so konfiguriert werden, dass stattdessen das Produkt dieser Zahlen berechnet wird. Diese Rekonfigurierbarkeit kann auch genutzt werden, um das System an Variationen des Zeitverhaltens zu adaptieren.

Ziele der Arbeit

Ziel der Arbeit ist der Entwurf eines Algorithmus, der das erwartete Zeitverhalten eines Hardwareentwurfs analysiert und zur Laufzeit mit dem gemessenen Zeitverhalten vergleicht. Durch diesen Vergleich bestimmt der Algorithmus die zusätzliche Signalverzögerung der einzelnen Komponenten der rekonfigurierbaren Hardwarestruktur, die durch Zeitverhalten-Variationen verursacht wurden. Mit dieser Information kann der Algorithmus entscheiden, welcher Hardwareentwurf in welchem Bereich der rekonfigurierbaren Hardwarestruktur konfiguriert werden kann, ohne dass es durch zusätzliche Signalverzögerungen zu Verzögerungsfehlern und schließlich zur Störung des Betriebs kommt. Anschließend wird untersucht, unter welchen Bedingungen und in welcher Größenordnung dieser Ansatz an Variationen des Zeitverhaltens adaptieren kann.

Im Betrieb müssen rekonfigurierbare Hardwarestrukturen durch weitere Komponenten verwaltet werden. Zu den Aufgaben dieser Verwaltungskomponenten zählen u.a. die Auswahl und das Starten einer Rekonfiguration. Diese Entscheidungen und damit auch die Ausführung des entworfenen Algorithmus werden oft von einem eingebetteten System übernommen und müssen innerhalb kurzer Zeit abgeschlossen werden, um die Verzögerung, die sich durch eine Rekonfiguration ergibt, zu minimieren. Der Algorithmus soll entsprechend geringe Ressourcenanforderungen an das System stellen. Dadurch korrelieren Speicherbedarf, Laufzeit des Algorithmus und die festgelegte Rekonfigurationsdauer, mit den Kosten des eingebetteten Systems. Ein weiteres Ziel der Arbeit ist daher die Untersuchung und Minimierung des Speicherbedarfs und der Laufzeit des entworfenen Algorithmus.

Übersicht

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Rekonfigurierbare Hardwarestrukturen: Hier werden rekonfigurierbare Hardwarestrukturen und deren dynamische Rekonfiguration erklärt.

Kapitel 3 – Zeitverhalten und Verzögerungsfehler: Die Grundlagen des Zeitverhaltens werden hier erläutert.

Kapitel 4 – Algorithmus zur Adaptierung an Zeitverhalten-Variationen beschreibt den entworfenen Algorithmus.

Kapitel 5 – Ergebnisse stellt die Ergebnisse zur Adaptionsfähigkeit und Speichereffizienz des Verfahrens dar.

Kapitel 6 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

2 Rekonfigurierbare Hardwarestrukturen

Hardwarestrukturen sind rekonfigurierbar, wenn sich ihre Struktur mehrmals konfigurieren lässt und sie dadurch zu unterschiedlichen Zeitpunkten unterschiedliche Hardwareentwürfe implementieren können.

In Abschnitt 2.1 wird ein Überblick über rekonfigurierbare Hardwarestrukturen gegeben und Möglichkeiten der Klassifizierung aufgezeigt. Anschließend wird in Abschnitt 2.2 erklärt wie FPGAs, die aufgrund ihrer rekonfigurierbaren Hardwarestruktur oft genutzt werden, funktionieren. Zuletzt wird die dynamische Rekonfiguration in Abschnitt 2.3 behandelt.

2.1 Klassifizierung

Rekonfigurierbare Hardwarestrukturen lassen sich anhand der Art der Konfiguration klassifizieren. Abbildung 2.1 zeigt die Klassen rekonfigurierbarer Hardwarestrukturen, die sich dadurch ergeben.

Rekonfigurierbare Hardwarestrukturen lassen sich entweder statisch oder dynamisch rekonfigurieren. Bei einer statischen Rekonfiguration ändert sich die Konfiguration der Hardwarestruktur einmalig, wonach der Hardwareentwurf in Betrieb genommen wird. Wenn ein anderer Hardwareentwurf implementiert werden soll, muss die rekonfigurierbare Hardwarestruktur zunächst neu gestartet werden. Anschließend lässt sie sich erneut konfigurieren. Bei einer dynamischen rekonfigurierbaren Hardwarestruktur lässt sich die Konfiguration der Struktur mehrmals während des Betriebs ändern. Dabei wird zwischen einer vollständigen und partiellen dynamischen Rekonfiguration unterschieden, wobei die Konfiguration entweder vollständig oder nur teilweise während des Betriebs geändert werden kann [Koc13]. Hardwarestrukturen, die sich nur einmalig konfigurieren lassen [MBMB07] werden jedoch, aufgrund der geringen Flexibilität, nicht in dieser Arbeit behandelt. Abhängig vom Detailgrad der Veränderung, der durch die rekonfigurierbare Hardwarestruktur unterstützt wird, werden Hardwarestrukturen zudem in feingranular oder grobgranular eingeteilt [BSV10]. In rekonfigurierbaren Hardwarestrukturen können grundlegende kombinatorische und sequentielle Komponenten

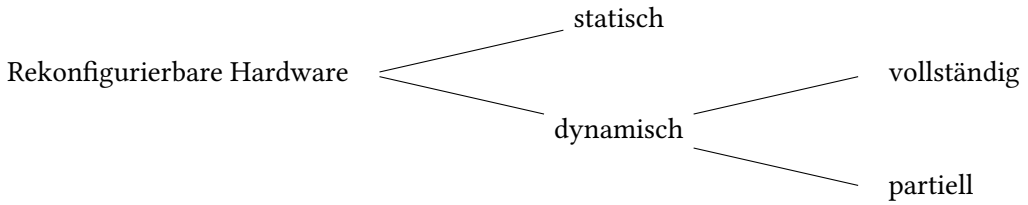


Abbildung 2.1: Klassifizierung rekonfigurierbarer Hardwarestrukturen nach Art der Konfiguration

angepasst und neu verbunden werden, um ein bestimmten Zielentwurf abzubilden. Feingranulare Hardwarestrukturen lassen sich auf Bit- bzw. Gatterebene konfigurieren. Bei grobgranularen Hardwarestrukturen werden komplexe Komponenten neu verschaltet.

2.2 Field Programmable Gate Array

Field Programmable Gate Arrays (FPGAs) haben eine rekonfigurierbare Hardwarestruktur. Die Art der Konfiguration eines FPGAs kann dabei, je nach Hersteller und Modell, fast allen oben genannten Klassen entsprechen. Diese Arbeit konzentriert sich auf FPGAs der Firma Xilinx und wird im Rahmen des OTERA [OTE] Projekts durchgeführt. Allerdings lässt sich der Ansatz auch auf partiell dynamisch rekonfigurierbare FPGAs anderer Firmen anwenden.

Ein FPGA ist ein programmierbarer integrierter Schaltkreis, der aus einer Menge von sequentielle Komponenten (Flipflops, RAM) und Booleschen Funktionsgeneratoren besteht, die über Verbindungsleitungen miteinander verbunden werden können. Die genaue Funktion des FPGAs muss durch den Anwender programmiert werden. Bei der Programmierung werden die Booleschen Funktionsgeneratoren und Verbindungsleitungen so konfiguriert, dass sie einen Zielhardwareentwurf implementieren. Daher wird die Programmierung des FPGAs auch Konfiguration bzw. Rekonfiguration genannt. Die Konfiguration eines FPGAs wird durch eine externe Komponente in einen eingebetteten Speicher des FPGAs geladen. Dieser Konfigurationsspeicher legt das Verhalten aller konfigurierbaren Verbindungsleitungen und Komponenten fest.

FPGAs bieten durch die Rekonfigurationsfähigkeit im Vergleich zu anwendungsspezifischen integrierten Schaltungen (ASIC) eine höhere Flexibilität. Dadurch lassen sich Hardwareentwürfe leichter korrigieren und erweitern. Zudem sind die Entwicklungskosten geringer, da die Entwicklungszeit im Schnitt 55% kürzer ist und keine Masken hergestellt werden müssen [Mat09].

Im Folgenden wird zunächst der generelle Aufbau von FPGAs dargestellt. Anschließend wird auf die Komponenten des FPGAs eingegangen und die notwendigen werden Implementierungsschritte skizziert.

2.2.1 Architektur

Der Aufbau von FPGAs ist regulär: Die Komponenten sind in einer regelmäßigen Struktur, typischerweise einem zweidimensionalem Feld, angeordnet. Diese sind mit konfigurierbaren Schnittpunkten verbunden. Durch Schaltmatrizen werden die FPGA Komponenten miteinander verbunden. Folgende Merkmale sind in modernen FPGAs zu finden und können wie in Abbildung 2.2 angeordnet werden:

- Konfigurierbare logische Blöcke (CLB), die aus Booleschen Funktionsgeneratoren und Speicherelementen bestehen.
- Verbindungsleitungen, die über konfigurierbare Schnittpunkte programmiert werden können.
- Eingebetteter Konfigurationsspeicher, in den die Konfiguration der konfigurierbaren Elemente geladen wird.

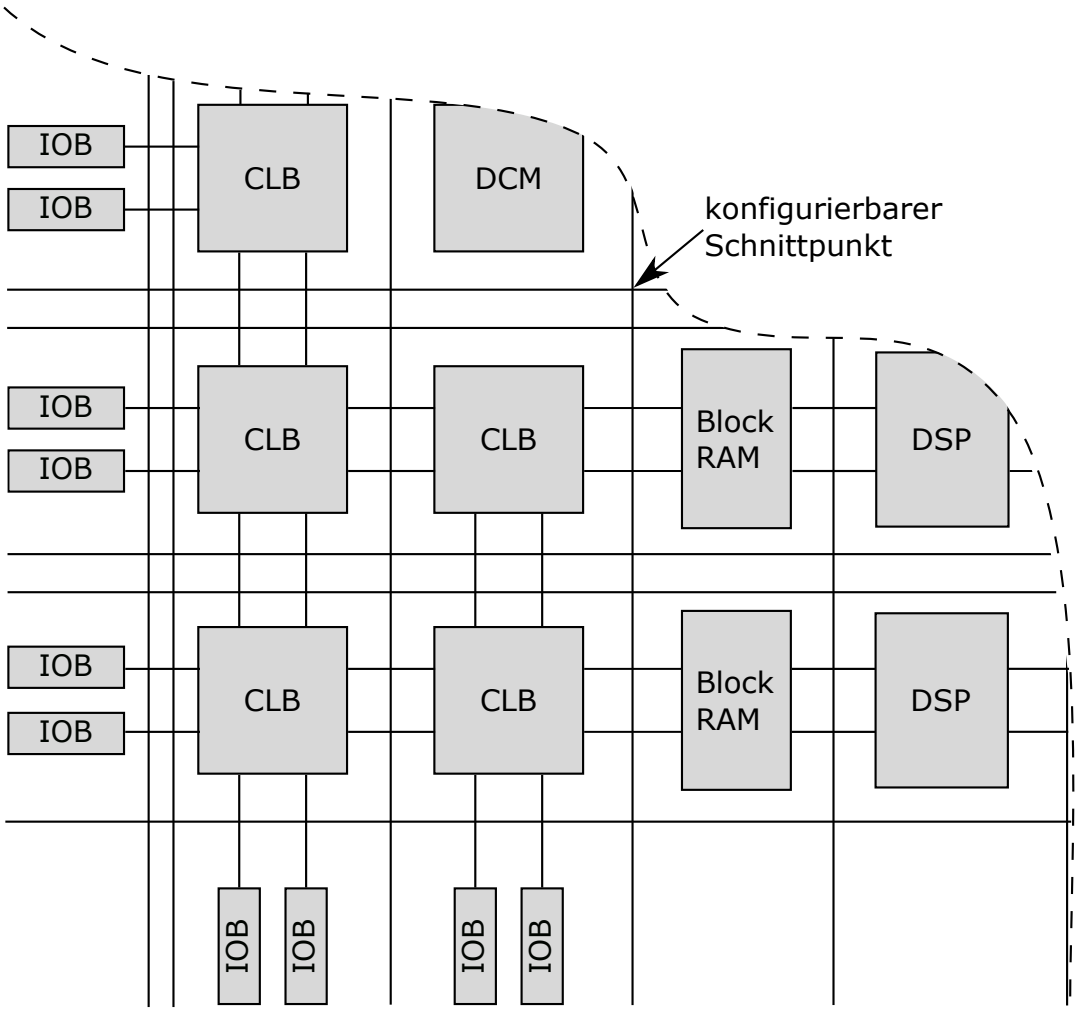


Abbildung 2.2: Ausschnitt der typischen Struktur eines modernen FPGAs (untere linke Ecke des zweidimensionalen Feldes)

- Ein-/Ausgabe Ports, die über Ein-/Ausgabe Blöcke (IOB) mit der Logik des FPGAs verbunden werden.
- IP-Cores, wie z.B. digitale Signalprozessoren (DSP) und eingebetteter Speicher (Block RAM)
- Infrastruktur zur Generierung (DCM) und Verbreitung des Taktsignals

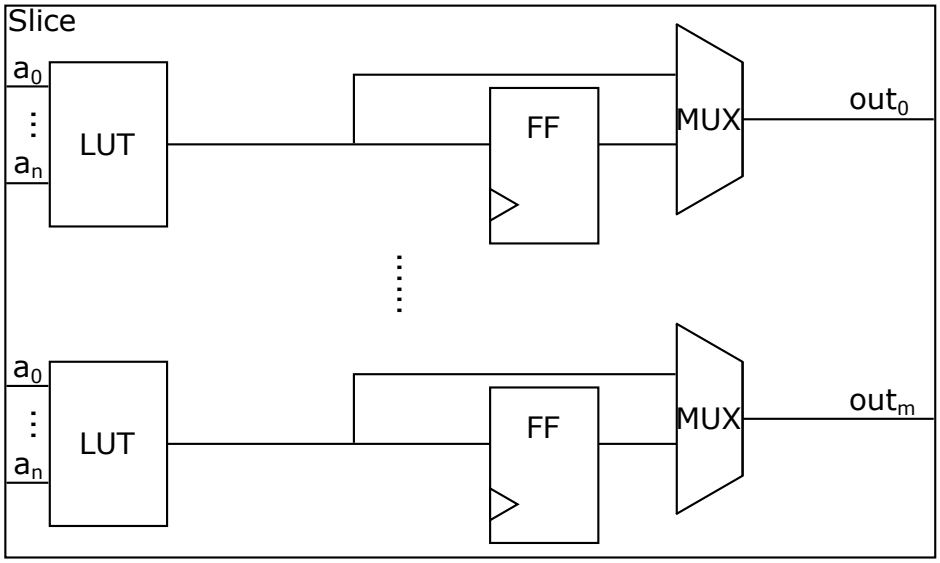


Abbildung 2.3: Allgemeiner Aufbau eines Slices

Konfigurierbare Schnittpunkte

Die Schnittpunkte der Verbindungsleitungen beinhalten konfigurierbare Schaltelemente. Abhängig vom Inhalt des Konfigurationsspeichers verbinden diese Schaltelemente Leitungsenden, die im Schnittpunkt aufeinander treffen, miteinander oder isolieren sie voneinander.

Konfigurierbare logische Blöcke

Konfigurierbare logische Blöcke (CLB) enthalten logische Funktionsgeneratoren, die Boolesche Funktionen implementieren und Speicherelemente (Flipflops und Latches), die optional genutzt werden können. Die CLB Komponenten werden in Slices zusammengefasst, die separat mit konfigurierbaren Schnittpunkten verbunden sind.

Die Anzahl und der Aufbau der Slices variiert je nach FPGA. Abbildung 2.3 zeigt den generellen Aufbau. Die logischen Funktionsgeneratoren werden über einen Multiplexer wahlweise mit einem Speicherelement, wie z.B. einem Flipflop (FF) oder einem Slice-Ausgang verbunden. In der dargestellten Abstraktion nicht sichtbar, sind weitere Multiplexer und Logikelemente, die in vielen Slicearchitekturen vorhanden sind und die Ausgänge der logischen Funktionsgeneratoren miteinander verknüpfen. Durch diese lokale Verknüpfung wird die Schaltmatrix in manchen Szenarien vermieden, sodass bestimmte arithmetischen Operationen, wie z.B. die Addition, effizienter implementiert werden können.

Ein logischer Funktionsgenerator wird durch eine programmierbare Wahrheitstabelle sog Lookup-Tabellen (LUT) realisiert. Beim Konfigurationsvorgang wird in den Konfigurationsspeicher (meistens SRAM [Dub08]) der LUT die Wahrheitstabelle der Booleschen Funktion gespeichert, die durch die LUT

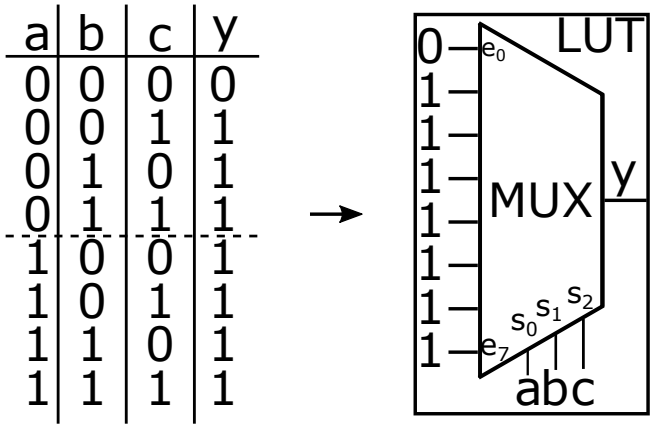


Abbildung 2.4: Aufbau einer LUT, die ein Oder-Gatter mit drei Eingängen implementiert

implementiert werden soll. Die Eingangssignale der LUT geben dabei die Adresse der Speicherzelle, bzw. die Zeile der Wahrheitstabelle an. Die LUT ist also ein Multiplexer, der je nach Belegung der Steuerungssignale, einen Wert der Wahrheitstabelle ausgibt. Abbildung 2.4 zeigt eine LUT mit drei Eingängen, die die Boolesche Funktion $f = a \vee b \vee c$, generiert. Diese LUT berechnet damit dieselbe Boolesche Funktion wie ein Oder-Gatter mit drei Eingängen. Durch Abspeicherung einer Wahrheitstabelle kann eine LUT mit n Eingängen jede beliebige Funktion $F : \mathbb{B}^n \to \mathbb{B}^1$ implementieren, wobei $\mathbb{B} = \{0, 1\}$ ist. Die Anzahl Eingänge einer LUT hängt von der FPGA Architektur ab. Virtex-5 FPGAs der Firma Xilinx besitzen LUTs mit sechs Eingängen [Xil12].

IP-Cores

Neben den konfigurierbaren logischen Blöcken können weitere Komponenten in der Matrixstruktur des FPGAs eingebunden sein. Häufig sind zusätzliche Speicherelemente vorhanden, die für Zustandselemente eines Entwurfs reserviert sind. Zudem sind oft Hardwarebeschleuniger eingebaut, die genutzt werden können, um Teile eines Hardwareentwurfs beschleunigt auszuführen. Hardwarebeschleuniger die in vielen FPGAs eingebaut sind, können beispielsweise Multiplikationen effizient berechnen, ohne dass weitere logische Blöcke benötigt werden. In der Regel sind diese Komponenten proprietär, und ihre Struktur wird von den FPGA Herstellern nicht veröffentlicht.

Ein-/Ausgabe Blöcke

Input/Output Blöcke (IOB) bilden die Schnittstelle zwischen Physischen Pins und den Komponenten des FPGAs. Modernen FPGAs besitzen viele verschiedene Pins, um die Hardwareentwürfe möglichst vieler Anwendungsfelder zu unterstützen.

2.2.2 Konfigurationsgenerierung

Damit Hardwareentwürfe, die durch Hardwarebeschreibungssprachen wie VHDL oder Verilog definiert werden, auf dem FPGA implementiert werden können, muss die Beschreibung in eine Konfiguration der CLBs und konfigurierbaren Verbindungsleitungen übersetzt werden. Anschließend muss die Konfiguration in den Konfigurationsspeicher des FPGAs geladen werden. Dieser Abschnitt behandelt wie, ausgehend von einer Hardwarebeschreibungssprache, eine Konfiguration eines Hardwareentwurfs generiert werden kann. Dafür muss die Beschreibung des Hardwareentwurfs zunächst synthetisiert und anschließend eine Platzierung und Verbindung der Komponenten berechnet werden.

Synthese

Um die Hardwarebeschreibung eines Hardwareentwurfs zu synthetisieren, wird zunächst die Eingabe auf Syntaxfehler überprüft, bevor sie weiter analysiert wird [MBMB07]. Anschließend werden Informationen aus der Hardwarebeschreibung extrahiert, die das Verhalten des Hardwareentwurfs definieren. Dabei werden Bausteine, wie z.B. Multiplexer, RAM, Addierer, etc., des Verhaltens inferiert und Zustandsmaschinen, die sich im Verhalten befinden, erkannt. Aus diesen Informationen wird eine strukturelle Netzliste generiert, die das extrahierte Verhalten abbildet [Xil13a]. Diese Netzliste kann simuliert werden, um zu überprüfen ob die Beschreibung des Hardwareentwurfs der Spezifikation entspricht. Oft werden Optimierungen an der Netzliste durchgeführt, bevor die strukturelle Netzliste an die Implementierung weitergegeben wird. Die Synthese und die strukturelle Netzliste sind unabhängig von der Technologie, die den Hardwareentwurf abbilden soll.

Implementierung

Um die strukturelle Netzliste des Hardwareentwurfs auf die Hardwarekomponenten eines FPGAs abzubilden, muss eine Implementierung durchgeführt werden. Durch die Implementierung wird die technologieunabhängige strukturelle Netzliste in eine Konfigurationsbeschreibung übersetzt, die auf einen FPGA geladen werden kann. Die Implementierung umfasst bei Xilinx vier Schritte [Xil13a]:

1. Translate
2. Map
3. Place and Route
4. Generierung der Konfigurationsdatei

Translate Während des 'Translate' Vorgangs wird die strukturelle Netzliste in eine Beschreibung des Hardwareentwurfs übersetzt, die aus Primitiven wie Und-Gattern, Oder-Gattern, LUTs, Flipflops und RAMs besteht. Die logische Beschreibung wird anschließend abgespeichert und dem Map Prozess übergeben.

Map Der Map Schritt liest die Beschreibung eines Hardwareentwurfs ein und bildet sie auf FPGA Elemente (CLB, Ein-/Ausgabe Blöcke, Hardwarebeschleuniger, etc.) ab.

Place and Route Die durch Map errechneten FPGA Elemente werden in der Place and Route Phase auf dem FPGA platziert und die Verbindungsleitungen, die die Elemente miteinander verbinden, werden berechnet. Die Place and Route Phase kann durch benutzerdefinierte Regeln gesteuert werden. Beispielsweise lässt sich dem Place and Route Prozess mitteilen, dass bestimmte FPGA Ressource vermieden werden sollen oder dass eine Zielfrequenz erreicht werden soll. Können die Nebenbedingungen durch Place and Route nicht erfüllt werden, so bricht der Prozess ab und meldet dies dem Nutzer.

Generierung der Konfigurationsdatei Nach erfolgreicher Ausführung der ersten drei Schritte, wurde errechnet welche Ressourcen des FPGAs genutzt werden müssen, um den Hardwareentwurf zu abzubilden. Zusätzlich ist das Verhalten der genutzten Ressourcen definiert und die Route aller Verbindungsleitungen wurde berechnet. Es liegt also eine Beschreibung des Hardwareentwurfs vor, der die Konfiguration der rekonfigurierbaren Elemente definiert. Diese Beschreibung muss noch in eine Form gebracht werden, die in den eingebetteten Konfigurationsspeicher des FPGAs geladen werden kann. Dafür wird ein Bitstream aus der Konfigurationsbeschreibung generiert, der die Konfiguration aller rekonfigurierbaren Elemente auf Bit-Ebene definiert. Dieser Stream kann anschließend in den Konfigurationsspeicher des FPGAs geladen werden.

2.3 Dynamische Rekonfiguration

Bei einer dynamischen Rekonfiguration wird ein Hardwareentwurf, der auf eine rekonfigurierbare Hardwarestruktur konfiguriert wurde, im laufenden Betrieb modifiziert. Die dynamische Rekonfiguration von rekonfigurierbaren Hardwarestrukturen wird im Folgenden anhand von Xilinx FPGAs vorgestellt. Bei Xilinx FPGAs wird während einer dynamischen Rekonfiguration der Bitstream, der sich im Konfigurationsspeicher des FPGAs befindet und die Konfiguration eines Hardwareentwurfs abbildet, teilweise überschrieben. Um die dynamische Rekonfiguration durchzuführen, existieren ein Modul-basierter und ein Differenz-basierter Ansatz.

