Fehlerinjektion für rekonfigurierbare Hardware

Bernhard Fechner
FernUniversität in Hagen
Lehrgebiet VLSI und Parallelität
Postfach 940
58084 Hagen, Germany
bernhard.fechner@fernuni-hagen.de

Abstract

In der vorliegenden Arbeit wird die Architektur und Funktionsweise eines Fehlerinjektionssystems mit Hilfe von FPGA-Bitfiles beschrieben. In Kapitel 1 wird erklärt, warum Fehler in komplexe Systeme injiziert werden müssen. In Kapitel 2 wird zunächst die typische Architektur eines FPGAs (Field-Programmable Gate Array) beschrieben. Danach wird die Menge der in Bitfiles injizierbaren Fehler identifiziert und gezeigt, wie diese eingebracht werden können. Weiterhin wird eine neue Methode zur komponentengenauen Injektion von Fehlern vorgestellt. Fehlerinjektionsläufe werden durch einen Optimierer beschleunigt, soweit die Aussagekraft der interessierenden Experimentergebnisse hierdurch nicht gefährdet ist. In Kapitel 3 folgt die Schlussfolgerung mit einer Zusammenfassung der Vor- und Nachteile des geschilderten Verfahrens.

1 Einleitung

Der Einsatz von Fehlertoleranzverfahren ist nur dann zu vertreten, wenn die Korrektheit des Verfahrens nachgewiesen werden kann und in sicherheitskritischen Bereichen die Korrektheit der gegen mögliche Fehler ergriffenen Maßnahmen gegenüber Genehmigungsinstanzen nachweisbar ist.
Hierzu gibt es zwei Ansätze:

- den praktischen Ansatz durch Fehlerinjektionsexperimente und
- die theoretische Analyse mit formalen Methoden wie z.B. Modellierung mit Markov-Ketten und/oder Petri-Netzen.

Theoretische Analysen haben den Vorteil, dass Aussagen über Ausfallwahrscheinlichkeit, etc. schon vor Realisierung eines Produktes getroffen werden können. Leider sind Modellierungen komplexer Systeme, wie z.B. eines Mikroprozessors mit Petri-Netzen, die sich wiederum in Markov-Ketten überführen lassen bei akkurater Modellierung aufgrund der dann explosionsartig wachsenden Anzahl von Zuständen sehr berechnungsintensiv, unübersichtlich und damit anfällig für Fehler. Daher wird in dieser Arbeit ein Ansatz unter Anwendung einer geeigneten *Fehlerinjektionstechnik* verfolgt. Allerdings ist eine Verifikation durch eine vollständige Überprüfung aufgrund der hohen Komplexität der untersuchten Systeme nicht immer möglich. Einführungen zum Thema Fehlerinjektion bieten [1][2][3][4]. In der Praxis wird die Fehlerinjektion meist aus Kostengründen entweder nur teilweise oder gar nicht durchgeführt. Stattdessen wird viel Mühe auf die korrekte Spezifikation der Fehlererkennungsverfahren gelegt und daraus abgeleitet, dass die geforderten Funktionen fehlerfrei erbracht werden können. Eine Restfehlerwahrscheinlichkeit bleibt in jedem Fall erhalten.
Bei der Durchführung von Tests für ein aus Soft- und Hardwarekomponenten bestehendes System können drei wesentliche Zwecke identifiziert werden:

1. **Verifikation und Validierung**
   Verifikation deckt Abweichungen zwischen der spezifizierten und der tatsächlich realisierten Systemfunktion auf. Validierung bezeichnet Methoden zur Aufdeckung von Diskrepanzen zwischen dem realisierten System und existierenden Systemanforderungen.
2. **Erhöhen des dem System entgegengebrachten Vertrauens**
   Im Rahmen von Zertifizierungsprozessen werden sicherheitskritische Systeme mittels Fehlerinjektion getestet. Die Durchführung umfangreicher Fehlerinjektionsexperimente kann eine wesentliche Zertifizierungsanforderung sein.
3. **Bestimmung von Kenngrößen**
   Fehlerinjektionsexperimente werden eingesetzt, um zu einem System, dessen Einsatz geplant ist, eine Bewertung zu erhalten, da bestimmte Kenngrößen komplexer Hard- oder Softwaresysteme in der Praxis schwer zu analysieren und zu erhalten sind. Solche Kenngrößen (z.B. Reaktionszeiten) lassen sich durch Tests experimentell bestimmen. Fehlertolerante Rechensysteme können bei Erkennung eines tolerierbaren Fehlers in einen Wiederherstellungsmodus (Recovery) übergehen, der es erfordert, das Rechensystem auf einen konsistenten Zustand aus der Vergangenheit zurückzusetzen.

