Zur Zuverlässigkeitsmodellierung von Hardware-Software-Systemen

On the Reliability Modeling of Hardware-Software-Systems

Michael A. Kochte, Institut für Technische Informatik, Universität Stuttgart, D-70569 Stuttgart
Rafal Baranowski, Institut für Technische Informatik, Universität Stuttgart, D-70569 Stuttgart
Hans-Joachim Wunderlich, Institut für Technische Informatik, Universität Stuttgart, D-70569 Stuttgart

Kurzfassung / Abstract

Zur Zuverlässigkeitsanalyse von Hardware-Software-Systemen ist ein Systemmodell notwendig, welches sowohl Struktur und Architektur der Hardware als auch die ausgeführte Funktion betrachtet. Wird einer dieser Aspekte des Gesamtsystems vernachlässigt, kann sich eine zu optimische oder zu konservative Schätzung der Zuverlässigkeit ergeben.
Ein reines Strukturmodell der Hardware erlaubt, den Einfluss von logischer und struktureller Fehlermaskierung auf die Fehlerhäufigkeit der Hardware zu bestimmen. Allerdings kann ein solches Modell nicht die Fehlerhäufigkeit des Gesamtsystems hinreichend genau schätzen. Die Ausführung der Funktion auf dem System führt zu speziellen Nutzungs- und Kommunikationsmustern der Systemkomponenten, die zu erhöhter oder verminderter Anfälligkeit gegenüber Fehlern führen.
Diese Arbeit motiviert die Modellierung funktionaler Aspekte zusammen mit der Struktur des Systems. Mittels Fehlerinjektion und Simulation wird der starke Einfluss der Funktion auf die Fehleranfälligkeit des Systems aufgezeigt. Die vorgestellte Methodik, funktionale Aspekte mit in die Zuverlässigkeitsmodellierung einzubinden, verspricht eine realistischere Bewertung von Hardware-Software-Systemen.

Estimating the reliability of hardware-software systems allows to determine the robustness of design alternatives during design exploration. A system model used to derive such a reliability estimate has to incorporate the hardware structure and architecture of the system as well as the performed function. If merely the functional model or the structural model is considered separate from the other one, reliability estimation may be either too optimistic or too conservative.
While an architectural model allows to determine the impact of logical and architectural fault masking on the design's error rate, it fails to correctly predict the failure rate of the overall system. The function that is performed by the design exhibits particular usage and communication patterns that may—depending on the function—result in increased or reduced susceptibility to faults.
This work motivates to model functional aspects together with the architecture of the system. Fault injection and simulation show the strong influence of the function on the susceptability of the system. The proposed methodology to incorporate functional aspects into the system model for reliability estimation promises a more accurate assessment of hardware-software systems.

Schlüsselwörter / Keywords

Modellierung, Zuverlässigkeit, eingebettete Systeme, System-Level, Systems-on-Chip
Modeling, reliability, embedded systems, system-level, systems-on-chip

1 Einleitung

Die Zuverlässigkeit eines Systems ist eine wichtige Kenngröße und muss in frühen Entwurfsphasen akkurat geschätzt werden, um Randbedingungen und Ziele des Entwurfs zu erfüllen. Die Zuverlässigkeit eines Systems ist als die Wahrscheinlichkeit des Ausfallfreien Betriebs während eines Zeitintervalls t definiert [1], [2]. Im Weiteren kann noch der Grad des Ausfalls, z.B. geringfügige und tolerierbare Abweichungen vom korrekten Resultat oder kritische Ausfälle, unterschieden werden.

Systems-on-Chip (SoC), die z.B. in eingebetteten Anwendungen verwendet werden, bestehen aus verschiedenen Hardwarekomponenten, wie Prozessoren, Speicher, Hardwarebeschleunigern oder E/A-Schnittstellen. Durch Software, die von eingebetteten Prozessoren ausgeführt wird, kann die Funktionalität stark erweitert werden.

Die Zuverlässigkeit dieser Hardware-Software-Systeme wird von mehreren Faktoren beeinflusst. Ausfallmechanismen in SoCs umfassen Fehler, die während dem Entwurf, der Fertigung und der Laufzeit auftreten. Dabei können Entwurfsfehler

sowohl in Software und Hardware auftreten, während Fertigungsfehler nur die Hardware schädigen. Fehler zur Laufzeit können den Zustand einer ausgeführten Berechnung oder Operation beeinflussen.

Zahlreiche Maßnahmen wurden zur Steigerung der Zuverlässigkeit von Hardware und Software vorgeschlagen. Die Verwendung von Redundanz verursacht jedoch erhöhte Kosten bezüglich der Chipfläche, Verlustleistung oder Systemleistung. Zur effizienten Zuverlässigkeitssteigerung ist es deshalb nötig, kritische Systemkomponenten zu identifizieren und zur Härtung oder für Fehlertoleranz-Maßnahmen auszuwählen. Ein Systemmodell für die akkurate Zuverlässigkeitsschätzung ist notwendig, um diese kritischen Komponenten zu identifizieren und mögliche Entwurfsalternativen zu bewerten.

Dabei müssen die funktionalen Aspekte der Zielanwendung des Systems in die Zuverlässigkeitsbewertung einfließen, um Eigenschaften wie Nutzungs- und Auslastungsprofile und algorithmische Eigenschaften der Anwendung zu berücksichtigen. Andernfalls ergäbe sich ein ungenauer Wert der Systemzuverlässigkeit, der entweder zu optimistisch oder zu konservativ ist.