Für den Modul-basierten Ansatz muss ein Hardwareentwurf zunächst in statische und rekonfigierbare Logik eingeteilt werden [Xil13b]. Die statische Logik wird von der dynamischen Rekonfiguration nicht beeinflusst und führt den normalen Betrieb fort, während die rekonfigurierbare Logik neu konfiguriert werden kann. Ein rekonfigurierbarer Hardwarebereich muss dafür, wie in Abbildung 2.5 dargestellt, definiert werden. Die statische Logik eines Entwurfs wird im Implementierungsschritt außerhalb des rekonfigurierbaren Bereichs platziert und vor dem Betrieb in den FPGA konfiguriert. Der rekonfigurierbare Bereich wird ebenso vor dem Betrieb mit einer Konfiguration der rekonfigurierbaren Logik initialisiert bzw. konfiguriert und kann im Betrieb mit der rekonfigurierbaren Logik erneut konfiguriert werden. Die rekonfigurierbare Logik eines Entwurfes kann aus mehreren Modulen bestehen, die zu unterschiedlichen Zeiten in einen rekonfigurierbaren Hardwarebereich konfiguriert und in Betrieb genommen werden. Von jedem Modul der rekonfigurierbaren Logik wird mindestens ein

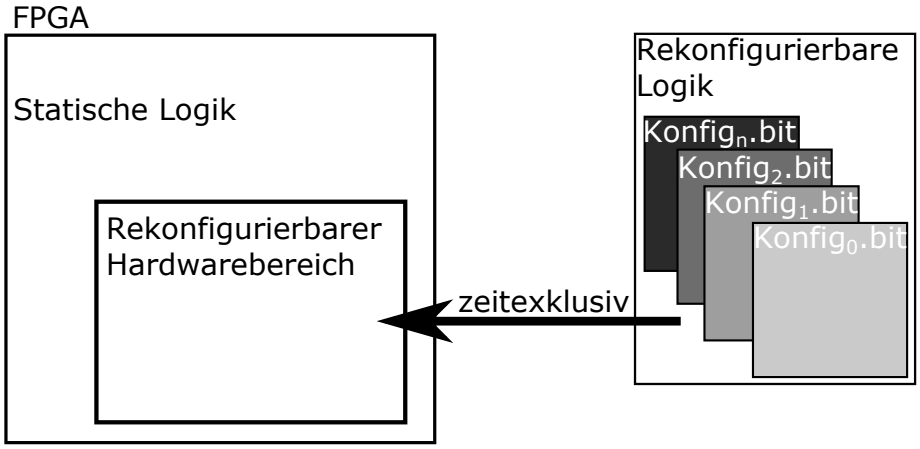


Abbildung 2.5: Definition eines rekonfigurierbaren Hardwarebereichs

Konfigurations-Bitstream erzeugt. In Teil 2.3.2 dieses Abschnitts wird erklärt, wie ein oder mehrere rekonfigurierbare Hardwarebereiche definiert und genutzt werden können.

Bei beiden Ansätzen muss der Entwurf die dynamische Rekonfiguration berücksichtigen und Komponenten bzw. Bereiche während sie rekonfiguriert werden, nicht nutzen. Der Rest dieses Abschnitts befasst sich zunächst mit den Vorteilen, die sich durch die Nutzung von dynamischer Rekonfiguration ergeben. Anschließend werden die Definition von rekonfigurierbaren Hardwarebereichen und das Vorgehen beim Laden von Konfigurationen in den FPGA skizziert. Abschließend wird auf die Grenzen bzw. Einschränkungen dynamischer Rekonfiguration eingegangen.

2.3.1 Vorteile

Durch die Nutzung dynamischer Rekonfiguration können sich für einen Entwurf einige Vorteile ergeben [Xil13b, Koc13]. Diese beinhalten:

- Reduzierung des FPGA-Ressourcen- und Energiebedarfs – Mehrere Funktionen können in einem rekonfigurierbaren Hardwarebereich implementiert werden, falls sie nicht zur gleichen Zeit benötigt werden. Damit kann sich der Platzbedarf eines Entwurfs verringern, was zu weniger aktivierten FPGA Komponenten und damit zu einem geringeren Energiebedarf führt.
- Flexibilität bezüglich verfügbarer Algorithmen und Protokolle – In einem Hardwarebereich können, je nach Anforderung einer Anwendung, unterschiedliche Algorithmen konfiguriert werden, die der Anwendungen nach der Rekonfiguration zur Verfügung stehen.
- Erhöhung der Fehlertoleranz in FPGAs – Um die gleiche Funktion zu realisieren, können unterschiedliche Entwürfe (N-Versionen Programmierung) oder Konfigurationen eines Entwurfs in einen Hardwarebereich konfiguriert und genutzt werde. Diese Konfigurationen nutzen unterschiedliche Komponenten eines Hardwarebereichs. Durch Zeitverhalten-Variationen kann die Nutzung von Komponenten eines Hardwarebereichs zu Fehlern führen. Diese Fehler können

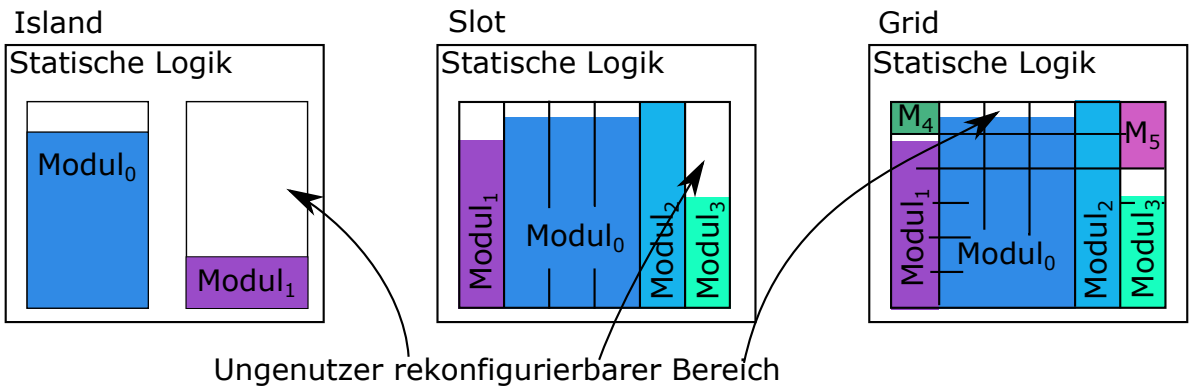


Abbildung 2.6: Island-, Slot- und Gridnutzung rekonfigurierbarer Hardwarebereiche

toleriert werden, indem eine Konfiguration in den Hardwarebereich konfiguriert wird, die fehlerhafte Komponenten nicht nutzt und die Anforderungen des System erfüllt.

- Beschleunigung von Berechnungen – Anstatt der Ausführung von Software durch einen Prozessor können Funktionen durch Hardwarebeschleuniger berechnet werden, die in einen rekonfigurierbaren Hardwarebereich geladen werden.

2.3.2 Definition und Nutzung von rekonfigurierbaren Hardwarebereichen

Die Definition von rekonfigurierbaren Hardwarebereichen wird von Xilinx hinsichtlich der Größe und Form eingeschränkt [Xil13b]. Beispielsweise können nur komplette CLBs zu einem rekonfigurierbaren Hardwarebereich hinzugefügt werden. Es ist nicht möglich, nur ein Slice eines CLBs zu einem rekonfigurierbaren Hardwarebereich hinzuzufügen, während das andere Slice außerhalb des rekonfigurierbaren Hardwarebereichs liegt. Wenn die Form eines rekonfigurierbaren Hardwarebereichs nicht rechteckig ist, kann es zu Problemen beim Routing kommen [Xil13b] Die Nutzung der definierten rekonfigurierbaren Hardwarebereiche kann ebenfalls variieren. Dabei existiert, wie in Abbildung 2.6 dargestellt, eine Slot-,Island- und Gridnutzung.

Island

Bei der Islandnutzung von rekonfigurierbaren Hardwarebereichen sind die Bereiche voneinander isoliert und können mit Modulen der rekonfigurierbaren Logik konfiguriert werden [Koc13]. Ist es dabei möglich, ein Modul in verschiedenen rekonfigurierbaren Bereichen zu konfigurieren, wird die Nutzung auch 'multi island' genannt. Falls nur ein bestimmtes Modul in jeweils einen Bereich konfiguriert werden kann, spricht man von 'single island' Nutzung. Bei der 'multi island' Nutzung müssen die verschiedenen Module der rekonfigurierbaren Logik eine gemeinsame Schnittstelle für die Kommunikation mit der statischen Logik eines Entwurfs zur Verfügung stellen. Diese Schnittstelle wird in jedem rekonfigurierbaren Hardwarebereich an eine fest definierte Stelle platziert, damit die statische Logik, unabhängig von der Konfiguration des Bereichs, mit der Schnittstelle verbunden

werden kann. Ein Hardwarebereich, der zu unterschiedlichen Zeiten mit verschiedenen Modulen rekonfiguriert werden kann, muss die Ressourcen- und Kommunikationsanforderungen (Größe der oben genannten Schnittstelle) dieser Module erfüllen. Dadurch kann es zu einer ineffizienten Nutzung der rekonfigurierbaren Bereiche kommen, da ein simples Modul, das nur wenige Ressourcen benötigt ebenso viel Platz auf dem FPGA einnimmt, wie ein komplexes Modul. Xilinx Werkzeuge unterstützen beide Islandnutzungstypen.

Slot

Bei der Slotnutzung von rekonfigurierbaren Hardwarebereichen entspricht jeder Bereich einem Slot. Ein Modul der rekonfigurierbaren Logik kann mehrere Slots bzw. rekonfigurierbare Hardwarebereiche nutzen, um in den FPGA konfiguriert zu werden. Ein Vorteil dieser Methode ist, dass rekonfigurierbare Hardwarebereiche effizienter genutzt werden, da die Ressourcenanforderungen der Module flexibler gelöst werden können. Die Slotnutzung von rekonfigurierbaren Bereichen wird nicht von Xilinx Werkzeugen unterstützt, allerdings existiert ein alternatives Werkzeug namens ReCoBus [KBT08].

Grid

Die Gridnutzung von rekonfigurierbaren Hardwarebereichen hat Ähnlichkeiten zur Slotnutzung. Hierbei kann ein Modul mehrere rekonfigurierbare Bereiche, die als Kacheln in einem zweidimensionalen Grid angeordnet sind, in Anspruch nehmen. Dadurch ist eine effizientere Nutzung des rekonfigurierbaren Bereichs möglich. Bisher existieren jedoch keine automatisierten Werkzeuge, die dies unterstützen. Die dynamische Rekonfiguration wird bei dieser Methode auch komplexer, da bei der Rekonfiguration entschieden werden muss, welche Kacheln des Grids mit welchem Modul zu welchem Zeitpunkt konfiguriert werden. Daraus ergibt sich ein dreidimensionales Packproblem, das zur Laufzeit gelöst werden muss [Koc13]. Zudem muss eine Kommunikationsstruktur implementiert werden, die den Status der rekonfigurierbaren Kacheln kennt und die Kommunikation zwischen der dort konfigurierten rekonfigurierbaren Logik mit der statischen Logik ermöglicht.

2.3.3 Laden der Bitstreams in den Konfigurationsspeicher

Im Allgemeinen wird ein Prozessor bzw. eine Zustandsmaschine genutzt, um ein Bitstream, der sich in einem nicht-flüchtigen Datenspeicher befindet, auszuwählen und zu einem Konfigurationsport zu transportieren [Xil13b]. Dabei kann sich der Prozessor bzw. die Zustandsmaschine innerhalb oder außerhalb des zu rekonfigurierenden FPGAs befinden. Externe Prozessoren können die JTAG, SelectMAP oder serielle Schnittstelle des FPGAs nutzen, um eine Rekonfiguration zu steuern. Die dynamische Rekonfiguration kann auch von Logik innerhalb des FPGAs gesteuert werden. Dies nennt sich Selbstrekonfiguration und dafür muss ein interner Konfigurationsport (ICAP) auf dem FPGA instanziiert und angesteuert werden. Der interne Konfigurationsport kann über eine interne Zustandsmaschine bzw. Prozessoren wie der MicroBlaze angesteuert werden. Die Konfigurationsports empfangen den übermittelten Bitstream und senden ihn an den Konfigurationsspeicher des rekonfigurierbaren Bereichs, der im Bitstream spezifiziert wurde [Xil13b].

Die Dauer der Rekonfiguration ist abhängig von der Größe des verwendeten Bitstreams und von der Bandbreite des verwendeten Konfigurationsports [Xil13b]. Der SelectMAP und ICAP Konfigurationsport haben mit $400\frac{\text{MB}}{\text{s}}$ die größte Bandbreite [Xil13b]. Angenommen ein Bitstream, der ein Xilinx Virtex XC7V2000T FPGA vollständig konfigurieren kann, soll über einen SelectMAP Konfigurationsport konfiguriert werden. Das Virtex XC7V2000T FPGA besitzt 152700 CLBs. Ein solcher Bitstream hat eine Größe von fast 56MB [Koc13]. Daraus ergibt sich für die ungefähre Konfigurationszeit einer vollständigen Konfiguration des FPGAs in diesem Szenario:

$$\text{Konfigurationszeit } = \frac{56\text{MB}}{400\frac{\text{MB}}{\text{s}}} \approx 140\text{ms}.$$

2.3.4 Einschränkungen & Nachteile

Die Nutzung von dynamischer Rekonfiguration in Xilinx FPGAs unterliegt einigen Einschränkungen bzw. führt zu einigen Nachteilen, die nun vorgestellt werden:

- Einige Xilinx Komponenten, wie z.B. der Taktgeber, lassen sich nicht im Betrieb rekonfigurieren. Eine vollständige dynamische Rekonfiguration wird daher zurzeit von Xilinx FPGAs nicht unterstützt [WM05].
- Das System, das die Rekonfigurierung steuert muss bei einer Rekonfiguration entscheiden, welche rekonfigurierbare Logik in welchem rekonfigurierbaren Hardwarebereich geladen wird. Zudem muss dieses System alle nötigen Bitstreams speichern. Dieser Speicher- und Rechenaufwand muss durch die Ressourcen des Systems gedeckt werden. Daher korrelieren die Kosten dieses Systems mit dessen Fähigkeit eine schnelle und flexible Rekonfiguration zu ermöglichen.
- Ein Angreifer kann ungesicherte Konfigurationsports ansteuern, um eigene Module in die rekonfigurierbaren Bereiche einzuschleusen [ZG05].

3 Zeitverhalten und Verzögerungsfehler

In diesem Kapitel wird das Zeitverhalten von Entwürfen, die durch FPGAs implementiert werden, genauer betrachtet. Zunächst werden die dafür nötigen Grundlagen in Abschnitt 3.1 erklärt. In Abschnitt 3.2 wird erklärt, wie Verzögerungsfehler auftreten können, was die Ursachen für solche Fehler sind und wie diese Fehler toleriert werden können. Anschließend wird in Abschnitt 3.3 erklärt, wie das Zeitverhalten von Entwürfen analysiert werden kann, um Verzögerungsfehler zu erkennen. Wie das Verzögerungsverhalten von Komponenten gemessen werden kann, wird in Abschnitt 3.4 dargestellt.

3.1 Grundlagen

In diesem Abschnitt wird zunächst eine graphen-basierte Beschreibung einer Schaltung erklärt. Anschließend werden die Begriffe Defekt, Fehler sowie Fehlermodell erklärt. Abschließend werden Pfade, deren Sensiblisierbarkeit und die Zeitanforderungen erläutert, die an eine Schaltung gestellt werden. Zuletzt wird dargestellt, wie die Verzögerung eines Signals modelliert werden kann.

3.1.1 Netzlisten

Eine Schaltung ist eine Strukturbeschreibung eines, meist implementierten, Hardwareentwurfs. Eine Netzliste auf Komponenten-Level beschreibt eine Schaltung und ist ein azyklischer gerichteter Graph $G = (V, E)$ mit Knotenmenge V und Kantenmenge E. Jeder Knoten $v \in V$ bezeichnet dabei eine Komponente der Schaltung. In FPGAs ist das beispielsweise eine LUT, ein DSP Block oder eine Block-RAM-Komponente. Eine gerichtete Kante $e = (v, w) \in E$ definiert dabei eine Verbindung zwischen einem Ausgang einer Komponente $v \in V$ und einem Eingang der Komponente $w \in V$. Pro Eingang einer Komponente kann höchstens eine Kante spezifiziert werden. In FPGAs werden die Verbindungen und damit auch die Kanten während des Place-and-Route Schrittes definiert. Dabei können sie verschiedene programmierbare Schaltmatrizen verwenden. Eine Kante wird in dieser Arbeit Verzögerungskante genannt, wenn sie mit der Verzögerung annotiert wird, die auf eine propagierende Signaltransition wirkt. Viele Zeitverhalten-Analyse-Werkzeuge nutzen genauere Netzlisten auf Komponentenpin-Level: Bei diesen Netzlisten ist die Knotenmenge durch die einzelnen Eingangs- und Ausgangspins der Komponenten definiert.

Knoten können anhand der Anzahl ihrer direkten Vorgänger in Logikebenen gruppiert werden.

$$\text{Logikebene}(v \in V) = \begin{cases} 0, \text{falls } \nexists e = (x, v) \in E, x \in V \\ max\{\text{Logikebene}(x)\} + 1, \forall e = (x, v), x \in V, \text{sonst} \end{cases}$$

Knoten ohne direkten Vorgänger werden primäre Eingänge genannt. Hat ein Knoten keinen direkten Nachfolger, so wird er primärer Ausgang genannt.

3.1.2 Defekte, Fehler und Fehlermodelle

Ein Defekt ist eine ungewollte Abweichung der physikalischen Struktur eines Schaltkreises durch beispielsweise zusätzliches, fehlendes oder verunreinigtes Material. Aus Defekten kann ein fehlerhaftes elektrisches Verhalten des Schaltkreises resultieren. Diese Fehler werden auf einer höheren Ebene modelliert, um eine effiziente algorithmische Bearbeitung zu ermöglichen [Wun13]. Zur Klärung der Begriffe wird ein Beispiel betrachtet:

- Defekt - an einer Stelle einer Verbindungsleitung befindet sich überschüssiges Material, das zu einem Kurzschluss mit Masse führt.
- Haftfehlermodell - dieser Defekt kann als Stuck-at-1 modelliert werden. Hierbei stecken die Komponenteneingänge, die durch die Verbindungsleitung angesteuert werden, auf dem Wert 1 fest.

Weder ein Defekt noch ein Fehler muss zwangsläufig zu einem Ausfall des betreffenden Systems führen.

3.1.3 Pfade, Sensibilisierbarkeit und Zeitverhalten-Anforderungen

Ein Pfad P ist eine Liste von Knoten (Komponenten) der Netzliste $G = (V, E)$, die über Kanten (Verbindungsleitungen) miteinander verbunden sind.

$$P = (v_0, v_1, \ldots, v_i), v_j \in V \text{ mit } e_k = (v_{k-1}, v_k) \in E$$

Ein Pfad P ist sensibilisierbar, wenn es eine Belegung der Eingangssignale der Schaltung gibt, sodass ein Signalwechsel an einem Eingang von $v_0 \in P$ zu einem Signalwechsel am Ausgang von $v_{0 \leq j \leq i} \in P$ führen kann. Die Länge eines Pfades P ist durch die Gesamtverzögerung, die auf ein Signal wirkt, wenn es durch den Pfad propagiert, definiert.

Eine Verletzung der Anforderung an das Zeitverhalten einer Schaltung liegt vor, wenn ein Signal nicht stabil anliegt bevor dessen Wert abgetastet bzw. benötigt wird. Der Slack ist als Differenz zwischen dem Stabilierungszeitpunkt und dem Abtastzeitpunkt definiert [HSC+82]. In sequentiellen Schaltungen wird der Abtastzeitpunkt durch die Taktperiode vorgegeben. Typischerweise befindet sich in sequentielle Schaltungen eine kombinatorische Schaltung zwischen zwei Speicherelementen, die ihr Eingangssignal nur während einer steigenden Taktflanke übernehmen bzw. speichern. Das Eingangssignal der kombinatorischen Schaltung kann sich also nur während einer steigenden Taktflanke ändern und muss bis zur nächsten steigenden Flanke einen stabilen bzw. finalen Wert annehmen. Die meisten Speicherelemente stellen zusätzliche Bedingungen an das Eingangssignal. Beispielsweise muss das Eingangssignal bei Flipflops einige Zeit vor und nach der Takflanke stabil anliegen, andernfalls kann das Flipflop in einen undefinierten Zustand gebracht werden. Wird eine kombinatorische Schaltung in einen rekonfigurierbaren Hardwarebereich geladen, ist oft dennoch die Taktperiode für den Abtastzeitpunk maßgebend, da oft Flipflops an den Ein- und Ausgängen der Schaltung platziert sind.

Um die geringste Taktperiode (und damit die höchste Taktfrequenz) zu ermitteln, muss also der längste sensibilisierbare Pfad des Schaltungsentwurfs bestimmt werden. Wenn der Pfad nicht sensibilisierbar

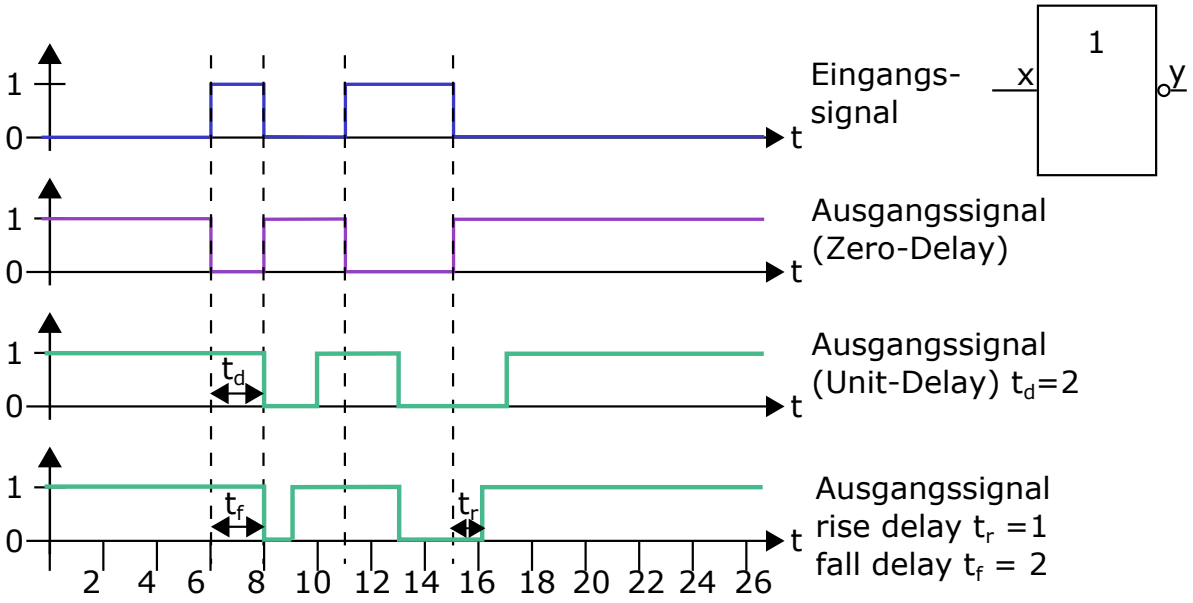


Abbildung 3.1: Ausschnitt der typischen Struktur eines modernen FPGAs (untere linke Ecke des zweidimensionalen Feldes)

ist, kann er die Zeitverhalten-Anforderungen der Schaltung nicht verletzen, da sich der Wert des abzutastenden Ausgangssignals nicht entlang dieses Pfades ändern kann. Der längste sensibilisierbare Pfad wird auch kritischer Pfad genannt.

Die Anzahl der Pfade in einer Schaltung kann exponentiell zu der Anzahl an Komponenten sein. Beispielsweise ist das bei manchen Multiplizierer-Implementierungen der Fall sein. Bei diesen Implementierungen sind die Komponenten einer Logikebene jeweils mit zwei Komponenten der nächsten Ebene verbunden. Dadurch kann eine Komponente der Logikebene n durch $\mathcal{O}(2^n)$ Pfade erreicht werden. Insgesamt besitzt der 16x16 Bit Multiplizierer des ISCAS'85 Benchmarks [Bry85] ca. $9.89 \cdot 10^{19}$ Pfade.