2 Fehlerinjektion in FPGA Bitfiles

Bei der Bewertung eines fehlertoleranten Systems mittels Fehlerinjektion ergibt sich die Notwendigkeit der Auswahl einer geeigneten Menge zu injizierender Fehler, die eine Untermenge der injizierbaren Fehler darstellt. Die einzubringenden Fehler werden vorher in einem Fehlermodell spezifiziert und abstrahiert, da die Fehlerinjektion nicht immer in der Lage ist, die genauen Auswirkungen eines physikalischen Fehlers nachzubilden. Die Festlegung einer großen Anzahl zu injizierender Fehler führt zu zeitaufwendigen Fehlerinjektionsexperimenten. Alternativ kann eine Injektion pessimistischer Fehler stattfinden. Dabei wird der Anteil harmloser Fehlerinjektionen innerhalb des Fehlerinjektionsexperiments reduziert. Harmlose Fehlerinjektionen sind z.B. Fehler, die keine erkennbare Auswirkung auf das zu analysierende System haben oder derart schwerwiegend und dadurch unwahrscheinlich sind, dass sie beispielsweise stets leicht erkennbare Crash-Fehler verursachen. Bestimmten Fehlern kann man so einen *Härtegrad* zuordnen. Wird bei der Injektion eines Fehlers *F* in ein Bitfile *B* unter beispielsweise keine Reaktion beobachtet, so ist die Injektion *zu weich*, da sie keinerlei Wirkung zeigt. Weiterhin kann bei unwahrscheinlichen Fehlermengen, bzw. Fehleranzahlen, die Injektionen *zu hart* werden. Die zu harten bzw. zu weichen Injektionen werden als *harmlose Injektionen* bezeichnet.
Die Durchführung pessimistischer Fehlerinjektionsexperimente kann aus den folgenden Gründen als sinnvoll angesehen werden:

- Reduktion des Zeitaufwands: eine geringere Anzahl von Fehlern wird injiziert, ohne die Anzahl der Fehler zu verringern, denen ein grundsätzliches Potential zur Systemüberlistung zugesprochen werden kann.
- Produktion konservativer Bewertungsergebnisse: Weist ein System auch bei der Bewertung durch eine pessimistische Fehlerinjektion eine gute Fehlererfassung auf, so kann dies das Vertrauen erhöhen, das der Fehlererfassungsfähigkeit des Systems entgegengebracht wird.

Soll ein Mikroprozessor durch Injektion einer realistischen Fehlermenge bewertet werden, so ergibt sich bei der Auswahl einer solchen Fehlermenge eine Reihe schwieriger Fragen:

- Welche Fehler sind beim Betrieb eines Mikroprozessors unter welchen Umgebungsbedingungen (z.B. Temperatur, Strahlung, Spannungsschwankungen) zu erwarten?
- Welche Fehler lassen sich mit der Fehlerinjektion modellieren?
- Mit welchen Wahrscheinlichkeiten treten diese Fehler auf?
- Welche Fehlerdauern sind jeweils zu erwarten?
- Mit welcher Wahrscheinlichkeit wiederholt sich ein Fehler, der bei einem betrachteten Mikroprozessor bereits beobachtet wurde?

Das hier vorgestellte System zur Fehlerinjektion injiziert Fehler durch Manipulation des für die Programmierung eines FPGAs erforderlichen Bitfiles. Dabei kann eine partielle oder eine totale Rekonfiguration vorgenommen werden. Bei einer partiellen Rekonfiguration wird nur ein bestimmter Bereich des FPGAs neu programmiert. Im Gegensatz dazu steht die totale Rekonfiguration bei der alle konfigurierbaren Komponenten neu programmiert werden. Das System kann Stuck-At Fehler und transiente Fehler in der Form von Single Event Upsets (SEUs) in das FPGA und den PCI-Adressraum einbringen, bzw. simulieren. Bei dieser Art der Fehlerinjektion handelt es sich also um eine hybride Form von soft- und hardwarebasierter Fehlerinjektion. Je nach Möglichkeiten des verwendeten Fehlerinjektors kommen ggf. weitere Fehlerarten, wie z.B. die Auswirkungen interner Fehler des Rechenwerkes oder auch Fehler in der Befehlsdekodierung hinzu. Meist werden alle Fehlerarten injiziert, die durch das eingesetzte Injektionswerkzeug injizierbar sind.

2.1 Zugriffspunkte für Bitfile-Fehlerinjektion

Um festzustellen, welche Fehlertypen in rekonfigurierbare Hardware injizierbar sind, müssen Zugriffspunkte für die Fehlerinjektion ermittelt werden. Dazu ist es notwendig, sich mit der Architektur der eingesetzten Hardware auseinanderzusetzen. Dies soll am Beispiel des Xilinx Virtex-E XCV1000 FPGAs geschehen. Abbildung 1 zeigt den schematischen Aufbau eines SRAM-basierten FPGAs. Der Schaltkreis enthält eine Matrix aus gleichförmig angeordneten konfigurierbaren Logikblöcken (CLB). Diese sind von programmierbaren Eingangs-/Ausgangsblöcken (I/O-Blöcke) umgeben. Diese Blöcke werden durch unterschiedliche Verdrahtungsressourcen miteinander verbunden, die sich an den Kreuzungspunkten zwischen CLBs/CLBs und IOBs befinden. Die logischen Funktionen der CLBs, die Zustände der IOBs (Eingang, Ausgang oder hochohmig) und die Verbindungsstruktur dieser Blöcke untereinander, wird durch ein Konfigurationsprogramm festgelegt. Die Konfigurationsdaten werden als *Bitstream* in das FPGA geladen und in den internen SRAMs gespeichert. Der Bitstream ist in einem Bitfile gespeichert. Zur Quantifizierung der Komplexität eines FPGAs wird üblicherweise die Maßeinheit Gatter oder auch Gatteräquivalent benutzt. Ein Gatter entspricht einem NAND-Gatter mit zwei Eingängen.