Der Entwurf solcher Hardware-Software-Systeme geht von einer Verhaltensbeschreibung, z.B. einem Datenflußgraphen, aus. Die Struktur der Hardware kann als Architekturgraph beschrieben werden, der die verschiedenen Hardwareblöcke und die Verbindungen zur Kommunikation darstellt [3].

Die Zuordnung der Operationen im Datenflußgraph zu Ressourcen des Systems wird im Mapping- und Binding-Schritt durchgeführt [4]. Operationen können dabei mittels Hardware oder Software, die von Prozessoren ausgeführt wird, implementiert werden.

Die Zuverlässigkeit einer speziellen Implementierung einer Operation hängt dann von den Zuverlässigkeitseigenschaften der zugrundeliegenden Ressource und der Art der Operation ab. Die Art der Operation bestimmt das Nutzungsprofil der Ressource, also z.B. die Laufzeit auf der Ressource oder ihre Auslastung. Ebenso unterscheidet sich das Nutzungsprofil signifikant, wenn eine Operation mittels Software oder vollständig in Hardware implementiert wird.

Wird eine Ressource des Systems zur Implementierung mehrerer Operationen verwendet, dann steigt die Auslastung dieser Ressource und entsprechend erhöht sich ihre Kritikalität hinsichtlich der Systemzuverlässigkeit. Weiterhin können die Zuverlässigkeiten der sich ergebenden Implementierungen nicht mehr als statistisch unabhängig von einander betrachtet werden, da die Fehler in der Ressource während der Ausführung einer Operation korrelieren. Die Annahme statistischer Unabhängigkeit führt hier zu einem ungenauen Wert der Systemzuverlässigkeit.

Fehler, die während der Ausführung einer Operation auftreten, können von darauf folgenden Operationen maskiert oder propagiert werden. Ein Fehler tritt auf Systemebene auf, wenn er bis zu den Primärausgängen propagiert und beobachtbar wird. Fehlermaskierung kommt dabei auf verschiedenen Ebenen vor: Auf elektrischer und logischer Ebene und auf Architektur- und Verhaltensebene. Auf Verhaltensebene werden Fehler beispielsweise maskiert, wenn Ergebnisse aus vorangehenden Operationen gerundet oder nur zum Teil verwendet werden. Im Datenflussgraph lassen sich aus den Datenabhängigkeiten der Operationen das Kommunikationsverhalten und mögliche Propagierungspfade von Fehlern ablesen.

Aber auch die Datenübertragung selbst kann durch Fehler wie Übersprechen oder Rauschen beeinträchtigt werden [5], [6]. Folglich müssen Kommunikationsvorgänge ebenso im Zuverlässigkeitsmodell berücksichtigt werden.

Ein umfassendes Systemmodell für die Zuverlässigkeitsbewertung muss neben der Hardware und Software auch das Verhalten des Systems abbilden. Dies schließt neben Fehlerpropagierung auf Anwendungsebene auch Abhängigkeiten ein, die durch Mapping-Informationen gegeben sind.

Der folgenden Abschnitt diskutiert bekannte Modelle zur Zuverlässigkeitsbewertung von Hardware-Software-Systemen. Abschnitt 3 schlägt eine Modellierungsmethodik vor, die bestehende Zuverlässigkeitsmodelle um Abhängigkeiten und funktionale Aspekte auf Anwendungsebene erweitert. Abschnitt 4 zeigt anhand von Experimenten die hohe Relevanz dieser Erweiterungen. Eine kurze Zusammenfassung und ein Ausblick auf folgende Arbeiten schließen diese Arbeit ab.

2 Vorarbeiten

Hardware-Software-Systeme werden hierarchisch modelliert, um den Aufbau aus Komponenten abzubilden. Für die Zuverlässigkeitsbewertung der einzelnen Komponenten werden jeweils passende Modelle verwendet, um die Eigenschaften der Hardware-Struktur und der Software zu berücksichtigen. Im Folgenden werden zunächst einige Modelle zur Zuverlässigkeitsbewertung von Software- und Hardware-Komponenten vorgestellt, gefolgt von Modellen, die das Gesamtsystem bewerten.

Die Modellierung von Hardware und Software unterscheidet sich, da die zugrundeliegenden Fehlermodelle unterschiedlicher Natur sind. Betrachtet man Software, so werden Ausfälle durch Entwurfsfehler begründet. Bei Hardware findet man zusätzlich physikalische Ausfallmechanismen wie Fertigungsfehler, intermittierende und transiente Fehler. In der Literatur werden verschiedene Modelle für Software und Hardware vorgeschlagen, die die jeweiligen Ausfallmechanismen berücksichtigen.

Zur Bewertung der Zuverlässigkeit von Software werden häufig Software Reliability Growth Modelle verwendet [7]: Da während der Lebensdauer der Software immer mehr Entwurfsfehler gefunden und behoben werden, steigt die Zuverlässigkeit an [8]. Anhand von Fehler- und Ausfalldaten aus dem Entwicklungsprozess und Komplexitätsmaße für Software lassen sich diese Modelle kalibrieren.