3.1.4 Verzögerungsmodelle

Wenn ein neues Signal an den Eingang einer Hardwarekomponenten angelegt wird, dauert es einige Zeit, bis sich diese Änderung am Ausgang der Komponente bemerkbar macht. Ebenso passiert ein Signal eine Verbindungsleitung nicht sofort, sondern breitet sich mit 0.6-facher Lichtgeschwindigkeit darin aus [Hof07]. Eine Signaltransition wird also verzögert während es durch Komponenten oder Verbindungsleitungen propagiert. Um diesen Effekt zu modellieren, existieren Verzögerungsmodelle, die auch während der Implementierung eines Entwurfs genutzt werden, um das Verhalten zu analysieren. Die für diese Arbeit wichtigsten Modelle werden kurz anhand Abbildung 3.1 vorgestellt. Diese Abbildung zeigt den Verlauf des Eingangs- bzw. Ausgangssignals eines Inverters, wenn unterschiedliche Verzögerungsmodelle angenommen werden.

Zero-Delay

Bei dem zero-delay Verzögerungsmodell existiert keine Verzögerung eines Signals, wenn es vom Eingang einer Komponente zum Ausgang propagiert. Eine Signaltransition propagiert unverzögert durch die Schaltung. Dieses Verzögerungsmodell ist hilfreich, wenn beispielsweise in einer Simulation nur das funktionale Verhalten betrachtet werden soll. Das zeitliche Verhalten eines Entwurfs kann dadurch nicht simuliert oder validiert werden.

Unit-Delay

Für jeden Komponentenausgang und Komponententyp wird beim unit-delay Verzögerungsmodell ein nomineller Verzögerungswert angenommen [BGHK96]. Um den nominellen Verzögerungswert einer Komponente zu bestimmen, wird das Worst-Case-Zeitverhalten dieser gemessen. Es wird nicht zwischen Signalverläufen unterschieden. Dies ist ein vereinfachtes Modell, das verglichen mit genaueren Modellen einen niedrigeren Rechenbedarf besitzt. Der vorgeschlagene Algorithmus arbeitet momentan mit dem unit-delay Modell.

Rise/Fall-Delay

Das rise/fall-delay Verzögerungsmodell ist genauer als das unit-delay Modell, da es zwischen steigenden und fallenden Signalverläufen unterscheidet und für diese jeweils ein Verzögerungswert pro Komponententyp angenommen wird. Pro Signalverlauf, Komponententyp und Komponentenausgang wird bei diesem Modell ein Verzögerungswert angegeben.

3.2 Verzögerungsfehler

Wenn eine Leitung von einem Verzögerungsfehler betroffen ist, werden Signaltransitionen, die durch die Leitung propagieren, bei der stabilen Annahme des korrekten Wertes verzögert. Dadurch ist es möglich, dass das Ergebnis einer Berechnung zu früh abgegriffen und damit falsch sein kann. Zudem kann es geschehen, dass Speicherelemente falsch angesteuert werden und in einen undefinierten Zustand gebracht werden, oder dass es zu Störimpulsen kommt.

Zu diesen Fehler kann es im Entwurf kommen, wenn für Verbindungsleitungen und Komponenten, die einen Entwurf implementieren, ein falsches Zeitverhalten bzw. eine falsche Verzögerung angenommen werden. Die Ursachen von Verzögerungsfehlern sind (neben einer fehlerhaften Implementierung) Variationen des Zeitverhaltens von Komponenten. Diese bewirken, dass das angenommene Verzögerungsmodell mit den tatsächlichen Verzögerungen der Komponenten nicht (mehr) übereinstimmt. Unterabschnitt 3.2.2 beschäftigt sich mit den Ursachen der Verzögerungsfehler. Abschließend wird in Unterabschnitt 3.2.3 erklärt, wie Verzögerungsfehler durch diversifizierte Entwurfskonfigurationen toleriert werden können.

3.2.1 Modelle

Verzögerungsfehlermodelle spezifizieren, wie sich Verzögerungsfehler auf das logische Verhalten eines Hardwareentwurfs auswirken können. Es wurden einige Modelle vorgeschlagen [KC98, MA98]. Diese nehmen an, dass die Komponenten eines Entwurfs einen steigenden und einen fallenden Signalverlauf unterschiedlich verzögern, wenn sie vom Eingangs- zum Ausgangspin einer Komponenten propagieren. Ebenso wird angenommen, dass Verbindungsleitungen einen steigenden und fallenden Signalverlauf unterschiedlich stark verzögern [MA98]. Die drei klassischen Modelle: das Transitionsfehler-Modell [WLRI87], das Gatterverzögerungs-Modell [CIR87] und das Pfadverzögerungfehler-Modell [Smi85] werden erläutert. Das Transitionsfehler- und Gatterverzögerungs-Modell nehmen an, dass Defekte an Gattern bzw. Komponenten auftreten und diese verzögern. Das Pfadverzögerungsfehler-Modell geht von Defekten aus, die mehrere Komponenten betreffen und somit zu einem verteilten Verzögerungsfehler führen können, der mehrere Komponenten betrifft. Obwohl die Modelle überwiegend Gatter behandeln, können sie auch auf FPGA-Komponenten wie LUTs angewandt werden.

Transitionsfehler

Beim Transitionsfehler-Modell wird angenommen, dass ein Verzögerungsfehler nur ein Gatter betrifft. Jedes Gatter kann dabei bei einer Transition des Ausgangssignals zu einer logischen Eins (slot-to-rise fault) oder zu einer logischen Null (slow-to-fall fault) verzögert werden, falls das Gatter von einem Verzögerungsfehler betroffen ist. Hierbei wird angenommen, dass ein Verzögerungsfehler die Transition so stark verzögert, dass sie nicht rechtzeitig stabil anliegt und somit falsch abgetastet wird. Das Modell unterscheidet also nicht zwischen kurzen und langen Pfaden des Entwurfs, sondern nimmt bei jedem Verzögerungsfehler eine Verletzung der Zeitverhalten Anforderungen der Schaltung an. Allerdings ist die Annahme einer einzelnen enormen Verzögerung pro Gatter, die in jedem Fall zu einer Verletzung der Zeitverhalten Anforderung führt, ein Nachteil. Auf kurzen Pfaden können größere Verzögerungen oft toleriert werden. Außerdem kann die Annahme, dass ein Verzögerungsfehler nur ein Gatter betrifft unrealistisch sein, da ein Verzögerungsfehler mehrere Gatter betreffen kann und die Summe der zusätzlichen Verzögerung dieser schließlich zu einer Verletzung der Zeitverhaltens Anforderungen führen kann [WWW06].

Gatterverzögerungsfehler

Beim Gatterverzögerungsfehler-Modell wird ebenso wie beim Transitionsfehler-Modell angenommen, dass ein Verzögerungsfehler nur ein Gatter betrifft. Allerdings bezieht dieses Modell die Verzögerungen der Komponenten mit ein und unterscheidet zwischen langen und kurzen Pfaden. In diesem Modell können nur hinreichend große Verzögerungsfehler, die Gatter auf Pfaden mit einem geringen Slack betreffen, den Schaltkreis stören.

Pfadverzögerungsfehler

Das Pfadverzögerungsfehler-Modell nimmt an, dass ein Schaltkreis nur dann ein fehlerhaftes Verhalten aufweist, wenn die Verzögerung entlang der Pfade des Schaltkreise ein bestimmtes Limit überschreiten. Dieses Limit wird in sequentiellen Schaltungen durch die Taktperiode gegeben. Es ist oft nicht möglich die Taktperiode erheblich zu verlängern, da ein Entwurf ein Ergebnis innerhalb eines bestimmten Zeitfensters berechnen können muss. In diesem Modell verteilt ein Verzögerungsfehler sich über die Schaltung, indem es an mehreren Gattern zu kleinen Verzögerungen eines Signals führt. Ein Pfadverzögerungsfehler (PDF) tritt auf, wenn ein Signal durch einen Pfad propagiert, und die Gesamtverzögerung des Signals durch diesen Pfad das gegebene Limit überschreitet. Der Vorteil dieses Modells ist, dass auch Defekte, die zwar zu kleinen Verzögerungsfehlern an Gattern führen, aber zusammenaddiert ein Signal erheblich verzögern, betrachtet und analysiert werden können. Die Anzahl der Pfade einer Schaltung kann jedoch exponentiell sein, siehe Unterabschnitt 3.1.3, wodurch es oft nicht möglich ist alle Pfade auf Pfadverzögerungsfehler zu überprüfen. Stattdessen existieren einige Strategien, um eine Stichprobe der zu überprüfenden Pfade einer Schaltung zu bestimmen. Z.B. können alle Pfade, deren Gesamtverzögerung (Pfadverzögerung) oberhalb einer gewissen Untergrenze liegt, betrachtet werden. In dieser Arbeit werden unterschiedliche Möglichkeiten zur Wahl der Stichprobe aufgezeigt. Unter anderem kann die Stichprobe aus den l längsten Pfaden pro LUT Komponente bestehen.

3.2.2 Ursachen

Verzögerungfehler können durch Alterung, zufälligen Variationen oder Defekten hervorgerufen werden. Durch diese Effekte kann die Verzögerung einer Komponente verändert werden, wodurch das reale Zeitverhalten vom erwarteten Zeitverhalten abweicht und es zu Verzögerungsfehlern kommt. Moderne FPGAs wie das Xilinx Virtex-7 FPGA werden mittels 28nm-Fertigung hergestellt. Mit zunehmender Miniaturisierung der Schaltkreise wird auch die Mindestgröße einer Variation bzw. eines Defekts, die nötig ist, um einen fehlerhaftes Verhalten zu verursachen, kleiner [McP06]. Variationen und Defekte können z.B. durch einen imperfekten Herstellungsprozess oder kleinste verunreinigende Partikeln in den Schaltkreis eingeschleust werden.

Die Stärke der Degradierung durch Alterung hängt von Stressfaktoren, wie Temperatur, Schaltaktivität oder elektrische Feldstärke des Schaltkreises ab [SKM+08]. Im Folgenden werden Mechanismen dargestellt die durch Defekte, Variationen oder Alterung auftreten können und u.a. zu Verzögerungsfehlern führen können.

Time-dependent Dielectric Breakdown

Beim Time-dependent Dielectric breakdown (TDDB) degradiert das dielektrische Material allmählich, das sich unter dem Gate Anschluss eines Transistors befindet, bis es leitfähig ist. Dadurch kommt es zu einem Kurzschluss im Transistor. Der TDDB wird als einer der Hauptgründe für permanente Ausfälle von modernen Schaltkreisen gesehen [SKM+08]. Dieser Mechanismus wird stark von der Temperatur und dem Gate-Leckstrom des Transistors beeinflusst und kann als Transitionsfehler modelliert werden.

Elektromigration

Elektromigration (EM) ist einer der Gründe für permanente Ausfälle von Verbindungsleitungen. Bei der EM wird Material (z.B. Metall einer Verbindungsleitung) allmählich durch die Bewegung von Elektronen in einem Leiter abgetragen. Dadurch können Hohlräume in einem Leiter bzw. einer Verbindungsleitung entstehen, die je nach Größe zu unterschiedlich stark ausgeprägten Verzögerungsfehlern führen können. Die erhöhte Stromdichte und verringerte Komponentengröße in modernen 28nm-FPGAs begünstigen den EM Effekt.

Hot-Carrier Effects

Der Hot-Carrier Effekt (HCE) bewirkt, dass Elektronen allmählich in ungewollte Materialien bzw. Regionen injiziert werden. Dadurch können Elektronen in das Dielektrikum eines Transistors eingeschlossen werden, wodurch sich Charakteristiken des Transistors ändern. Eine der ersten Folgen des HCEs ist eine Veränderung der Schwellenspannung des Transistors. Die Stärke des HCEs in FPGAs ist von der Schaltaktivität der im FPGA vorhandenen Transistoren, die sich u.a. in LUTs und konfigurierbaren Schnittpunkten befinden, abhängig. Je nach Stärke des HCE wird eine Transition verhindert oder verzögert.

Negative Bias Temperature Instability

Durch den Negative Bias Temperature Instability (NBTI) Effekt wird die Schwellenspannung eines Transistors erhöht [Mas04]. NBTI wird durch geringe negative Spannungen und hohe Temperaturen begünstigt. Dieser Effekt kann das Schalten eines Transistor verhindern oder verzögern und damit zu Verzögerungsfehlern führen.

3.2.3 Fehlertoleranz durch diversifizierte Entwurfskonfigurationen

Permanente Ausfälle und Verzögerungsfehler von Ressourcen können in Rekonfigurierbaren Systemen durch eine veränderte Nutzung der Ressource toleriert werden [ZBK+13]. Beispielsweise kann ein FPGA rekonfiguriert werden, nachdem ein Fehler erkannt und lokalisiert wurde. Die neue Konfiguration des FPGAs kann die Nutzung fehlerhafter Komponenten vermeiden, indem stattdessen funktionstüchtige Komponenten für die Implementierung eines Entwurfs verwendet werden. Die Rekonfiguration kann dabei dynamisch oder statisch geschehen.

Die Autoren von [ZBK+13] stellen ein Verfahren vor, durch das Fehler in Entwürfen toleriert werden können, die in rekonfigurierbare Hardwarebereiche konfiguriert werden. Dafür wird zunächst eine Menge K an Konfigurationen für jeden Entwurf der Menge E erzeugt. Die Konfigurationen eines Entwurfs berechnen die selbe Funktion und nutzen dabei andere Komponenten eines Hardwarebereichs, sodass:

$$\forall \text{CLB} \in \text{Bereich} : \forall e \in E \; \exists k \in K \; \text{CLB} \notin k$$

Für jeden CLB eines rekonfigurierbaren Hardwarebereichs existiert, für alle Entwürfe, mindestens eine Konfiguration des Entwurfs, die den CLB nicht benötigt. Wenn ein beliebiger CLB eines Hardwarebereichs Fehler verursacht, existiert nun mindestens eine Konfiguration pro Entwurf, die den CLB nicht benötigt. Diese kann somit ohne Fehler zu produzieren in den rekonfigurierbaren Hardwarebereich konfiguriert und in Betrieb genommen werden. Zusätzlich wird die Anzahl der Konfigurationen minimiert, indem maximal diversifizierte Konfigurationen generiert werden, die sich in möglichst vielen CLB-Nutzungen voneinander unterscheiden. In dieser Arbeit wird dieser Ansatz weiterverwendet und untersucht, inwiefern die Nutzung von diversifizierten Entwurfskonfigurationen und mehrerer rekonfigurierbarer Hardwarebereiche, Variationen des Zeitverhaltens adaptieren können.

3.3 Zeitverhalten-Analyse

Eine Zeitverhalten-Analyse wird durchgeführt, um das Zeitverhalten eines Entwurfs zu verifizieren. Dabei werden u.a. benutzerdefinierte Zeitverhalten-Nebenbedingungen geprüft und z.B. untersucht, ob der Entwurf unter der gewählten Taktperiode operieren kann ohne, dass es zu Verzögerungsfehlern kommt. Im Folgenden werden der dynamische und statische Ansatz zur Zeitverhalten-Analyse vorgestellt.

3.3.1 Statisch

Die statische Zeitverhalten-Analyse (STA) kann an unterschiedlichen Stellen des Synthese- und Implementierungsvorgangs durchgeführt werden. Beispielsweise kann die STA vor dem Logikoptimierung-Schritt der Synthese ausgeführt werden, um kritische Pfade zu identifizieren, die während der Logikoptimierung berücksichtigt und eventuell verkürzt werden können [BC09].

Bei der STA wird die Netzliste eines Entwurfs um Verzögerungskanten erweitert. Zu jedem Komponentenknoten werden Verzögerungskanten hinzugefügt, die jeweils einen Komponenteneingang mit einem Komponentenausgang verknüpfen, sodass alle Eingänge über Verzögerungskanten mit den Ausgängen verknüpft sind. Falls die Netzliste auf Komponentenpin-Level vorliegt, müssen keine weiteren Kanten hinzugefügt werden. Zudem wird für jede Kante die Verzögerung als Kosten eingetragen, die auf ein Signal wirkt, wenn es durch die Verbindung propagiert, die durch die Kante modelliert wird. Die eingetragene Verzögerung entspricht einem Worst Case Unit-delay Verzögerungsmodell der tatsächlichen Komponentenverzögerungen. Um das Zeitverhalten eines Entwurfes zu analysieren, muss der erstellte Graph traversiert und die Kosten der dabei genutzten Kanten betrachtet werden. Wenn die geringste Taktperiode bestimmt werden soll, muss der längste Pfad, der in einem Flipflop-Eingang endet und dabei keine anderen Flipflop-Eingänge durchquert, gefunden werden. Dieser längste Pfad kann z.B. durch eine Tiefensuche bestimmt werden. Bei der STA werden weder die Belegung der Eingangssignale noch die Sensibilisierbarkeit der Pfade berücksichtigt. Das Zeitverhalten-Analyse Werkzeug, das in dieser Arbeit verwendet wird (TRACE) führt eine STA durch und verwendet dabei ein abgeändertes Unit-Delay Modell, das mehrere Verzögerungen pro Komponenteneingang und -ausgang erlaubt. Die STA nimmt das Worst Case Zeitverhalten von Komponenten an, wodurch zwar Verzögerungsfehler vermieden werden, aber die Performanz einer Schaltung nicht maximiert wird. Das Verfahren ist stark von dem angenommen Verzögerungsmodell abhängig. Ist die tatsächliche

Verzögerung einer Komponente, durch Alterung oder ähnlicher Effekte, größer als durch das Verzögerungsmodell angenommen, kann es zu Verzögerungsfehlern kommen. Hierbei setzt diese Arbeit an und bietet eine Schnittstelle für aktuelle Verzögerungswerte an.

3.3.2 Dynamisch

Bei der dynamischen Zeitverhalten-Analyse (DTA) wird das Zeitverhalten eines Entwurfs durch Simulation verifiziert [BC09]. Testvektoren werden an den Eingängen der Schaltung angelegt. Anschließend wird die Schaltung simuliert und nach einer spezifizierten Simulationsdauer werden die Ausgänge abgetastet. Ist der Wert der Ausgänge zum Abtastzeitpunkt ein anderer als durch die Beschreibung des Entwurfs vorgegeben, existieren Verzögerungsfehler in der Schaltung. Allerdings hängt das Entdecken solcher Verzögerungsfehler durch dieses Verfahren von den angelegten Testvektoren ab. Abhängig von den Testvektoren schalten Signal-Transitionen entlang mehreren sensibilisierten Pfaden. Wenn eine Transition entlang eines kritischen Pfades propagiert, wird dieser durch den Testvektor abgedeckt und das Zeitverhalten des Pfades wird verifiziert. Verschiedene Taktperioden können nun auf Verzögerungsfehler untersucht werden, indem die Simulationsdauer angepasst wird. Je kürzer die Simulationsdauer ist, desto weniger Zeit steht für die Propagierung der Transitionen zu den Schaltungsausgängen zur Verfügung, die durch die Testvektoren stimuliert wurden. Das Zeitverhalten eines komplexen Entwurfs mit 10-100 Millionen Gattern mittels DTA zu überprüfen ist sehr langsam, wodurch es oft nicht möglich ist das Zeitverhalten vollständig zu überprüfen und alle Pfade durch Testvektoren abzudecken [BC09].

3.4 Charakterisierung der Verzögerung

Bei der Charakterisierung der Verzögerung wird das Zeitverhalten von Komponenten oder Pfaden gemessen. Die gemessenen Verzögerungswerte können z.B. in Zeitverhalten-Analysen einfließen. Die Verzögerungen einzelner FPGA Komponententypen werden durch deren Hersteller gemessenen und in den Verzögerungsmodellen der STA Werkzeuge verwendet. Allerdings kann sich das Verzögerungsverhalten von FPGA Komponenten durch deren Alterung ändern und von den gemessenen nominellen Verzögerungen abweichen. Um einen aktuellen Verzögerungswert in der STA verwenden zu können, muss die Verzögerung der FPGA Komponenten regelmäßig charakterisiert werden. In der Literatur werden dafür vorwiegend Ringoszillatoren [LWLL04] und At-Speed Tests [KKKO06] durchgeführt [SC06].

4 Algorithmus zur Adaptierung an Zeitverhalten-Variationen

In diesem Kapitel wird der Ansatz und der Algorithmus zur Adaptierung von Variationen des Zeitverhaltens erklärt. Zunächst wird in Abschnitt 4.1 der Ansatz des Algorithmus dargestellt. Dabei werden die Problemstellung und Annahmen beschrieben. Anschließend wird der vorgeschlagene Algorithmus in Abschnitt 4.3 erklärt. Zuletzt werden in Abschnitt 4.4 und Abschnitt 4.5 die Laufzeit und Speicheranforderung des Algorithmus analysiert und aufgezeigt, welche Möglichkeiten es gibt diese an die vorhandenen Ressourcen anzupassen.

4.1 Ansatz

Der Ansatz adaptiert Variationen des Zeitverhaltens, um falsche Ergebnisse durch Verzögerungsfehler zu vermeiden und soll dabei Ressourcenvorgaben einhalten. Um das Auftreten von Verzögerungsfehler zu minimieren, nutzt der Ansatz dabei die Rekonfigurationsfähigkeit rekonfigurierbarer Hardwarestrukture. Bei der Entscheidung, welcher Hardwareentwurf in welchem rekonfigurierbaren Hardwarebereich konfiguriert werden soll, wird das Zeitverhalten der Hardwarebereiche analysiert und nur die Hardwareentwürfe darin rekonfiguriert, deren Zeitverhaltenvorgaben durch die genutzten Komponenten des Hardwarebereichs nicht verletzt werden. Abbildung 4.1 zeigt das Vorgehen bei mehreren Konfigurationen eines Hardwareentwurfs. Eine Konfiguration eines Hardwareentwurfs besitzt die gleiche Funktionalität wie der Hardwareentwurf. Allerdings können von jeder Konfiguration unterschiedliche Komponenten genutzt werden, um die gleiche Funktion abzubilden. Bei Nutzung mehrerer Konfigurationen pro Entwurf hat dies den Vorteil, dass stark gealterte Komponenten eines Hardwarebereichs nur von einer Teilmenge der Konfigurationen genutzt werden [ZBK+13]. Durch die Bestimmung von geeigneten Konfigurationen für einen Hardwarebereich ist es möglich, gealterte Komponenten bei einer Rekonfiguration auszulassen und Entwürfe dennoch zu platzieren.

Ist keine Konfiguration eines Hardwareentwurfs mehr konfigurierbar, ohne dass durch die Nutzung von gealterten Komponenten Verzögerungsfehler auftreten, so kann der Nutzer benachrichtigt werden. Andernfalls können die rekonfigurierbaren Hardwarebereiche mit den vorgeschlagenen Konfigurationen rekonfiguriert werden, ohne dass gealterte Komponenten genutzt werden oder Verzögerungsfehler auftreten.

Sei h_i der $i$te Hardwareentwurf, $C_{h_i}^j$ die $j$te Konfiguration des $i$ten Hardwareentwurfs und B die Menge der rekonfigurierbaren Hardwarebereiche. Ferner sei das Tupel $(C_{h_i}^j, b)$ definiert als verzö-

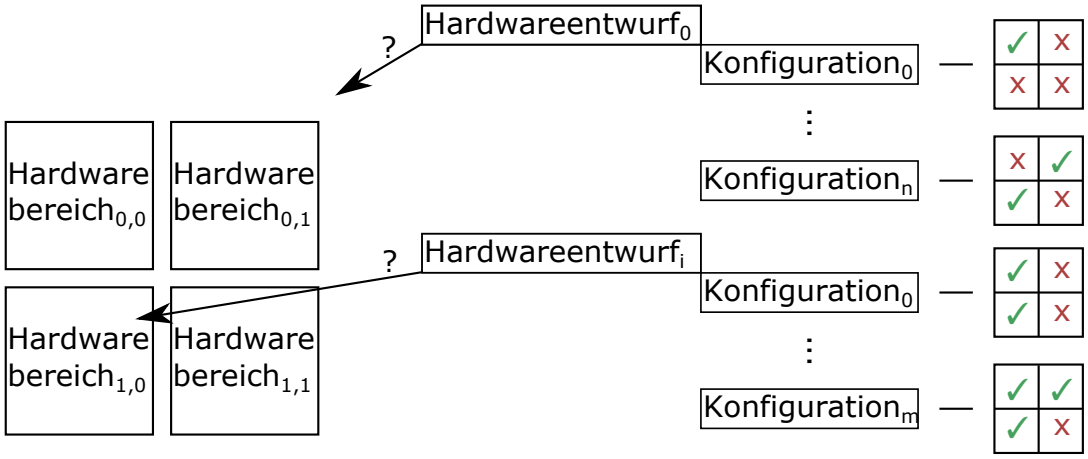


Abbildung 4.1: Ausschnitt der typischen Struktur eines modernen FPGAs (untere linke Ecke des zweidimensionalen Feldes)

gerungsfehlerfreie Platzierung von $C^j_{h_i}$ in $b \in B$. Dann lässt sich das Ziel des Ansatzes wie folgt definieren:

maximiere	$\|\{(C^j_{h_i}, b)\}\|$
sodass	Speicheraufwand $\leq$ Speichervorgabe,
	Laufzeit $\leq$ Laufzeitvorgabe.

