Effiziente Simulation von strukturellen Fehlern für die Zuverlässigkeitsanalyse auf Systemebene

Michael A. Kochte, Christian G. Zöllin, Rafal Baranowski,
Michael E. Imhof, Hans-Joachim Wunderlich
Universität Stuttgart
Institute für Technische Informatik
Pfaffenwaldring 47
D-70569 Stuttgart, Deutschland

Nadereh Hatami, Stefano Di Carlo,
Paolo Prinetto
Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24
I-10129 Torino TO, Italy

***Kurzfassung*—In aktueller Prozesstechnologie muss die Zuverlässigkeit in allen Entwurfsschritten von eingebetteten Systemen betrachtet werden. Methoden, die nur Modelle auf unteren Abstraktionsebenen, wie Gatter- oder Registertransferebene, verwenden, bieten zwar eine hohe Genauigkeit, sind aber zu ineffizient, um komplexe Hardware/Software-Systeme zu analysieren. Hier werden ebenenübergreifende Verfahren benötigt, die auch hohe Abstraktion unterstützen, um effizient die Auswirkungen von Defekten im System bewerten zu können. Diese Arbeit stellt eine Methode vor, die aktuelle Techniken für die effiziente Simulation von strukturellen Fehlern mit Systemmodellierung auf Transaktionsebene kombiniert. Auf diese Weise ist es möglich, eine präzise Bewertung der Fehlerauswirkung auf das gesamte Hardware/Software-System durchzuführen. Die Ergebnisse einer Fallstudie eines Hardware/Software-Systems zur Datenverschlüsselung und Bildkompression werden diskutiert und die Methode wird mit einem Standard-Fehlerinjektionsverfahren verglichen.**

***Schlüsselwörter*—Transaktionsebenen-Modellierung, Ebenenübergreifende Fehlersimulation**

***Title*—Efficient Simulation of Structural Faults for System Reliability Evaluation**

***Abstract*—Reliability assessment has become indispensable in the course of embedded systems development. Evaluation at gate- and register transfer level is accurate but requires high computational effort and is therefore not applicable to contemporary hardware/software systems. Precise low-level fault simulation techniques need to be combined with fast, high-level models to evaluate the effect of physical defects on entire system operation. In this work, state-of-the-art techniques for parallel fault simulation at gate-level are combined with concurrent system simulation at transaction-level. The proposed approach enables accurate evaluation of structural fault effects on the operation of complex hardware/software systems. Its accuracy and performance gain is confirmed by a comparison with a standard gate-level/RTL mixed-level approach in several case studies.**

***Index Terms*—Transaction level modelling, multi-level fault simulation**

I. Einleitung

Strukturfehler modellieren die Folgen von Defekten auf Gatter- und Logikebene. Hohe Variabilität und komplexe Defektmechanismen in nano-elektronischen CMOS-Schaltungen erfordern, dass Strukturfehler nicht nur im Test [1], sondern auch für den funktionalen Betrieb [2] berücksichtigt werden müssen. Fehlertoleranz zum Beispiel kombiniert oft Techniken auf Gatter- und Registertransferebene (RTL) mit Techniken auf Architektur- bzw. Systemebene einschließlich Software. Die Auswirkungen von Fehlern hängen dabei noch von den Anwendungsszenarien ab [3, 4], die die Hardware unterschiedlich nutzen und auslasten.

Nur eine Untermenge der auf der Logikebene beobachtbaren Fehlern führt tatsächlich zu Ausfällen auf der Systemebene [5]. Für diese Fehler muss jedoch eine sehr genaue Analyse durchgeführt werden. Durch ihre Betrachtung in frühen Entwurfsphasen ergeben sich wichtige Rückschlüsse für die Entwicklung zuverlässiger [6, 7] und sicherer Systeme [8]. Für komplexe Systeme ist es aufgrund der Modellgröße oder -verfügbarkeit nicht praktikabel, Modelle auf Gatterebene zu simulieren. Stattdessen werden ebenenübergreifende (multi-level) Verfahren eingesetzt. Diese Verfahren verwenden Modelle auf unterschiedlichen Abstraktionsebenen, z.B. Modelle mit hoher Genauigkeit zur Fehlerinjektion, und Modelle mit hoher Abstraktion zur Bewertung der Konsequenzen [5]. Oft werden dabei erst die Fehler bestimmt, die an Modellgrenzen und Schaltungsausgängen sichtbar sind, um sie dann auf hoher Abstraktionsebene möglichst ohne Verlust an Genauigkeit zu propagieren. So werden die Vorteile der strukturellen Modellierung beibehalten, und trotzdem eine viel höhere Simulationsgeschwindigkeit erzielt.

Für die unterschiedlichen Abstraktionsebenen wurden zahlreiche Ansätze vorgeschlagen, die hier nur auszugsweise genannt werden können:

- Ebenenübergreifende Simulation von Modellen auf Schalter- und Gatterebene [9]
- Serielle ereignisgesteuerte Simulation struktureller Fehler in Modellen auf Gatterebene und RTL [10, 11]
- Ebenenübergreifende Fehlersimulation von Modellen auf Gatterebene und RTL mit nebenläufiger (concurrent) Simulation [12, 13]
- Simulation von Strukturfehlern auf Gatter-/Architekturebene mit symbolischer Simulation im Modell auf Architekturebene [14, 15]
- Serielle Fehlerinjektion auf RTL mit Fehlerpropagierung auf Systemebene [5]
- Strukturelle Fehlerinjektion in Modelle auf Gatterebene und SystemC-Modelle mit hoher Abstraktion [16]
- Mutationsbasierte Injektion von Fehlern in RTL Modelle und Transaktionsebenen-Modellen (TLM) [17].

Die hier vorgestellte Arbeit ist der erste Ansatz, der eine nebenläufige Fehlersimulation auf Gatter- und Transaktionsebene in einer integrierten Simulationsumgebung implementiert. Unser Ansatz basiert auf einem strukturellen Fehlermodell und einem effizienten nebenläufigen Fehlersimulator auf Gatterebene. Die an Komponentengrenzen sichtbaren Auswirkungen der injizierten Fehler werden dann auf der Transaktionsebene nebenläufig propagiert. Auf der Systemebene werden so die Auswirkungen technologienah modellierter Fehler bewertbar. Der dabei verwendete Rollback-Mechanismus kann einfach in vorhandenen Modellen und TLM-Simulatoren verwendet werden. Der Vorteil dieses Ansatzes ist die Kombination der Genauigkeit der Simulation auf Gatterebene mit der sehr hohen Simulationsgeschwindigkeit von TLMs. Dadurch ergibt

sich die Möglichkeit, den Einfluss der Systemanwendung auf die Zuverlässigkeit mit zu untersuchen.

TLMs zur Bewertung des Entwurfsraums können hier für eine Zuverlässigkeitsanalyse wiederverwendet werden. Die vorgeschlagene Methode kann in frühen Entwurfsphasen eingesetzt werden, da genaue Strukturmodelle nur von einzelnen Komponenten oder IP-Cores erforderlich sind, nicht aber für den gesamten Entwurf. Dies ist häufig in Core-basierten Entwurfsprozessen der Fall.