Umfangreiche Softwaresysteme bestehen aus Komponenten, die miteinander kommunizieren und interagieren. Die Zuverlässigkeit solcher Systeme lässt sich aus der Zuverlässigkeit der einzelnen Komponenten und ihrem Fehlermaskierungs- und propagierungsverhalten ableiten. Dabei hängt die Maskierung bzw. Propagierung von Fehlern von den Kommunikationsmustern im Gesamtsystem ab [9]. Der Ansatz in [10] analysiert verschiedene Eigenschaften von Fehlern in Komponenten, insbesondere auch Fehlerpropagierung und die Kritikalität eines Fehlers für die Gesamtfunktionalität.

Ebenso kann die Zuverlässigkeit von Hardware durch Entwurfsfehler beeinträchtigt werden. Wichtiger ist hier jedoch die korrekte Modellierung physikalischer Ausfallmechanismen, wie permanenten oder intermittierenden Fehlern aufgrund von unerkannten Fertigungsfehlern oder transienten Fehlern, verursacht z.B. durch Partikeleinschläge.

Die strukturelle Anfälligkeit gegenüber permanenten Fehlern kann durch Analyse der Testbarkeit der Schaltung erfolgen. Die Anfälligkeit bezüglich transienter Fehler wurde auf Gatterebene [11], [12], [13], Register-Transfer-Ebene [14] und speziell für Prozessoren auf Architekturebene [15], [16] untersucht. Eingebettete Systeme werden oft auf FPGAs implementiert. Für diese Technologie analysiert [17] die Empfindlichkeit des Entwurfs für transiente Fehler.

Auf Systemebene kann Hardware und Software nicht mehr separat bewertet werden und Zuverlässigkeitsmodelle müssen Hardware, Software und ihre Interaktion umfassen. Sowohl Fehlerbäume als auch Zuverlässigkeitsblockdiagramme [18] können zur Modellierung von Hardware-Software-Systemen verwendet werden. In [19] werden Zuverlässigkeitsblockdiagramme zur Darstellung der Hardware- und Software-Komponenten benutzt. Die Zuverlässigkeit einer einzelnen Komponente wird dabei von dem entsprechenden zugrunde liegenden Modell bestimmt. Verfahren, die wie in [20] auf Fehlerbäumen basieren, analysieren Ausfälle des Gesamtsystem in Abhängigkeit vom Fehlverhalten der Hardware- und Software-Komponenten. In [21] werden die entsprechenden Fehlerbäume automatisch aus einer SystemC-Beschreibung des Systems extrahiert.

Die Erweiterung von Fehlerbäumen und Zuverlässigkeitsblockdiagrammen zu dynamischen Fehlerbäumen und dynamischen Blockdiagrammen vergrößert die Modellierungsmöglichkeiten, so dass auch Abhängigkeiten zwischen den modellierten Einheiten dargestellt werden können [22], [23], [24]. So können z.B. Fehler modelliert werden, die weitere Fehler in anderen Komponenten auslösen. In [25] werden Zuverlässigkeitsblockdiagramme explizit um bestimmte Ausfallabhängigkeiten erweitert, um Fehlerpropagierung in die Zuverlässigkeitsbewertung mit einzubeziehen.

Die Arbeiten [26], [27] betrachten strukturelle Abhängigkeiten, die aufgrund von mehrfach und gemeinsam benutzten Ressourcen im System auftreten. Während diese beiden Publikationen strukturelle Abhängigkeiten analysieren, die sich durch das Verhalten des Systems ergeben, führt [28] eine empirische Zuverlässigkeitsbewertung anhand von Fehlerinjektionen durch und untersucht die Fehlerpropagierung zwischen strukturellen Komponenten.

Im Mapping-Schritt des Systementwurfs werden Operationen auf Ressourcen des Systems abgebildet [4]. Entsprechend schlagen [29], [30] und [31] vor, das System als Operationen oder Aufgaben und Ressourcen zu modellieren. Dabei ist es möglich, Operationen zu replizieren oder verschiedenen Ressourcen zuzuweisen.

Im Folgenden greifen wir die Konzepte aus [29] und [30] auf und schlagen Erweiterungen des Modells vor, um Abhängigkeiten zwischen den modellierten Einheiten und Verhaltensinformationen wie Fehlermaskierung auf Anwendungsebene zu berücksichtigen.

3 Hardware-Software-Modell

Das hier vorgeschlagene Zuverlässigkeitsmodell für Hardware-Software-Systeme basiert auf Informationen, die in frühen Entwurfsphasen vorliegen. Die Funktions- oder Verhaltensbeschreibung des Systems ist üblicherweise als Ablauf von Operationen gegeben und kann, wie in Abbildung 1 (a) als Datenflussgraph dargestellt werden. Die strukturelle Beschreibung des Systems stellt die verfügbaren Hardwareressourcen des Systems wie z.B. in Abbildung 1 (b) dar [30].

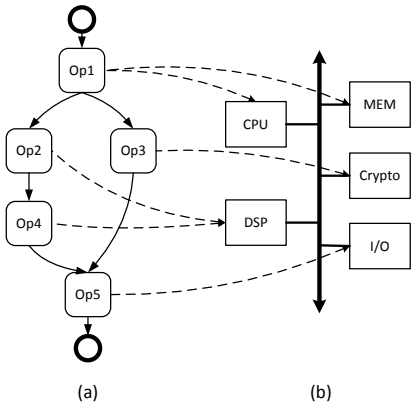

Bild 1 Systemmodell (a) Verhaltenssicht als Datenflussgraph, (b) Strukturelle Sicht mit Hardwareressourcen; Abbildung von Operationen auf Ressourcen.

Der Datenflussgraph $G_{df} = (V_o, E_o)$ besteht aus Knoten $v \in V_o$, die Operationen repräsentieren, und der Menge gerichteter Kanten $E_o \subseteq (V_o \times V_o)$, die Datenabhängigkeiten zwischen Operationen beschreiben. Dadurch beschreiben die Kanten E_o auch mögliche Pfade der Fehlerpropagierung im funktionalen Modell.

Die Transparenz einer Operation bezüglich Fehler in den Eingabedaten wird durch ein Attribut dem entsprechendem Knoten zugeordnet. Dabei entspricht die Transparenz einer Operation der bedingten Wahrscheinlichkeit, einen Fehler in der Ausgabe der Operation zu beobachten, falls ein Fehler in der Eingabe vorliegt [10]. Operationen mit hoher Transparenz propagieren also Fehler in der Eingabe mit hoher Wahrscheinlichkeit, während niedrige Transparenz einer erhöhten Fehlermaskierung in der Operation entspricht. Bei der Verschlüsselung von Daten, beispielsweise, liegt eine hohe Transparenz vor, d.h. Fehler werden mit hoher Wahrscheinlichkeit propagiert, während bei verlustbehafteter Kompression eine geringere Transparenz vorliegt. Zur Bestimmung der Transparenz einer Operation lassen sich verschiedene Verfahren verwenden. Als erste Näherung kann der Anteil der Information bestimmt werden, die durch eine Funktion oder Operation von der Eingabeseite zur Ausgabeseite übertragen wird (Information Transmission Coefficient, [32]). Genauere Ergebnisse liefern Testbarkeitsmaße wie z.B. [33], die die Analyse von Datenflussgraphen auf funktionaler Ebene erlauben. Empirisch kann die Transparenz auch in einer Verhaltenssimulation der Operation mit Fehlerinjektion abgeschätzt werden.

Die strukturelle Sicht stellt die unterschiedlichen Arten der Hardwareressourcen des Systems dar und umfasst auch die Kommunikationsverbindungen zwischen den Ressourcen, z.B. Busse oder ein Network-on-Chip. Der Strukturgraph $G_s = (V_r, E_c)$ besteht entsprechend aus Knoten $v \in V_r$, die Ressourcen modellieren und Kanten $E_c \subseteq (V_r \times V_r)$, die Kommunikationsverbindungen zwischen den Ressourcen modellieren.

Darüber hinaus wird die Zuverlässigkeit der Ressourcen und Kommunikationsverbindungen in Abhängigkeit von ihrer Nutzung beschrieben. Dazu weist die Funktion $\rho : (V_r \cup E_c, u, t) \rightarrow [0, 1]$ einer Ressouce ihre Zuverlässigkeit abhängig von dem Nutzungsprofil u und der Laufzeit t der Operation zu. Das Nutzungsprofil einer Operation gibt an, zu welchem Grad die Ressource verwendet wird. Als Beispiel sei eine Operation gegeben, die lediglich einen kleinen Teil des vorhandenen Systemspeichers verwendet. Ähnlich lassen sich Operationen für Prozessoren unterscheiden, die auf Ganzzahlarithmetik beruhen und eine vorhandene Fliesskommaeinheit nicht nutzen.

Im Mapping-Schritt des Entwurfs wird das Verhaltensmodell mit der Strukturbeschreibung verbunden. Für jede Operation wird eine Implementierung ausgewählt, so dass Randbedingungen wie Verlustleistung, Chipfläche und Geschwindigkeit des Systems erfüllt werden.

Das Mapping von Operationen auf Ressourcen ist eine Menge von Relationen $M \subseteq (V_o \times V_r)$ zwischen Knoten aus dem Datenflussgraphen und Knoten aus der strukturellen Beschreibung. Verschiedene Implementierungen einer Operation führen zu unterschiedlichen Nutzungsprofilen auf den Ressourcen, z.B. der Laufzeit auf einem Hardwarebeschleuniger verglichen mit einer Software-basierten Implementierung. Entsprechend variieren auch die Zuverlässigkeitseigenschaften der Implementierungen. Deshalb wird jedem Mapping $m \in M$ ein Nutzungsprofil und die geschätzte Laufzeit zugewiesen.

Ähnlich der Arbeit in [30] werden der Datenflussgraph G_{df}, der Strukturgraph G_S und die Mappingvorschrift M als Grundlage des Zuverlässigkeitsmodell verwendet. Das System wird nun mittels Implementierungen und dem Kommunikationsverhalten modelliert, um sowohl die strukturellen als auch die funktionalen Aspekte mit einzubeziehen.

Die Zuverlässigkeit einer einzelnen Implementierung wird von unterschiedlichen Faktoren beeinflusst: Der Zuverlässigkeit der verwendeten Ressource unter Berücksichtigung eventuell vorhandener Fehlertoleranzmaßnahmen, der Art der Operation und dem Nutzungsprofil. Die Zuverlässigkeit von Hardwareressourcen kann dabei von Technologiedaten und der abgeleiteten Verteilungsfunktion für Fehler bestimmt werden [34]. Ebenso lassen sich bekannte Verfahren zur strukturellen Zuverlässigkeitsbewertung wie [11] verwenden. Im Hinblick auf die Analyse von Software bieten sich die eingangs erwähnten Software Reliability Growth Modelle an [8], [7].