Die vom Algorithmus berechneten Konfigurationsvorschläge werden anschließend von einem Platzierungsalgorithmus analysiert, der entscheidet, welche Konfiguration in welchem rekonfigurierbaren Hardwarebereich mittels statischer oder dynamischer Rekonfiguration konfiguriert wird. In diese Entscheidung können viele Faktoren, wie z.B. das erwartete Anwendungsprofil des Systems oder die geforderte Energieeffizienz einfließen. Die Entwicklung eines Platzierungsalgorithmus liegt nicht im Rahmen dieser Arbeit. Allerdings wurde zu Testzwecken ein einfacher Platzierungsalgorithmus entworfen, der n Entwürfe über m Konfigurationen in n rekonfigurierbare Hardwarebereiche platziert (falls möglich).

Um festzustellen, ob der Einsatz einer Konfiguration in einem rekonfigurierbaren Hardwarebereich zu Verzögerungsfehlern führt, muss das Zeitverhalten der Konfiguration und die Zeitverhalten-Variationen des rekonfigurierbaren Hardwarebereichs analysiert werden. Wenn ein rekonfigurierbarer Hardwarebereich mit einer Konfiguration konfiguriert wird, verwendet die Konfiguration gewisse Komponenten des Hardwarebereichs, um die Funktion des zugeordneten Hardwareentwurfs zu implementieren. Weicht das Zeitverhalten dieser Komponenten, beispielsweise aufgrund von Alterung, von dem Zeitverhalten ab, das bei der Synthese der Konfiguration erwartet wurde, so kann es zu Verzögerungsfehlern kommen. Diese äußern sich unter anderem in Störimpulsen oder undefiniertem Verhalten der Speicherelemente des rekonfigurierbaren Hardwarebereichs. Abschnitt 3 befasst sich genauer mit Verzögerungsfehler und der Analyse des Zeitverhaltens.

4.1.1 Voraussetzungen

Der Ansatz wird in dieser Arbeit auf FPGAs der Firma Xilinx angewandt. Allerdings lässt sich der Ansatz auch auf andere Architekturen anwenden, falls folgende Punkte von ihnen erfüllt werden:

- Die Architektur besitzt eine Hardwarestruktur, die sich statisch oder dynamisch rekonfigurieren lässt
- Mindestens eine Konfiguration jedes Hardwareentwurfs lässt sich für die rekonfigurierbare Hardwarestruktur synthetisieren. Wenn mehrere Konfigurationen eines Hardwareentwurfs für die Hardwarestruktur synthetisierbar sind, erhöht sich die Effektivität des Ansatzes.
- Das Zeitverhalten der Hardwareentwürfe und deren Konfigurationen kann analysiert und ausgegeben werden.
- Die Zeitverhalten-Variationen der rekonfigurierbaren Hardwarestrukturen sind messbar. Die dabei gewonnen Daten können vom vorgeschlagenen Algorithmus geladen werden.
- Der vorgeschlagene Algorithmus kann die Platzierung von Hardwareentwürfen bzw. deren Konfigurationen durch Mitteilung von Konfigurationsvorschlägen beeinflussen.

4.2 Entitäten

Im Folgenden werden die vom vorgeschlagenen Algorithmus und vom Ansatz genutzten Entitäten und deren Relationen zueinander erklärt bzw. definiert. Das UML-Klassendiagramm in Abbildung 4.2 gibt einen Überblick über die Entitäten und Relationen. Wie im vorherigen Abschnitt gezeigt, implementieren Konfigurationen einen Hardwareentwurf. Die Konfigurationen werden in die rekonfigurierbaren Hardwarebereiche der rekonfigurierbaren Struktur konfiguriert. Dabei besteht die rekonfigurierbare Hardwarestruktur aus Komponenten, die durch die Definition der rekonfigurierbaren Hardwarebereiche diesen zugeordnet werden. Jede Komponente hat mehrere Pins und zwischen je zwei Pins können mehrere Kanten existieren, falls diese Pins eine direkte leitende Verbindung zueinander haben. Eine Verzögerungskante bezeichnet, wie stark eine Signaltransition verzögert wird, wenn es vom Anfangspunkt der Kante zum Endpunkt geleitet wird. Beispielsweise existiert keine Kante zwischen zwei Eingangspins einer Komponente. Es kann jedoch je nach Zusammensetzung einer Komponenten mehrere Kanten zwischen einem Eingangs- und Ausgangspin einer Komponente geben (siehe auch Abschnitt 3.1). Eine Konfiguration eines Hardwareentwurfs definiert die leitenden Verbindungen zwischen den Komponenten des rekonfigurierbaren Hardwarebereichs, in das sie konfiguriert wird. Dadurch werden implizit auch alle Pfade einer Konfiguration definiert. Pfade besitzen eine Pfadverzögerung, die sich aus den Verzögerungen der Kanten zusammensetzt. Wird ein Signal, wenn es durch die Kanten eines Pfades geleitet wird, stärker verzögert als durch die Pfadverzögerung des Pfades angenommen, so kann es zu Verzögerungsfehlern kommen. Kanten, die den selben Pfaden angehören, werden zusammen mit ihren Pfaden in partielle Segmente zusammengefasst. Die grauen Kanten in Abbildung 4.5 würden zusammen mit den Pfaden ($Pfad_1$,$Pfad_2$) in einem partiellen Segment gespeichert werden.

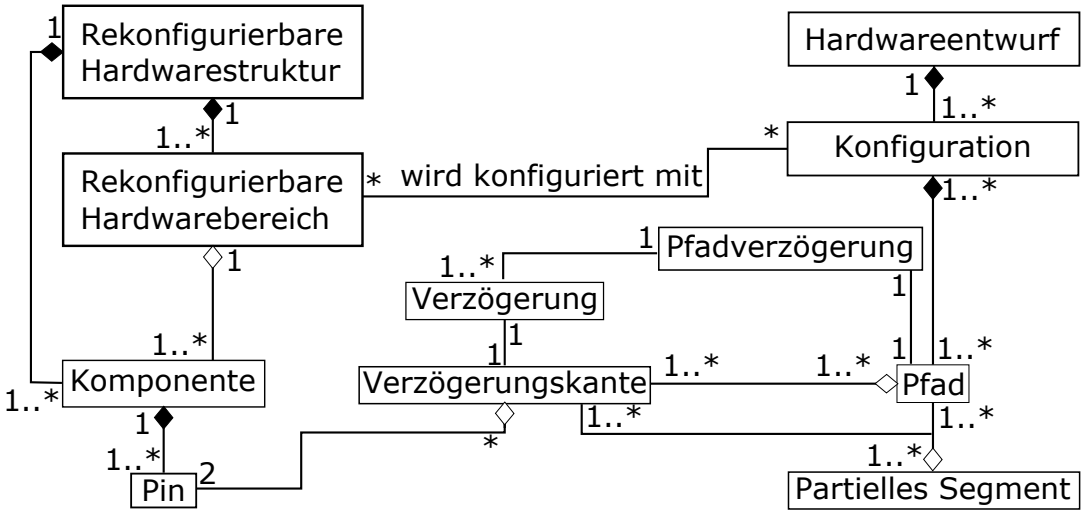


Abbildung 4.2: UML-Klassendiagramm der genutzten Entitäten und deren Relationen zueinander

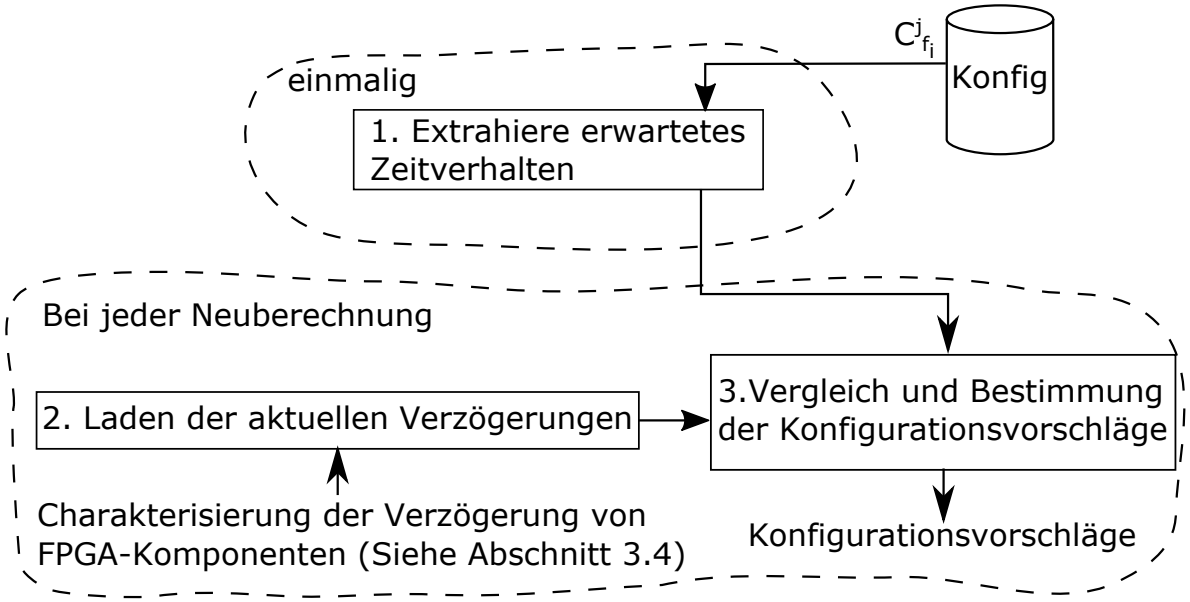


Abbildung 4.3: Schritte zur Bestimmung der Konfigurationsvorschläge

4.3 Ablauf

Im Folgenden wird der Algorithmus, der im Rahmen dieser Arbeit entworfen wurde, beschrieben. Der Algorithmus berechnet die Konfigurationsvorschläge für alle rekonfigurierbaren Hardwarebereiche. Dabei wird dem oben skizzierten Ansatz gefolgt. Abbildung 4.3 zeigt die Schritte, die vom Algorithmus durchgeführt werden, um die Konfigurationsvorschläge zu berechnen. Zunächst wird das erwartete

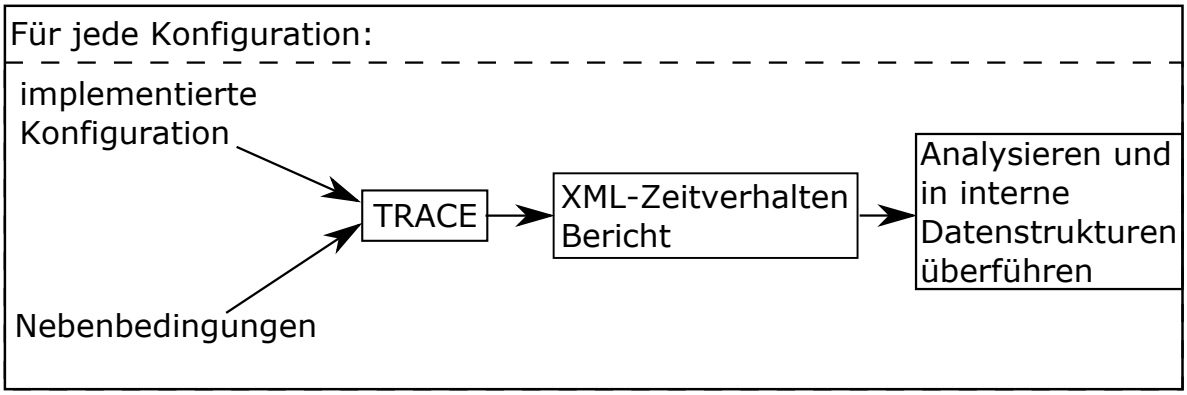


Abbildung 4.4: Vorgehensweise bei der Extraktion des erwarteten Zeitverhaltens

Zeitverhalten jeder Konfiguration analysiert. Dabei wird eine Stichprobe der Pfadinformationen aller Konfigurationen extrahiert und kompakt abgespeichert. Die Details dieses Schrittes werden in 4.3.1 erklärt. Anschließend wird in 4.3.2 gezeigt, wie die Referenzpfadverzögerungen in den Algorithmus eingegeben werden. Die Referenzpfadverzögerungen können dabei durch die Zeitverhalten Charakterisierung des rekonfigurierbaren Hardwarebereichs oder durch ein probabilistisches Modell für die Pfadverzögerungen bestimmt werden. Der letzte Schritt des Algorithmus ist die Bestimmung der Konfigurationsvorschläge. Hierfür wird das Referenzzeitverhalten, das in Schritt 2 eingegeben wurde, mit dem erwarteten Zeitverhalten der Konfigurationen verglichen.

4.3.1 Extraktion des erwarteten Zeitverhaltens

Das erwartete Zeitverhalten einer Konfiguration kann durch die Extraktion der nominellen Pfadverzögerungen festgestellt werden. Die nominellen Pfadverzögerungen werden bei der Synthese einer Konfiguration genutzt, um die Pfadverzögerungen der implementierenden Komponenten zu schätzen. Eine Pfadverzögerung bezeichnet die Verzögerung eines Signals, das von der Anfangs- zur Endkomponenten des Pfades geleitet wird und dabei mehrere Komponenten durchquert. Weichen die realen Pfadverzögerungen der genutzten Komponenten von den nominellen ab, so kann es zu Verzögerungsfehlern kommen. Um die nominelle Pfadverzögerungen zu extrahieren, wird in dieser Arbeit das statische Zeitverhalten Analyse-Werkzeug TRACE genutzt. Abbildung 4.4 zeigt die Vorgehensweise bei der Extraktion des erwarteten Zeitverhaltens. Die vorgestellte Vorgehensweise muss für jede Konfiguration wiederholt werden.

TRACE benötigt als Eingabe eine synthetisierte Konfiguration und Nebenbedingungen, die die Zeitverhalten-Analyse steuern [Xil13a]. Nebenbedingungen können zum Beispiel angeben, dass Pfade, die durch eine bestimmte Komponenten oder Komponentenpin verlaufen, analysiert werden sollen. TRACE berechnet die Verzögerung dieser Pfade, gibt diese in einem Zeitverhalten Bericht aus und bestimmt dabei, ob die Pfadverzögerungen eine benutzerdefinierte Maximalgrenze überschreiten. Aus der synthetisierten Konfiguration gewinnt TRACE eine Strukturbeschreibung, die bei der Zeitverhalten Analyse genutzt wird. Für die Zeitverhalten Analyse werden nominelle Pfadverzögerungen hinzugezogen, um zu überprüfen, ob die Pfade der Konfiguration die Nebenbedingungen einhalten.

Die nominelle Verzögerung eines Pfades wird durch die Art und Anzahl der im Pfad enthaltenen Komponenten und Verbindungsleitungen berechnet.

Nachdem TRACE die Zeitverhalten Analyse abgeschlossen hat, wird ein Bericht ausgegeben. In diesem Bericht sind die durch die Nebenbedingungen spezifizierten, längsten Pfade einer Konfiguration zusammen mit ihrer Verzögerung enthalten. Zusätzlich werden die Pins aller Komponenten eines Pfades zusammen mit der nominellen Verzögerung, die jeweils zwischen zwei Pins besteht, aufgelistet. Dieser Bericht wird vom vorgeschlagenen Algorithmus eingelesen und in interne Datenstrukturen überführt: Pfade werden als Pfad-Objekt angelegt und die Verbindungsstücke zwischen zwei Komponentenpins werden als Kanten-Objekt angelegt und den entsprechenden Pfaden über partielle Segmente zugeordnet. Anschließend können die Datenstrukturen noch modifiziert und unerwünschte Informationen, wie z.B. redundante Kanten oder Pfade entfernt werden.

Über die eingegebenen Nebenbedingung wird kontrolliert, welche Pfade und Kanten einer Konfiguration im Zeitverhalten Bericht enthalten sind. Die Anzahl der Pfade in einer Schaltung kann exponentiell zu der Anzahl der Komponenten sein, daher ist es bereits bei Konfigurationen mit wenigen Komponenten nicht möglich alle Pfade aufzuzählen. Über die Nebenbedingungen ist es möglich eine Stichprobe der Kanten und Pfaden zu definieren, die in einer Konfiguration enthalten sind. Allerdings werden auch nur die Pfade der Stichprobe auf fehlerhaftes Zeitverhalten überprüft. Bei der Gestaltung der Stichprobe muss zwischen Speicheraufwand und Abdeckung abgewägt werden, da bei einer zu kleinen Stichprobe der Algorithmus falsch positive Ergebnisse liefern könnte und bei einer zu großen Stichprobe die Ressourcen des eingebetteten Systems nicht mehr reichen könnten. Um die Abdeckung einer Stichprobe festzustellen, müssen die Kanten der Stichprobe mit den Kanten verglichen werden, die von der Konfiguration genutzt werden. Die Auswahl der Stichprobe muss nicht auf dem eingebetteten System geschehen, daher ist die Laufzeit und der Speicheraufwand für die Auswahl nicht kritisch.

4.3.2 Laden der Referenzpfadverzögerungen

Bevor die Konfigurationsvorschläge berechnet werden können, müssen die Referenzpfadverzögerungen der rekonfigurierbaren Hardwarebereiche bestimmt und geladen werden. Diese werden mit dem erwarteten Zeitverhalten, das im ersten Schritt extrahiert wurde, verglichen, um die Konfigurationsvorschläge der Hardwarebereiche zu bestimmen. Die Referenzpfadverzögerungen können in einem FPGA beispielsweise von einer Zeitverhalten Charakterisierung durch Ringoszillatoren stammen [LWLL04]. Die Charakterisierung stellt den Grad der Alterung von Komponenten fest, indem gemessen wird, wie stark ein Signal verzögert wird, wenn es durch die gemessene Komponente geleitet wird. Details der Zeitverhalten Charakterisierung in FPGAs werden in Abschnitt 3.4 erklärt. Die Referenzpfadverzögerungen müssen für jeden Hardwarebereich $b \in B$ bestimmt werden, da eine Kante durch die Platzierung der dazugehörigen Konfiguration in jedem Hardwarebereich aktiv sein kann. Dabei kann eine Kante in jedem Hardwarebereich ein Signal unterschiedlich stark verzögern, da die Kante unterschiedlich stark gealterte Komponenten nutzen kann. Das Laden der Pfadverzögerungen hängt von der Granularität der Zeitverhalten Charakterisierung ab. Der entworfene Algorithmus unterstützt eine feingranulare Zeitverhalten Charakterisierung, indem für jede Kante die zusätzliche Pfadverzögerungen in jedem rekonfigurierbaren Hardwarebereich als Eingabe erwartet wird. Beispielsweise kann für die Eingabe und Speicherung eine Hashtabelle genutzt werden, die Kanten

auf ein Array abbildet, das für jeden Hardwarebereich die zusätzliche Pfadverzögerung speichert. Ist das Verzögerungsprofil nur grob messbar, z.B. auf CLB-Ebene, so kann der Anstieg der Pfadverzögerung auf jede Kante addiert werden, die sich im CLB befindet. Dadurch lassen sich falsch-positive Konfigurationsvorschläge durch eine zu grobe Zeitverhalten Charakterisierung vermeiden.

4.3.3 Bestimmung der Konfigurationsvorschläge

Das erwartete Zeitverhalten wurde in Schritt 1 extrahiert und kann nun mit dem Referenzzeitverhalten, das in Schritt 2 eingelesen wurde, verglichen werden. Wenn das Referenzzeitverhalten eines Hardwarebereichs nicht mit dem erwarteten Zeitverhalten einer Konfiguration übereinstimmt, kann es zu Verzögerungsfehlern kommen, falls die Konfiguration in den Hardwarebereich konfiguriert wird. Durch die Bestimmung der Konfigurationsvorschläge werden Verzögerungsfehler minimiert, die durch eine unpassende Konfiguration auftreten.

Die Stichprobe des erwarteten Zeitverhaltens liegt als Liste von partiellen Segmenten vor, und das Referenzzeitverhalten einer Kante kann über die Hashtabelle nachgeschlagen werden. Die einzelnen Schritte des Vergleichs werden in Algorithmus 4.1 aufgezeigt. Zunächst wird ein Array angelegt,

Algorithmus 4.1 Bestimmung der Konfigurationsvorschläge

```
for all Partielles_Segment ps in Stichprobe do
    double[] ps_verzögerungen = new double[Anzahl_Hardwarebereiche]
    for all Verzögerungskante e in ps do
        double[] tmp = Verzögerung_Nachschlagen(e)
        ps_verzögerungen += tmp
    end for
    for all Pfad p in ps do
        p.Addiere_Zu_Pfadverzögerung(ps_verzögerungen)
    end for
end for
```

das für jede Konfiguration aller Entwürfe festhält, welcher Hardwarebereich für eine Konfiguration in Frage kommt. Sobald ein Pfad einer Konfiguration die Zeitverhalten Vorgaben in einem Hardwarebereich nicht einhalten kann, da die zum Pfad gehörigen Kanten zu stark verzögern, wird an der entsprechenden Stelle des Arrays ein Flag gesetzt. Anschließend wird über jede Kante des partiellen Segments iteriert und die zusätzliche Referenzpfadverzögerung jedes Hardwarebereichs für die Kante nachgeschlagen. Die Referenzpfadverzögerungen werden für alle Kanten eines partiellen Segments zusammenaddiert und schließlich zu den Pfadverzögerungen des Segments dazugerechnet. Bei der Addition zu den Pfaden wird überprüft, ob die Verzögerung des Pfades höchstens der Verzögerung des kritischen Pfades, multipliziert mit einem Toleranzfaktor, entspricht. Abschließend wird das Array, das die Konfigurationsvorschläge abspeichert an einen Platzierungsalgorithmus übergeben, der entscheidet, welche Konfiguration in welchem Hardwarebereich konfiguriert wird.

4.4 Laufzeit

Die Laufzeit zur Bestimmung der Konfigurationsvorschläge setzt sich aus der Laufzeit, die für das Charakterisieren und Laden des Referenzzeitverhaltens benötigt wird und der Laufzeit des vorgeschlagenen Algorithmus zusammen. Der Algorithmus iteriert über die partiellen Segmente und behandelt jedes dort referenzierte Datenobjekt (Kanten & Pfade) höchstens ein mal. Damit ist die Anzahl ausgeführter Operationen wie folgt:

$$\begin{aligned} \#\text{ Additionen} &= \mathcal{O}((\#\text{ Kanten} + \#\text{ Pfade}) \cdot \#\text{ Hardwarebereiche}) \\ \#\text{ Vergleiche} &= \mathcal{O}(\#\text{ Pfade} \cdot \#\text{ Hardwarebereiche}) \end{aligned}$$

Kanten und # Pfade bezeichnet dabei die Anzahl der Referenzierungen von Kanten- bzw. Pfadobjekten in den partiellen Segmenten. Die Laufzeit des Algorithmus ist also linear zur Größe der Stichprobe. Dadurch kann über die Größe der Stichprobe kontrolliert werden, wie viel Zeit die Bestimmung der Konfigurationsvorschläge beansprucht.

Die genaue Laufzeit des Algorithmus hängt von den Referenzpfadverzögerungen ab. Wenn früh bestimmt werden kann, dass eine Konfiguration nicht in einen Hardwarebereich platziert werden kann, müssen weitere Pfad- & Kantenobjekte der Konfiguration nicht für den Hardwarebereiche evaluiert werden. Vergleichs- und Additionsoperationen können dadurch eingespart werden. Um frühzeitig Konfigurationsvorschläge auszuschließen, können die partiellen Segmente in eine Reihenfolge gebracht werden, die sicherstellt, dass lange Pfade einer Konfiguration früh vom Algorithmus untersucht werden.

Weitere Additions- & Vergleichsoperationen können durch die Abspeicherung fehlgeschlagener Konfigurationsvorschläge eingespart werden. Diese müssen bei einer erneuten Berechnung der Konfigurationsvorschläge nicht betrachtet werden, sodass auch die Pfade und Kanten der betroffenen Konfigurationen nicht mehr für zu stark degradierte Hardwarebereiche evaluiert werden müssen. Allerdings können dadurch Konfigurationen aufgrund kurzzeitiger Fehler bei der Messung des Zeitverhaltens dauerhaft von der Konfiguration in einen Hardwarebereich abgehalten werden.