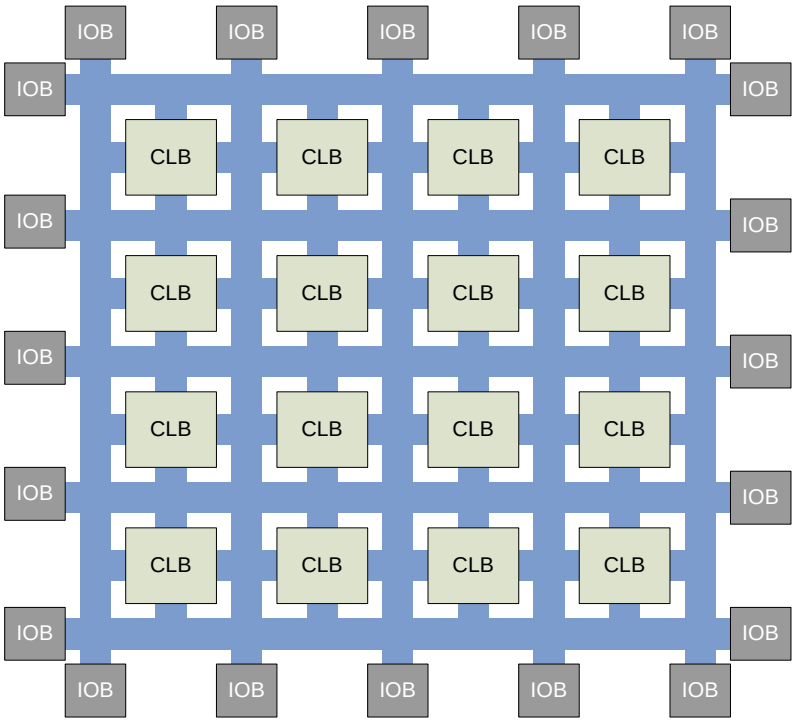


Abbildung 1. Architektur eines FPGAs

Zum Speichern von größeren Datenmengen sind außerdem in ihrer Adress- und Datenbreite programmierbare Speicher BRAMs (BlockRAMs) vorhanden. Weiterhin enthält der Schaltkreis Global Clock Buffer (GBUF), die die Taktnetzwerke puffern. Außerdem sind mehrere Delayed Locked Loops (DLL) enthalten, die die Verzögerungen bei der Taktverteilung auf dem Chip ausgleichen und auch zur Taktteilung benutzt werden können.
Konfigurierbare Strukturen sind:

- **Konfigurierbare Logikblöcke (CLBs)**
  Eine quadratische Matrix von CLBs bildet das Kernstück des FPGAs. Die CLBs beinhalten programmierbare Logikressourcen und besitzen herstellerübergreifend einen typischen Aufbau, bestehend aus kombinatorischen Logikeinheiten, Multiplexern und Registern. Die CLBs der Virtex-Serie beinhalten zwei Lookup-Tabellen mit jeweils vier Eingängen. Außerdem ist eine Verwendung der Lookup-Tabellen (LUT) als kleines synchrones RAM oder ROM-Block möglich. Abbildung 2 zeigt den Aufbau eines Virtex-E CLBs.