Auch die Art der Operation, ihre Laufzeit und ihr Nutzungsprofil bezüglich einer bestimmten Ressource beeinflussen die Zuverlässigkeit der Implementierung. Diese Kenndaten können mit Werkzeugen während der Entwurfsraumexploration [35] oder durch Verhaltenssimulationen geschätzt werden.

Im Beispiel aus Abbildung 1 werden Operation 1 und Operation 2 auf die gleiche Ressource (DSP) abgebildet. Die Zuverlässigkeiten der zwei Implementierungen korrelieren miteinander, da Fehler in der Ressource beide Implementierungen beeinträchtigen können. Im Modell werden solche Beziehungen der Implementierungen annotiert, um die durch das Verhalten gegebene Abhängigkeiten zwischen ihnen anzuzeigen.

Die Kommunikation zwischen Operationen des Systems wird durch die Datenabhängigkeiten E_o im Datenflussgraph beschrieben. Im Strukturgraphen beschreiben die Kanten E_c die vorhandenen Verbindungen für die Kommunikation zwischen Ressourcen und ihre Zuverlässigkeitseigenschaften. Datenübertragungen werden hier auf die entsprechende Ressourcen ab-

gebildet und müssen auch im Zuverlässigkeitsmodell wiedergegeben werden.

Mit den vorgeschlagenen Erweiterungen ergibt sich als Zuverlässigkeitsmodell ein Systemgraph $G_I = (V_I, E_I)$, dessen Knoten $v \in V_I$ die Implementierungen von Operationen und Kommunikationsvorgänge zwischen Implementierungen repräsentieren. Die Menge der gerichteten Kanten $E_I \subseteq (V_I \times V_I)$ bezeichnet den Datenfluss zwischen den Knoten und damit mögliche Pfade der Fehlerpropagierung. Dieser Graph verbindet nun die relevanten strukturellen und funktionalen Informationen der Systembeschreibung. Abbildung 2 zeigt den sich ergebenden Systemgraphen für das Beispiel aus Abbildung 1.

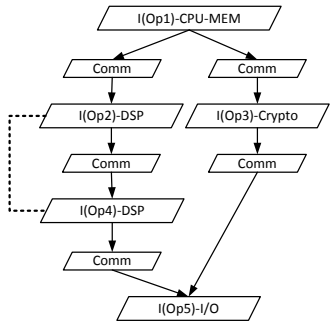

Bild 2 Systemgraph mit Implementierungen, Kommunikationsvorgängen und Abhängigkeiten zwischen Implementierungen.

Dieses Modell kann als Basis zur Zuverlässigkeitsbewertung auf Anwendungsebene benutzt werden. Die Zuverlässigkeit kann analytisch oder empirisch durch Fehlerinjektion und Simulation oder Emulation [36] ermittelt werden. Außerdem erlaubt die Zuverlässigkeitsschätzung die Bestimmung kritischer Implementierungen bezüglich der Systemzuverlässigkeit.

Die Zuverlässigkeit des Systems kann dann auf verschiedene Wege verbessert werden, je nach Ursache der unzureichenden Zuverlässigkeit, sei es hohe Transparenz einer Operation, unzuverlässige Ressourcen oder spezielle Nutzungsprofile. Maßnahmen zur Zuverlässigkeitssteigerung umfassen u.a. selektives Härten, Fehlertoleranzmaßnahmen für Hardware und Software sowie ein angepasstes Mapping auf Anwendungsebene.

4 Empirische Untersuchung funktionaler Aspekte

Anhand eines einfachen Systems wird im Folgenden die Relevanz der beschriebenen Parameter aus dem Verhaltensmodell auf die Zuverlässigkeit auf Anwendungsebene aufgezeigt. Hierbei wird insbesondere der starke Einfluss der Fehlermaskierung auf die Systemzuverlässigkeit deutlich.

Das gewählte Beispielsystem basiert auf einer modularen Implementierung des JPEG-Kompressionsalgorithmus. Durch Fehlerinjektion und Simulation einer einzelnen Komponente des Entwurfs wurde die Anfälligkeit dieser Komponente gegenüber Fehlern untersucht. Weiterhin wurde die Auswirkung von Fehlern in dieser Komponente auf die gesamte Kompression eines Bildes ausgewertet.

Der JPEG-Kompressionsalgorithmus kann in fünf Schritte zerlegt werden:

1) Konvertierung vom RGB-Farbraum zum YCbCr-Farbraum (Darstellung als Helligkeits- und Chrominanzwerte)
2) Unterabtastung der Chrominanz-Anteile
3) Zweidimensionale diskrete Kosinustransformation (2D-DCT) von Helligkeits- und Chrominanzwerten auf 8x8 Pixelblöcken
4) Quantelung der DCT-Koeffizienten im Frequenzbereich
5) Komprimierung der gequantelten Koeffizienten durch Lauflängenkodierung.

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. Dadurch erreicht der JPEG-Algorithmus ein sehr hohes Kompressionsverhältnis.

Die vorgestellte Modellierung wird nun auf eine Implementierung des JPEG-Algorithmus angewendet. Untersucht werden die Modellierung von Struktur und Verhalten, die Transparenz von Operationen, der Einfluss der ausgeführten Funktion auf die Zuverlässigkeit und die Bewertung der Kritikalität eines Ausfalls auf Anwendungsebene.

4.1 Modellierung von Struktur und Verhalten

Entsprechend Kapitel 3 kann der JPEG-Algorithmus als Datenflussgraph wie in Abb. 3 a) dargestellt werden. Die Struktur der Hardware ist in Abb. 3 b) wiedergegeben. Die Operationen im Datenflussgraph werden hier direkt auf Hardwareblöcke abgebildet.

4.2 Transparenz von Operationen

Im funktionalen Modell des JPEG-Algorithmus können Fehler entlang der Datenflusskanten propagiert werden. Die Wahrscheinlichkeit der Fehlerpropagierung durch eine Operation hängt von der Transparenz der Operation ab. Anhand von Fehlerinjektionen in der 2D-DCT-Implementierung zeigen wir den Einfluss von Transparenzen auf die Fehlermaskierung auf.

Dazu wurde die Implementierung für eine generische Gatterbibliothek synthetisiert und mit Mentor ModelSim 6.1 simuliert. In den Experimenten wird vom

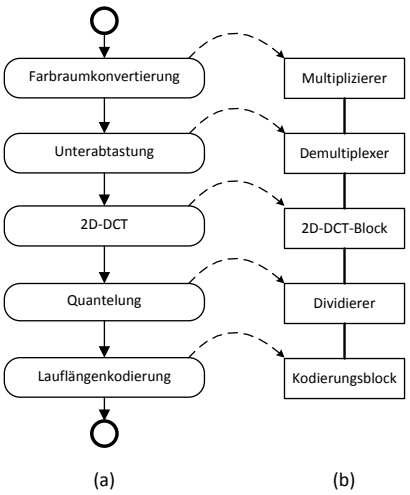

Bild 3 JPEG-Kompression als (a) Datenflussgraph und (b) Systemstruktur.

Einzel-Haftfehlermodell ausgegangen. Insgesamt ergeben sich etwa 20000 Haftfehler in der 2D-DCT-Implementierung, von denen 15% zufällig zur Fehlerinjektion ausgewählt wurden.

Zunächst wurde die Empfindlichkeit der 2D-DCT-Implementierung bezüglich der ausgewählten Fehler untersucht. Dazu wurden Fehler injiziert und die Funktion der 2D-DCT-Implementierung (vollständige Transformation eines zufälligen 8x8 Pixelblocks) in einer Testbench evaluiert. Die berechneten DCT-Koeffizienten wurden dann mit den korrekten Koeffizienten aus einem fehlerfreien Durchlauf verglichen.

Von 2926 Fehlerinjektionen in der 2D-DCT-Implementierung hatten 718 Fehler keine Auswirkung auf die Berechnung, d.h. beinahe 25% der Haftfehler wurden innerhalb der Implementierung maskiert und lediglich 75% der Fehler manifestierten sich tatsächlich im Ergebnis der Berechnung.

In einem zweiten Experiment wurde die Auswirkung der Fehler in der 2D-DCT-Implementierung auf der Anwendungsebene analysiert. Die Menge der injizierten Fehler deckt sich mit den Fehlern aus dem ersten Experiment, das die 2D-DCT-Implementierung isoliert betrachtete. Ein einzelner Versuch besteht aus der Fehlerinjektion in die 2D-DCT-Implementierung und der Durchführung einer Bildkompression eines 64x48 Pixel großen Bildes. Das komprimierte Ergebnis wird dann mit dem korrekten Ergebnis einer fehlerfreien Implementierung verglichen. Abbildung 4 zeigt ein Ausgangsbild und ein fehlerhaftes Bild, das durch Fehlerinjektion und Simulation berechnet wurde.

910 der 2926 injizierten Fehler haben überhaupt keine Auswirkung auf das komprimierte Bild. Auf Anwendungsebene führen also nur etwa 69% der Fehler zu einem beobachtbaren Systemfehler, wohingegen 75% der Fehler beobachtbar sind wenn die 2D-DCT-Implementierung isoliert betrachtet wird.

Bild 4 Korrektes und fehlerhaftes Bild aufgrund eines Fehlers in der 2D-DCT-Implementierung.

Einige Fehler der 2D-DCT-Implementierung werden maskiert. Die Operationen, die der 2D-DCT-Operation folgen, Quantelung und Lauflängenkodierung, sind also nicht völlig transparent.

4.3 Einfluss der ausgeführten Funktion auf die Zuverlässigkeit

Die Transparenz des Quantelungsschritts wird noch weiter reduziert, wenn das Bild mit einem erhöhten Kompressionsfaktor im JPEG-Algorithmus verarbeitet wird. Der Kompressionsfaktor bestimmt die Stärke der Quantelung der DCT-Koeffizienten und führt damit zu einer verminderten Wahrscheinlichkeit der Fehlerpropagierung. Betrachten wir wieder die Fehler in der 2D-DCT-Implementierung, die keine Auswirkung auf das Ergebnis haben, so steigt ihre Anzahl bei starker Kompression auf 1237 Fehler an.

Die spezielle, auf dem System ausgeführte Funktion, hier die Kompression mit unterschiedlicher Stärke, beeinflusst das Verhalten des Systems auf der Anwendungsebene. Durch die Veränderung der Transparenz der Quantelungsoperation steigt die Fehlermaskierung um mehr als 26% an.

4.4 Bewertung der Kritikalität von Ausfällen

Zum Vergleich der Resultate und zur Bewertung der Fehlerauswirkung auf Anwendungsebene wurde weiterhin die Ähnlichkeit der berechneten Bilder entsprechend der Wahrnehmung des Menschen bestimmt, da ein einfacher bit-weiser Vergleich nicht geeignet ist, die Auswirkung realistisch zu bewerten. Zur Berechnung der Ähnlichkeit zweier Bilder wurde die Programmsammlung ImageMagick [37] verwendet.

Die Auswirkung eines Fehlers wird als Störabstand (Signal-to-Noise Ratio, SNR) zwischen den berechneten Bildern gemessen. Ein unbeschränkter SNR-Wert bezeichnet die Identität von zwei Bildern. SNR-Werte von etwa 40 dB weisen auf leichte, aber erkennbare Unterschiede in den Bildern hin. Bilder, die sich völlig unterscheiden, haben typischerweise SNR-Werte von unter 10 dB. Der Störabstand des fehlerhaften Bildes zum korrekten Ergebnis in Abbildung 4 beträgt 21 dB.

In Tabelle 1 sind die Ergebnisse für die zwei Versuchsreihen mit niedrigem und erhöhtem Kompressionsfaktor dargestellt. Die Tabelle zeigt die Zahl der

injizierten Fehler, die zu einem Bild mit dem angegebenen Störabstand—verglichen mit dem korrekten Ergebnis—führen.

In der Zeile $SNR = 0$ sind die kritischen Fehler zusammengefasst, die eine Störung der kompletten Berechnung verursachen, so dass die Kompression nicht abgeschlossen wird. Die Zahl dieser kritischen Fehler wird bei starker Kompression um den Faktor acht reduziert.

Wenn die menschliche Wahrnehmung in die Bewertung der Fehlerwirkung einfließt und nur Bilder mit einem Störabstand kleiner 40 dB als beobachtbare Systemfehler gezählt werden, dann ergeben lediglich 47% der Fehler in der 2D-DCT-Implementierung einen wahrnehmbaren Ausfall. Eine detaillierte Untersuchung des Einflusses von permanenten und transienten Fehlern unter Berücksichtigung der menschlichen Wahrnehmung findet sich in [38].

SNR (dB)	Anzahl Fehler Niedrige Komp.	Anzahl Fehler Hohe Komp.
0	227	30
1.3	0	0
2.0	0	0
3.1	0	0
4.8	0	0
6.0	10	7
7.5	211	271
9.3	267	272
11.6	155	261
14.6	70	96
18.2	80	93
22.7	123	102
28.4	110	100
35.5	131	151
44.4	239	178
55.5	260	127
69.4	132	0
∞	910	1237

Tabelle 1 Häufigkeit verschiedener Störabstände (SNR) durch Fehlerinjektion bei niedrigem und hohem Kompressionsfaktor.

Die 2D-DCT-Ressource des Beispielsystems ist der größte Hardwareblock des Entwurfs. Eine wahllose Anwendung von Fehlertoleranzmaßnahmen verursacht hier hohe Kosten. Es ist also notwendig, die Kritikalität der jeweiligen Ressource bezüglich des Gesamtsystems akkurat zu bestimmen. Die Simulationsergebnisse zeigen, dass dabei die ausgeführte Funktion beachtet werden muss, da sie einen direkten Einfluss auf die Zuverlässigkeit des Systems hat.

5 Zusammenfassung

Zur Zuverlässigkeitsbewertung von Hardware-Software-Systemen muss sowohl das Verhalten als auch die Hardware-Struktur betrachtet werden. In dieser Arbeit wurde eine Methodik zur Modellierung beschrieben, in die grundlegenden Einflüsse auf die Zuverlässigkeit einfließen. Dies umfasst neben der zugrundeliegenden Zuverlässigkeit der Hardware, dem Mapping und Nutzungsprofilen auch Abhängigkeiten und Fehlermaskierung auf Systemebene.

Der Verwendungszweck des erstellten Modells liegt sowohl in der empirischen Zuverlässigkeitsbewertung durch Fehlerinjektion und Simulation als auch in der analytischen Auswertung mittels statistischer Methoden. Die präsentierten Simulationsergebnisse zeigen die Notwendigkeit, Struktur und Verhalten gemeinsam zu modellieren und zu analysieren.

Aufbauend auf dieser Modellierung werden in künftigen Arbeiten weitere Systeme evaluiert. Außerdem sollen statistische Methoden zur Zuverlässigkeitsbewertung und Maßnahmen zur Steigerung der Zuverlässigkeit untersucht werden.

6 Danksagung

Diese Arbeit entstand im Rahmen der DFG Forschergruppe 460 unter Wu 245/3-3.

7 Literatur

[1] I. Koren and C. M. Krishna, *Fault tolerant systems*, 1st ed. Morgen Kaufmann Publishers, 2007.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[3] W. H. Wolf, "An architectural co-synthesis algorithm for distributed, embedded computing systems," in *Readings in hardware/software co-design*. Norwell, MA, USA: Kluwer Academic Publishers, 2002, pp. 338–349.

[4] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System level design: Orthogonalization of concerns and platform-based design," *IEEE Transactions on Computer-Aided Design*, vol. 19, no. 12, December 2000.

[5] N. Shanbhag, "Reliable and efficient system-on-chip design," *IEEE Transactions on Computers*, vol. 37, no. 3, pp. 42–50, March 2004.

[6] F. Caignet, S. Delmas-Bendhia, and E. Sicard, "The challenge of signal integrity in deep-submicrometer CMOS technology," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 556–573, April 2001.