4.5 Effizientes Abspeichern nominaler Pfadverzögerungen

Die Rekonfiguration des FPGAs wird durch ein eingebettetes System mit begrenzten Ressourcen angesteuert. Der Speicheraufwand des Algorithmus muss also gering sein und sich an die Ressourcen des externen System anpassen lassen. Im Folgenden werden Methoden zur Verminderung des Speicheraufwands bei der Abspeicherung des erwarteten Zeitverhaltens aufgezeigt. Dies geschieht hauptsächlich über die Anpassung der Stichprobe. Zunächst wird erklärt, wie sich die Größe der Stichprobe anpassen lässt, um Speichervorgaben einzuhalten. Der zweite Punkt zeigt auf, wie durch Verkürzung der Pfade weiterer Speicherplatz eingespart werden kann. Auf die speichereffiziente Speicherung der Datenstrukturen des Algorithmus und das laufzeit-& energieeffiziente Komprimieren der Datenstrukturen wird zuletzt eingegangen. Die eben genannten Techniken werden bei der Auswahl der Stichprobe angewandt, dadurch ergeben sich keine weiteren Laufzeitkosten während des Betriebs. Bis auf das Entpacken komprimierter Daten werden durch diese Techniken keine weiteren

Ressourcen des eingebetteten Systems genutzt. Allerdings muss bei der Modifikation der Stichprobe beachtet werden, dass die Konfigurationen dennoch ausreichend abdeckt sind. Ansonsten werden die Verzögerungen von Kanten und Pfaden, die nicht in der Stichprobe enthalten sind, nicht vom Algorithmus überprüft und es kann zu unerkannten Verzögerungsfehlern und dadurch zum Versagen des Systems kommen.

4.5.1 Stichprobe aller Pfade

Schon bei kleinen Schaltkreisen ist es, wie in Abschnitt 3.1 gezeigt, nicht möglich alle Pfade aufzulisten bzw. abzuspeichern. Allerdings ist es möglich, die aktivsten und längsten Pfade des Schaltkreises zu betrachten. Die Aktivität kann z.B. durch Analyse der Schaltaktivität bestimmt werden. Darauf wird in dieser Arbeit nicht eingegangen. Stattdessen wird angenommen, dass die aktivsten Pfade u.a. durch Booleschen Funktionsgeneratoren (LUTs) verlaufen, deren Ausgang nicht konstant ist. Bei diesen Pfaden ist der Einfluss der Alterung kritisch: Die Alterung ist durch häufige Signalübertragungen stark vorangeschritten und die Wahrscheinlichkeit, dass es durch die Nutzung dieser Pfade zu Verzögerungsfehlern kommt, ist hoch. Sie können deshalb als Indikator für die restlichen Pfade des Schaltkreises verwendet werden.

Das verwendete Zeitverhalten-Analyse Werkzeug (TRACE) erlaubt es die l längsten Pfade, die durch eine Stelle verlaufen oder die an einer bestimmten Stelle starten bzw. enden, zu extrahieren. Es lassen sich z.B. die l längsten Pfade, die durch eine LUT, Slice, Slice-Pin oder einen Ausgang verlaufen, extrahieren. Die Größe der Stichprobe lässt sich nun durch folgende Punkte manipulieren:

- Einstellung des Parameters l - damit wird die Anzahl der zurückzugebenden längsten Pfade pro Stelle definiert. Je geringer diese ist, desto weniger Pfade sind später in der Stichprobe.
- Grobheit der zu durchlaufenen Stelle - Sei n die Anzahl der Slices, k die Anzahl der LUTs in jeder Slice, und l die Anzahl der zu extrahierenden längsten Pfade. Werden die längsten Pfade durch eine Slice gesucht, werden maximal $l \cdot n$ vom TRACE Werkzeug zurückgegeben. Werden LUTs als zu durchlaufende Stelle ausgewählt, so werden maximal $k \cdot l \cdot n$ Pfade ausgegeben (typischerweise: $k = 4$). Werden die Pins der Slices bzw. LUTs ausgewählt, so erhöht sich die Anzahl der Pfade erneut.
- Kombination der zu durchlaufenden Stellen - Beispielsweise lassen sich die l_0 längsten Pfade durch die Ausgänge einer Schaltung und die l_1 längsten Pfade durch eine LUT in einer Stichprobe kombinieren.

Der Nachteil dieser Technik ist, dass, insbesondere bei groben zu durchlaufenden Stellen, viele Kanten und Pfade nicht in der Stichprobe enthalten sind. Werden beispielsweise die l längsten Pfade durch eine Slice extrahiert, können diese l Pfade alle durch dieselbe LUT verlaufen. Die anderen LUTs wären in einem solchen Fall nicht von der Stichprobe abgedeckt. Um alle Haftfehler zu erkennen, muss jede Kante mindestens einmal in der Stichprobe enthalten sein. Haftfehler sind z.B. durchtrennte Verbindungsleitungen oder konstante Ein-/Ausgänge von Komponenten wie LUTs. Ein weiteres Problem ist die Pfadredundanz, die bei der Kombination verschiedener zu durchlaufender Stellen entstehen kann. Diese Redundanz kann jedoch durch nachträgliches Filtern entfernt werden.

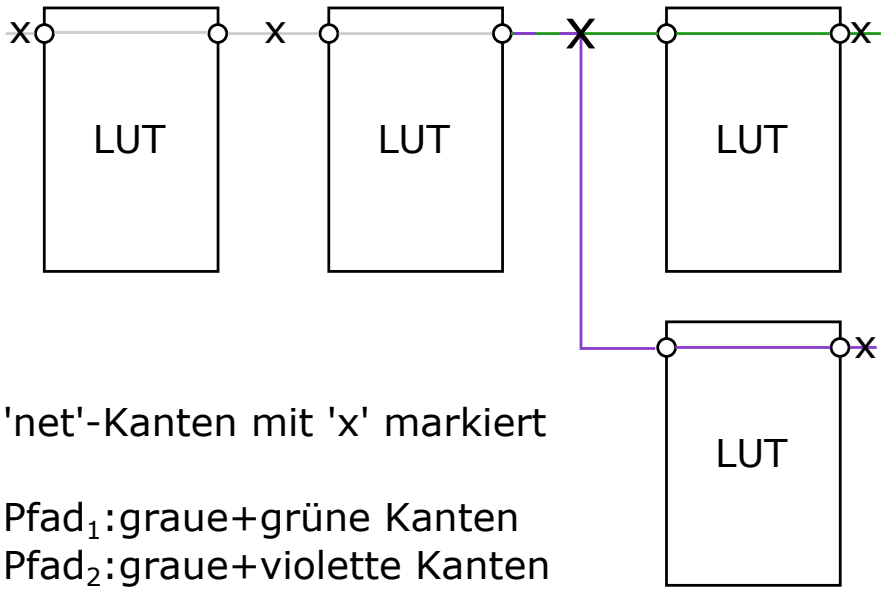


Abbildung 4.5: Verkürzung von Pfaden aufgrund von Redundanz oder 'net'-Kantentyp

4.5.2 Verkürzen der gespeicherten Pfade

Neben der Selektion von Pfaden zu einer Stichprobe, können einzelne Kanten der Pfade von der Stichprobe entfernt werden, um Speicherplatzvorgaben einzuhalten. Dafür muss entschieden werden, welche Kanten von welchen Pfaden entfernt werden. Beispielsweise können bestimmte Kantentypen von der Stichprobe entfernt werden. Eine Kante ist vom Typ 'net', falls sie zwei Komponentenpins miteinander verbindet, ohne dabei eine andere Komponenten zu benutzen. Dies ist bei Verbindungsleitungen der Fall. Andere Kantentypen bezeichnen die Komponentenpins, die durch die Kante verbunden werden und welche weitere Komponenten dabei durchquert werden. Wenn der Speicherplatz des eingebetteten Systems nicht ausreicht, um komplette Pfade abzuspeichern, ließen sich z.B. alle Kanten des Typs 'net' von der Stichprobe entfernen. Abbildung 4.5 zeigt ein Beispiel, in dem die 'net'-Kanten gestrichen werden (mit 'x' markiert). Wenn die 'net'-Kanten gestrichen werden, werden die Referenzverzögerungen für die Kante nicht vom Algorithmus nachgeschlagen, sondern die nominelle Pfadverzögerung angenommen. Ein Nachteil dieser Methode ist also, dass manche Pfade, deren Zeitverhalten nicht mehr den Vorgaben entsprechen, nicht vom Algorithmus erfasst werden können, wenn die Verletzung des Zeitverhaltens aufgrund einer nicht gespeicherten Kante auftritt. Wenn Kanten von der Stichprobe entfernt werden, ist es zudem wahrscheinlicher, dass Verzögerungsfehler, die durch geringfügige Verzögerung vieler Kanten auftreten, nicht erkannt werden. Grund dafür sind verkürzte Pfade, deren Referenzpfadverzögerung wegen der fehlenden Kanten nicht vollständig nachgeschlagen wird.

Andererseits lassen sich so harte Verzögerungsfehler speichereffizient entdecken, indem jede Kante genau einmal in der Stichprobe gespeichert wird. Die Pfade der Stichprobe werden dafür um redundante Kanten verkürzt. Beispielsweise sind die grauen Kanten des vorherigen Abbilds (Abb. 4.5) redundant und kommen in mehreren Pfaden vor. Daher können alle Pfade bis auf einen, in denen die

Kanten vorkommen um diese verkürzt werden. Hierbei muss auch entschieden werden, welche Pfade gekürzt werden. Eine Strategie ist, die längsten Pfade nicht zu verkürzen, da diese kritisch für das Einhalten der Zeitverhaltenvorgaben sind und jede verzögerte Kante zu Verzögerungsfehlern führen kann.

4.5.3 Speichereffiziente Datenstruktur

Um den Speicheraufwand des Algorithmus noch weiter zu reduzieren, müssen speichereffiziente Datenstrukturen eingesetzt werden. Die Datenstrukturen sollen dabei nur die nötigsten Informationen speichern, und diese Informationen sollen speichereffizient kodiert werden. Im Folgenden wird von Byte-Adressierung ausgegangen. Der Algorithmus benötigt Pfade, Kanten und die Information, welche Kanten in welchen Pfaden enthalten sind, um die Konfigurationsvorschläge berechnen zu können. Durch die Speicherung von Kanten und Pfaden im selben partiellen Segment wird die Zuordnung von Kanten zu Pfaden implizit festgehalten. Um Pfade abzuspeichern muss festgehalten werden, in welchem Hardwareentwurf und in welcher Konfiguration der Pfad vorkommt und welche Pfadverzögerung der Pfad besitzt. Die Konfigurationen und Hardwareentwürfe können durchnummeriert werden, womit die Speicherung dieser Zuordnung

$$\left\lceil \frac{log_2(\#\text{ Hardwareentwürfe}) + log_2(\max(\#\text{ Konfigurationen pro Entwurf}))}{8} \right\rceil$$

Bytes benötigt.

Die Pfadverzögerung wird durch TRACE als eine Festkommazahl mit drei Nachkommastellen ausgegeben. Der Algorithmus multipliziert den Wert der Pfadverzögerung mit Tausend und muss somit keine Kommastellen berücksichtigen. Anschließend wird die Pfadverzögerung als 16-bit Wert gespeichert. Die Referenzpfadverzögerungen müssen vom Algorithmus auf die Kanten zugeordnet werden, auf die sie sich beziehen. Daher muss jede Kante vom Algorithmus identifiziert werden können. Wenn die rekonfigurierbaren Hardwarebereiche eine reguläre Struktur besitzen, können die Kanten über eine Reihenfolge identifiziert werden. Dadurch wird kein weiterer Speicherplatz für die Identifikation der Kanten benötigt. Andernfalls lassen sich die Kanten über den Anfangs- und Endpin und derer Komponente identifizieren. Die Verzögerung der Kanten kann als 16-bit Wert gespeichert werden, wenn wie bei den Pfadverzögerungen vorgegangen wird.

4.5.4 Laufzeit- und energieeffizientes Packen der Daten

Als weitere Option zur Reduzierung des Speicheraufwands für die Stichprobe der Pfadverzögerungen, kommt die Nutzung von Kompressionsalgorithmen in Frage. Für diese Arbeit und der Ausführung auf einem eingebetteten System ist neben der Kompressionsrate, auch die Laufzeitkomplexität und Energieeffizienz des Kompressionsalgorithmus wichtig. Bei einer hohen Kompressionsrate und einer hohen Laufzeitkomplexität, ist zwar der benötigte Speicherplatz für die Stichprobe gering, allerdings wird die Rekonfiguration weiter verzögert, da die Bestimmung der Konfigurationsvorschläge durch das Entpacken der Daten mehr Zeit in Anspruch nimmt. Die Autoren von [BA06] haben verschiedene Kompressionsalgorithmen untersucht und bezüglich Laufzeit, Energieeffizienz und Kompressionsrate

einen Ranking ermittelt. Einer der Favoriten bezüglich Energie- und Laufzeiteffizienz (lzo) wird auch in dieser Arbeit verwendet und untersucht.

5 Ergebnisse

In diesem Kapitel wird der entworfene Algorithmus durch Experimente untersucht. Der Experimentaufbau wird in Abschnitt 5.1 erklärt und in Abschnitt 5.2 werden die unterschiedlichen Modelle für Pfadverzögerungen erläutert, die in den Experimenten verwendet werden. Die Adaptierung von Zeitverhalten-Variationen durch den Algorithmus wird in Abschnitt 5.3 untersucht. Die Wechselwirkung zwischen Laufzeit und Speicheraufwand des Algorithmus und der Stichprobengröße wird in 5.4 analysiert.

5.1 Experimentaufbau

Ein Experiment lässt sich in dieser Arbeit über folgende Eigenschaften charakterisieren:

- Menge der untersuchten Entwürfe.
- Anzahl der erzeugten diversifizierten Konfigurationen k_i [ZBK+13] pro Entwurf.
- Anzahl modellierter Hardwarebereiche h_j.
- Angewandte Verzögerungsverteilung - meist wird zusätzlich ein Parameter der zugrundeliegenden Wahrscheinlichkeitsverteilung variiert.
- Wertemenge des untersuchten Zeitpufferfaktors - die nominelle Pfadverzögerung des längsten Pfades jeder Konfiguration k_i wird mit einem Zeitpufferfaktor multipliziert. Daraus ergibt sich eine Nebenbedingung des Zeitverhaltens der Konfiguration, die vom entworfenen Algorithmus geprüft wird.
- Stichprobe der Pfade - diese wird durch Techniken, die in Abschnitt 4.5.1 und Abschnitt 4.5.2 vorgestellt wurden, modifiziert.

Die untersuchten Entwürfe stammen hauptsächlich aus den ITC'99 [Dav99], ISCAS'85 [Bry85] und ISCAS'89 [BBK89] Benchmarks, die der IWLS2005 [Alb05] Veröffentlichung entnommen wurden. Zusätzlich werden neun Hardwarebeschleuniger-Entwürfe des OTERA Projekts [OTE] verwendet.

Die Durchführung der Experimente ist wie folgt: Der vorgeschlagene Algorithmus wird mit den oben genannten Parametern gestartet und berechnet Konfigurationsvorschläge. Dafür berechnet der Algorithmus zuvor Modellverzögerungen nach einer Verzögerungsverteilung für jeden modellierten Hardwarebereich, sodass jede Verzögerungskante für jeden modellierten Hardwarebereich eine Modellverzögerung besitzt. Der Algorithmus bestimmt anschließend die Verzögerungen aller in der Stichprobe enthaltenen Pfade für jeden Hardwarebereich anhand der Modellverzögerungen. Überschreitet die Verzögerung eines Pfades einer Konfiguration k_i in einem Hardwarebereich h_j das

Produkt p_k aus Zeitpufferfaktor und der längsten Pfadverzögerung von k_i, so geht der Algorithmus davon aus, dass Verzögerungsfehler während des Betriebs auftreten, wenn k_i in h_j platziert wird. Hierbei wird angenommen, dass eine Konfiguration mit der Taktperiode p_k in einem Hardwarebereich betrieben wird. Die Autoren von [SHB11] beschreiben, wie die Taktgeber des FPGAs rekonfiguriert werden können, um jeden Hardwarebereich mit einem an die Konfiguration angepassten Taktsignal versorgen zu können. Der Algorithmus kann damit für jeden Hardwarebereich zurückmelden, welche Konfigurationsplatzierungen zu Verzögerungsfehlern führen würden.

Die Ausführung des entworfenen Algorithmus wird pro Datenpunkt 100 Mal wiederholt, um Ausreißer im verwendeten Verzögerungsverteilung auszugleichen. Anschließend wird ein Schaubild gezeichnet, das den Einfluss der Parameter auf die Platzierungen zeigt.

5.2 Angenommene Verzögerungsverteilungen

Um Alterung bzw. Variationen des Zeitverhaltens zu modellieren werden die Verzögerungen der Verzögerungskanten vor der Ausführung des Algorithmus anhand einer Verzögerungsverteilung angepasst. Die Verzögerungsverteilungen basieren auf zugrundeliegenden Wahrscheinlichkeitsverteilungen. Für die mathematischen Hintergründe der genutzten Verteilungen wird auf [MFJ65] verwiesen. Die hier vorgestellten Verzögerungsverteilung modellieren Defekte auf einer hohen abstrakten technologieunabhängigen Ebene, wodurch Ungenauigkeiten erwartet werden [Wun13]. Durch Verzögerungsverteilungen soll der vorgeschlagene Ansatz untersucht werden, indem die Verzögerung mancher Komponenten erhöht wird und anschließend geprüft wird, ob dies zu Verzögerungsfehlern führt und ob diese durch die Nutzung des vorgeschlagenen Algorithmus adaptiert werden können.

Die nominelle Verzögerung einer Komponente ist der Erwartungswert und wird im Folgenden μ genannt. σ ist die Standardabweichung der Normalverteilung. Sie wird genutzt, um mittels Verzögerungsverteilungen eine Modellverzögerung zu berechnen. Anschließend bestimmt der Algorithmus anhand der Modellverzögerungen der Komponenten Vorschläge für die Platzierung von Konfigurationen in Hardwarebereiche. Nur die Verzögerungen der in der Stichprobe enthaltenen Verzögerungskanten werden über die Verzögerungsverteilung neu berechnet.

Die Experimente dieses Abschnitts besitzen eine Stichprobe, die den längsten Pfad pro Slice beinhaltet. Weiter wird eine Konfiguration pro Entwurf und ein modellierter Hardwarebereich als Parameter eingegeben. Damit zeigen diese Experimente wie sich das Zeitverhalten der Entwürfe unter den gegebenen Verzögerungsverteilungen entwickelt, ohne dass der vorgeschlagene Algorithmus zur Adaptierung an Zeitverhalten-Variationen genutzt wird.

5.2.1 Normalverteilung

Die Normalverteilung wird genutzt, um Variationen des Zeitverhaltens der Komponenten einer Schaltung zu modellieren. Hierzu wird eine Standardnormalverteilung auf die nominelle Verzögerung einer Komponente angewandt. In Java wird die Modellverzögerung einer Komponente wie folgt vom Algorithmus berechnet:

$$\text{Modellverzögerung} \;=\; \text{Random.nextGaussian()} \;\cdot (\mu \cdot \sigma) + \mu$$

Wobei die Standardabweichung als Anteil der nominellen Verzögerung gegeben ist. Die Methode "Random.nextGaussian()"gibt zufällige standardnormalverteilte Gleitkomma Zahlen in $[-1, 1]$ zurück. Diese Verteilung soll Variationen des Zeitverhaltens modellieren, bei der die Abweichung der tatsächlichen Verzögerung von der nominellen Verzögerung normalverteilt ist. Die Annahme hinter diesem Modell ist, dass die tatsächliche Verzögerung einer Komponente geringer oder höher sein kann als die angenommen nominelle Verzögerung, aber in der Regel nicht 'stark' von der spezifizierten Standardabweichung abweicht.

Abbildung 5.1 zeigt wie der Zeitpufferfaktor gewählt werden muss, damit die Konfiguration eines Entwurfs in 98% aller Wiederholungen fehlerfrei platziert werden kann. Die Pfadverzögerungen der betrachteten Konfigurationen sind dabei normalverteilt mit Standardabweichung $\sigma = 0, 0.01, .., 0.25$.

In Abbildung 5.1 kann beispielsweise abgelesen werden, dass eine Konfiguration des s382 Entwurfs in 98% der Instanzen fehlerfrei in einem Hardwarebereich platziert werden kann, falls ein Zeitpufferfaktor von ca. 0.2 bei einer Standardabweichung von ca. 0.06 gewählt wurde. Durch eine Erhöhung der Taktperiode um 20%, kann also an die Zeitverhalten-Variationen adaptiert werden, die durch die oben genannte Normalverteilung in die Konfiguration von s382 gestreut wurden. Weiterhin ist sichtbar, dass der s382 Entwurf im Vergleich zu den anderen betrachteten Entwürfen, den höchsten Zeitpufferfaktor benötigt, um an die Variationen zu adaptieren. Der Einfluss der normalverteilten Verzögerungen auf den SADrow_4 Entwurf ist hingegen gering: Bei einer Standardabweichung von 0.25 müsste die Taktperiode lediglich um 10% erhöht werden, um an die Zeiverhalten-Variationen zu adaptieren. Die Szenarien der weiteren Schaubilder dieses Abschnitts sind bis auf die Wahl und Parametrisierung der Wahrscheinlichkeitsverteilung gleich.

5.2.2 Gefaltete Normalverteilung

Die gefaltete Normalverteilung gibt normalverteile Werte aus, deren Vorzeichen nicht beachtet werden. Der Java Code für die Berechnung der Modellverzögerung ähnelt dem der Normalverteilung: In Java wird die Modellverzögerung einer Komponente wie folgt vom Algorithmus berechnet:

$$\text{Modellverzögerung } = |\text{Random.nextGaussian()}| \cdot (\mu \cdot \sigma) + \mu$$

Die Annahme dieses Modells ist, dass die tatsächliche Verzögerung einer Komponente nicht kleiner sein kann als die angenommene nominale Verzögerung. Dadurch sollen Alterungseffekte, die ausschließlich zu einer erhöhten Verzögerung führen, modelliert werden.

Abbildung 5.2 zeigt wie der Zeitpufferfaktor gewählt werden muss, damit die Konfiguration eines Entwurfs in 98% aller Wiederholungen fehlerfrei platziert werden kann. Die Standardabweichung der gefaltet-normalverteilten Pfadverzögerungen ist $\sigma = 0, 0.01, .., 0.25$. Im Vergleich zur Abbildung 5.1 ist der benötigte Zeitpuffer für den s382 Entwurf in Abbildung 5.2 nur leicht erhöht. Zudem sind zwei Trends erkennbar:

- Die benötigten Zeitpufferfaktoren für eine, in 98% aller untersuchten Instanzen, verzögerungsfehlerfreie Platzierung der Entwürfe verlaufen dichter beieinander.