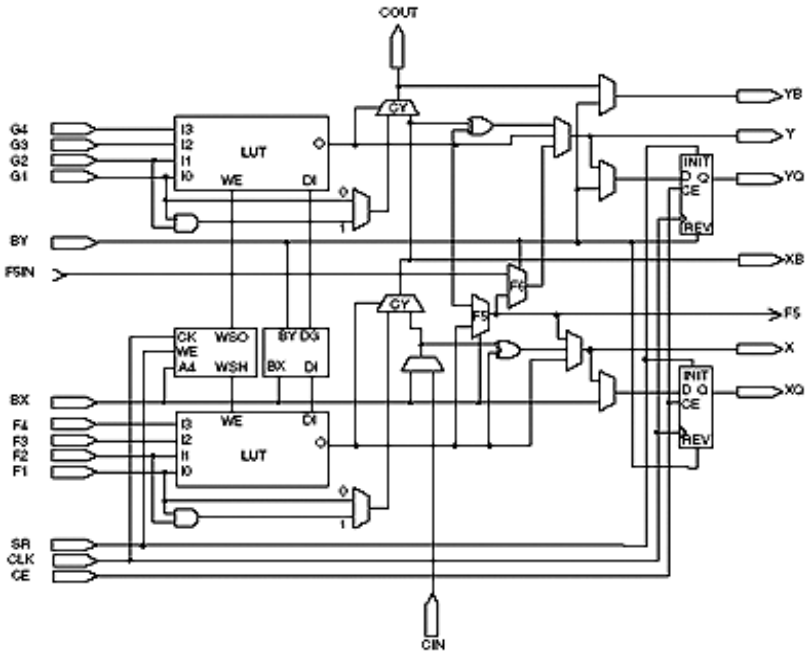


Abbildung 2. Der CLB eines Xilinx Virtex-E XCV1000

Bei entsprechender Konfiguration ist es möglich, die Logikzelle als Verdrahtungsressource zu nutzen. Es lässt sich dafür nur ein Ausgang benutzen, wahlweise der direkte oder der durch das Register gepufferte Ausgang der LUT. Es besteht auch die Möglichkeit, die LUTs zu umgehen und die Eingänge direkt den beiden jeweils als Latch oder FF konfigurierbaren Registern zuzuführen.

- **I/O-Blocks (IB)**
  Die Anbindung zum Systembus wird über die IOBs realisiert. Diese können als Ein-, Ausgang oder als bidirektionaler (Tri-State) Port konfiguriert werden. Prinzipiell sind I/O-Blöcke verschiedener FPGA-Typen ähnlich strukturiert. Unterschiede bestehen in den elektrischen Eigenschaften, wie zum Beispiel der maximalen Stromaufnahme oder in der Anzahl der Speicherelemente. Xilinx Virtex IOBs sind überaus komplex. Der Einfachheit halber wurde in Tabelle 1 nur ein Teil der IOB-Architektur dargestellt. Zur Signalpufferung befinden sich in IOBs drei FFs. Wenn die IOB-Konfiguration im Bitfile verändert wird, können IOB-Modus, I/O Logikpegel, etc. verändert werden. Timing-Fehler können durch verändern der Taktpolarität, schwere Systembus-Fehler können durch Aktivierung des Tri-State Puffers eines als Eingabeport konfigurierten IOBs induziert werden.

- **Flexibles Verbindungsnetzwerk**
  Für die Verdrahtung von IOBs und CLBs steht ein flexibles Verbindungsnetzwerk zur Verfügung. Es besteht aus einem Gitter von Verbindungskanälen, die zwischen den CLBs angeordnet sind. Kanäle können über Multiplexer mit anderen Kanälen, IOBs oder CLBs verbunden werden. Die Kreuzungspunkte dieser Kanäle, an denen Schalterelemente sitzen, werden bei Xilinx-FPGAs *Switchmatrix* genannt. Bei Xilinx-FPGAs gibt es drei Verbindungsebenen. Dazu gehört die Verdrahtung von Switch-Matrizen (single length lines), Verdrahtung, die eine Matrix überspringt (double length lines) und Verdrahtung, die über den ganzen Chip geführt wird (long lines).

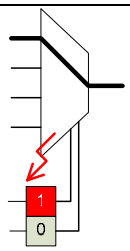
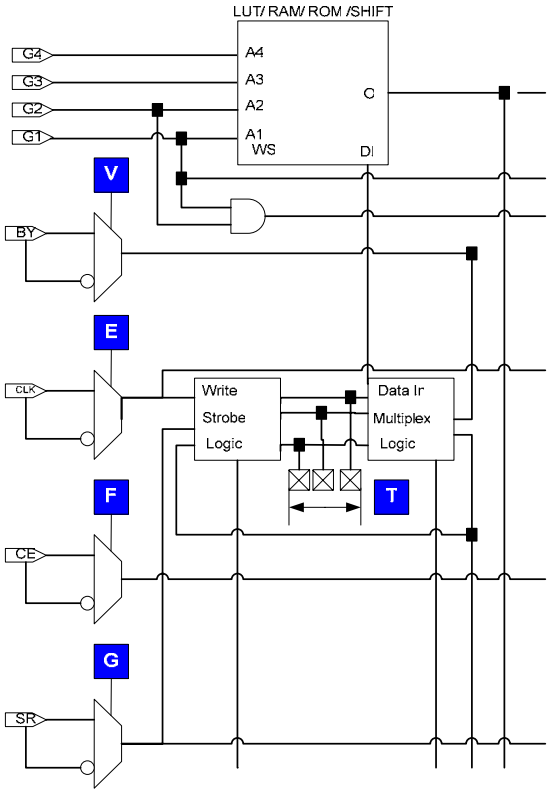
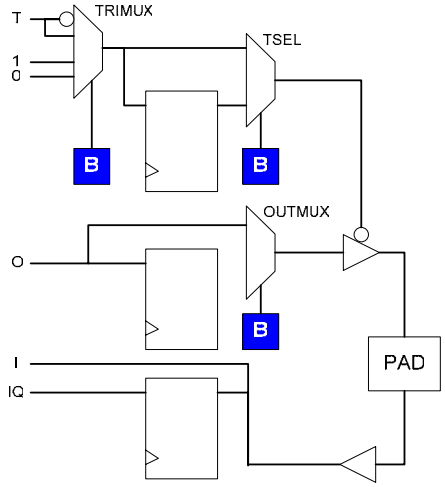
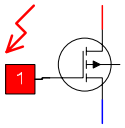
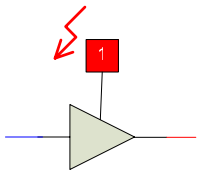
Mögliche Zugriffspunkte für eine Fehlerinjektion stellen alle auf dem Chip veränderbaren Strukturen dar. Dazu zählen auch Flip-Flops/ Latches innerhalb der CLBs. Da wir eine Fehlerinjektion über das Bitfile vornehmen, können wir diese darüber nicht verändern. Bei entsprechendem Aufwand können aber SEUs (Single Event Upsets) und permanente SA (Stuck At 0 oder 1) Fehler injiziert werden. Um permanente Fehler einzubringen, kann man z.B. das Write-Enable Signal deaktivieren. Konfigurierbare Strukturen stellen Look-Up-Tabellen, Switch-Boxen und die Konfigurationsregister dar.