[7] W. Blischke and D. Murthy, Eds., *Case studies in reliability and maintenance*, 1st ed. John Wiley and Sons, Inc., 2003.

[8] A. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. 28, no. 1, pp. 206–211, 1979.

[9] S. Yacoub, B. Cukic, and H. Ammar, "A scenario-based reliability analysis approach for component-based software," *IEEE Transactions on Reliability*, vol. 53, no. 4, pp. 465–480, December 2004.

[10] M. Hiller, A. Jhumka, and N. Suri, "EPIC: profiling the propagation and effect of data errors in software," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 512–530, 2004.

[11] M. Sonza Reorda and M. Violante, "Accurate and efficient analysis of single event transients in VLSI circuits," in *Proceedings of 9th IEEE International On-Line Testing Symposium*, July 2003, pp. 101–105.

[12] S. Hellebrand, C. Zoellin, H.-J. Wunderlich, T. Coym, S. Ludwig, and B. Straube, "A refined electrical model for drift processes and its impact on SEU prediction," in *Proceedings of the 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, September 2007, pp. 50–58.

[13] A. Nieuwland, S. Jasarevic, and G. Jerin, "Combinational logic soft error analysis and protection," in *Proceedings of 12th IEEE International On-Line Testing Symposium*, July 2006, pp. 6–11.

[14] A. Ammari, R. Leveugle, M. Sonza-Reorda, and M. Violante, "Detailed comparison of dependability analyses performed at RT and gate levels," in *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*, November 2003, pp. 336–343.

[15] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 29–41.

[16] X. Li, S. Adve, P. Bose, and J. Rivers, "SoftArch: an architecture-level tool for modeling and analyzing soft errors," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2005, pp. 496–505.

[17] O. Heron, T. Arnaout, and H.-J. Wunderlich, "On the reliability evaluation of SRAM-based FPGA designs," in *International Conference on Field Programmable Logic and Applications*, August 2005, pp. 403–408.

[18] K. S. Trivedi, *Probability and statistics with reliability, queuing and computer science applications*, 2nd ed. John Wiley and Sons, Inc., 2002.

[19] J. Cano and D. Rios, "Reliability forecasting in complex hardware/software systems," in *Proceedings of the First International Conference on Availability, Reliability and Security*, 2006, pp. 300–304.

[20] L. Kaufman, J. Bechta Dugan, R. Manian, and K. Kumar Vemuri, "System reliability analysis of an embedded hardware/software system using fault trees," in *Proceedings of Annual Reliability and Maintainability Symposium*, 1999, pp. 135–141.

[21] H. Zarandi and S. Miremadi, "Fault tree analysis of embedded systems using SystemC," in *Proceedings of Annual Reliability and Maintainability Symposium*, 2005, pp. 77–81.

[22] J. Dugan, S. Bavuso, and M. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems," *IEEE Transactions on Reliability*, vol. 41, no. 3, pp. 363–377, Sep 1992.

[23] S. Distefano and A. Puliafito, "Dynamic reliability block diagrams vs dynamic fault trees," in *Proceedings of Annual Reliability and Maintainability Symposium*, 2007, pp. 71–76.

[24] H. Boudali, P. Crouzen, and M. Stoelinga, "Dynamic fault tree analysis using input/output interactive markov chains," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2007, pp. 708–717.

[25] C. Trinitis and M. Walter, "How to integrate inter-component dependencies into combinatorial availability models," in *Proceedings of Annual Reliability and Maintainability Symposium*, 2004, pp. 226–231.

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

[27] C. Betous-Almeida and K. Kanoun, "Stepwise construction and refinement of dependability models," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2002, pp. 515–524.

[28] M. Hosseinabady, M. H. Neishaburi, Z. Navabi, A. Benso, S. D. Carlo, P. Prinetto, and G. D. Natale, "Analysis of system-failure rate caused by soft-errors using a UML-based systematic methodology in an SoC," in *Proceedings of the 13th IEEE International On-Line Testing Symposium*, 2007, pp. 205–206.

[29] A. Jhumka, S. Klaus, and S. Huss, "A dependability-driven system-level design approach for embedded systems," in *Proceedings of the Design, Automation and Test in Europe Conference*, 2005, pp. 372–377.

[30] M. Glass, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich, "Reliability-aware system synthesis," in *Proceedings of the Design, Automation and Test in Europe Conference*, April 2007, pp. 1–6.

[31] A. Israr and S. A. Huss, "Specification and design considerations for reliable embedded systems," in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2008.

[32] G. Koob, "An abstract complexity theory for boolean functions," PhD Thesis, University of Illinois at Urbana-Champaign, 1987.

[33] K. Thearling and J. Abraham, "An easily computed functional level testability measure," in *Proceedings of the International Test Conference*, August 1989, pp. 381–390.

[34] I. Koren and Z. Koren, "Defect tolerance in VLSI circuits: techniques and yield analysis," *Proceedings of the IEEE*, vol. 86, no. 9, pp. 1819–1838, September 1998.

[35] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, "Efficient power co-estimation techniques for system-on-chip design," in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2000, pp. 27–34.

[36] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *IEEE Transactions on Computers*, vol. 30, no. 4, pp. 75–82, 1997.

[37] "ImageMagick," 2008. [Online]. Available: http://www.imagemagick.org

[38] D. Nowroth, I. Polian, and B. Becker, "A study of cognitive resilience in a JPEG compressor," in *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN)*, June 2008.