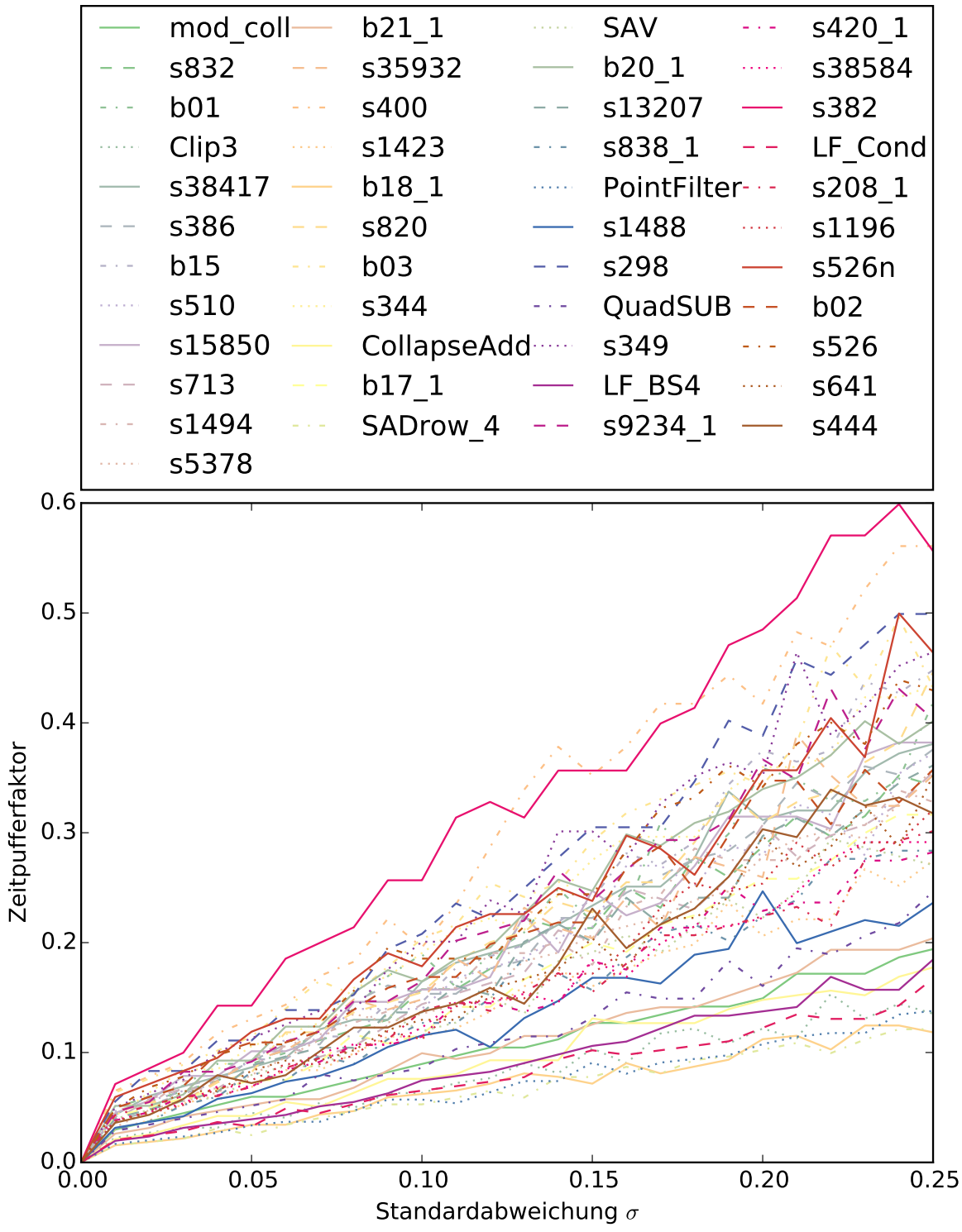


Abbildung 5.1: Zeitpufferfaktor in Abhängigkeit zur Standardabweichung (Normalverteilung) für die untersuchten Benchmark-Schaltungen

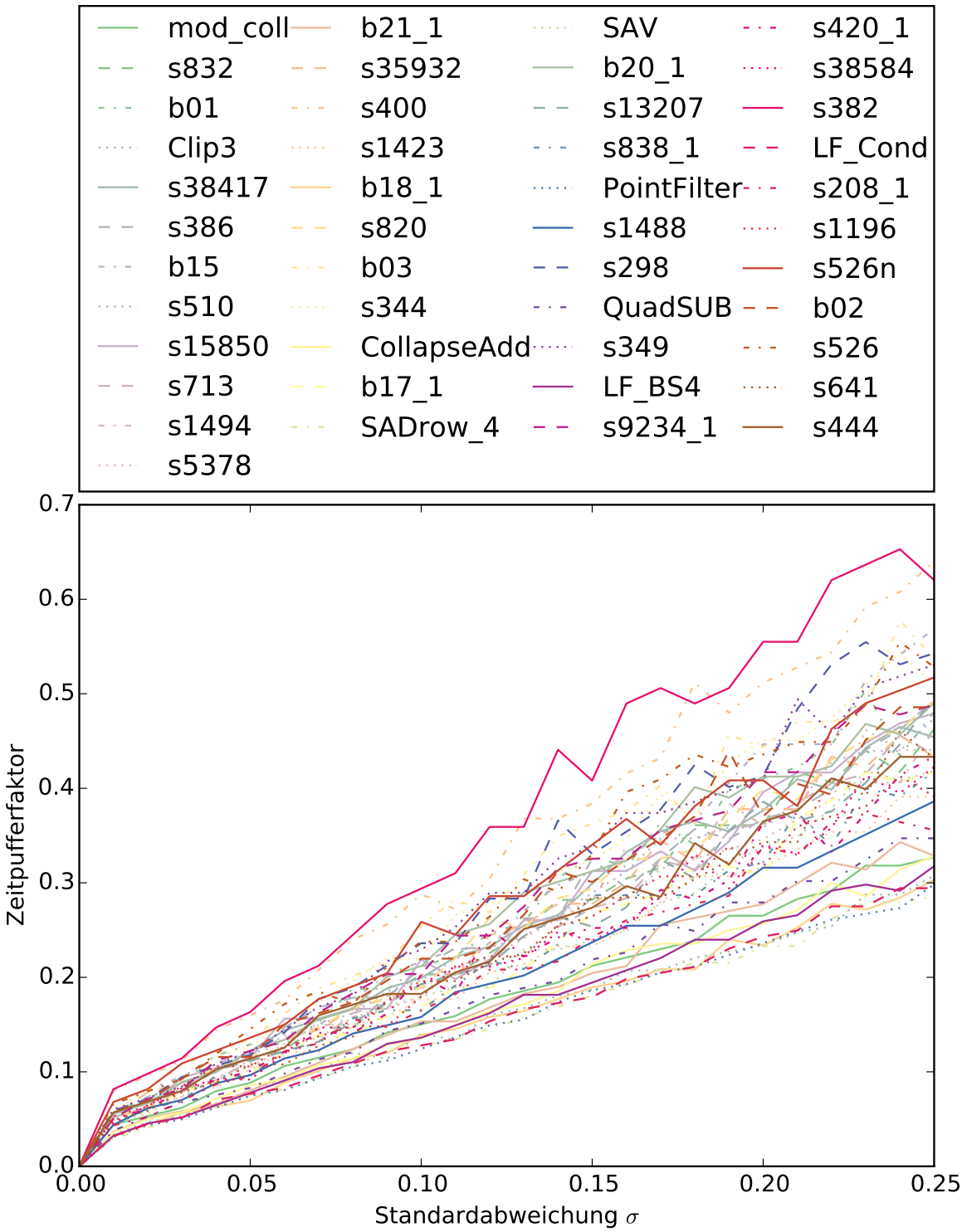


Abbildung 5.2: Zeitpufferfaktor in Abhängigkeit zur Standardabweichung (gefaltete Normalverteilung) für die untersuchten Benchmark-Schaltungen

- Entwürfe, deren Zeitverhalten sich geringfügig von der Normalverteilung beeinflussen lassen (z.B. SADrow_4), benötigen bei der gefalteten Normalverteilung bei gleicher Standardabweichung einen höheren Zeitpuffer.

Die gefaltete Normalverteilung nimmt im Vergleich zur Normalverteilung unproportional Einfluss auf das Zeitverhalten der Entwürfe: der benötigte Zeitpufferfaktor für $\sigma = 0.25$ hat sich beim SADrow_4 Entwurf fast verdreifacht, wohingegen sich der Zeitpufferfaktor des s382 Entwurfs sich von 0.6 auf 0.65 erhöht hat.

5.2.3 Verschobene Normalverteilung

Bei der verschobenen Normalverteilung wird neben der Standardabweichung auch der Erwartungswert während der Experimente verändert. In Java wird die Verteilung vom Algorithmus wie folgt implementiert:

$$\text{Modellverzögerung} \;=\; \text{Random.nextGaussian()} \;\cdot (\mu \cdot \sigma) + \mu \cdot (1 + \delta)$$

Standardabweichung und Erwartungswert sind erneut als Anteil an der nominellen Verzögerung zu verstehen. Bei diesem Modell wird angenommen, dass die Verzögerung vieler Komponenten, um einen an der nominellen Verzögerung anteiligen Wert verschlechtert wird. Weiterhin degradieren die Komponenten unterschiedlich stark.

Abbildung 5.3 zeigt wie der Zeitpufferfaktor gewählt werden muss, damit die Konfiguration eines Entwurfs in 98% aller Wiederholungen fehlerfrei platziert werden kann. Die Standardabweichung und Erwartungswert der gefaltet-normalverteilten Pfadverzögerungen ist wie oben beschrieben. In Abbildung 5.3 ist erkennbar, dass die benötigten Zeitpufferfaktoren für verzögerungsfehlerfreie Platzierungen durchschnittlich, im Vergleich zur Normalverteilung und zur gefalteten Normalverteilung, weiter erhöht sind.

5.2.4 Gleichverteilung

Mit der Gleichverteilung sollen die Verzögerungseffekte von Punktdefekten modelliert werden. Jede Verzögerungskante hat eine Wahrscheinlichkeit p von einem Punktdefekt getroffen zu werden. Ist eine Verzögerungskante von einem Punktdefekt betroffen, so ergibt sich deren Modellverzögerung aus der Multiplikation eines Verzögerungsfaktors d mit der nominellen Verzögerung. Für den Algorithmus wurde dies wie folgt in Java implementiert:

$$\text{Modellverzögerung} \;=\; \begin{cases} \mu \cdot d, & \text{falls Random.nextDouble()} \;< p \\ \mu, & \text{sonst} \end{cases}$$

Abbildung 5.4 zeigt wie der Zeitpufferfaktor gewählt werden muss, damit die Konfiguration eines Entwurfs in 98% aller betrachteten Instanzen fehlerfrei platziert werden kann. Die Wahrscheinlichkeit, dass die Verzögerung einer Verzögerungskante erhöht wird, wird von $p = \{0.1, 0.11, ..., 0.15\}$ variiert. Der Verzögerungsfaktor wurde auf $d = 2$ gesetzt - die Verzögerung einer Verzögerungskante wird verdoppelt, falls sie von einem Punktdefekt betroffen ist. Unter den gewählten Verzögerungsverteilun-

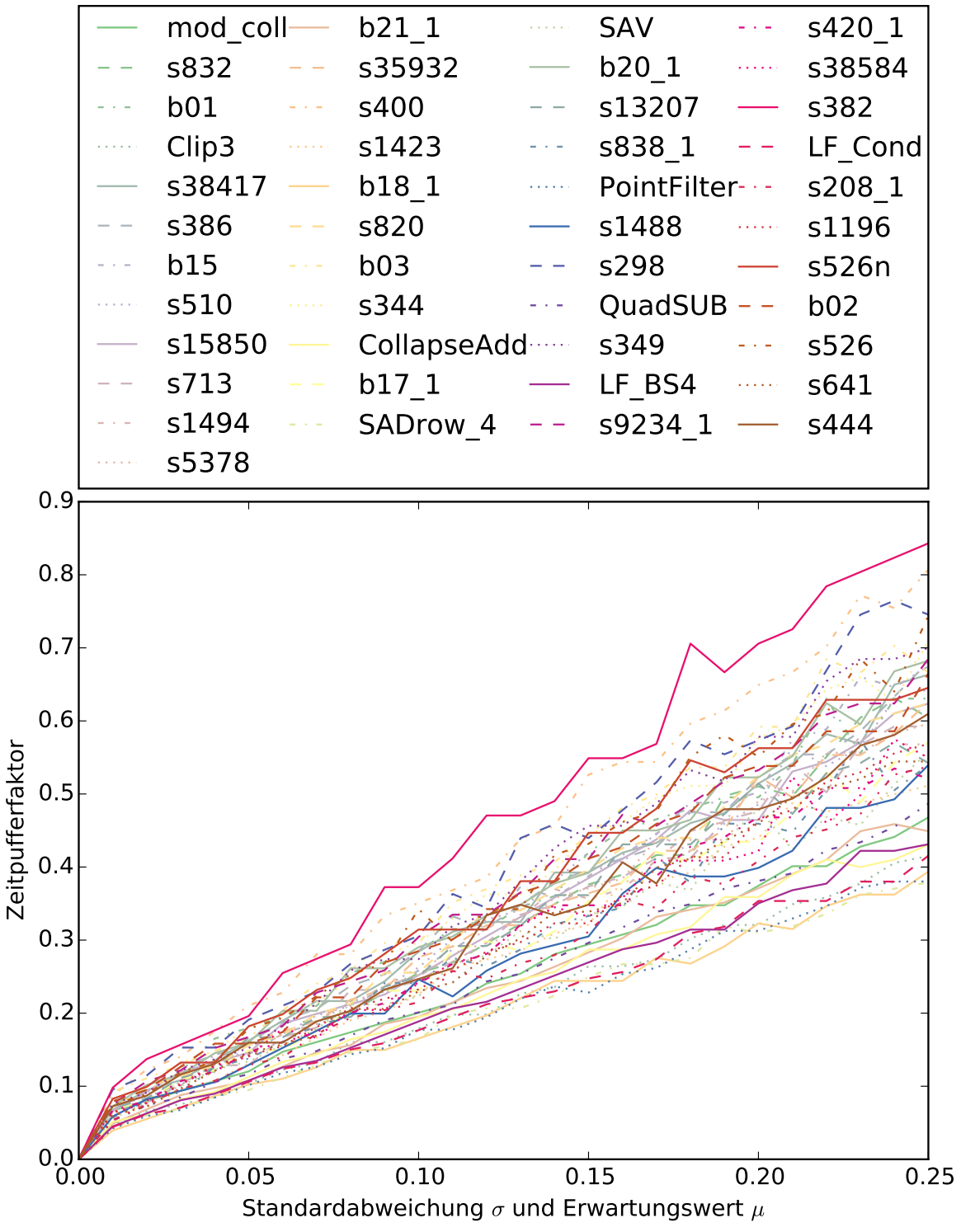


Abbildung 5.3: Zeitpufferfaktor in Abhängigkeit zur Standardabweichung/Erwartungswert (verschobene Normalverteilung) für die untersuchten Benchmark-Schaltungen

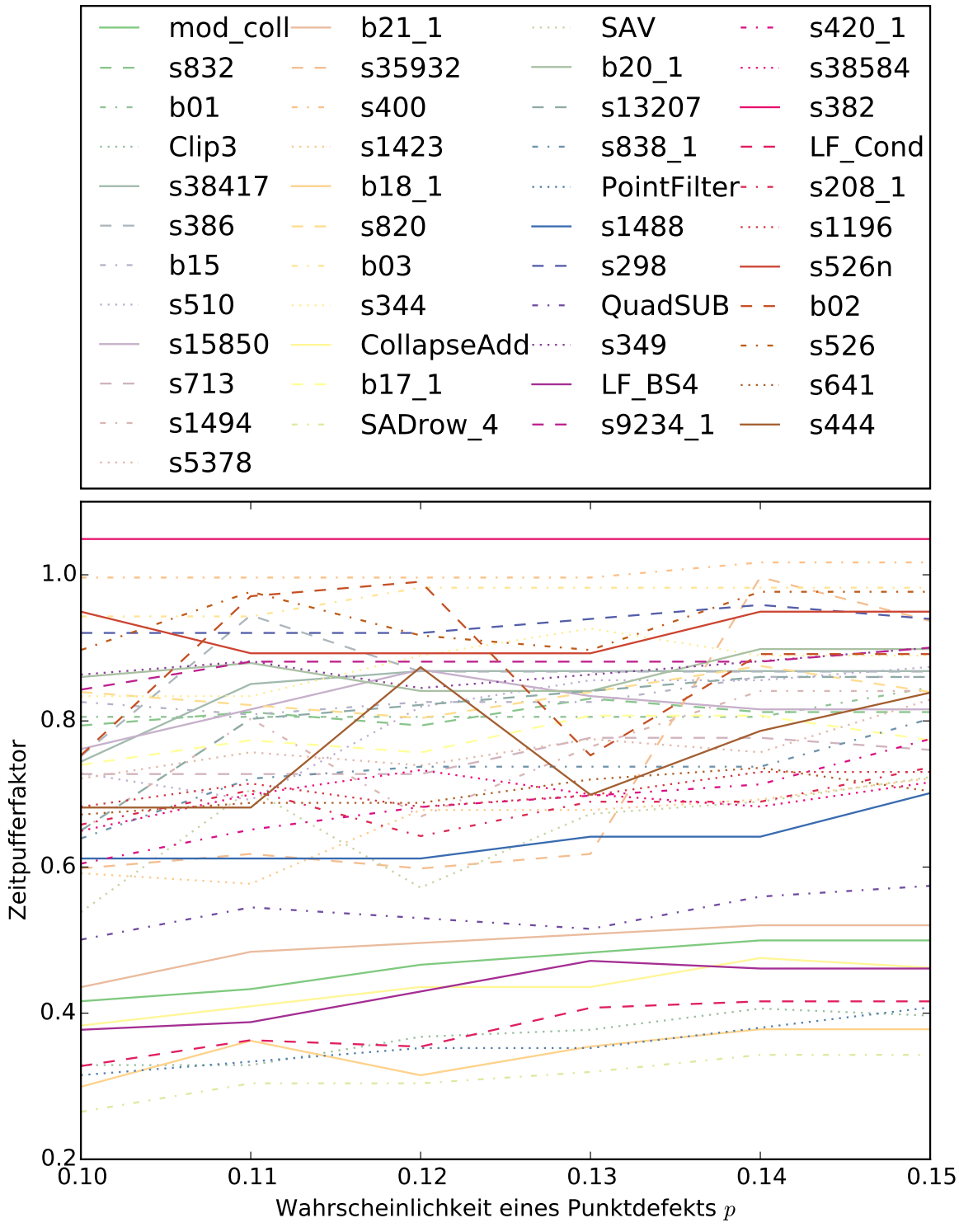


Abbildung 5.4: Zeitpufferfaktor in Abhängigkeit zur Wahrscheinlichkeit eines Punktdefekts für die untersuchten Benchmark-Schaltungen

gen ist der benötigte Zeitpufferfaktor für verzögerungsfreie Platzierungen mit fast 1.1 am höchsten. Der Zeitpufferfaktor für den s382 Entwurf verläuft dabei horizontal und die Wahrscheinlichkeit eines Punktdefekts ist für den gewählten Ausschnitt nicht ausschlaggebend für die Höhe des benötigten Zeitpufferfaktors.

5.3 Adaption an Zeitverhalten-Variationen

Im Abschnitt Adaption an Zeitverhalten-Variationen wird die Effektivität des Ansatzes bzw. des Algorithmus untersucht. Dazu werden zunächst die Experimente des vorherigen Abschnitts mit einer höheren Anzahl an Konfigurationen und modellierter Hardwarebereichen wiederholt, um einen Überblick über die Adaptionsfähigkeit an Zeitverhalten-Variationen für die betrachteten Entwürfe durch den vorgeschlagenen Algorithmus zu untersuchen. Anschließend wird der s382 Benchmark Entwurf, aufgrund der Ergebnisse des Abschnittes 5.2 bezüglich des benötigten Zeitpufferfaktors, im Detail betrachtet. Für den diesen Abschnitt wurde die Anzahl Konfigurationen und die Anzahl modellierter Hardwarebereiche auf 10 gesetzt. Die Benchmark Entwürfe wurden einzeln und nicht im Verbund untersucht. D.h. es existieren 100 Platzierungsmöglichkeiten für jeden betrachteten Entwurf, da von jedem Entwurf 10 diversifizierte Konfigurationen erzeugt wurden, die jeweils in 10 modellierten Hardwarebereichen platziert werden können.

5.3.1 Überblick

Die Abbildungen 5.5, 5.6, 5.7 und 5.8 zeigen in welcher Größenordnung der entworfene Algorithmus ein System mit 10 Konfigurationen und 10 Hardwarebereichen an Variationen des Zeitverhaltens adaptieren kann. Die Zeitverhalten-Variationen folgen dabei den Verzögerungsverteilungen, die in 5.2 vorgestellt wurden. Die Menge der betrachteten Entwürfe wurde jedoch angepasst, da für einige Entwürfe aufgrund ihrer Größe keine 10 diversifizierten Konfigurationen synthetisiert werden konnten oder die Zeitverhalten-Analyse durch die kommerziellen Werkzeuge eingeschränkt wurde.

Die Abbildungen dieses Unterabschnitts zeigen, dass der benötigte Zeitpufferfaktor für eine verzögerungsfehlerfreie (in 98% aller betrachteten Instanzen) Platzierung von Entwürfen enorm verringert werden kann, wenn mehrere Konfigurationen und Hardwarebereiche genutzt werden können. Neben der Parametrisierung und der Art der betrachteten Verzögerungsverteilungen, ist die Höhe des Zeitpufferfaktors hauptsächlich von der Struktur der untersuchten Entwürfe abhängig.

5.3.2 s382 Benchmark Entwurf

Die Abbildungen 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, 5.15 und 5.16 zeigen die Adaptierung des s382 Benchmark Entwurfs durch den vorgeschlagenen Algorithmus im Detail.

Jede Zelle der Matrixvisualisierung entspricht der durchschnittlichen Anzahl an verzögerungsfehlerfreien Entwurfsplatzierungen über 100 Instanzen bei einem gegeben Wert für die Standardabweichung (Spalte) und dem Zeitpufferfaktor (Zeile). Beispielsweise steht in Abbildung 5.11.a) die sechste Spalte

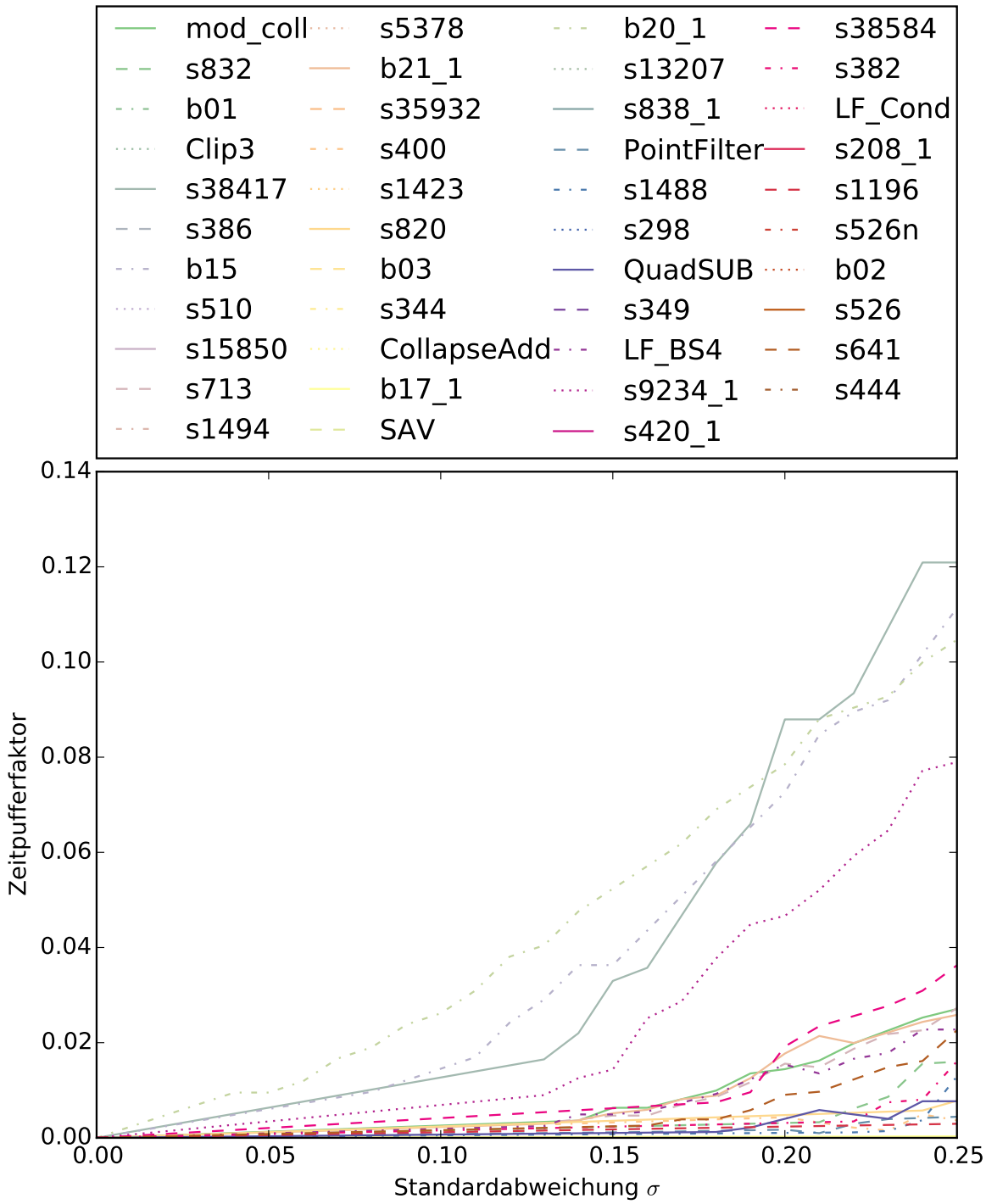


Abbildung 5.5: Zeitpufferfaktor in Abhängigkeit zur Standardabweichung (Normalverteilung) für die untersuchten Benchmark-Schaltungen und Adaption durch den Algorithmus mit 10 Konfigurationen und 10 modellierten Hardwarebereichen