Der Artikel ist wie folgt aufgebaut: Das zweite Kapitel beschreibt kurz die Modellierung auf Transaktionsebene. Im dritten Kapitel wird die vorgeschlagene Methode und die Integration eines sequentiellen Fehlersimulators auf Gatterebene in TLMs diskutiert. Kapitel vier stellt die nebenläufige Fehlerpropagierung auf Transaktionsebene dar. Im fünften Kapitel werden Implementierungsdetails der vorgeschlagenen Methode beschrieben. Das letzte Kapitel stellt zwei Fallstudien vor und diskutiert die Geschwindigkeit und Genauigkeit des Verfahrens, als auch die Ergebnisse der Fehlersimulation zur Zuverlässigkeitsbewertung der zwei Systeme.

II. Modellierung auf der Transaktionsebene

Die wachsende Systemkomplexität bedingt den Wechsel zu höheren Abstraktionsebenen in der Systemmodellierung [18]. Transaktionsebenenmodelle (TLMs) ermöglichen die simulationsgestützte Untersuchung des Entwurfsraums und die Entwurfsverifikation [19] für große Hard-/Softwaresysteme durch eine Beschleunigung um mehrere Größenordnungen im Vergleich zur Modellierung auf Gatter- oder Registertransferebene.

Die Beschleunigung wird erreicht, indem von der Kommunikation auf der Signalebene abstrahiert wird und komplexe Kommunikationsoperationen in atomaren Transaktionen zusammengefasst werden. Der ereignisbasierte Simulator muss dadurch eine wesentlich geringere Zahl von Ereignissen abarbeiten und die Zahl der Kontextwechsel zwischen Simulationsprozessen wird reduziert [20]. Die Modularität und Trennung von Kommunikation und Funktionalität in TLMs erlaubt die schnelle Untersuchung verschiedener Implementierungsalternativen. Die so entstehenden Modelle bieten jedoch noch ausreichend Details um wichtige Entscheidungen in Bezug auf die Performanz, Chipfläche und Verlustleistung zu treffen [21, 22].

Funktionale Einheiten in TLMs sind *Module*, die eine Menge von nebenläufigen Prozessen besitzen, welche das Verhalten von Hard- und/oder Softwaremodulen abbilden. Diese Module kommunizieren über Transaktionen, die durch abstrakte Kommunikationskanäle (*Channels*) mit wohldefinierten Schnittstellen übertragen werden. SystemC und der TLM-2.0-Standard [23] definieren einen Simulationskern und gemeinsame Datentypen sowie Schnittstellen, welche die Modellierung von busbasierten System on Chip (SoC) Plattformen vereinfachen. Zu den wohldefinierten Basis-Schnittstellen gehören zum Beispiel blockierende und nicht-blockierende Transport-Schnittstellen, die benutzt werden, um Transaktionen zwischen Kommunikations-Initiatoren, Verbindungsressourcen und Zielen zu versenden.

Die Modellierung auf der Transaktionsebene, z.B. mit SystemC, erlaubt die Modellierung des Zeitverhaltens mit Granularitäten von taktgenau (cycle-accurate) über approximativ (approximately timed) bis hin zum Verzicht auf ein Zeitmodell (untimed) [24]. Der TLM-2.0-Standard konzentriert sich auf Modelle mit ungefährem Zeitverhalten. Der Standard unterscheidet zwei Zeitauflösungen für Transaktionen, die “Loosely Timed“ und “Approximately Timed“ genannt werden.

III. Ebenenübergreifendes Systemmodell für die Fehlersimulation

Dieses Kapitel beschreibt die Integration von Modellen aus unterschiedlichen Abstraktionsebenen in eine gemeinsame Umgebung für die Fehlersimulation. Zuerst wird der Einfluss des TLM Entwurfsstils auf die Genauigkeit der Fehlerinjektion diskutiert. Darauf aufbauend werden die Wrapper behandelt, die an Modellgrenzen Kommunikationsvorgänge übersetzen und den Fehlersimulator auf Gatterebene beinhalten.

Die hier vorgeschlagene Methode zur ebenenübergreifenden Fehlersimulation kombiniert die Genauigkeit der Fehlersimulation auf der Gatterebene mit der hohen Simulationsgeschwindigkeit von Verhaltensmodellen. Ein Modell des Systems auf der Transaktionsebene wird um genaue strukturelle Modelle der Komponenten auf Gatterebene erweitert. Die Fehlersimulation auf der Gatterebene bestimmt, welche Fehler an den Ausgängen des Modells beobachtbar sind. Für diese Untermenge der Fehler wird eine funktionale Fehlerpropagierung auf der Transaktionsebene durchgeführt. Auf Systemebene wird dann bestimmt, ob der Fehler tatsächlich zu einem Fehlverhalten des Systems führt, d.h. ob er in der Anwendung beobachtbar ist. Gängige Klassifizierungen [25] sind:

- Harmloser Fehler (Benign/recoverable error)
- Kritischer Fehler (Silent data corruption)
- Erkannter, nicht zu behebender Fehler (Detected unrecoverable error)

Die Fehlerinjektion basiert auf einem Mutationsmodell für TLM [17], statt zufälligen Veränderungen bildet sie aber die strukturellen Veränderungen wie folgt ab:

- Verfälschung eines Parameters wie Adresse oder Nutzlast einer Transaktion, die von dem fehlersimulierten Gatterebenenmodell ausgeht.
- Verfälschung des Rückgabewerts wie Nutzlast oder Zustand einer Transaktion, die die Fehlersimulation auf Gatterebene initiiert hat.
- Eine Transaktion, die fälschlicherweise durch das fehlersimulierte Gatterebenenmodell erzeugt wurde.
- Ein Transaktions-Deadlock in der auf Gatterebene fehlersimulierten Komponente.

Abb. 1 zeigt den Ablauf der vorgeschlagenen Methode. Das System und die Zielanwendung werden auf Transaktionsebene modelliert. Für die zu untersuchenden Hardwareblöcke werden Instanzen eines Fehlersimulators erzeugt, die entsprechende Gatterebenen-Modelle verwenden.

Die Schnittstelle zwischen den Abstraktionsebenen bildet ein Wrapper (s. Abb. 2). Der Wrapper umschliesst eine auf Gatterebene abgebildete Komponente und übersetzt die Transaktionen in das entsprechende Kommunikationsprotokoll takt- und pin-genau. Der Wrapper enthält auch den Fehlersimulator und evaluiert dessen Fehlerinformation und veranlasst gegebenenfalls die Fehlerpropagierung auf Transaktionsebene.

A. Modelle auf Transaktionsebene

Da TLMs von Hard- und Softwaremodulen oft zur Exploration des Entwurfsraums verwendet werden, können sie zur Fehlersimulation wiederverwendet werden. Die hier vorgeschlagene ebenenübergreifende Methode benötigt nur die für die Injektion von Fehlern relevante Komponente als Gatterebenen-Modell. Dadurch wird nur eine partielle Abbildung des Gesamtsystems benötigt und die Bindung ist auf die untersuchte Komponente beschränkt. Damit kann das Verfahren in frühen Phasen genutzt werden um Allokations- und Bindungsentscheidungen, als auch Entwurfsalternativen z.B. im Hinblick auf die Zuverlässigkeit zu untersuchen.

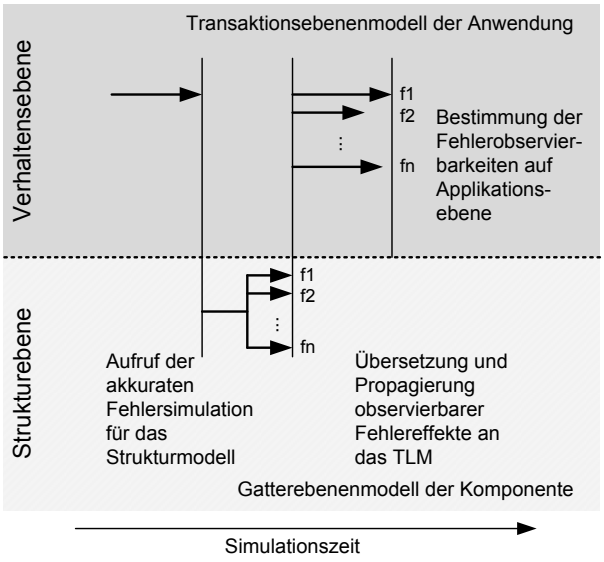

Abbildung 1. Konzeptueller Überblick der vorgeschlagenen ebenenübergreifenden Fehlersimulation

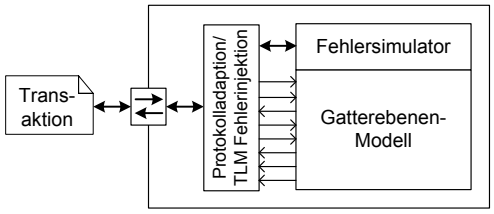

Abbildung 2. Wrapper zwischen System-TLM und Fehlersimulator auf Gatterebene

Die zeitliche Genauigkeit eines Modells auf Transaktionsebene kann über mehrere Größenordnungen reichen und der Designer hat große Freiheiten in der Modellierung der zeitlichen Aspekte. Dagegen sind RT-Modelle meist zyklengenau und Gatterebenen-Modelle enthalten Gatterverzögerungen. Es gibt strukturelle Fehler, die das sequentielle Verhalten einer Komponente beeinflussen und zum Beispiel zu längeren Ausführungszeiten für bestimmte Operationen führen. Dadurch verursachte Systemfehler können in TLMs, deren Zeitverhalten ungenau modelliert ist, nicht immer beobachtet werden. Um die Genauigkeit für diese Arten von Fehlern zu erhöhen, wird eine adaptive Zeitgenauigkeit [26] zumindest in der direkten Nachbarschaft der betrachteten Komponente verwendet.

B. Wrapper für Modelle auf Gatterebene

Die Genauigkeit von Modellen auf Gatterebene erlaubt die Modellierung mehrerer Aspekte eines Systems, die normalerweise nicht auf der Transaktionsebene betrachtet werden, wie z.B. mehrwertige Logik, mehrere Taktphasen und Reset-Signale. In einer funktionalen Simulation ohne Fehlerinjektion können diese Aspekte an der Grenze der Komponente vernachlässigt werden: Mehrwertige Logik kann einfach umgesetzt werden, z.B. für konfliktfreie Busse. Taktphasen verhalten sich deterministisch, wenn ihre Beziehung bekannt und konstant ist. Reset-Signale entfernen alle unbekannten Werte aus dem Gatterebenen-Modell.

In einer von einem strukturellen Fehler betroffenen Gatterebenen-Komponente sind diese Modellierungsaspekte jedoch auch an der Modulgrenze sichtbar: Auf Bussen können Konflikte entstehen, die berücksichtigt werden müssen. Taktphasen, die sich zuvor in einer bekannten Beziehung zueinander befanden, verhalten sich indeterministisch bzw. undefiniert. Strukturelle Fehler in Reset-Signalen führen aufgrund ihres hohen Fan-outs zu jeder Kombination von uninitalisierten Latches oder Flip-Flops, die sich als unbekannte Werte an der Grenze zwischen Gatterebene und TLM äußern. In diesen Fällen hat der Wrapper erheblichen Einflußauf die Genauigkeit der Systemanalyse, da ein signifikanter Teil der Fehler auf Gatterebene mit Reset- oder Takt-Signalen zusammenhängt.

Der Wrapper, der das Gatterebenen-Modell im Transaktionsebenenkontext kapselt, ist also nicht nur für die Umsetzung von Transaktionen in das pin- und taktgenaue Protokoll der Gatterebenen-Komponente zuständig. Da die unbekannten Werte nicht einfach in der regulären Transaktionsebenenmodellierung repräsentiert werden können, entscheidet der Wrapper, wie mit solchen Werten umgegangen wird. Der Wrapper ersetzt hierzur die unbekannten Werte mit zufälligen Werten oder mit Werten für den ungünstigsten oder besten Fall, abhängig davon ob pessimistische oder optimistische Grenzen für die Systemzuverlässigkeit bestimmt werden sollen. Die genaue vorgehensweise ist hierbei von der Funktion des betrachteten Moduls abhängig.

Für die im Wrapper ausgeführte Fehlersimulation auf Gatterebene kann ein beträchtlicher Anteil an Parallelität durch die effiziente parallele Evaluation von Fehlern, Testmustern und Gattern erreicht werden [27]. Für die hier betrachteten sequentiellen Schaltungen erreicht die nebenläufige Fehlersimulation (concurrent fault simulation) [28] durch die gleichzeitige Simulation mehrerer Fehler und die Ausnutzung gemeinsamer Sensibilisierungskriterien eine hohe Effizienz. In der nebenläufigen Simulation müssen für jeden Fehler die Werte der Knoten, die sich von der fehlerfreien Simulation unterscheiden, gespeichert und für jeden Eingangsvektor aktualisiert werden.

IV. Fehlerpropagierung und Evaluierung auf Transaktionsebene

Die Fehlersimulation auf Gatterebene bestimmt die Beobachtbarkeit von Fehlereffekten an den Primärausgängen der Komponente. Um zu bestimmen, ob ein Fehler zu einem Systemausfall oder einem anderen unerwünschten Einfluss auf die Systemfunktionalität führt, muss die Propagierung von Fehlereffekten im System bzw. Anwendungskontext betrachtet werden. Dieses Kapitel stellt die auf der Transaktionsebene durchgeführte, effiziente und nebenläufige Fehlerpropagierung und -evaluierung vor.

A. Fehlerinjektion zur Fehlerpropagierung

Zur Fehlerpropagierung auf Transaktionsebene werden Fehler, die an der Grenze zwischen Gatter- und Transaktionsebene beobachtbar sind, in einer atomaren Transaktion injiziert. Die genaue Veränderung wird durch den Wrapper des Gatterebenenmodells bestimmt, sobald der Gatterebenen-Fehlersimulator eine Fehlerpropagierung anfordert. Der Wrapper wertet dazu die vom Gatternetzlisten-Fehlersimulator bereitgestellte Fehlerinformation aus. Hierfür kann ein bestehender Wrapper aus der funktionalen Validierung erweitert werden, um auch die fehlerbehafteten Antworten des Fehlersimulators zu betrachten. Der nebenläufige Simulator auf Gatterebene verwendet algorithmische Optimierungen und speichert mehrere Schaltungszustände differentiell in einem Wort.

Um den Rechenaufwand zu minimieren und den Fehlereffekt schnell zu klassifizieren, werden zuerst nur einige der Ausgänge betrachtet. Es muss zum Beispiel keine Fehlerpropagierung durchgeführt werden, wenn der Fehler bereits durch das Busprotokoll selbst maskiert wird (zum Beispiel wenn der Fehlersimulator eine fehlerhafte Antwort in einem Bit der Adresse anzeigt, das zugehörige Statusbit den Buszugriff aber als ungültig deklariert). Solche Antworten werden bereits im Wrapper als harmlos eingestuft und so wird die unnötige Fehlerpropagierung vermieden.

B. Bestimmung von Systemfehlern

Als Systemfehler wird eine Abweichung des Verhalten des Systems von der Anwendungsspezifikation bezeichnet. Die Spezifikation wird in der funktionalen Verifikation durch unterschiedliche Testszenarien durchgeführt. Diese Tests werden auch für die Fehlersimulation wiederverwendet. In einem ganzheitlichen Modell, das auch die Umwelt beinhaltet, kann das Erreichen bestimmter Systemvorgaben evaluiert werden, z.B. in einem Lenk- oder Stabilitätskontroll-System.

Falls die fehlersimulierte Komponente selbsttestend oder selbstüberprüfend [29] ist, wird dieser Mechanismus genutzt, um Fehler zu erkennen und zu klassifizieren. Ebenso wird auf eine eventuelle Modellinstrumentierung aus der funktionalen Validierung zurückgegriffen. Assertions implementieren beispielsweise Plausibilitätsprüfungen, um Fehlerzustände und Verletzungen im Protokoll oder Kontrollfluss zu finden. Auf Systemebene prüfen diese Assertions auf verbotene Wertebereiche von externen Variablen, beispielsweise den Ausgang eines Aktuators in einem Kontrollsystem.

Um die Fehlersimulation zu beschleunigen, wird die im TLM aufgerufene Fehlerpropagierungen beendet, sobald für die Klassifikation des Fehlers ausreichend Informationen verfügbar sind. Bei Anwendungen mit Signalverarbeitung wird eine Prüfsumme des Ausgangsdatenstroms berechnet. Die so gewonnene Prüfsumme wird dann an Kontrollpunkten ausgewertet und verglichen.

C. Nebenläufige Fehlerpropagierung

Um eine große Anzahl an Fehlern effizient zu propagieren, müssen der fehlerfreie Simulationszustand effizient wiederhergestellt und die durch Propagierung verursachten Änderungen rückgängig gemacht werden. In Fehlersimulatoren auf der Gatterebene wird dies durch die Speicherung von Änderungen auf einem Stack, durch Markierungen oder durch sog. groupIDs [30] erreicht. Für TLM Simulationen ist dies jedoch nicht durchführbar, da die Modelle größtenteils aus funktionalen Abstraktionen in Form von übersetztem Programmcode bestehen. Neben Code-Modifikationen basieren existierende Fehlerinjektionsmechanismen für TLM auf der Instrumentierung der übersetzten Simulationsbinärdateien [31] oder des TLM Simulationskerns [32]. Dennoch kann mit diesen Methoden eine Simulatorsitzung nur für eine einzelne Fehlerinjektion genutzt werden.

Die hier vorgeschlagene Methode zur Fehlerinjektion basiert auf dem Konzept der nebenläufigen Fehlersimulation mit einer fehlerfreien Maschine und mehreren fehlerbehafteten Maschinen bzw. Schaltungszuständen, die parallel ausgewertet werden. Die fehlerfreie Maschine läuft als Hauptprozess. Fehlerbehaftete Maschinen werden mit Hilfe des Betriebssystems effizient als Unterprozesse erzeugt. Da Prozesse voneinander abgeschottet sind, müssen zum Zurücksetzen des Simulationszustands lediglich diese Unterprozesse terminiert werden. Neben den geringen Kosten für die Erzeugung von fehlerbehafteten Maschinen und ihrer Terminierung, ist das vorgeschlagene Verfahren auf Rechnern mit mehreren Kernen prinzipbedingt nebenläufig.

Das Verfahren lässt sich einfach für jedes existierende Transaktionsebenenmodell implementieren. Es sind keine Änderungen am Simulationskern nötig und IP-Cores, die nur in Binärform verfügbar sind, können weiterhin verwendet werden. Die Evaluierung von Systemfehlern erfolgt komplett in der fehlerbehafteten Maschine. Nur für die Fehlerklassifizierung muss eine Kommunikation zwischen den Prozessen der fehlerfreien und der fehlerbehafteten Maschinen erfolgen. Dies wird sehr kostengünstig mittels den Rückgabewerten der Unterprozesse erreicht.

V. Implementierung

Die ebenenübergreifende Fehlersimulationsmethode wurde basierend auf dem sequentiellen Gatterebenen-Fehlersimulator Hope [30] und den OSCI SystemC 2.2 und TLM-2.0 Bibliotheken implementiert. Um die Integration in die objektorientierte SystemC Simulationsumgebung zu ermöglichen, wurde der Fehlersimulator nach C++ portiert. Im objekt-orientierten Hope wurden relevante Datenstrukturen und Methoden freigelegt, um auf Fehlererkennungsinformationen zugreifen zu können. Weiterhin wurden Schnittstellen für das Anstoßen der Fehlerpropagierung auf Transaktionsebene hinzugefügt. Für die betrachteten Modelle auf Gatterebene werden dynamisch separate Instanzen des Fehlersimulators erzeugt. Während die algorithmischen Optimierungen in Hope auf das Haftfehlermodell ausgerichtet sind, können diese mit Hilfe des Konzepts der bedingten Haftfehler [33] für andere strukturelle Fehlermodelle erweitert werden.

Das Klassendiagramm in Abb. 3 zeigt die Beziehungen zwischen den verschiedenen Entitäten in der Implementierung. Abgeleitet vom generischen SystemC-Modul

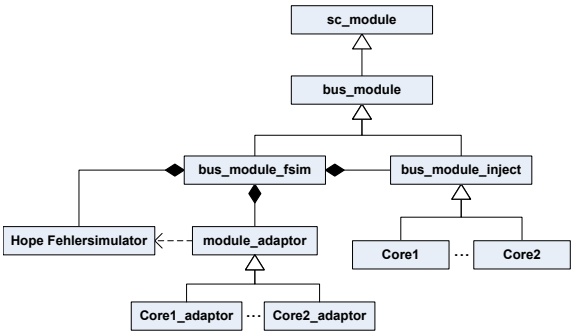

Abbildung 3. Klassendiagramm des ebenenübergreifenden System-Modells

sc_module implementiert die Klasse *bus_module* das Busprotokoll des Systems. Die Klasse *bus_module_inject* erweitert diese Klasse um Methoden für den Zugriff auf die fehlerfreie Transaktions-Nutzlast und die Fehlerinjektion in eine Transaktion. Falls keine Fehlersimulation ausgeführt wird, verhält sich *bus_module_inject* wie das Verhaltensmodell. Während einer Fehlersimulation ist diese Klasse für die Injektion eines Fehlers in eine Transaktion zuständig. Die funktionalen Modelle der modellierten Hardwarekomponenten sind von *bus_module_inject* abgeleitet.

Die Klasse *bus_module_fsim* ist für die Verarbeitung von Transaktionen von und zu den fehlersimulierten Komponenten zuständig. Sie enthält Referenzen zu Instanzen der *bus_module_inject* Klasse, der *module_adaptor* Klasse und der Fehlersimulator-Instanz des betrachteten Gatterebenen-Modells. Wenn keine Fehlersimulation durchgeführt wird, delegiert *bus_module_fsim* die Transaktionen an die *bus_module_inject* Instanz. Während einer Fehlersimulation wird die Transaktion zusätzlich an die *module_adaptor* Instanz weitergeleitet, die die Transaktion in das zyklengenaues Protokoll des Gatterebenen-Modells übersetzt. *module_adaptor* leitet die Daten an die Fehlersimulator-Instanz weiter und kontrolliert die Gatterebenen-Fehlersimulation. Es liest die Information über erkannte Fehler und deren Auswirkungen zurück. Da sich Schnittstellen und Protokolle der Komponenten unterscheiden, müssen separate *module_adaptor*s für jeden benutzten Kern implementiert oder bestehende erweitert werden.

Abbildung 4 zeigt die Interaktion zwischen dem Wrapper der Komponente, der Instanz des Fehlersimulators und der fehlerbehafteten Maschine auf der Transaktionsebene. Der Gatterebenen-Fehlersimulator ist Teil der fehlerfreien Maschine. Fehlerbehaftete TLM Maschinen werden bei Bedarf mit Hilfe des POSIX $fork()$ Kommandos erzeugt. Dies erlaubt die schnelle Erzeugung einer fehlerbehafteten Maschine, da Unix Prozess-Forks mittels Copy-on-Write erzeugt. Die fehlerfreie und fehlerbehaftete Maschine teilen sich dieselben Speicherbereiche bis eine Speicherseite in der fehlerbehafteten Maschine geändert wird. Die verwendeten Mechanismen sind für die meisten Systemmodelle transparent. Eine Ausnahme bilden Dateizeiger, die für den Schreibzugriff besonders behandelt werden müssen. In den fehlerbehafteten Maschinen sollten sie geschlossen werden.

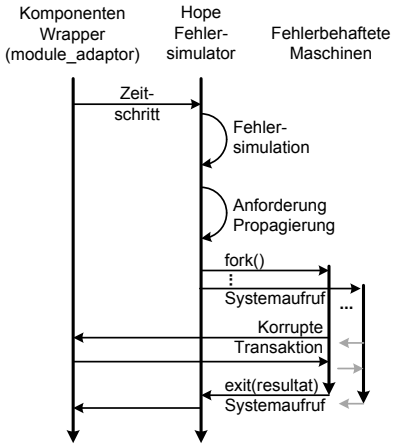

Abbildung 4. Schritte der ebenenübergreifenden Fehlersimulation

VI. Evaluierung

Die Systemzuverlässigkeitsanalyse benötigt ein Modell, das unterschiedliche Parameter, wie z.B. Spannung, Temperatur, oder Strahlungsintensität, berücksichtigt [34]. Hier konzentriert sich die Evaluierung der vorgestellten ebenenübergreifenden Fehlersimulationmethode auf die Genauigkeit der Fehlerklassifikation und die Laufzeit für zwei unterschiedliche Anwendungen, die auf einem AMBA-basierten System-On-a-Chip (SoC) mit einem Leon3-Prozessor ausgeführt werden. Das SoC beinhaltet Hardwarebeschleuniger für den Triple-DES Verschlüsselungsalgorithmus und die zweidimensionale diskrete Kosinus-Transformation (2D-DCT) (s. Abb. 5). Als Hardware/Software Zielanwendungen werden die Verschlüsselung von Daten, sowie die JPEG Kodierung von Bilder betrachtet.

Um von der parallelen Ausführung von Simulatorinstanzen profitieren zu können, werden die Experimente auf einem Multiprozessorsystem mit 8 Intel Xeon CPUs (2,67 GHz) und 48 GB Speicher ausgeführt. Die Speichernutzung überschreitet in keinem Experiment 250 MB.

A. Validierung

Die vorgeschlagene Methode wird durch Fehlerinjektion und Simulation auf Gatter- und Registertransferebene validiert. Der dafür verwendete kommerzielle Simulator entspricht dem Stand der Technik, die hier gezeigten Laufzeitvergleiche untersagen leider eine Namensnennung aufgrund von Lizenzvereinbarungen. Das SoC wird auf Registertransferebene modelliert bis auf den Systemteil, der der Fehlerinjektion ausgesetzt wird und deshalb auf Gatterebene modelliert werden muss. Die Simulation des Gesamtsystems auf der Gatterebene mittels

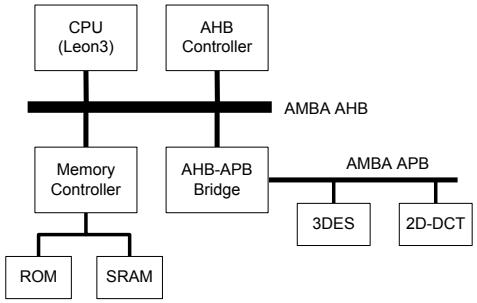

Abbildung 5. System mit Beschleunigern für Triple-DES und 2D-DCT

Hope würde zu wesentlich längeren Rechenzeiten führen und erfordern, dass auch die Testbench vollständig auf Gatterebene modelliert ist.

Während der Initialisierung des Simulators wird ein einzelner Haftfehler injiziert. Die Simulation läuft, bis das Ergebnis einer Anwendung vorliegt. Ein Zeitlimit wird gesetzt, damit die Fehler, die zu Verzögerungen und Deadlocks führen, auch erkannt werden. Das Ergebnis der Simulation wird von der Fehlerinjektionsumgebung bewertet und der Fehler entsprechend klassifiziert. Die Simulation wird danach mit dem nächsten Fehler neu gestartet. Aufgrund der hohen Laufzeiten wird eine Stichprobe von 3000 Fehler auf diese Weise untersucht.

Jeder Fehler wird entsprechend den Kategorien aus Abschnitt IV-B klassifiziert. Zur Validierung werden folgende Fälle unterschieden:

- Abgedeckt: Die Fehlerklassifizierung in der vorgeschlagenen Methode stimmt mit der Validierung überein.
- Falsch kritisch: Der Fehlereffekt wird in der vorgeschlagenen Methode als kritisch, in der Validierung jedoch als harmlos bezeichnet.
- Falsch harmlos: Der Fehlereffekt wird in der vorgeschlagenen Methode als harmlos, in der Validierung jedoch als kritisch bezeichnet.

Die Klasse der erkannten und nicht zu behebenden Fehler wird hier nicht betrachtet, da das SoC keine Fehlererkennung beinhaltet.

Die Validierung wurde auf einer Rechnerfarm ausgeführt, deren Maschinen über 2,4 GHz AMD Athlon 64 Dual Core Prozessoren und 4 GB Speicher verfügen. Um einen Laufzeitvergleich der Validierungsszenarien zu ermöglichen, wurde die vorgeschlagene Methode ebenfalls auf dem 8-Kern Multiprozessorsystem ausgeführt.

B. Triple-DES Verschlüsselungsanwendung

Die erste Fallstudie basiert auf einer Verschlüsselungsanwendung, die von dem Triple-DES Kern aus Abb. 5 beschleunigt wird. Diese Anwendung wurde als Beispielanwendung gewählt, da sie fast keine Maskierung aufweist. Ein Datenblock wird verschlüsselt (entweder die Zeichenfolge eines ASCII-kodierten Textes oder zufällige Werte), wobei die Software, die auf dem Prozessor läuft, für die Übertragung und den Abruf von Daten zuständig ist.

Der Triple-DES Verschlüsselungsalgorithmus stellt eine dreifache Ausführung des DES-Algorithmus (Data Encryption Standard) dar, in dem die Daten mittels XOR-Verknüpfungen, Substitutionen und Permutationen verschlüsselt werden. Die Eingabe besteht aus 64 Datenbits und 64 Schlüsselbits, und das verschlüsselte Ergebnis ist ebenso 64 Bit lang.

Hier wird ein Triple-DES Beschleuniger von OpenCores[1] verwendet und für die allgemeine LSI10k Bibliothek synthetisiert. Die Schaltung enthält 19.917 Logikzellen und 53.010

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

Haftfehler. Zunächst präsentieren wir die Ergebnisse für die Maskierung von Fehlern in dem Beschleuniger auf Systemebene, gefolgt von der Simulationsgeschwindigkeit und der Genauigkeit der Methode.

Tabelle I beschreibt die Fehlermaskierung auf Systemebene für vier Datensätze. Die erste Spalte gibt die Art und die Länge der Daten an, die verschlüsselt werden. Die Schlüssel werden für jedes Szenario zufällig gewählt. In der zweiten Spalte ist die Anzahl der sensibilisierten Fehler gegeben, d.h. Fehler, die an den Ausgängen des Triple-DES-Core beobachtbar sind, aber nicht zwangsläufig zu Fehlern auf Systemebene führen. Die dritte und vierte Spalte geben die Anzahl der kritischen bzw. harmlosen Fehler an. In allen Szenarien führen mehr als 99% der sensibilisierten Fehler zu einem Fehler auf Systemebene (kritische Fehler). Dies lässt sich dadurch erklären, dass die Ergebnisse von dem Beschleuniger direkt zum Systemausgang übertragen werden, so dass keine Fehlermaskierung im Datenpfad stattfindet. Für 193 sensibilisierte Fehler war keine Auswirkung auf Systemebene beobachtbar während das "data ready" Signal aktiv war. Daher wurden die Fehler als harmlos klassifiziert.

Szenario	Sensibilisierte Fehler	Kritische Fehler	Harmlose Fehler
Englisch 3,5 KB	32916	32723	193
Italienisch 21 KB	33247	33054	193
Italienisch 20 KB	32901	32708	193
Zufallswerte 8 KB	32953	32760	193

Tabelle I
FEHLERMASKIERUNG IN DER VERSCHLÜSSELUNGSANWENDUNG

In Tabelle II sind die Laufzeiten des Ansatzes auf der 8-Kern-Maschine wiedergegeben. Die Spalte "Anzahl Sim.-kontexte" beschreibt die Zahl der Fehlerfortpflanzungen, die mittels $fork$ durchgeführt werden. Die dritte und vierte Spalte geben die CPU-Zeit an, die für die Durchführung der parallelen Fehlersimulation bzw. des TLM-Modells benötigt wurde. Die Fehlersimulation in Hope ist im Mittel viermal so aufwändig wie die Fehlerpropagierung im TLM. Die dritte Spalte liefert die Gesamtlaufzeit des Ansatzes, einschließlich der System-CPU-Zeit, die u.a. die Kosten für die Erzeugung der Unterprozesse beinhaltet. Da in den Szenarien die Fehlerfortpflanzung im Vergleich zur Erzeugung von Unterprozessen weniger aufwändig ist, beträgt die System-CPU-Zeit etwa 50% der Gesamtlaufzeit. Dies ist verglichen mit explizitem Rücksetzen des Zustands in einer sequenziellen Fehlersimulation immer noch günstig.

Szenario	Anzahl Sim.-kontexte	Gatterebene Hope CPU-Zeit	TLM CPU-Zeit	Gesamt-laufzeit
Englisch 3,5 KB	32253	1m 03s	0m 15s	4m 51s
Italienisch 21 KB	32584	4m 51s	1m 08s	9m 32s
Italienisch 20 KB	32238	4m 36s	1m 04s	9m 13s
Zufallswerte 8 KB	32290	2m 01s	0m 29s	6m 06s

Tabelle II
LAUFZEIT FÜR DIE VERSCHLÜSSELUNGSAPPLIKATION (IN MIN/SEC)

Die Ergebnisse der Validierung sind in Tabelle III zusammengefasst. Die Experimente wurden für 3000 Fehler durchgeführt, die nach dem Zufallsprinzip aus der Gesamtmenge von 53.010 Fehlern ausgewählt wurden. Die Validierung wurde für einen Text mit 3576 Zeichen ausgeführt. Vier Szenarien mit unterschiedlichen Schlüsseln, die in der ersten Spalte aufgelistet sind, werden untersucht. Der Klassifikation aus Abschnitt VI-A zufolge wurden alle Fehler als "abgedeckt" eingestuft, d.h. kein Fehler wurde im vorgeschlagenen Ansatz falsch klassifiziert. Die letzte zwei Spalten vergleichen die Laufzeiten für die Validierung (RTL/Gatter) und dem vorgeschlagenen Ansatz (TLM/Hope) auf der Athlon-Rechner. Es wurde eine völlige Übereinstimmung der Fehlererkennung bei einer durchschnittlichen Beschleunigung von ca. 16.500x erzielt.

Szenario	CPU-Zeit RTL/Gatter	CPU-Zeit TLM/Hope
Alle "0"	233h	47,5s
Alle "1"	243h	73,2s
Sequenz	234h	40,1s
Zufallswerte	242h	46,1s

Tabelle III
VALIDIERUNG FÜR DIE VERSCHLÜSSELUNG (3,5 KB ENGLISCHER TEXT, STICHPROBE VON 3000 FEHLER)

C. JPEG-Bildkompression

In der zweiten Fallstudie wird eine hohe Fehlermaskierung in einer komplexen Anwendung am Beispiel der JPEG-Kompression untersucht. Der JPEG-Kompressionsalgorithmus gliedert sich in vier Schritte: (1) Farbraumkonvertierung, (2) Zweidimensionale diskrete Kosinustransformation (2D-DCT) der Pixeldaten, (3) Quantelung der DCT-Koeffizienten, und (4) Lauflängenkodierung und Huffman-Kodierung. Das Ergebnis der 2D-DCT-Operation ist die Darstellung der Helligkeits- und Chrominanzwerte der Pixel im Frequenzbereich. Im Quantelungsschritt wird die Genauigkeit dieser DCT-Koeffizienten durch frequenzabhängige Skalierung und Rundung reduziert. Mit steigender Frequenz wird die Genauigkeit zunehmend reduziert, da die visuelle Wahrnehmung des Menschen weniger empfindlich für hohe Frequenzen ist.

Basierend auf dem SoC aus Abb. 5 kodiert die Anwendung 8x8 Pixelblöcke in eine Folge von Huffman-Symbolen. Bei der Kodierung wird die zweidimensionale Kosinus-Transformation (2D-DCT) von einem Hardware-Block beschleunigt, weil sie den Hauptanteil des Rechenaufwands darstellt. Die restlichen Operationen werden in Software von dem LEON3-Prozessor ausgeführt. Der 2D-DCT-Beschleuniger stammt von OpenCores und wurde für die LSI10k Bibliothek synthetisiert. Die Komponente enthält 28.001 Logikzellen und 78.914 Haftfehler.

Tabelle IV beschreibt die Fehlereffekte in dem 2D-DCT Beschleuniger. In der ersten Spalte wird die Art und Größe der Bilder, die im gegebenen Szenario kodiert werden, aufgelistet. Die nachfolgende Spalten sind analog zur bereits diskutierten Tabelle I. Im Vergleich zur Triple-DES Anwendung wird in der JPEG-Anwendung eine viel höhere Zahl von sensibilisierten Fehlern auf Systemebene maskiert. 16% bis zu 36% der sensibilisierten Fehler sind harmlos. Dies ist auf die verlustbehaftete Kompression durch die Quantelungsoperation zurückzuführen.

Szenario	Sensibilisierte Fehler	Kritische Fehler	Harmlose Fehler
Weiß 8x8	25297	16295	9002
Schwarz 8x8	22729	14420	8309
Rauschen 8x8	48794	34064	14730
Obst 64x48	64797	54141	10656

Tabelle IV
FEHLERMASKIERUNG IN DER JPEG-ANWENDUNG

Die Experimente wurden auf dem bereits erwähnten 8-Kern-Rechner ausgeführt. Die Laufzeiten finden sich in Tabelle V. Die Spalten sind analog zu Tabelle II. Die Anzahl von Simulationskontexten hängt von der Art und Größe der Bilddaten ab, da für jeden 8x8 Pixelblock alle Fehler, die

noch nicht als kritisch eingestuft wurden, betrachtet werden. Aufgrund der Maskierung in der Quantelungsoperation müssen u.U. mehrere Fehlerpropagierungen durchgeführt werden, bevor ein Fehler als kritisch klassifiziert werden kann. Aufgrund der Fehleraufgabe steigt die Laufzeit nicht linear mit der Bildgröße. Die Laufzeit für das Bild mit 48 Blöcken ist entsprechend nur siebenfach höher als bei den Szenarien mit einem einzigen Block. Die verbleibenden JPEG Operation zur Quantelung und Huffman-Kodierung werden so effizient auf Verhaltensebene ausgeführt, dass ähnlich zur Triple-DES Anwendung die Erzeugung von Unterprozessen im Betriebssystem einen großen Anteil an den Gesamtkosten ausmachen.

Szenario	Anzahl Sim. kontexte	Gatterebene Hope CPU-Zeit	TLM CPU-Zeit	Gesamt-laufzeit
Weiß 8x8	21813	1m 00s	0m 12s	2m 04s
Schwarz 8x8	19728	1m 02s	0m 12s	2m 06s
Rauschen 8x8	44739	1m 07s	0m 16s	4m 55s
Obst 64x48	275887	8m 17s	2m 03s	34m 49s

Tabelle V
LAUFZEITEN FÜR DIE JPEG-ANWENDUNG

Die Validierung wurde ähnlich der Triple-DES Anwendung für eine Stichprobe von 3000 Fehlern ausgeführt. Aufgrund hohen Laufzeiten der Validierungsexperimente wurden nur Szenarien mit einem Pixelblock untersucht. Die Ergebnisse sind in Tabelle VI zusammengefasst. Die Spalten sind analog zu Tabelle III, bis auf die zusätzlichen Spalten, die die Anzahl von "harmlosen" und "falsch kritischen" Fehlern (s. Abschnitt VI-A) unter den 3000 Fehlern in der Stichprobe angeben. Zwei bis sieben Fehler pro Szenario wurden in der vorgeschlagenen Methode irrtümlich als "kritisch" eingestuft, was pessimistisch ist. Die Analyse der Fehler im Gattermodell ergab, dass sie entweder einen unbekannten Wert auf dem "data ready" Signal verursachen, wenn das Signal inaktiv sein sollte, oder zusätzliche Impulse auf dieser Leitung generieren, nachdem die angelegten Daten ungültig werden. Diese Fehler wurden in der Validierung nur aufgrund der kürzen Applikationsdauer und einer günstigen Synchronisierung als harmlos bezeichnet. In einem realen Chip und unter ungünstigen Umständen können sie in der Tat zum Datenverlust führen. Unter der pessimistischen Annahme, dass die Validierungsexperimente als Referenz dienen, erzielt der vorgeschlagene Ansatz eine Übereinstimmung von 99,8% der Fehler bei einer durchschnittlichen Beschleunigung von ca. 15.400x.

Szenario	Harmlose Fehler	Falsch kritisch	CPU-Zeit RTL/Gatter	CPU-Zeit TLM/Hope
Weiß 8x8	2377	6	115h	25,7s
Schwarz 8x8	2463	7	117h	24,4s
Sequenz 8x8	2067	2	119h	32,8s
Rauschen 8x8	1580	2	148h	33,8s

Tabelle VI
VALIDIERUNG FÜR DIE JPEG-ANWENDUNG FÜR EINE STICHPROBE VON 3000 FEHLERN

VII. SCHLUSSBEMERKUNG

Die vorgeschlagene Fehlersimulationsmethode erlaubt die Analyse struktureller Fehler in einer ebenenübergreifenden Simulation auf Gatter- und Transaktionsebene. Durch effiziente Fehlersimulation auf Gatterebene und nebenläufiger Fehlerfortpflanzung auf Transaktionsebene wird die Simulationszeit um mehrere Größenordnungen reduziert. Dabei können bestehende TLMs aus frühen Entwurfsphasen und der Entwurfsvalidierung wiederverwendet werden. Neben der hohen Beschleunigung wird gleichzeitig die Genauigkeit einer Fehlersimulation auf Gatterebene erreicht.

DANKSAGUNG

Diese Arbeit wurde durch den Deutschen Akademischen Austauschdienst (DAAD) im Rahmen des Vigoni-Programms gefördert.

LITERATUR

[1] K. Roy, T. Mak, and K. Cheng, "Test consideration for nanometer-scale CMOS circuits," *IEEE Design & Test of Computers*, vol. 23, no. 2, pp. 128–136, 2006.

[2] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE MICRO*, pp. 10–16, 2005.

[3] N. Wattanapongsakorn and S. P. Levitan, "Reliability optimization models for embedded systems with multiple applications," *IEEE Transactions on Reliability*, vol. 53, no. 3, pp. 406–416, 2004.

[4] J. Cano and D. Rios, "Reliability forecasting in complex hardware/software systems," in *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES, April 20-22 2006, Vienna University of Technology, Austria*, 2006, pp. 300–304.

[5] R. Leveugle, D. Cimonnet, and A. Ammari, "System-level dependability analysis with RT-level fault injection accuracy," in *19th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2004), 10-13 October 2004, Cannes, France, Proceedings*, 2004, pp. 451–458.

[6] R. Leveugle and K. Hadjiat, "Multi-level fault injections in VHDL descriptions: Alternative approaches and experiments," *J. Electronic Testing*, vol. 19, no. 5, pp. 559–575, 2003.

[7] A. Jhumka, S. Klaus, and S. A. Huss, "A dependability-driven system-level design approach for embedded systems," in *2005 Design, Automation and Test in Europe Conference and Exposition (DATE 2005), 7-11 March 2005, Munich, Germany*, 2005, pp. 372–377.

[8] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, and A. Mühlberger, "High level fault injection for attack simulation in smart cards," in *13th Asian Test Symposium (ATS 2004), 15-17 November 2004, Kenting, Taiwan*, 2004, pp. 118–121.

[9] W. Meyer and R. Camposano, "Active timing multilevel fault-simulation with switch-level accuracy," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 14, no. 10, pp. 1241–1256, 1995.

[10] M. B. Santos and J. P. Teixeira, "Defect-oriented mixed-level fault simulation of digital systems-on-a-chip using HDL," in *1999 Design, Automation and Test in Europe (DATE '99), 9-12 March 1999, Munich, Germany*, 1999, pp. 549–.

[11] Z. Navabi, S. Mirkhani, M. Lavasani, and F. Lombardi, "Using RT level component descriptions for single stuck-at hierarchical fault simulation," *J. Electronic Testing*, vol. 20, no. 6, pp. 575–589, 2004.

[12] S. Gai, P. L. Montessoro, and F. Somenzi, "MOZART: a concurrent multilevel simulator," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 7, no. 9, pp. 1005–1016, 1988.

[13] K. P. Lentz and J. B. Homer, "Handling behavioral components in multi-level concurrent fault simulation," in *Proceedings 33th Annual Simulation Symposium (SS 2000), 16-22 April 2000, Washington, DC, USA*, 2000, pp. 149–156.

[14] M. S. Hsiao and J. H. Patel, "A new architectural-level fault simulation using propagation prediction of grouped fault-effects," in *1995 International Conference on Computer Design (ICCD '95), October 2-4, 1995, Austin, TX, USA, Proceedings*, 1995, p. 628ff.

[15] O. Sinanoglu and A. Orailoglu, "Rt-level fault simulation based on symbolic propagation," in *19th IEEE VLSI Test Symposium (VTS 2001), 29 April - 3 May 2001, Marina Del Rey, CA, USA*, 2001, pp. 240–245.

[16] S. Misera, H. T. Vierhaus, and A. Sieber, "Simulated fault injections and their acceleration in SystemC," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 32, no. 5-6, pp. 270–278, 2008.

[17] G. Beltrame, C. Bolchini, and A. Miele, "Multi-level fault modeling for transaction-level specifications," in *Proceedings of the 19th ACM Great Lakes Symposium on VLSI 2009, Boston Area, MA, USA, May 10-12 2009*, 2009, pp. 87–92.

[18] A. Gerstlauer, C. Haubelt, A. Pimentel, T. Stefanov, D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1517 –1530, oct. 2009.

[19] F. Ghenassia, Ed., *Transaction-Level Modeling with SystemC - TLM Concepts and Applications for Embedded Systems*. Springer, 2005.

[20] M. Radetzki, "Object-oriented transaction level modelling," in *Advances in Design and Specification Languages for Embedded Systems*, S. Huss, Ed. Springer, 2007.

[21] Y. Hwang, S. Abdi, and D. Gajski, "Cycle-approximate retargetable performance estimation at the transaction level," in *Proc. Design, Automation and Test in Europe (DATE)*, 2008, pp. 3–8.

[22] M. Cheema and O. Hammami, "Introducing Energy and Area Estimation in HW/SW Design Flow Based on Transaction Level Modeling," in *Proc. International Conference on Microelectronics (ICM)*, 2006, pp. 182–185.

[23] Open SystemC Initiative (OSCI) TLM Working Group, "Transaction level modeling standard 2 (OSCI TLM 2)," June 2008, www.systemc.org.

[24] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *Proc. International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2003, pp. 19–24.

[25] S. S. Mukherjee, J. S. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th International Conference on High-Performance Computer Architecture (HPCA-11 2005), 12-16 February 2005, San Francisco, CA, USA*, 2005, pp. 243–247.

[26] M. Radetzki and R. S. Khaligh, "Accuracy-adaptive simulation of transaction level models," in *Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, 2008*, 2008, pp. 788–791.

[27] M. A. Kochte, M. Schaal, H.-J. Wunderlich, and C. G. Zoellin, "Efficient fault simulation on many-core processors," in *Design Automation Conference*, 2010.

[28] E. Ulrich and T. Baker, "The concurrent simulation of nearly identical digital networks," in *Proc. 10th Workshop on Design Automation*, 1973, pp. 145–150.

[29] P. Lala, *Self-checking and fault-tolerant digital design*. Morgan Kaufmann, 2001.

[30] H. K. Lee and D. S. Ha, "Hope: An efficient parallel fault simulator for synchronous sequential circuits," in *29th Design Automation Conference, Anaheim, California, USA, June 8-12, 1992*, 1992, pp. 336–340.

[31] A. da Silva Farina and S. Prieto, "On the use of dynamic binary instrumentation to perform faults injection in transaction level models," in *4th International Conference on Dependability of Computer Systems*, 2009, pp. 237 –244.

[32] J. Na, "A novel simulation fault injection using electronic systems level simulation models," *IEEE Design Test of Computers*, no. Early Access Article, 2009.

[33] H. Wunderlich and S. Holst, "Generalized fault modeling for logic diagnosis," in *Models in Hardware Testing*. Springer, 2009, pp. 133–155.

[34] A. Benso and P. Prinetto, Eds., *Fault injection techniques and tools for embedded systems reliability evaluation*. Kluwer Academic Publishers, Boston, 2003.