Tabelle 1 zeigt eine Übersicht der über das Bitfile rekonfigurierbaren Komponenten. Über die Bits V, E, F, G können die Eingänge BY, CLK, CE und SR invertiert werden. Wenn bei einer Programmierung diese Bits verändert werden, so führt dies bei erneuter Rekonfiguration zur Simulation eines SEUs auf der Verdrahtung. Wird diese nicht durchgeführt, so wird ein permanenter SA-Fehlers innerhalb des CLBs simuliert, da Konfigurationsbits nur durch eine Rekonfiguration überschrieben werden können; die auf dem FPGA implementierte Logik ist dazu nicht in der Lage. Die T Bits geben u.a. an, ob der LUT als RAM, ROM oder Shift-Register konfiguriert ist. Mit ihnen lassen sich permanente Fehler in LUTs simulieren. Die Injektion permanenter Fehler ist oft sehr zeitaufwendig, weil die Fehlerwirkungen während des Injektionslaufes aufrechterhalten werden müssen. Es bestehen hierzu folgende Möglichkeiten:

- Verändern der FPGA-Konfiguration innerhalb des Bitfiles und ein Aufspielen einer neuen Konfiguration nach jedem Takt.
- Signalisieren jedes Schreibzugriffes innerhalb des Designs nach Außen. Dazu ist eine Änderung des implementierten Entwurfs notwendig, da ein interner Schreibzugriff des Prozessors nach Außen mitgeteilt werden muss. Die Möglichkeit führt zur effizientesten Realisierung, wenn permanente Fehler injiziert werden, da nur dann eine Rekonfiguration erfolgt, wenn es unbedingt notwenig ist. Für die Praxis ist dieser Ansatz allerdings nicht zu empfehlen, da der Entwurf verändert werden muss. Wenn der Quellcode der Architektur nicht verfügbar ist, so kann eine Änderung nicht mehr durchgeführt werden. Wenn bestimmte Beschränkungen (*constraints*) in Bezug auf Platz und Zeit eingehalten werden müssen, kann eine Veränderung der Implementierung diese constraints außer Kraft setzen.
- LUTs als ROM konfigurieren.
- Die Funktion eines LUT so verändern, dass dieser nur noch ‚0' oder ‚1' ausgibt.

Temporäre Fehler werden ganz analog zu permanenten injiziert, nur mit dem Unterschied, dass bei einer erneuten Rekonfiguration eine fehlerfreie, bzw. eine Konfiguration mit permanenten Fehlern auf das FPGA geladen wird.

Tabelle 1. Mögliche Bitfile-Zugriffspunkte

<table>
<tr><th>LUT</th><th>Routing Netzwerk</th><th>Slice im CLB [5]</th></tr>
<tr><td>
<table>
<tr><td colspan="5">O=F1*F2*F3*F4</td></tr>
<tr><td>F1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>F2</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>F3</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>F4</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
</table>
</td><td> 
</td><td rowspan="4"> 
</td></tr>
<tr><td>IOB</td><td>Routing MUX</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td></td><td>Programmable Interconnect Point (PIP)</td></tr>
<tr><td></td><td> 
</td></tr>
<tr><td></td><td>Buffer</td></tr>
</table>

2.2 Funktionsweise des Injektor-Moduls

Wie in Abbildung 3 dargestellt, sind der Fehlerinjektor für den FPGA- und PCI-Adressraum Programmmodule innerhalb des Fehlerinjektionssystems. Um ein Fehlerinjektionsexperiment in Abhängigkeit von einem vorher spezifizierten Fehlerszenario (z.B. missionsspezifische Parameter, die in vorher durchgeführten Missionen gesammelt wurden) zu starten, kann die Fehlerinjektion automatisch durchgeführt werden. Dabei wird die Spezifikation der Fehlerinjektion aus einer separaten Datei gelesen. Tabelle 2 zeigt eine Übersicht der Konfigurationsparameter, wobei die Möglichkeit der Injektion von SEUs/ SA-Fehlern in FFs und bestimmte Adressbereiche des PCI-Adressraumes des Hostsystems der Vollständigkeit halber mit aufgenommen wurde.

Tabelle 2. Daten in der Spezifikation für die Fehlerinjektion

Name	Bezeichnung	Bereich
PCI_EP	Globale Wahrscheinlichkeit für einen SEU im PCI-Adressraum	IR∈[0..1]
pPCI_EP	Globale Wahrscheinlichkeit für einen permanenten Fehler im PCI-Adressraum	IR∈[0..1]
FPGA_EP	Globale Wahrscheinlichkeit für einen SEU im FPGA	IR∈ [0..1]
pFPGA_EP	Globale Wahrscheinlichkeit für einen permanenten Fehler im FPGA	IR∈ [0..1]
RAM_EP	Wahrscheinlichkeit für einen SEU-Fehler im SRAM (Konf. als RAM).	IR∈ [0..1]
ROM_EP	Wahrscheinlichkeit für einen permanenten Fehler im SRAM (Konf. als ROM).	IR∈ [0..1]
sFF_EP	Globale Wahrscheinlichkeit im FPGA für SEUs in FFs innerhalb FPGA_EP	IR∈ [0..1]
pFF_EP	Globale Wahrscheinlichkeit im FPGA für permanente Fehler in FFs.	IR∈ [0..1]
LOC	Fehlerort P=PCI-Adressraum, F=FPGA	F|P
Lxxxxxxxx	Adresse des Fehlers in Abhängigkeit von LOC	$0..2^{32}$-1
Zxxxxxxxx	Zeitangabe in Takten	$0..2^{32}$-1
Ixx	xx∈[0x1,0xFE]: Transienter, xx=0xFF: Permanenter Fehler	1..255

Abbildung 3 zeigt das Zusammenspiel der einzelnen Module des Fehlerinjektionssystems. Manuelle Fehlerinjektion ist nur nützlich, um die Funktionsweise des Fehlerinjektors zu testen.

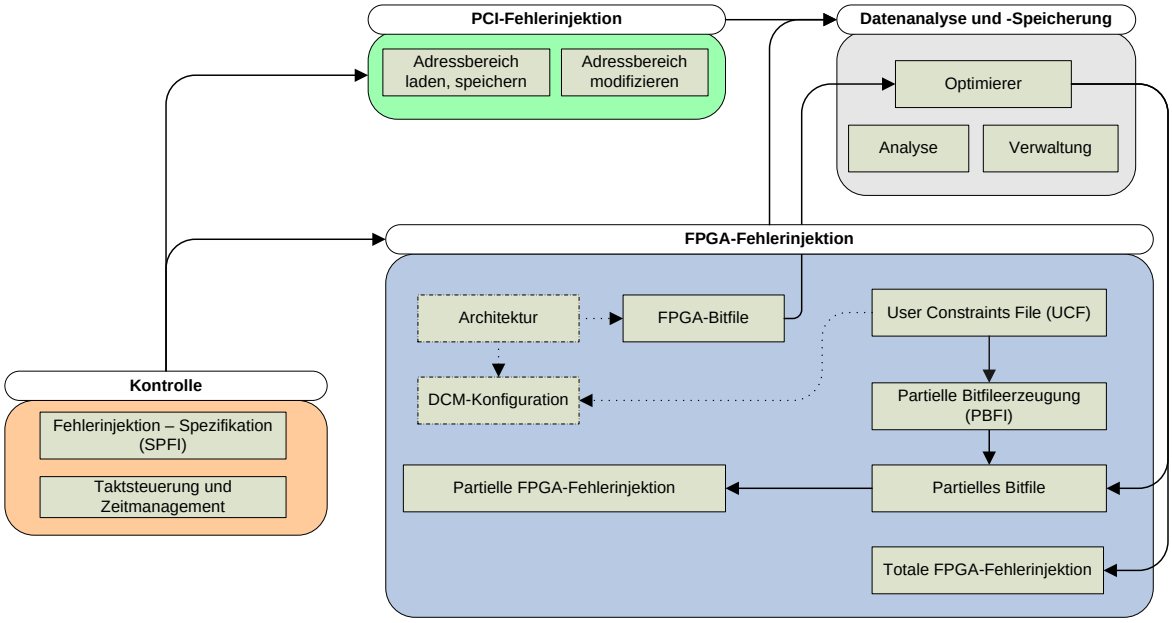


Abbildung 3. Architektur des Fehlerinjektionssystems

In einem zweiten Schritt werden Referenzergebnisse in einem *Golden Run* produziert. Dabei wird die zu untersuchende, als fehlerfrei angenommende Bitfile-Konfiguration B_1 wird auf das FPGA geladen. Das Programm *P* wird für alle vom Benutzer festgelegten Testeingaben E ausgeführt, ohne dass Fehler injiziert werden. Die auf diese Weise erzielten Referenzergebnisse werden zu folgenden Zwecken genutzt: Wird ein Fehler *F* in das System injiziert, so wird das Bitfile verändert. Die entstehende Bitfile-Konfiguration wird B_i; i={2,..,n}; n∈IN genannt. Optional kann über eine Architektur-Datei die Zusammensetzung des verwendeten Bitfile geladen werden, so dass die FPGA-Fehlerinjektion typunabhängig wird. Anhand der Referenzergebnisse lässt sich überprüfen, ob die von B_i unter Fehlereinwirkung produzierten Ergebnisse mit der gleichen Eingabe E korrekt oder fehlerhaft sind. Zusätzlich wird die Zeit *z* gemessen, die von B_1 zur Produktion des einer Eingabe *E* zugeordneten Referenzergebnisses benötigt wird. Der Wert *z* wird verwendet, um Zeitschranken für die Ausführung einer Fehlerinjektionsvariante B_i zu ermitteln. Eine solche Zeitschranke ergibt sich durch Multiplikation von *z* mit einem vom Benutzer einstellbaren Faktor. Eine auf diese Weise festgelegte Zeitschranke berücksichtigt die evtl. von *E* abhängige Programmlaufzeit wie auch die sich durch die Fehlerinjektion ergebenden Verzögerungen. Ein zu frühzeitiger Ablauf eines Timeouts kann nicht ausgeschlossen werden, da unter Fehlerinjektion beliebige Verfälschungen des Kontrollflusses denkbar sind.
Beim Entwurf eines Injektionswerkzeugs ist zu berücksichtigen, dass die Ausführung einer Fehlerinjektionsvariante B_i einen fehlerbedingten Programmabsturz zur Folge haben kann. Da der Fehlerinjektor auf dem PCI-Host und das zu bewertende Programm auf dem Entwicklungsboard mit unabhängigen Prozessoren ausgeführt werden, wird die Funktion des Injektionswerkzeugs nicht durch Abstürze von B_i gefährdet. Der Rechner, der den Fehlerinjektor ausführt, wird in diesem Zusammenhang als *Host*, der Experimentrechner als *Target* bezeichnet. Bei der gegenwärtigen Konfiguration ist das Target teil des Hosts, die Kommunikation findet über den PCI-Bus statt. Fehler in Speicherbereichen des Targets werden über den PCI-Bus in den entsprechenden PCI-Adressbereich des Targets injiziert. Speicherschutzmechanismen des Host-Prozessors und entsprechende Routinen in der Modulbibliothek des Entwicklungsboards verhindern, dass unbeabsichtigt Speicherbereiche des Hosts so verändert werden, dass dessen Funktion beeinträchtigt wird.

Nachdem die geschilderten Initialisierungsarbeiten durchgeführt wurden, untersucht das Optimierer-Modul die Fehler aus der Spezifikation auf deren möglichen Wirkungsgrad, denn es ist möglich, dass injizierte Fehler

- in einen nicht-konfigurierten Bereich des FPGAs injiziert werden,
- ein permanenter Fehler an den gleichen Ort zum zweiten Mal eingebracht wird und
- die Funktion eines LUTs nicht beeinträchtigt wird, da dessen logische Funktion gleich bleibt.

Der Optimierer kann keine Reduktion des Zeitaufwands, u.a. bei folgenden Faktoren herbeiführen:

- Erhöhte Anzahl erforderlicher Programmläufe unter Fehlerinjektion, wenn das Verhalten des zu bewertenden Systems für verschiedene Eingaben oder Systemkonfigurationen beobachtet werden soll,
- Änderung des Kontrollflusses des zu bewertenden Systems zu z.B. Endlosschleifen und
- zeitaufwendiges Rücksetzen des zu bewertenden Systems in einen definierten Startzustand zwischen zwei Injektionsläufen.

2.3 Komponentenbasierte Fehlerinjektion durch partielle Bitfiles

Nach Optimierung der Fehlermenge, können Fehler - wie in Abschnitt 2.1 geschildert - entweder in das komplette Bitfile durch totale Rekonfiguration oder – wenn dies die Architektur zulässt – in ein partielles Bitfile eingebracht werden. Partielle Bitfiles ermöglichen komponentenbasierte Fehlerinjektion, falls Fläche und Ort dieser Komponenten im User Constraints File (UCF) spezifiziert wurden. Fläche und Ort beziehen sich auf FPGA-spezifische CLB-Zeilen und Spalten. Die Fläche hat dabei immer eine rechteckige Form. Die partiellen Bitfiles pB_i werden aus dem Bitfile B_1 unter Analyse des UCF extrahiert. Eine Fehlerinjektion kann dann auf pB_i stattfinden und pB_i durch partielle Rekonfiguration erneut auf das FPGA geladen werden. Dies erhöht die Genauigkeit und die Geschwindigkeit der Fehlerinjektion, da man aufgrund des UCF den Ort einer Komponente kennt und weniger Daten auf das FPGA geladen werden müssen. In [6] wird berichtet, dass die Zeit für eine minimale partielle Rekonfiguration mit 16 CLBs ca. 0.6ms beträgt und exponentiell mit der Anzahl der CLBs wächst. Die Zeit für eine totale Rekonfiguration kann hingegen bis einige hundert ms betragen [7]. Das Fehlerinjektor-Modul kann angewiesen werden, eine tabellarische Übersicht zu den Ergebnissen der einzelnen Injektionsläufe zu erstellen. Tabelle 3 zeigt eine Übersicht der gespeicherten Daten dieses Moduls.

Tabelle 3. Aus der Fehlerinjektion gewonnene Daten

Name	Bezeichnung	Bereich
A_____	Ausgaben des Optimierers	String
Lxxxx	Fehler konnte lokalisiert (L) werden, xxxx: Fehlerort xxxx = 0000 Fehler konnte nicht lokalisiert werden.	$0..2^{16}$-1
Zxxxxxxxx	Zeitangabe in Takten (Target), wann der Fehler erkannt wurde	$0..2^{32}$-1
Yxxxxxxx	Zeitangabe in Takten (Target), wann der Fehler injiziert wurde	$0..2^{32}$-1
Exxxxxxxx	Zeit vom injizieren des Fehlers bis zum Erkennen (Target)	$0..2^{32}$-1
Pxxxxxxxx	Programmausführungszeit in Takten (Target)	$0..2^{32}$-1
I_____	Eingaben der einzelnen Injektionsläufe	String
O_____	Ausgaben der einzelnen Injektionsläufe	String
Exxxx	Auswirkung und Interpretation des Fehlers; die hier festgehaltenen Werte sind stark vom eingesetzten System und dessen Fähigkeiten Fehler zu erkennen und zu tolerieren abhängig.	$0..2^{16}$-1

3 Schlussfolgerungen

Mit den oben geschilderten Möglichkeiten innerhalb des Fehlerinjektionssystems ergeben sich die folgenden Vorteile:

- Es lassen sich parallel unterschiedliche Fehlerarten und mehrere Fehler einbringen.
- Die Fehlerdauer kann modelliert werden, dadurch ist eine Modellierung permanenter und transienter Fehler möglich.
- Es werden bestimmte Fehler direkt in der Hardware simuliert.
- Die Fehler können komponentengenau injiziert werden, wodurch Untersuchungen zur Ausbreitung von Fehlern unterstützt werden. Diese Art der Injektion kann aufgrund partieller Rekonfiguration schnell durchgeführt werden, weiterhin ist die klassische Methode der (ungezielten) Fehlerinjektion durch totale Rekonfiguration möglich.
- Aufgrund der Fehlerspezifikation, kann die Dauer eines Fehlers und der Zeitpunkt seines Auftretens festgelegt werden. Dadurch ist es möglich, missionsspezifische Fehlercharakteristika wie z.B. bei Weltraummissionen den Strahlungsanstieg bei Passage eines strahlungsemitierenden Himmelskörpers zu simulieren.

Da das Verfahren auf FPGA-Fehlerinjektion beruht, ergibt sich hieraus der Nachteil, dass aufgrund des unterschiedlichen Aufbaus von FPGAs das Verfahren nur für bestimmte FPGA-Typen eingesetzt werden kann. Dieser Nachteil wird durch Verwendung einer Architektur-Datei ausgeglichen. Diese muss jedoch erst für den jeweiligen FPGA-Typ erstellt werden. Bei komponentengenauer Fehlerinjektion ist man auf FPGA-Typen beschränkt, die partiell rekonfigurierbar sind. Bei der Umsetzung in ein ASIC (Application Specific Integrated Cirtuit)-Design, können die Ergebnisse aus den Fehlerinjektionsexperimenten nicht ohne weiteres übertragen werden. Ein Hauptgrund dafür ist die unterschiedliche Flächenaufteilung, falls das FPGA-Design nicht eins-zu-eins in einen ASIC umgesetzt wird. FPGAs bestehen zu großen Teilen aus der Verdrahtung. So ist die Wahrscheinlichkeit für einen Fehler bei einer physikalischen Fehlerinjektion hier am größten.

Literatur

[1] P. Folkesson: Assessment and Comparison of Physical Fault Injection Techniques; Tech. Report 377, Dep. of Computer Engineering, Chalmers University of Technology, Göteborg, Schweden, Dissertation, 1999.

[2] J. Carreira, J. Silva: Why do Some (weird) People Inject Faults? ACM Software Engineering Notes, 01.1998, 42-43.

[3] M.-C. Hsueh, T.K. Tsai, R.K. Iyer: Fault Injection Techniques and Tools; IEEE Transactions on Computers, vol. 30, no. 4, 1997, 75-82.

[4] K. Echtle, J.G. Silva: Fehlerinjektion – ein Mittel zur Bewertung der Maßnahmen gegen Fehler in komplexen Rechensystemen; Informatik Spektrum 21(6), 1998, 328-336.

[5] Xilinx, *Virtex architecture guide*, Tech. Rep., Xilinx, San Jose, CA, September 2000, Provided as a part of the JBits 2.8 SDK.

[6] U. Malik, K. So, O. Diessel, *Resource-Aware Run-time Elaboration of Behavioral FPGA Specifications*, IEEE International Conference on Field-Programmable Technology, pp. 68-75, Dec. 2002.

[7] L. Antoni, R. Leveugle, B. Fehér, *Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes*. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 245-253, 2002.