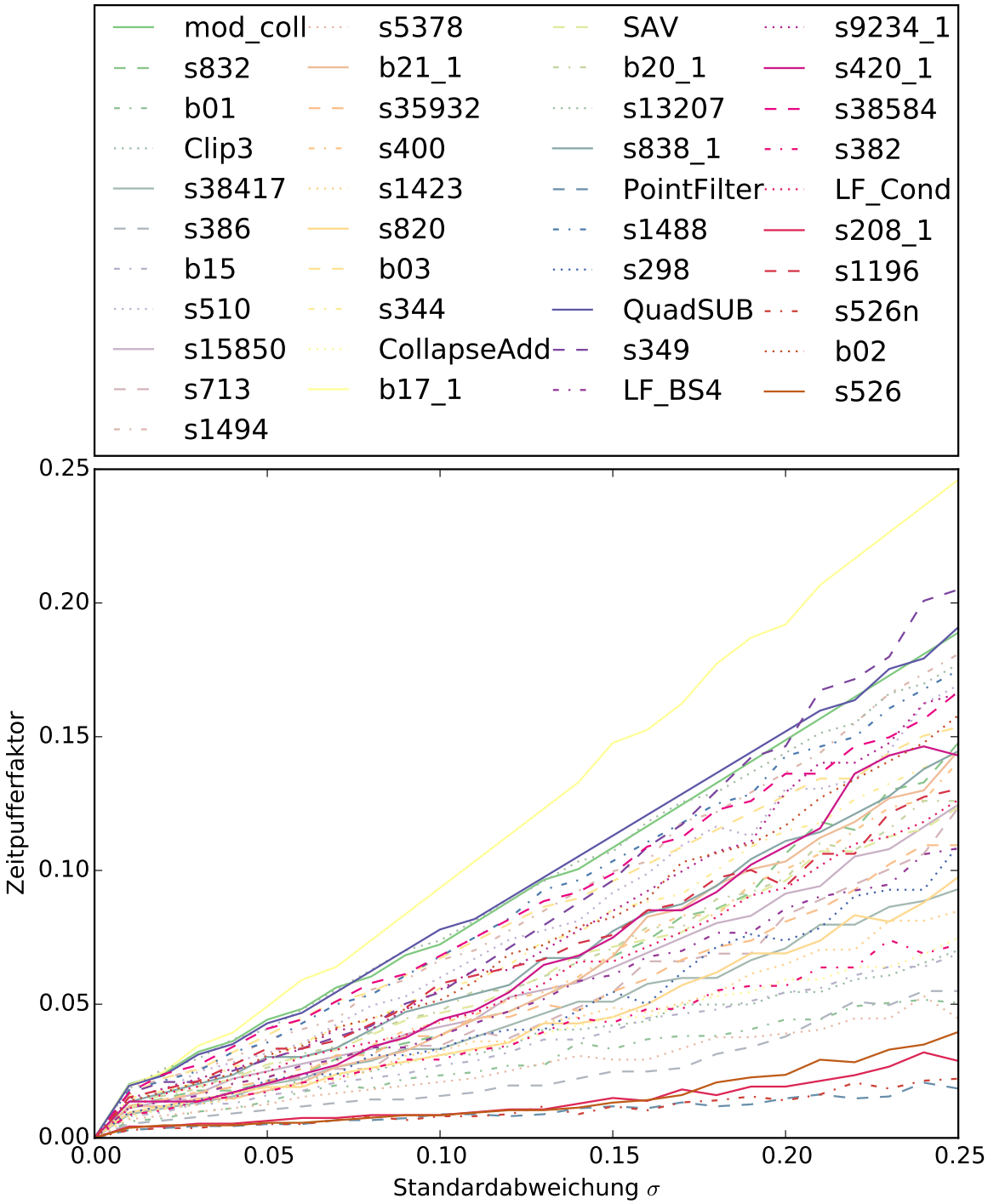


Abbildung 5.6: Zeitpufferfaktor in Abhängigkeit zur Standardabweichung (gefaltete Normalverteilung) für die untersuchten Benchmark-Schaltungen und Adaption durch den Algorithmus mit 10 Konfigurationen und 10 modellierten Hardwarebereichen

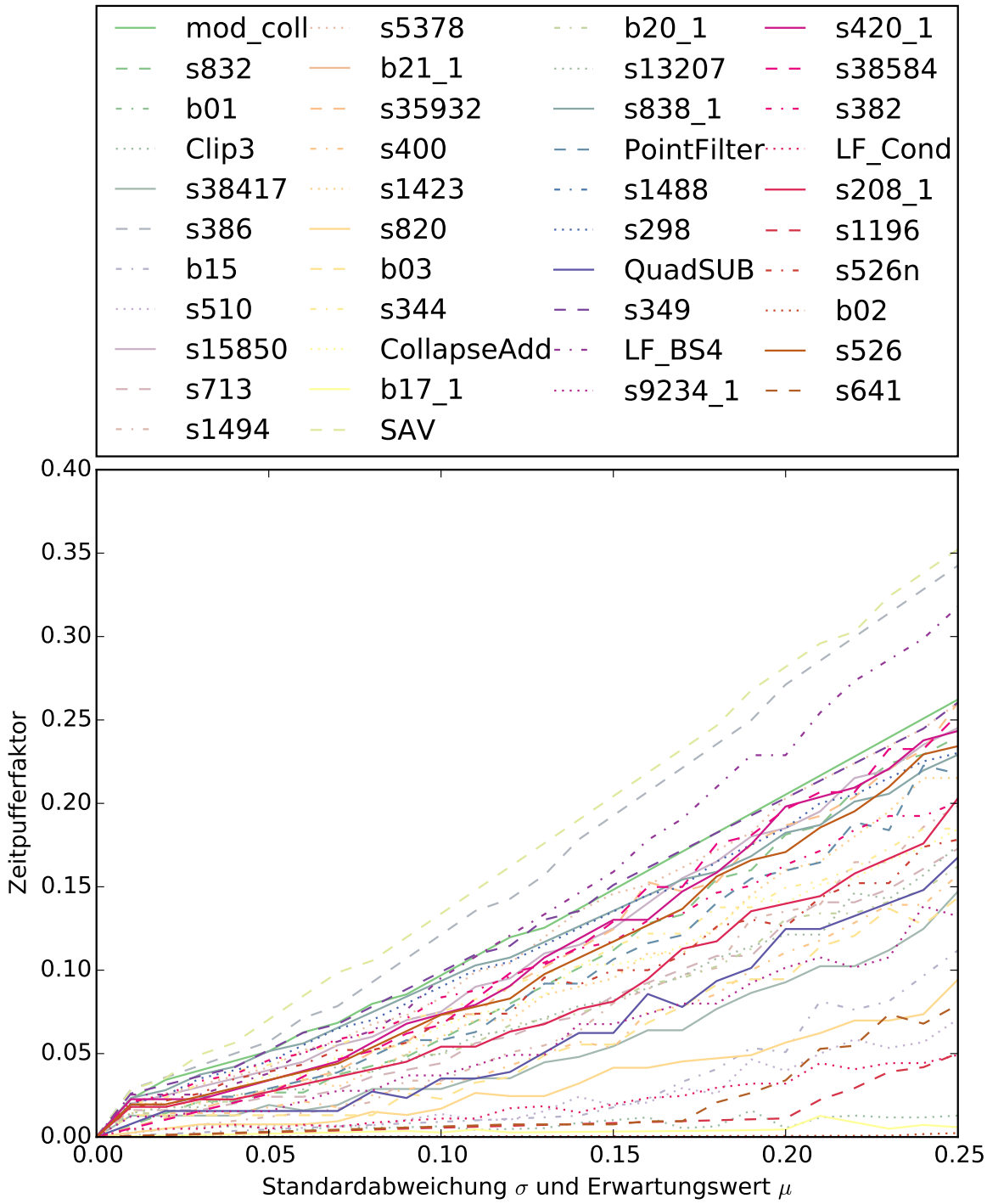


Abbildung 5.7: Zeitpufferfaktor in Abhängigkeit zur Standardabweichung/Erwartungswert (verschobene Normalverteilung) für die untersuchten Benchmark-Schaltungen und Adaption durch den Algorithmus mit 10 Konfigurationen und 10 modellierten Hardwarebereichen

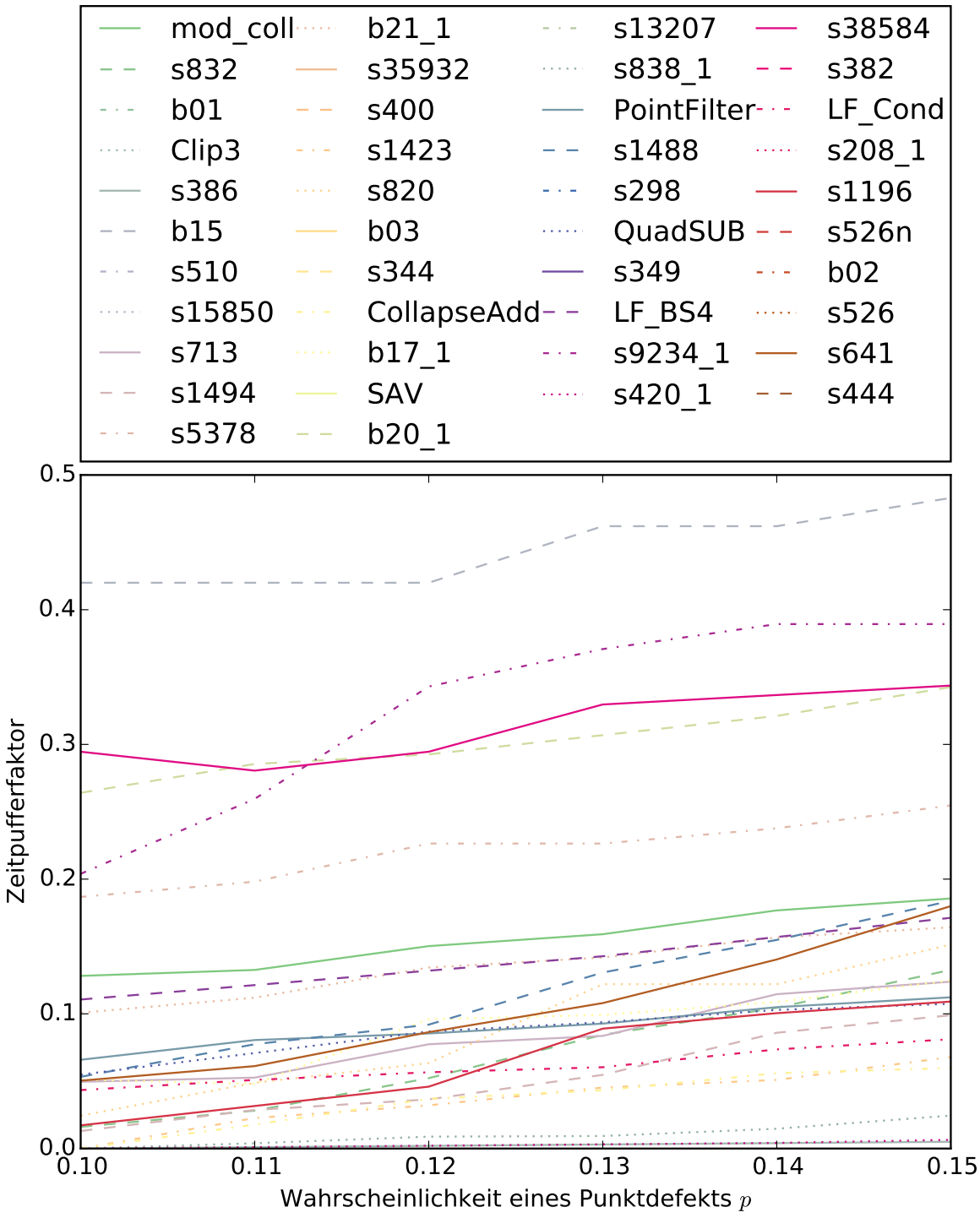
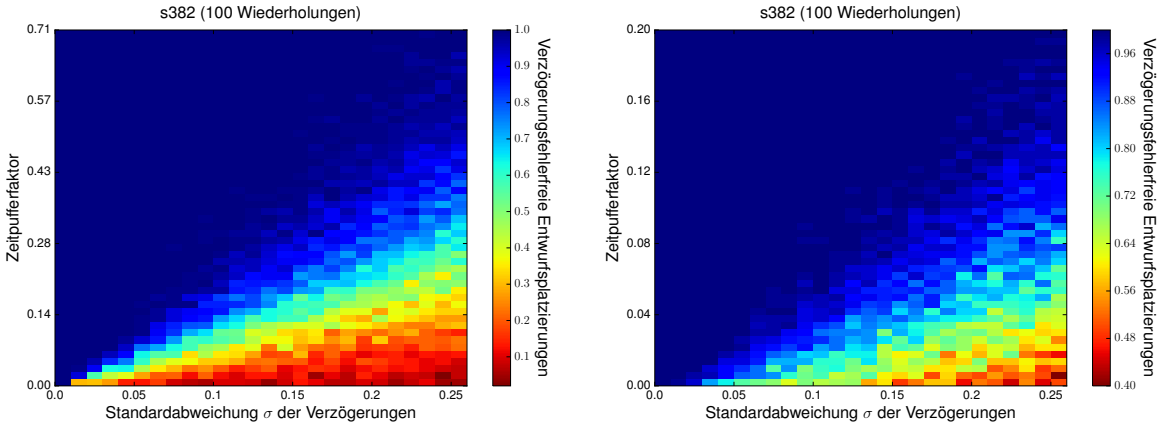


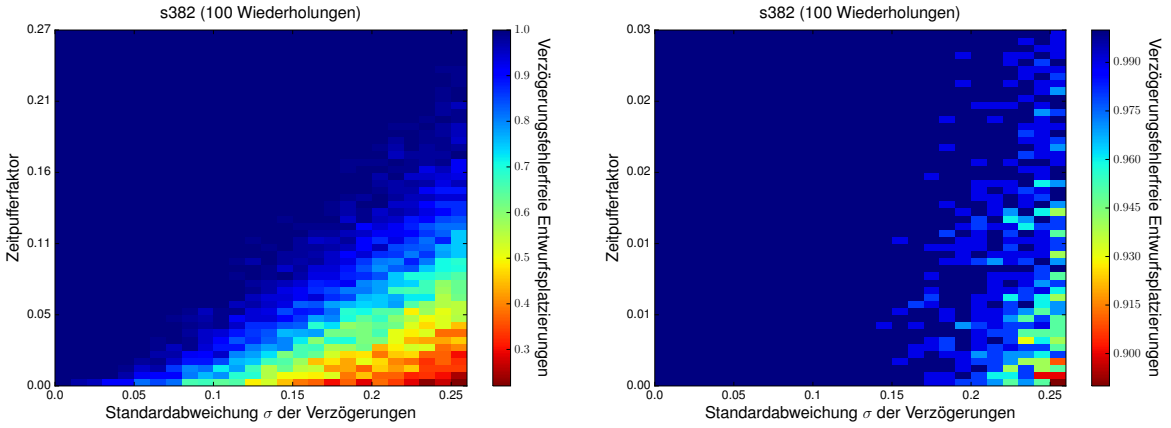
Abbildung 5.8: Zeitpufferfaktor in Abhängigkeit zur Wahrscheinlichkeit eines Punktdefekts für die untersuchten Benchmark-Schaltungen und Adaption durch den Algorithmus mit 10 Konfigurationen und 10 modellierten Hardwarebereichen



(a) Normalverteilung, eine Konfiguration, ein Hardwarebereich (ohne Adaption durch den Algorithmus)

(b) Normalverteilung, eine Konfiguration, 10 Hardwarebereiche

Abbildung 5.9: Adaptierung an Zeitverhalten-Variationen bei normalverteilten Verzögerungen und einer Konfiguration für den s382 Benchmark Entwurf

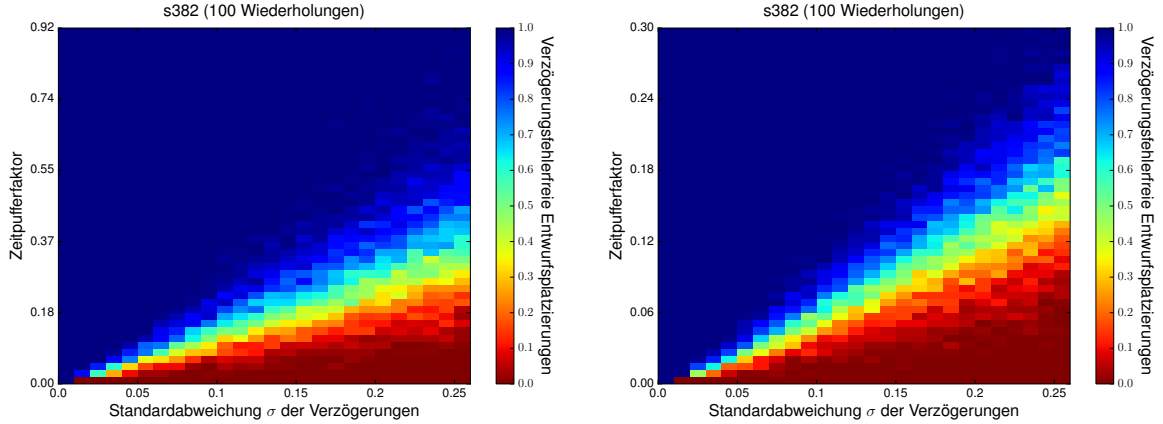


(a) Normalverteilung, 10 Konfiguration, ein Hardwarebereich

(b) Normalverteilung, 10 Konfiguration, 10 Hardwarebereiche

Abbildung 5.10: Adaptierung an Zeitverhalten-Variationen bei normalverteilten Verzögerungen und 10 Konfiguration für den s382 Benchmark Entwurf

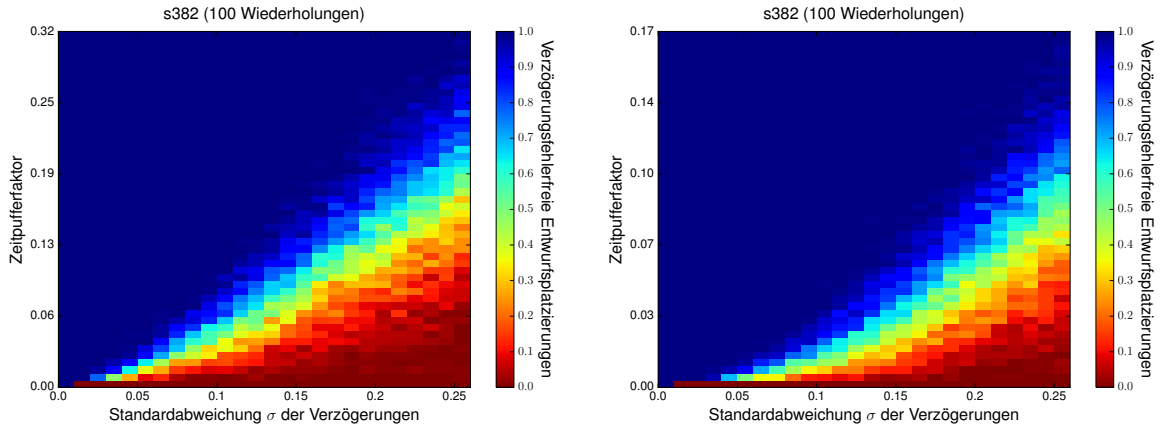
der Matrix für eine Standardabweichung von $\sigma = 0.06$ und die zweite Zeile der Matrix (vom Ursprung aus) für einen Zeitpufferfaktor von ca. 0.05. Die Zelle an der sechsten Spalte der zweiten Zeile der Matrix, gibt an, dass bei einem Zeitpufferfaktor von 0.05 und einer Standardabweichung von 0.05 ca. 25% der 100 versuchten Entwurfsplatzierungen durchgeführt werden könnten, ohne dass Verzögerungsfehler verursacht werden.



(a) gefaltete Normalverteilung, eine Konfiguration, ein Hardwarebereich (ohne Adaption durch den Algorithmus)

(b) gefaltete Normalverteilung, eine Konfiguration, 10 Hardwarebereiche

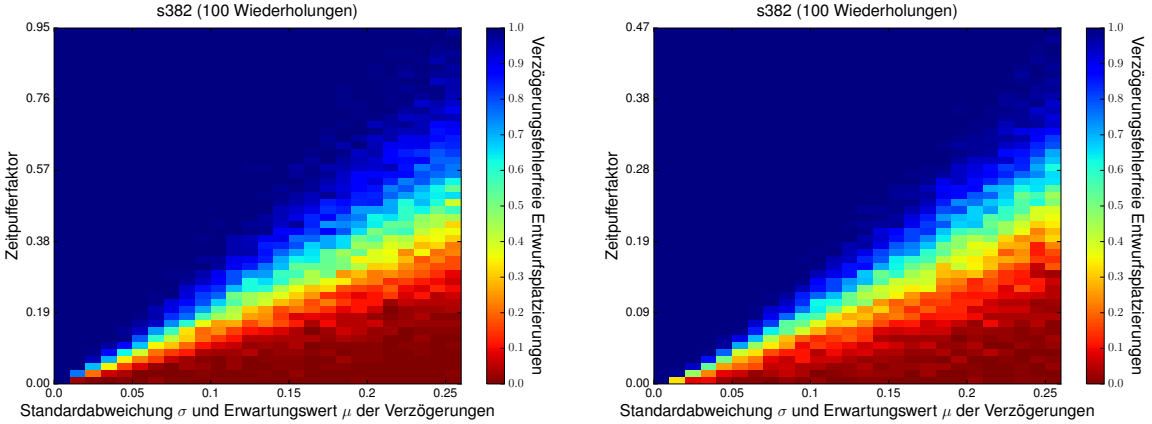
Abbildung 5.11: Adaptierung an Zeitverhalten-Variationen bei gefaltet-normalverteilten Verzögerungen und einer Konfiguration für den s382 Benchmark Entwurf



(a) gefaltete Normalverteilung, 10 Konfiguration, ein Hardwarebereich

(b) gefaltete Normalverteilung, 10 Konfiguration, 10 Hardwarebereiche

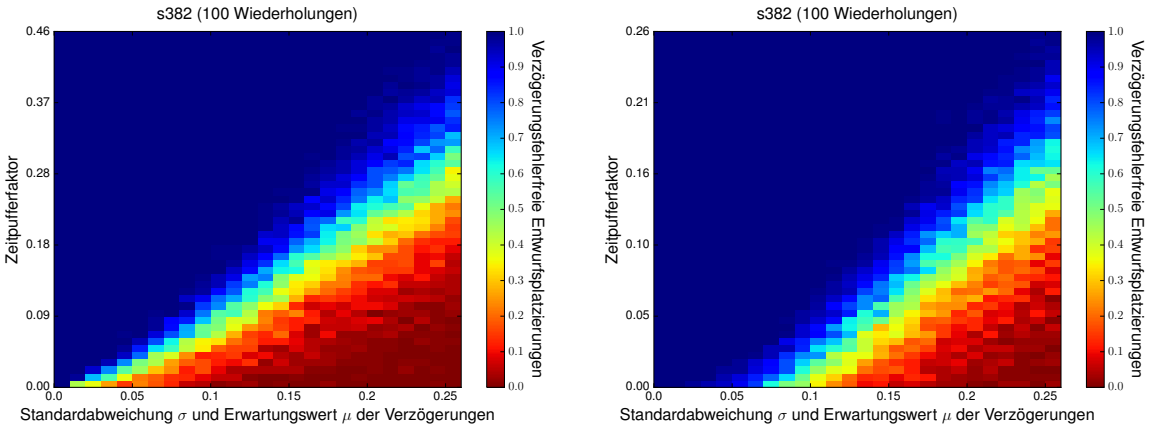
Abbildung 5.12: Adaptierung an Zeitverhalten-Variationen bei gefaltet-normalverteilten Verzögerungen und 10 Konfiguration für den s382 Benchmark Entwurf



(a) verschobene Normalverteilung, eine Konfiguration, ein Hardwarebereich (ohne Adaption durch den Algorithmus)

(b) verschobene Normalverteilung, eine Konfiguration, 10 Hardwarebereiche

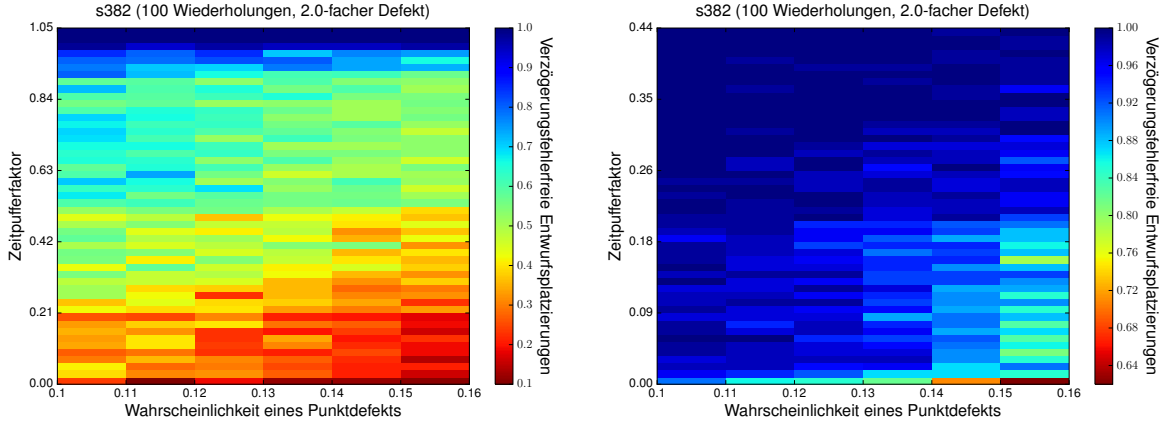
Abbildung 5.13: Adaptierung an Zeitverhalten-Variationen bei verschoben-normalverteilten Verzögerungen und einer Konfiguration für den s382 Benchmark Entwurf



(a) verschobene Normalverteilung, 10 Konfiguration, ein Hardwarebereich

(b) verschobene Normalverteilung, 10 Konfiguration, 10 Hardwarebereiche

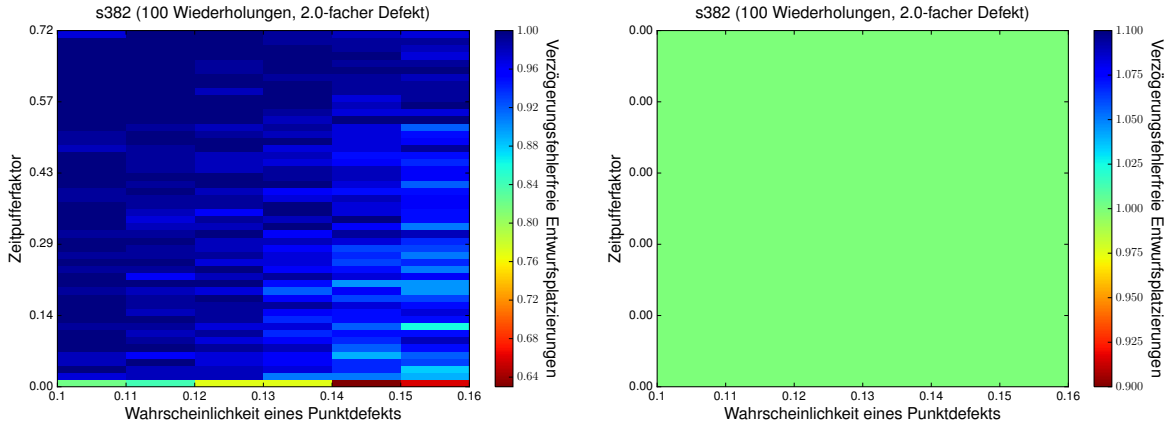
Abbildung 5.14: Adaptierung an Zeitverhalten-Variationen bei verschoben-normalverteilten Verzögerungen und 10 Konfiguration für den s382 Benchmark Entwurf



(a) Gleichverteilung, eine Konfiguration, ein Hardwarebereich (ohne Adaption durch den Algorithmus)

(b) Gleichverteilung, eine Konfiguration, 10 Hardwarebereiche

Abbildung 5.15: Adaptierung an Zeitverhalten-Variationen bei gleichverteilten Verzögerungen und einer Konfiguration für den s382 Benchmark Entwurf



(a) Gleichverteilung, 10 Konfiguration, ein Hardwarebereich

(b) Gleichverteilung, 10 Konfiguration, 10 Hardwarebereiche. Es war dem Algorithmus nicht möglich fehlgeschlagene Entwurfsplatzierungen zu finden.

Abbildung 5.16: Adaptierung an Zeitverhalten-Variationen bei gleichverteilten Verzögerungen und einer Konfiguration für den s382 Benchmark Entwurf

5.4 Speicherplatz und Laufzeit

Im Abschnitt Speicherplatz und Laufzeit werden die Merkmale unterschiedlich erzeugter Pfad-Stichproben für den s382 Benchmark Entwurf untersucht. Untersucht werden dabei Modifikationen der Stichprobe, die in Abschnitt 4.5.1 und Abschnitt 4.5.2 vorgestellt wurden. Dargestellt werden folgende Merkmale (die Reihenfolge entspricht den Spalten der Tabellen):

1. Durch Nebenbedingungen definierte zu durchlaufenden Stelle. Es werden entweder die längsten Pfade pro LUT oder pro Slice vom Zeitverhalten-Analyse Werkzeug (TRACE) rückgemeldet. $\Rightarrow A$
2. Anzahl der längsten Pfade, die pro definierte Stelle ausgegeben werden. $\Rightarrow B$
3. Filterung der Verzögerungskanten - es werden entweder alle Verzögerungskanten in der Stichprobe behalten, oder nur diejenigen, die direkt durch eine LUT verlaufen, oder nur 'net'-Kante. Eine Verzögerungskante ist vom Typ 'net', falls sie zwei Komponentenpins miteinander verbindet, ohne dabei eine andere Komponenten zu benutzen. Dies ist bei Verbindungsleitungen der Fall. $\Rightarrow C$
4. LUT Abdeckungsraten. $\Rightarrow D$
5. Abdeckung der Verzögerungskanten, die einen Slice-Eingangspin mit einem Slice-Ausgangspin verbinden. $\Rightarrow E$
6. Abdeckung der 'net'-Kanten. $\Rightarrow F$
7. Redundanz in Form der durchschnittlichen Anzahl von Vorkommnissen einer Verzögerungskanten, in mehreren Pfaden. $\Rightarrow G$
8. Maximale Anzahl von Verzögerungskanten in einem Pfad. $\Rightarrow H$
9. Relative Größe der kodierten (s. Abs. 4.5.3) Stichprobe im unkomprimierten Zustand zur Bitstreamgröße der betrachteten Konfigurationen (in Promille). $\Rightarrow I$
10. Relative Größe der kodierten und komprimierten (s. Abs. 4.5.4) Stichprobe zur Bitstreamgröße der betrachteten Konfigurationen (in Promille). $\Rightarrow J$
11. Worst-Case Zahlen für die Anzahl der benötigten Vergleichs- und Additionsoperationen pro modellierten Hardwarebereich. $\Rightarrow K, L$

Dargestellt werden diese in Tabelle 5.1 und Tabelle 5.2.

Die Ergebnisse zeigen, dass die Speicheranforderungen und Laufzeit des vorgeschlagenen Algorithmus durch geeignete Auswahl und Modifikation der Pfad-Stichprobe flexibel an das ausführende System angepasst werden können. Allerdings sinkt je nach Art der Erzeugung bzw. Modifikation einer Stichprobe auch deren Abdeckungsraten. Wenn auf Grundlage dieser Stichprobe Konfigurationsvorschläge berechnet werden sollen, ist die Wahrscheinlichkeit von falsch-positiven Vorschlägen hoch.

Da als Nebenbedingung für die Zeitverhalten-Analyse die l längsten Pfade pro LUT bzw. Slice ausgegeben werden ist die Abdeckung der Verzögerungskanten von Verbindungsleitungen gering.

A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
LUT	100	Alle	0.88	0.37	0.02	1.44	1.4190	0.8653	7	973	425
LUT	100	Nur LUTS	0.88	0.37	0.00	1.09	0.6108	0.4368	4	399	197
LUT	100	Nur 'net'	0.88	0.00	0.02	1.01	0.5877	0.4810	3	380	194
LUT	10	Alle	0.86	0.31	0.02	1.30	1.1836	0.7347	7	795	373
LUT	10	Nur LUTS	0.86	0.31	0.00	0.98	0.5080	0.3771	4	325	171
LUT	10	Nur 'net'	0.86	0.00	0.02	0.92	0.4972	0.4136	3	316	170
LUT	1	Alle	0.76	0.11	0.00	0.56	0.3059	0.2028	7	174	135
LUT	1	Nur LUTS	0.76	0.11	0.00	0.47	0.1234	0.1116	4	70	53
LUT	1	Nur 'net'	0.76	0.00	0.00	0.44	0.1275	0.1244	3	70	57
SLICE	100	Alle	0.88	0.37	0.02	1.55	1.5244	0.9008	7	1061	439
SLICE	100	Nur LUTS	0.88	0.37	0.00	1.11	0.6129	0.4342	4	401	197
SLICE	100	Nur 'net'	0.88	0.00	0.02	1.08	0.6422	0.5193	3	420	207
SLICE	10	Alle	0.86	0.27	0.00	1.04	0.8303	0.5329	7	533	290
SLICE	10	Nur LUTS	0.86	0.27	0.00	0.77	0.3291	0.2522	4	201	122
SLICE	10	Nur 'net'	0.86	0.00	0.00	0.73	0.3650	0.3162	3	222	136
SLICE	1	Alle	0.49	0.07	0.00	0.29	0.1434	0.1190	7	76	72
SLICE	1	Nur LUTS	0.49	0.07	0.00	0.37	0.0627	0.0689	4	34	30
SLICE	1	Nur 'net'	0.49	0.00	0.00	0.12	0.0627	0.0817	3	31	33

Tabelle 5.1: Eigenschaften unterschiedlich generierter Stichproben des s382 Benchmark-Schaltkreises (jeweils eine Konfiguration).

In der letzten Zeile der Tabelle 5.1 führt die Komprimierung durch den lzo Algorithmus zu einer Stichprobe, die mehr Speicherplatz benötigt. Dies ist jedoch nur ein Unterschied von 70 Bytes und lässt sich auf die Heuristik des Komprimierungsalgorithmus zurückführen. Anhand von Tabelle 5.2 lässt sich sehen, dass mehr diversifizierte Konfigurationen pro Entwurf auch zu größeren Stichproben führen, da sich die Konfigurationen nicht vollständig überlappen und dadurch mehr FPGA Komponenten durch Verzögerungskanten in der Stichprobe abgespeichert werden.

5.5 Diskussion der Ergebnisse

Die Abbildungen dieses Kapitels zeigen eine starke Korrelation zwischen den Platzierungsmöglichkeiten, gegeben durch die Anzahl Konfigurationen pro Entwurf und Anzahl modellierter Hardwarebereiche, und der Adaptierung an Zeitverhalten-Variationen durch den entworfenen Algorithmus. Je mehr Platzierungsmöglichkeiten für einen Entwurf bestehen, desto eher kann er auch bei geringen Zeitpuffer ohne Verzögerungsfehler zu verursachen in der rekonfigurierbare Struktur platziert werden.

Bei den durchgeführten Experimenten wurde eine Stichprobe aller Pfade und Verzögerungskanten genutzt. Dies hat den Vorteil, dass die Laufzeit und die Speicheranforderung des Algorithmus gering

A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
LUT	100	Alle	0.90	0.36	0.02	1.67	1.4693	0.9382	7	10379	3947
LUT	100	Nur LUTS	0.90	0.36	0.00	1.29	0.6133	0.4320	4	4143	1826
LUT	100	Nur 'net'	0.90	0.00	0.02	1.12	0.5947	0.4820	3	3929	1863
LUT	10	Alle	0.90	0.31	0.02	1.42	1.2219	0.8066	7	8402	3516
LUT	10	Nur LUTS	0.90	0.31	0.00	1.11	0.5080	0.3667	4	3336	1608
LUT	10	Nur 'net'	0.90	0.00	0.02	0.96	0.5008	0.4128	3	3222	1657
LUT	1	Alle	0.80	0.09	0.00	0.71	0.3170	0.2005	7	1939	1174
LUT	1	Nur LUTS	0.80	0.09	0.00	0.66	0.1212	0.0862	4	733	452
LUT	1	Nur 'net'	0.80	0.00	0.00	0.46	0.1304	0.1053	3	747	531
SLICE	100	Alle	0.90	0.36	0.02	1.84	1.5757	0.9818	7	11319	4042
SLICE	100	Nur LUTS	0.90	0.36	0.00	1.33	0.6204	0.4388	4	4212	1826
SLICE	100	Nur 'net'	0.90	0.00	0.02	1.25	0.6500	0.5176	3	4373	1957
SLICE	10	Alle	0.90	0.23	0.00	1.31	0.7871	0.5094	7	5329	2356
SLICE	10	Nur LUTS	0.90	0.23	0.00	1.03	0.3001	0.2171	4	1946	977
SLICE	10	Nur 'net'	0.90	0.00	0.00	0.90	0.3358	0.2705	3	2135	1140
SLICE	1	Alle	0.69	0.04	0.00	0.69	0.1176	0.0771	7	714	449
SLICE	1	Nur LUTS	0.69	0.04	0.00	0.72	0.0493	0.0352	4	301	185
SLICE	1	Nur 'net'	0.69	0.00	0.00	0.40	0.0509	0.0432	3	285	218

Tabelle 5.2: Eigenschaften unterschiedlich generierter Stichproben des s382 Benchmark-Schaltkreis (jeweils 10 Konfigurationen).

sind. Allerdings können so Verzögerungsfehler nicht vom Algorithmus erkannt werden, wenn sie durch Verzögerungen hervorgerufen werden, die nicht auf den Pfaden und Verzögerungskanten der Stichprobe liegen. Durch die unterschiedlichen Abdeckungsraten lässt sich jedoch die Güte einer Stichprobe abschätzen.

6 Zusammenfassung und Ausblick

Variationen des Schaltverhaltens in rekonfigurierbaren Hardwarestrukturen können durch Alterungseffekte und zufälligen Variationen verursacht werden. Wenn diese Effekte nicht bei der Rekonfiguration eines rekonfigurierbaren Hardwarebereichs berücksichtigt werden, kann es durch eine Fehlplatzierung eines Entwurfs zu Verzögerungsfehlern kommen. Verzögerungsfehler können die Funktionalität des Systems beeinträchtigen und in sicherheitskritischen Anwendungen zur Gefährdung von Personen führen.

Um Verzögerungsfehler durch eine optimistische Platzierung von Entwürfen zu vermeiden und dadurch an Variationen des Zeitverhaltens zu adaptieren, wurde in dieser Arbeit ein Algorithmus entworfen und untersucht. Der vorgeschlagene Algorithmus analysiert das Zeitverhalten von Konfigurationen der Hardwareentwürfe, um das Zeitverhalten zu bestimmen, das eine Konfiguration bei der Platzierung in einem Hardwarebereich von diesem erwarten würde. Durch das erwartete Zeitverhalten stellt eine Konfiguration Zeitverhalten-Bedingungen an einen rekonfigurierbaren Hardwarebereich. Anschließend kann das aktuelle Zeitverhalten der rekonfigurierbaren Hardwarebereiche in den vorgeschlagenen Algorithmus geladen werden. Durch einen Vergleich von aktuellem, möglicherweise durch Alterungseffekte beeinträchtigten, Zeitverhalten eines Hardwarebereichs mit den Zeitverhalten-Bedingungen der Konfigurationen kann der vorgeschlagene Algorithmus für jeden Hardwarebereich bestimmen, welche Konfigurationen bzw. Entwürfe darin platziert werden können, ohne dass Verzögerungsfehler den Betrieb beeinträchtigen. Diese Information kann schließlich von einem Platzierungsalgorithmus genutzt werden, um Verzögerungsfehler durch geeignete Platzierung von Konfigurationen zu minimieren.

Die durchgeführten Experimente zeigen, dass der vorgeschlagene Algorithmus an Variationen des Schaltverhaltens adaptieren kann, wenn mehrere Konfigurationen pro Entwurf existieren oder mehrere Hardwarebereiche für die Platzierung genutzt werden können. Der vorgeschlagene Algorithmus ist am effektivsten, wenn für einen Entwurf mehrere diversifizierte Konfigurationen synthetisiert wurden und mehrere rekonfigurierbare Hardwarebereiche zur Platzierung eines Entwurfs zur Verfügung stehen. Dabei können die Speicherplatzanforderung und die Laufzeit des vorgeschlagenen Algorithmus flexibel an das ausführende System angepasst werden.

Ausblick

Der Algorithmus besitzt noch Erweiterungspotenzial. Beispielsweise kann ein eigenes Werkzeug zur Analyse des Zeitverhaltens entwickelt werden, um die Analyse genauer steuern zu können und auch große Schaltungen zu unterstützen. Das momentan genutzte kommerzielle Werkzeug kommt selbst nach mehreren Stunden bei größeren Schaltungen zu keinem Ergebnis.

Da der Anteil der Verzögerungen durch Verbindungsleitungen an der Gesamtverzögerung einer Schaltung bei kleiner werdenden Strukturgrößen steigt, ist es sinnvoll weitere Methoden zu entwickeln, die die Abdeckung der geprüften Verbindungsleitungen durch den vorgeschlagenen Algorithmus erhöht. Zurzeit wird nur ein kleiner Teil der Verbindungsleitungen auf Verzögerungsfehler überprüft.

Dennoch kann der vorgeschlagene Algorithmus bereits genutzt werden, um in rekonfigurierbaren Hardwarestrukturen an Zeitverhalten-Variationen zu adaptieren. Die erarbeiteten Ergebnisse sind z.B. in einem rekonfigurierbaren System mit mehreren Hardwarebeschleunigern, die dynamisch in einen Hardwarebereich konfiguriert werden, einsatzfähig. Hierbei kann der entworfene Algorithmus die Platzierung der Hardwarebeschleuniger steuern, um Systemausfälle durch Verzögerungsfehler aufgrund von Fehlplatzierungen selbst bei anhaltendem Betrieb und gealterten Komponenten zu verringern.

Literaturverzeichnis

[Alb05] C. Albrecht. IWLS 2005 Benchmarks. 2005. (Zitiert auf Seite 45)

[BA06] K. C. Barr, K. Asanović. Energy-aware Lossless Data Compression. *ACM Trans. Comput. Syst.*, 24(3):250–291, 2006. (Zitiert auf Seite 43)

[BBK89] F. Brglez, D. Bryan, K. Kozminski. Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems*, S. 1929–1934. 1989. (Zitiert auf Seite 45)

[BC09] J. Bhasker, R. Chadha. *Static Timing Analysis for Nanometer Designs: A Practical Approach.* Springer US, 2009. (Zitiert auf den Seiten 30 und 31)

[BGHK96] A. Blecker, M. Goedecke, S. Huss, K. Kran. *Praktikum des modernen VLSI-Entwurfs: Eine Einführung in die Entwurfsprinzipien und -beschreibungen, unter besonderer Berücksichtigung von VHDL; mit einer umfangreichen Anleitung zum Praktikum.* Vieweg+Teubner Verlag, 1996. (Zitiert auf Seite 26)

[Bry85] D. Bryan. The ISCAS'85 benchmark circuits and netlist format. *North Carolina State University*, 1985. (Zitiert auf den Seiten 25 und 45)

[BSV10] N. Battezzati, L. Sterpone, M. Violante. *Reconfigurable field programmable gate arrays for mission-critical applications.* Springer Science & Business Media, 2010. (Zitiert auf Seite 11)

[CIR87] J. Carter, V. Iyengar, B. Rosen. Efficient test coverage determination for delay faults. In *Proceedings of IEEE International Test Conference*, S. 418–427. 1987. (Zitiert auf Seite 27)

[Dav99] S. Davidson. ITC'99 benchmark circuits-preliminary results. In *Proceedings of IEEE International Test Conference*, S. 1125–1125. 1999. (Zitiert auf Seite 45)

[Deu13] Deutsche Bahn. NetzNachrichten. Newsletter, 2013. (Zitiert auf Seite 9)

[Dub08] R. Dubey. *Introduction to Embedded System Design Using Field Programmable Gate Arrays.* Springer London, 2008. (Zitiert auf Seite 14)

[Hof07] D. W. Hoffmann. *Grundlagen der technischen Informatik: mit 57 Tabellen und 90 Aufgaben.* Hanser Verlag, 2007. (Zitiert auf Seite 25)

[HSC+82] R. Hitchcock, G. L. Smith, D. D. Cheng, et al. Timing analysis of computer hardware. *IBM journal of Research and Development*, 26(1):100–105, 1982. (Zitiert auf Seite 24)

[KBT08] D. Koch, C. Beckhoff, J. Teich. ReCoBus-Builder: A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAS. In *Proceedings of International Conference on Field Programmable Logic and Applications (FPL'08)*, S. 119–124. 2008. (Zitiert auf Seite 20)

[KC98] A. Krstic, K.-T. Cheng. *Delay fault testing for VLSI circuits*, Band 14. Springer Science & Business Media, 1998. (Zitiert auf Seite 27)

[KKKO06] K. Katsuki, M. Kotani, K. Kobayashi, H. Onodera. Measurement Results of Within-die Variations on a 90Nm LUT Array for Speed and Yield Enhancement of Reconfigurable Devices. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, ASP-DAC '06, S. 110–111. 2006. (Zitiert auf Seite 31)

[Koc13] D. Koch. *Partial Reconfiguration on FPGAs - Architectures, Tools and Applications*, Band 153 von *Lecture Notes in Electrical Engineering*. Springer, 2013. (Zitiert auf den Seiten 11, 18, 19, 20 und 21)

[LWLL04] X.-Y. Li, F. Wang, T. La, Z.-M. Ling. FPGA as Process Monitor-an effective method to characterize poly gate CD variation and its impact on product performance and yield. *IEEE Transactions on Semiconductor Manufacturing*, 17(3):267–272, 2004. (Zitiert auf den Seiten 31 und 38)

[MA98] A. Majhi, V. Agrawal. Delay fault models and coverage. In *Proceedings of Eleventh International Conference on VLSI Design*, S. 364–369. 1998. (Zitiert auf Seite 27)

[Mas04] J. Massey. NBTI: what we know and what we need to know - a tutorial addressing the current understanding and challenges for the future. In *Proceedings of IEEE International Integrated Reliability Workshop Final Report*, S. 199–211. 2004. (Zitiert auf Seite 29)

[Mat09] V. Matrose. *Optimierung der Verdrahtbarkeit unter Berücksichtigung heterogener Verdrahtungsressourcen hierarchischer FPGA-Architekturen*. Dissertation, 2009. (Zitiert auf Seite 12)

[MBMB07] U. Meyer-Baese, U. Meyer-Baese. *Digital signal processing with field programmable gate arrays*, Band 65. Springer, 2007. (Zitiert auf den Seiten 11 und 16)

[McP06] J. W. McPherson. Reliability Challenges for 45Nm and Beyond. In *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, S. 176–181. 2006. (Zitiert auf Seite 28)

[MFJ65] I. Miller, J. E. Freund, R. A. Johnson. *Probability and statistics for engineers*, Band 1110. Prentice-Hall Englewood Cliffs, NJ, 1965. (Zitiert auf Seite 46)

[OTE] OTERA. http://www.iti.uni-stuttgart.de/abteilungen/rechnerarchitektur/projekte/otera.html. Letzter Zugriff: 20.04.2015. (Zitiert auf den Seiten 12 und 45)

[SC06] P. Sedcole, P. Cheung. Within-die delay variability in 90nm FPGAs and beyond. In *Proceedings of IEEE International Conference on Field Programmable Technology (FPT'06)*, S. 97–104. 2006. (Zitiert auf Seite 31)

[SHB11] C. Schuck, B. Haetzer, J. Becker. Reconfiguration Techniques for Self-X Power and Performance Management on Xilinx Virtex-II/Virtex-II-Pro FPGAs. *Int. J. Reconfig. Comp.*, 2011, 2011. (Zitiert auf Seite 46)

[SKM+08] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M. J. Irwin, K. Sarpatwari. Toward Increasing FPGA Lifetime. *IEEE Transactions on Dependable and Secure Computing*, 5(2):115–127, 2008. (Zitiert auf Seite 28)

[Smi85] G. L. Smith. Model for Delay Faults Based upon Paths. In *ITC*, S. 342–351. 1985. (Zitiert auf Seite 27)

[WLRI87] J. Waicukauski, E. Lindbloom, B. K. Rosen, V. Iyengar. Transition Fault Simulation. *IEEE Design Test of Computers*, 4(2):32–38, 1987. (Zitiert auf Seite 27)

[WM05] K. Wu, J. Madsen. Run-time dynamic reconfiguration: a reality check based on FPGA architectures from Xilinx. In *Proceedings of 23rd IEEE NORCHIP Conference*, S. 192–195. 2005. (Zitiert auf Seite 21)

[Wun13] H.-J. Wunderlich. *Hochintegrierte Schaltungen: Prüfgerechter Entwurf und Test.* Springer-Verlag, 2013. (Zitiert auf den Seiten 24 und 46)

[WWW06] L.-T. Wang, C.-W. Wu, X. Wen. *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon).* Morgan Kaufmann Publishers Inc., 2006. (Zitiert auf Seite 27)

[Xil12] Xilinx. *Virtex-5 FPGA User Guide*, 2012. Version 5.4. (Zitiert auf Seite 15)

[Xil13a] Xilinx. *Command Line Tools User Guide*, 2013. Version 14.7. (Zitiert auf den Seiten 16 und 37)

[Xil13b] Xilinx. *Partial Reconfiguration User Guide*, 2013. Version 14.5. (Zitiert auf den Seiten 17, 18, 19, 20 und 21)

[ZBK+13] H. Zhang, L. Bauer, M. Kochte, E. Schneider, C. Braun, M. Imhof, H.-J. Wunderlich, J. Henkel. Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures. In *Proceedings of IEEE International Test Conference (ITC'13)*, S. 1–10. 2013. (Zitiert auf den Seiten 29, 33 und 45)

[ZG05] A. Zeineddini, K. Gaj. Secure partial reconfiguration of FPGAs. In *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT'05)*, S. 155–162. 2005. (Zitiert auf Seite 21)

Alle URLs wurden zuletzt am 20. April 2015 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift