Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Integrierte Systeme

# Verfahren zur Abschätzung des zeitlichen Maskierungsverlaufs transienter Fehler in digitalen Schaltungen

**Robert Hartl**

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktor-Ingenieurs**

genehmigten Dissertation.

| | |
|---|---|
| Vorsitzender: | Univ.-Prof. Dr.-Ing. Ulf Schlichtmann |
| Prüfer der Dissertation: | 1. apl. Prof. Dr.-Ing. habil. Walter Stechele |
| | 2. Univ.-Prof. Dr. Wolfgang Rosenstiel (Eberhard Karls Universität Tübingen) |

Die Dissertation wurde am 08.05.2014 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 25.08.2014 angenommen.

# Abstract

Modern integrated circuits are susceptible to ionized particles causing temporary current pulses in the sensitive area of a semiconductor. If such a particle hits a memory cell, the digital memory content might be inverted and causing a so called SEU (Single Event Upset), but no permanent damage of the device (Soft errors). Four major effects, logic masking, timing derating, information lifetime and transitive effects, can cause error masking and are evaluated in detail. Several methods to quantify error masking and failure rate have been examined for their working principle, effectiveness and precision.

Special error effects caused by SEUs in FPGAs are explained as well as techniques to protect them from errors. Configuration data was analyzed to derive the amount of SRAM dedicated to each FPGA function and allow an estimation of the design error rate.

Based on the capabilities and disadvantages of previously discussed error masking estimation methods, a completely new algorithm was proposed. Backwards Analysis (BA) uses a simple circuit simulation step followed by an analysis of the time-reversed simulation data generated before. This approach allows to cover the four major error masking effects with very high accuracy and extremely low, linear runtime. The algorithm can be applied to circuits in gate-list representation and produces results in level of detail from average error masking through to a sensitivity waveform for each memory in the circuit. The weaknesses and limitations of the new algorithm were shown, based on this, further optimization concepts were introduced.

The working principle and the correctness of the generated results was verified in several ways. Theoretical, in-detail analysis of small application scenarios is followed by short interval fault injection simulation of large usecases. Small deviations from correct results were found, which were caused by simplified boundary conditions. A large microcontroller circuit running a custom program was evaluated in detail, showing an unexpected low sensitivity of some register file contents due to operand forwarding capablities of the CPU pipeline. Only 4% of the whole systems memory in this scenario can contribute to SEU based error rate. Tests covering a large range of circuit sizes confirmed the runtime expectations. BA runtime is typically only three to six times longer than circuit simulation of the same scenario.

BA can help to assess the soft error rate of future electronic systems and minimize the hardening effort by providing detailed sensitivity results.

# Zusammenfassung

Moderne integrierte Schaltungen sind empfindlich gegenüber ionisierten Partikeln, die temporäre Strompulse in den empfindlichen Bereichen eines Halbleiters verursachen können. Wenn ein solcher Partikel eine Speicherzelle trifft, kann er dort einen Single Event Upset (SEU) verursachen, der sich als invertierter Speicherwert zeigt. Insgesamt vier dominierende Fehlermaskierungseffekte, die logische und transitive Maskierung, Gatterverzögerungszeiten und Informationslebensdauer, wurden detailliert untersucht. Verschiedene Methoden zur Quantisierung der Fehlermaskierung wurden hinsichtlich ihres Funktionsprinzips, Effizienz und Genauigkeit bewertet.

Besondere SEU-bedingte Fehlereffekte in FPGAs wurden erklärt, ebenso wie Möglichkeiten zum Schutz vor diesen Fehlern. Die verwendete Speichermenge für einzelne Teilbereiche eines FPGAs wurde aus den Konfigurationsdaten hergeleitet, um eine Abschätzung der Fehlerrate eines Gesamtsystems zu ermöglichen.

Ausgehend von den Eigenschaften der zuvor untersuchen Verfahren zur Abschätzung der Fehlermaskierung wurde ein gänzlich neuartiges Verfahren vorgeschlagen. Die Rückwärts-Analyse (Backwards Analysis BA) verwendet eine einfache Schaltungssimulation in einem ersten Schritt, um danach in einem Analyse-Schritt die generierten Simulationsdaten in zeitlich umgekehrter Reihenfolge zu verwenden. Mittels diesem Ansatz wurden die vier Fehlermaskierungseffekte mit einem Schema abgedeckt, was eine sehr hohe Genauigkeit der Ergebnisse und geringe Laufzeiten ermöglicht. Das Verfahren kann auf Schaltungs-Netzlisten angewandt werden und liefert Ergebnisse, die von einem einfachen Empfindlichkeits-Mittelwert für die gesamte Schaltung bis zu Empfindlichkeitsverläufen einzelner Signale reichen. Auch die Schwachstellen und Grenzen dieses Algorithmus wurden aufgezeigt und weitere Verbesserungen hierzu vorgeschlagen.

Die Arbeitsweise und die Richtigkeit der erzeugten Ergebnisse wurden auf verschiedene Arten nachvollzogen. Kleine Schaltungen wurde durch theoretische Überlegungen verifiziert, größere Schaltungen konnten durch Fehlerinjektion mit geringem Testabstand ebenfalls überprüft werden. Es wurden nur geringe Abweichungen von korrekten Ergebnissen erkannt, die auf vereinfachte Randbedingungen bei der Analyse zurückzuführen sind. Darüber hinaus wurde ein Anwendungsfall einer großen Mikroprozessorschaltung untersucht. An einigen Stellen im Registersatz der CPU wurde unerwartet niedrige Empfindlichkeit festgestellt, die auf die Operand-Forwarding-Fähigkeiten der CPU zurückzuführen ist. Nur ca. 4% des gesamten im System enthaltenen Speichers kann zur SEU-basierten Fehlerrate beitragen. Weitere Tests, die einen weiten Bereich an Schaltungsgrößen abdeckten, bestätigten die niedrige Laufzeit der BA in einer Größenordnung von drei- bis sechsfacher Laufzeit einer einfachen Simulation des gleichen Szenarios.

Die Rückwärts-Analyse kann dabei helfen, die Soft Error Rate von zukünftigen Schaltungen abzuschätzen und den Aufwand zur Härtung dieser Schaltungen durch detaillierte Kenntnisse zu verringern.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Einleitung

Die fortschreitende Integration und Miniaturisierung hat es ermöglicht Ende der 1970'er Jahre DRAM-Bausteine mit einer Speicherkapazität bis zu 64 kBit herzustellen. Da bei dieser damals noch recht neuartigen Technologie für Speicher viele Probleme bei der Zuverlässigkeit bestanden, wurden viele Untersuchungen in diesem Themenbereich durchgeführt. Neben Fehlern, die im wesentlichen durch Schwankungen im Fertigungsprozess verursacht wurden, konnten aber auch Fehler beobachtet werden, für die kein kausaler Zusammenhang gefunden wurde. Nach weiteren Untersuchungen konnten Alpha-Partikel, d.h. die positiv geladenen Rümpfe von Helium-Atomen, als Ursache identifiziert werden. Diese Alpha-Partikel entstehen beim Zerfall radioaktiver Isotope in nahezu allen Materialien. Wenn nun ein solcher Alpha-Partikel Bereiche eines Halbleiters durchläuft, in denen bei einem DRAM die Ladung gespeichert ist, so kann dieser Partikel durch Ionisation weitere Ladungsträger erzeugen. Je nach Implementierung der Speicherzelle und dem momentan vorhandenen Zustand kann es zu einer ausreichend starken Veränderung der gespeicherten Gesamtladung kommen, dass ein nachfolgender Leseverstärker beim DRAM einen anderen, als den ursprünglich gespeicherten, digitalen Zustand erkennt. Da diese Fehler, außer bei der gespeicherten Information, keine dauerhaften Defekte verursachen, wurde die Bezeichnung "Soft Error" eingeführt [1].

Weitere Untersuchungen zeigten, dass die Reichweite von Alpha-Partikeln in anderen Medien als Vakuum sehr begrenzt ist. Daher konnte als Quelle für die Alpha-Partikel schnell die Vergussmasse der Chip-Gehäuse und passivierende Glas-Schichten auf den Halbleitern selbst identifiziert werden. In den folgenden Jahren wurden daher die Bemühungen verstärkt, an diesen Stellen nur noch Materialien zu verwenden, die entweder generell eine möglichst geringe spezifische Aktivität haben oder besonders gereinigt wurden.

Parallel zu den Bemühungen, die durch Alpha-Partikel erzeugte Fehlerrate zu reduzieren, wurden als weitere Fehlerquelle mit ähnlichem Wirkmechanismus hochenergetische Neutronen und Protonen identifiziert [2]. Anders als Protonen oder Alpha-Partikel zeigen Neutronen auf Grund der fehlenden Ladung nur wenig Interaktion mit anderer Materie, d.h. ihre Reichweite in Materie ist wesentlich größer. Untersuchungen zu durch Neutronen verursachten Soft Errors

zeigten kein einheitliches Verbesserungspotential, selbst bei meterdicken Beton-Abschirmungen [3]. Daher mussten ab diesem Zeitpunkt, neben lokal erzeugter Strahlung, auch Partikel auf Grund von kosmischer Strahlung in den Überlegungen berücksichtigt werden. Besonders hohe Fehlerraten wurden daher speziell in der Luft- und Raumfahrt festgestellt, da hier die abschirmende Wirkung der Erdatmosphäre verringert oder nicht mehr vorhanden ist [4]. Die maximale Neutronen-Flussdichte auf Grund kosmischer Strahlung wurde in einer Höhe von ca. 20 km festgestellt, auf Meereshöhe ist die Flussdichte schon um den Faktor 100 geringer. Speziell für erdnahe Satelliten stellt die Protonen-Strahlung die größte Bedrohung dar.

Mit weiter fortschreitender Verringerung der Strukturgröße integrierter Schaltungen wurde dieser Fehlermechanismus zu einem allgemeinem Problem. Alle Fehlereffekte, die durch hochenergetische Partikel (teilweise mit einer Energie größer als 100 MeV) verursacht werden, sind durch den Begriff Single Event Effect SEE zusammengefasst. Hochenergetische Neutronen, Protonen oder Alpha-Partikel verursachen bei einer Kollision mit einem dotierten Halbleitermaterial häufig in einer Kettenreaktion weitere Partikel mit niedriger Energie. Am Ende dieser Reaktionskette stehen verschiedene ionisierte Partikel (Alpha-Teilchen, Protonen, Elektronen). Beim Durchlaufen des dotierten Substrats werden von den ionisierten Teilchen sekundäre Ladungsträger freigesetzt, die sich als zeitlich begrenzter Störstrom-Impuls an dem betroffenen Bauteil zeigen. Bei einem Speicherkondensator in einem DRAM führt dies zu einer veränderten gespeicherten elektrischen Ladung, welche stellvertretend für die digitale Information ist. Im Falle eines MOS-Transistors in einer digitalen Schaltung kann sich der Störstrom-Impuls dann sogar zu weiteren Transistoren und Gattern fortpflanzen und zu beobachtbaren Auswirkungen führen [5]. In einer höheren Abstraktionsebene äußert sich ein solcher Partikeltreffer dann als Glitch eines Logiksignals oder als invertierter gespeicherter Wert in einer Speicherzelle (SRAM oder DRAM). Nicht jeder Fehler muss aber für das Systemverhalten kritisch sein. Manche Fehler führen nur zu vernachlässigbaren Auswirkungen, andere hingegen bleiben auch komplett wirkungslos. Die Logikfunktion der Schaltung kann unter anderem hierfür verantwortlich sein.

Die Soft Errors unter den Single Event Effects SEE [6] werden in zwei Bereiche aufgeteilt:

- Single Event Transient SET: Ein zeitlich begrenzter Störimpuls an einem Schaltungssignal
- Single Event Upset SEU: Der Störimpuls verursacht einen Zustandswechsel in einer Speicherzelle, der invertierte Signalwert wird fortan gespeichert

Da es bei einem SEE auch zu einer dauerhaften, zerstörerischen Veränderung der Schaltung kommen kann, werden weiterhin für SEEs mit permanenter Auswirkung folgende Klassen unterschieden:

- Single Hard Error SHE: Ein SEU, der zu einem permanenten Fehler wie z.B. einem Stuck-at-Fehler führt

- Single Event Latchup SEL: Ein SEE, der zum Verlust der Funktion des Bausteins führt und nur durch Abschalten der Versorgungsspannung beseitigt werden kann
- Single Event Burnout SEB: Ein SEE verursacht durch hohen Stromfluss einen dauerhaften Schaden in einem Leistungstransistor
- Single Event Gate Rupture SEGR: Ein SEE, der einen leitfähigen Pfad durch das Gate-Oxid eines Leistungstransistors erzeugt

## 1.1 Messbare Größen bei SEEs

Aus Sicht der physikalischen Vorgänge kann einfach eine Bestimmungsgleichung für die Anzahl der wirksamen ionisierenden Partikel aufgestellt werden [2]:

$$Y = \phi \cdot N \cdot \sigma \cdot A \cdot l \tag{1.1}$$

$\phi$: Partikelflussdichte pro $cm^2$, N: Anzahl Atome pro $cm^3$
$\sigma$: Wirkungsquerschnitt in $\frac{cm^2}{bit}$, A: Fläche der betrachteten Speicherzelle in $cm^2$
l: Länge bzw. Dicke des wirksamen Bereichs in $cm$

In Abhängigkeit von der Anzahl der ionisierenden Partikel und der von ihnen abgegebenen Energie werden unterschiedlich viele Ladungsträger erzeugt. Abhängig vom Aufbau des betroffenen Bauteils kann eine sogenannte kritische Ladung $Q_{Crit}$ definiert werden, welche die minimale Ladungsmenge zum Auslösen eines SEUs beschreibt. Obwohl die kritische Ladung einher mit schrumpfender Strukturgröße sinkt, bleibt die resultierende Fehlerrate nahezu konstant [7]. Da auch das empfindliche Volumen sinkt, welches in etwa durch die Fläche und Tiefe der dotierten Bereiche des Bauteils definiert wird, kann ein Partikel dort nur eine begrenzte Menge an Ladungsträgern erzeugen. Abhängig von der kritischen Ladung können unterschiedliche Partikeltypen zu stark unterschiedlichen resultierenden Fehlerraten führen. Auf Grund der geringeren Größe führen Protonen bei sehr kleiner kritischer Ladung zu einer wesentlich höheren Fehlerrate als die gleiche Menge an Alpha-Partikeln.

In der Arbeit von Ziegler et al [8] wurde eine Vielzahl an Experimenten mit verschiedenen DRAM-Chips durchgeführt. Abhängig von der Struktur der verwendeten Speicherzellen wurden stark unterschiedliche Fehlerraten festgestellt. Die höchsten Fehlerraten wurden bei Trench-Speicherzellen mit externem Ladungsspeicher (TEC) festgestellt. Hier lag die Fehlerrate zwischen 1300 und 1500 Fehlern pro $10^{15}$ Bit und Betriebsstunde (entspricht 1300 bis 1500 FIT/MBit, siehe Abschnitt 2.4.1). Geringere Fehlerraten wurden für DRAMs mit Stacked Capacitor Technologie (SC) festgestellt, hier lag die Fehlerrate im Bereich 110 bis 490 FIT/MBit. Beste Ergebnisse wurden für DRAMs mit Trench-Speicherzellen und internem Ladungsspeicher (TIC) gemessen, hier lagen die Fehlerraten unter 1 FIT/MBit.

Für SRAM-Zellen zeichnet sich ein ähnliches Bild ab. Auf verschiedene Art und Weise, z.B. durch Experimente mit Teilchenbeschleunigern oder durch Langzeittests mit möglichst vielen Bausteinen, wurden repräsentative FIT-Raten bestimmt. Diese FIT-Raten liegen im Bereich von 10 bis 1000 FIT/MBit [9]. Obwohl die beobachteten FIT-Raten mit schrumpfender Strukturgröße sogar leicht sinken, gilt dies nicht für die Gesamt-Fehlerwahrscheinlichkeit. Da die Menge an Speicher in einer Schaltung deutlich schneller wächst [10] als die FIT-Rate sinkt, nimmt die Bedeutung von Soft Errors insgesamt zu.

## 1.2 Motivation

Um dem wachsenden Druck nach zuverlässigen Bauteilen gerecht zu werden, soll in dieser Arbeit ein genauerer Blick auf die Bestimmung der Fehlerraten in integrierten Schaltungen geworfen werden. Diese Fehlerrate setzt sich im wesentlichen aus einem technologieabhängigen Teil und einem schaltungsabhängigen Teil zusammen. Für den technologieabhängigen Teil wurden bereits viele Untersuchungen durchgeführt, einige davon wurden auch im vorherigen Abschnitt schon erwähnt. Da sich die Fertigungstechnik, die Strukturgröße und die verwendeten Materialien schnell ändern, können hier keine längerfristig gültigen Erkenntnisse gewonnen werden. Der Schwerpunkt dieser Arbeit soll daher auf die wenig untersuchten Auswirkungen von Schaltungsstruktur und Anwendungsfall auf die Fehlerrate gelegt werden. Insbesondere SRAM-lastige Bausteine, wie z.B. FPGAs, sollen genauer untersucht werden. Mit den ermittelten Fehlerraten ist es dann möglich, einen quantitativen Vergleich zwischen verschiedenen Realisierungen der gleichen Funktion zu ziehen, die Eignung einer Schaltung für eine Anwendung festzustellen oder auch das notwendige Maß an Verbesserung festzulegen.

## 1.3 Überblick

In den folgenden Teilen der Arbeit werden die Voraussetzungen, der Stand der Technik und ein neuartiger Ansatz für die Bestimmung der Fehlermaskierung aufgezeigt. In Kapitel 2 werden die bekannten Fehlermaskierungseffekte genau untersucht und bekannte Verfahren zur Abschätzung der Fehlermaskierung erklärt und bewertet. Im Weiteren wird auch Bezug zu den anwendbaren Normen und Standards für zuverlässige elektronische Systeme genommen. Kapitel 3 erklärt, wie der Aufbau von FPGAs sich von ASICs unterscheidet und welche speziellen Fehlereffekte hier auftreten können. Eine Analyse der verfügbaren Fehlererkennungsmaßnahmen und eine Abschätzung der Speichermengen in FPGAs bildet hier den Abschluss. In Kapitel 4 wird das Konzept für einen neuartigen Ansatz zur Bestimmung der Fehlermaskierung erklärt. Insbesondere wird auf die Eigenschaften dieses Verfahrens bei der Integration verschiedener Maskierungseffekte

und den notwendigen Rechenaufwand eingegangen. Kapitel 5 stellt einige Verbesserungen gegenüber dem initialen Ansatz dar, mit dem das Verfahren für eine noch größere Menge an Schaltungen anwendbar wird. Der vorletzte Abschnitt, Kapitel 6 beinhaltet das Vorgehen zur Verifizierung des vorgeschlagenen Verfahrens. Durch mehrere unterschiedliche Vergleichsverfahren wird das Verfahren selbst und die gezeigten Verbesserungen überprüft. Die gewonnenen Ergebnisse für eine Mikroprozessorschaltung werden ausführlich erklärt und durch eine theoretische Herleitung ebenfalls verifiziert. Zum Abschluss dieser Arbeit wird in Kapitel 7 eine Zusammenfassung und ein Ausblick auf weitere Anwendungsgebiete des Verfahrens gegeben.

# Kapitel 2

# Stand der Technik

In diesem Kapitel werden die grundlegenden Effekte und Techniken für das Verhalten von digitalen Schaltungen im Zusammenhang mit transienten Fehlern erläutert. Weiterhin werden Methoden zur Quantifizierung der Fehlereffekte erklärt und die in der Industrie anwendbaren Normen aufgezeigt.

## 2.1 Fehlerschutztechniken

Verschiedene Ansätze können verfolgt werden, um Schaltungen mit Eigenschaften auszustatten die eine Fehlererkennung bzw. eine Fehlerkorrektur erlauben. Gemeinsamkeit bei allen diesen Ansätzen ist die zusätzliche erforderliche Redundanz, die Kosten (z.B. in Form von vergrößerter Chipfläche, erhöhter Verlustleistung oder verringerter Arbeitsgeschwindigkeit) können jedoch sehr unterschiedlich sein.

### 2.1.1 Funktionale Redundanz

Funktionale Redundanz wird durch mehrfache Verfügbarkeit eines Bauteils, einer Baugruppe oder einer Komponente erreicht. Fällt eines der Bauteile aus oder es wird ein Fehler erkannt, so können die verbleibenden Bauteile dessen Funktion übernehmen bzw. zur Fehlerkorrektur herangezogen werden [11].

Im Allgemeinen wird bei der funktionalen Redundanz zwischen folgenden Ansätzen unterschieden:

- Heiße Redundanz: Alle verfügbaren Bauteile befinden sich im Betrieb, deren funktionaler Ausgang wird mittels einem Mehrheitsentscheider verglichen. Der Ausgang dieses Mehrheitsentscheid wird dem restlichen System zur Verfügung gestellt. Das Prinzip Mehrheitsentscheid erfordert mindestens drei parallel arbeitende Bauteile.
- Kalte Redundanz: Bei der kalten Redundanz wird der Ausgang eines im Betrieb befindlichen Bauteils mittels einer separaten Einheit überwacht, die im Fehlerfall auf ein weiteres vorhandenes Bauteil umschaltet. Dabei ist zu beachten, dass die Fehlerwahrscheinlichkeit der Überwachungseinheit

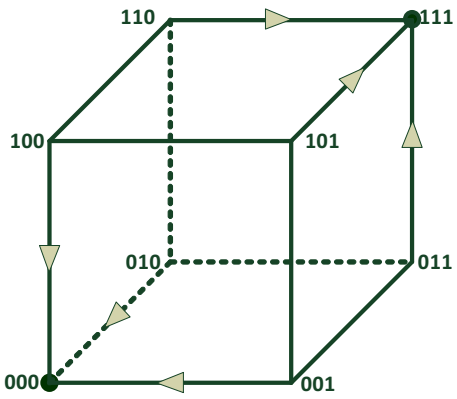



Abbildung 2.1: Hypercube

und des Umschalters wesentlich geringer sein muss als die der funktionalen Einheit.

### 2.1.2 Informations-Redundanz

Der Begriff Redundanz wird in der Informationstechnik für Information verwendet, die eigentlich nicht gespeichert oder verarbeitet werden müsste, da sie mehrfach vorhanden ist oder durch Abhängigkeiten aus anderen Informationen bestimmt werden kann. Wie viel Redundanz in einer Nachricht bzw. Information enthalten ist richtet sich nach deren Entropie [12], dem Informationsgehalt einer Nachricht. Die Entropie wird nach folgender Methode berechnet:

$$H = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i \tag{2.1}$$

Redundanz kann in der Informationstechnik nutzbringend zur Fehlererkennung und Fehlerkorrektur eingesetzt werden. Ein einfacher Indikator für die Leistungsfähigkeit einer Codierung ist die Hamming-Distanz. Bei einem beliebigen gültigen Codewort der Länge $n$ gibt die Hamming-Distanz an, wie viele Bits des Codeworts verändert werden müssen, bis ein weiteres gültiges Codewort erreicht wird. Dadurch lassen sich die fehlererkennenden und fehlerkorrigierenden Eigenschaften einfach ableiten. Eine Hamming-Distanz von 2 erlaubt die Erkennung eines 1-bit Fehlers, Hamming-Distanz 3 erlaubt die Fehlerkorrektur eines 1-bit Fehlers und die Erkennung von 2-bit Fehlern. Je nach tatsächlicher Implementierung eines Codes kann es auch zur Falsch-Korrektur von Mehrfach-Fehlern kommen. Zur Visualisierung der Hamming-Distanz können Hypercubes (Abbildung 2.1) oder Karnaugh-Veitch-Diagramme verwendet werden.

### 2.1.3 Zeitliche Redundanz

Zeitliche Redundanz ist im Allgemeinen mit einer verlängerten Ausführungsdauer einer Operation oder Funktion verbunden. Dabei wird zusätzlich zu einer initialen Ausführung der Funktion zu einem späteren Zeitpunkt ein oder mehrere weitere Male die gleiche Funktion ausgeführt und die Ergebnisse verglichen, die in unterschiedlichen Zeiträumen entstanden sind. Durch den zeitlichen Versatz soll damit der Einfluss von zeitlich bedingten Störeinflüssen vermindert oder ganz verhindert werden. Die eigentliche Auswahl oder Gewichtung innerhalb der Ergebnisse kann wieder auf unterschiedliche Art und Weise erfolgen.

## 2.2 Fehlermaskierungseffekte

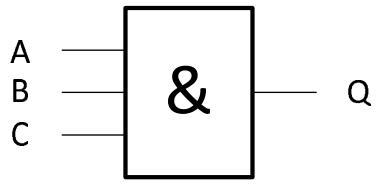
Transiente Fehler können zu jeder Zeit und an jedem Ort in einer digitalen Schaltung auftreten und eine Abweichung im Zustand gegenüber einer fehlerfreien Schaltung verursachen. Die Anzahl der aufgetretenen Fehler ist jedoch nicht gleich der Anzahl der beobachtbaren Fehler, was auf eine anteilige Fehlermaskierung schließen lässt. In der Literatur sind mehrere Fehlermaskierungseffekte bekannt.

### 2.2.1 Logische Maskierung

Digitale kombinatorische Gatter bestimmen den Wert ihres Ausgangssignals direkt aus allen Eingangssignalen des Gatters. Bei einer Änderung des Wertes eines oder mehrerer Eingangssignale kann der neue Wert des Ausgangssignals aus der Wertetabelle des Gatters abgelesen werden und nachfolgend die sogenannte boolesche Empfindlichkeit bestimmt werden. Die Empfindlichkeit gibt daher an, ob bei einer Änderung eines oder mehrerer Eingangssignale auch eine Veränderung des Ausgangssignals erfolgt. Obwohl die Empfindlichkeit für eine beliebige Anzahl sich ändernder Signale bestimmt werden kann, wird im weiteren Verlauf immer die Einzelfehler-Empfindlichkeit verwendet, sofern nicht etwas anderes angegeben wird.

Am Beispiel eines UND-Gatters (Abbildung 2.2) und der dazu gehörigen Wertetabelle kann die Empfindlichkeit leicht abgelesen werden. Für jeden Eingangsvektor werden alle dazu gehörigen Eingangsvektoren mit einer Hamming-Distanz von eins betrachtet. Der hier betrachtete Eingangsvektor (C=1, B=1, A=0) und die drei zugehörigen Eingangsvektoren mit Hamming-Distanz eins zeigen, dass nur der Eingang A als empfindlich erachtet wird. Nur an diesem Eingang würde ein Fehler zu einer Änderung am Gatterausgang führen. Zur Darstellung der Empfindlichkeit hat sich eine Notation ähnlich einer partiellen Ableitung einer Funktion wie in der Algebra eingebürgert, d.h. $\frac{\partial Q}{\partial I}$ stellt die Empfindlichkeit des Ausgangs $Q$ in Abhängigkeit vom Eingang $I$ dar.

Neben der Möglichkeit die Empfindlichkeit durch Ablesen aus einer Wertetabelle zu bestimmen, besteht die Möglichkeit diese direkt durch boolesche Be-




| C | B | A | Q | $\frac{\partial Q}{\partial C}$ | $\frac{\partial Q}{\partial B}$ | $\frac{\partial Q}{\partial A}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Abbildung 2.2: Boolesche Empfindlichkeit für Einzelfehler

rechnungen zu erhalten. Für jede boolesche Funktion $f(\mathbf{x})$ lässt sich analog zum Vorgehen mit einer Wertetabelle die Empfindlichkeit durch die Berechnung von der Booleschen Differenz bestimmen.

$$\frac{\partial Q}{\partial x_i} = f(x_1, ..., x_i, ..., x_n) \oplus f(x_1, ..., \bar{x_i}, ..., x_n) \tag{2.2}$$

Diese Form (Gleichung 2.2) erlaubt neben der Auswertung anhand eines gegebenen Eingangsvektors auch eine symbolische Auswertung. Durch Anwenden von Regeln zur Umformung boolescher Funktionen kann jeweils eine geschlossene Form für die Empfindlichkeit einer gegebenen logischen Funktion für jedes der Eingangssignale bestimmt werden.

Besonderheiten bei der Bestimmung der Empfindlichkeit ergeben sich, wenn die logische Maskierung gatterweise bestimmt wird und wenn es zu rekonvergenten Pfaden in der Schaltung kommt. Bei einer einfachen gatterweisen Bestimmung der Empfindlichkeit wird möglicherweise außer Acht gelassen, dass sich ein Fehler auf mehreren Pfaden zu einem Gatter hin fortpflanzen kann. In diesem Fall manifestiert sich ein Einzelfehler als Mehrfachfehler an einem Gatter, was wiederum zur Verletzung der Annahme von Einzelfehlern bei der Bestimmung der Empfindlichkeit führt. Kann innerhalb einer sequentiellen Stufe einer Schaltung eine einzelne logische Funktion für jeden der Ausgänge angegeben werden, so besteht keine Gefahr einer Falsch-Bewertung der Mehrfachmanifestierung eines Einzelfehlers. In größeren Schaltungen können rekonvergente Pfade aber auch über die Grenzen sequentieller Stufen verlaufen. Dadurch sind neben den logischen Signalen auch noch zeitliche Abhängigkeiten zu berücksichtigen was die Bestimmung der logischen Maskierung mittels Boolescher Differenz verhindert.

Betrachtungen zur Empfindlichkeit von logischen Funktionen finden häufig Verwendung im Bereich des Schaltungstests bzw. der Testmustergenerierung. Empfindlichkeit kann hier analog zur Beobachtbarkeit verwendet werden. Beobacht-

barkeit ist beim Schaltungstest eines der beiden Kriterien, die notwendig sind, um einen permanenten Fehler erkennen zu können. Das zweite Kriterium, die Steuerbarkeit, ist hier in jedem Fall erfüllt, da im Fehlerfall der zu betrachtende Schaltungszustand schon vorliegt.

Bei der logischen Maskierung bestehen zwei grundsätzliche Ansätze zur Quantifizierung. Im ersten Ansatz, der booleschen Differenz, wird die Maskierung für einen konkret vorliegenden Eingangsvektor einer logischen Funktion angegeben. Als zweiter Ansatz wird eine statistische Betrachtung herangezogen. Unter Verwendung der Statistik der Eingangssignalvektoren $p(\overline{x})$ und der logischen Maskierung für jeden möglichen Eingangsvektor kann eine gewichtete Summe, d.h. der Mittelwert der logischen Maskierung berechnet werden.

$$d_{Logic,i} = \sum_{\overline{x}} \frac{\partial Q(\overline{x})}{\partial x_i} \cdot p(\overline{x}) \tag{2.3}$$

### 2.2.2 Transitive Maskierung

Transitive Maskierung entsteht in Folge einer Verkettung von logischer Maskierung in einer sequentiellen Schaltung. Die logische Maskierung wird in der Regel nur für die kombinatorischen Gatter zwischen zwei sequentiellen Elementen bestimmt. Die logische Maskierung beeinflusst aber auch die Verwendung von Ergebnissen aus vorangegangenen Schaltungsteilen. Damit sind neben den Eingangssignalen des Gatters noch zusätzliche Signale zu berücksichtigen. Wenn also in einem Datenpfad mit mehreren sequentiellen Stufen in den vorderen Stufen empfindliche Pfade erkannt werden, jedoch Fehler die in diesen Stufen entstehen in späteren Stufen logisch maskiert werden, dann spricht man von transitiver Maskierung. Weitergehende Betrachtungen sind aber nur schwer möglich, da das Zeitverhalten der sequentiellen Elemente bedeutenden Einfluss auf die Funktion der Schaltung und damit auch auf die Maskierung hat.

### 2.2.3 Zeitliche Maskierung

Neben der bisher betrachteten digitalen Funktion von Schaltungen spielen in der Realität aber auch die analogen Eigenschaften von Gattern eine Rolle bei der Fehlermaskierung [13]. Insbesondere dem zeitlichen Verhalten von Gattern kommt eine große Bedeutung bei der Fehlermaskierung zu.

Als Ausgangspunkt für Betrachtungen hinsichtlich der Fehlerausbreitung kann das Verhalten eines einfachen Registers herangezogen werden. Hier existieren zwei zeitliche Bereiche abhängig vom Auftreten einer aktiven Flanke des Taktsignals in denen der Eingangswert stabil bleiben müsste, damit eine fehlerfreie Funktion des Registers gewährleistet werden kann. Jegliche Signalzustands-Wechsel während der Setup- und Hold-Zeit des Registers führen zu einem undefinierten

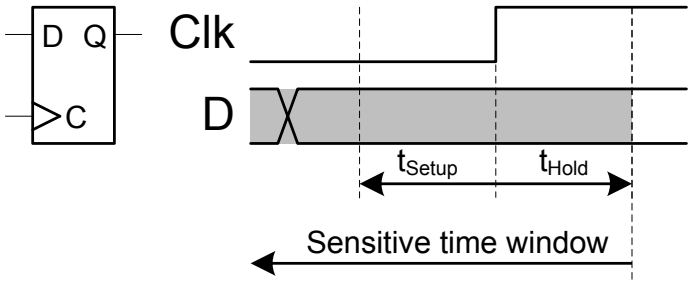



Abbildung 2.3: Kritisches Zeitfenster an einem Register

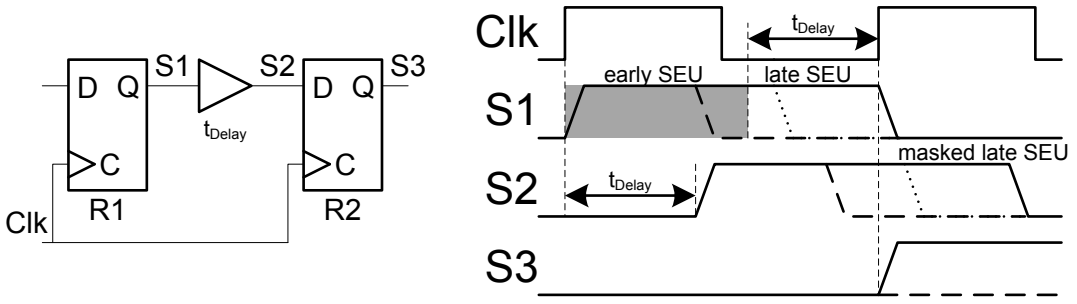



Abbildung 2.4: Zeitliche Maskierung

Ausgangssignal des Registers, weswegen dieser Zeitraum als empfindlich anzusehen ist [14]. Sämtliche fehlerhaften Eingangssignale, die nach der Hold-Zeit am Eingang des Registers eintreffen, haben keinen Einfluss mehr auf das Verhalten des Registers in Bezug auf die vorangegangene Taktflanke. Der Eingang des Registers wird deshalb unmittelbar nach Ablauf der Hold-Zeit als nicht empfindlich betrachtet. Im Gegensatz dazu kann ein fehlerhaftes Eingangssignal beliebig früh vor der Taktflanke eintreffen. In diesem Fall besteht kein Unterschied zur regulären Funktion des Registers, es wird eben nur ein fehlerhafter Wert abgetastet und gespeichert. Das empfindliche Zeitfenster beginnt daher im schlechtesten Fall unmittelbar nach der vorhergehenden Taktflanke. In Abbildung 2.3 wird die Herleitung der empfindlichen Zeitbereiche für den Eingang eines Registers grafisch erläutert.

Diese Überlegungen können nun für eine komplette Stufe einer sequentiellen Schaltung weitergeführt werden, wobei in zwei Gruppen, die Quellen- und Senken-Register, unterschieden werden muss. Die Quellen-Register werden hier als Ort für ein mögliches Auftreten eines SEUs gesehen. Daher kann in dieser Betrachtung ein SEU immer erst nach einer Taktflanke am Quellen-Register auftreten, da hier dann der neu gespeicherte Wert verfälscht wird. Die Zeitspanne, in der ein fehlerhaftes Quellen-Ausgangssignal den Eingang der Senke erreicht, ist durch die Laufzeit der dazwischen liegenden Gatter definiert. Folglich definieren im Umkehrschluss daher der Zeitpunkt der Taktflanke am Senken-Register und die Gatterlaufzeit auf jedem möglichen Pfad einen spätesten Zeitpunkt, bis zu dem

ein Quell-Register kritische Information speichert. Ab diesem Zeitpunkt würde ein fehlerhaftes Register-Ausgangssignal erst nach dem kritischen Zeitfenster zum Eingang eines Senken-Registers propagiert sein. Um das empfindliche Zeitfenster des Quell-Registers zu erhalten, muss daher nur das empfindliche Zeitfenster des Senken-Registers um die Pfad-Verzögerungszeit in die Vergangenheit verschoben werden (Abbildung 2.4). Dadurch wird ein Block empfindlicher Zeit am Quell-Register definiert. Der Beginn dieses Blocks ist abhängig von der Taktflanke am Quell-Register, das Ende von der Taktflanke am Senken-Register.

Ein Ansatz zur Quantifizierung der zeitlichen Maskierung basiert auf der statischen Timing-Analyse. Dabei wird für jedes Register der nachfolgende Pfad mit kürzester Laufzeit gesucht, diese Laufzeit im Verhältnis zur Taktperiode kann als Laufzeit-Maskierungsfaktor gesehen werden. In dieser Betrachtung wird aber nicht berücksichtigt, dass nicht jeder dieser kürzesten Pfade immer aktiv ist und nicht alle Register werden jeden Takt getriggert, was zu einer Informationslebensdauer von mehr als einem Takt führen kann.

$$d_{Timing} = \frac{T_{Shortest\ path}}{T_{Clk}} \tag{2.4}$$

### 2.2.4 Informationslebensdauer

Ein weiterer Maskierungseffekt wird durch die Lebensdauer von gespeicherter Information begründet [15]. Prinzipiell ist die Lebensdauer einer gespeicherten Information durch zwei aufeinander folgende Schreib-Vorgänge begrenzt. Die Lebensdauer einer Information beginnt mit dem Beschreiben des Speichers mit der gewünschten und hier betrachteten Information und endet mit einem nachfolgenden Überschreiben mit neuen Daten. Da diese Schreibvorgänge nicht notwendigerweise an ein Taktsignal gebunden sind, kann es zu Lebensdauern von wesentlich mehr als nur einem Taktzyklus kommen. Als Beispiel seien hier die Konfigurationsregister eines Mikrocontrollers genannt, die in der Regel einmal beim Systemstart beschrieben werden und dann während der ganzen System-Laufzeit unverändert bleiben.

Eine feinere Begrenzung der Lebensdauer ist möglich, wenn auch die Zeitpunkte des Lesezugriffs auf die Information bekannt sind. Nur weil eine Information für einen längeren Zeitraum unverändert in einem Speicher vorliegt, muss das nicht bedeuten, dass diese Information auch die gesamte Zeit in der Schaltung tatsächlich verwendet wird. Tatsächlich kommt es häufiger vor, dass zwischen Schreiben und Lesen eines Speichers nur eine kurze Zeit liegt und danach der Speicher für einen längeren Zeitraum ungenutzt bleibt (sogenannte "Dead values" [16]). Die Erkennung eines Lesezugriffs auf einen Speicher jeglicher Art kann aber nicht von diesem Speicher selbst, sondern nur von den nachgeschalteten, weiterverarbeitenden Einheiten ausgelöst werden.

### 2.2.5 Betriebsdauer und Energiezustand

Der einfachste Fehlermaskierungseffekt wird durch den Betrieb der betrachteten Schaltung erzielt. Falls die Schaltung sich nicht im Betrieb befindet, verursachen aufgetretene Fehler selbstverständlich kein Fehlverhalten der Schaltung. Ähnliche Effekte können in verschiedenen Energiespar-Zuständen von Schaltungen auftreten, jedoch hängt die Empfindlichkeit der Schaltung von vielen Details der Implementierung ab, so dass keine pauschal gültige Aussage getroffen werden kann. Unter Umständen verursacht ein Energiesparzustand mit abgesenkter Betriebsspannung der Schaltung sogar eine erhöhte Grundfehlerrate im Gegensatz zur Erwartung einer geringeren Fehlerrate bei verkürzter Nutzungsdauer der Schaltung.

## 2.3 Methoden zur Quantifizierung von Maskierungseffekten

Der Quantifizierung von einzelnen Maskierungseffekten bzw. des gesamten Maskierungseffekts kommt eine sehr hohe Bedeutung zu. Auf Basis der Maskierungsfaktoren werden Abschätzungen des Risikopotentials und der Verbesserungsmöglichkeiten von Schaltungen getroffen. Neben den bereits vorher erwähnten Möglichkeiten der Verwendung von Signalstatistiken (siehe 2.2.1) oder Daten der statischen Timing-Analyse (siehe 2.2.3) sind in der Literatur weitere Ansätze bekannt.

### 2.3.1 Fehlerinjektion

Ein weit verbreiteter und naheliegender Ansatz zur Bestimmung der Fehlermaskierung von digitalen Schaltungen ist die sogenannte Fehlerinjektion [17]. Dabei wird das Verhalten einer Schaltung beobachtet, in die ein Fehler eingebracht wird. Die Fehlererkennung wird durch einen Vergleich der Ausgangssignale der fehlerhaften und einer fehlerfreien Schaltung realisiert. Werden dabei viele solcher Tests mit unterschiedlichen Fehlern durchgeführt, so kann eine statistisch belastbare Aussage über die Fehlermaskierung der Schaltung getroffen werden. Die Fehlerinjektion kann prinzipiell in Form einer Schaltungssimulation oder auch mit echter Hardware, häufig aber unter Verwendung von Prototyping-Platformen mit FPGAs [18], durchgeführt werden.

In Abbildung 2.5 ist der grundsätzliche Aufbau eines Testsetups für die Durchführung der Fehlerinjektion abgebildet. Neben der Schaltung, in die Fehler injiziert werden (Unit Under Test UUT), muss noch eine weitere Instanz dieser Schaltung vorhanden sein, dessen Ausgangssignale zur Fehlererkennung herangezogen werden (UUT Reference). Da die Fehlererkennung auf einem unmittelbaren Vergleich der Ausgangssignale beider Schaltungsinstanzen besteht, ist es nicht notwendig die Testbench-Umgebung ebenfalls zweifach auszuführen. Während bei

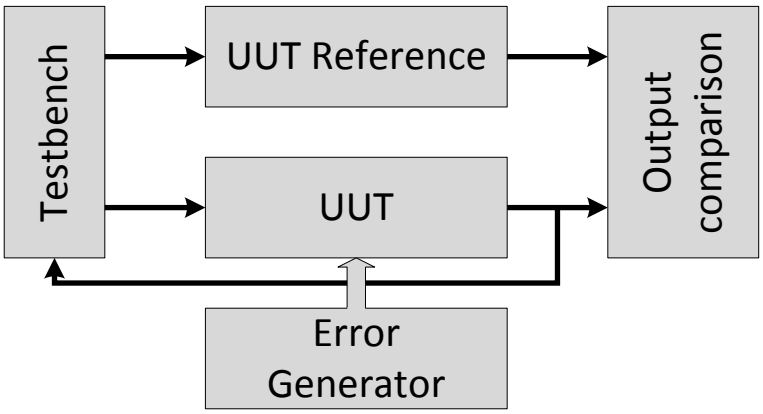



Abbildung 2.5: Testsetup für Fehlerinjektion

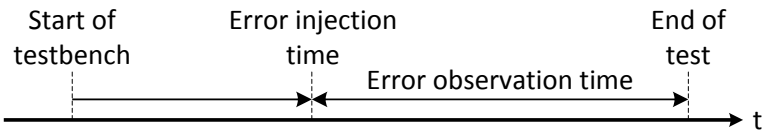



Abbildung 2.6: Zeitlicher Ablauf bei der Fehlerinjektion

einer Schaltungsinstanz eine Rückkopplung evtl. notwendig sein kann, darf die zweite Instanz auch rein gesteuert werden, d.h. deren Ausgangssignale haben keinen Einfluss auf das Verhalten der Testbench. Wann immer eine Fehlerinjektion eine Abweichung der Ausgangssignale von ihrem Soll-Wert verursacht, so wird dies durch den Ausgangs-Vergleich erkannt. Das weitere Verhalten der Testbench ist ab diesem Zeitpunkt irrelevant. Aus Gründen der Aufwandsoptimierung für die Durchführung der Fehlerinjektion kann daher auf eine zweite Testbench-Instanz verzichtet werden.

Der zeitliche Ablauf der Fehlerinjektion ist in Abbildung 2.6 dargestellt. Nach dem Start der Testbench muss die Schaltung so lange betrieben werden bis der gewünschte Testzeitpunkt erreicht ist. Dort wird dann der Fehler in die Schaltung eingespeist und der reguläre Betrieb fortgesetzt. Für eine vorab festgelegte Zeitspanne, der Beobachtungszeit, werden die Ausgänge der UUT gegenüber den Ausgängen der Referenzschaltung verglichen um Fehler erkennen zu können. Die Länge der Beobachtungszeit ist von mehreren Faktoren abhängig:

- Taktrate und sequentielle Tiefe der Schaltung
- Rückkopplungseigenschaften der Schaltung
- Verzögerungszeit an Ein- und Ausgängen der Schaltung
- Anwendungsfall der Schaltung

Die Länge der Beobachtungszeit hat Einfluss auf die Qualität der Ergebnisse der Fehlerinjektion. Eine zu kurze Beobachtungszeit relativ zur tatsächlichen Fehlerlatenz führt zu nicht erkannten Fehlern und damit zu einer Verfälschung der Ergebnisse. Dabei handelt es sich um einen nicht behebbaren Nachteil der Feh-

lerinjektion. Durch den Ausgangsvergleich kann nur sicher festgestellt werden, dass ein Fehler aufgetreten ist, nicht jedoch, das bei längerer Beobachtung kein Fehler mehr auftreten wird. Bei einfachen Schaltungen ohne interner Rückkopplung kann die maximale Fehlerlatenz genau bestimmt werden, dementsprechend ist auch die maximale Beobachtungszeit genau definiert. Da jedoch die Mehrzahl der Schaltungen mit Rückkopplung interner Zustände arbeitet, kann in diesem Fall keine Obergrenze für die Fehlerlatenz festgelegt werden. Insofern muss hier für eine hohe Genauigkeit der Ergebnisse einer Fehlerinjektion die Beobachtungszeit immer ein Vielfaches der durchschnittlichen Fehlerlatenz sein.

Die Durchführungszeit für eine ganze Kampagne der Fehlerinjektion hängt von mehreren Eigenschaften ab:

- Ausführungsgeschwindigkeit des Systems
- Zeit bis zur Fehlerinjektion
- Beobachtungszeit
- Anzahl der Testpunkte

Grundsätzlichen Einfluss auf die Ausführungsdauer einer Fehlerinjektion hat die Anzahl der Testpunkte, d.h. wie viele Versuche insgesamt durchgeführt werden müssen. Als zweite Größe beeinflusst die Ausführungsgeschwindigkeit des Systems die Dauer der Fehlerinjektion. Bei einer Fehlerinjektion in Simulation unterscheiden sich Simulationsdauer und simulierte Zeit meistens um mehrere Größenordnungen, Fehlerinjektion in Hardware läuft dagegen häufig in Echtzeit oder um einen konstanten Faktor verlangsamt ab. Die Ausführungsdauer einer einzelnen Fehlerinjektion hängt aber von der Schaltung selbst und ihrem Anwendungsfall ab. Zum Einen müssen die einzelnen Fehler in einer Versuchsreihe zu verschiedenen Zeitpunkten eingespeist werden. Nach einem Test muss jedoch, unabhängig vom Ausgang des Tests, der Anfangszustand wieder hergestellt werden, da nicht ausgeschlossen werden kann, dass noch fehlerhafte Zustände im System vorhanden sind. Zum Anderen muss die Schaltung nach der Fehlerinjektion mindestens für die Beobachtungszeit weiter betrieben werden, um ein aussagekräftiges Ergebnis zu erhalten. Bei einer Hardware-Fehlerinjektion muss also nach jedem Test ein vollständiger Reset der Schaltung stattfinden um einen garantiert fehlerfreien Anfangszustand zu erreichen. Bei der Fehlerinjektion durch Schaltungssimulation kann unter Umständen der Zustand der Schaltung vor der Fehlerinjektion gespeichert werden und nach Ende der Beobachtungszeit wiederhergestellt werden. Wenn der Aufwand zur Wiederherstellung bzw. die Vorlaufzeit der Hardware gering ist, kann die gesamte zu betrachtende Systemzeit näherungsweise durch die Gleichung 2.5 abgeschätzt werden.

$$T_{System} = T_{Observation} \cdot N_{Tests} \tag{2.5}$$

Zur Durchführung eines Tests mittels Fehlerinjektion sind einige Voraussetzungen notwendig. Für diese Tests wird eine Schaltungssimulation durchgeführt, für die ein geeignetes Simulationsmodell vorhanden sein muss. Je nach Detaillierungs-

grad des Modells kann es schwierig sein die zu testenden Ressourcen eindeutig zu bestimmen. Weiterhin können bei der Verfeinerung des Modells hin zur physikalischen Implementierung Verfälschungen des Ergebnisses auftreten, d.h. nicht alle Ressourcen, die in einem High-Level-Modell empfindlich sind, müssen auch in einem realen Chip empfindlich sein und umgekehrt. Insbesondere die für gewöhnlich verwendete Schaltungssynthese kann starke Abweichungen in der Empfindlichkeit einer Implementierung gegenüber der Schaltungsbeschreibung verursachen. Hierzu sollen allgemein anwendbare Techniken für den Schaltungsentwurf wie Retiming, Register Duplication, State-Machine-Coding, Pipeline Unrolling und parallele Implementierungen genannt werden. Für ein vorhandenes High-Level-Modell einer Schaltung kann nach einer Synthese nicht präzise vorhergesagt werden, welche und wie viele Ressourcen überhaupt in der Schaltung eingesetzt werden.

Weitere Problematik der Fehlerinjektion ist das Vorgehen zum gezielten Einspeisen eines Fehlers. Hier existieren für eine simulative Fehlerinjektion transienter Fehler zwei Ansätze. Zum Einen kann eine Möglichkeit zur Erzeugung von Fehlern durch direkten Eingriff in den Simulator geschaffen werden. Dies ist der Fall, wenn der Simulator speziell für die Fehlerinjektion gemacht wurde oder eine Schnittstelle zur Anbindung von User-Erweiterungen hat. Letztendlich wird in beiden Varianten der Zustand der simulierten Schaltung während der Laufzeit des Simulators verändert. Um aber die notwendige Änderung des Simulatorzustandes durchführen zu können, ist für jeden Typ verwendeter Ressource unter Umständen ein anderes Vorgehen notwendig, da sich die Implementierung der Simulationsmodelle unterscheiden können. Detailwissen ist hier also notwendig, zusätzlich kann der momentane Zustand auch die Auswahl des Vorgehens beeinflussen, d.h. es muss Interaktion mit dem Simulator stattfinden.

Zum Anderen kann die Schaltung selbst modifiziert werden, um zusätzliche Steuersignale einzubauen, mit denen ein Fehler erzeugt werden kann. Dieses Vorgehen beeinflusst aber durch die zusätzlichen Komponenten zur Fehlerinjektion auch das Verhalten der Schaltung selbst. Der Einfluss dieser Modifikationen sollte aber relativ gering bleiben, damit keine großen Abweichungen vom Verhalten der unmodifizierten Schaltung entstehen. Problematisch gestaltet sich hier das zeitliche Verhalten der Fehlerinjektion. Im Gegensatz zur Simulation von Schaltungen mit permanenten Fehlern, wo hier ein Signal zu Fehlererzeugung dauerhaft gesetzt würde, muss zur Erzeugung transienter Fehler auch das Verhalten der Schaltung berücksichtigt werden. Wenn z.B. an einem Register das Auftreten eines SEUs simuliert werden soll, was durch Einsetzen eines XOR-Gatters erreicht werden kann, so wird das entsprechende Steuersignal für die gesamte Fehlerzeit gesetzt. Problematisch an diesem Ansatz ist aber die notwendige Rückkopplung des Schaltungsverhaltens auf das Fehler-Steuersignal. Für eine korrekte Realisierung eines SEU-ähnlichen Verhaltens eines Registers wird das Steuersignal zum Zeitpunkt des Fehlers gesetzt und muss zurückgesetzt werden, wenn durch die normale Funktion des Registers der Fehler wieder korrigiert werden würde. Da für diese Funktionalität das Fehlerinjektionssystem weiter mit der

Schaltung verknüpft werden müsste, ist eine Realisierung ohne Beeinflussung des ursprünglichen Schaltungsverhalten nochmals schwieriger. Hingegen eine einfache Erzeugung eines SETs erfordert keine Rückkopplung von der Schaltung, durch die Impulsbreite des Steuersignals kann die Länge des SETs direkt gesteuert werden.

Basierend auf dem Prinzip der Fehlereinspeisung durch Modifikation der Schaltung kann die Fehlerinjektion auch in Hardware durchgeführt werden. Dabei wird die zum Zwecke der Fehlerinjektion modifizierte Schaltung in der Regel auf einer Prototyping-Plattform realisiert. Während die Schaltung betrieben wird muss eine geeignete Testumgebung die Eingangssignale der Testbench bereitstellen, die Ansteuerung der Fehlerinjektion und auch die Auswertung der Ausgangssignale übernehmen. Speziell bei größeren Schaltungen mit vielen Registern bereitet die große notwendige Anzahl an Steuersignalen für die Fehlerinjektion Probleme bei der Umsetzung der Schaltung auf der Prototyping-Plattform, so dass diese Signale nicht als primäre Eingänge zum System realisiert werden können. Daneben bedeutet das Verwenden einer Emulations-Plattform auch den Verzicht auf Beurteilung der zeitlichen Maskierung, da das Timing-Verhalten anders als in der echten Hardware ist. Die restlichen digitalen Maskierungseffekte können aber ohne Einschränkungen betrachtet werden.

Ein wesentlicher Vorteil der Fehlerinjektion in Hardware ist die hohe Verarbeitungsgeschwindigkeit. Anders als bei einer Schaltungssimulation mit Fehlerinjektion arbeitet die Hardware-Schaltung in Echtzeit und die Schaltungsgröße hat keinen Einfluss auf die Arbeitsgeschwindigkeit. Im Vergleich zur Simulation lassen sich damit lange Beobachtungszeiten realisieren, ohne die reelle Ausführungszeit einer Fehlerinjektions-Kampagne drastisch zu verlängern. Dagegen ist die schlechtere Steuerbarkeit des Modells bei Hardware-Fehlerinjektion ein deutlicher Nachteil für diesen Ansatz. In der Software-Fehlerinjektion kann der Simulator-Zustand gespeichert werden und ohne großen Aufwand zu einem späteren Zeitpunkt wiederhergestellt werden. Dies erlaubt nach durchlaufener Beobachtungszeit einen weiteren Test mit dem gleichen Zustand der Schaltung, aber einer anderen Ressource, oder die Fortsetzung der Simulation um einen späteren Testbench-Zeitpunkt zu erreichen. Bei geringem Aufwand zur Wiederherstellung des Schaltungszustandes im Vergleich zum erneuten Durchlaufen der Vorlauf-Phase kann die Verwendung von solchen "Checkpoints" zu einer Beschleunigung der gesamten Fehlerinjektion führen.

Auch der zugrunde liegende Anwendungsfall der Schaltung hat Einfluss auf die mit der Fehlerinjektion gewonnenen Ergebnisse. Dies bietet zum Einen den Vorteil, die Schaltung im Hinblick auf einen spezifischen Anwendungsfall testen zu können. Die gewonnenen Ergebnisse können zur Identifizierung besonders empfindlicher Schaltungsteile verwendet werden. Tritt allerdings in Realität ein anderer Anwendungsfall auf, so sind die Ergebnisse aus dem Testszenario nur bedingt oder evtl. gar nicht übertragbar. Da bei unterschiedlichen Anwendungsfällen das Verhalten der Schaltung nicht identisch ist, unterscheidet sich auch das Toleranz-Verhalten bzw. die Fehleranfälligkeit der Schaltung. Bei einer Vielzahl

an möglichen Anwendungsfällen kann eine Mittelung der Ergebnisse unterschiedlicher Szenarien sinnvoll sein.

Da bei der Fehlerinjektion die Experimente unabhängig voneinander sind, kann dieser Umstand zur Parallelisierung der Aufgabe und damit zur Beschleunigung der gesamten Kampagne genutzt werden. So kann z.B. das Testsetup um weitere fehlerbehaftete Schaltungen erweitert werden, was den Aufwand zur Simulation der Referenzschaltung im Vergleich zum Rest reduziert. Weiterhin können mit speziellen Simulatoren aus dem Bereich der Fehlersimulation mehrere identische logische Berechnungen mit unterschiedlichen Werten parallel auf einem Simulationsrechner ausgeführt werden. Dabei kann die volle Busbreite einer ALU verwendet werden, d.h. momentan bis zu 64 bit. Dies kann natürlich nur angewendet werden, wenn komplexe Logikfunktionen auf die grundlegenden Funktionen herunter gebrochen werden.

### 2.3.2 Vorwärts-Analyse

In der Veröffentlichung von Li et al. [19] wird ein interessanter Ansatz zur Bestimmung der Empfindlichkeit von digitalen Schaltungen gezeigt. Alle Vorgänge werden während einer Zyklen-genauen Simulation der Schaltung unter Verwendung eines Performance-Modells durchgeführt [20]. Bei dem vorgestellten Ansatz SoftArch wird grundsätzlich jeder Speicher in der Schaltung (Register, Latch) als potentielle Fehlerquelle eines spezifischen Fehlers angesehen. Diese einzelnen Fehlerorte werden dann unter Verwendung der vorhandenen kombinatorischen Funktionen zusammengefasst um die Abhängigkeit der Schaltungssignale untereinander darzustellen, d.h. welche Fehler einen Einfluss auf welche Signale haben könnten. Diese Fehlergruppen wiederum werden dann, gemäß dem Signalfluss in der Schaltung, weiterpropagiert. Wenn wiederum einzelne Signale von mehreren Fehlergruppen abhängen, so führt dies zu einer Vereinigung der Fehlergruppen.

Da beim Propagieren der Fehlergruppen der Schaltungszustand berücksichtigt wird, kommt der zeitliche Einfluss zu den bestehenden Ergebnissen hinzu. Für jede Fehlergruppe muss bestimmt werden, wann sie einen spezifischen Speicher betrifft. Dies kann durch Überwachung der Schreibzugriffe auf die Speicherzellen erreicht werden, die Fehlergruppen erhalten dadurch einen Zeitstempel, der den Gültigkeitsbereich definiert. Wenn eine Zelle neu beschrieben wird, egal ob sich neuer und vorheriger Wert unterscheiden, bedeutet dies, das alle Fehlergruppen von Signalen, die zur Berechnung des neuen Wertes verwendet wurden, in der neu beschriebenen Zelle vereinigt und gespeichert werden. Dadurch wird erreicht, dass auch in Schaltungen mit mehreren sequentiellen Verarbeitungsstufen die Abhängigkeiten aller Signale bestimmt werden können.

Bei dieser Art der Analyse müssen vorab die kritischen Ausgänge bzw. die beobachtbaren Signale definiert werden. Dabei handelt es sich um eine Entscheidung des Anwenders bei der die besonderen Gegebenheiten des betrachteten Systems berücksichtigt werden können. In den Anwendungsfällen von Li [19] wurde ein

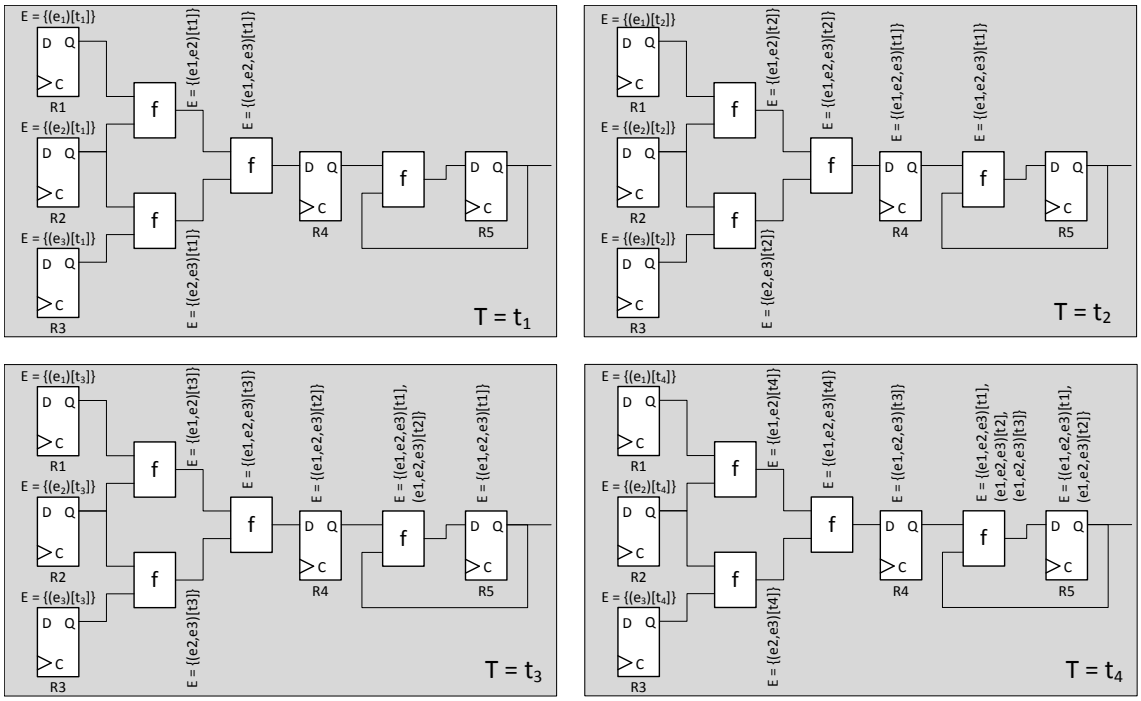



Abbildung 2.7: Fehler-Propagation bei SoftArch

moderner superskalarer Prozessor ähnlich der IBM POWER4 Architektur [21] betrachtet, wobei die Speicher für Instruktionen, für den TLB und die Speicher außerhalb des Prozessorkerns als kritisch betrachtet wurden. Wann immer beim Weiterpropagieren von Fehlergruppen einer dieser zuvor spezifizieren Speicher erreicht wird, so kann die Aussage getroffen werden, dass diese Fehler Einfluss auf diesen Speicher haben würden. Je nach sequentieller Tiefe und Struktur der betrachteten Schaltung muss unterschiedlich lange simuliert werden, bis eine Aussage über die Empfindlichkeit eines Speichers getroffen werden kann. Die gleiche Problematik wurde schon bei der Fehlerinjektion beobachtet.

In Abbildung 2.7 wird die Ausbreitung und Weiterleitung der Fehlergruppen für mehrere Zeitschritte dargestellt. Dabei sind auch einige problematische Fälle dargestellt, die bestimmte systematische Schwächen der Vorwärts-Analyse aufzeigen. Zuallererst sei auf die Gefahr der Ansammlung von Fehlermarkierungen hingewiesen. Am Register R4 sind zu jedem Zeitpunkt die Fehlermarkierungen aller Vorgänger-Register vom vorherigen Zeitschritt vorhanden. Je nach tatsächlich vorhandener Logik-Struktur der betrachteten Schaltung kann die Anzahl der Vorgänger-Register und damit die Anzahl der Markierungen sehr hoch ausfallen. Zudem entstehen im Falle eines Fan-Out von mehr als eins weitere Kopien der Fehlermarkierungen. Ein typischer, nicht ungewöhnlicher Anwendungsfall, in dem diese Eigenschaft negativ zum Tragen kommt, sind einfache arithmetische Schaltungen, hier steigt die Anzahl der Fehlermarkierungen vom LSB zum MSB der

Ausgangsregister treppenförmig an. Die Anzahl der dabei entstehenden Markierungen für eine arithmetische Funktion mit zwei Eingangsworten der Bitbreite N ist näherungsweise $\frac{1}{2} \cdot N^2$. Durch die Ansammlung von Fehlermarkierungen steigt auch der Rechenaufwand bei der Durchführung des Verfahrens. In der Implementierung von SoftArch werden die Auswirkungen dieser Eigenschaft auf die Ausführungsgeschwindigkeit und den Speicherbedarf begrenzt, indem die maximale Anzahl der Fehlermarkierungen pro Speicherzelle auf 100 limitiert wird. Wird diese Grenze überschritten werden ältere Markierungen gelöscht, bis die Grenze wieder eingehalten wird. Mit diesem Vorgehen geht prinzipiell ein Verlust der Genauigkeit einher, da nicht ausgeschlossen ist, dass diese Markierungen evtl. noch bis zu einem Ausgang propagiert werden könnten.

Ein besonderes Problem tritt bei Schaltungen mit interner Rückkopplung auf, dies ist in Abbildung 2.7 mit den Registern R4 und R5 dargestellt. Systembedingt wächst die Anzahl der Fehlermarkierungen hier stetig an, da bereits vorhandene Fehlermarkierungen immer wieder durch die Rückkopplung an die gleiche Stelle gelangen und aber trotzdem weitere Fehlermarkierungen von Schaltungsteilen hinzu kommen, die nicht Teil der Rückkopplung sind. Wenn in der Schaltung von der Rückkopplung zu einem Ausgang relativ selten ein empfindlicher Pfad besteht, dann besteht die Gefahr, dass auf Grund der Begrenzung der Anzahl der Fehlermarkierungen einige Markierungen verworfen werden, die Einfluss auf den Ausgang haben könnten. Diese Schaltungen mit Rückkopplung zeichnen sich auch dadurch aus, dass sie eine sehr hohe, möglicherweise unbegrenzt lange maximale Fehlerlatenz haben. Für alle diese Schaltungen liefert SoftArch mit hoher Wahrscheinlichkeit falsche Ergebnisse der Fehlerempfindlichkeit. Zusätzlich tritt bei diesen Schaltungen auch das vorher beschriebene Problem der Ansammlung von Fehlermarkierungen auf, d.h. sowohl die Ausführungsgeschwindigkeit als auch die Genauigkeit der Ergebnisse kann von der Schaltungsstruktur stark abhängen.

Häufig kommt es in Schaltungen vor, dass bestimmte Register relativ selten getriggert werden, die speichernde Zelle wird daher selten neu beschrieben. Wird diese Zelle jedoch oft von nachfolgenden logischen Funktionen gelesen, so entsteht eine Vielzahl an Fehlermarkierungen, welche alle vom gleichen gespeicherten Wert bzw. der gleichen Fehlerquelle abhängen. Dadurch ergibt sich eine Dominanz von Fehlergruppen, die zu einem späteren Zeitpunkt weiterpropagiert werden. Ein Fehler, der sehr spät über der Lebensdauer eines gespeicherten Wertes auftritt, wirkt sich sicherlich nicht auf frühe Lesezugriffe dieses Wertes als Fehler aus, weil zu diesem Zeitpunkt der gespeicherte Wert noch nicht fehlerhaft war. Ein Fehler der jedoch früh, d.h. kurz nach dem Beschreiben der Zelle auftritt, wirkt sowohl auf früh als auch auf späte Lesezugriffe als Fehler. Daher dominieren später weiterpropagierte Fehlergruppen ihre zeitlichen Vorgänger. Aus Effizienzsicht der Vorwärts-Analyse ist eigentlich die Propagation von frühen Fehlergruppen unnötig, da sich dadurch keine weiteren Erkenntnisse gewinnen lassen. Vielmehr muss bei der Auswertung der Fehlergruppen, bei Erreichen eines beobachtbaren Ausgangs der Schaltung, diese Vereinigung und Überlappungsprüfung durchgeführt

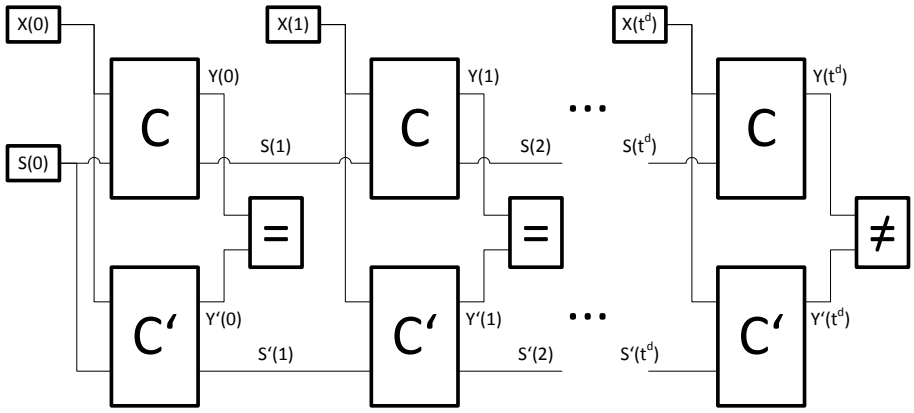



Abbildung 2.8: Test-Szenario für Anwendung eines SAT-Solvers

werden, damit die Ergebnisse konsistent sind. Zusätzlich zu diesem unnötigen Aufwand kommt der Aufwand für die Kopier-und Verschiebeoperationen der irrelevanten Fehlergruppen. Die zuvor besprochene Begrenzung der maximalen Anzahl der Fehlermarkierungen durch Verwerfung der ältesten Fehlermarkierungen kann die unnötigen Operationen begrenzen, jedoch nicht vollständig verhindern.

### 2.3.3 SAT-Solver

Neben den herkömmlichen Methoden, in denen konkrete Anwendungsfälle der Schaltungen untersucht werden, haben Fey et al. [22] ein Verfahren vorgestellt, welches unter Verwendung eines SAT-Solvers ohne Anwendungsdaten die Empfindlichkeit einer Schaltung charakterisieren kann. Dabei wird die Problematik der zu betrachtenden Fehler in eine formale Darstellung umgewandelt und mittels SAT-Solver Lösungen gesucht, die Rückschlüsse auf die Schaltungsempfindlichkeit zulassen.

In Abbildung 2.8 wird das verwendete Schema zur Analyse der Schaltung dargestellt. Der Block C stellt dabei die ausgerollte Schaltung dar welche neben den primären Ein- und Ausgängen zusätzlich über Ein- und Ausgänge für den Schaltungszustand verfügt. Die Schaltung wird dabei maximal bis zu einer Tiefe von $t^d$ ausgerollt, was der maximalen sequentiellen Tiefe der Schaltung entspricht. Mehrere ausgerollte Schaltungsinstanzen werden hintereinander gehängt, dadurch kann das Verhalten der Schaltung für die selbe Anzahl an Taktzyklen abgebildet werden. Eine solche ausgerollte Schaltung wird als Referenzschaltung daneben verwendet. Im Block C' können einzelne Schaltungskomponenten als fehlerhaft betrachtet werden. Weiterhin wird angenommen, das die primären Schaltungsausgänge beider Ketten in allen Stufen, ausgenommen der letzten Stufe, gleiche Werte aufweisen müssen.

Im folgenden wird nun ein Test für jede in der Schaltung vorhandene Komponente durchgeführt, d.h. diese wird als fehlerhaft betrachtet. Eine fehlerhafte

Komponente $g$ wird im Testsystem durch eine gesetztes Fehlerprädikat $p_g$ dargestellt, wobei jeweils nur ein Fehlerprädikat gesetzt ist bzw. eine Komponente fehlerhaft sein darf. Auch bei diesem Verfahren werden nur Einzelfehler betrachtet. Mehrfachfehler sind nach Ansicht der Autoren nicht zu betrachten, da diese allerhöchstwahrscheinlich zu katastrophalem Fehlerverhalten führen würden, jedoch durch eine in die Schaltung integrierte Fehlererkennung abgefangen werden können.

Der verwendete SAT-Solver wird nun benutzt, um für das Test-Szenario mit einem spezifischen gesetzten Fehlerprädikat Eingangsmuster zu suchen, die zu einem Unterschied beim Vergleich der primären Ausgänge der letzten Stufe führen. Dabei wird prinzipiell ein sehr großer Zustandsraum durchsucht.

$$N_{StateSpace} = 2^{(N_{State}+N_{PI})\cdot t^d} \tag{2.6}$$

$N_{StateSpace}$ steht dabei für die maximale Größe des Zustandsraumes, $N_{State}$ für die Anzahl der in der Schaltung vorhandenen Zustandsbits, $N_{PI}$ für die Anzahl der primären Eingänge und $t^d$ für die maximale sequentielle Tiefe der Schaltung. Hier ist einfach zu erkennen, das schon eine kleine maximale sequentielle Tiefe der Schaltung den Suchraum dramatisch vergrößert. Allerdings wird die in den meisten Fällen unrealistische Annahme getroffen, dass alle Schaltungszustände erreicht werden können.

Zur Verringerung der Problemgröße haben Fey et. al. [22] die Menge der zulässigen Zustände eingeschränkt. Ausgehend von einem relativ kleinen Satz an bekannten erreichbaren Zuständen, z.B. dem Reset-Zustand der Schaltung, wird eine vereinfachte Erreichbarkeits-Analyse durchgeführt. Das Ergebnis dieser Analyse ist eine Unter-Abschätzung der erreichbaren Zustände $S\downarrow$.

Jeder Zustand aus der eben definierten Unterabschätzung der erreichbaren Zustände $S\downarrow$, wie auch Zustände aus einer Überabschätzung $S\uparrow$, können in der Analyse als Ausgangssituation $S(0)$ (siehe Abbildung 2.8) verwendet werden. Der SAT-Solver wird nun benutzt um unter folgenden Randbedingungen Lösungen, d.h. Eingangvektoren der Schaltung zu suchen:

- Die primären Ausgangssignale der letzten Instanz der ausgerollten Schaltung müssen unterschiedlich sein, d.h. der Fehler ist erst in der letzten ausgerollten Instanz beobachtbar. Alle vorhergehenden Instanzen müssen identische Ausgangssignale haben.
- Der Anfangszustand der Schaltung ist bei beiden Schaltungen identisch und darf nur aus einer vorher festgelegten Menge an Zuständen entnommen werden.
- Die primären Eingangssignale beider ausgerollter Schaltungen müssen identisch sein, dürfen aber ohne Einschränkungen gewählt werden.
- Jeweils ein Fehlerprädikat muss gesetzt sein.

Sobald eine Eingangskombination in Zusammenhang mit einem Fehlerprädikat gefunden wurde, wird die Suche weiterer Lösungen für dieses Prädikat abgebro-

chen und die von diesem Prädikat beeinflusste Schaltungskomponente als nicht robust betrachtet. Ein gefundener Eingangsvektor würde die Auswirkungen einer fehlerhaften Komponente am Schaltungsausgang sichtbar machen, der Fehler hätte also einen beobachtbaren Effekt. Zur Klassifizierung als robuste Komponente müssen bei gesetztem Fehlerprädikat die Schaltungsausgänge innerhalb der betrachteten Ausrolltiefe $t^d$ und der Schaltungszustand der letzten Instanz identisch sein. Wenn auch am Schaltungszustand der letzten Instanz kein Unterschied mehr besteht, können Fehler mit sehr hoher Latenz ausgeschlossen werden.

Das vorgestellte Verfahren versucht in einem iterativen Prozess alle anfangs nicht klassifizierten Komponenten nach und nach in die Kategorien eindeutig robust oder nicht robust einzuordnen. Da nicht alle Komponenten eine Fehlerlatenz von $t^d$ haben, wird zunächst mit einem vereinfachten Testsetup begonnen, welches nur aus einer einzelnen ausgerollten Schaltungsinstanz besteht. Mit diesem Setup wird dann begonnen alle Komponenten, die eine Fehlerlatenz von einem Takt besitzen, zu klassifizieren. Dabei bleiben alle Komponenten mit einer Fehlerlatenz größer als eins unklassifiziert. Zu diesem Zweck wird das Testsetup dann stetig um eine weitere ausgerollte Schaltungsinstanz erweitert, um alle Komponenten mit höherer Fehlerlatenz klassifizieren zu können. Bei insgesamt $t^d$ ausgerollten Schaltungsinstanzen, oder wenn keine unklassifizierten Komponenten mehr vorhanden sind, endet der Algorithmus.

Großer Vorteil dieses Verfahrens ist seine Unabhängigkeit von einem konkreten Anwendungsfall, lediglich die Menge der Anfangszustände muss von Hand spezifiziert werden. Das Verfahren ist also nicht von einem Testmustersatz abhängig, der evtl. nur eine mangelnde Abdeckung der Schaltung erreichen könnte. Weiterhin kann dies auch als Vorteil während der Entwicklungsphase gesehen werden, in der evtl. noch keine Testbenches vorhanden sind. Zudem werden durch das formale Vorgehen auch unrealistische oder unplausible Anwendungsfälle berücksichtigt, was theoretisch eine vollständige Testabdeckung ermöglicht.

Genau diese Eigenschaften führen aber auch zu Nachteilen. Einer realistischen Verteilung der Eingangsmuster und Schaltungszustände wird keinerlei Rechnung getragen. Schon ein einzelnes, einfach zu findendes Testmuster, welches eine bestimmte Komponente testen könnte, führt zur Annahme die Komponente sei nicht robust. Ob der notwendige Schaltungszustand dafür häufig in der Praxis eintritt wird nicht bewertet. Generell, eine rein diskrete Bewertung der Robustheit einer Schaltung bietet wenig Differenzierungsmöglichkeiten. Auf Grund des Zustandsraumes der durchsucht wird, kann erwartet werden, das nahezu alle Komponenten als nicht robust angesehen werden. Einzige Ausnahme bilden Schaltungsteile die Teil eines Fehlerkorrekturmechanismus sind. Diese Unterabschätzung der Robustheit tritt vor allen Dingen in Verbindung mit der Überabschätzung der gültigen Anfangszustände $S\uparrow$ auf. In den Ergebnissen der Arbeit [22] für Schaltungen ohne Fehlerschutz werden fast alle Komponenten als nicht robust betrachtet. Bei genauerer Untersuchung der gezeigten Ergebnisse für mehrere ITC'99-Schaltungen [23] sind noch weitere Ungereimtheiten zu erkennen, die auf man-

gelnde Stabilität und Zuverlässigkeit des Verfahrens schließen lassen. Theoretisch sollte mit steigender Anzahl der ausgerollten Schaltungsinstanzen die Anzahl der unklassifizierten Komponenten stetig zurückgehen, mindestens aber gleich bleiben. Bei einigen Schaltungen wird diese Stetigkeit verletzt und die Anzahl der unklassifizierten Komponenten steigt wieder. Zusätzlich zu dieser Problematik ist häufiger auch der Anteil der unklassifizierten Komponenten relativ hoch, z.B. bei b05, b07, b12 und b13. Dadurch sind die Ergebnisse für diese Schaltungen nur ungenügend abdeckend.

### 2.3.4 Probabilistische Ansätze

Auf Basis von Signalwahrscheinlichkeiten präsentierten Asadi et al. [24] 2005 einen Ansatz zur Bestimmung der Fehlerwahrscheinlichkeit von digitalen Schaltungen. Basierend auf einer Schaltungssimulation werden die Signalwahrscheinlichkeiten für jedes Signal bestimmt. Die Schaltungsbeschreibung wird dann in einen gerichteten Graphen überführt, welcher für weitere Berechnungen als Basis dient.

Für jedes in der Schaltung vorhandene Signal werden die möglichen Pfade für die Fehlerausbreitung bestimmt. Alle Signale, die auf den möglichen Fehlerpfaden liegen, werden im Weiteren als On-Path-Signal bezeichnet. Alle anderen Signale, die nicht direkt im Fehlerpfad liegen, jedoch die Fehlerpropagation beeinflussen können, werden als Off-Path-Signal bezeichnet. Mittels mehrerer Regeln für die elementaren Logikgatter können nun Gleichungen für die Fehlerausbreitungswahrscheinlichkeit bestimmt werden. Dabei werden für den Gatterausgang vier Fälle berücksichtigt:

- Gatterausgang Logisch '0'
- Gatterausgang Logisch '1'
- Gatterausgang $a$: Fehlerhafter Eingangswert wird weitergegeben
- Gatterausgang $\bar{a}$: Fehlerhafter Eingangswert wird invertiert weitergegeben

Mit dieser Unterscheidung der Signalwerte ähnlich dem D-Algorithmus [25] ist es unter Zuhilfenahme aller Off-Path-Signalwahrscheinlichkeiten möglich, die Wahrscheinlichkeit einer Abweichung der primären Ausgangssignale zu bestimmen. Weiterhin wird mit dem gleichen Ansatz auch die Fehlerausbreitungswahrscheinlichkeit hin zu internen Registern bestimmt. Primäre Ausgänge und interne Register werden also gleichgestellt. In einem weiteren Schritt wird dann die Wahrscheinlichkeit der Entstehung eines Fehlers mit einbezogen. Damit kann für jede Systemkomponente eine Fehlerwahrscheinlichkeit bestimmt werden.

Für eine größere Auswahl an Testschaltungen aus den ISCAS89-Benchmarks [26] wurde das Verfahren durchgeführt und die gewonnenen Ergebnisse der Gesamtfehlerwahrscheinlichkeit der Schaltung mit Ergebnissen aus einer Fehlerinjektion verifiziert. Nach Angaben der Autoren ergab sich im Mittel eine Abweichung von 5% der gewonnenen Fehlerwahrscheinlichkeiten. Bei den kleineren Schaltungen wurde die Fehlerinjektion in allen Komponenten durchgeführt, bei den grö-

$$M = \begin{pmatrix} EPP_{1,1} & EPP_{2,1} & ... & EPP_{n,1} \\ EPP_{1,2} & EPP_{2,2} & ... & EPP_{n,2} \\ ... & ... & ... & ... \\ EPP_{1,n} & EPP_{2,n} & ... & EPP_{n,n} \end{pmatrix} \tag{2.7}$$

Matrix mit Wahrscheinlichkeiten der Fehlerpropagation (Error Propagation Probability EPP)

$$S = \begin{pmatrix} P(FF_1) \\ P(FF_2) \\ ... \\ P(FF_n) \end{pmatrix} \tag{2.8}$$

Vektor mit gegenwärtigen Fehlerwahrscheinlichkeiten einzelner Register

ßeren Schaltungen wurden die Test stichprobenartig durchgeführt. Im Vergleich zur Laufzeit der Fehlerinjektion ist das vorgestellte probabilistische Verfahren um mehrere Größenordnungen schneller.

Leider werden bei diesem Verfahren die Abhängigkeiten der Signale untereinander nicht berücksichtigt. Auf Grund der logischen Abhängigkeiten der Signale untereinander muss von teilweise stark korrelierten Signalen bzw. Signalwahrscheinlichkeiten ausgegangen werden. Dadurch verliert der probabilistische Ansatz bei Signalabhängigkeiten seine Gültigkeit, was sich in starken Abweichungen der Fehlerwahrscheinlichkeiten zeigen müsste. In den gezeigten Analyseergebnissen aus [24] werden nur Schaltungsmittelwerte angegeben, eine Verteilung bzw. ein Vergleich der Abweichung bei den Ergebnissen einzelner Signale ist nicht verfügbar. Weiterhin sollte bemerkt werden, dass der verwendete Anwendungsfall zur Bestimmung der Signalwahrscheinlichkeiten damit auch das Ergebnis der Analyse beeinflusst.

In einer weiteren Arbeit [27] wurden die gewonnenen Ergebnisse verwendet um die Fehlerwahrscheinlichkeit von sequentiellen Schaltungen zu bestimmen. Bei diesem Verfahren werden die Fehlerausbreitungseigenschaften der Schaltung strukturiert in einer Matrix dargestellt.

Mit dem vorher vorgestellten Verfahren konnte die Wahrscheinlichkeit einer Fehlerpropagation vom Entstehungsort zum nächsten Register oder primären Ausgang bestimmt werden. Diese einzelnen Wahrscheinlichkeiten werden in eine Matrix $M$ eingetragen.

In der Matrix $M$ (siehe Gleichung 2.7) sind die Wahrscheinlichkeiten für Fehlerpropagation EPP nach Fehler-Entstehungsort (Zeilenindex) und Ziel der Fehlerfortpflanzung (Spaltenindex) eingetragen. Bei der Matrix handelt es sich um eine quadratische Matrix der Dimension $n \times n$, wobei $n$ der Anzahl der Register in der Schaltung entspricht.

Zusätzlich zur Matrix $M$ wird ein Vektor $S$ (siehe Gleichung 2.8) der Größe $n$ definiert, welcher die momentane Fehlerwahrscheinlichkeit jedes Registers

$$S(t=1) = M \times S(t=0) \tag{2.9}$$

Bestimmung der Fehlerwahrscheinlichkeiten für den nächsten Zeitschritt

$$S(t=c) = M^c \times S(t=0) \tag{2.10}$$

Bestimmung der Fehlerwahrscheinlichkeiten für den Zeitschritt c

beinhaltet. Das sequentielle Verhalten der Schaltung kann durch die Gestaltung der Matrix $M$ und des Vektors $S$ in dieser Form recht einfach bestimmt werden. Durch eine einfach Matrix-Vektor-Multiplikation kann die Ausbreitung eines Fehlers bzw. die Fehlerwahrscheinlichkeit einzelner Register für den nächsten Zeitschritt der Schaltung bestimmt werden. (siehe Gleichung 2.9)

Die Anwendung dieses Konzepts kann auch mehrere Male wiederholt werden, dadurch ist die Ausbreitung eines Fehlers auch über mehrere Zeitschritte hinweg bestimmbar. Jede Multiplikation des Vektors $S$ mit der Matrix $M$ bestimmt die Fehlerwahrscheinlichkeiten für den nächsten Zeitschritt. In Gleichung 2.10 ist eine Form für eine beliebige Anzahl an Zeitschritten dargestellt.

Nachdem mit diesem Ansatz die Fehlerwahrscheinlichkeit für jeden Zeitschritt sehr einfach bestimmt werden kann, besteht auch die Möglichkeit die Verteilung der Fehlerwahrscheinlichkeiten zu betrachten. In der Arbeit von Asadi et al. [27] wurde die Mean-Time-to-Manifest MTTM eines Fehlers untersucht. Die MTTM wird hier als Grenze festgelegt, an der die kumulative Wahrscheinlichkeit für das Sichtbarwerden eines Fehlers eins übersteigt. Anhand der MTTM wurde die Einstufung der Register vorgenommen, wobei Register mit kleiner MTTM als kritischer angesehen werden.

Wesentlicher Vorteil dieses Verfahrens ist die geringe Ausführungszeit zur Gewinnung von Ergebnissen. Den Kern des Verfahrens stellt die Matrix-Vektor-Multiplikation dar, deren Aufwand nur linear mit der Problemgröße, d.h. mit der Größe der Schaltung wächst. Als Nachteil des Verfahrens muss aber angesehen werden, dass nur mit Mittelwerten gerechnet wird. Wie schon bei dem zu Grunde liegenden Verfahren zur Bestimmung der Fehlerwahrscheinlichkeit von kombinatorischen Schaltungen werden alle Signale und die Fehlerausbreitung vollkommen unabhängig voneinander angenommen. Da dies nicht immer zutrifft, sind sehr große Schwankungen bei den Ergebnissen zu erwarten, was in dieser Arbeit aber nicht betrachtet wurde.

### 2.3.5 ACE-Analyse

Ein wesentlicher Beitrag zur Bewertung der Empfindlichkeit von Mikroprozessoren wurde von Mukherjee et al. [15] geleistet. Neben grundsätzlichen Definitionen wurden dort mehrere Konzepte zur Bewertung der Empfindlichkeit, Maskierungseffekte und Anwendungsfälle gezeigt.

In seinen Ausführungen zur ACE-Analyse [15] beginnt Mukherjee mit der Definition eines Empfindlichkeitsfaktors, dem sogenannten Architectural Vulnerability Factor AVF. Der AVF entspricht der Wahrscheinlichkeit, mit der ein aufgetretener Fehler eine beobachtbare Auswirkung auf das Verhalten des Prozessors bzw. das Ergebnis des Programms hat. Da eben nicht jedes Zustandsbit zu jedem Zeitpunkt kritische Information speichert, gibt es häufig einen großen Unterschied zwischen der Roh-Fehlerrate und der tatsächlichen Fehlerrate. Der AVF ist für den Bereich 0 bis 1 definiert, wobei 0 einem immer unempfindlichen Speicher und ein AVF-Wert von 1 für einen immer empfindlichen Speicher entspricht.

Zur Bestimmung des AVF ist es natürlich notwendig zu wissen, ob ein Zustandsbit kritisch ist. Alle Zustandsbits, die für eine Architecturally Correct Execution (ACE) benötigt werden, sind kritisch. Un-ACE-Bits haben daher keinen Einfluss auf das Ergebnis.

Im Weiteren beschäftigt sich Mukherjee mit verschiedenen Effekten, die Un-ACE-Bits in einem Mikroprozessor zur Folge haben. Um eine pessimistische Abschätzung der Empfindlichkeit des Prozessors geben zu können, wird davon ausgegangen, das ein Bit ACE ist, solange nicht das Gegenteil bewiesen wird. Aus Sicht der Prozessor-Mikroarchitektur können mehrere Eigenschaften zu Un-ACE-Bits führen:

- Nicht zugewiesener, unbenutzter oder als ungültig markierte Speicher (Idle oder Invalid).
- Bits, die durch einen nicht eingetroffenen Fall bei spekulativer Verarbeitung betroffen sind. Beispiele hierzu sind die Funktionsblöcke Branch-Predictor, Jump-Predictor, Return-Stack-Predictor.
- Bits, die vormals ACE waren

Weiterhin haben bedingt durch die Prozessor-Architektur folgende Instruktionen Un-ACE-Bits zur Folge:

- NOP-Instruktionen: Alle Zustandsbits außer dem Instruction Code einer NOP-Operation in einer Pipeline-Stufe sind unkritisch.
- Instruktionen zur Verbesserung der Prozessor-Performance wie z.B. Cache-Prefetch-Instruktionen.
- Instruktionen mit Gültigkeits-Prädikat, einer Besonderheit des hier betrachteten IA64-Instruction-Set [28]. Unkritisch sind alle Bits einer Instruktion mit Falsch-Prädikat, mit Ausnahme der Prädikat-Bits selbst.
- Dynamically Dead Instructions: Instruktionen, die auf Grund des Ablaufs der Instruktionen keine Auswirkung auf das Programm-Ergebnis haben können. Beispiel hierzu ist ein Register, welches zwar durch eine Instruktion mit einem neuen Wert beschrieben wird, dieser Wert aber niemals verwendet wird und stattdessen ungenutzt überschrieben wird.
- Logische Maskierung: Betrachtet wird hier nur die logische Maskierung der Operanden etwa bei bitweisen Operationen der CPU oder bei bedingten Sprungbefehlen wenn ein Operand gleich oder ungleich Null ist.

Diese vorgestellten Kriterien werden nun benutzt um in einem Programmablauf

$$AVF = \frac{\sum_{All\ bits\ in\ structure} ACE - time\ of\ bit}{Number\ of\ bits\ in\ structure \cdot excecution\ time} \tag{2.11}$$

Adaption des zeitlichen Mittelwertes zur AVF-Bestimmung

$$AVF = \frac{B_{ACE} \cdot L_{ACE}}{Number\ of\ bits\ in\ structure} \tag{2.12}$$

Adaption von Little's Law zur AVF-Bestimmung

für alle Zustandsbits eine Unterscheidung in ACE- und Un-ACE-Bits zu ermöglichen. Eine Bewertung der einzelnen Bits findet anhand der Zeit statt, in der das Bit als ACE markiert ist. Der AVF einer Schaltung bzw. eines Schaltungsteils kann folglich als Mittelwert der ACE-Zeiten eines jeden enthaltenen Bits gesehen werden. In Gleichung 2.11 wird zur Bestimmung des AVF die Summe aller ACE-Zeiten eines jeden Bits, die Gesamtzahl der in der Schaltung vorhandenen Bits und die betrachtete Laufzeit der Schaltung herangezogen.

In diesem Zusammenhang wird zur Abschätzung des AVF einer Struktur auch Little's Law [29] angewandt. Für diese tendenziell ungenauere Abschätzung des AVF wird die Bandbreite an ACE-Bits in die Schaltung und deren mittlere Latenz herangezogen (siehe Gleichung 2.12).

Als dritte Methode, die eine wesentliche Neuerung darstellt, wurde die AVF-Bestimmung mittels Prozessor-Performance-Modell vorgestellt. Ein vorhandenes Performance-Modell eines Prozessors wurde erweitert um Informationen über die Verweildauer von Instruktionen in Teilbereichen der Schaltung zu sammeln. Wenn dann eine Instruktion die Schaltung vollständig durchlaufen hat, wird sie zur weiteren Bestimmung von Abhängigkeiten für einen begrenzten Zeitraum gespeichert. Mittels diesem Speicher soll erkannt werden, ob es sich um Dynamically Dead Instruktionen handelt oder ob die Ergebnisse auf Grund von logischer Maskierung ohne Einfluss auf das Programm bleiben werden. Prinzipiell ist der Zeitraum, in denen sich gegenseitige Abhängigkeiten noch auswirken können, unbegrenzt. Bei den durchgeführten Experimenten stellte sich eine Pufferzeit von 40000 Prozessor-Zyklen als ausreichend heraus. Zur Erkennung der Abhängigkeiten war es nötig eine Liste von Produzenten und Konsumenten der Daten jedes Registers oder Speicherbereichs mitzuführen.

Für die Instruction Queue und die Excecution Units des Prozessors wurde nun spezifiziert, unter welchen Bedingungen und für welche Instruktionen welche Bits als kritisch angesehen werden. Durch diese Spezifikation kann ein Teil der logischen Maskierung des Prozessors auf dem Datenpfad abgebildet werden. Die Autoren stellen aber auch fest, dass Berücksichtigung der logischen Maskierung bei Steuer-Bits eine zu komplexe Aufgabe wäre und deshalb nicht durchgeführt werden konnte. Weiterhin konnte auch nicht die transitive logische Maskierung berücksichtigt werden. Die rückwirkende Maskierung der Eingangsdaten späterer

Instruktionen die ihrerseits wiederum logisch maskiert werden, konnte auf Grund erhöhter Komplexität nicht berücksichtigt werden. Das Einbeziehen dieser Maskierungseffekte in die Untersuchungen hätte zu einer weiteren Verringerung der gewonnenen AVF-Werte geführt.

Mit der vorgestellten Methode zur Bestimmung des AVF war es möglich, Anwendungsfälle des Prozessors mit einer Länge von 100 Millionen Instruktionen mittels eines Performance-Modell zu untersuchen. Verschiedene Testprogramme der SPEC2000 Benchmark Suite [30] wurden betrachtet und detailliert aufgezeigt, welche Mechanismen ACE- und Un-ACE-Bits zur Folge hatten. Der nötige Zeitaufwand für die Analyse und eine Größenabschätzung der betrachteten Schaltung wurde nicht dargestellt.

2005 haben Biswas et al. [31] diese Arbeiten fortgeführt und zusätzlich adressierbare Strukturen in ihre Betrachtungen miteinbezogen. Für ihre Überlegungen wurden zuerst die Folgen von auftretenden Fehlern genauer unterschieden. Dabei wurden verschiedene Fehlerkategorien festgelegt:

- Silent Data Corruption SDC: Hierunter fallen alle Fehler, die nicht vorzeitig erkannt werden und eine Verfälschung des Ausgangs hervorrufen.
- Detected Unrecoverable Error DUE: In die Schaltung integrierte Mechanismen erkennen diese Fehler, können das falsche Ergebnis aber nicht korrigieren. Weiterhin wird zwischen Fehlern unterschieden welche das Ergebnis tatsächlich beeinflussen (True DUE) und welche ohne Auswirkung bleiben (False DUE).

Adressierbare Speicher haben die Eigenschaft, dass meist nur ein geringer Anteil des Speichers, z.B. ein Wort zu 32 bit, angesprochen werden kann, während der überwiegende Teil des Speichers ruht. Mit adressierbaren Speichern in CPUs in Verbindung mit den darauf ausgeführten Programmen zieht große Flexibilität in das System mit ein, dadurch wird in den meisten Anwendungsfällen der Speicher nicht vollständig und die komplette Zeit genutzt. Folgende Ereignisse/Zustände bei der Speichernutzung wurden daher definiert:

- Idle: Am Beginn des betrachteten Zeitraums ist der Speicher ungenutzt.
- Fill: Ein erstmaliges Beschreiben eines Speichers kann den Beginn eines ACE-Zeitblocks markieren.
- Write: Überschreiben eines bereits vorhandenen Speicherinhalts beendet den vorherigen ACE-Zeitblock und markiert den Beginn eines neuen ACE-Zeitblocks.
- Read: Abruf der gespeicherten Information. Ein ACE-Zeitblock kann hier zu Ende sein, aber eine weitere Verwendung des gespeicherten Wertes ist nicht ausgeschlossen.
- Evict: Durch Evict kann ein Speicherbereich gezielt als ungültig markiert werden, ein ACE-Zeitblock wird an diese Stelle definitiv beendet.
- End: Die Verwendung des Speichers nach dem Ende des betrachteten Zeitraumes ist nicht definiert. Es kann daher keine eindeutige Unterscheidung des ACE-Zustandes getroffen werden.

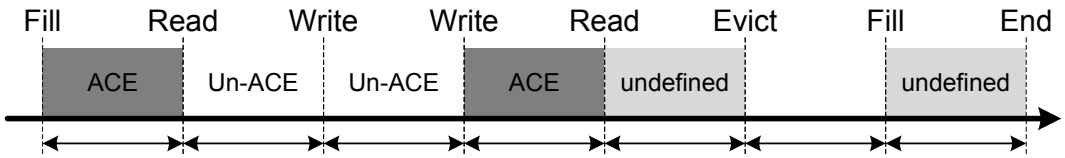



Abbildung 2.9: Ereignisse bei der Speichernutzung und resultierende ACE-Zeitblöcke

Durch die oben definierten Ereignisse ergeben sich mehrere Kombinationen, in denen ein Bit ACE sein kann. Fill und Write markieren den Anfang eines ACE-Zeitblocks, Read und Evict können das Ende eines Blocks festlegen. Auf Grund der Unsicherheit die durch die unbekannte zukünftige Verwendung des Speichers besteht, kann die Zeit bis zum Ende des betrachteten Zeitraumes und zwischen Read- und Evict-Ereignissen nicht eindeutig als ACE oder Un-ACE bestimmt werden (Abbildung 2.9). Durch eine pessimistische Annahme, undefinierte Zeiträume werden als ACE-Zeiten angesehen, kann zumindest eine obere Schranke für den AVF abgeschätzt werden. Zur Vermeidung dieser Unsicherheiten am Ende des Betrachtungszeitraumes wurde das Konzept Cooldown eingeführt. Ähnlich wie bei der Warmup-Phase bei einer Performance-Simulation zum Füllen der Caches wird beim Cooldown die Simulation über das geplante Ende hinweg fortgeführt. Damit werden weitere Daten gesammelt, wie gespeicherte Werte in Zukunft verwendet werden. Die Methodik zur Bestimmung relevanter Zeitbereiche wird aber nicht komplett weitergeführt, es wird lediglich versucht bereits begonnene Zeitbereiche abzuschließen. Je länger die Cooldown-Phase dauert, desto mehr können undefinierte Bereiche reduziert werden.

Die beiden besprochenen Arbeiten [15] [31] zeigen die Möglichkeiten zur AVF-Analyse eines High-Level-Prozessormodells eindrucksvoll auf. Durch die Verwendung eines bereits vorhandenen Modells, welches sehr detailliert die Eigenschaften des betrachteten Prozessors modelliert, ist dieses Verfahren relativ einfach durchführbar. Dennoch ist es notwendig, die Eigenschaften hinsichtlich der Fehlerausbreitung und zur Bestimmung relevanter Zeitbereiche spezifisch von Hand dem Modell hinzuzufügen. Trotz dieses wahrscheinlich hohen zusätzlichen Aufwands erwähnen die Autoren eine Vielzahl an Effekten bei der Fehlermaskierung und bei der Bestimmung von Datenabhängigkeiten, die nicht berücksichtigt werden konnten. Dabei ist es schwierig abzuschätzen, wie groß der Effekt weiterer Verbesserungen der Methodik hinsichtlich einer Verringerung des AVF sein könnte. Durch Betrachtung der logischen und transitiven Maskierung innerhalb der Pipeline werden die größten Anteile erwartet.

## 2.4 Sonstiges, Normen

Zur Erfassung und Bestimmung von Bauteilfehlerraten sind in der Vergangenheit eine Vielzahl an Normen und Standards entstanden. Abhängig vom ursprünglich angedachten Anwendungsbereich der Normen ergeben sich teilweise stark unterschiedliche Herangehensweisen an die Problematik.

### 2.4.1 Bauteilzuverlässigkeit

Die Abschätzung der Bauteilzuverlässigkeit bildet die Grundlage für Abschätzungen einer Systemzuverlässigkeit. Die Zuverlässigkeitsgröße $\lambda$ kann bei einer über der Zeit exponentiell wachsenden Ausfallwahrscheinlichkeit zur Berechnung herangezogen werden (siehe Gleichung 2.13). Weitere Kenngrößen wie die Mean-Time-Between-Failures MTBF (mittlere Zeit zwischen zwei Fehlern, siehe Gleichung 2.14) und die Failure-In-Time-Rate FIT-Rate [32](Fehler pro $10^9$ Betriebsstunden, siehe Gleichung 2.15) sind direkt von $\lambda$ abhängig.

$$p(t) = 1 - e^{-\lambda \cdot t} \tag{2.13}$$

Ausfallwahrscheinlichkeit in Abhängigkeit von $\lambda$

$$MTBF = \frac{1}{\lambda} \tag{2.14}$$

Umrechnung von $\lambda$ in MTBF

$$FIT = \lambda \cdot 10^9 h \tag{2.15}$$

Umrechnung von $\lambda$ in FIT

Bei Systemen, die aus mehreren Bauteilen bestehen und bei denen ein einzelner Ausfall zum Ausfall des ganzen Systems führt, kann die Ausfallrate durch einfaches Aufsummieren der Ausfallraten der einzelnen Bauteile bestimmt werden. Ohne weitere Betrachtung von systembedingter Redundanz oder Fehlerkorrekturmaßnahmen kann dies aber eine Überabschätzung der Ausfallrate zur Folge haben. Als verlässliche Datenquellen für Ausfallraten verschiedenster elektrischer und elektronischer Bauteile haben sich der Standard der US-Streitkräfte MIL-HDBK-217 und der Siemens-Standard SN 29500 durchgesetzt. Während die Zuverlässigkeitswerte der US-Norm nicht mehr gepflegt werden, findet für die Siemens-Norm eine regelmäßige Überarbeitung und Erweiterung der gelisteten Bauteile statt.

Die Annahmen der beiden Standards basieren auf konstanten Umgebungsbedingungen, was in der Realität meistens aber nicht zutrifft. Für alle Systeme, bei denen stark wechselnde Betriebsbedingungen auftreten können, bietet sich

die Anwendung der IEC 62380 zur Bestimmung der Ausfallrate an. Hier werden neben dem Normalbetrieb folgende Zustände berücksichtigt [33]:

- Herstellungsjahr des Bauteils: Veränderungen im Herstellungsprozess können bei gleicher Bauteilspezifikation unterschiedliche Fehlerraten zur Folge haben.
- Wear-Out: Erhöhte Ausfallraten am Ende der erwarteten Lebensdauer des Bauteils.
- Ausfälle auf Grund von kurzzeitiger Überlastung des Bauteils.
- Anwendungsprofil des Bauteils
  - Durchschnittliche Umgebungstemperatur
  - Durchschnittliche Temperatur der Platine/PCB in der Umgebung des Bauteils
  - Jährliche Betriebszeit des Bauteils bei spezifischen Temperaturen
  - Jährliche Betriebszeit des Bauteils im Ruhezustand/Standby
  - Jährliche Anzahl an Temperaturzyklen und durchschnittliche Differenztemperatur

### 2.4.2 Normen für sicherheitskritische Systeme

Der wachsende Einsatz von komplexen Steuerungen in allen Systemen des täglichen Lebens bringt für den Anwender eine Vielzahl von Vorteilen mit sich. Gerade die Automatisierung ist Grund für die stetig wachsende Produktivität, bessere Produktqualität und angenehmere Arbeitsbedingungen der Menschen in den Betrieben. Leider ist der Mensch als Bediener von Maschinen oder in Zusammenarbeit mit diesen in vielen Fällen körperlich unterlegen, so dass die Maschinen auf Grund des Verletzungspotentials eine ernsthafte Gefahr für den Menschen darstellen können. Eine erste Entwicklungsstufe im Gefahrenbewusstsein waren Maßnahmen an den Maschinen, die eine Fehlbedienung oder missbräuchliche Verwendung verhindern sollten. Im weiteren Verlauf der Geschichte wurde immer mehr Wert auf ein geringeres Verletzungsrisiko und Gefahrenpotential gelegt. Damit war es nötig, die Eigenschaften und das Verhalten von solchen Maschinen detaillierter zu untersuchen.

Hauptproblem für einen Hersteller ist nicht etwa die Entwicklung von sicherheitskritischen Systemen, sondern die Zertifizierung dieser Systeme bei Fällen der Produkthaftung. Wenn es zu bedeutenden Schadensereignissen kommt, etwa im Falle der Verletzung oder gar bei Tod eines Menschen in Zusammenhang mit einem Produkt, müssen oftmals Gerichte die Schuld- und Haftungsfrage klären.

Ein fiktiver Gerichtsprozess soll hier die Problematik kurz erläutern. Es treffen zwei Parteien, ein Geschädigter und der Hersteller des Produkts, aufeinander, die natürlich stark unterschiedliche Ansichten haben. Objektivität bei der nachträglichen Bewertung der Beschaffenheit eines Produktes ist demnach schwer herstellbar. Für den Geschädigten sind die getroffenen Sicherheitsvorkehrungen offensichtlich nicht ausreichend gewesen. Der Hersteller beteuert seinerseits, dass

alle gängigen Normen und Vorschriften eingehalten wurden und der Schaden evtl. durch unsachgemäße Verwendung, Missachtung der Betriebsanleitung oder nicht durchgeführte Reparaturen am Gerät zustande gekommen ist. Für eine Entscheidungsfindung werden nun mehrere Gutachten zum betroffenen System erstellt, die nicht immer objektiv sein müssen, da die Interessen des Auftraggebers des Gutachtens bei der Erstellung eine Rolle spielen werden. Weiterhin werden Vergleiche zu ähnlichen Produkten herangezogen, die nicht oder nur schwer vergleichbar sind wegen unterschiedlichem Preis, Herstellungsdatum oder Funktionalität. Eine Einigung wird dann häufig durch einen Vergleich hergestellt, mit dem beide Seiten nicht vollständig zufrieden sind.

Bei einer gerichtlichen Untersuchung eines Schadenfalls wird üblicherweise auch der Stand der Technik zur Beurteilung herangezogen. An dieser Stelle bieten sich natürlich wiederum viele Auslegungsmöglichkeiten, da der allgemeine Stand der Technik nicht exakt abgegrenzt ist. So könnten z.B. Fehlererkennungstechniken aus dem Raumfahrtbereich herangezogen werden, die aber noch in keinem Produkt der Automatisierungsindustrie angewandt worden sind.

Wegen der schwierigen Situation bei der Produkthaftung und den damit unkontrollierbaren Risiken für den Hersteller sehen viele Hersteller von der Entwicklung und Verkauf solcher Produkte ab. Für hochtechnisierte Länder wie Deutschland spielen diese Produkte aber eine sehr große Rolle, zumal bei sicherheitskritischen Systemen die meisten Innovationen zum Einsatz kommen und große Wertschöpfung entsteht. Dementsprechend groß ist das Interesse der Industrie auch in diesem Bereich tätig sein zu können.

Um die Entwicklung sicherheitskritischer Systeme in einem gesicherten rechtlichen Rahmen zu ermöglichen, wurden deshalb verschiedene technische Normen eingeführt um den Zertifizierungsprozess strukturiert und nach einheitlichen Vorgaben durchführen zu können. Normen stellen auch einen definierten Stand der Technik dar, der als Vorlage für allgemein übliches Vorgehen herangezogen werden kann. Die Grundlage für die meisten solcher Normen bildet die internationale Norm "IEC 61508: Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme". Diese Norm wurde erstmals 1998 veröffentlicht, Teile der Norm, die in anderen Normen schon verwendet wurden, reichen bisin Jahr 1990 zurück. Die IEC 61508 bildet eine Sicherheits-Grundnorm, anwendungsspezifische Spezialisierungen der Norm sind möglich. Wann immer keine Spezialisierung der Norm vorhanden ist, fällt das jeweilige Produkt in den Geltungsbereich der IEC 61508.

Eine inzwischen weit verbreitete und mit großem Aufwand erarbeitete Spezialisierung dieser Norm ist die ISO 26262 [34], welche die funktionale Sicherheit für Straßenfahrzeuge behandelt. Wichtiger Bestandteil dieser Norm ist ein definierter Sicherheitslebenszyklus (Abbildung 2.10) und die Einstufung in einen Sicherheitsintegritätslevel ASIL (Automotive Safety Integrity Level). Der Sicherheitslebenszyklus beinhaltet Konzeptionierung, Entwicklung, Herstellung, Inbetriebnahme, Betrieb und Ausserbetriebsetzung des Produkts. In den Entwicklungsphasen soll

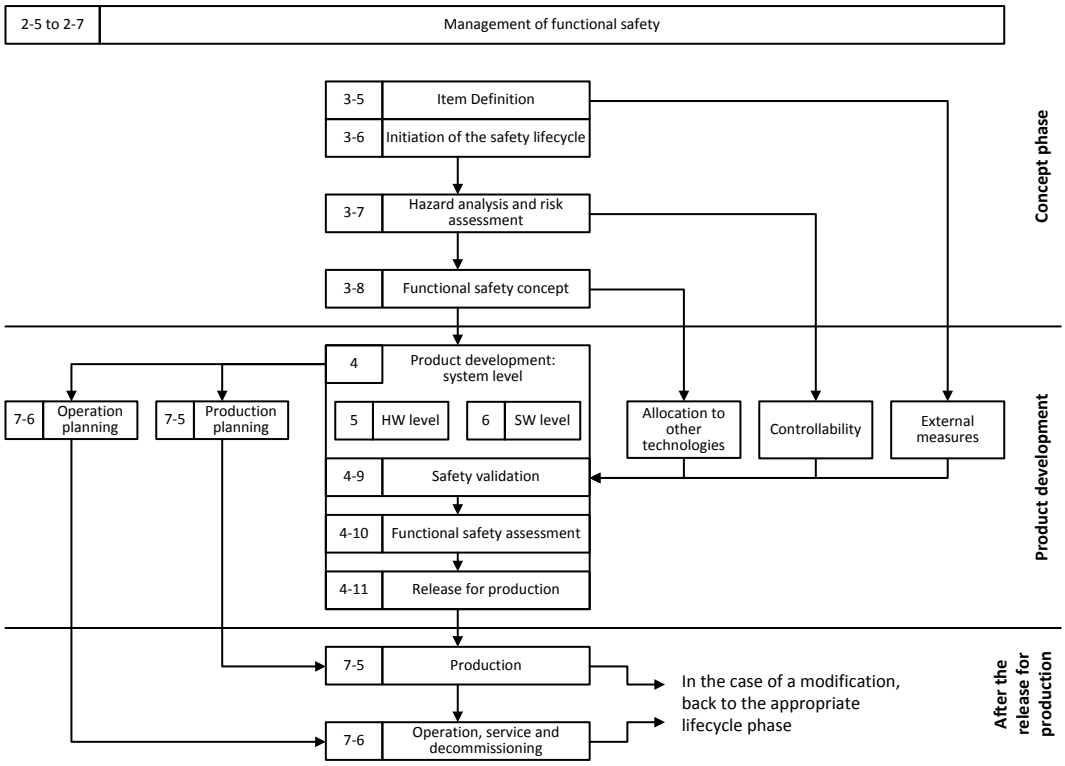



Abbildung 2.10: Gesamter Sicherheitslebenszyklus nach ISO 26262 [35]

eine detaillierte Dokumentation über das Vorgehen und die getroffenen Entscheidungen erstellt werden.

Zur Bestimmung des ASIL werden zuerst drei unterschiedliche Kriterien untersucht. Die Schwere des Schadensausmaßes (Tabelle 2.1), die Häufigkeit des Auftretens (Tabelle 2.2) und die Beherrschbarkeit für den Fahrer (Tabelle 2.3) bei einem Fehler einer Sicherheitsfunktion wird ermittelt und in eine der vorgegebenen Klassen eingeordnet. Mit allen drei Einstufungen zusammen kann aus der Tabelle 2.4 der zugehörige ASIL entnommen werden, wobei ASIL A für die niedrigsten Anforderungen und ASIL D für die höchsten Anforderungen steht. Die Einstufung QM (Quality Management) schließt ein Gefährdungspotential aus.

Aus der ASIL-Einstufung können nun verschiedene Anforderungen an das Produkt und dessen Entwicklungsablauf abgeleitet werden. Insbesondere wird auf notwendige Entwicklungsprozesse, Konzepte zur Fehlererkennung und Redundanz im System bei der Hardware- und Software-Entwicklung eingegangen. In diesem Kontext wird explizit nicht von der Zuverlässigkeit der Sicherheitsfunktion gesprochen. Sicherheitsintegrität beinhaltet sowohl die Zuverlässigkeit dieses Systems als auch die Wirksamkeit von gefahrvermindernden Teilsystemen. Dies bedeutet, das mit einer bestimmten Ausfallwahrscheinlichkeit gerechnet wird, die Zusatzsysteme aber einen Fehler erkennen und schädliche Auswirkungen verhin-

| | Class | | | |
|---|---|---|---|---|
| | S0 | S1 | S2 | S3 |
| Description | No injuries | Light and moderate injuries | Severe and life-threatening injuries (survival probable) | Life-threatening injuries (survival uncertain), fatal injuries |

Tabelle 2.1: Klassen für das Schadensausmaß nach ISO 26262-3 [36]

| | Class | | | | |
|---|---|---|---|---|---|
| | E0 | E1 | E2 | E3 | E4 |
| Description | Incredible | Very low probability | Low probability | Medium probability | High probability |

Tabelle 2.2: Klassen für die Auftrittswahrscheinlichkeit von Situationen im Betrieb nach ISO 26262-3 [36]

| | Class | | | |
|---|---|---|---|---|
| | C0 | C1 | C2 | C3 |
| Description | Controllable in general | Simply controllable | Normally controllable | Difficult to control or uncontrollable |

Tabelle 2.3: Klassen der Beherrschbarkeit nach ISO 26262-3 [36]

dern oder zumindest auf ein erträgliches Maß reduzieren sollen.

Für die Betrachtung der Sicherheitsintegrität werden folgende Fehlerarten nach ISO 26262-10 [37] definiert:

- Single-point fault: Einzelne Fehler, die direkt zur Verletzung eines Sicherheitsziels führen
- Residual fault: Fehler, die zu einer Verletzung eines Sicherheitsziels führen und zudem nicht von einem Sicherheitsmechanismus erkannt werden
- Detected dual-point fault: Ein unabhängiger Fehler, der zusammen mit einem weiteren unabhängigen Fehler zu einem Fehlerverhalten führt, welches von einem Sicherheitsmechanismus erkannt wird
- Perceived dual-point fault: Ein unabhängiger Fehler, der zusammen mit einem weiteren unabhängigen Fehler zu einem Fehlerverhalten führt, welches nur vom Fahrer wahrgenommen wird
- Latent dual-point fault: Mehrere unabhängige Fehler, die weder vom Fahrer noch von Sicherheitsmechanismen erkannt werden
- Safe fault: Fehler, deren Auftreten die Wahrscheinlichkeit zur Verletzung eines Sicherheitsziels nicht signifikant erhöht

Eine Besonderheit, die dem Stand der Technik Rechnung trägt, ist die gesonderte Betrachtung von transienten Fehlern in digitalen Schaltungsbestandteilen. Speziell für Microcontroller, RAM- oder EEPROM/Flash-Bausteine sollen die

| Severity class | Probability class | Controllability class | | |
|---|---|---|---|---|
| | | C1 | C2 | C3 |
| S1 | E1 | QM | QM | QM |
| | E2 | QM | QM | QM |
| | E3 | QM | QM | A |
| | E4 | QM | A | B |
| S2 | E1 | QM | QM | QM |
| | E2 | QM | QM | A |
| | E3 | QM | A | B |
| | E4 | A | B | C |
| S3 | E1 | QM | QM | A |
| | E2 | QM | A | B |
| | E3 | A | B | C |
| | E4 | B | C | D |

Tabelle 2.4: ASIL-Bestimmung nach ISO 26262-3 [36]

Auswirkungen von transienten Fehlern separat betrachtet werden. Zielwerte für die Fehlerrate durch zufällige Hardware-Fehler können entweder aus Tabelle 2.5 entnommen werden oder aus anderen Quellen (z.B. Analyse von Felddaten) stammen.

| ASIL | D | C | B |
|---|---|---|---|
| Random hardware failure target values | $\leq 10^{-8}h^{-1}$ | $\leq 10^{-7}h^{-1}$ | $\leq 10^{-6}h^{-1}$ |

Tabelle 2.5: Mögliche Quelle für die Herleitung der Zielwerte für zufällige Hardware-Fehler nach ISO 26262-5 [38]

Auf Grund der Natur von transienten Fehlern, deren typischer Auftrittswahrscheinlichkeit und der im Verhältnis relativ kurzen Betriebsdauer eines Fahrzeugs, können Vereinfachungen angenommen werden. Es wird als sehr unwahrscheinlich betrachtet, dass mehrere unabhängige transiente Fehler in einem kurzen Zeitraum auftreten und dadurch auch Dual-Point-Fehler wirksam werden könnten. Im Regelfall werden die latenten Fehler durch ein Abschalten der Fahrzeugelektronik häufig genug gelöscht, damit keine Dual-Point-Fehler wirksam werden können. Dies ermöglicht eine klare Unterscheidung zwischen Fehlern mit beobachtbaren Auswirkungen und ohne sichtbare Auswirkungen.

Weitere bekannte Normen für sicherheitskritische Systeme nach Anwendungsbereich sind [39]:

- DO-178B: Software Considerations in Airborne Systems and Equipment Certification
- DO-254: Design Assurance Guidance for Airborne Electronic Hardware

- EN 50128: Bahnanwendungen — Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme — Sicherheitsrelevante elektronische Systeme für Signaltechnik
- IEC 60601: Medizinische elektrische Geräte — Allgemeine Festlegungen für die Sicherheit
- IEC 61511: Funktionale Sicherheit — Sicherheitstechnische Systeme für die Prozessindustrie
- IEC 61513: Kernkraftwerke — Leittechnik für Systeme mit sicherheitstechnischer Bedeutung — Allgemeine Systemanforderungen
- IEC 62061: Sicherheit von Maschinen — Funktionale Sicherheit sicherheitsbezogener elektrischer, elektronischer und programmierbarer elektronischer Steuerungssysteme

## 2.5 Zusammenfassung

In diesem Kapitel wurde eine Übersicht über die grundsätzlichen Ansätze zum Schutz vor den Auswirkungen transienter Fehler gezeigt. Neben diesen Maßnahmen können transiente Fehler auch durch ohnehin in Schaltungen vorhandene Fehlermaskierungseffekte verhindert werden. Eine Übersicht über bekannte Methoden zur Quantifizierung von Fehlermaskierungseffekten und deren Arbeitsweise wurde im folgenden Abschnitt gegeben. Die vorgestellten Verfahren haben individuelle Stärken und Schwächen und können daher nicht für alle Typen von Schaltungen verwendet werden. Im Hinblick auf die notwendige Qualifizierung der Fehlerraten bei sicherheitskritischen Systemen wäre ein leistungsfähigeres Verfahren wünschenswert.

# Kapitel 3

# Spezielle Fehlerquellen in FPGAs

Der Absatz von Field Programmable Gate Arrays FPGAs ist in den letzten Jahren stetig angestiegen, nicht nur wegen der häufigen Verwendung in Produkten der Unterhaltungsindustrie. Auch in der Luft- und Raumfahrt werden die flexiblen Bausteine beim Einsatz in Systemen mit relativ kleiner Stückzahl wegen ihrer geringen Kosten geschätzt. Hinzu kommt, das FPGAs durch die Programmierung für einen speziellen Anwendungsfall sehr hohe Leistungen bei der Datenverarbeitung ermöglichen. In vielen Produkten aus den Video- und Audio-Bereich werden deshalb FPGAs eingesetzt. Der Trend zur Flexibilisierung und Individualisierung setzt nun auch in der Automobilbranche ein. Deshalb finden FPGAs jetzt auch vermehrt in Automobilen ihre Anwendung. In modernen Fahrzeugen werden die meisten Innovationen durch neue elektronische Systeme ermöglicht. Dies sind nicht nur Unterhaltungs- und Komfortsysteme (Navigationssystem, Video-Displays), sondern auch Systeme, welche die Fahrzeugsicherheit betreffen. Verschiedene Fahrassistenzsysteme wie Abstandsregelung, Spurhalteassistent, Kollisionswarner und automatisches Einparksystem erfordern die Auswertung der Fahrzeugumgebung durch eine Vielzahl von Sensoren. Gerade bei bildgebenden Sensoren ist die Rechenleistung von FPGAs fast unerlässlich zur Realisierung der gewünschten Funktionalität. Da viele dieser Fahrassistenz-Systeme auch direkt Einfluss auf das fahrende Fahrzeug nehmen können, fallen sie prinzipiell auch in die Klasse der sicherheitskritischen Systeme. Aus diesem Grund ist es unerlässlich die Ausfallmechanismen von FPGAs genauer zu untersuchen.

## 3.1 Aufbau und Verwendung von FPGAs

Für das Verständnis von FPGA-typischen Ausfallmechanismen ist es grundsätzlich notwendig auch die prinzipielle Struktur eines FPGAs zu verstehen. Wie schon der Name FPGA vermuten lässt, sind FPGAs weitestgehend regelmäßig aufgebaut. Neben den Logikfunktionen, die zeilen- und spaltenweise auf dem Chip angeordnet sind, befinden sich eine Vielzahl an konfigurierbaren Verbindungsstrukturen in jedem FPGA. Der Anschluss zur Außenwelt des Bausteins wird durch ebenfalls konfigurierbare I/O-Blöcke hergestellt. In Abbildung 3.1 ist die Anordnung der vorhandenen Logikfunktionen und Verbindungsblöcke eines

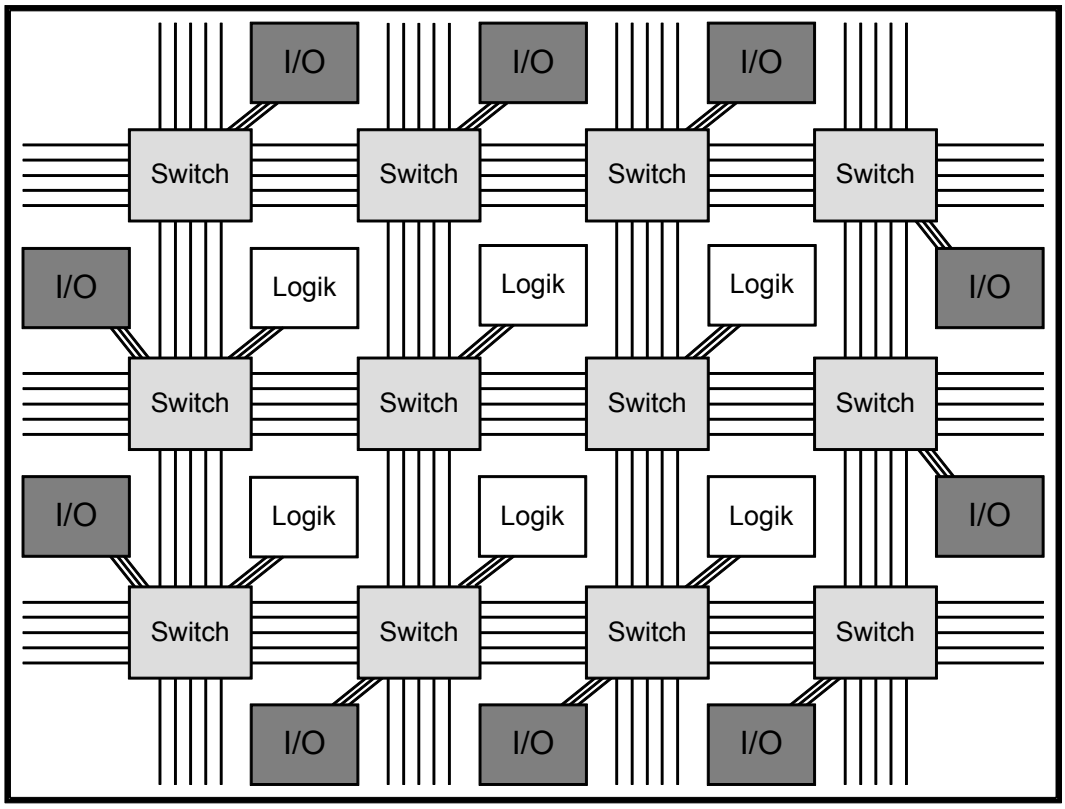



Abbildung 3.1: Prinzipieller Aufbau eines FPGAs

typischen FPGAs dargestellt.

### 3.1.1 Logikressourcen

Das wesentliche Element zur Realisierung einer beliebigen Logikfunktion stellen die sog. Look-Up-Tables LUT dar, welche wie eine Wertetabelle verwendet werden. Eine LUT besteht aus einem kleinen, adressierbaren Speicher mit einem ein Bit breiten Ausgang. Die Adresseingänge des Speichers werden stellvertretend als Eingangssignale der Logikfunktion verwendet, am Speicherausgang wird der zu den Eingangssignalen passende Logikwert ausgegeben. Der Speicher selbst kann während des Betriebs auf reguläre Weise nicht beschrieben werden, er arbeitet also als Read-Only-Memory ROM. Beim Konfigurationsvorgang werden die Speicherzellen passend zur gewünschten Logikfunktion initialisiert. Da der Speicher aber mit beliebigen Werten gefüllt werden kann besteht kein Unterschied im Aufwand zur Realisierung zwischen einfachen (AND, OR, NOT) und komplexen Logikfunktionen. Die verfügbare Speichermenge und die Anzahl der möglichen Eingangssignale hängen direkt voneinander ab. Die Anzahl der Eingänge $n$ bestimmt die notwendige Speichermenge in Bit durch $2^n$ mögliche

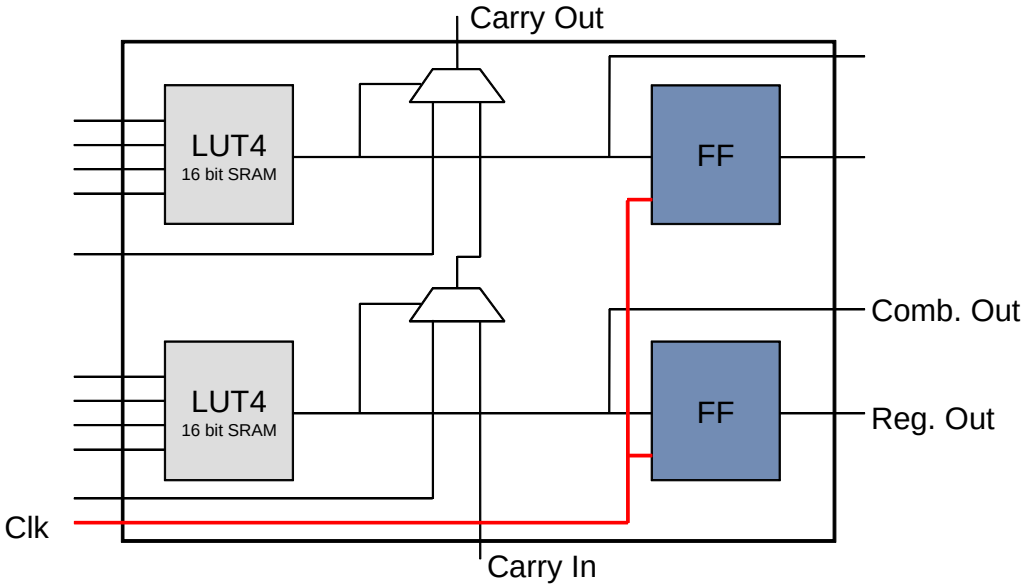



Abbildung 3.2: Prinzipieller Aufbau einer FPGA-Logikzelle

Eingangssignalkombinationen. Typische LUTs bieten derzeit vier, fünf oder sechs Eingangssignale, d.h. die notwendigen Speicherblöcke haben die Größe 16 bit, 32 bit oder 64 bit. Funktionen, die mehr Eingangssignale benötigen als die verfügbaren LUTs bieten, können einfach durch Kaskadierung mehrerer LUTs hergestellt werden.

Da sich aus einfachen kombinatorischen Logikfunktionen keine komplexen digitalen Schaltungen aufbauen lassen, sind zusätzliche sequentielle Elemente in FPGAs notwendig. Üblicherweise wird in FPGAs jeweils ein D-FlipFlop zusammen mit einer LUT zur Verfügung gestellt. Die integrierten FlipFlops bieten meistens synchrone und asynchrone Set- und Reset-Eingänge, Takteingang mit Enable-Signal und wählbarer Flankensensitivität. Damit sind praktisch alle möglichen Kombinationen durch die konfigurierbaren FlipFlops abgedeckt.

Ein letztes Standard-Element von FPGA-Logikzellen zielt speziell auf arithmetische Anwendungen ab. In Verbindung mit einer LUT werden die notwendigen Komponenten zur Realisierung einer schnellen Carry-Chain für Addierer- und Subtrahierer-Schaltungen in den Logikzellen implementiert. Würden Addierer-Schaltungen nur aus LUTs erstellt, so müssten für jedes Bit Eingangsbreite des Datenwortes zwei LUTs verwendet werden, da pro Eingangsbit-Paar auch zwei Ausgänge (Carry- und Summenausgang) nötig sind. Zudem wäre das Worst-Case-Delay des gesamten Addierers sehr hoch, was die maximale Taktfrequenz und damit auch die Leistungsfähigkeit von FPGA-Schaltungen stark begrenzen würde. Eine vereinfachte, aber dennoch typische Logikzelle eines Low-Cost-FPGAs ist in Abbildung 3.2 abgebildet.

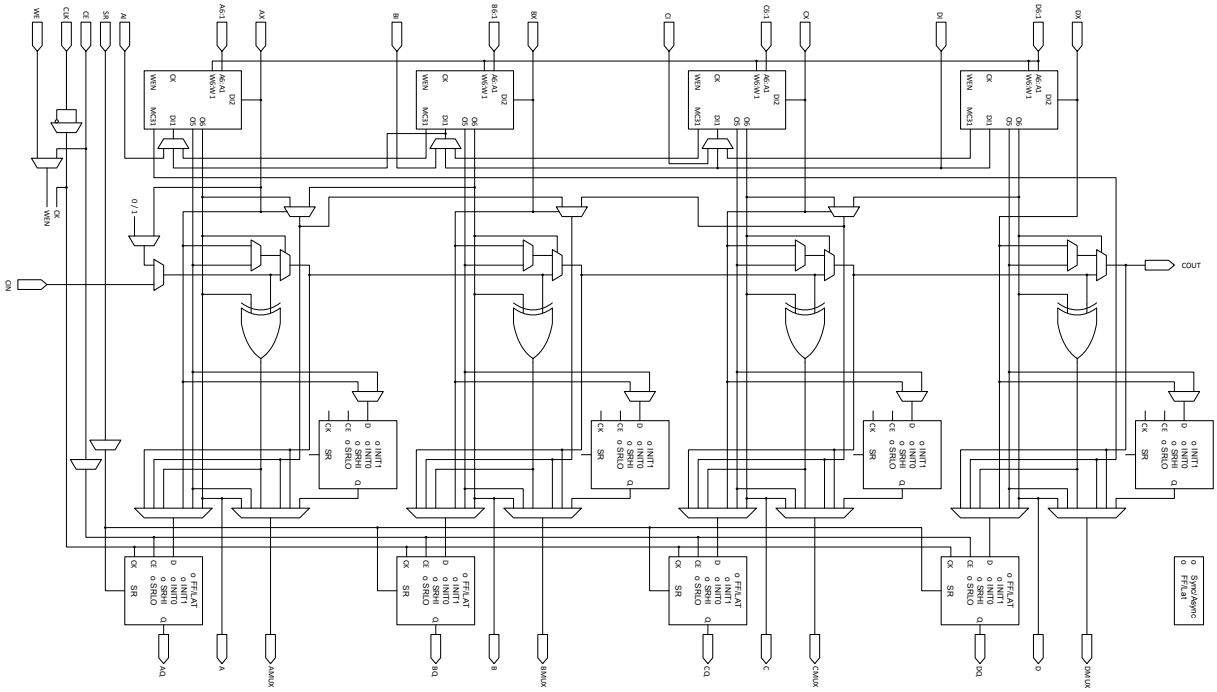



Abbildung 3.3: Logikblock (SLICEM) eines Xilinx Virtex7 FPGA [40]

Natürlich sind nicht alle FPGA-Zellen identisch, jeder Hersteller gestaltet die Zellen nach seinen eigenen Vorstellungen und optimiert unter Umständen sogar für spezielle Anwendungsdomänen. Beispielhaft für die Komplexität eines High-End-FPGAs sind in Abbildung 3.3 alle Bestandteile einer Logikzelle eines Xilinx Virtex7 FPGAs dargestellt.

Neben den bereits benannten Bestandteilen der FPGA-Logik sind herstellerspezifisch weitere konfigurierbare Funktionen Bestandteil der FPGAs:

- Taktstabilisierung und Frequenzmultiplikation durch Phase-Locked-Loops PLLs
- Speicherblöcke in der Größenordnung einiger kbit, sogenannte BlockRAMs
- Digital Signal Processing DSP Blöcke mit integriertem Multiplizierer

### 3.1.2 Verbindungsressourcen

Die zweite wesentliche Komponente eines FPGAs sind die programmierbaren Verbindungsstrukturen, die Logikzellen untereinander und mit den I/O-Ressourcen verbinden. Dieses Netz aus schaltbaren Verbindungen in horizontaler und vertikaler Richtung soll möglichst kurze und damit auch schnelle Verbindungen ermöglichen. Auch gilt es aus Gründen der Verlustleistung eines FPGAs eine Leitungsstruktur mit minimierter Kapazität zu finden.

Abhängig von der Schaltung, die in dem FPGA dargestellt werden soll, werden unterschiedlich viele der Verbindungsressourcen genutzt. Es besteht dabei

die Gefahr, dass die gewünschte Schaltung mit der damit verbundenen Platzierung der Logikelemente im FPGA nicht darstellbar ist, weil zu wenig Verbindungsressourcen verfügbar sind. Beim Entwurf des FPGAs muss der Hersteller daher eine ausreichende Anzahl an Leitungen einplanen, damit möglichst jede Schaltung darstellbar ist und es nicht zu einem lokal begrenzten Mangel kommt. Durch eine veränderte Platzierung der notwendigen Logikkomponenten können diese Engpässe aber häufig verhindert werden. Da die initiale Platzierung der Elemente durch das Synthesewerkzeug wohl aber als optimale Platzierung angesehen werden muss, stellt zwangsweise jede Abweichung von der ursprünglichen Platzierung eine Verschlechterung der Schaltungseigenschaften dar. Insbesondere die Verzögerungszeiten der Leitungen sind durch eine veränderte Platzierung der Logikelemente betroffen. Eine sehr große Anzahl an Verbindungsressourcen kann sich aber auch negativ auswirken. So kann die Fläche des FPGA-Chips unnötig groß werden, wenn zu viele Verbindungsressourcen eingeplant werden, die in den meisten Fällen ungenutzt bleiben. Die Chipgröße wiederum beeinflusst die Kosten zur Herstellung des Chips und verschlechtert auch die Signallaufzeiten, da die Leitungslängen größer werden. Die Hersteller von FPGAs müssen darum bei der Entwicklung ihrer Bausteine sehr genau abwägen, wie viele programmierbare Verbindungsressourcen sie zur Verfügung stellen wollen.

Für die Effizienz der Verbindungsressourcen spielt die genaue Struktur der möglichen Verbindungen eine große Rolle. Keine Kompromisse hinsichtlich der möglichen Verbindungen stellt das vollvermaschte, bidirektionale Stern-Schema dar (Abbildung 3.4). Hier ist eine direkte Verbindung zwischen allen angeschlossenen Leitungsstücken und den Anschlüssen der Logikzelle möglich.

Dieses theoretisch optimale Schema bringt für eine praktische Implementierung aber zwei wesentliche Nachteile mit sich. Wichtigster Punkt ist die sehr hohe Anzahl der nötigen schaltbaren Verbindungen. Dies würde zu einem viel zu großen Flächenverbrauch für die benötigten Schalter (z.B. Transmission Gate) an sich und die notwendigen Konfigurations-Speicherzellen führen.

Die resultierende Anzahl der Verbindungen ist in Gleichung 3.1 gegeben. Da die Anzahl der Verbindungen quadratisch mit der Anzahl der Anschlusspunkte ansteigt, kann dieses Schema in der Praxis keine Anwendung finden. Ein weiterer Nachteil wird bei Betrachtung des globalen Zusammenspiels dieser Struktur ersichtlich. Dadurch, dass auf jedes Leitungsstück mehrere Treiber wirken könnten, müssen alle Treiber mit Tri-State-Funktionalität ausgestattet sein. Dies erhöht den Aufwand zur Realisierung des Verbindungsschemas nochmals.

Allein aus diesen Betrachtungen kann die theoretisch optimale Verbindungsstruktur für eine praktische Anwendung ausgeschlossen werden. In einer realistischen Verbindungsstruktur ist nur ein Bruchteil aller denkbaren Verbindungen realisiert. Zudem wird die Problematik von flächig über dem Chip verteilten Tri-State-Treibern für einzelne Leitungsstücke vermieden, indem nur unidirektionale schaltbare Verbindungen implementiert werden. Dies bedeutet, das pro vorhandenen Leitungsstück genau ein Treiber existiert. Eine naheliegende Implemen-

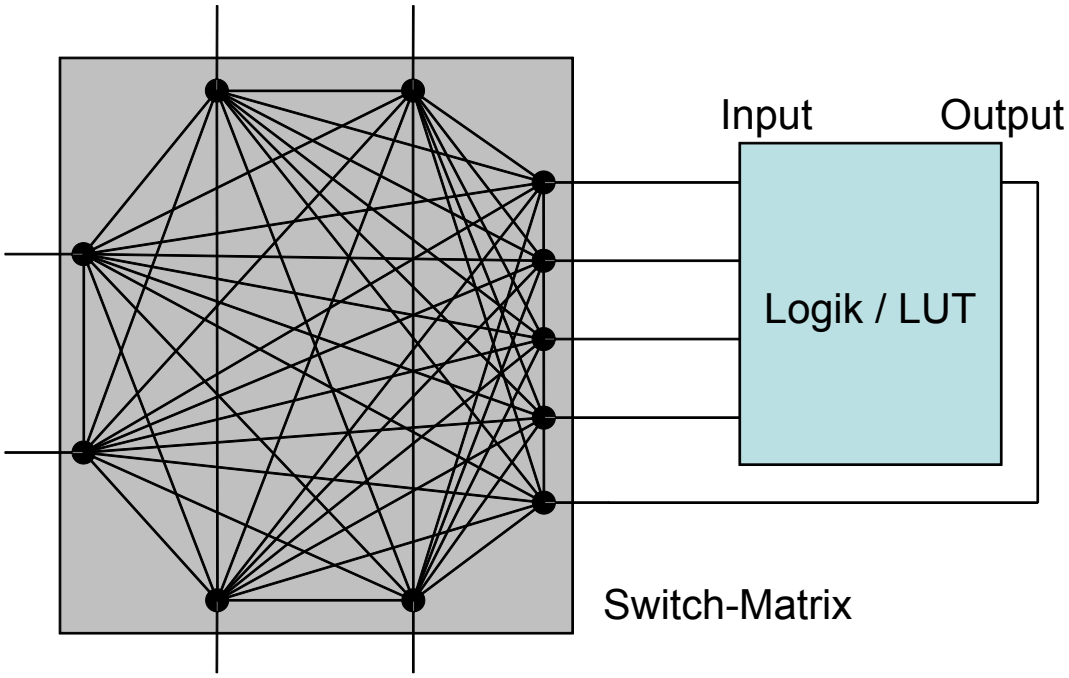



Abbildung 3.4: Theoretische, optimale Verbindungsstruktur für FPGAs

tierung verwendet daher Multiplexer zur Auswahl des Eingangssignals. Weitere Überlegungen zum Aufbau der Verbindungsstrukturen und der verwendeten Multiplexer sollen mit Daten aus realen FPGAs untermauert werden.

Als Beispiel wird ein Xilinx Virtex2Pro FPGA "xc2vp30" mit folgenden konfigurierbaren Ressourcen verwendet:

- 27392 LUTs mit 4 Eingängen
- 27392 FlipFlops
- 136 Multiplizierer für 18 x 18 bit
- 136 BlockRAMs mit 18 kbit Kapazität

Mittels eines Kommandozeilen-Werkzeugs der Entwicklungsumgebung von Xilinx kann eine Datei mit den Programmable Interconnect Points PIPs (eine schaltbare Verbindung) und deren gesamter Anzahl für jedes verfügbare FPGA erstellt werden. Für unser Beispiel lautet der Befehl demnach: "xdl -report xc2vp30 -pips". In unserem Beispiel-FPGA sind insgesamt 14.907.365 Pips bei 3424 Switch-Matrizen (27392 $LUT/8$) vorhanden bzw. ca. 4300 Pips pro Switch-Matrix implementiert. Bei insgesamt 432 Signalen, die an einer Switch-Matrix anliegen, davon 76 Signale mit Verbindung zur Logik, ergibt dies ein Verhältnis von ca. 10 Pips pro Leitungsstück (Abbildung 3.6 und Abbildung 3.7).

Aus dem Datenblatt des Beispiel-FPGAs kann die Größe der Datei, welche zur Konfiguration der FPGA-Funktion notwendig ist, direkt entnommen werden. Da die hierzu angegebene Dateigröße von 11.589.920 bit an sich schon kleiner ist als die Anzahl der verfügbaren Pips, kann ausgeschlossen werden, dass jedem PIP ein

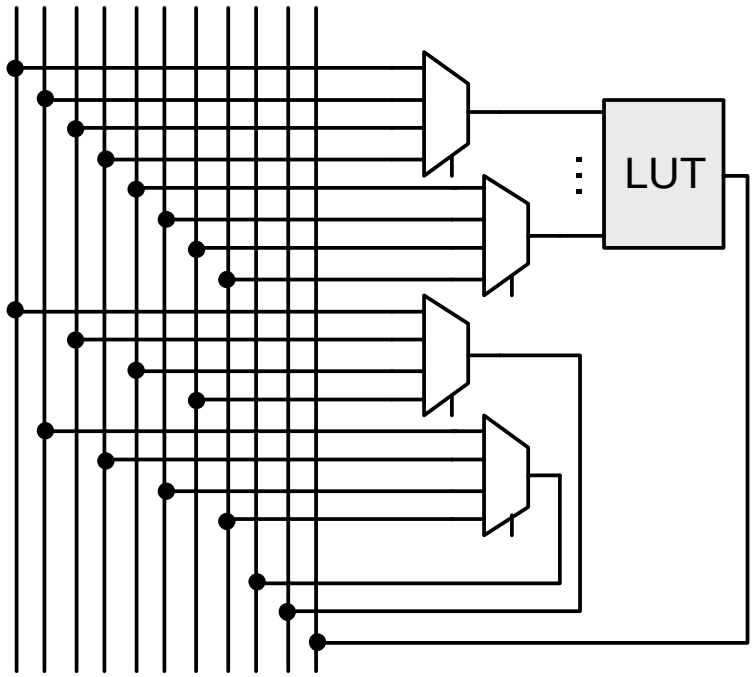



Abbildung 3.5: Realistische Verbindungsstruktur in FPGAs

$$N_{Connections} = \frac{n \cdot (n-1)}{2} \tag{3.1}$$

Anzahl der Verbindungen $N_{Connections}$ in Abhängigkeit von der Anzahl der Knoten $n$ eines vollvermaschten Graphen

Konfigurationsbit zugeordnet ist. Nach Berücksichtigung der BlockRAMs und der Logikzellen stehen für die Verbindungsstrukturen nur noch ca. 7,5 MBit Daten zur Verfügung, was ein Verhältnis Pips zu Konfigurationsbits von ca. 2:1 bedeutet. Auch deshalb müssen multiplexer-artige Strukturen in den Switch-Matrizen von FPGAs Verwendung finden, damit mit möglichst wenig Konfigurationsbits eine Verbindung von zwei Leitungsstücken konfiguriert werden kann. Die genauen Zusammenhänge von Konfigurationsbits und den damit konfigurierten Ressourcen sind Betriebsgeheimnis der FPGA-Hersteller.

### 3.1.3 Syntheseprozess von Schaltungen für FPGAs

Um ein FPGA verwenden zu können, muss die gewünschte Schaltung in Konfigurationsdaten, den sogenannten Bitstream, umgesetzt werden. Dies wird mit den herstellerspezifischen Entwicklungswerkzeugen durchgeführt. Der komplette Prozess besteht i.A. aus mehreren Schritten:

- Schaltungssynthese
- Übersetzung

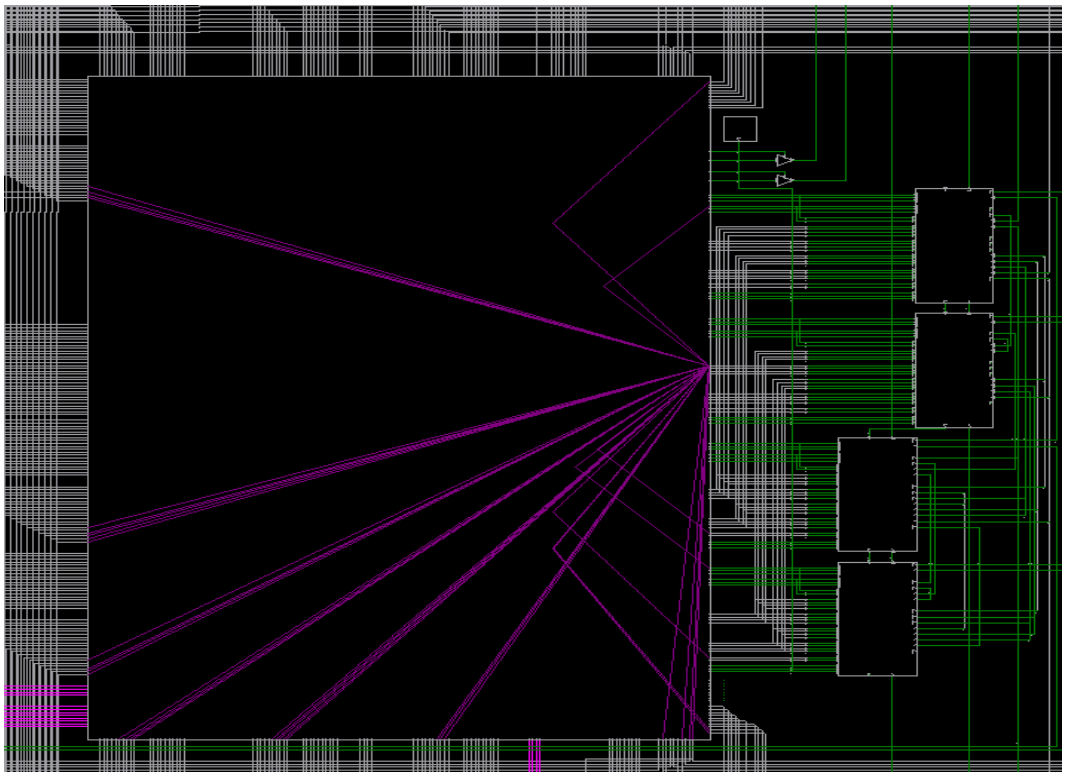


Abbildung 3.6: Xilinx FPGA Editor: Ansicht aller möglichen Verbindungen zu einem Leitungsstück hin (violette Linien)

- Platzierung
- Verdrahten

Im ersten Schritt, der Schaltungssynthese, werden die gegebenen Quelldateien automatisch verarbeitet und daraus eine Schaltung bzw. Netzliste erstellt. In dieser Netzliste werden technologieunabhängige Bauelemente ohne genaue Spezifikation verwendet, prinzipiell stehen dort alle digitalen Funktionen zur Realisierung einer Funktion zur Verfügung. Das Syntheseprogramm ist bei der Art der Quelldatei nicht etwa nur auf grafische Schaltplandateien beschränkt. Die gebräuchlichste Form der Beschreibung von Schaltungen, die in FPGAs realisiert werden sollen, ist mittlerweile eine Hardwarebeschreibungssprache wie VHDL oder Verilog. Ebenso üblich ist die Verwendung von Zustandsübergangsdiagrammen zur Beschreibung von Zustandsautomaten. Da das Ergebnis der Synthese eine Schaltung bestehend aus generischen Elementen ist, kann diese Aufgabe auch durch Nicht-FPGA-Hersteller-Software erledigt werden. Mehrere Firmen bieten Synthesewerkzeuge für FPGAs an, die zusätzlich zu den bisher genannten Eingangsdaten z.B. auch C- oder SystemC-Code in einer High-Level-Synthese verarbeiten können. Neben unterschiedlichen Eingangsformaten für die Synthese können auch unterschiedliche Vorgaben und Ziele berücksichtigt werden. Hinsichtlich der ma-

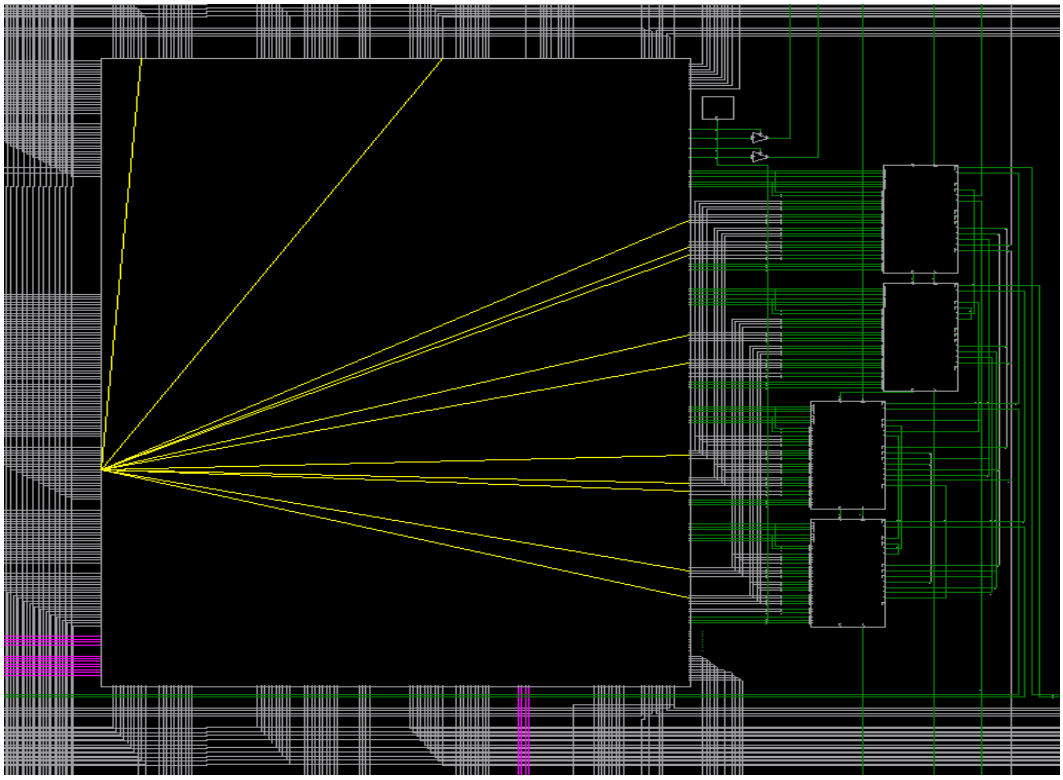


Abbildung 3.7: Xilinx FPGA Editor: Ansicht aller möglichen Verbindungen von einem Leitungsstück weg (gelbe Linien)

ximalen Taktfrequenz der Schaltung spielt die Logiktiefe der erstellten Schaltungen eine große Rolle. Häufig kann die Schaltung deshalb mit Ziel einer maximalen Taktfrequenz oder minimalem Ressourcenbedarf synthetisiert werden. Die Syntheseprogramme können aber auch die Logikfunktionen, Zustandskodierung, erwartete Verlustleistung oder das sequentielle Verhalten durch geeignete Maßnahmen automatisiert optimieren.

Nach der Erstellung einer Schaltung ausschließlich aus generischen Elementen wird die Schaltung in eine technologieabhängige Version übersetzt. Dabei werden die Eigenschaften des verwendeten FPGAs berücksichtigt, ja nach FPGA-Typ kann es also zu unterschiedlichen Ergebnissen kommen. Die vorhandenen FPGA-Logikelemente müssen dabei hinsichtlich ihrer Funktionalität eine Übermenge gegenüber den tatsächlich benötigten Funktionen bilden. Nicht benötigte Funktionalität wird dabei häufig durch Zuweisung von Konstanten an Eingänge deaktiviert. Als Beispiel sei hier eine LUT mit vier Eingängen genannt, die natürlich auch alle Logikfunktionen mit nur einem, zwei oder drei Eingängen darstellen kann.

Der dritte Schritt behandelt das Problem der physikalischen Platzierung im FPGA. Alle Elemente der zuvor erstellten technologieabhängigen Schaltung wer-

den nun auf die im FPGA vorhandenen Logikressourcen zugewiesen, wobei deren räumliche Position eine entscheidende Rolle spielt. Mehrstufige Logikfunktionen sollen natürlich möglichst eng aneinander platziert werden, um die Verzögerungszeit zwischen den Logikzellen möglichst gering zu halten. Die Vorgabe, welche Ein- und Ausgangspins verwendet werden sollen, beeinflusst das Platzierungsergebnis ebenfalls.

Im letzten Schritt wird die Verdrahtung der Logikzellen durchgeführt. Unter der Grundvoraussetzung, dass alle verwendeten Logikzellen richtig untereinander verbunden werden, wird versucht möglichst kurze und damit auch schnelle Verbindungen im FPGA zu verwenden. Da die Verdrahtungsressourcen natürlich auch nur begrenzt vorhanden sind, ist es häufig unvermeidlich nicht optimale Verbindungen zu benutzen. Bei wenigen Schaltungen kann es vorkommen, das diese aufgrund der Verdrahtungs-Anforderungen im FPGA nicht darstellbar sind. In diesen Fällen, aber auch zur Verbesserung der maximalen Taktfrequenz, kann die Platzierung und Verdrahtung unter Verwendung von Zufallsgrößen (Seed) erneut durchgeführt werden. Eine veränderte Platzierung der notwendigen Komponenten erlaubt dann möglicherweise eine vollständige Verdrahtung der Schaltung. Je höher die Anzahl der benötigten Logikkomponenten im Vergleich zur Größe des verwendeten FPGAs, desto komplexer und zeitaufwändiger wird der Platzierungs- und Verdrahtungsprozess.

Um die fertig platzierte und verdrahtete Schaltung mit einem FPGA nun verwenden zu können, müssen diese Daten noch in den Bitstream umgewandelt werden. Die Größe der Bitstream-Datei richtet sich im Allgemeinen dabei nach der Größe des verwendeten FPGAs und nicht nach der Anzahl der verwendeten Ressourcen, da immer das komplette FPGA konfiguriert werden muss. Verschiedene Ansätze zur Komprimierung von Bitstreams werden zwar von den Herstellern von FPGAs angeboten, haben sich in der Praxis aber nicht durchgesetzt. Derzeit verfügbare FPGAs verwenden zur Konfiguration Dateien bis zu ca. 450 MBit Größe (z.B. Xilinx Virtex7 XC7V2000T [41]).

### 3.1.4 FPGA-Technologie

Bei den bisherigen Ausführungen zur Funktion von FPGAs wurde noch nicht auf die technologische Realisierung dieser Bausteine eingegangen. Da für die konfigurierbaren Funktionen sowohl für die Logik als auch für die Verbindungsstruktur große Mengen an Speicher benötigt werden, sind diese Speicher ein wesentlicher Anteil des FPGAs. Die gebräuchlichsten Speicherkonzepte derzeit sind:

- Dynamic Random Access Memory DRAM
- Static Random Access Memory SRAM
- Single Level Cell SLC und Multi Level Cell MLC Flash Speicher

Da zur Verwendung in FPGAs als Konfigurationsspeicher alle Speicher kontinuierlich gelesen werden, kann DRAM wegen des erforderlichen Leseverstärkers nicht verwendet werden. Sollte zu jeder DRAM-Zelle ein Leseverstärker imple-

mentiert werden, so würde dies nur einen Mehraufwand gegenüber einer einfachen SRAM-Zelle bedeuten. SRAM bietet den Vorteil ohne weitere Maßnahmen als Konfigurationsspeicher verwendbar zu sein. Zudem kann SRAM auf Grund seiner regelmäßigen Struktur sehr gut und dicht integriert werden. Lediglich die relativ hohe Verlustleistung auf Grund der Leckströme der Zellen muss als negative Eigenschaft hervorgehoben werden. Flash-Speicherzellen können ebenfalls sehr gut integriert werden, jedoch ist die Verbindung mit einem High-Performance-Prozess teurer als ein Fertigungsprozess, der nur auf eine Anwendung hin optimiert ist. MLC Flash ist hier schlecht geeignet, da zur Verwendung der Zellen eine Fehlerkorrektur notwendig ist, Latenzen beim Lesen und Schreiben vorhanden sind und der Betriebsstrom höher ist.

Grundsätzlich geeignet für die Verwendung in FPGAs erscheinen also SRAM und SLC Flash. Die Mehrzahl der FPGAs wird mit SRAM-Technik realisiert, da sich hier die höchste Integrationsdichte und Arbeitsgeschwindigkeit erreichen lässt. Wesentlicher Nachteil von SRAM-basierten FPGAs ist die Flüchtigkeit der gespeicherten Information, d.h. ein SRAM-FPGA muss nach jedem PowerUp-Vorgang neu konfiguriert werden. Dies macht einen weiteren Speicher für die Konfigurationsdaten an anderer Stelle unabdingbar. Diese Problematik wird durch Verwendung von Flash-Speicher umgangen. Nur wenige Hersteller bieten Flash-basierte FPGAs an, die aber keine so hohen Taktfrequenzen und so große Logikkapazität ermöglichen wie SRAM-FPGAs. Eine dritte Gruppe von FPGAs verwendet sowohl SRAM und Flash. Dadurch steht zur Konfiguration des FPGAs ein nicht-flüchtiger Speicher zur Verfügung, diese Daten werden beim PowerUp in die SRAM-Zellen kopiert, welche den eigentlichen Konfigurationsspeicher wie bei normalen SRAM-basierten FPGAs darstellen. Die enge Kopplung von Bitstream-Speicher und Konfigurations-Speicher bringt einige Vorteile bei der Anwendung mit sich, verursacht aber auch höhere Kosten.

### 3.1.5 Konfiguration

Die Konfiguration stellt den letzten Schritt vor der eigentlichen Anwendung des FPGAs dar. Je nachdem, ob es sich bei dem verwendeten FPGA um ein Flash- oder SRAM-basiertes FPGA handelt, muss die Konfiguration nur einmal oder bei jedem PowerUp durchgeführt werden. Bei der Konfiguration werden die Bitstream-Daten in das FPGA übertragen um so die gewünschte Funktionalität herzustellen. Für die Konfiguration bieten die Hersteller meist mehrere Konfigurationsinterfaces zur Wahl an:

- IEEE 1149 JTAG [42]
- Serial Peripheral Interface SPI [43]
- Paralleles Interface mit Busbreiten von 8, 16 oder 32 bit
- Herstellerspezifische Interfaces

Durch diese Auswahlmöglichkeiten kann meist ein Konfigurationsinterface ausgewählt werden, welches am Besten zu den weiteren Anforderungen des gesamten

Systems passt. Hierbei müssen primär die zwei Faktoren benötigte Pinanzahl und Konfigurationszeit berücksichtigt werden. Abhängig von der Systemarchitektur können die Konfigurationspins am FPGA nicht für die konfigurierbare Logik verwendet werden, ein paralleles Konfigurationsinterface schränkt daher die Anzahl der verbleibenden I/O-Pins des Systems ein. Weiterhin ist die Arbeitsfrequenz des FPGAs während der Konfiguration stark eingeschränkt, was eine begrenzte Übertragungsbandbreite pro Konfigurationspin bedeutet. Für Flash-basierte FPGAs, die in der Regel nur einmal vom Systemhersteller programmiert werden, spielt die Konfigurationszeit eine untergeordnete Rolle. Da SRAM-FPGAs aber bei jedem Systemstart konfiguriert werden müssen, kann eine Konfigurationszeit von z.B. mehreren Sekunden in Konflikt mit den Anwendungsanforderungen stehen. Selbst Konfigurationszeiten von wenigen Millisekunden sind nicht in jeder Anwendung hinnehmbar. Durch die direkte Einbindung eines nicht-flüchtigen Speichers sind Flash-FPGAs nahezu instantan betriebsbereit. SRAM-FPGAs mit integriertem Flash-Speicher können breitere und höher getaktete Konfigurationsinterfaces benutzen, deren Konfigurationszeit ist deshalb i.A. deutlich geringer als bei reinen SRAM-FPGAs.

### 3.1.6 Sicherheitsziele bei der Konfigurationsdatenübertragung

Die Konfiguration an sich stellt eine einfache Datenübertragung von einem externen Speicher in das FPGA dar. Da diese Daten die Funktionsfähigkeit des FPGAs beeinflussen und hinter den meisten FPGA-Designs auch wirtschaftliche Interessen stehen, müssen mehrere Sicherheitsziele bei der Konfiguration beachtet werden:

- Korrektheit
- Datenintegrität
- Vertraulichkeit

Bei der Übertragung des Bitstreams muss primär sicher gestellt werden, dass das FPGA fehlerfrei konfiguriert wurde. Diese Fehler können sowohl bei der Übertragung z.B. durch EM-Einstrahlung entstehen als auch im externen Speicher des Bitstreams auftreten. Für eine Fehlererkennung wird in der Regel Redundanz zum Bitstream hinzugefügt. Ein gebräuchliches Verfahren ist das Anfügen von CRC-Prüfsummen an den eigentlichen Bitstream, so dass ein im FPGA fehlerhaft empfangener Bitstream mit höchster Wahrscheinlichkeit erkannt wird. Die Verwendung von Prüfsummen zur Fehlererkennung in FPGA-Bitstreams ist Standard bei allen FPGA-Herstellern. Im Gegensatz dazu ist kein Hersteller bekannt, der eine Fehlerkorrektur für Bitstream-Daten verwendet.

Weiterer Sicherheitsaspekt bei FPGA-Bitstreams ist die Datenintegrität. Damit ein FPGA erfolgreich konfiguriert werden kann, müssen unter den bisherigen Anforderungen lediglich Bitstream und Prüfsumme des Bitstreams zueinander

passen. Prinzipiell ist damit jeder Bitstream in das FPGA konfigurierbar. Es kann aber nicht ausgeschlossen werden, dass der ursprüngliche Bitstream durch einen anderen Bitstream ersetzt wurde. Damit ist auch nicht ausgeschlossen, dass die Funktion des FPGAs unbemerkt manipuliert wurde. Für den Einsatz in sicherheitskritischen Systemen stellt diese unbefugte Manipulation ein Problem dar, weil dadurch unkalkulierbare Risiken sowohl für den Nutzer als auch den Hersteller entstehen.

Der letzte Sicherheitsaspekt ist bei der Wahrung der wirtschaftlichen Interessen der Hersteller relevant. Bei der Konfiguration können die Bitstream-Daten einfach an den Konfigurationspins abgegriffen werden, damit können Kopien des Bitstreams einfach erstellt werden. Um dies zu verhindern, kann eine Verschlüsselung des Bitstreams vorgenommen werden. Gängige, in FPGAs verwendete symmetrische Verschlüsselungsalgorithmen sind:

- Data Encryption Standard DES
- Triple Data Encryption Standard 3DES
- Advanced Encryption Standard AES

In praktisch allen FPGAs neuerer Generation steht AES-Verschlüsselung zur Verfügung. Allerdings unterscheiden sich die Ansätze zur Speicherung des Schlüssels deutlich voneinander. Es können verschiedene Technologien zum Speichern der Schlüssel verwendet werden:

- SRAM-Zellen im FPGA, die durch eine externe Pufferbatterie dauerhaft versorgt werden
- Hart codierter Schlüssel, z.B. Seriennummer des Chips oder ähnliches
- OTP-Speicher auf Fuse/Antifuse-Basis
- Flash-Speicher

Damit stehen für die meisten Anwendungen ausreichend sichere Verschlüsselungsverfahren zur Verfügung. In der praktischen Anwendung stellt die Verwendung solcher Schlüssel einen Mehraufwand in der Produktion dar. Sowohl die Verschlüsselung des Bitstreams mit evtl. verschiedenen Schlüsseln (z.B. bei Verwendung der Seriennummer) als auch das Einprogrammieren des Schlüssels in das FPGA müssen separat und in sicherer Umgebung durchgeführt werden.

Nachdem die verschlüsselten Daten in das FPGA übertragen wurden, werden sie entschlüsselt und dem internen Konfigurationsinterface zugeführt, wo sie dann in unverschlüsselter Form im Konfigurationsspeicher liegen. Moderne FPGAs bieten häufig die Möglichkeit, die Konfigurationsdaten durch interne Interfaces auszulesen, um z.B. auch nachträglich eine Fehlerüberprüfung durchführen zu können. Bei FPGAs mit der Fähigkeit zur dynamischen Rekonfiguration besteht auch die Möglichkeit den Konfigurationsspeicher teilweise neu zu beschreiben. Durch diese Fähigkeiten entstehen für einen potentiellen Angreifer viele Schwachstellen, mit deren Hilfe er in das System eindringen und Konfigurationsdaten auslesen oder verändern kann. Zur Einhaltung der Vertraulichkeit und Integrität müssen daher durch die FPGA-Hersteller Mechanismen auch gegen solche Angriffe implementiert werden.

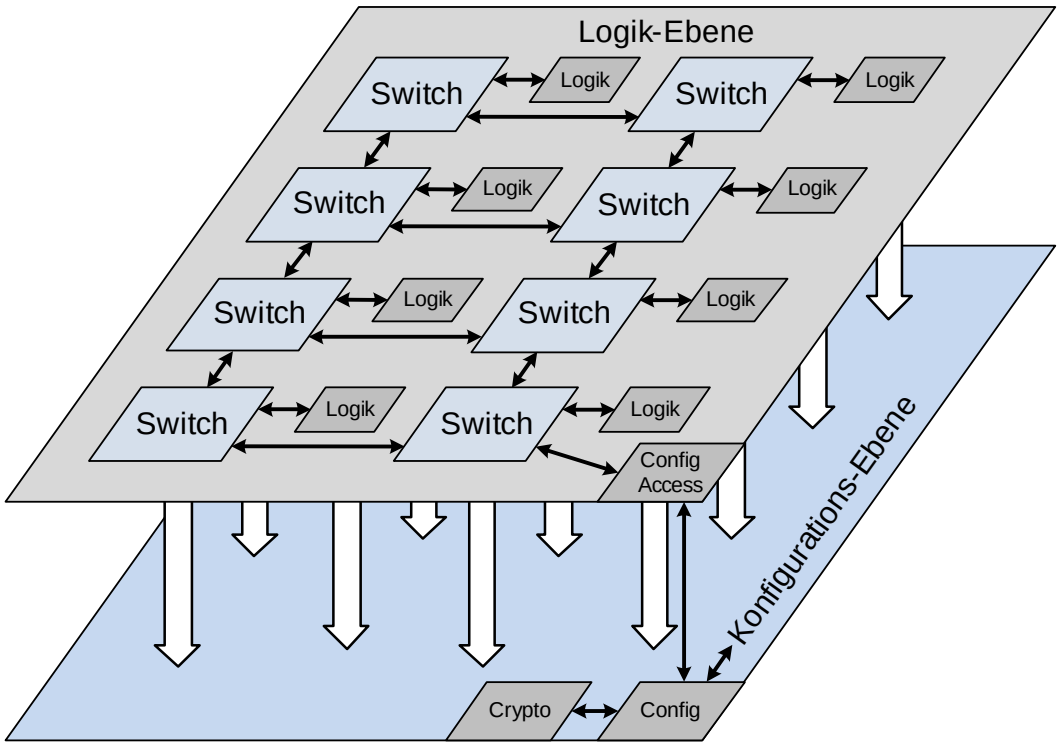



Abbildung 3.8: Zwei-Ebenen-Struktur eines FPGAs

Eine Bewertung welcher Typ von Speicher für kryptographische Schlüssel am sichersten gegenüber einem physikalischen Angriff ist, soll an dieser Stelle nicht vorgenommen werden.

## 3.2 Fehlerquellen im programmierbaren Teil von FPGAs

Bei allen Betrachtungen der Fehlermechanismen in FPGAs müssen die Vorgänge in der konfigurierbaren Funktionalität separat vom restlichen Teil betrachtet werden.

### 3.2.1 Programmierbare Logikzellen

Die programmierbaren Logikzellen bestehen wesentlichen aus einer LUT, einem nachfolgenden FlipFlop und wenigen anderen fest verdrahteten Logikfunktionen für schnelle Multiplexer und arithmetische Schaltungen. Die fest verdrahteten Logikfunktionen sind in der Regel nur durch Routing-Ressourcen beeinflussbar, deshalb sollen die Fehlerauswirkungen darauf analog zu Fehlern in den Switch-Matrizen im nächsten Unterabschnitt behandelt werden.

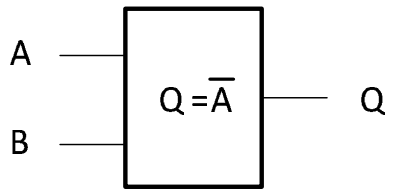
Da eine LUT durch einen kleinen Speicher implementiert wird, besteht prinzipiell die Gefahr, dass dieser Speicher durch einen SEU verfälscht wird. Weil der Speicher auf sehr kleinem Raum implementiert ist, kann es bei einem SEU deshalb sowohl zu Einfach- als auch zu Mehrfachfehlern kommen. Der veränderte Speicherinhalt der LUT äußert sich in einer abweichenden Logikfunktion, sofern die zu den betroffenen Speicherzellen korrespondierenden Eingangssignale in der Anwendung tatsächlich auftreten. Die Schwere der Auswirkungen des Fehlers kann nicht pauschal bestimmt werden, jedoch kann ein Fehler in einer LUT prinzipiell keine weiteren Schäden wie z.B. einen Treiberkurzschluss verursachen. Da solche Fehler aber von permanenter Natur sind, bzw. zumindest bis zur nächsten Konfiguration des FPGAs erhalten bleiben, muss mit einer beobachtbaren Auswirkung des Fehlers gerechnet werden.

Für ein FPGA-Design werden aber nicht immer alle Ressourcen des verwendeten Bausteins genutzt. SEUs in ungenutzten LUTs haben wegen des nicht verwendeten LUT-Ausgangssignals keine Auswirkungen. Eine weitere Unterscheidung muss aber für teilweise verwendete LUTs gemacht werden. Ein Teil des vorhandenen LUT-Speichers wird praktisch nicht verwendet, wenn die in der LUT dargestellte Logikfunktion weniger als die maximale Anzahl der verfügbaren Eingangssignale benötigt. Dementsprechend sind die Speicherzellen, die wegen der nicht benutzten Eingangssignale nicht angesteuert werden können, unkritisch für das Ausgangsverhalten der LUT. Da diese überzähligen Eingangssignale aber vorhanden sind, muss auch eine definierte Belegung des Signals vorhanden sein. Hierfür können verschiedene Ansätze gewählt werden:

- Belegung mit einer Konstante
- Auffüllen der überzähligen Eingangssignale mit anderen Signalen, z.B. einem bereits an dieser LUT verwendeten Signal
- undefiniertes/offenes Signal

Für einen definierten Betrieb der LUT sind stabile Eingangssignale notwendig, deswegen können offene Eingangssignale an ungenutzten LUT-Eingängen ausgeschlossen werden. Die Mehrfachverwendung von bereits an der LUT verwendeten Signalen ist prinzipiell nicht ausgeschlossen. Allerdings müssen dann die Initialisierungswerte der LUT angepasst werden. Negativer Aspekt bei einem solchen Vorgehen ist aber die erhöhte dynamische Verlustleistung durch die größere gesamte geschaltete Eingangskapazität der LUT sowie die daraus resultierende größere Verzögerungszeit des Eingangssignals. Aus diesen Gründen sollten gleiche Eingangssignale nicht mehrfach an einer LUT verwendet werden.

Die Verwendung von konstanten Signalen zum Auffüllen überzähliger LUT-Eingänge stellt hier das beste Vorgehen dar. Durch die konstanten Eingangssignale entsteht keine zusätzliche dynamische Verlustleistung und führt auch zu keiner unnötig erhöhten Lastkapazität. Abhängig von der mit konstanten Signalen belegten LUT-Eingänge wird die Menge der tatsächlich möglichen Eingangssignalkombinationen eingeschränkt. Die nicht ansprechbaren Speicherzellen der LUT können dann auch keine beobachtbaren Fehler verursachen. Zur Erhöhung




| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Abbildung 3.9: LUT-Initialisierung, wenn nicht alle Eingangssignale benötigt werden

der Fehlertoleranz sollte trotzdem der ungenutzte Teil der LUT mit "gespiegelten" Konfigurationsdaten programmiert werden. Selbst dann, wenn das Eingangssignal keine Konstante mehr wäre (z.B. bei einem Fehler im Routing), wäre der Ausgang der LUT unabhängig von diesem Eingangssignal.

Zweiter Typ Speicher in programmierbaren Logikzellen sind die integrierten FlipFlops zu jeder LUT. In den meisten Schaltungen übersteigt die Anzahl der benötigten LUTs die Zahl der notwendigen FlipFlops. Häufig werden bei komplexen Logikfunktionen mehrere LUTs miteinander verschalten, um Logikfunktionen mit einer höheren Anzahl an Eingängen darstellbar zu machen. Für jedes Ausgangssignal in einer synchronen Schaltung wird dem zufolge ein FlipFlop und ein oder mehrere LUTs benötigt. Obwohl die Anzahl der notwendigen FlipFlops meist geringer ist als die Anzahl der LUTs, sind die FlipFlops aus sicherheitskritischer Sicht anders zu bewerten. Der Speicherinhalt der LUT-SRAM-Zellen bleibt während der Nutzung des FPGAs konstant. Der Inhalt der FlipFlops ist Teil des Schaltungszustandes und daher abhängig von der Verwendung des FPGAs. Abhängig vom Anwendungsfall der Schaltung können sich unterschiedlich viele Fehler in FlipFlops bis zu den beobachtbaren Ausgängen auswirken.

### 3.2.2 Programmierbarer Interconnect

Der programmierbare Interconnect ist ein Kernstück jedes FPGAs. Wie bereits zuvor vorgestellt, verwenden moderne FPGAs eine Architektur, bei der jedes Leitungsstück durch einen Multiplexer genau einen Treiber besitzt. Dieser Multiplexer kann ein Signal aus verschiedenen Eingangssignalen auswählen und dieses wiederum an ein ausgangsseitig angeschlossenes Leitungsstück weiterleiten. Jedes Leitungsstück wiederum ist Eingangssignal für mehrere Multiplexer und auch Logikzellen.

Abhängig von der Anzahl der Signale, aus denen der Multiplexer auswählen kann, sind unterschiedlich viele Konfigurationsbits notwendig. Die Anzahl der notwendigen Konfigurationsbits liegt dabei zwischen zwei praktisch bestimmbaren Grenzen. Die Untergrenze wird durch die Anzahl der mit $n$ Bits darstellbaren

Kombinationen $m = 2^n$ gemäß einem binären Zahlensystem gebildet, die Obergrenze ergibt sich bei einer One-Hot-Codierung, d.h. die exklusive Ansteuerung eines Schalters von einem einzelnen Konfigurationsbit. Darüber hinaus würde die Möglichkeit bestehen, durch zusätzliche Redundanz die Anzahl der Konfigurationsbits für einen Multiplexer weiter nach oben zu treiben, was jedoch auf Grund der damit verbundenen Kosten nur äußerst unwahrscheinlich in der Praxis zum Einsatz kommen würde. Bei allen Codierungsschemata, mit Ausnahme der Untergrenze, ist Redundanz vorhanden.

Ein Fehler in den Konfigurationsbits der Verbindungsressourcen kann in zwei Gruppen aufgeteilt werden:

- Der von dem Fehler betroffene Multiplexer und das von ihm getriebene Leitungsstück ist notwendig zur Darstellung der Schaltung, d.h. sein Ausgangssignal wird in der Schaltung verwendet.
- Der betroffene Multiplexer treibt ein nicht verwendetes Leitungsstück, er ist also nicht notwendig für die Funktion der synthetisierten Schaltung.

In beiden Fällen kann ein Fehler die Auswahl des Eingangssignals beeinflussen:

- Das fehlerhaft ausgewählte Eingangssignal wird in der Schaltung bereits verwendet.
- Das fehlerhaft ausgewählte Eingangssignal wird in der Schaltung nicht verwendet.
- Durch die bestehende Redundanz bei der Codierung der Konfigurationsbits zur Ansteuerung des Multiplexers kommt es zu keiner Veränderung des ausgewählten Eingangssignals.

Bei einem Fehler in den Konfigurationsbits eines verwendeten Multiplexers droht also die Auswahl eines falschen Eingangssignals. Da das Leitungsstück aktiv in der synthetisierten Schaltung verwendet wird, muss nach längerer Betriebszeit allerhöchstwahrscheinlich mit beobachtbaren Auswirkungen gerechnet werden. Dabei besteht kein nennenswerter Unterschied, ob das falsch ausgewählte Eingangssignal seinerseits in der Schaltung verwendet wird oder ob es von einer ungenutzten Verbindungsstruktur stammt.

Fehler in den Konfigurationsbits von ungenutzten Verbindungsstrukturen führen zu weniger kritischen Auswirkungen. Wählt ein Multiplexer ein bereits in der Schaltung anderweitig verwendetes Signal aus, so erhöht sich dessen kapazitive Last. Dies hat dann eine größere dynamische Verlustleistung insgesamt und eine erhöhte Signallaufzeit des zusätzlich belasteten Signals zur Folge. Im Extremfall führt diese erhöhte Signallaufzeit zu einer Timing-Verletzung der Schaltung, was wiederum zu beobachtbaren Fehlern führen kann. Bei der Verbindung eines ungenutzten Leitungsstücks mit einem anderen ungenutzten Leitungsstück treten in der Regel keine Probleme auf. Die ungenutzten Leitungsstücke werden in den synthetisierten Schaltungen zwar nicht verwendet, was aber nicht bedeuten muss, dass deren Signalpegel undefiniert ist. Verstärkte Auswirkungen durch externe und interne Störeinstrahlungen auf diese Leitungen können daher ausgeschlossen werden.

Durch die Architektur der Verbindungsstrukturen mit jeweils einem Treiber pro Leitungsstück kann sicher ausgeschlossen werden, dass es zu Kurzschlüssen in Folge von Fehlern in den Konfigurationsbits kommen kann. Dadurch sind auch mögliche Fehlereffekte wie Einbruch der Versorgungsspannung und lokale Zerstörung des FPGAs nicht zu erwarten. In FPGAs, die von dem bisher besprochenen Verbindungsschema abweichen, z.B. in denen Tri-State-Treiber für interne Leitungen implementiert sind, besteht die Gefahr von Beschädigung durch SEU-bedingte Fehler.

### 3.2.3 Konfigurierbare Speicher

In synthetisierten FPGA-Schaltungen können neben den Logikressourcen auch eine Vielzahl an Speichern verwendet werden. Dazu gehören neben den bereits erwähnten BlockRAMs (BRAM) in typischen Größen von 512, 1024 oder 2048 Worten zu je 9 Bit, 18 Bit oder 36 Bit auch kleinere Speicher, die häufig durch alternative Verwendung der LUTs dargestellt werden können. In der typischen Verwendung wird eine LUT durch das Konfigurationsinterface einmal während der FPGA-Konfiguration beschrieben. Danach liegt das Interface zum Beschreiben brach, es werden lediglich die Logik-Eingänge zur Adressierung des fortan als ROM betriebenen SRAM verwendet. Für eine Verwendung von LUTs als RAM müssen daher durch das FPGA-Design nur die notwendigen Steuersignale in der synthetisierten Schaltung verwendbar sein. Durch den Einsatz von LUTs als allgemeiner Speicher lassen sich RAM, Schieberegister und FIFOs sehr feingranular in verschiedenen Größen und sehr ressourcensparend erzeugen. Der Einsatz von LUTs in diesen Funktionen findet daher auch sehr häufig statt. Verbunden mit der effizienten Nutzung der LUTs als RAM geht auch einher, dass ein Großteil der vorhandenen Speicherzellen von der Schaltung tatsächlich verwendet werden kann. Daher ist auch der Anteil der Speicherzellen, die durch die Nicht-Verwendbarkeit eine logische Maskierung erfahren, relativ gering.

Im Gegensatz dazu sind bei BRAMs wegen der weniger flexiblen Größe häufig größere Teile des vorhandenen Speichers ungenutzt und dementsprechend unkritisch für das Schaltungsverhalten. Ebenfalls positiven Einfluss auf die Fehlerrate von BRAMs haben speziell hier verfügbare Fehlerkorrektur- und Fehlererkennungsmechanismen. Wegen der höheren Speicherdichte und der größeren Datenwortbreite als bei 1-Bit LUT-RAMs kann die Fehlerkorrektur- und Fehlererkennungslogik als fest verdrahtete Funktionalität zusammen mit den BRAM-Zellen implementiert werden. Wird nun bei der Anwendung des BRAMs nicht die volle Datenwortbreite benötigt, so können die überzähligen Bits eines Datenworts zur Speicherung von redundanten Daten verwendet werden. Bei der Schaltungssynthese muss keine Logik mit dieser Funktionalität erzeugt werden, sondern lediglich der gewünschte Mechanismus aktiviert werden. Durch die Verwendung dieser zusätzlichen Funktionalität entstehen aber i.A. auch verlängerte Speicherzugriffszeiten bzw. Logiklaufzeiten und eine erhöhte Verlustleistung durch zusätz-

liche aktive Schaltungsteile. Gerade bei sehr tiefen Speichern ist der zusätzliche Flächenaufwand bei der Implementierung vernachlässigbar, da diese Logik eine relativ kleine Fläche gegenüber den Speicherzellen des BRAMs darstellt. Da sich bei LUT-RAMs beim Flächenvergleich ein anderes, deutlich schlechteres Verhältnis einstellen würde, sind fest verdrahtete Fehlerkorrekturmechanismen bei diesen Speichern bei keinem FPGA-Hersteller verfügbar.

Genauso wie bei den FlipFlops in den Logikressourcen, ist die gespeicherte Information in den LUT-RAMs und BRAMs Teil des Schaltungszustandes. Wie viele SEUs zu beobachtbaren Auswirkungen führen, hängt deshalb auch wieder von der Schaltungsstruktur und dem Anwendungsfall der Schaltung ab.

## 3.3 ASIC-Teil von FPGAs

Neben den bisher betrachteten konfigurierbaren Logik-, Speicher- und Verbindungsressourcen sind in jedem FPGA eine Reihe von zusätzlichen Schaltungsteilen mit fester Funktionalität vorhanden. Dazu gehören z.B.

- Konfigurationslogik
- PowerUp- und Reset-Logik
- Chip-Überwachung (Brown-Out, Temperatur)
- Integrierte Oszillatoren mit fester Frequenz

## 3.4 Anteile bei der Verwendung von Speicher im FPGA

Die im FPGA vorhandenen Speicher können in zwei Bereiche unterteilt werden. Ein Teil des Speichers wird von der synthetisierten Schaltung verwendet, der andere Teil ist zur Konfiguration des FPGAs notwendig. Am Beispiel der Virtex4-FPGAs von Xilinx sollen die Anteile bei der Verwendung von Speicher im FPGA aufgezeigt werden. Die verwendeten Daten stammen dabei größtenteils aus Datenblättern von Xilinx, sowie von Entwicklern mit dem Fokus auf partieller Rekonfiguration.

Zuerst muss geklärt werden, wie viele und welche Ressourcen auf einem FPGA wirklich zur Verfügung stehen. Die meisten Hersteller haben eigene Bezeichnungen für die typischen Bestandteile von FPGAs. Um Vergleichbarkeit herstellen zu können, müssen die Hardware-Bestandteile erst in allgemein bekannte Bestandteile zerlegt werden. In der nachfolgenden Tabelle ist eine Übersicht der Virtex4-Ressourcen dargestellt.

In Virtex4-FPGAs sind LUTs und FlipFlops jeweils paarweise in sogenannten Slices angeordnet, die Anzahl der FlipFlops und LUTs ist daher zweimal so hoch wie die Anzahl der Slices. Die Anzahl der DSP-Slices und BRAMs steht in keinem festen Zusammenhang dazu.

| Device | Slices | LUTs | FF | 18 kB BRAMs | DSP-Slices |
|---|---|---|---|---|---|
| XC4VLX15 | 6144 | 12288 | 12288 | 48 | 32 |
| XC4VLX25 | 10752 | 21504 | 21504 | 72 | 48 |
| XC4VLX40 | 18432 | 36864 | 36864 | 96 | 64 |
| XC4VLX60 | 26624 | 53248 | 53248 | 160 | 64 |
| XC4VLX80 | 35840 | 71680 | 71680 | 200 | 80 |
| XC4VLX100 | 49152 | 98304 | 98304 | 240 | 96 |
| XC4VLX160 | 67584 | 135168 | 135168 | 288 | 96 |
| XC4VLX200 | 89088 | 178176 | 178176 | 336 | 96 |
| XC4VSX25 | 10240 | 20480 | 20480 | 128 | 128 |
| XC4VSX35 | 15360 | 30720 | 30720 | 192 | 192 |
| XC4VSX55 | 24576 | 49152 | 49152 | 320 | 512 |
| XC4VFX12 | 6144 | 12288 | 12288 | 36 | 32 |
| XC4VFX20 | 9216 | 18432 | 18432 | 68 | 32 |
| XC4VFX40 | 19968 | 39936 | 39936 | 144 | 48 |
| XC4VFX60 | 26624 | 53248 | 53248 | 232 | 128 |
| XC4VFX100 | 43520 | 87040 | 87040 | 376 | 160 |
| XC4VFX140 | 64512 | 129024 | 129024 | 552 | 192 |

Tabelle 3.1: Übersicht aller Virtex4-FPGAs von Xilinx [44]

Weitere wichtige Daten sind die Größe der Konfigurationsdatei und die notwendige Datenmenge zu Konfiguration einer einzelnen Ressource. Die Größe der Konfigurationsdatei kann dem Datenblatt direkt entnommen werden.

In Virtex4-Bausteinen wird die Konfiguration in sogenannten Columns, d.h. spaltenweise durchgeführt. Zur Konfiguration jedes der drei Column-Typen werden unterschiedlich viele Configuration Frames benötigt.

Aus den Tabellen 3.1, 3.3 und 3.4 kann nun eine Übersicht über die Aufteilung des gesamten verfügbaren Speichers in einem FPGA auf die jeweiligen funktionalen Bereiche erstellt werden (Tabelle 3.5). Die Tabelle 3.6 stellt die selben Daten im Verhältnis zur Größe der Konfigurationsdatei laut Tabelle 3.2 dar. Im Mittel werden in Virtex4-FPGAs 71,6% des Konfigurationsspeichers für das FPGA-Routing verwendet. Der zweitgrößte Anteil liegt bei den BRAMs, hier werden als Mittelwert 22,1% des Konfigurationsspeichers verwendet. Diese relativ hohe Zahl wird dadurch erreicht, das alle BRAMs bei der Konfiguration mit nutzerspezifischen Daten initialisiert werden können. Zur Konfiguration der Logik werden nur ca. 6% des Konfigurationsspeichers verwendet, der größte Teil davon dient zur Initialisierung der LUT-Speicherzellen. Weil die DSP-Blöcke hauptsächlich aus fest verdrahteter Logik bestehen, werden zur Konfiguration der DSP-Blöcke auch nur ca. 0,2% des gesamten Speichers verwendet.

Da es sich bei den hier vorgestellten Daten nur um Abschätzungen, nicht aber

| Device | Non-Configuration Frames | Configuration Frames | Device Frames | Configuration Array Size (words) |
|---|---|---|---|---|
| XC4VLX15 | 140 | 3,600 | 3740 | 147600 |
| XC4VLX25 | 234 | 5928 | 6162 | 243048 |
| XC4VLX40 | 376 | 9312 | 9688 | 381792 |
| XC4VLX60 | 536 | 13472 | 14008 | 552352 |
| XC4VLX80 | 710 | 17720 | 18430 | 726520 |
| XC4VLX100 | 948 | 23376 | 24324 | 958416 |
| XC4VLX160 | 1,260 | 30720 | 29460 | 1259520 |
| XC4VLX200 | 1,620 | 39120 | 37500 | 1603920 |
| XC4VSX25 | 440 | 6940 | 7380 | 284540 |
| XC4VSX35 | 660 | 10410 | 11070 | 426810 |
| XC4VSX55 | 1,104 | 17304 | 18408 | 709464 |
| XC4VFX12 | 248 | 3600 | 3848 | 147600 |
| XC4VFX20 | 360 | 5488 | 5848 | 225008 |
| XC4VFX40 | 660 | 10296 | 10956 | 422136 |
| XC4VFX60 | 1,040 | 15976 | 17016 | 665016 |
| XC4VFX100 | 1,660 | 25170 | 26830 | 1031970 |
| XC4VFX140 | 2,424 | 36444 | 3886 | 1494204 |

Tabelle 3.2: Virtex4 Frame Anzahl, ein Frame entspricht 41 words zu 32 bit, Configuration Overhead ist für alle Devices 1312 words zu 32 bit [45]

vom Hersteller gelieferte Daten handelt, sind kleine Ungenauigkeiten unvermeidlich. Einen direkten Vergleich für ein spezielles FPGA liefert Morgan et al. [46], die gezeigten Werte unterschieden sich weniger als 1% von den hier bestimmten Werten für die einzelnen Teilbereiche des FPGAs. In Tabelle 3.6 ist zusätzlich ein Vergleich zu Configuration Array Size (Tabelle 3.2) dargestellt. Die Abschätzung ist für alle FPGAs mit einer einzigen Ausnahme (XC4VSX55) plausibel und mit einem Mittelwert von 94,7% sehr nah an einer exakten Berechnung.

Die Ergebnisse von Tabelle 3.6 ermöglichen es nun eine erste Abschätzung für die Fehlerempfindlichkeit eines FPGAs gegenüber SEUs abzugeben. Mit Hilfe der Speicheranteile und abgeschätzten mittleren Derating-Faktoren für einzelne Bereiche kann für jeden Teilbereich eine Fehlerhäufigkeit bestimmt werden. Die gesamte Fehlerwahrscheinlichkeit wird durch die Summe aller Teilfehlerwahrscheinlichkeiten gebildet. Die Teilfehlerwahrscheinlichkeiten liefern auch einen Anhaltspunkt für evtl. Verbesserungen des Systems.

| Column-Typ | Anzahl pro Column | Frames Routing | Frames Logik | Frames DSP | Frames BRAM |
|---|---|---|---|---|---|
| CLB | 16 | 20 | 2 | | |
| DSP | 4 | 20 | | 1 | |
| BRAM | 4 | 20 | | | 64 |

Tabelle 3.3: Virtex4 Column-Typen und Anzahl an Konfigurations-Frames

| Ressourcen-Typ | Routing | Logic | DSP | BRAM | Gesamt |
|---|---|---|---|---|---|
| CLB | 1640 | 164 | 0 | 0 | 1804 |
| DSP | 6560 | 0 | 328 | 0 | 6888 |
| BRAM | 6560 | 0 | 0 | 20992 | 27552 |

Tabelle 3.4: Virtex4 Anzahl Konfigurationsbits pro Ressource

## 3.5 Schutzkonzepte für FPGAs

Wegen des besonderen Aufbaus und der Funktionsweise von FPGAs müssen angepasste Schutzkonzepte verwendet werden. Für die Auswahl derer lässt sich ein FPGA in zwei Bereiche unterteilen:

- Konfigurationsspeicher, die während der Laufzeit konstant bleiben
- Speicher, die der synthetisierten Schaltung zur Verfügung stehen

### 3.5.1 Fehlererkennung mittels Readback

Alle Konfigurationsspeicher werden während der Konfiguration einmalig beschrieben und bleiben dann im Normalbetrieb der Schaltung unverändert. Kein Fehler in diesem Bereich, der durch einen SEU verursacht wurde, kann durch den Betrieb der Schaltung selbständig korrigiert werden. Damit kann ein Fehler ohne zeitliche Beschränkung durch ein Auslesen der Konfigurationsdaten eindeutig erkannt werden. Ein häufig eingesetztes Verfahren ist der zyklische Readback der Konfigurationsdaten, bei dem der gesamte Speicherbereich des FPGAs sequentiell vom Anfang bis zum Ende ausgelesen wird. Zur Fehlererkennung kann dabei ein direkter Vergleich mit den Konfigurationsdaten gezogen werden oder mit den zurückgelesenen Konfigurationsdaten erst eine Prüfsumme gebildet werden, die dann abschließend mit einem Soll-Wert verglichen werden kann. Generellen Einfluss auf die Geschwindigkeit bei der Fehlererkennung mittels Readback hat die Geschwindigkeit des Readback-Interfaces, d.h. mit welcher Taktrate und welcher zur Verfügung stehenden Busbreite können die Daten aus dem Konfigurationsspeicher ausgelesen werden. Diese Daten und zusätzliche Eigenschaften (z.B. Burst-Länge, Zeit für einen ersten Zugriff, Verzögerungen durch den Vergleich bedingt) lassen

| Device | Routing | Logic | DSP | BRAM | Gesamt |
|---|---|---|---|---|---|
| XC4VLX15 | 3043840 | 251904 | 10496 | 1007616 | 4313856 |
| XC4VLX25 | 5195520 | 440832 | 15744 | 1511424 | 7163520 |
| XC4VLX40 | 8606720 | 755712 | 20992 | 2015232 | 11398656 |
| XC4VLX60 | 12385280 | 1091584 | 20992 | 3358720 | 16856576 |
| XC4VLX80 | 16531200 | 1469440 | 26240 | 4198400 | 22225280 |
| XC4VLX100 | 22356480 | 2015232 | 31488 | 5038080 | 29441280 |
| XC4VLX160 | 30228480 | 2770944 | 31488 | 6045696 | 39076608 |
| XC4VLX200 | 39360000 | 3652608 | 31488 | 7053312 | 50097408 |
| XC4VSX25 | 5877760 | 419840 | 41984 | 2686976 | 9026560 |
| XC4VSX35 | 8816640 | 629760 | 62976 | 4030464 | 13539840 |
| XC4VSX55 | 15534080 | 1007616 | 167936 | 6717440 | 23427072 |
| XC4VFX12 | 2965120 | 251904 | 10496 | 755712 | 3983232 |
| XC4VFX20 | 4434560 | 377856 | 10496 | 1427456 | 6250368 |
| XC4VFX40 | 9446400 | 818688 | 15744 | 3022848 | 13303680 |
| XC4VFX60 | 13277440 | 1091584 | 41984 | 4870144 | 19281152 |
| XC4VFX100 | 21359360 | 1784320 | 52480 | 7892992 | 31089152 |
| XC4VFX140 | 31330560 | 2644992 | 62976 | 11587584 | 45626112 |

Tabelle 3.5: Virtex4 Anzahl Konfigurationsbits nach Ressourcenbereich

sich in der langfristigen Bandbreite $B_{Readback}$ des Readback-Interfaces zusammenfassen.

Wie aus Abbildung 3.10 ersichtlich ist, kann es ja nach Verfahren und Fehlerzeitpunkt unterschiedlich lange dauern, bis ein Fehler in den Konfigurationsdaten entdeckt wird. Im besten Fall tritt ein Fehler in einer Speicherzelle auf, kurz bevor diese Zelle mittels Readback ausgelesen wird. Beim direkten Vergleich kann dieser Fehler also innerhalb eines Speicherzugriffszyklus (Zugriffszeit) erkannt werden. Tritt der Fehler in einer Zelle aber auf, die unmittelbar zuvor ausgelesen wurde, so wird der Fehler erst erkannt, nachdem der gesamte Konfigurationsspeicher einmal ausgelesen wurde. Bei statistisch gleichverteilt auftretenden Fehlern ergibt sich für die Fehlererkennungszeit ein Mittelwert gleich der halben Zeit zum Auslesen des gesamten Konfigurationsspeichers (siehe Gleichung 3.2).

Wenn die Fehlererkennung mittels Prüfsummenvergleich realisiert wird, verlängert sich die Fehlererkennungszeit nochmals. Dies ist durch die Tatsache bedingt, dass die Prüfsumme natürlich nur für den gesamten Konfigurationsspeicher bekannt ist. Damit muss auf jeden Fall der gerade laufende Readback-Zyklus bis zum Ende durchlaufen werden. Tritt der Fehler aber an einer Position im Konfigurationsspeicher auf, die im gegenwärtigen Readback-Zyklus bereits gelesen wurde, so verzögert sich die Fehlererkennung bis zum Abschluss des nächsten Zyklus.

Die mittlere Fehlererkennungszeit für diesen Fall (Abbildung 3.11) kann auf

| Device | Routing / Gesamt | Logik / Gesamt | DSP / Gesamt | BRAM / Gesamt | Verhältnis |
|---|---|---|---|---|---|
| XC4VLX15 | 70,6% | 5,8% | 0,2% | 23,4% | 91,3% |
| XC4VLX25 | 72,5% | 6,2% | 0,2% | 21,1% | 92,1% |
| XC4VLX40 | 75,5% | 6,6% | 0,2% | 17,7% | 93,3% |
| XC4VLX60 | 73,5% | 6,5% | 0,1% | 19,9% | 95,4% |
| XC4VLX80 | 74,4% | 6,6% | 0,1% | 18,9% | 95,6% |
| XC4VLX100 | 75,9% | 6,8% | 0,1% | 17,1% | 96,0% |
| XC4VLX160 | 77,4% | 7,1% | 0,1% | 15,5% | 97,0% |
| XC4VLX200 | 78,6% | 7,3% | 0,1% | 14,1% | 97,6% |
| XC4VSX25 | 65,1% | 4,7% | 0,5% | 29,8% | 99,1% |
| XC4VSX35 | 65,1% | 4,7% | 0,5% | 29,8% | 99,1% |
| XC4VSX55 | 66,3% | 4,3% | 0,7% | 28,7% | 103,2% |
| XC4VFX12 | 74,4% | 6,3% | 0,3% | 19,0% | 84,3% |
| XC4VFX20 | 70,9% | 6,0% | 0,2% | 22,8% | 86,8% |
| XC4VFX40 | 71,0% | 6,2% | 0,1% | 22,7% | 98,5% |
| XC4VFX60 | 68,9% | 5,7% | 0,2% | 25,3% | 90,6% |
| XC4VFX100 | 68,7% | 5,7% | 0,2% | 25,4% | 94,1% |
| XC4VFX140 | 68,7% | 5,8% | 0,1% | 25,4% | 95,4% |
| Mittelwert | 71,6% | 6,0% | 0,2% | 22,1% | 94,7% |

Tabelle 3.6: Virtex4 Anteile Konfigurationsspeicher nach Ressourcenbereich

einfache Weise hergeleitet werden:

- Ein Readback-Zyklus dauert $t_0$.
- Der Fehler tritt im Abstand von $t$ vor dem Ende des aktuellen Readback-Zyklus an unbestimmter Stelle in der Konfiguration auf.
- Der Fehler tritt mit der Wahrscheinlichkeit $p$ in einem Teil des Konfigurationsspeichers auf, der im aktuellen Zyklus noch gelesen wird.
- Wegen des linearen Fortschritts beim Readback ist die Wahrscheinlichkeit $p = \frac{t}{t_0}$.

Die mittlere Fehlererkennungszeit kann unter diesen Annahmen rechnerisch wie in Gleichung 3.3 bis 3.6 bestimmt werden.

Die Konfigurationsspeicher-Fehlererkennung durch Readback kann prinzipiell auf zwei Arten implementiert werden:

- Controller im FPGA integriert, sowohl als synthetisierte Schaltung als auch fest verdrahtet.
- Externer Controller

Die Fehlererkennung mittels Readback und direktem Vergleich hat den Vorteil der schnelleren Fehlererkennung, kann aber in vielen Fällen nicht angewandt werden. Dafür müsste dem Vergleichssystem sowohl ein schneller Zugriff auf die im FPGA gespeicherten Konfigurationsdaten, als auch auf einen Speicher mit

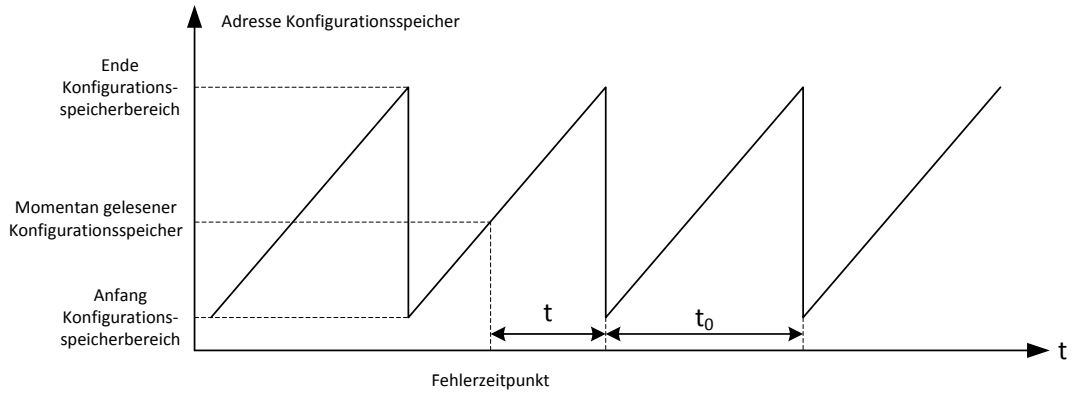



Abbildung 3.10: Fehlererkennung durch Readback, resultierende Zeiten bis zur Fehlererkennung

$$t_{Detect,Avg} = \frac{1}{2} \cdot \frac{Size\ of\ configuration\ memory}{B_{Readback}} \tag{3.2}$$

Mittlere Fehlererkennungszeit mit zyklischem Readback und direktem Vergleich

den Originaldaten, etwa einem Konfigurations-PROM-Baustein, zur Verfügung stehen. Dabei macht es keinen Unterschied ob der notwendige Controller für die Fehlererkennung im FPGA oder außerhalb implementiert ist. Die aktuellen Konfigurationsdaten müssten entweder schnell aus dem FPGA gesendet werden oder die Originaldaten schnell ins FPGA eingelesen werden können. Beides erfordert eine größere Anzahl an I/O-Pins um die nötige Bandbreite bei dieser Übertragung zu erreichen. Falls die Konfigurationsdaten zusätzlich mit einer Verschlüsselung vor unberechtigtem Zugriff geschützt wurden, erhöhen sich die Anforderungen an das Fehlererkennungssystem weiter.

Eine Fehlererkennung mittels Prüfsumme hingegen wird fast ausschließlich als interner Controller realisiert. Da hier zur Fehlererkennung nur eine kleine Menge an zusätzlichem Speicher für die Prüfsumme notwendig ist und der Readback an sich durch einen ohnehin vorhandenen Controller durchgeführt wird, haben die meisten FPGA-Hersteller eine solche Fehlererkennung fest in ihren FPGAs integriert. Diese Methodik bietet auch die Vorteile, dass keine Konfigurationsdaten das FPGA verlassen müssen (siehe Abschnitt 3.1.6) und dass keine zusätzlichen I/O-Pins benötigt werden. Damit spielt die Bandbreite der I/O-Pins in diesem Fall keine Rolle und die sonst benötigten I/O-Pins können für andere Zwecke verwendet werden. Ein weiterer Vorteil ist auch der reduzierte Leistungsbedarf dieses Verfahrens gegenüber dem direkten Vergleich, weil die Datentransfers am externen Konfigurationsspeicher nicht notwendig sind.

Der Nachteil einer höheren Worst-Case-Fehlererkennungszeit und auch eines höheren Mittelwerts kann fast vollständig durch mehrere blockweise gebildete Prüfsummen verhindert werden. Dabei wird der Konfigurationsspeicher in meh-

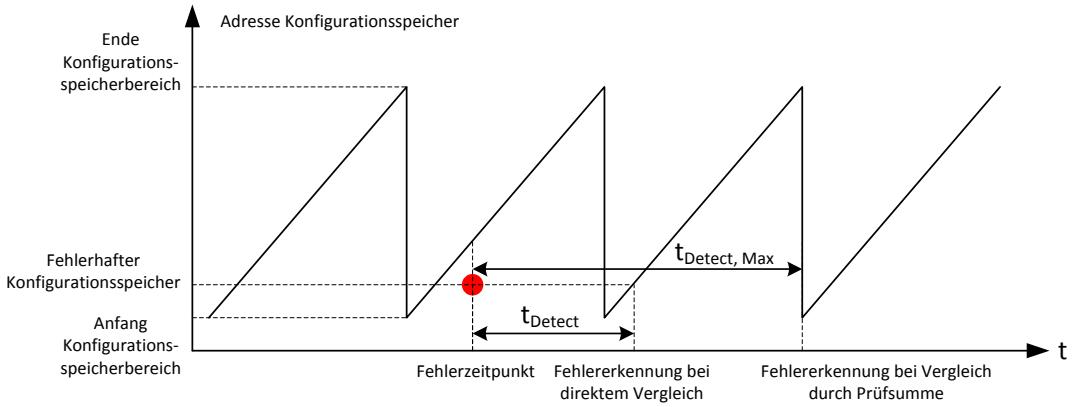



Abbildung 3.11: Fehlererkennung durch Readback, resultierende Zeiten bis zur Fehlererkennung mit Prüfsumme

$$
\begin{aligned}
t_{Detect,Avg} &= p \cdot t + (1-p) \cdot (t + t_0) && (3.3)\\
&= p \cdot t + t + t_0 - p \cdot t - p \cdot t_0 && (3.4)\\
&= t + t_0 - \frac{t}{t_0} \cdot t_0 && (3.5)\\
t_{Detect,Avg} &= t_0 && (3.6)
\end{aligned}
$$

Mittlere Fehlererkennungszeit mit zyklischem Readback und Vergleich der Prüfsumme

rere, kleine Bereiche anstatt einem großen Bereich eingeteilt und für jeden Bereich eine separate Prüfsumme gebildet. Durch diese Maßnahme verkürzt sich die Zeit, bis eine Aussage über die Richtigkeit des zuletzt ausgelesenen Blocks getroffen werden kann. Sowohl die Worst-Case- als auch die mittlere Fehlererkennungszeit verringern sich durch eine solche Maßnahme fast um den Faktor zwei. In einer Grenzfallbetrachtung, bei der jeder Block nur die Größe von einem Konfigurationswort hat, entspricht dieses Vorgehen wieder dem direkten Vergleich von Konfigurationsdaten (siehe Gleichung 3.7).

Schon bei einer Anzahl von mehr als zehn Speicherblöcken unterscheiden sich sowohl Worst-Case- als auch mittlere Fehlererkennungszeit bei beiden Ansätzen nur noch um zehn Prozent. Wegen der Vorteile hinsichtlich der benötigten Pinanzahl, des Energieaufwands und des vergleichsweise geringen Mehraufwands zur

$$k = 1 + \frac{1}{n} \qquad (3.7)$$

Faktor $k$ für die Verlängerung der Fehlererkennungszeit gegenüber direktem Readback-Vergleich, Anzahl der Speicherblöcke $n$

Speicherung der zusätzlichen Prüfsummen wird die Fehlererkennung mittels Prüfsumme häufiger als der direkte Vergleich eingesetzt. Nach erfolgter Fehlererkennung müssen geeignete Maßnahmen getroffen werden, um auch die Auswirkungen der erkannten Fehler korrigieren zu können.

### 3.5.2 Fehlerkorrektur mittels Scrubbing

Bei der zyklischen Rekonfiguration von FPGAs, dem sogenannten Scrubbing, wird ein anderes Konzept verfolgt [47] [48]. In besonderen Umgebungen mit hoher Intensität radioaktiver Strahlung muss bei FPGAs mit häufigeren Fehlern in den Konfigurationsdaten gerechnet werden. Da nach jedem Fehler die Funktion des FPGAs gegenüber der gewünschten Funktionalität verändert sein kann, müssen diese Fehler korrigiert werden, um das FPGA weiter verwenden zu können. Dies kann durch Rekonfiguration erreicht werden, welche im Zuge eines System-Resets durchgeführt wird. Alternativ dazu kann versucht werden, während des laufenden FPGA-Betriebs die Fehler mittels dynamischer Rekonfiguration zu korrigieren. Die verwendeten FPGAs müssen diese Funktionalität natürlich unterstützen. Wenn in einem Konfigurationsbereich des FPGAs ein Fehler erkannt wurde, so kann dieser Teil der Konfiguration neu beschrieben werden, um den Ursprungszustand wieder herzustellen. Alternativ dazu können die gesamten Konfigurationsdaten des FPGAs kontinuierlich durch Überschreiben erneuert werden. Sofern dieser Rekonfigurationsvorgang ohne Störung der Funktion der synthetisierten Schaltung ablaufen kann, werden alle Fehler nach einer definierten Zeit korrigiert. Dadurch reduziert sich die Wahrscheinlichkeit einer beobachtbaren Auswirkung der korrigierten Fehler. Fehlerfreier Betrieb kann dennoch nicht garantiert werden. Der Vergleich der Arbeitsgeschwindigkeit der Schaltung ($f_{Clk} \geq 100MHz$) mit der Rekonfigurationsrate des Konfigurationsspeichers ($f_{Reconfiguration} \leq \frac{100}{s}$) zeigt deutlich die Schwachstelle dieses Ansatzes.

Ein Fehler in der Konfiguration kann eine Vielzahl an Speicherzellen in der synthetisierten Schaltung beeinflussen bis der Fehler wieder korrigiert wird. Wenn die Fehler in der synthetisierten Schaltung durch Rückkopplung sehr lange oder auch dauerhaft im Schaltungszustand bleiben, so korrigiert die Rekonfiguration in diesem Fall nicht alle Fehler. Findet jedoch nur eine einfache Pipeline-Datenverarbeitung ohne Rückkopplungen statt, so stellt sich nach einiger Zeit wieder ein fehlerfreier Schaltungszustand ein. Abhängig von Schaltungstyp bzw. Schaltungsverhalten kann eine zyklische Rekonfiguration alle Fehler der Schaltung beseitigen. Da bei diesem Verfahren aber nicht sicher gestellt ist, dass keine fehlerhaften Daten von der Schaltung ausgegeben werden, ist eine zusätzliche Fehlererkennung vorteilhaft. Mit einer integrierten Fehlererkennung können die von der Schaltung produzierten Ausgangsdaten kontrolliert werden und folglich auch die Korrektur fehlerhafter Konfigurationsdaten gestartet werden.

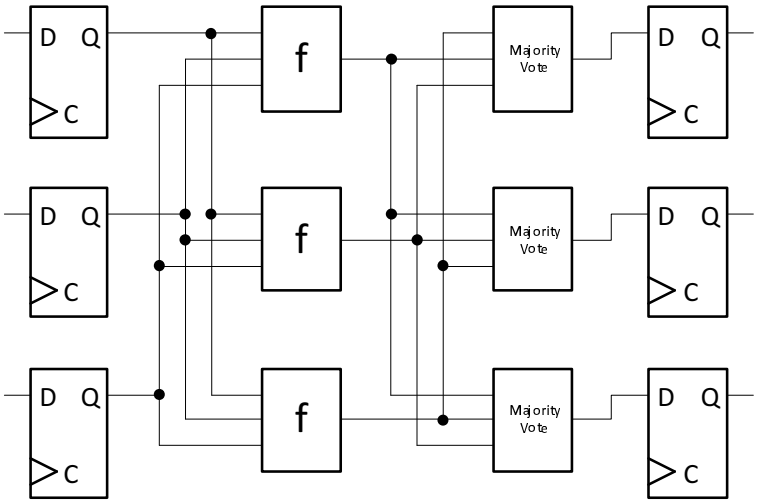



Abbildung 3.12: Implementierung von TMR für eine Teilschaltung

### 3.5.3 Double und Triple Modular Redundancy

Digitale Schaltungen werden gegenüber transienten Fehlern häufig durch Vervielfachung von Schaltungsteilen geschützt. Ein Ansatz ist die Verdopplung von Schaltungsteilen zur redundanten Berechnung von Ergebnissen (Double Modular Redundancy DMR), die dann einfach verglichen werden, um eine Fehlererkennung zu realisieren. Dabei kann aber nicht bestimmt werden, welche Teilschaltung fehlerhaft ist. Der Hardwareaufwand für diesen Ansatz ist etwas mehr als zweimal so hoch wie bei einer ungeschützten Schaltung, da die wesentlichen Schaltungsteile zweimal vorhanden sein müssen und eine zusätzliche Vergleichslogik benötigt wird. Bei der Triple Modular Redundancy TMR (Abbildung 3.12) wird dann die Schaltung verdreifacht [46]. Dies erlaubt dann zusätzlich zur Fehlererkennung auch eine Fehlerkorrektur, da mit drei Ergebnissen ein Mehrheitsentscheid möglich ist [49]. Auch beim Auftreten eines einzelnen Fehlers kann die Schaltung gefahrlos weiter betrieben werden. Zusätzlich kann die Teilschaltung mit abweichendem Ausgangssignal eindeutig als fehlerhaft identifiziert werden.

Diese Verfahren werden sowohl im ASIC-Bereich als auch in synthetisierten Schaltungen in FPGAs eingesetzt. Bei der Anwendung in einem FPGA ergibt sich aber ein zusätzlicher Vorteil. Da ein Mehrheitsentscheid basierend auf den Ausgangssignalen der Teilschaltungen eine allgemeine Fehlererkennung ermöglicht, sind dadurch alle Fehlertypen abgedeckt. Der Vergleich von Ausgangssignalen ermöglicht also die Erkennung von Fehlern in der synthetisierten Schaltung und von Fehlern bedingt durch falsche Konfigurationsdaten. Durch den Einbau eines Mehrheitsentscheiders als Hardwarefunktion kann in jedem Taktzyklus die Funktion der Schaltung überprüft werden. Dies unterscheidet den TMR-Ansatz ebenfalls deutlich von Readback oder Scrubbing.

Bei der Realisierung von DMR und TMR sind aber auch einige Nachteile

in Kauf zu nehmen. An erster Stelle steht der erhöhte Hardwareaufwand für die zusätzlichen Instanzen der ursprünglichen Schaltungsfunktion und die Voter-Logik (Mehrheitsentscheider). Damit bedeutet die Verwendung von DMR einen Hardware-Mehraufwand von etwas mehr als 100% und die Verwendung von TMR einen Mehraufwand von mehr als 200% gegenüber der ursprünglichen, einfachen Implementierung der Schaltung. Einher mit der vergrößerten Gesamtgröße der Schaltung geht auch ein erhöhter Energiebedarf. Wegen des identischen Aufbaus der vervielfachten Schaltungen wird die Gesamtschaltung den doppelten bzw. dreifachen Energiebedarf haben. Weiterhin muss für die Anwendung von DMR und TMR die Schaltung modifiziert werden, da der Voter eine zusätzliche Schaltungsfunktion darstellt. Dies erhöht die Verzögerungszeit der jeweiligen Logikfunktion und verringert damit unter Umständen die maximale Taktfrequenz der Schaltung.

Wegen der hohen Kosten bei der Realisierung von Schaltungen mit DMR und TMR werden diese Verfahren, trotz des guten Schutzes gegenüber transienten Fehlern, nur bei Anwendungen mit sehr hoher geforderter Zuverlässigkeit wie z.B. in der Luft- und Raumfahrt verwendet. Ansätze zur Verringerung des Aufwandes durch z.B. partielles DMR/TMR existieren zwar [50], erfordern aber neben angepassten Entwicklungswerkzeugen auch Information über die Verteilung der kritischen Schaltungsteile, damit dieser Ansatz auch effizient eingesetzt werden kann.

# Kapitel 4

# Alternatives Konzept Rückwärtsanalyse

In diesem Kapitel werden die notwendigen Voraussetzungen und Annahmen für ein neuartiges Konzept zur Bestimmung der Fehlermaskierungsfaktoren digitaler Schaltungen gegenüber transienten Fehler gezeigt. Das Verfahren wird im Weiteren als Rückwärts-Analyse (Backwards Analysis BA) bezeichnet [51] [52].

## 4.1 Eigenschaften einer digitalen Schaltung bzgl. regulärer Betrieb

Kombinatorische digitale Schaltungen ohne rückgekoppelte Signale zeichnen sich durch ihr deterministisches Verhalten aus. Für eine gegebene digitale Logikfunktion und gegebene Eingangswerte kann der Ausgangswert zweifelsfrei bestimmt werden (Gleichung 4.2). Die zeitliche Reaktion des Ausgangssignals auf eine Änderung eines oder mehrerer Eingangssignale kann durch charakteristische Verzögerungszeiten abgebildet werden, wobei auch physikalische Parameter, wie z.B. Versorgungsspannung der Gatter und kapazitive Last der Signale, hier eine Rolle spielen. Komplexere Logikfunktionen können durch Verschaltung einfacherer Logikgatter erzeugt werden. Die resultierende Logikfunktion kann sowohl in geschlossener Form als auch in Anlehnung an die Schaltungsstruktur in einer verketteten Form dargestellt werden. Insbesondere diese Form der Darstellung ist bei Schaltungen mit mehr als einem Ausgangssignal geeignet, da hier häufig Teile der Logikfunktion überlappend genutzt werden können.

Die Verwendung von sequentiellen Elementen (taktflankengesteuerte Register

$$Q = f(i_1, ..., i_x) \tag{4.1}$$
$$Q = f(I) \tag{4.2}$$

Ausgangssignal $Q$ in Abhängigkeit von Logikfunktion $f$ und den Eingangssignalen $I$

$$\begin{pmatrix} s_1(t+1) \\ ... \\ s_y(t+1) \\ q_1(t) \\ ... \\ q_z(t) \end{pmatrix} = \begin{pmatrix} f_1(i_1(t), ..., i_x(t), s_1(t), ..., s_y(t)) \\ ... \\ f_y(i_1(t), ..., i_x(t), s_1(t), ..., s_y(t)) \\ g_1(i_1(t), ..., i_x(t), s_1(t), ..., s_y(t)) \\ ... \\ g_z(i_1(t), ..., i_x(t), s_1(t), ..., s_y(t)) \end{pmatrix} \tag{4.3}$$

$$\begin{pmatrix} S(t+1) \\ Q(t) \end{pmatrix} = \begin{pmatrix} F(I(t), S(t)) \\ G(I(t), S(t)) \end{pmatrix} \tag{4.4}$$

Darstellung eines Automaten durch Logikfunktionen $F$ und $G$ sowie primäre Eingangssignale $I$, Zustandssignale $S$ und Ausgangssignale $Q$

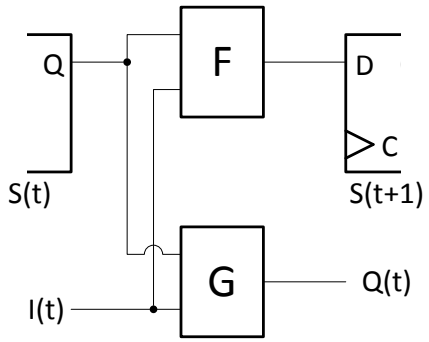



Abbildung 4.1: Ausgerollte sequentielle Schaltung

oder pegelgesteuerte Latches) ermöglicht das Verschalten mehrerer kombinatorischer Logikfunktionen zu einem größeren Automaten mit zustandsabhängigem Verhalten, was komplexe Schaltungsfunktionen ermöglicht. Als Nebeneffekt der sequentiellen Elemente hängt die Verzögerungszeit eines Eingangssignals bis zu einem Schaltungsausgang hauptsächlich von der Anzahl der sequentiellen Stufen und der Periodenzeit des verwendeten Taktsignals ab. Eingangssignale der Teilfunktionen der sequentiellen Schaltung können sowohl die primären Eingangssignale als auch die Ausgangssignale der einzelnen sequentiellen Stufen sein. Damit sind auch Rückkopplungseffekte in den Schaltungen möglich, ohne das es zu Oszillation oder instabilen und undefinierten Signalen in der Schaltung kommt.

Gemäß Gleichung 4.4 wird der Schaltungszustand $S(t+1)$ für den nächsten Zeitschritt aus den momentanen Eingangssignalen $I(t)$ und dem Schaltungszustand $S(t)$ gebildet. Abbildung 4.1 stellt den gleichen Sachverhalt als ausgerollte sequentielle Schaltung dar. Es findet also eine eindeutige Abbildung des gegenwärtigen Zustandes auf einen zukünftigen Zustand statt, im Gegensatz dazu kann aus dem gegenwärtigen Zustand nicht der vorherige Zustand bestimmt werden, da die Umkehrfunktionen $F^{-1}$ und $G^{-1}$ keine eindeutige Abbildung erlauben.

## 4.2 Eigenschaften einer digitalen Schaltung bzgl. Schaltungstest

In den Anfangszeiten der integrierten Schaltungen, in denen nur einzelne oder wenige Logikgatter in ICs integriert waren, stellte der Schaltungstest nur ein kleines Problem dar. Zum Einen waren im gesamten System nur sehr wenige Gatter verbaut, zum Anderen waren praktisch alle Signale durch Testpunkte abgreifbar, was die Diagnose stark vereinfacht hat. Mit steigender Integrationsdichte verringert sich die Anzahl der Ausgangssignale im Verhältnis zur gesamten Anzahl an Signalen in der Schaltung, was eine Fehlererkennung zunehmend erschwert.

Aufgabe des Schaltungstests bei digitalen Schaltungen war das Erkennen von fehlerhaft produzierten, defekten Gattern welche die vorgegebene Logikfunktion gar nicht oder nur unter erhöhter Gatterverzögerungszeit erfüllen. Je nach verwendetem Fehlermodell müssen unterschiedlich viele Testfälle pro Gatter abgearbeitet werden. Ein häufig verwendetes Fehlermodell für integrierte digitale Schaltungen ist das Stuck-at-Fehlermodell. Hier wird angenommen, dass ein Schaltungssignal durch einen Fehler dauerhaft mit einer Betriebsspannungen ($V_{dd}$ oder $GND$) verbunden ist.

Um also alle möglichen Stuck-at-Fehler (Stuck-at-1 s-a-1 und Stuck-at-0 s-a-0) in einer Schaltung testen zu können, muss mit den angelegten Testsignalen erreicht werden, dass jedes Gatterausgangssignal mindestens einmal den Logikpegel 1 und 0 ausgibt. Die Fähigkeit beim Schaltungstest einen Gatterausgang auf einen gewünschten Logikpegel zu steuern heißt sinngemäß Steuerbarkeit des Signals und ist eine Voraussetzung für den Schaltungstest.

Um das korrekte Verhalten des getesteten Logikgatters nachvollziehen zu können ist eine weitere Voraussetzung für den Schaltungstest, dass ein fehlerhaftes Signal sich auch auf mindestens einen Schaltungsausgang auswirkt. Diese Eigenschaft nennt man Beobachtbarkeit des Signals. Beobachtbarkeit ist in einer bestimmten Schaltung gegeben, wenn vom zu beobachtenden Signal bis zu mindestens einem primären Ausgang ein durchgehender empfindlicher Pfad besteht. Im Fehlerfall würde sich der am Ausgang anliegende Signalpegel von dem erwarteten Signalpegel unterscheiden.

Ja nachdem wo sich das zu testende Signal innerhalb der Schaltung befindet, kann es unterschiedlich schwierig sein die Steuerbarkeit und die Beobachtbarkeit zu erfüllen. Tendenziell steigt der Aufwand zum Erreichen von Steuerbarkeit mit der Anzahl der Gatter, die hierarchisch dem zu testenden Signal vorgeschaltet sind. Die Beobachtbarkeit ist dagegen tendenziell schwerer mit steigender Anzahl dem zu testenden Signal nachgeschalteter Gatter zu erreichen. Bei sequentiellen Schaltungen verschärft sich diese Problematik nochmals, da hier gewöhnlich die Mehrzahl der Signale nicht direkt von primären Eingangssignalen gesteuert werden kann. Das sequentielle Verhalten der Schaltung erfordert den Wechsel weg von einfachen Testsignal-Mustern hin zu komplexen Testsignal-Sequenzen

$$\begin{pmatrix} E_{s_1}(t+1) \\ ... \\ E_{s_y}(t+1) \end{pmatrix} = \begin{pmatrix} \frac{\partial s_1(t+1)}{\partial s_1(t)} \cdot E_{s_1}(t) \vee ... \vee \frac{\partial s_1(t+1)}{\partial s_y(t)} \cdot E_{s_y}(t) \\ ... \\ \frac{\partial s_y(t+1)}{\partial s_1(t)} \cdot E_{s_1}(t) \vee ... \vee \frac{\partial s_y(t+1)}{\partial s_y(t)} \cdot E_{s_y}(t) \end{pmatrix} \quad (4.5)$$

$$E_S(t+1) = \begin{pmatrix} \frac{\partial s_1(t+1)}{\partial s_1(t)} & ... & \frac{\partial s_1(t+1)}{\partial s_y(t)} \\ ... & ... & ... \\ \frac{\partial s_y(t+1)}{\partial s_1(t)} & ... & \frac{\partial s_y(t+1)}{\partial s_y(t)} \end{pmatrix} \cdot E_S(t) \quad (4.6)$$

$$E_S(t+1) = \frac{\partial S(t+1)}{\partial S(t)} \cdot E_S(t) \quad (4.7)$$

Fehlerfortpflanzung in digitalen Schaltungen, Bestimmung mittels empfindlicher Pfade $\frac{\partial s_m}{\partial s_n}$ und Fehlersignal $E_{s_n}$ bzw. Fehlervektor $E_S$

über mehrere Taktzyklen hinweg. Diese Sequenzen müssen ebenfalls Steuerbarkeit und Beobachtbarkeit unter Berücksichtigung des zeitlichen Verhaltens der Schaltung erreichen. Da in sequentiellen Schaltungen häufig auch Rückkopplungen vorkommen, spielt die gegenseitige Beeinflussung der Signale hier eine große Rolle. Während bei rein kombinatorischen Schaltungen maximal alle Eingangssignale für das Testsignal-Muster zu berücksichtigen sind, spielt bei sequentiellen Schaltungen die sequentielle Tiefe eine entscheidende Rolle. Abhängig von der sequentiellen Tiefe bzw. von der Anzahl der Taktschritte bis ein zu testendes Signal stimuliert werden kann und bis das getestete Signal am Ausgang beobachtet werden kann, erhöht sich die Anzahl der zu berücksichtigenden Eingangssignale. In jedem notwendigen Taktschritt müssen unter Umständen alle Eingangssignale definierte Pegel aufweisen. Die Menge der zu berücksichtigenden Eingangssignale berechnet sich demnach ähnlich wie in Gleichung 2.6 und wächst exponentiell mit der sequentiellen Tiefe der Schaltung.

Wie bereits im Kapitel 4.1 erwähnt, kann aus einem Schaltungszustand zusammen mit den primären Eingangssignalen eindeutig auf den Zustand der Schaltung zu einem späteren Zeitpunkt geschlossen werden. Ebenso kann durch Anwendung der booleschen Differenz die Fehlerausbreitung dargestellt werden. Die Gleichungen 4.5 bis 4.7 stellen dies dar. In dieser Darstellung können jedoch keine Mehrfachfehler berücksichtigt werden.

## 4.3 Schwachstellen alternativer Konzepte

Für den Entwurf eines leistungsfähigen Konzepts zur Bestimmung der Fehlermaskierungsfaktoren ist es notwendig alle besonderen Eigenschaften, sowohl positive als auch negative, der bekannten Verfahren genau zu kennen. Hier soll ein kurzer

Vergleich der in Kapitel 2.3 beschriebenen Verfahren gezeigt werden.

### 4.3.1 Fehlerinjektion

Die Fehlerinjektion erlaubt zwar die Betrachtung von allen durch das Simulationsmodell dargestellten Fehlermechanismen, jedoch liefert ein einzelner Test nur ein für einen Punkt gültiges Ergebnis. Mit diesem Ergebnis kann für die Fehlermaskierung eine Aussage weder für den zeitlichen Bereich unmittelbar davor oder danach getätigt werden. Deshalb muss der Testabstand bei der Fehlerinjektion relativ kurz sein, um eine Aussage bzgl. der Fehlermaskierung eines Signals für einen durchgehenden Bereich treffen zu können. Der zeitliche Aufwand zur Durchführung von Fehlerinjektion zur Bestimmung von Fehlermaskierungsfaktoren ist daher immens. Fehlerinjektion mit statistisch verteilten Testpunkten führt zwar bei ausreichend hoher Testanzahl zu relativ genauen Mittelwerten der Fehlermaskierung, eine Aussage über den zeitlichen Verlauf kann damit aber nicht getroffen werden.

### 4.3.2 Probabilistische Ansätze

Probabilistische Ansätze zur Bestimmung von Fehlermaskierungsfaktoren reduzieren die Komplexität einer digitalen Schaltung auf eine statische Matrix. Dadurch können Ergebnisse zwar mit äußerst geringem Rechenaufwand erzeugt werden, das anwendungsfall-abhängige Verhalten der Schaltung wird aber ungenügend abgebildet. Weiterhin irritiert die Tendenz dieses Ansatzes bei größerer sequentieller Tiefe einer Schaltung stets kleinere Fehlermaskierungsfaktoren zu produzieren, weswegen dieses Verfahren zur Erlangung von exakten Ergebnissen ebenfalls ungeeignet erscheint.

### 4.3.3 SAT-Solver

Die Bestimmung von Fehlermaskierungsfaktoren unter Verwendung von SAT-Solvern bietet den deutlichen Vorteil, dass das Verhalten der Schaltung exakt abgebildet werden kann. Dafür entsteht aber speziell bei großer sequentieller Tiefe der Schaltungen ein sehr hoher Aufwand zur Berücksichtigung aller Abhängigkeiten ähnlich wie bei der Testmuster-Generierung, vor allem bedingt durch die wachsende Anzahl der Schaltungseingänge bei den ausgerollten Schaltungen. Da beim SAT-Solver-Ansatz auch keine Anwendungsdaten verwendet werden, werden alle prinzipiell möglichen Schaltungszustände berücksichtigt. Viele dieser Zustände werden in sinnvollen Anwendungen nur selten erreicht werden, ebenfalls viele Zustände sind auf Grund der Schaltungsfunktion generell nicht erreichbar. Damit die gewonnenen Ergebnisse nicht verfälscht werden, müssen diese unerreichbaren Zustände vorab bestimmt werden und dann von der Analyse ausgeschlossen werden. Weiterhin werden mit dieser formalen Analyse zwar alle Mög-

lichkeiten der Auswirkung von einzelnen transienten Fehlern berücksichtigt, reelle Anwendungsszenarien der Schaltung werden aber nicht gesondert betrachtet. Daher tendieren die gewonnenen Ergebnisse stark zu einer pessimistischen Abschätzung der Fehlermaskierung. Ein Großteil des Geschwindigkeitsvorteils wird durch eine vereinfachte Unterscheidung von kritischen und unkritischen Zustandsbits erzielt. Schon eine einzelne gefundene Eingangskombination, die einen transienten Fehler an einem Zustandsbit beobachtbar machen würde, reicht aus um dieses Zustandsbit als kritisch einzustufen. Wenn eine solche Kombination gefunden wurde, kann selbstverständlich die Suche nach weiteren Eingangskombinationen in Bezug auf dieses Zustandsbit abgebrochen werden. Durch diesen Ansatz hat ebenfalls eine Überabschätzung der Schaltungsempfindlichkeit zur Folge.

### 4.3.4 ACE-Analyse

Zur Durchführung einer ACE-Analyse ist ein spezielles Modell zur Bewertung der Schaltungsempfindlichkeit notwendig. Dieses Modell muss aber zusätzlich zum eigentlichen Schaltungsmodell erstellt werden und muss die zu betrachtenden Fehlermaskierungseffekte explizit darstellen. Dieser Vorgang kann mit Fehlern beim Modellentwurf behaftet und abhängig von der Schaltungsstruktur komplexer als die Schaltung selbst sein. Die Abbildung der Fehlermaskierungseffekte kann daher stark lückenhaft sein. Bei einer Modifikation der Schaltung muss auch das Empfindlichkeitsmodell nachgebessert werden.

### 4.3.5 Vorwärts-Analyse

Dominierende Schwachstelle bei der Vorwärts-Analyse ist das schlechte Verhalten bei Schaltungen mit interner Rückkopplung. Die Markierungen für beobachtbare Fehler werden jeden Taktzyklus eine sequentielle Stufe weitergeschoben, zusätzlich entsteht jeden Taktzyklus an jedem Speicher eine neue zusätzliche Markierung. Dadurch ist es höchstwahrscheinlich, dass es zu einer Überflutung mit Markierungen kommt. Um dem entgegen zu wirken werden die Markierungen nach einer einfachen Heuristik verworfen, wenn zu viele vorhanden sind. Mit jeder verworfenen Markierung geht das Risiko einher, eine evtl. wichtige Markierung verworfen zu haben, die einen kritischen Zeitraum an einem Speicher zugeordnet ist. Dadurch könnten systematisch falsche Analyseergebnisse erzeugt werden. Weiterhin führt die Überflutung mit Markierungen auch zu einer deutlich verlangsamten Analyse, da die zu bearbeitende Datenmenge sehr hoch ist.

## 4.4 Grundkonzept Rückwärts-Analyse

Als Hauptproblem der bisher bekannten Verfahren zur Bestimmung von Fehlermaskierungsfaktoren wurden die Probleme und Ineffizienz bei der Behandlung

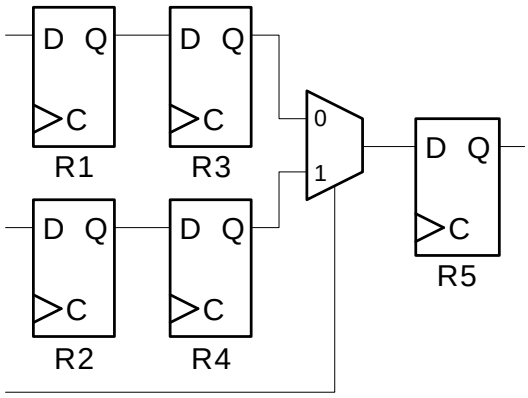



Abbildung 4.2: Einfache Pipeline mit Maskierung am Pipeline-Ende

von transitiven Maskierungseffekten erkannt. Häufig wird hier ein spekulatives Vorgehen verwendet, d.h. es wird zunächst vermutet, dass ein Speicher kritisch für eine Anwendung sein könnte. Im weiten Verlauf der Analyse wird dann diese Spekulation durch die modellierten Maskierungseffekte entweder bestätigt oder verworfen. Der Rechenaufwand hängt dabei davon ab, wie bald eine solche Spekulation verworfen wird. Wird die Spekulation früh verworfen, entsteht prinzipiell ein wesentlich kleinerer Rechenaufwand als wenn die Spekulation nach vielen sequentiellen Stufen sogar bestätigt wird. Im Bezug auf transitive Maskierung können Fälle auftreten, in denen eine Spekulation erst kurz vor dem Schaltungsausgang verworfen wird und somit ein relativ hoher Rechenaufwand entstanden ist. Das im Folgenden vorgestellte Konzept der Rückwärts-Analyse soll systematisch diese Nachteile vermeiden.

### 4.4.1 Erster Ansatz transitive Effekte

Grundsätzlich würde sich eine starke Verringerung des notwendigen Rechenaufwandes ergeben, wenn die unnötigen Berechnungen vermieden werden könnten. Der neue Ansatz soll mit Hilfe von Abbildung 4.2 erklärt werden. Hier ist einfach zu erkennen, dass die kritischen Register in erster Linie durch die Auswahl des Multiplexers am Schaltungsende festgelegt werden. Ohne weitere Einschränkung wird festgelegt, dass das Ausgangsregister R5 immer gegenüber SEUs empfindlich ist, da es immer beobachtbar sein soll. Das Register R3 oder R4 ist abhängig von der Auswahl des Multiplexers als kritisch anzusehen. Für den vorherigen Zeitschritt kann unter Kenntnis der Schaltung gefolgert werden, dass wenn R3 oder R4 kritisch waren, auch R1 bzw. R2 kritisch gewesen sein muss. Wenn R3 oder R4 unkritisch war, kann sicher ausgeschlossen werden, dass R1 bzw. R2 kritisch war. Dadurch könnte auch der Rechenaufwand eingespart werden, der zur Bestimmung der Empfindlichkeit von R1 bzw. R2 notwendig wäre.

Durch Umformung und Uminterpretation von Gleichung 4.5 kann ein neuer

$$\begin{pmatrix} c_{s_1}(t) \\ ... \\ c_{s_y}(t) \end{pmatrix} = \begin{pmatrix} \frac{\partial s_1(t+1)}{\partial s_1(t)} \cdot c_{s_1}(t+1) \vee ... \vee \frac{\partial s_y(t+1)}{\partial s_1(t)} \cdot c_{s_y}(t+1) \\ ... \\ \frac{\partial s_1(t+1)}{\partial s_y(t)} \cdot c_{s_1}(t+1) \vee ... \vee \frac{\partial s_y(t+1)}{\partial s_y(t)} \cdot c_{s_y}(t+1) \end{pmatrix} \tag{4.8}$$

$$C_S(t) = \begin{pmatrix} \frac{\partial s_1(t+1)}{\partial s_1(t)} & ... & \frac{\partial s_y(t+1)}{\partial s_1(t)} \\ ... & ... & ... \\ \frac{\partial s_y(t+1)}{\partial s_1(t)} & ... & \frac{\partial s_y(t+1)}{\partial s_y(t)} \end{pmatrix} \cdot C_S(t+1) \tag{4.9}$$

$$C_S(t) = \frac{\partial S(t+1)}{\partial S(t)} \cdot C_S(t+1) \tag{4.10}$$

Bestimmung der Kritikalität von Zustandsspeichern $c_{s_n}(t)$ durch empfindliche Pfade zwischen Zustandsspeichern $\frac{\partial s_m(t+1)}{\partial s_n(t)}$ und der Kritikalität von Zustandsspeichern zu einem späteren Zeitpunkt $c_{s_m}(t+1)$

Sachverhalt dargestellt werden. Die Empfindlichkeit $C$ eines Zustandsspeichers zu einem früheren Zeitpunkt kann aus den empfindlichen Pfaden ausgehend von diesem Speicher zu allen Zustandsspeichern und deren Empfindlichkeit zum nachfolgenden Zeitpunkt bestimmt werden. Ein Zustandsspeicher auf der Quellenseite einer ausgerollten sequentiellen Schaltung ist demnach als kritisch bzw. empfindlich anzusehen, wenn mindestens ein empfindlicher Pfad zwischen diesem Zustandsspeicher und einem anderen Zustandsspeicher auf der Senken-Seite besteht, welcher zudem selbst als kritisch betrachtet wird.

Durch Anwendung der Bestimmungsgleichung für die Kritikalität der Speicher werden bildlich gesehen Markierungen für die Kritikalität rückwärts durch die Schaltung propagiert. Dieses Propagieren findet jeden Zeitschritt und so lange statt, bis von einem Zustandsspeicher keine empfindlichen Pfade in Richtung der Quellen in der ausgerollten sequentiellen Schaltung mehr führen oder bis eine Markierung an einem primären Eingang angelangt ist. Sind zu einem Zeitpunkt keine empfindlichen Pfade von einem Zustandsspeicher weg vorhanden, so wird die Markierung gelöscht. An einem Zustandsspeicher kann zu jedem Zeitpunkt nur eine Markierung über die Kritikalität vorhanden sein.

Neue Markierungen werden jeden Taktzyklus an vom Benutzer vorgegebenen Schaltungsknoten erzeugt. Im Allgemeinen sind dies alle Zustandsspeicher, welche nur noch über kombinatorische Logikfunktionen direkt mit primären Schaltungsausgängen verbunden sind. Für diese Speicher wird angenommen, dass sie immer beobachtbar sind. Wenn aus der Sicht des Nutzers andere Schaltungsknoten besonders kritisch für die Anwendung sind, können auch an diesen Knoten die Markierungen erzeugt werden. Alle Schaltungsknoten, die in diesem Kontext vom Benutzer ausgewählt werden, sind sinngemäß immer empfindlich.

Für einen Speicher auf der Senken-Seite ist nicht ausgeschlossen, dass kritische Pfade zu mehreren Speichern auf der Quellen-Seite existieren, da ein Folge-

Zustand meist von mehreren vorherigen Zustandsspeichern abhängt (Abbildung 4.4). In diesem Fall kann jeweils eine Markierung an alle Speicher auf der Quellen-Seite weitergeben werden, sofern empfindliche Pfade dorthin vorhanden sind.

Wenn mehrere empfindliche Pfade zwischen den als kritisch betrachteten Speichern auf der Senken-Seite und einem Speicher auf der Quellen-Seite bestehen, so spiegelt dies nur die Tatsache wieder, dass sich ein einzelner SEU im weiteren Betrieb der Schaltung durchaus auf mehrere Zustandsspeicher auswirken kann. Beim Propagieren der Markierungen würden daher bei einem Quellen-Speicher mehrere Markierungen eintreffen (Abbildung 4.3). Da ein Speicher aber nur kritisch oder unkritisch sein kann, werden mehrere eintreffende Markierungen zu einer einzelnen Markierung reduziert.

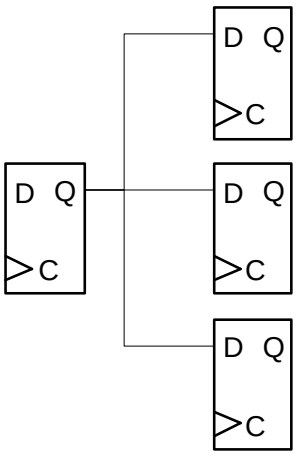



Abbildung 4.3: Fan-Out-Situation

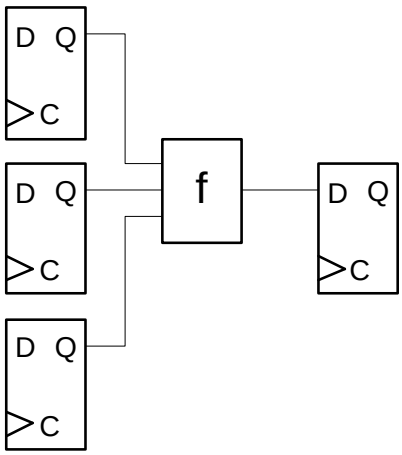



Abbildung 4.4: Fan-In-Situation

Bemerkenswert ist, dass zur Berechnung der empfindlichen Pfade die Werte der Zustandssignale und Eingangssignale zum Zeitpunkt t herangezogen werden müssen, jedoch die Kritikalität der Zustandsspeicher vom Zeitpunkt t+1 stammt. Die Bestimmungsgleichung für die Kritikalität der Zustandsspeicher zum Zeitpunkt t ist lösbar, wenn die Kritikalität in unmittelbarer Zukunft bekannt ist.

### 4.4.2 Separate Simulation und Analyse

Da in einer Schaltungssimulation niemals bereits vorab der Endzustand bekannt ist, zeigt sich mit der Bestimmungsgleichung 4.10 ein Problem auf. Durch die Abhängigkeiten im Zeitbereich ist ausgeschlossen, dass der bisher beschriebene Ansatz parallel und schritthaltend mit einer Schaltungssimulation durchgeführt werden kann. Die Simulation bestimmt zwar den Wert der Schaltungssignale, kann aber keine Auskunft über die Kritikalität zu zukünftigen Zeitpunkten geben.

Um dieses Problem zu lösen soll ein zweigeteiltes Verfahren verwendet werden. Der erste Teil des Verfahrens ist eine einfache Schaltungssimulation der unverän-

derten Schaltung, bei der die Pegel aller Schaltungsknoten aufgezeichnet werden. Die Berechnungen der Simulation entsprechen denen von Gleichung 4.4 und sind auf Grund der zeitlichen Abhängigkeiten der verwendeten Signale problemlos durchführbar.

Der zweite Teil des Verfahrens stellt die eigentliche Analyse dar. Der wesentliche Ansatz basiert auf dem Versuch, die Probleme bei der Lösung von Gleichung 4.10 zu beseitigen. Durch die Aufzeichnung des Signalverlaufs im ersten Teil des Verfahrens ist der zukünftige Zustand der Schaltung zu jedem Zeitpunkt bekannt. Eine Analyse der Schaltungsempfindlichkeit, gemäß dem im vorigen Abschnitt beschriebenen Ansatz, besteht nun aus zwei sich wiederholenden Teilaufgaben:

- Stimulation der Schaltungsknoten mit Signalpegeln zu einem Zeitpunkt, welche aus den zuvor aufgezeichneten Signalverläufen stammen
- Bestimmung der empfindlichen Pfade in der Schaltung und Weiterpropagieren der Empfindlichkeitsmarkierungen

Da diese Art der Analyse iterativ die Empfindlichkeit der Zustandsspeicher in zeitlich umgekehrter Reihenfolge berechnet, muss daher mit der Stimulation der Schaltung mit den Signalpegeln am Ende des aufgezeichneten Zeitbereichs begonnen werden. Nach dem Weiterpropagieren der Empfindlichkeitsmarkierungen werden dann Signalpegel zur Stimulation der Schaltung zum nächstmöglichen früheren Zeitpunkt verwendet. Die Analyse ist demnach beendet, wenn Stimulationsdaten vom Anfang der Simulation verwendet wurden.

Bei der Anwendung auf reelle Schaltungen sind aber einige zusätzliche Effekte zu betrachten, die in den bisherigen Betrachtungen keine Rolle spielten. Diese zusätzlichen Effekte entstehen durch die Verzögerungszeiten der Schaltungselemente. In erster Konsequenz ist festzuhalten, dass nicht alle Schaltungsknoten exakt zeitgleich ihren Signalpegel ändern. Durch die Signallaufzeit kommt es bei mehrstufigen kombinatorischen Schaltungen innerhalb einer sequentiellen Stufe der gesamten Schaltung zu einer zeitlich wellenförmigen Änderung der Signalpegel. Schaltungsknoten nah an den Registerausgängen werden bereits kurz nach einer Flanke des Taktsignals ihren Pegel verändern können, Signale in den nachgeschalteten Logikstufen ändern ihren Pegel tendenziell eher zu einem späteren Zeitpunkt. Kurz vor Ablauf einer Taktperiodenzeit sollten alle Signale innerhalb der Schaltung auf einem stabilen Zustand sein, d.h. es finden keine Änderungen mehr statt. Dieses Verhalten ist auch notwendig, damit keine Verletzungen der Setup- und Hold-Bedingung auftreten kann, was gegen einen korrekten Betrieb der Schaltung sprechen würde.

Für die Stimulation der Schaltung hat dies den Einfluss, dass nicht alle Schaltungssignale ihren Wert zu diskreten Zeitpunkten etwa im Abstand der Taktperiode ändern, sondern Änderungen zu jedem Zeitpunkt auftreten können (Abbildung 4.5). In der Praxis treten bei den Änderungszeitpunkten natürlich Häufungen im Bereich nach den Taktflanken auf. Dies ist durch die meist kleine Anzahl kritischer Pfade in Schaltungen bedingt, d.h. nur in wenigen Schaltungsteilen finden Signaländerungen bis kurz vor der nächsten Taktflanke statt. Für den Ablauf

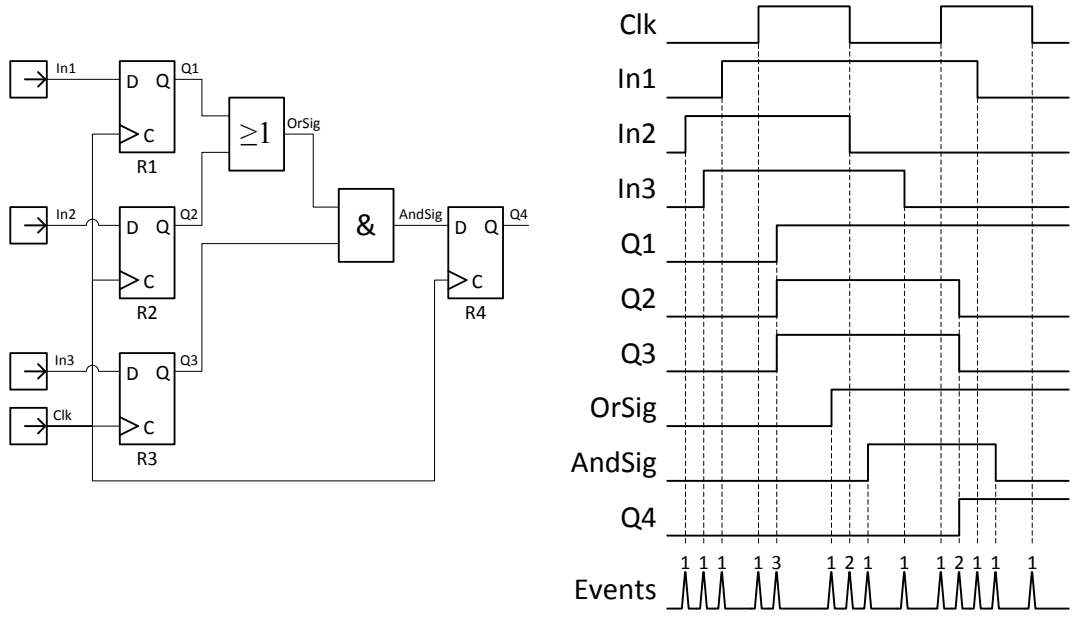



Abbildung 4.5: Verteilung der Signalpegel-Änderungen in einer digitalen Schaltung

der Analyse bringt dies eine Veränderung mit sich. Die Stimulation der Schaltung mit aufgezeichneten Signalpegeln findet nach wie vor in zeitlich umgekehrter Reihenfolge statt. Es werden aber nicht nach jeder Stimulation der Schaltung die empfindlichen Pfade bestimmt und die Empfindlichkeitsmarkierungen weiterpropagiert. Vielmehr muss aus dem Schaltungszustand bzw. den Signalpegeln selbst bestimmt werden, wann die Markierungen weiterpropagiert werden.

### 4.4.3 Trigger-Mechanismen bei Speichern

Die notwendigen Bedingungen für ein weiterpropagieren der Empfindlichkeitsmarkierungen leiten sich direkt aus der Funktion des speichernden Bauteils ab. Dies soll am Beispiel eines einfachen D-FlipFlops erklärt werden. Der entscheidende Zeitpunkt für die Fehlerfortpflanzung bei einem FlipFlop ist in dem Moment der Taktflanke, wobei die Richtung der Taktflanke auch mit der spezifizierten Flankenrichtung (steigend oder fallend) des FlipFlops übereinstimmen muss. Dieser Zeitpunkt soll im weiteren Verlauf Triggerzeitpunkt genannt werden. Zu diesem Zeitpunkt wird der Wert des Eingangssignals eingelesen und zum Ausgang durchgereicht. Nachdem das Eingangssignal des FlipFlops von den primären Eingangssignalen der Schaltung und den Ausgangssignalen der Zustandsspeicher abhängt, muss zum Triggerzeitpunkt die Bestimmung der empfindlichen Pfade und das Propagieren der Empfindlichkeitsmarkierungen durchgeführt werden.

Ausgehend vom zuvor genannten Beispiel eines D-FlipFlops müssen die Triggerbedingungen verallgemeinert werden. Dies ist notwendig, wenn bei einem Zu-

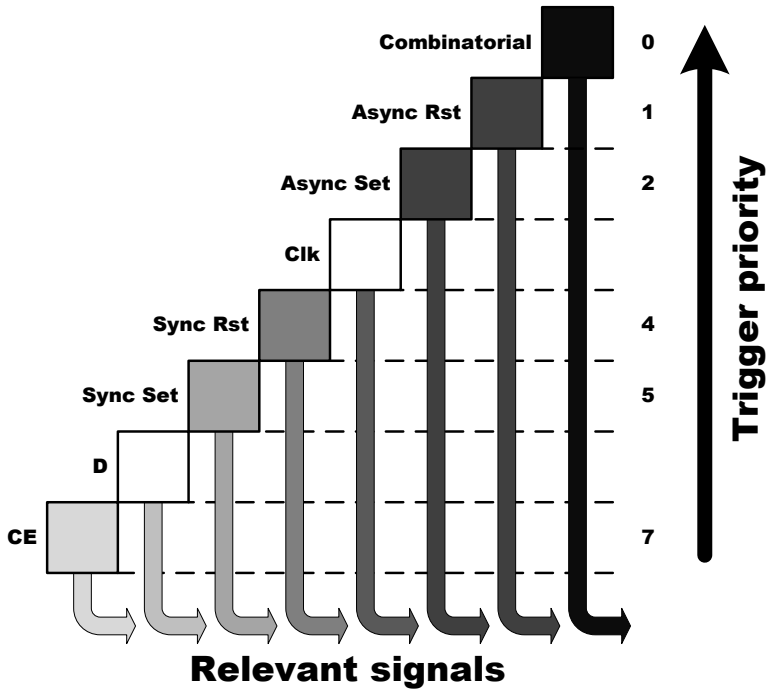



Abbildung 4.6: Priorität der Trigger-Quellen (vertikal) und zu berücksichtigende Signale abhängig von der Trigger-Quelle (horizontal)

standsspeicher mehrere Bedingungen zur Aktivierung des Speichers möglich sind. Typische Trigger-Möglichkeiten eines Speichers sind:

- Asynchroner Reset
- Asynchroner Set
- In Verbindung mit einer Taktflanke passender Flankenrichtung:
  - Synchroner Reset
  - Synchroner Set
  - Übernahme vom Dateneingang, evtl. Berücksichtigung eines Clock-Enable-Signals

Bei einem Speicher können aber auch nur Teile dieser Trigger-Möglichkeiten vorhanden sein. Bei mehr als einer einzigen Möglichkeit muss aber durch eine Vergabe von Prioritäten exakt definiert sein, welcher Triggermechanismus Vorrang hat. Beim Betrieb der Schaltung kann es häufig vorkommen, dass mehrere Trigger-Bedingungen gleichzeitig erfüllt sind, jedoch nur eine Einzelne auf Grund der Funktion des Speichers ausgeführt wird. Über die Priorität des Triggermechanismus wird der dominierende Effekt gekennzeichnet. Das tatsächliche Auftreten und Erfüllen einer Trigger-Bedingung soll im weiteren Trigger-Event genannt werden.

Abbildung 4.6 stellt den Zusammenhang der Triggerquellen, deren Priorität und der zu berücksichtigenden Signale für ein D-FlipFlop dar. Abhängig von der

Priorität der Triggerquelle haben unterschiedlich viele Signale Einfluss auf den Wert des Ausgangssignals des Speichers. Durch diesen Ansatz wird die logische Maskierung unter den Triggersignalen realisiert. So hat z.B. ein aktives ARst-Signal an einem FlipFlop zur Folge, dass keine weiteren Triggersignale bei diesem Trigger-Event als relevant betrachtet werden, da diese funktional untergeordnet sind. Im Gegensatz dazu wird bei einem einfachen Trigger mittels Taktsignal und aktivem CE-Signal der Ausgangswert durch das D-Signal bestimmt. Dieser Triggermechanismus hat in diesem Beispiel die niedrigste Trigger-Priorität (Priorität 7), daher sind alle Eingangssignale als relevant zu betrachten. In der vereinfachten Annahme würde ein Fehler an jedem der überhalb angeordneten Triggersignale (Priorität 6 bis 0) zu einem abweichenden Ausgangssignal führen. Die logische Maskierung der Triggersignale könnte jedoch noch detaillierter ausgeführt werden, wenn die Abhängigkeiten der Speicherfunktion hinsichtlich gegenwärtigem und zukünftigem Wert des Ausgangssignals berücksichtigt würden. Die beiden möglichen Szenarien sollen beispielhaft erklärt werden:

- An einem speichernden Element wird eine einzige aktive Triggerbedingung erkannt. Abhängig vom unverfälschten Wert des Ausgangssignals des Speichers würde eine Aktivierung einiger höherpriorer Trigger-Möglichkeiten keine Verfälschung des Ausgangssignals verursachen. Diese höherprioren Trigger-Möglichkeiten müssen daher nicht als relevant betrachtet werden. Beispiel: Aktiver Clk-CE-Trigger (Priorität 7), D-Eingang auf Wert 1, daher unkritische Eingänge SSet (Priorität 5) und ASet (Priorität 2).
- An einem speichernden Element werden mehrere aktive Triggerbedingungen erkannt, die Funktion der dominierenden Triggerbedingung ist für den Ausgangswert des Speichers verantwortlich. Im Falle eines Fehlers eines Signals, dass für die dominierende Triggerbedingung verantwortlich ist, kann evtl. die weitere erfüllte niederpriore Triggerbedingung zum gleichen Wert des Ausgangssignals führen. Die Signale der aktiven Triggerbedingung können daher als nicht relevant betrachtet werden. Beispiel: Aktiver ARst-Trigger, aktiver Clk-CE-Trigger, D-Eingang auf Wert 0, daher unkritischer ARst-Eingang.

Da die zuvor beschriebenen Maskierungs-Mechanismen eine zusätzliche detaillierte Beschreibung des Speichers erfordern, mehr Signale ausgewertet werden müssen und zudem mehrfache Trigger-Bedingungen zum gleichen Zeitpunkt häufig auf Grund der Schaltungssynthese nicht eintreten können, sollen diese Effekte im Weiteren nicht betrachtet werden.

Das Eintreten einer Triggerbedingung wird durch die Auswertung der Eingangssignale des Speichers bestimmt. Je nach Bedingung muss dabei der Pegel eines Signals mit einem Vorgabewert übereinstimmen (z.B. bei asynchronen Triggersignalen) oder eine Folge von Eingangssignalen durchlaufen werden (bei flankengetriggerten Funktionen). Zur Erkennung einer Folge muss zur Auswertung der vorherige Wert gespeichert werden. Für die Rückwärts-Analyse bedeutet dies, dass auch die Signalfolgen in umgekehrter Reihenfolge durchlaufen werden.

Dementsprechend kehrt sich die Flankenrichtung bei flankengesteuerten Eingängen um, d.h. eine steigenden Flanke wird bei einem Wechsel des Signalpegels von Logisch 1 zu Logisch 0 erkannt.

### 4.4.4 Berücksichtigung der Informations-Lebensdauer

In Kapitel 2.2.4 wurde bereits dargestellt, dass die Ansteuerung der vorhandenen Speicherzellen einen starken Einfluss auf die Lebensdauer der gespeicherten Informationen hat. Die maximale Lebensdauer einer Information ist durch den Zeitpunkt zweier Schreibzugriffe auf diese Zelle beschränkt. Durch die Überwachung der Eingangssignale des Speichers und der damit verbundenen Erkennung von Trigger-Events kann die Lebensdauer einer Information genauer eingegrenzt werden. Der Zeitpunkt, zu dem eine Speicherzelle neu beschrieben wird, stimmt weitestgehend mit dem Triggerzeitpunkt überein.

Zur Erkennung eines Lese-Zugriffs auf einen Speicher sind die nachfolgenden Gatter ausschlaggebend. Lese-Zugriffe auf einen Speicher entstehen durch Schreib-Zugriffe anderer, nachgeschalteter Speicher. Wenn also ein Speicher neu beschrieben wird, ist der neu geschriebene Speicherinhalt von seinen Eingangssignalen abhängig. Welche Eingangssignale Einfluss auf den neuen Speicherwert haben hängt von der Priorität des aktivierten Trigger-Mechanismus ab. Für alle als relevant betrachteten Eingangssignale werden nun alle Pfade bis hin zum nächsten Speicher durchlaufen. Alle Speicher, die am Beginn dieser ermittelten Pfade stehen, hätten daher einen Einfluss auf den neuen Wert des gerade getriggerten Speichers. Daher findet an allen diesen Speichern am Beginn der Pfade ein Lese-Zugriff statt.

Wie bereits erwähnt, kann die Informationslebensdauer eines Speichers durch zwei Schreib-Zugriffe grob eingegrenzt werden. Für jeden gespeicherten Wert existiert genau ein Schreib-Zugriff. Der genaue Zeitpunkt und das Auftreten eines Schreib-Zugriffs wird im Weiteren Write-Event genannt. Der Zeitpunkt und das Auftreten eines Lese-Zugriffs auf einen Speicher wird daher sinngemäß als Read-Event bezeichnet.

In Abbildung 4.7 ist beispielhaft eine Situation mit mehreren Senken-Registern dargestellt. Durch die steigenden Taktflanken der registerspezifischen Taktsignale werden die Write-Events an den Registern definiert. Die beiden nachgeschalteten Register R2 und R3 verursachen aber zeitgleich mit den eigenen Write-Events auch Read-Events am Register R1, da R2 und R3 die gespeicherte Information von R1 weiterverarbeiten. Tritt kein Read-Event zwischen zwei Write-Events auf, so wurde die gespeicherte Information nicht weiter verarbeitet. Tritt an einem Speicher nach einem Write-Event mindestens ein oder auch mehrere Read-Events auf, so definiert das am spätesten eingetretene Read-Event das Ende der Lebensdauer der gespeicherten Information. Treten Write- und Read-Event an einem Speicher exakt gleichzeitig auf, so sollte im Sinne einer "Pipeline"-Datenverarbeitung das Read-Event einem früheren Write-Event zugeordnet wer-

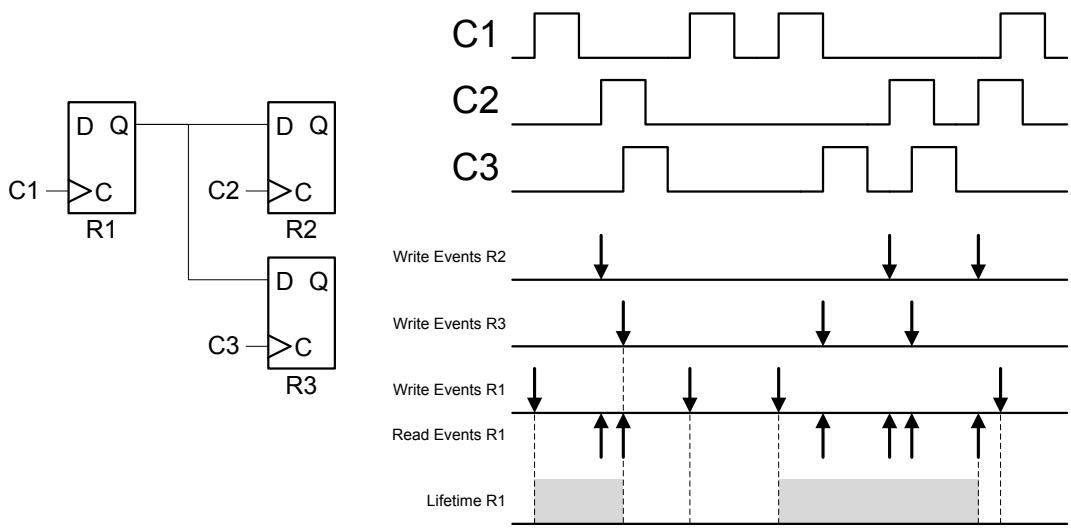



Abbildung 4.7: Generierung von Write- und Read-Events sowie Bestimmung der Informationsdauer in Abhängigkeit davon

den. Dieses Verhalten ist analog zu Abbildung 2.9.

Bei der Auswertung der Write- und Read-Events mittels Rückwärts-Analyse treten Read-Events wegen des umgekehrten zeitlichen Ablaufes natürlich vor den zugehörigen Write-Events auf. Um also die Lebensdauer einer Information bestimmen zu können, müssen für diesen Speicher lediglich alle Read-Events in einer Liste gesammelt werden und beim Auftreten eines Write-Events aus dieser Liste das am spätesten aufgetretene Read-Event ausgewählt werden. Alle anderen Read-Events, die zwischen dem Write-Event und dem letzten Read-Event liegen, können ohne weitere Berücksichtigung gelöscht werden. Mit Auftreten und Auswertung eines Write-Events ist die Bestimmung eines Zeitraums von Informationslebensdauer bei der Rückwärts-Analyse eindeutig abgeschlossen. Beim Konzept der Vorwärts-Analyse besteht diese Eindeutigkeit nicht.

### 4.4.5 Integration der Maskierungseffekte in ein Analyse-Schema

Die bisher vorgestellten Teil-Aspekte Informationslebensdauer, logische und transitive Maskierung lassen sich effizient in ein Analyse-Schema verbinden. Beim Auftreten eines Write-Events kann die Lebensdauer der gespeicherten Information bestimmt werden. Wenn die Lebensdauer größer als null ist, d.h. ein oder mehrere Read-Events zu diesem Write-Event vorhanden sind, so wurde die gespeicherte Information weiterverarbeitet und könnte daher einen Einfluss auf die primären Ausgangssignale haben. Wären im Gegensatz dazu aber keine Read-Events vorhanden und daher die Lebensdauer gleich null, so hätte die gespeicherte Information sicher keinen Einfluss auf die Ausgangssignale.

Zusätzlich muss gefolgert werden, dass bei einer Informationslebensdauer gleich null auch keines der Eingangssignale dieses Speichers einen Einfluss auf die primären Ausgangssignale haben kann, da diese gespeicherte Information nicht weiterverarbeitet wurde. Bei Lebensdauer gleich null sollen daher keine Read-Events an die Vorgänger-Speicher gesendet werden. Durch das Anwenden dieser Bedingung bei der Propagation der Read-Events kann die transitive Maskierung zusammen mit der Informationslebensdauer dargestellt werden.

Auf ähnliche Art kann auch die logische Maskierung der kombinatorischen Gatter und die Maskierung auf Grund der Triggerpriorität integriert werden. Nachdem durch Bestimmung der Lebensdauer entschieden wurde, ob generell bei dem untersuchten Trigger-Event Read-Events propagiert werden, kann die Anzahl der Empfänger von Read-Events durch die weiteren Maskierungseffekte eingeschränkt werden. Read-Events werden vom gerade betrachteten Speicher demnach nur an Speicher am Beginn von empfindlichen Pfaden gesendet. Zusätzliche Empfindlichkeitsmarkierungen wie bisher immer beschrieben sind daher nicht notwendig, deren Funktion für das Rückwärts-Analyse-Verfahren wird durch die Read-Events vollständig dargestellt.

### 4.4.6 Berücksichtigung des zeitlichen Schaltungsverhaltens

Neben den bisher betrachteten rein digitalen Fehlermaskierungseffekten spielt auch schon wie in Kap. 2.2.3 beschrieben das zeitliche Verhalten der Schaltung eine Rolle. Alle Arten von Verzögerungszeiten (z.B. Leitungsverzögerung, Gatterverzögerung, $t_{C2Q}$) und funktionsspezifische Zeitkonstanten (z.B. $t_{Hold}$) können dabei herangezogen werden um die Zeitpunkte der generierten Events zu modifizieren.

Der Zeitpunkt der Write-Events hängt hauptsächlich vom Zeitpunkt des damit verbundenen Trigger-Events ab. Nach einem Trigger erscheint der neue gespeicherte Wert nicht unmittelbar am Ausgang, sondern in der Regel erst nach einer kleinen Verzögerungszeit. Diese Verzögerungszeit kann berücksichtigt werden, in dem das Write-Event um diese Zeitspanne zu einem späteren Zeitpunkt hin verschoben wird.

Ähnlich kann bei der Gatterverzögerungszeit verfahren werden, nur dass hier der Zeitpunkt der Read-Events modifiziert werden muss. Gemäß Abbildung 2.4 leitet sich das Ende der Lebensdauer eines gespeicherten Wertes auf der Quellen-Seite aus der Verzögerungszeit des betrachteten Signalpfades und dem Abtastzeitpunkt auf der Senken-Seite ab. Der Abtastzeitpunkt auf der Senken-Seite ist der Trigger-Zeitpunkt, dementsprechend müssen die durch diesen Trigger verursachten Read-Events um die Verzögerungszeit des Signalpfades in die Vergangenheit, d.h. zu früheren Zeitpunkten hin, verschoben werden. Zusätzlich kann an dieser Stelle im Falle eines Registers oder eines ähnlich arbeitenden Bauteils die Hold-Zeit berücksichtigt werden. Das Data-Eingangssignal eines Registers muss nämlich bis nach Ablauf der Hold-Zeit konstant bleiben, daher müssen die Read-

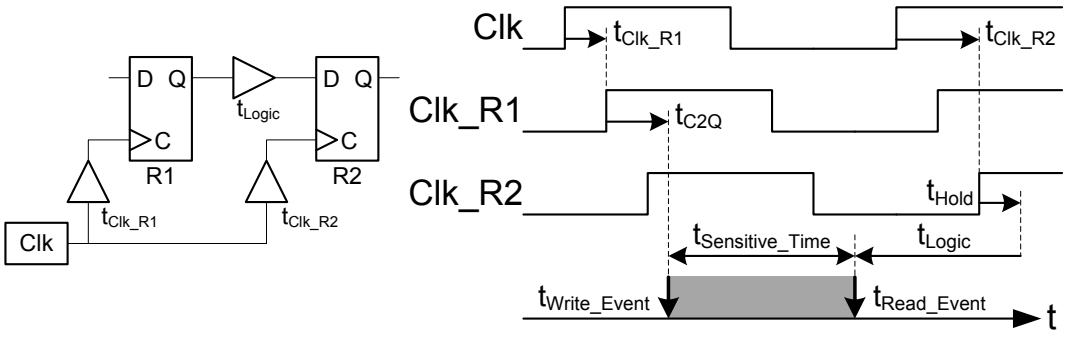



Abbildung 4.8: Modifikation der Zeitpunkte von Write- und Read-Events durch Timing-Parameter der Schaltung

$$t_{WriteEvent} = t_{TriggerEvent} + t_{Clk} + t_{C2Q} \quad (4.11)$$

$$t_{ReadEvent} = t_{TriggerEvent} + t_{Clk} + t_{Hold} - t_{Logic} \quad (4.12)$$

Verschiebung des Read- und Write-Events gegenüber dem Trigger-Event in Abhängigkeit schaltungsspezifischer Größen

Events wiederum um diese Zeitspanne zu späteren Zeitpunkten hin verschoben werden.

### 4.4.7 Zusammenfassung

Das vorgestellte Grundkonzept der Rückwärts-Analyse ermöglicht durch die Propagation von Read-Events die vier vorgestellten Maskierungseffekte (logische und transitive Maskierung, Informations-Lebensdauer und Verzögerungseffekte) elegant miteinander zu verknüpfen. Dadurch kann ein effizientes Verfahren zur Bestimmung kritischer Zeiträume von jeglichen Speichern in digitalen Schaltungen realisiert werden. Notwendige Komponenten für dieses Verfahren sind:

- Eine Datenstruktur zur Darstellung der Schaltungsstruktur (Schaltungsgraph) und des zeitlichen Verhaltens der Schaltung (Timing)
- Eine Beschreibung der grundlegenden Funktionalität aller in der Schaltung verwendeter Schaltungselemente
- Ein Mechanismus zur Stimulation der Schaltung mit Signalverläufen einer Simulation in zeitlich umgekehrter Reihenfolge
- Ein Mechanismus zur Erkennung und Auswertung von Änderungen an Schaltungssignalen (ähnlich zu einem Prozess in VHDL)
- Ein Mechanismus zur Bestimmung der empfindlichen Pfade anhand der gegebenen Schaltungsstruktur und dem Schaltungszustand
- Eine Datenstruktur zur Speicherung der Zwischenergebnisse (Read- und Write-Events) und der Endergebnisse (ermittelte empfindliche Zeiträume)

sowie ein Auswerte- und Bewertungsverfahren der Ergebnisse

### 4.4.8 Implementierungsplatform für das Verfahren

Die obige Auflistung zeigt deutlich, dass für die Durchführung der BA ein Schaltungssimulator mit stark erweiterter Funktionalität erforderlich ist. Für eine Implementierung des Verfahrens könnte nun ein eigenständiger Simulator entworfen werden, was aber mit enorm hohem Entwicklungsaufwand und höchstwahrscheinlich schlechterer Qualität des Endergebnisses verbunden wäre. In kommerziellen Schaltungssimulatoren steckt enorm viel finanzieller und personeller Aufwand, weitaus mehr, als in jedem Forschungsprojekt zu leisten möglich ist. Daher können sinnvolle Implementierungen des Verfahrens nur auf zwei Arten erfolgen:

- Verwendung eines Open-Source-Schaltungssimulators in Verbindung mit einer spezifischen Modifikation zur Implementierung der BA
- Verwendung eines kommerziellen Schaltungssimulators mit der Möglichkeit zur Einbindung von User-Spezifischer Funktionalität

Da die BA selbst schon wie vorgeschlagen aus zwei Teilaufgaben (Simulation und Analyse) besteht, wäre ein einzelner Schaltungssimulator also nicht ausreichend. Für die BA müssten dann also zwei unterschiedliche Varianten eines Simulators erstellt werden. Im Gegensatz dazu enthalten die Hardwarebeschreibungssprachen VHDL [53] und Verilog [54] in ihren neuesten Überarbeitungen Softwareschnittstellen zur Anbindung von eigener Funktionalität. Das VHDL Procedural Interface VHPI und Verilog Procedural Interface VPI bieten die Möglichkeit eigenen C/C++ Code in Interaktion mit dem Schaltungssimulator arbeiten zu lassen. Gegen eine Verwendung dieser Programmier-Schnittstellen sprechen eigentlich nur praktische Gründe. Zum Beginn dieser Arbeit und des ersten Konzepts der BA war diese Funktionalität in kommerziellen Simulations-Tools nicht oder nur teilweise implementiert. Aus diesem Grund wurde eine vom Funktionsumfang her sehr ähnliche, aber proprietäre Lösung gewählt. Der ebenfalls sehr weit verbreitete Schaltungssimulator ModelSim der Firma Mentor Graphics bietet mit dem Foreign Language Interface FLI eine ausgereifte Programmierschnittstelle in Verbindung mit VHDL als Hardware-Beschreibungssprache.

Alle drei Programmierschnittstellen (VHPI, VPI und FLI) bieten folgenden typischen Funktionsumfang:

- Auslesen von Schaltungssignalen, Einsetzen von Transaktionen zum Verändern von Schaltungssignalen
- Erkennung von Signaländerungen (ähnlich einem Prozess in VHDL) mit anschließender Ausführung benutzerdefinierter Funktionen
- Informationen über die Struktur und Hierarchie der simulierten Schaltung
- Verwendung weiterer Funktionen wie z.B. Zugriff auf Massenspeicher und Netzwerkverbindungen des Simulationsrechners, eigene grafische Benutzeroberflächen

Die BA wurde daher mit dem FLI unter Verwendung von C++ Programmcode implementiert. In allen weiteren Erklärungen wird aber, sofern möglich, auf Details und Spezialisierungen in direkter Verbindung zu FLI verzichtet. Das Verfahren selbst sollte prinzipiell mit jeder der zuvor benannten Möglichkeiten implementiert werden können.

## 4.5 Vergleiche zur Testmustergenerierung

Das Prinzip der Rückwärts-Analyse unterscheidet sich in einigen Punkten stark in den Ansätzen, die bei der Testmustergenerierung verfolgt werden. Die Testmustergenerierung bearbeitet zwei elementare Probleme, die Steuerbarkeit und die Beobachtbarkeit der Schaltung. Bei der Rückwärts-Analyse entfällt die Betrachtung der Steuerbarkeit durch die Verwendung von Signalverläufen, die durch eine Schaltungssimulation erzeugt wurden.

Dadurch entfällt auch der Rechenaufwand zur Festlegung von Implikationen auf Signalwerte und spekulative Signalzuweisungen wie bei der Testmustergenerierung [25] [55]. Bei hoher Schaltungskomplexität kann das Finden einzelner Testmuster für spezifische Fehler einen extremen Mehraufwand gegenüber dem Durchschnittsaufwand zur Bestimmung eines Testmusters bedeuten. Die Rückwärts-Analyse arbeitet dagegen mit gegebener Schaltungsbelegung, daher ist der Rechenaufwand weitestgehend unabhängig von der Komplexität der Schaltung.

Bei der Testmustergenerierung ist es im Allgemeinen jedoch ausreichend, wenn für jeden möglichen Fehler mindestens ein Testmuster gefunden wird. Primäres Ziel sind daher eher wenige Testmuster mit möglichst hoher Testabdeckung und damit verbunden eine möglichst kurze Testzeit der Schaltung. Die Ergebnisse der Rückwärtsanalyse haben im Ergebnisraum eine weitere Dimension, die Zeit. Es werden daher als Ergebnis alle Kombinationen von Fehlerort und Zeitpunkt in Betracht gezogen. Die zeitliche Länge des betrachteten Anwendungsfalls wirkt sich daher direkt auf den Rechenaufwand aus.

Logische Maskierung in der Schaltung erschwert das Finden eines Testmusters für einen Fehler im Allgemeinen. Die Maskierung verringert sowohl Steuerbarkeit und Beobachtbarkeit von Fehlern, dadurch verringert sich die Anzahl der möglichen Signalkombinationen für einen spezifischen Test. Logische Maskierung ist für die Durchführung und die Ergebnisse der Rückwärts-Analyse jedoch vorteilhaft. Zum Einen verringert sich die Anzahl der empfindlichen Zeiträume der Schaltung, zum Anderen wird die Anzahl der zu propagierenden Read-Events reduziert, was einen schnelleren Ablauf der Analyse zur Folge hat.

## 4.6 Sonderfälle im Schaltungsdesign

In den bisherigen Überlegungen zur Rückwärts-Analyse wurde von stets rein digital arbeitenden Schaltungen ausgegangen, welche zudem auch keine kritischen Strukturen hinsichtlich des Schaltungstests enthalten. Im folgenden Abschnitt sollen die am häufigsten auftretenden Problemfälle erläutert werden und Lösungsvorschläge gezeigt werden.

### 4.6.1 Schaltungen mit mehreren Taktsignalen

In größeren Schaltungen wird häufig mehr als ein globales Taktsignal verwendet. Die verwendeten Taktsignale können sich dabei in ihrer Frequenz als auch der Phasenlage unterscheiden. Dadurch können keine feststehenden Beziehungen zwischen den Trigger-Events der einzelnen Speicher ermittelt werden. Da bei der Rückwärts-Analyse die Trigger-Events durch die Signalverläufe der Schaltung bestimmt und ausgelöst werden, ist eine statische Bestimmung der Trigger-Zeitpunkte nicht notwendig.

Unter Umständen ist eine solche statische Bestimmung der Trigger-Zeitpunkte nicht möglich, da die Frequenz und Phasenlage eines Taktsignals sich während des Betriebs auch verändern kann. Als Beispiel hierfür können Schaltungen mit variabler Taktfrequenz herangezogen werden, die häufig eingesetzt werden um die dynamische Verlustleistung zu reduzieren. Um die Zeitpunkte der Taktflanken indirekt bestimmen zu können, müssten wiederum andere Schaltungssignale unter Kenntnis der Schaltungsfunktionalität ausgewertet werden. Hierfür wäre eine schaltungsspezifische Funktion notwendig, was eine deutliche Abkehr vom bisher verfolgten allgemeinen Konzept darstellen würde.

Generell wäre solch ein Schema bestenfalls geeignet, innerhalb einer synchronen Taktdomäne die Trigger-Events zu bestimmen. Beim Übergang zwischen verschiedenen Taktdomänen müsste deren gegenseitige Abhängigkeit bestimmt werden, was wiederum tiefere Kenntnis der Takterzeugung erfordern würde. Der Ansatz der Rückwärts-Analyse bietet hier den großen Vorteil die Simulationsergebnisse wiederzuverwenden. Werden in der Simulationsphase zur Erzeugung der Signalverläufe die unterschiedlichen Schaltungstaktsignale richtig bestimmt, so sind folgerichtig auch die Ergebnisse der Rückwärts-Analyse richtig. Wie in Abbildung 4.6 und Abbildung 4.7 dargestellt, werden auch asynchrone Steuersignale zum Triggern von Speichern verwendet. Daher stellt das Szenario mit mehreren Taktsignalen kein Problem für das Konzept der Rückwärts-Analyse dar.

### 4.6.2 Rekonvergente Pfade

Rekonvergente Pfade, d.h. Signalpfade, die ausgehend von einem Punkt in der Schaltung sich in parallel verlaufende Pfade aufteilen und später wieder an einem Gatter vereinigen (konvergieren), stellen in der Testmustergenerierung häufig ein

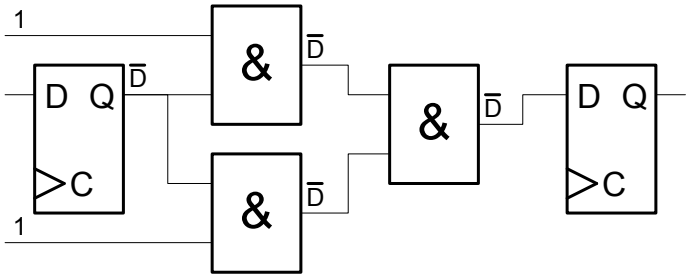



Abbildung 4.9: Mehrfachstimulation eines Gatters am Ende eines rekonvergenten Pfades durch einen Einzelfehler

Problem dar. Die Probleme entstehen, weil sich nach Änderung eines einzelnen Signals am Anfang des rekonvergenten Pfades mehrere Eingangssignale an dem Gatter am Ende des rekonvergenten Pfades ändern können. Dies wiederum erschwert die Signalzuweisung bei der Testmusterbestimmung, da einzelne Signale nicht mehr unabhängig steuerbar sind [56].

Durch die Tatsache, dass sich zwei oder mehr Signale an den Eingängen eines Gatters auf Grund einer einzelnen Signaländerung am Beginn des rekonvergenten Pfades ändern, können verschiedene Effekte auftreten. Ein angenommener Einzelfehler kann sich demnach als Mehrfachfehler an dem Rekonvergenz-Gatter manifestieren (Abbildung 4.9). Zur Abdeckung der Auswirkungen von Mehrfachfehlern muss eine explizite Bestimmung der Mehrfachfehler-Empfindlichkeit durchgeführt werden. Diese unterscheidet sich im Vorgehen nicht prinzipiell vom Ansatz bei Einzelfehlern, die gewonnenen Ergebnisse beziehen sich jedoch immer auf eine Kombination von fehlerbehafteten Eingängen. Aus der Wertetabelle der betrachteten Funktion am Ende des rekonvergenten Pfades kann natürlich der Wert des Ausgangssignals für den fehlerfreien und fehlerbehafteten Fall direkt bestimmt werden und damit auch die Empfindlichkeit durch den Vergleich der beiden Ausgangswerte. Allein durch die Bestimmung der Einzelfehler-Empfindlichkeit kann keine gültige Aussage über die Mehrfachfehler-Empfindlichkeit getroffen werden.

In einem zweiten Szenario kann es bei gleichzeitiger Änderung mehrerer Eingangssignale, davon mindestens eines empfindlichen Signals, zu einer Auslöschung kommen (Abbildung 4.10). Demnach wäre ein Gatter in einer solchen Situation bei einem Mehrfachfehler unempfindlich, jedoch empfindlich bei Einzelfehlern.

In Abbildung 4.9 und 4.10 wurden nur vereinfachte Fälle von Einzelfehlern mit Mehrfachausbreitung und der dadurch resultierenden Mehrfachstimulation oder Selbstmaskierung dargestellt. In einer verallgemeinerten Version ist es nicht notwendig, dass der rekonvergente Pfad nur innerhalb einer kombinatorischen Stufe einer sequentiellen Schaltung verläuft. Der rekonvergente Pfad kann dabei über beliebig viele sequentielle Stufen verlaufen, einzig die fehlerhaften Signale müssen zum gleichen Zeitpunkt am Rekonvergenz-Gatter anliegen. Damit ist die Erkennung von rekonvergenten Pfaden kein rein statisches Problem mehr, sondern

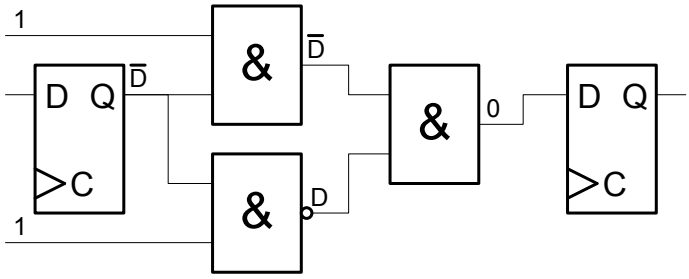



Abbildung 4.10: Selbstmaskierung eines Einzelfehlers mit Mehrfachausbreitung am Ende eines rekonvergenten Pfades

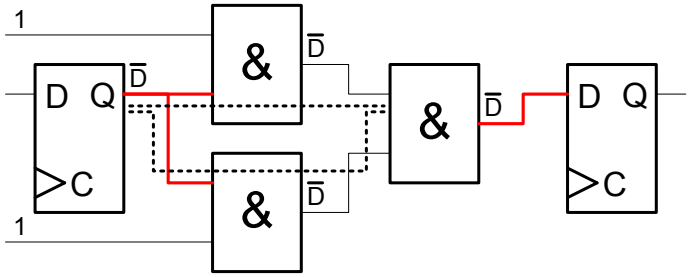



Abbildung 4.11: Empfindliche (rot) und unempfindliche Pfade (schwarz) als Teil eines rekonvergenten Pfades mit Mehrfachstimulation, resultierender empfindlicher Pfad (gestrichelt)

hängt dynamisch vom Schaltungsverhalten ab.

Würden die rekonvergenten Pfade in der Rückwärts-Analyse nicht berücksichtigt, so müsste mit einer Verfälschung der Ergebnisse gerechnet werden. Bei Betrachtung eines rekonvergenten Pfades mit Mehrfachstimulation besteht ein empfindlicher Pfad vom Anfang des Pfades bis zu dessen Ende, obwohl die parallelen Teilpfade nicht durchgängig als empfindlich betrachtet werden.

In Abbildung 4.11 werden die Teilabschnitte eines rekonvergenten Pfades mit Mehrfachstimulation gezeigt. In jedem der parallelen Pfade ist am Rekonvergenz-Gatter zunächst ein unempfindlicher Teilabschnitt. Der Fehler kann durch das Gatter nur propagieren, weil mehr als ein Fehler gleichzeitig anliegt. In den restlichen Teilen der parallelen Teilpfade müssen alle Teilabschnitte empfindlich sein, damit das fehlerhafte Signal auf diesen Pfaden bis zum Rekonvergenz-Gatter propagieren kann. Wäre ein weiteres Teilstück in einem der parallelen Pfade zusätzlich unempfindlich, so würde auf Grund dieses unempfindlichen Teilstücks das fehlerhafte Signal eben nicht bis zum Rekonvergenz-Gatter propagiert werden. Damit wäre auch die Mehrfachstimulation nicht zutreffend.

Für einen rekonvergenten Pfad mit Selbstmaskierung sind ähnliche Voraussetzungen gegeben. Damit es zu Selbstmaskierung kommen kann, muss mindestens ein Gattereingang als empfindlich betrachtet werden. Die Auswirkungen dieses

| A | B | C | Q | $\frac{\partial Q}{\partial A}$ | $\frac{\partial Q}{\partial B}$ | $\frac{\partial Q}{\partial C}$ | $\frac{\partial Q}{\partial A \partial B}$ | $\frac{\partial Q}{\partial A \partial C}$ | $\frac{\partial Q}{\partial B \partial C}$ | $\frac{\partial Q}{\partial A \partial B \partial C}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Tabelle 4.1: Wertetabelle und Boolesche Empfindlichkeit gegenüber Einzel- und Mehrfachfehlern an einem UND-Gatter mit drei Eingängen

fehlerbehafteten Eingangs werden dann durch einen oder mehrere weitere unempfindliche Eingänge wieder aufgehoben. Die Selbstmaskierung muss daher an dem Rekonvergenz-Gatter für alle Kombinationen von jeweils mindestens einem empfindlichen und einem unempfindlichen Eingang betrachtet werden. Alle Teilstücke der parallel verlaufenden Teilpfade, mit Ausnahme der Teilstücke unmittelbar am Rekonvergenz-Gatter, müssen ebenfalls empfindlich sein, damit die fehlerhaften Signale das Rekonvergenz-Gatter erreichen könnten.

Bei kombinatorischen Gattern mit mehreren Eingängen kann es häufig vorkommen, dass für bestimmte Belegungen sowohl empfindliche als auch unempfindliche Eingänge an einem Gatter vorhanden sind. Mehrfachstimulation kann nur an Kombinationen von zwei oder mehr unempfindlichen Eingängen auftreten, für Selbstmaskierung sind Kombinationen von sowohl unempfindlichen und empfindlichen Eingängen notwendig.

In Tabelle 4.1 werden beispielhaft die möglichen Fälle für ein Und-Gatter mit drei Eingängen dargestellt. Es sind dort jeweils zwei Belegungen vorhanden, in denen ein Einzel-, Doppel- oder Dreifachfehler zu einer Änderung des Ausgangssignals führen könnte. Mehrfachfehler, die durch rekonvergente Pfade verursacht werden, müssen jedoch nur für die Belegungen berücksichtigt werden, in denen keiner der beteiligten Einzelfehler als empfindlich betrachtet wird. Demnach bleiben nur noch jeweils eine Belegung für die Doppel- und Dreifachfehler übrig, die in rekonvergenten Pfaden wirksam werden könnten.

Zur Behandlung der rekonvergenten Pfade soll auf die eben erläuterten Regeln aufgebaut werden. Demnach ist eine Ausnahme von den Propagationsregeln für Read-Events nur in den Teilabschnitten des rekonvergenten Pfades unmittelbar am Rekonvergenz-Gatter notwendig. Zusätzlich muss aber festgestellt werden, ob die rekonvergenten Signale von der gleichen Quelle und vom gleichen gespeicherten Wert stammen. Dies kann mit einem spekulativen Ansatz erreicht werden.

Zur Erkennung einer Mehrfachstimulation auf Grund eines rekonvergenten Pfades werden daher an allen kombinatorischen Gattern zuerst die Eingänge mit

Einzelfehler-Empfindlichkeit bestimmt. Daraus kann nun auch die Menge der gegenüber Einzelfehlern unempfindlichen Eingänge einfach bestimmt werden. Mit dieser Signalmenge werden nun alle möglichen und wirksamen Kombinationen von Eingängen für eine Mehrfachstimulation bestimmt. Über alle Eingänge mit Einzelfehler-Empfindlichkeit werden nun regulär die Read-Events weiter propagiert. Zusätzlich wird für jede identifizierte Kombination von Gattereingängen mit möglicher Mehrfachstimulation eine eindeutige Identifikationsnummer (ID) generiert. Die ID wird auf den in dieser Kombination enthaltenen Eingängen zusammen mit dem Read-Event weiterpropagiert. Diese Read-Events im Zusammenhang mit einer ID werden an der nächsten erreichten Speicherzelle zunächst in einer separaten Liste gespeichert. Im Falle eines echten rekonvergenten Pfades wird ein weiteres Read-Event mit der selben ID an dieser Speicherzelle eintreffen und dort ebenfalls gespeichert.

Die Read-Events werden in gewohnter Weise beim nächsten Write-Event ausgewertet. In der Liste der Read-Events mit ID wird nun nach mindestens zwei Read-Events mit der gleichen ID gesucht. Wird eine solche Übereinstimmung gefunden, ist ein rekonvergenter Pfad gefunden, der zu einer Mehrfachstimulation führen könnte. Aus den beteiligten Read-Events mit ID wird auf Grund deren Zeitstempel ein reguläres Read-Event für diese Speicherzelle generiert, das auch im Zusammenhang mit dem Write-Event ausgewertet wird. Falls an dieser Speicherzelle bereits ein einfaches Read-Event passend zu dem Write-Event vorhanden ist, so ist der Inhalt der Speicherzelle auf jeden Fall relevant. Durch das Erkennen der gleichen IDs wird sichergestellt, dass an diesem Gatter der Beginn eines rekonvergenten Pfades war.

Wenn durch einen der beiden Mechanismen, reguläre Read-Events oder Read-Events mit ID, eine Speicherzelle als relevant betrachtet wird, so führt dies zu einer Überdeckung aller anderer Read-Events mit ID, d.h. wird eine Speicherzelle als relevant betrachtet, so spielen andere mögliche rekonvergente Pfade für Mehrfachstimulation keine Rolle mehr (Abbildung 4.12). Alle anderen Read-Events mit ID und einem Zeitstempel größer als der Write-Event-Zeitpunkt können daher gelöscht werden.

Wenn an einer Speicherzelle zwar ein Trigger-Event auftritt und Read-Events mit ID vorhanden sind, jedoch eine Auswertung aller Read-Events zu keinem empfindlichen Zeitbereich führt, so dürfen die Read-Events mit ID nicht gelöscht werden, sondern müssen mit neuem Zeitstempel weiterpropagiert werden.

Für mehrfach verschachtelte rekonvergente Pfade muss dieses Schema nochmals erweitert werden. Wenn beim Verfolgen eines Teilpfades eines vermuteten rekonvergenten Pfades wiederum an einem Gatter eine mögliche Mehrfachstimulation durch einen rekonvergenten Pfad auftreten könnte, so muss dies berücksichtigt werden. Das eingehende Read-Event mit ID wird daher an weitere übergeordnete Read-Events mit ID angehängt und wird beim Eintreffen an einer Speicherzelle erst berücksichtigt, wenn die übergeordneten Read-Events wieder zusammengetroffen sind.

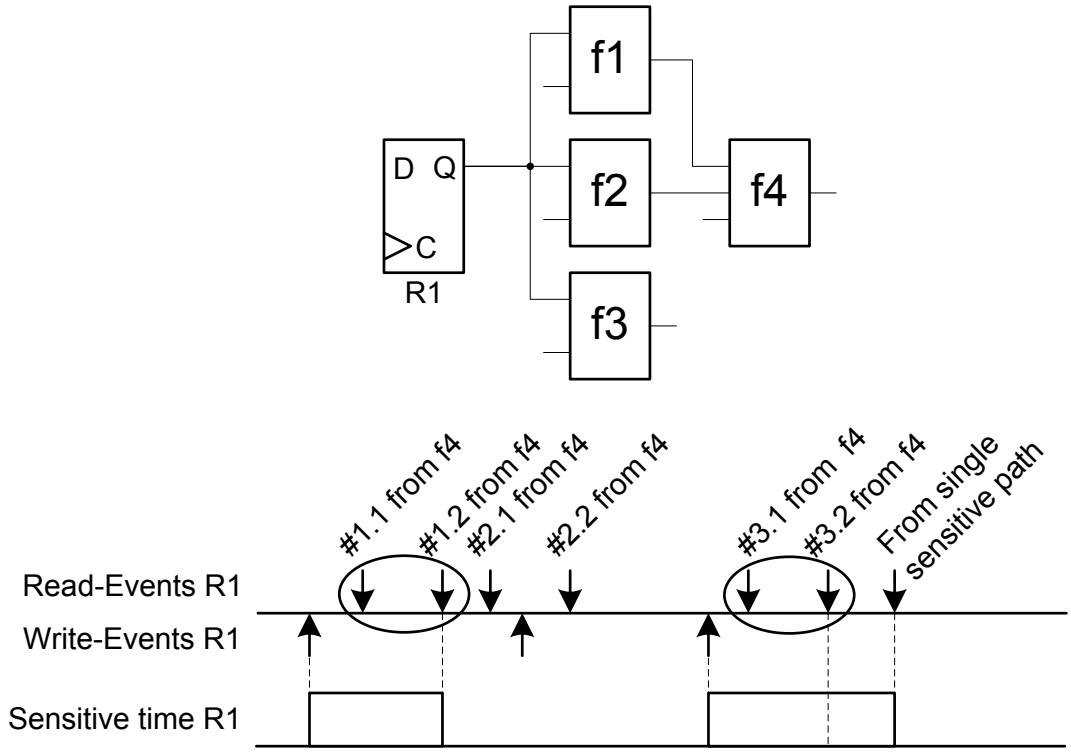



Abbildung 4.12: Auswertung von Read-Events sowohl aus rekonvergenten Pfaden als auch aus Einzelfehler-Pfaden, resultierende empfindliche Zeiträume

Die Betrachtung von Selbstmaskierung könnte zu einer weiteren Verringerung der ermittelten Empfindlichkeit der Schaltung führen. Zusammen mit der spekulativen Erkennung der Mehrfachstimulation würde sich eine sehr große Menge an Read-Events ergeben, die in den meisten Fällen keine Auswirkung auf das Analyse-Ergebnis haben. Da als Grundvoraussetzung eine Unterabschätzung der Schaltungsempfindlichkeit ausgeschlossen werden soll, kann auf die Bewertung der Mehrfachstimulation nicht verzichtet werden. Aus Effizienzgründen und auf Grund des erwarteten geringen Einflusses wird die Selbstmaskierung nicht weiter betrachtet.

### 4.6.3 Tri-State und mehrere Signaltreiber

Eine weitere Abweichung von einfachen digitalen Schaltungen stellt die Verwendung von Tri-State-Treibern für interne Schaltungsknoten dar. Der Einsatz von Tri-State-Treibern an primären Ein- und Ausgängen stellt ein Standardverfahren dar, um bidirektionale Datenverbindungen realisieren zu können. Die an diese Ein- und Ausgänge verbundenen Schaltungsteile sind dann in der Regel aber für jede Datenrichtung unabhängig voneinander ausgeführt. Bei schaltungsinter-

nen Tri-State-Treibern entstehen die Probleme durch die Tatsache, dass mehrere Treiber für ein Signal vorhanden sind. Dementsprechend kann für die Aufgabenstellung der Rückwärts-Analyse nicht eindeutig die Fehlerausbreitung bestimmt werden, da ein fehlerhaftes Signal aus mehreren Quellen getrieben werden kann.

Der Ansatz mehrerer Treiber für interne Signale wird im Allgemeinen nur bei nicht-CMOS-Schaltungstechnologie eingesetzt. In diesen Technologien (z.B. TTL oder NMOS) lassen sich damit einfache Logikgatter mit sehr vielen Eingängen realisieren. Diese Technologien finden zwar heute üblicherweise keine Anwendung in der Hochintegration von digitalen Schaltungen, jedoch können sie als Bestandteil von Schaltungen bestehend aus mehreren ICs immer noch eingesetzt werden.

Für eine Analyse der Fehlerausbreitung ist es daher notwendig eine passende Graphenstruktur zu haben, die es erlaubt mehrere Treiber für ein Signal zu spezifizieren und auch den aktiven Treiber bestimmen zu können. Durch Spezifikation der Logiktabellen der Gatterfunktionen mit Ausgangssignalwerten in 9-wertiger IEEE1164-Logik [57] kann dies erreicht werden. Dazu muss der Wert des Ausgangssignals auf zwei Weisen bestimmt werden. Durch Auswertung der Eingangssignale des Gatters kann mit der Wertetabelle der theoretische Ausgangswert bestimmt werden. Dieser wird dann mit dem tatsächlichen Ausgangswert, der ja hier von mehreren Treibern beeinflusst werden kann, verglichen. Dadurch kann wiederum bestimmt werden, welche Eingangssignale des Treibers sich im Fehlerfall auf das Ausgangssignal auswirken könnten.

Ein alternativer Ansatz zur Behandlung des Spezialfalls mehrerer Treiber für ein einzelnes Signal wäre ein High-Level-Analyse der Schaltung. Dabei könnten die treibenden Gatter und die betroffenen Leitungen in eine oder mehrere angepasste Logikfunktionen umgewandelt werden um wieder eine klassische digitale Schaltung zu erhalten. Allerdings stellt dies einen komplexen Vorgang dar und die zusätzlichen Probleme bei der Umsetzung des Zeitverhaltens der originalen Schaltung wären nur schwer automatisierbar zu handhaben.

## 4.7 Schwachstellen und Grenzbereiche der Anwendung

Wie im vorherigen Abschnitt beschrieben, können Tri-State-Treiber zu Problemen bei der Analyse führen bzw. erfordern einen angepassten Algorithmus. Die Tri-State-Treiber sind aber nur ein Beispiel für andere analoge Schaltungsfunktionen die häufig in eine digitale Schaltungssimulation integriert werden. Da die Rückwärts-Analyse auf einer deterministisch bestimmbaren Fehlerausbreitung beruht, muss für jede enthaltene analoge Funktion eine digitale Beschreibung des Verhaltens vorhanden sein. Gerade bei Komponenten zur Taktgenerierung ist häufig aber ein zeitabhängiges Verhalten in Verbindung mit einem integrierten Zustandsautomaten dieser Komponenten zu beobachten.

Für eine Analyse solcher Komponenten ist es aber dann zwingend notwendig, zum Einen den Zustand der Komponente dem Algorithmus zugänglich zu machen, zum Anderen für jeden dieser Zustände eine Beschreibung des E/A-Verhaltens anzubieten. Die Verfügbarkeit des Zustandes stellt im Allgemeinen aber ein Problem dar, da dieser in der Regel nicht als Signal aus der Komponente herausgeführt wird und demnach auch nicht aufgezeichnet wird. Weiterhin kann der Zustand einer solchen Komponente evtl. durch mehrere Signale/Variablen dargestellt sein oder nur durch den aktiven Prozess gekennzeichnet sein. Bei letzterer Variante besteht ohne Änderung der Funktionsbeschreibung der Komponente keine Möglichkeit zu Analyse.

Generell lässt sich aus diesem Problem folgern, dass alle Schaltungsbestandteile, die sich nicht in einfachste Teilblöcke zerlegen lassen oder für die kein komplexes Fehlerausbreitungsmodell vorhanden ist, mit der Rückwärts-Analyse nicht analysierbar sind. Als Abhilfe kann aber für diese Blöcke ein vereinfachtes Modell herangezogen werden. Unter einem zugrundeliegenden konservativen Ansatz kann die Annahme getroffen werden, dass sich Fehler an jedem der Eingänge immer auf alle Ausgänge auswirken. Detailliertere Kenntnisse über die Funktion des komplexen Schaltungsblocks können zur Spezifikation der Verzögerungszeit verwendet werden, dabei ist auch z.B. eine taktgesteuerte Verzögerung modellierbar wenn eine sequentielle Verzögerung vorhanden ist.

Diese vereinfachten Modelle erlauben es auch komplexe Schaltungsblöcke in die Analyse mit einzubeziehen, was aber mit einem Abstrich an Genauigkeit verbunden ist. Wie hoch dieser Genauigkeitsverlust ist, hängt von dem betrachteten Modell, der gesamten Schaltung und dem Anwendungsfall der Schaltung ab, so dass darüber keine pauschale Aussage getroffen werden kann. Der konservative Ansatz soll aber ein Unterschätzen der Schaltungsempfindlichkeit verhindern.

Eine potentielle Quelle für eine Überabschätzung der Empfindlichkeit entsteht, wenn Schaltungen eine Ausgangslogik besitzen, d.h. wenn zwischen Ausgangsregistern und Ausgangsports noch Logikgatter vorhanden sind. Generell ist keine Information vorhanden, wann die Ausgangsports von der Umwelt "gelesen" werden. Daher wurde der konservative Ansatz gewählt, die Ausgangsregister immer als relevant zu betrachten. Wenn jedoch die Ausgangslogik genau eben die Beobachtbarkeit der Ausgangsregister für einen Zeitraum verhindert, so wäre der vorherige Ansatz zumindest teilweise falsch. Die unnötig als empfindlich betrachteten Ausgangsregister würden ihrerseits wiederum Read-Events aussenden, was eine Fortpflanzung dieses Fehlers ermöglicht. Typische Anwendungen für Ausgangslogik sind Ausgangsmultiplexer oder bidirektionale Ports, die durch Tri-State-Treiber realisiert werden. Für eine genauere Betrachtung müssten alle Signale der Ausgangsregister und der Ausgangslogik überwacht werden, sowie ein Mechanismus zum Generieren von Read-Events direkt in den Ausgangsports hinzugefügt werden.

Ein weiteres Problem kann eine starke Verlangsamung der Analysegeschwindigkeit sein. Bei der Analyse der empfindlichen Zeiträume unter Berücksichtigung

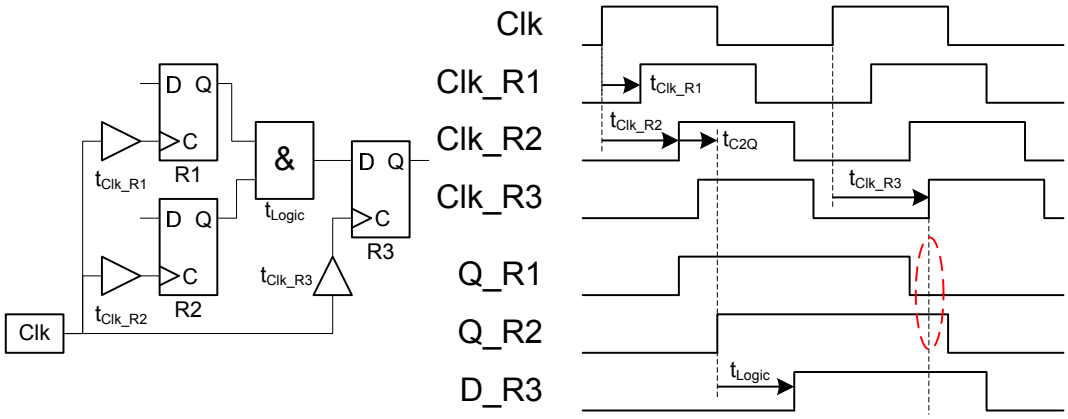



Abbildung 4.13: Auswirkung von Verzögerungszeiten auf das Analyse-Ergebnis

der rekonvergenten Pfade kann es evtl. zu einer "Überschwemmung" mit IDs für rekonvergente Pfade kommen. Je nach Schaltungsstruktur und Menge der empfindlichen Pfade werden pro Taktzyklus unterschiedliche viele IDs generiert. Da ein Löschen der IDs nur an unempfindlichen Pfaden stattfindet, kann die Anzahl und Lebensdauer dieser IDs sehr groß werden. Durch die Abhängigkeit von der Leistungsfähigkeit des Rechners, auf dem der Algorithmus ausgeführt wird, kommt es durch diesen Effekt früher oder später zu einem Einbruch der Performance.

Ein weiteres Problem könnte zu systematisch falschen Ergebnissen führen. Bisher wurde immer davon ausgegangen, dass die Rückwärts-Analyse unter allen Umständen richtig arbeitet. Da das Analyse-Ergebnis direkt vom Schaltungszustand abhängig ist, soll an dieser Stelle der zeitliche Einfluss genauer betrachtet werden. Für einen Logikpfad zwischen zwei speichernden Elementen sind die in Abbildung 4.13 vorgestellten Verzögerungszeiten zu berücksichtigen.

Zwar wird der Zustand der Simulation fortwährend durch Schaltungsstimulation mitgeführt, jedoch werden die Berechnungen für die empfindlichen Pfade alle unmittelbar zum Triggerzeitpunkt ausgeführt. Basis für diese Berechnungen bilden natürlich auch die momentanen Signalwerte. In Abbildung 4.13 kann erkannt werden, dass zu dem markierten Triggerzeitpunkt von R3 der Wert des Eingangssignals D zwar den Wert 1 hat, jedoch die Eingangswerte des UND-Gatters zu einem Ausgangssignal von 0 führen würden. Unter Berücksichtigung der Verzögerungszeiten ist diese Signalkombination erklärbar, dennoch führt sie zu einem falschen Ergebnis bei der Rückwärts-Analyse. Aus den Eingangssignalwerten des UND-Gatters wird nur ein einzelner empfindlicher Pfad zu R1 bestimmt. Würden Eingangssignalwerte verwendet, die zum Ausgangssignalwert des UND-Gatters passen, so würden beide Pfade zu R1 und R2 als empfindlich betrachtet werden. Diese falsch berechneten empfindlichen Pfade führen wiederum zu einer falschen Propagation von Read-Events und damit zwangsweise zu

einem inkorrekten Analyse-Ergebnis.

Dieses Problem rührt von der Tatsache her, dass sich Quellen-Signale in einem kombinatorischen Logikbaum bereits wieder ändern können, bevor das Senken-Register den Ausgangswert gespeichert hat. Bei einer einfachen Schaltungssimulation kann ein ähnliches Phänomen auftreten. Wenn die Verzögerungszeit eines Signalpfades kürzer als der positive Clock-Skew dieses Pfades ist, kann es zu einer Hold-Time-Verletzung kommen. Im schlimmsten Fall ist die Verzögerungszeit des Taktpfades des Senken-Registers größer als die Summe der Clock-2-Q- und der Gatterverzögerungszeit, so dass mit einer Taktflanke ein Signal auf einem sequentiellen Pfad um zwei Stufen weitergereicht wird. Im übertragenen Sinn für die Rückwärts-Analyse soll dieses Problem daher Pseudo-Hold-Time-Violation genannt werden. Für eine korrekt ablaufende Rückwärts-Analyse leitet sich unter Berücksichtigung der Pseudo-Hold-Time-Violation folgende notwendige Bedingung ab:

$$t_{Skew\ Dest,Source} < t_{C2Q} \tag{4.13}$$

In einer Schaltung kann daher bei allen sequentiellen Stufen mit negativem Clock-Skew, also einer verkürzten effektiven Taktperiode, davon ausgegangen werden, bei der Rückwärts-Analyse keine Probleme zu verursachen. Weiterhin unkritisch sind alle sequentiellen Pfade, bei denen Clock-to-Q größer als der Clock-Skew ist. Lediglich der Teil der sequentiellen Pfade bei denen Clock-to-Q kleiner als der Clock-Skew ist, führt zu falschen Analyseergebnissen selbst wenn die Schaltung korrekt funktioniert, d.h. keine der normalen Timing-Bedingungen verletzt. Ohne weitere Verbesserungen können Schaltungen mit Pseudo-Hold-Time-Violation mit der Rückwärts-Analyse nicht untersucht werden, da eine richtige Berechnung der empfindlichen Pfade vorausgesetzt wird.

## 4.8 Ausgabe von Signalverläufen in zeitlich umgekehrter Reihenfolge

Aufzeichnungsdateien für Signalverläufe können nach einem von zwei möglichen Prinzipien hinsichtlich der Sortierung der enthalten Signale aufgebaut sein. In einer Datei können alle Zustandswechsel eines einzelnen Signals zu einem zusammenhängenden Datenblock zusammengefasst werden. Dies bietet den Vorteil, dass der Signalverlauf eines einzelnen Signals sehr schnell dargestellt werden kann. Für den Aufbau einer einzelnen Datei mit mehreren Signalen entsteht aber der Nachteil, dass bis zum Ende der Simulation noch Signalübergänge stattfinden können und deshalb die Datei nicht oder nur mit Einschränkungen geschrieben werden kann, da die endgültige Position der Datenblöcke zueinander nicht festgelegt werden kann oder mit Lücken gerechnet werden muss. Die zweite Möglichkeit zur Aufzeichnung der Signalverläufe verwendet eine chronologische Ordnung der

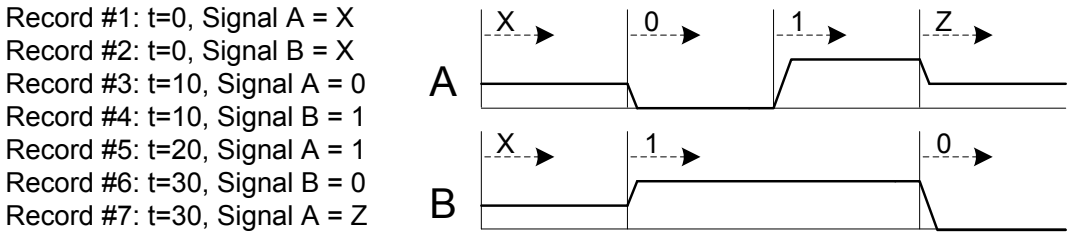



Abbildung 4.14: Darstellung des vereinfachten Aufzeichnungsformats für Signalübergänge in WLF- und VCD-Dateien sowie zugehöriger Signalverlauf

Signalübergänge, gespeichert wird dabei der Zeitpunkt selbst und der Wert des Signals nach diesem Zeitpunkt. Mit fortschreitender Simulation können daher alle Signalübergänge bereits abschließend in eine Datei auf einem Massenspeicher geschrieben werden. Dies verringert den Speicherbedarf (RAM) während der Simulation deutlich. Auch das nachträgliche Anzeigen eines oder mehrerer Signale für einen festgelegten Zeitraum lässt sich schnell erledigen. Lediglich die Suche nach dem nächstfrüheren oder -späteren Signalübergang des gleichen Signals ist ein Vorgang der nur über eine lineare Suche in der Datei erledigt werden kann.

In industriellen Anwendungen finden hauptsächlich zwei Dateiformate nach dem zweiten Prinzip Anwendung. Die einfachen Value Change Dump VCD Dateien [58] sind als unkomprimierte, human-readable Textdateien gestaltet. Die proprietären WLF-Dateien werden ausschließlich von Produkten der Firma Mentor Graphics verwendet.

Ein komplexes Problem entsteht aber durch die Aufgabe, den Signalverlauf der ursprünglichen Schaltung in zeitlich umgekehrter Reihenfolge ablaufen zu lassen. Weil in WLF- und VCD-Dateien nur der Wert des Signals nach dem spezifizierten Zeitpunkt gespeichert ist, kann bei einer mehrwertigen Logik, d.h. bei prinzipiell jeder Logik mit mehr als zwei Signalzuständen, nicht eindeutig auf den Wert vor dem Signalübergang geschlossen werden (Abbildung 4.14). In Folge dessen, um einen eindeutigen Wert des Signals zu erhalten, muss in der Datei die nächstfrühere Transition des gleichen Signals gesucht werden, welches den Signalwert vor der zuletzt betrachteten Transition festlegt. Da mit dem hier verwendeten Dateiformat dieser Vorgang nicht effizient unterstützt wird, findet zwangsweise eine lineare Suche statt. Die Transitionen werden daher blockweise eingelesen und in einem Zwischenpuffer sowohl in einer zeitlich sortierten Liste als auch in einer weiteren Liste abgelegt, die primär nach Signalen und erst sekundär zeitlich sortiert ist. Dies erfordert zwar mehr Speicher als unbedingt notwendig, bietet aber bei den folgenden Schritten einen Geschwindigkeitsvorteil.

Eine wichtige notwendige Größe zur zeitlichen Invertierung eines Signalverlaufs ist die Gesamtlänge der Signalaufzeichnungen. Da im Analyse-Teil der BA der Simulator regulär funktioniert und deshalb auch dessen Simulationszeit nur in

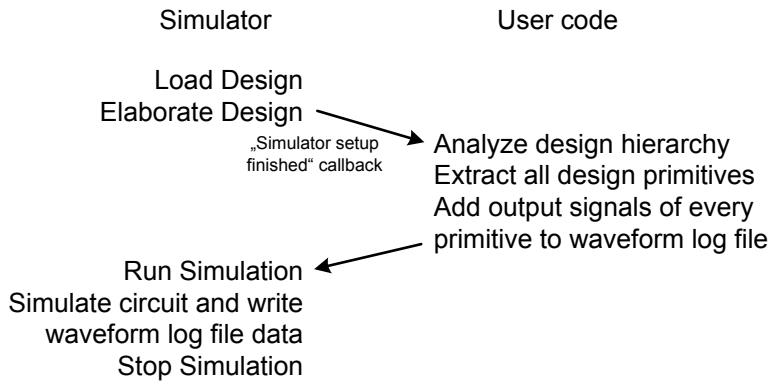



Abbildung 4.15: Software-Ablauf des Simulationsteils der Rückwärts-Analyse

positiver Richtung abläuft, hier aber ein Signalverlauf in zeitlich umgekehrter Reihenfolge wiedergegeben werden soll, muss eine Konvertierung der Transition-Zeitpunkte vorgenommen werden. Durch die einfache Gleichung

$$t_{Transition,Reverse} = T_{Total} - t_{Transition,Forward} \tag{4.14}$$

kann der Zeitpunkt für eine Transition im Analyse-Teil bestimmt werden.

Zur Einspeisung der Signalübergänge in die Schaltung muss nur die zeitlich sortierte Liste der Transitionen abgearbeitet werden. Für jede Transition muss der Zeitpunkt auf die Zeitachse der Analyse umgerechnet werden. Für diesen Zeitpunkt wird der Signalwert der nächstfrüheren Transition eingespeist, welcher mit nur geringem Aufwand aus der für jedes Signal sortierten Liste entnommen werden kann. Nachdem eine Transition verarbeitet wurde, kann diese aus beiden Listen gelöscht werden um den benötigten Speicher wieder freizugeben. Falls die nach Signalen sortierte Liste nur einen einzelnen Eintrag enthält, muss die Suche nach einer früheren Transition gestartet werden. Im Extremfall stoppt die Suche erst, wenn der Initialisierungswert des Signals am Beginn der Aufzeichnungen gefunden wurde. Sind alle Transitionen an einem Simulationszeitpunkt verarbeitet, wird aus der zeitlich sortierten Liste der Zeitpunkt der nächsten Transition entnommen. Für die Dauer vom jetzigen Simulationszeitpunkt bis zur nächsten Transition kann die Steuerung wieder an den Simulator zurückgegeben werden.

## 4.9 Software-Ablauf

In den Abbildungen 4.15 und 4.16 ist der Software-Ablauf im Schaltungssimulator und dem zugehörigen FLI-Programm dargestellt.

Der Simulationsteil erfordert nur einfach Interaktion des BA-Programms mit dem Simulator. Die Funktionalität der BA dient im wesentlichen nur dazu, die notwendigen Signale automatisch der Stimulus-Datei hinzuzufügen, damit diese bei der nachfolgenden Simulation aufgezeichnet werden.

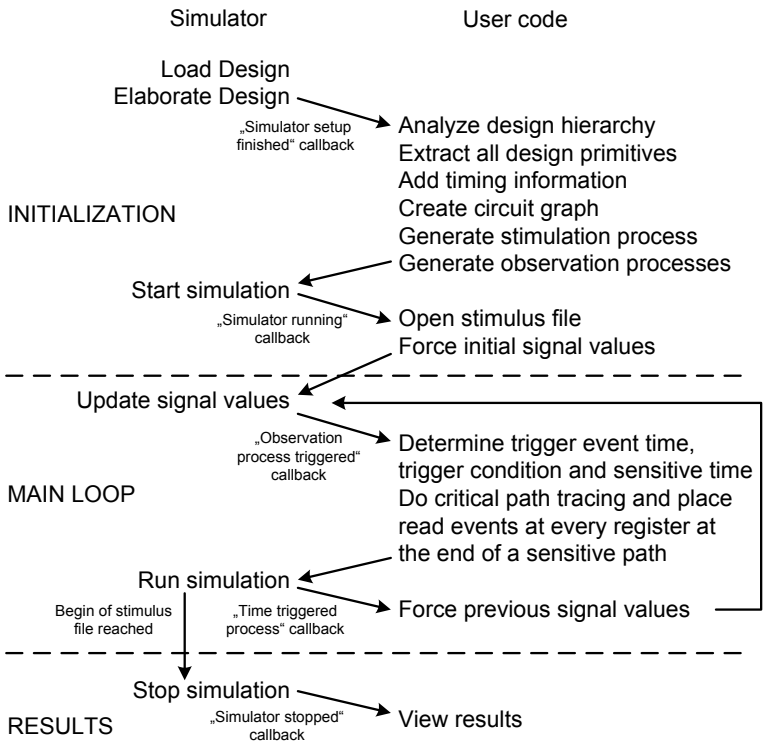



Abbildung 4.16: Software-Ablauf des Analyseteils der Rückwärts-Analyse

Im Analyseteil sind die notwendigen Vorgänge zahlreicher. Grundlage für die Analyse ist die Graphenstruktur der Schaltung und die Timing-Information. Beides wird gleich zu Beginn eingelesen, damit dann auch die Überwachungs-Prozesse erzeugt werden können. Im Folgenden wird die Stimulus-Datei geöffnet und die Schaltung mit den initialen Werten stimuliert. Den Hauptteil der Analyse bildet ein Wechsel zwischen Stimulation der Schaltung (und zugehörigem Lesen aus der Stimulus-Datei) und die Bearbeitung der durch die Stimuli ausgelösten Prozesse. Nachdem der Anfang der Stimulus-Datei erreicht ist und die letzten Stimuli in die Schaltung eingespeist wurden, kann die Analyse abgeschlossen werden. Letzte Aufgabe im Analyseteil ist die Anzeige der Ergebnisse.

## 4.10 Abschätzung der Performance

Die Rückwärts-Analyse zeichnet sich durch besonderes Verhalten auch bei der Abschätzung der Laufzeit aus. Dafür sollen beide Ausführungsschritte separat betrachtet werden. Da der erste Schritt nur aus einer einfachen Schaltungssimulation mit zusätzlicher Aufzeichnung der simulierten Signalverläufe besteht, kann dessen Ausführungsdauer einfach abgeschätzt werden.

Die Ausführungsdauer der Schaltungssimulation hängt von der Größe der simulierten Schaltung, d.h. im wesentlichen von der Anzahl der Schaltungselemen-

te und der Länge der simulierten Zeit bzw. der Menge an Eingangsstimuli ab. Diese beiden Größen zusammen stellen im weiteren Verlauf die sogenannte Problemgröße dar [59]. Da die üblicherweise verwendeten Schaltungselemente nur eine begrenzte Anzahl an Eingangssignalen haben, ist die Komplexität der Beschreibung der Schaltungselemente ebenfalls beschränkt. Dadurch kann gefolgert werden, dass jedes Schaltungselement i.A. eine Obergrenze der Berechnungsdauer eines Aufrufes/Iteration hat. Die Komplexität der Schaltungselemente wächst daher nur bis zu einem bestimmten Grad mit der Größe der Schaltung mit, darüber hinaus bleibt diese annähernd konstant. Daher kann angenommen werden, dass die Simulationsdauer linear mit der Größe der Schaltung wächst. Lediglich wenn der die Simulation ausführende Rechner im Verhältnis zur simulierten Schaltung wenig Rechenleistung und Speicherkapazität hat, wächst die Simulationsdauer mehr als linear im Verhältnis zur Schaltungsgröße. Dies ist aber ein Phänomen, dass jeden Computeralgorithmus betrifft.

Die Länge der simulierten Zeit hat einen Einfluss auf die Ausführungsdauer der Simulation, weil mit der simulierten Zeitspanne auch die Anzahl der zu betrachtenden Eingangsstimuli mitwächst. In atypischen Eingangsstimuli-Daten kann es vorkommen, dass die simulierte Schaltung z.B. nahezu keine Aktivität zeigt und damit unabhängig von der Länge der gesamten Eingangsstimuli mit sehr geringem Rechenaufwand simuliert werden kann. Jedoch sind solche Stimulus-Daten nur in seltenen Fällen sinnvoll und für eine Analyse interessant, deshalb sollen diese auch keinen Einfluss auf die Abschätzung der Performance finden.

Als einfacher Teil des Simulationsteils können die Vorbereitungen, d.h. das Erstellen der Liste der aufzuzeichnenden Signale, gesehen werden. Dies kann entweder direkt als Vorgabe für den Algorithmus bereits gegeben sein oder zur Laufzeit durch den Algorithmus selbst bestimmt werden. Hierfür muss nur die Schaltungsbeschreibung analysiert werden. Der Aufwand dafür wächst linear mit der Größe der Schaltung und ist von der Simulationsdauer unabhängig. Im schlechtesten Fall ist die Simulationsdauer äußerst kurz, d.h. die Vorbereitungen zur Simulation dominieren den Zeitaufwand für den Simulationsteil. Deshalb ist die Ausführungszeit für die Vorbereitung linear von der Schaltungsgröße abhängig.

Die Aufzeichnung der Signalverläufe verursacht nur einen geringen zusätzlichen Aufwand. Zur Durchführung der Schaltungssimulation müssen ohnehin alle Signale bestimmt werden. Das Speichern und die Verfügbarkeit der Signalverläufe über einen längeren Zeitraum ist jedoch für eine einfache Simulation nicht notwendig. Das Aufzeichnen findet in der Regel im Arbeitsspeicher oder in einem externen Massenspeicher des Simulationsrechners statt. Der resultierende Speicherbedarf hängt wie die Simulationsdauer i.A. von der Menge der Signale in der Schaltung und der Länge des simulierten Zeitraums ab.

Die Rückwärts-Analyse nutzt das WLF-Dateiformat zur Aufzeichnung der Signale, der zeitliche Aufwand hierfür wächst daher linear mit der Anzahl der aufzuzeichnenden Signalübergänge und damit auch mit der Problemgröße.

Für den zweiten Teil der BA können nun ähnliche Überlegungen angestellt wer-

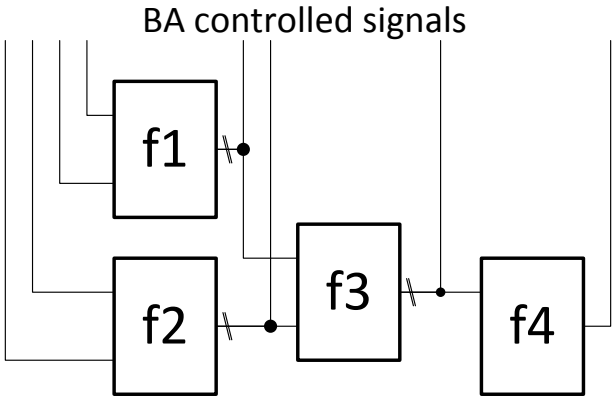



Abbildung 4.17: Blockierte Gatterausgänge durch BA-Stimulation

den. Zuerst soll der Aufwand zur Stimulation der Schaltung abgeschätzt werden. Je nach Problemgröße müssen unterschiedlich viele Transitionen in die Schaltung eingebracht werden. Hier muss angenommen werden, das dieser Vorgang für jedes Signal annähernd gleich aufwändig ist und daher gleich lange dauert.

Wie im vorherigen Abschnitt 4.8 erklärt, können zum Einlesen und Ausgeben der Signal-Transitionen geeignete Datenstrukturen verwendet werden. Die verwendeten Algorithmen (Sortieren, Hinzufügen und Löschen von Listenelementen) haben eine Komplexität der Ordnung $O = \log n$. Zusätzlich kann erwartet werden, dass die betrachteten Listen in den allermeisten Fällen sehr klein sind, daher ist der Rechenaufwand für diesen Teil der Gesamtfunktion dementsprechend niedrig.

Jedoch findet auf jede eingespeiste Signaländerung auch potentiell eine Reaktion der Schaltung statt, da die in der Schaltung vorhandenen Prozesse mit deren Sensitivity-List angestoßen werden können. Weil aber alle Schaltungssignale durch die Rückwärtsanalyse mit Vorrang gegenüber Signaländerungen durch Prozesse gesteuert werden und damit niemals eine Änderung eines Gatterausgangssignals stattfindet, werden keine weiteren Prozesse angestoßen (Abbildung 4.17). Da das Schaltungsverhalten in zeitlich umgekehrter Reihenfolge abläuft, hat ein Gatterausgangssignal bereits einen Wert, der sich auch nach den eben eingespeisten Signalen und der Logikfunktion des Gatters ergeben würde. Eine Reaktion des Simulators auf eingespeiste Signale findet also nur für die unmittelbar von Signaländerungen betroffenen Gatter statt, dennoch werden in Summe die Simulationsprozesse genauso oft wie im Simulationsteil ausgeführt. Selbst die Simulationsprozesse aufgrund von Glitches werden ausgeführt. Die benötigte Rechenleistung für die Ausführung der Simulationsprozesse ist unnötig und könnte einen Ansatzpunkt für weitere Optimierungen bilden.

Ein Nachverfolgen der empfindlichen Prozesse wird erst gestartet, wenn ein geeignetes Signal an einem speichernden Element stimuliert wurde. Daher ist der Rechenaufwand hierfür prinzipiell unabhängig vom Aufwand zur Schaltungsstimulation und zur Simulation der nachfolgend getriggerten Prozesse. Wenn die

Nachverfolgung der empfindlichen Pfade gestartet wurde, wird der Logikbaum maximal bis zum Erreichen aller nächsten sequentiellen Elemente durchsucht.

Für die Größe des zu durchsuchenden Logikbaums kann ebenfalls eine Abschätzung getroffen werden. Jede größere digitale Schaltung kann wie in Abbildung 4.1 dargestellt werden. Hier ist es praktisch ausgeschlossen, dass bei den Funktionen $F$ und $G$ jedes einzelne der Ausgangssignale von allen Eingangssignalen abhängt. In realistischen Schaltungen ist die Anzahl der Eingangssignale für jede der Teilfunktionen von $F$ und $G$ weitaus geringer. Dies wird auch dadurch begründet, dass mit wachsender Schaltungsgröße die maximal mögliche Taktfrequenz der Schaltung nicht automatisch sinkt, sondern die maximale Länge der kombinatorischen Pfade eine Sättigung erfährt. Zusammen mit der zuvor getroffenen Annahme einer ebenfalls nach oben hin begrenzten Komplexität eines einzelnen Gatters wächst die mittlere Größe der Logikbäume der Schaltung weniger als linear mit der Schaltungsgröße bzw. sättigt auf einem konstanten Wert. Der Rechenaufwand zur Nachverfolgung eines empfindlichen Pfades wird daher als konstant gegenüber der Schaltungsgröße angenommen, jedoch steigt die Anzahl der Trigger-Events und damit auch die Anzahl der nachzuverfolgenden empfindlichen Pfade mit der Problemgröße.

Letzte Teilaufgabe während der Analyse ist die Auswertung der Read- und Write-Events. Da die Read-Events bereits in einer sortierten Liste vorliegen, sind selbst bei größeren Listen die in Frage kommenden Read-Events einfach auszumachen. Hierfür kann eine binäre Suche mit logarithmischer Komplexität in Abhängigkeit von der Listengröße verwendet werden. Wie viele Read-Events tatsächlich in der Liste eingetragen sind, hängt zum Einen von der Schaltungsstruktur ab. Wenn ein speicherndes Element auf viele nachfolgende Speicher Einfluss nehmen kann, können diese umgekehrt auch viele Read-Events senden. Zweiter beeinflussender Faktor für die Anzahl der Read-Events ist die Trigger-Häufigkeit der nachfolgenden Speicher. Abhängig von der Schaltungsstruktur und des Anwendungsfalls der Schaltung werden die Quellen- und Senken-Speicher unterschiedlich oft getriggert. Wird der Quellenspeicher häufiger als die Senken-Speicher getriggert, verringert sich die Anzahl der zu betrachtenden Read-Events. Wenn die Senken-Speicher häufiger als der Quellen-Speicher getriggert werden, wächst zwar die Liste der Read-Events, jedoch wird dieser Speicher im Vergleich zur gesamten Schaltung relativ selten getriggert. Diese gegensätzlich beeinflussenden Faktoren führen zu einem abgeschätzten linear wachsenden Aufwand gegenüber der Problemgröße.

Für den Simulationsteil bzw. dessen Teilaufgaben können folgende Komplexitäten für die Ausführungszeit angenommen werden:

- Vorbereitungen Schaltungssimulation $\in O(n))$
- Schaltungssimulation $\in O(n)$
- Aufzeichnen der Signalverläufe $\in O(n)$

Da alle Teilkomponenten des Simulationsteils nur lineare Abhängigkeit gegenüber der Problemgröße zeigen, kann für die Zeitkomplexität des Simulationsteils

insgesamt eine lineare Abhängigkeit abgeschätzt werden.

Der Analyse-Teil setzt sich aus den nachfolgend aufgelisteten Teilen zusammen:

- Vorbereitungen zur Schaltungsanalyse $\in O(n)$
- Einspeisen der Signalverläufe $\in O(n)$
- Nachverfolgen der empfindlichen Pfade $\in O(n)$
- Auswertung der Read- und Write-Events $\in O(n)$

Auch der wesentlich aufwändigere Analyse-Teil zeigt über alle Teilaufgaben maximal einen linear wachsenden Rechenaufwand gegenüber der Problemgröße. Dadurch ist die Rückwärts-Analyse prinzipiell geeignet auch sehr große Schaltungen zu bearbeiten. In nachfolgenden Kapiteln wird der absolute und relative Aufwand gegenüber einer einfachen Schaltungssimulation bestimmt werden.

## 4.11 Zusammenfassung

Im diesem Kapitel wurden die notwendigen Überlegungen zur Konzipierung der Rückwärts-Analyse vorgestellt. Durch die genaue Analyse der Eigenschaften anderer Konzepte und der Erfordernisse konnte der Algorithmus so gestaltet werden, dass die meisten Nachteile vermieden werden konnten. Im weiteren wurden die Grenzen des Anwendungsbereichs, der Software-Ablauf und die theoretische Laufzeit des Algorithmus erläutert.

# Kapitel 5

# Optimierungsmöglichkeiten

Im vorherigen Kapitel wurden die grundlegenden Ideen hinter der Rückwärts-Analyse vorgestellt, wobei auch einige Schwachstellen aufgezeigt wurden. Für die benannten Probleme sollen systematische Lösungsmöglichkeiten aufgezeigt werden und die Anwendungsbereiche der Rückwärtsanalyse erweitert werden.

## 5.1 Simulation ohne Timing

Im Abschnitt 4.7 wurde erklärt, warum in Abhängigkeit von Clock-to-Q-Zeit und Clock-Skew falsche Analyseergebnisse entstehen können. Die grundsätzliche Problemquelle liegt in der Pseudo-Hold-Time-Violation und der damit unerwarteten Änderung eines Quellen-Signals vor dem zugehörigen Trigger des Senken-Signals. Zur Lösung dieses Problems könnten mehrere Wege eingeschlagen werden:

- Zeitverzögertes Nachverfolgen der empfindlichen Pfade
- Erhöhung der Clock-to-Q-Zeit der betroffenen Speicher
- Verringerung des Clock-Skew

### 5.1.1 Zeitverzögertes Nachverfolgen der empfindlichen Pfade

Bei der ersten vorgeschlagenen Lösung würde von dem bisherigen Schema, der Auswertung der Schaltungssignale unmittelbar nach dem Trigger-Event, abgewichen werden. Wie in Abbildung 5.1 ersichtlich, könnten unter Kenntnis der Signalwerte zu einem früheren Zeitpunkt (bestimmt durch den Trigger-Zeitpunkt und die Gatterverzögerungszeit) die empfindlichen Pfade richtig berechnet werden. Da aber die Gatterverzögerungszeit nicht für alle Pfade identisch sein muss, kann eine mit der Stimulation schritthaltenden Analyse ausgeschlossen werden. Eine zeitlich verzögerte Ausführung der Analyseprozesse würde keinen Vorteil mit sich bringen, da bei unterschiedlich langen Gatterverzögerungszeiten abhängig vom betrachteten Pfad kein Zeitpunkt existiert, an dem alle Gattereingangssignale ausgelesen werden können um damit die empfindlichen Pfade zu berechnen. Die unterschiedliche Pfad-Laufzeit führt zwangsweise dazu, dass die Signalwerte

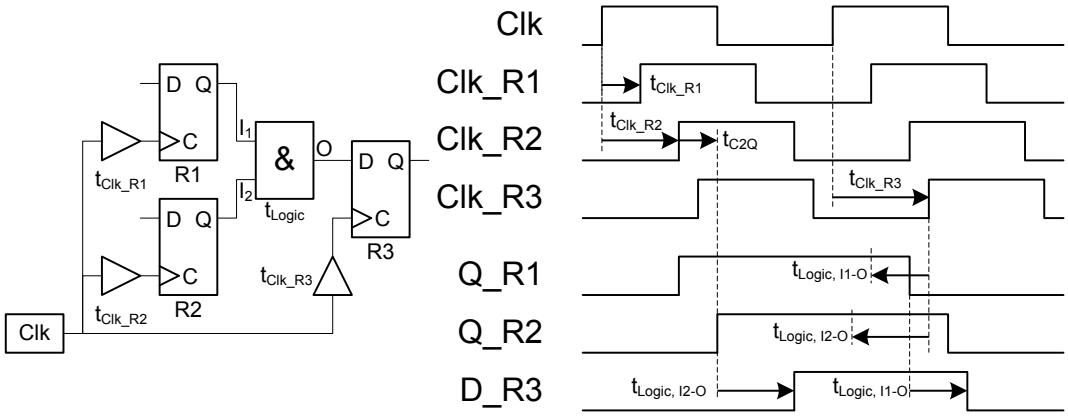



Abbildung 5.1: Zeitpunkte für die Bestimmung der empfindlichen Pfade bei der universellen Rückwärts-Analyse

von unterschiedlichen Zeitpunkten für die Berechnung herangezogen werden müssen. Aus den Stimulationsdaten könnte diese Information aber gewonnen werden. Hierzu müsste der Wert der betroffenen Eingangssignale in der Datei gesucht werden, was aber, wie in Abschnitt 4.8 erklärt, nicht effizient durchgeführt werden kann. Für jedes Signal müsste die Datei vom Triggerzeitpunkt an ein Abschnitt der Länge entsprechend der Pfad-Verzögerungszeit durchsucht werden, um entweder eine Transition dieses Signals zu finden oder ausschließen zu können, das es eine Transition gibt.

Durch diese Konzeptänderung verringert sich die Ausführungsgeschwindigkeit des Algorithmus stark, da sehr viel zusätzliche Rechenzeit für das Suchen der Signalwerte verwendet wird. Dafür garantiert diese Modifikation richtige BA-Ergebnisse ohne Einschränkungen bei den Schaltungseigenschaften.

### 5.1.2 Erhöhte Clock-to-Q-Zeit

Der zweite potentielle Ansatz zur Vermeidung von falschen BA-Ergebnissen auf Grund von Pseudo-Hold-Time-Violation ist die Erhöhung der Clock-to-Q-Zeit der betroffenen Speicher. Durch eine Timing-Analyse der Schaltung kann die notwendige zusätzliche Clock-to-Q-Zeit bestimmt werden. Für die nachfolgende Analyse ist es aber nicht ausreichend, einfach nur die betroffenen Verzögerungszeiten passend zu verändern, da die durch die Simulation entstandenen Signalverläufe immer noch das unveränderte Timing wiederspiegeln. Daher würden bei der Analyse immer noch nicht zusammengehörige Signalwerte zur Bestimmung der empfindlichen Pfade herangezogen, weshalb diese Maßnahme dann wirkungslos wäre. Damit der Einfluss des veränderten Schaltungstimings zum Tragen kommt, muss der Simulationsteil neu durchgeführt werden. Das veränderte Schaltungstiming kann aber auch einen negativen Einfluss auf das Schaltungsverhalten haben. Eine Erhöhung der Clock-to-Q-Zeit eines Speichers kann zu einfachen Setup-Zeit-

Verletzungen am anderen Ende der betroffenen Pfade führen. Dadurch käme es dann zu einer Verfälschung anderer Signalwerte und damit ebenfalls zu falschen BA-Ergebnissen. Zusätzlich fließt die Clock-To-Q-Zeit auch in die Berechnung der empfindlichen Zeit ein. Die Analyse einer modifizierten Schaltung würde demnach immer verfälschte empfindliche Zeiten liefern. Dieser negative Effekt könnte aber einfach verhindert werden, wenn für die Analyse das unveränderte Schaltungstiming herangezogen würde.

Welche Auswirkungen das veränderte Schaltungstiming auf das Simulationsergebnis und auch auf das Analyse-Ergebnis hat, muss noch genauer betrachtet werden. Für alle Logikpfade an deren Anfang ein Speicher mit veränderter Clock-to-Q-Zeit steht, müssen nur die originalen Timing-Werte der Schaltung verwendet werden. Da der Triggerzeitpunkt der nachfolgenden Speicher nicht beeinflusst wurde und auch die Verzögerungszeiten der dazwischenliegenden Logikgatter unverändert sind, entspricht das Analyse-Ergebnis dem richtigen Ergebnis. Wenn aber von einem Ausgang eines Speichers mit veränderter Clock-to-Q-Zeit wiederum die Triggerquellen anderer Speicher abhängen, so muss dieser Effekt bei der Bestimmung des Triggerzeitpunkts berücksichtigt werden. Eine vergrößerte Clock-To-Q-Zeit muss mit einem in die Vergangenheit verschobenen Triggerzeitpunkt kompensiert werden, da in der unveränderten Schaltung die Trigger-Events sonst früher aufgetreten wären. Für einen Speicher mit veränderten Timing-Eigenschaften selbst ist bei einem Trigger-Event zur Kompensation der Veränderungen nichts zu unternehmen.

Bei diesem Ansatz muss vor der Durchführung genau betrachtet werden, ob das veränderte Timing eine Änderung des sequentiellen Verhaltens der Schaltung verursacht. Ist dies nicht der Fall, so können mit dem Ansatz der veränderten Clock-to-Q-Zeit und den notwendigen Kompensationsmaßnahmen die Auswirkungen der Pseudo-Hold-Time-Violation verhindert werden. Dieser Ansatz hat bis auf den Rechenaufwand für Kompensationsrechnungen keinen Mehraufwand gegenüber dem ursprünglichen Verfahren der BA.

### 5.1.3 Modifiziertes Timing für Simulation

Im dritten Ansatz zur Verbesserung der BA soll der Grundgedanke des zweiten Ansatzes aufgenommen werden, aber in drastischerer Weise auf die BA angewandt werden. Um die Auswirkungen der Pseudo-Hold-Time-Violation auf das Analyse-Ergebnis zu verhindern, könnte der Clock-Skew verringert werden. Da der Clock-Skew aber kein direkter Simulationsparameter ist, sondern nur eine rechnerische Größe, müssen die Veränderungen an anderen Größen durchgeführt werden. Der Clock-Skew ist direkt von der Verzögerungszeit des Quellen- und Senken-Speichers einer sequentiellen Stufe abhängig. Um den Clock-Skew auf das notwendige Maß zu reduzieren, könnte die Verzögerungszeit auf dem Pfad des Taktsignals zum Quellen-Register erhöht werden. Dies führt aber wiederum auf den Signalpfaden zu den vorhergehenden Speichern zu einer Erhöhung

des Clock-Skew. Alternativ dazu könnte auch die Verzögerungszeit des Taktsignals am Senken-Speicher verringert werden, dadurch erhöht sich aber der Clock-Skew bei den nachfolgenden sequentiellen Stufen. Weil aber der Clock-Skew evtl. auch bewusst in einer Schaltung eingesetzt wurde, um eine Verletzung der Setup-Bedingung einer sequentiellen Stufe zu vermeiden, muss über die weiteren Auswirkungen bei Änderungen der Timing-Parameter nachgedacht werden. Beide Lösungsvorschläge zur Anpassung des Clock-Skew führen nicht ohne Nebeneffekte zum Ziel.

Für die Rückwärts-Analyse ist neben den beiden für die Schaltungsfunktion ausschlaggebenden Bedingungen (Setup-Bedingung, Hold-Bedingung) auch die Pseudo-Hold-Bedingung einzuhalten. Da der speicherspezifische Wert der Hold-Zeit keinen Einfluss auf das Simulationsergebnis hat, zumindest solange es zu keiner Verletzung der Hold-Bedingung kommt, könnte dieser Wert pauschal für alle Speicher in der Schaltung auf Null gesetzt werden um die Einschränkungen bei der Wahl der restlichen Schaltungs-Timing-Parameter möglichst gering zu halten. Wenn die Hold-Zeit auf Null gesetzt wird und gleichzeitig die Clock-to-Q-Zeit aller Speicher der Schaltung größer als Null ist, hängt die Einhaltung der Hold-Bedingung nur noch vom Clock-Skew und den Gatterverzögerungszeiten ab. In einem zweiten Schritt könnte nun der Clock-Skew auf Null reduziert werden. Dazu müsste durch eine Modifikation des Schaltungstimings auf allen Taktsignal-Pfaden die gleiche Verzögerungszeit hergestellt werden. Für die asynchronen Steuersignale der Speicher in der Schaltung muss ebenfalls die gleiche Verzögerungszeit eingestellt werden.

Ein günstiger Sonderfall für gleiche Verzögerungszeiten auf allen Signalpfaden, die einen Trigger auslösen können, stellt die Reduktion aller Verzögerungszeiten auf den Wert Null dar [60]. Auch die Verzögerungszeiten der Logikgatter sollen verändert werden. Durch diese Maßnahme wird das bisher angenommene möglichst realitätsnahe Schaltungsmodell zu einem Schaltungsmodell auf Register-Transfer-Ebene verändert (Abbildung 5.2). Dieses Modell ist damit aber nicht pauschal für die Rückwärts-Analyse einsetzbar.

Wesentliche Probleme entstehen, wenn bei diesem Wechsel von einem Schaltungsmodell mit Timing zu einem RT-Modell grundsätzliche Abweichungen im Schaltungsverhalten auftreten. Speziell am Anfang der Simulation kann es häufig zu solchen Abweichungen kommen, wenn die Wellenformen einzelner Signale bei einem Vergleich der unmodifizierten und modifizierten Schaltung um mehr als eine Taktperiode verschoben sind.

Dies kann typischerweise bei folgenden Elementen auftreten:

- Taktgeber, Taktmodifikation (DLL, PLL und ähnliche Funktionen)
- Komponenten für den Schaltungsreset
- Andere Komponenten mit analogen Funktionen

Bei diesen Elementen kann es bei Initialisierung, Reset oder dem normalen Betrieb zu einem anderen Simulationsverlauf kommen. Dementsprechend verändert sich auch das Analyseergebnis. Abhilfe kann eine zusätzliche manuelle Manipu-

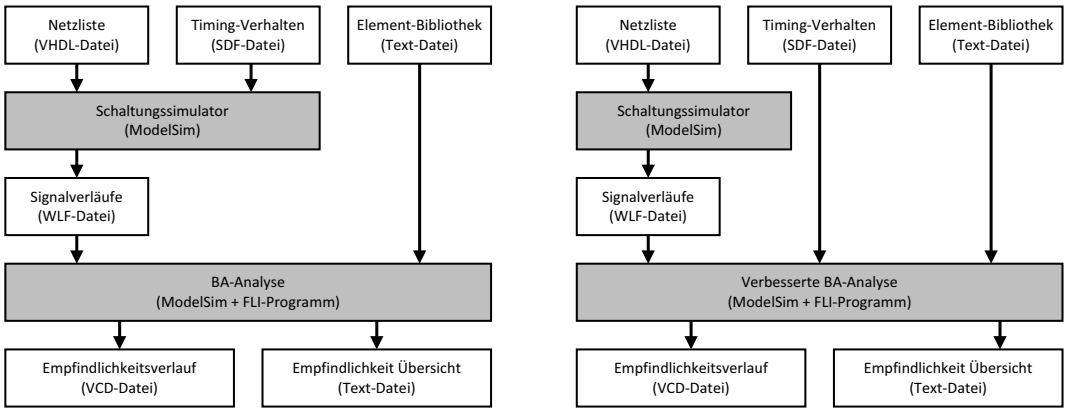



Abbildung 5.2: Vergleich einfache und verbesserte BA

lation der Schaltungssimulation schaffen, um den abweichenden Verlauf auszugleichen. Als Beispiel kann ein Taktgeber herangezogen werden dessen Ausgang auf einem festen Wert festgehalten wird. Dadurch wird erreicht, dass sich die im Verhaltensmodell kürzere Anlaufzeit des Taktgebers nicht in Form eines früher zur Verfügung stehenden Taktsignals äußert (Abbildung 5.3). Abhängig von den verwendeten Schaltungskomponenten können unterschiedliche Maßnahmen zum Ziel führen.

Bei der Analyse einer derart modifizierten Schaltung ergeben sich mehrere Vorteile. Durch das vereinfachte Timing läuft die Simulation zur Erzeugung der Signalverläufe schneller ab. Durch den Entfall der Verzögerungszeiten auf den Taktpfaden und durch Logikgatter werden in der Regel die allermeisten Simulationsprozesse zeitgleich mit dem Taktsignal ausgeführt. Dies führt dazu, dass der Schaltungssimulator nur an wenigen Zeitpunkten aktiv ist, dann aber in der Regel die ganze Schaltung bearbeitet. Gegenüber der Bearbeitung von wenigen Prozessen an vielen Zeitpunkten entsteht hier ein Vorteil in der Ausführungszeit. Weiterhin entstehen bei einer Simulation mit vereinfachtem Timing wesentlich weniger Glitches auf Schaltungssignalen. Daher werden insgesamt weniger Prozesse in kombinatorischen Gattern ausgeführt, da sich deren Eingangssignale alle zum gleichen Zeitpunkt, häufig sogar im gleichen Iterationszyklus ändern. Die Anzahl der Signalübergänge geht direkt in den Aufwand zur Stimulation der Schaltung ein.

Auf Grund der nicht vorhandenen Gatterverzögerungszeit kann eine weitere tiefgreifende Optimierung vorgenommen werden. Da nun alle Speicher einer Schaltung in einer Taktdomäne zum gleichen Zeitpunkt getriggert werden, ändern sich auch deren Ausgangssignale und die Ausgangssignale der nachfolgenden kombinatorischen Logikgatter zum gleichen Zeitpunkt. Bisher wurde für den Analyseteil immer angenommen, dass alle Ausgangssignale anhand der aufgezeichneten Signalverläufe stimuliert werden müssen. Wenn nun ein Ausgangssignal eines beliebigen Gatters (kombinatorische als auch sequentielle Gatter) stimuliert wird, so

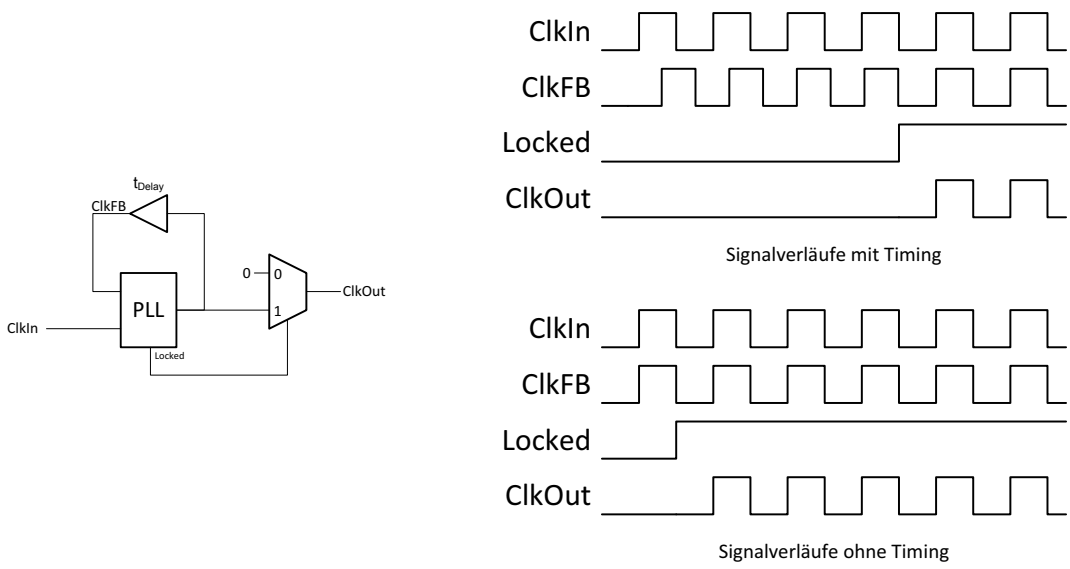



Abbildung 5.3: Signalverläufe einer Schaltung zur Taktgenerierung mit und ohne Einfluss des Schaltungstimings

führt dies zu einer Reaktion am nachfolgenden Gatter. Wenn sich eines oder mehrere der Eingangssignale dieses Gatters verändert hat, wird in Folge dessen ein gatterinterner Prozess gestartet, der den Wert des Ausgangssignals neu bestimmt. Nachdem aber auch dieser Ausgang durch die Rückwärtsanalyse stimuliert wird, ist die gattereigene Ausgangszuweisung nutzlos. Aus diesem Sachverhalt ergeben sich nun zwei weitere Optimierungsansätze:

- Deaktivierung der gatterinternen Prozesse um Rechenzeit einzusparen
- Verzicht auf Stimulation der kombinatorischen Logikgatter, da diese ohnehin selbständig ihre Ausgangssignale bestimmen

Der erste Ansatz ist prinzipiell möglich, jedoch muss für den Analyseteil eine andere Gatterbibliothek vorhanden sein. Diese Bibliothek beinhaltet nicht wie im Simulationsteil verwendet die Funktion aller verwendeten Gatter, sondern darf nur leere, funktionslose Hüllen enthalten, da sonst keine Rechenzeit bei der Analyse gespart werden kann. Nachdem bei diesem Ansatz neben der Gatterbibliothek für die Rückwärts-Analyse nun auch noch eine zusätzliche Simulationsbibliothek notwendig wäre, wurde dieser Ansatz nicht weiter verfolgt.

Der zweite Optimierungsansatz bietet hier die größeren Vorteile. Wenn es nicht mehr notwendig ist, die Ausgangssignale der kombinatorischen Gatter zu stimulieren, kann der Aufwand für das Einlesen dieser Signale eingespart werden, ebenso wie die eigentliche Stimulation mit diesen Signalen. Mehr noch, selbst das Aufzeichnen dieser Signale ist dann nicht mehr notwendig, was zu einer deutlich verkleinerten Datei mit Signalverläufen führt. Je nachdem, wie das Verhältnis von kombinatorischen und sequentiellen Gattern in der Schaltung ist, führt dieser Ansatz zu einem unterschiedlich hohen Zugewinn an Analysegeschwindig-

keit. Speziell bei Taktsignalen, die üblicherweise als einzige Quelle durch Verzögerungselemente mit vielen Eingängen verbunden sind und zudem die höchste Schaltaktivität aufweisen, führt diese Optimierung zu deutlichen Verbesserungen der Ausführungsgeschwindigkeit.

Weiteres Optimierungspotential kann bei überlappenden Logikbäumen ausgemacht werden. Beispielsweise haben in arithmetischen Funktionen (z.B. Addiererschaltungen) große Teile der Schaltung Einfluss auf mehrere Ausgangsregister. Die Logikbäume aller Ausgangssignale dieser Funktionen überlappen sich zu einem großen Teil (Abbildung 5.4). Wenn nun bei der Rückwärtsanalyse die Ausgangsregister getriggert werden, führt dies im schlimmsten Fall dazu, dass alle Logikgatter in überlappenden Logikbäumen mehrfach untersucht werden. Die Analyse der empfindlichen Pfade wird für jedes getriggerte Register unabhängig durchgeführt.

In einer unmodifizierten Schaltung treten die Events zwar gehäuft in einem kurzem Zeitraum nach der Taktflanke auf, nicht notwendigerweise aber gleichzeitig. Dagegen treten in einer modifizierten Schaltung alle Events, die durch dass gleiche Taktsignal ausgelöst werden gleichzeitig auf, weil die Verzögerungszeit auf den Taktpfaden gleich null ist. Damit ist sichergestellt, dass die Analyse der empfindlichen Pfade zum gleichen Simulationszeitpunkt stattfindet, da auch alle Register gleichzeitig getriggert werden. Wenn ein Gatter in überlappenden Logikbäumen zum gleichen Zeitpunkt bereits untersucht wurde, kann auf das Einlesen der Eingangssignalwerte und die Bestimmung der empfindlichen Pfade verzichtet werden. Statt dessen werden die empfindlichen Pfade des vorherigen Aufrufes verwendet. Weil die Verzögerungszeiten auf unterschiedlichen Pfaden auch abweichend sein können und evtl. rekonvergente Pfade betrachtet werden, kann auf das Weiterpropagieren der Read-Events nicht verzichtet werden. Eine Filterung der Read-Events anhand des Event-Zeitpunkts kann nicht vorgenommen werden, da am Quellenregister ausschlaggebende Write-Events prinzipiell zu einem früherem Simulationszeitpunkt auftreten und deshalb keine Entscheidungsbasis vorhanden ist.

## 5.2 Modifikation der Netzliste

Wie schon im vorherigen Abschnitt zuletzt aufgezeigt, können bestimmte Strukturen in der zu untersuchenden Netzliste sich negativ auf die Analysegeschwindigkeit auswirken. Die genaue Struktur der Netzliste wird im wesentlichen von dem Programm bestimmt, welches zur Erzeugung der Netzliste verwendet wurde. Im diesem Abschnitt wird im speziellen auf die Auswirkungen und die mögliche Optimierung von Netzlisten eingegangen, welche durch die Softwarewerkzeuge von Xilinx erstellt wurden. Um den Ansatz dieser Optimierung zu Veranschaulichen, muss zuerst auf die Besonderheiten bei der Integration des Schaltungstimings eingegangen werden.

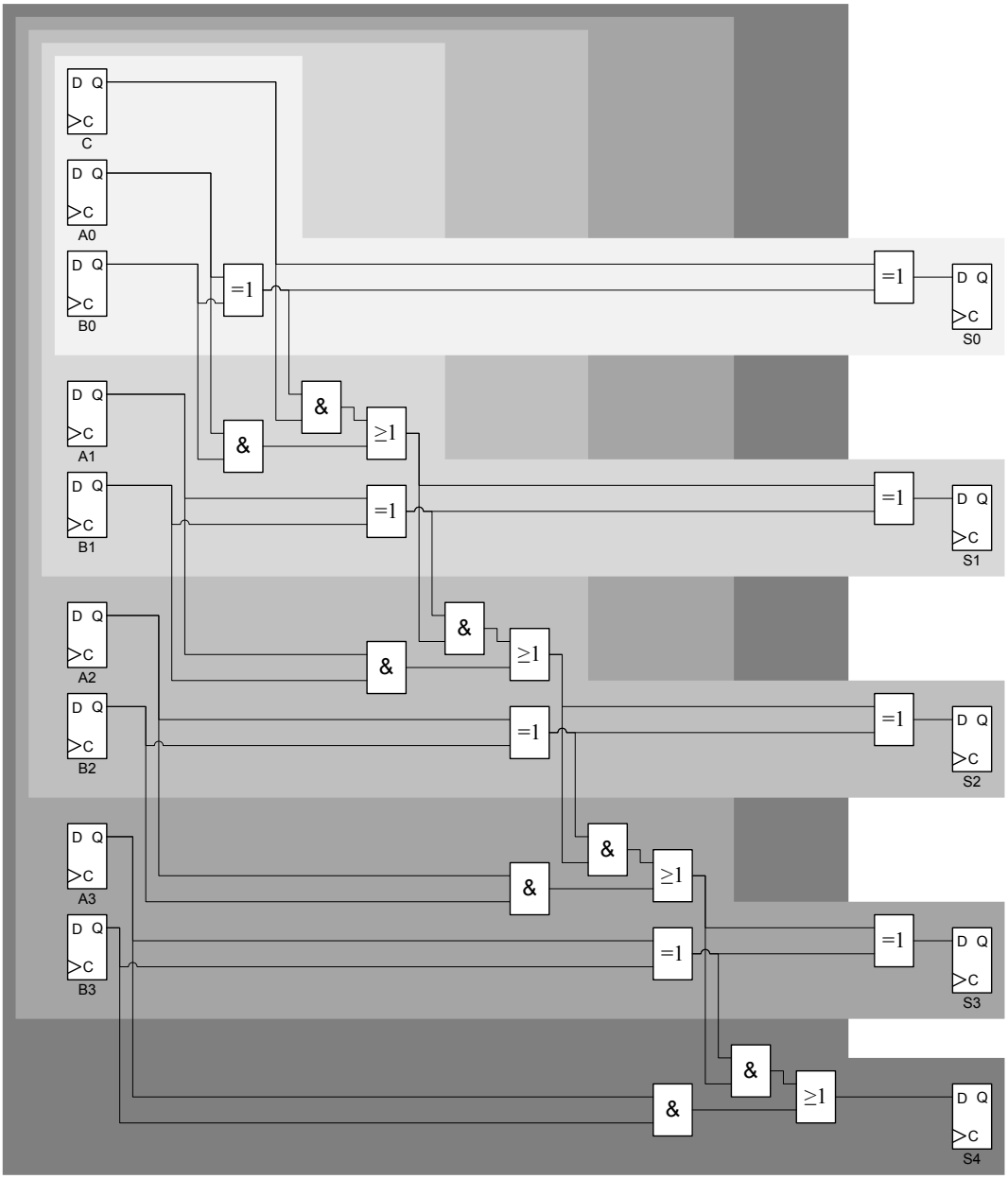



Abbildung 5.4: Überlappende Logikbäume

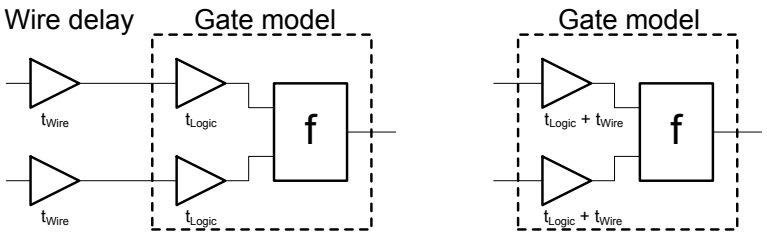



Abbildung 5.5: Einzelnes Gatter mit Verzögerungselementen und optimiertes Gatter mit zusammengefasster Gatter- und Leitungsverzögerung

Die SDF-Spezifikation erlaubt die Angabe von zweierlei Verzögerungszeiten für Eingangssignale:

- Verzögerungszeit für einen Eingang
- Verzögerungszeit für einen Eingangspfad (Eingang zu Ausgang)

Typische Xilinx-Simulationsmodelle in VHDL bestehen aus fünf Gruppen von funktionalen Blöcken:

- Darstellung der Resolution Function (IEEE1164 std_logic nach std_ulogic)
- $t_{ipd}$-Block für eine Eingangsverzögerungszeit
- $t_{pd}$-Block für eine Pfadverzögerung
- Logikblock zur Darstellung der logischen Schaltungsfunktion
- Assertion-Block zur Überwachung des zeitlichen Verhaltens und Ausgabe von Warnungen

Die beiden Verzögerungsblöcke eignen sich um die Verzögerungszeiten nach der SDF-Spezifikation im Simulationsmodell abzubilden. Beim Prozess der Netzlistenerstellung für eine Post-Place-And-Route-Simulation (PPR-Simulation), d.h. mit allen Verzögerungseffekten auf Grund der Leitungsführung, Platzierung der Komponenten und der Gatterverzögerungszeit, werden die resultierenden Verzögerungszeiten nicht auf die schon vorhandenen Verzögerungsblöcke abgebildet. Stattdessen werden als zusätzliche Komponenten einfache Verzögerungsgatter eingebaut, welche die Leitungsverzögerung implementieren.

Da bei der Rückwärts-Analyse die Anzahl der Gatter und der zu stimulierenden Signale Einfluss auf die Ausführungsgeschwindigkeit hat, wäre es vorteilhaft, diese an sich unnötigen Signale zu vermeiden. Mit einem einfachen Ansatz, bei dem diese Verzögerungszeiten in Signalflussrichtung vorwärts zum nächsten Gatter geschoben werden, können die zusätzlichen Verzögerungsgatter vermieden werden. Sofern das Ausgangssignal des Verzögerungselements mehrere Eingangssignale treibt, muss die zugehörige Verzögerungszeit auf alle Eingangspfade aufgeschlagen werden. Da nach dieser Änderung das Verzögerungselement praktisch nutzlos ist, kann es ersatzlos entfernt werden und eine verzögerungsfreie Verbindung zwischen dessen Ein- und Ausgangssignal hergestellt werden.

Bei dieser Modifikation der Netzliste werden Signale entfernt bzw. durch andere Signale ersetzt. Dadurch ist die originale und die modifizierte Netzliste nicht mehr

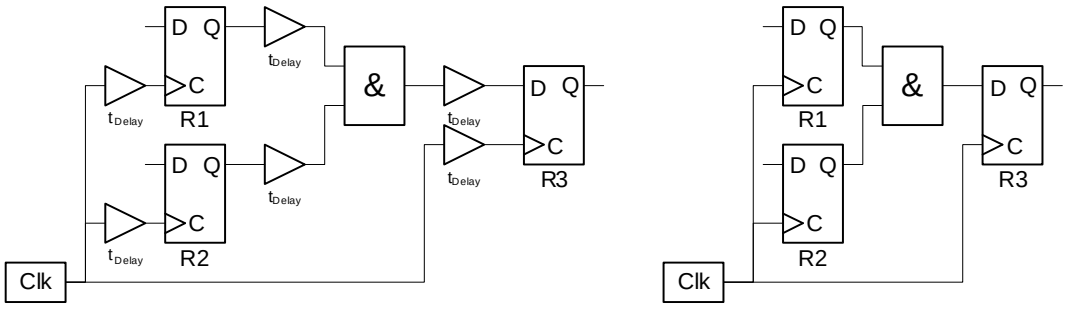



Abbildung 5.6: Ursprüngliche Struktur von Xilinx-PPR-Netzlisten und optimierte Netzlistenstruktur

direkt miteinander vergleichbar. Für die Signalverläufe der Speicher und aller primären Ausgänge ist identisches Verhalten aber trotzdem sichergestellt.

Durch diese Optimierung wird ebenfalls die Anzahl der Signale verringert die zur Stimulation der Schaltung notwendig sind, wobei dieser Effekt nur zum tragen kommt, wenn die Optimierung durch Weglassen des Timings nicht angewandt wird. In diesem Fall ist die Aufzeichnung der kombinatorischen Signale ohnehin nicht notwendig. Diese Optimierung ist dennoch in jedem Fall sinnvoll, da die Anzahl der Schaltungselemente um etwa 50% reduziert wird. Dadurch verringert sich die Simulationszeit und der Speicherbedarf für die Simulation. Während der Analyse sind bei der Untersuchung der empfindlichen Pfade ebenfalls weniger Gatter zu bearbeiten, damit verringert sich auch hier die Ausführungszeit.

## 5.3 Parallelisierung des Codes

Wie aus Abbildung 4.16 ersichtlich ist, wird der Programmablauf der Rückwärts-Analyse sequentiell ausgeführt. Im Analyseteil befindet sich der Algorithmus in einer Schleife, in der nacheinander alle Stimulationszeitpunkte abgearbeitet werden. Nach der Stimulation, die vom FLI-Programm ausgeführt wird, werden die Überwachungsfunktionen vom Simulator aufgerufen, sofern sich mindestens eines der überwachten Signale verändert hat. Bei gleichzeitig auftretenden Events (z.B. in größeren Schaltungen bei Verwendung des gleichen Taktsignals) ist die Reihenfolge der Ausführung der Überwachungsfunktionen aus Sicht des FLI-Programms undefiniert, aber auch prinzipiell irrelevant. Damit könnte dieser Programmabschnitt parallelisiert abgearbeitet werden.

Damit eine Parallelisierung möglich wird, dürfen aber auch keine Konflikte bzw. Datenabhängigkeiten entstehen. Nachdem bei der Untersuchung der empfindlichen Pfade als einzige Auswirkung die Read-Events in den Registern am Ende der empfindlichen Pfade hinzugefügt werden, besteht hier kein Problem. Die Read-Events, die bei gleichzeitig auftretenden Trigger-Events eingetragen werden können, müssen sich auf einen weiter in der Vergangenheit liegenden Zeitpunkt bezie-

hen und haben damit keine verfälschende Auswirkung auf das Analyse-Ergebnis. Die Reihenfolge der Abarbeitung der Trigger-Events spielt daher keine Rolle.

Folglich können die Triggerfunktionen von Speichern und alle nachfolgenden Funktionen zur Nachverfolgung der empfindlichen Pfade parallel auf mehreren Rechnerkernen ausgeführt werden. Den Engpass in der Ausführungsgeschwindigkeit bildet der Schaltungssimulator selbst, da dieser bis zum jetzigen Zeitpunkt nur auf einem Rechenkern ausgeführt wird. Ziel dieser Parallelisierung ist es, vom Hauptausführungsstrang so viel Rechenlast wie möglich zu entfernen. Auf diesem Rechenkern werden nach der Phase der Schaltungsstimulation nacheinander alle Überwachungsfunktionen ausgeführt. Wenn dann eine Veränderung eines überwachten Schaltungssignals aufgetreten ist, wird der zugehörige Teil des FLI-Programms ausgeführt. Dort kann dann ein eigener Ausführungsstrang für jede Triggerfunktion bzw. jeden betroffenen Speicher gestartet werden. Für den Haupt-Ausführungsstrang bedeutet dies, dass nach dem Start dieses zusätzlichen Ausführungsstrangs sofort zum Simulator zurückgekehrt werden kann, wo die Bearbeitung weiterer Überwachungsfunktionen durchgeführt werden kann. Der zusätzlich gestartete Ausführungsstrang kann zeitgleich arbeiten (Abbildung 5.7). Damit aber nicht etwa schon der nächste Stimulationsschritt ausgeführt wird, bevor die Bearbeitung des letzten Aufrufes einer Triggerfunktion abgeschlossen ist, muss an dieser Stelle ein Synchronisationsmechanismus verwendet werden. Der Simulatorkern darf erst dann einen weiteren Zeitschritt bearbeiten, wenn alle zusätzlichen Ausführungsstränge abgeschlossen sind.

Während der parallelen Ausführung der Triggerfunktionen darf es keine verfälschende Wechselwirkung zwischen den Ausführungssträngen geben. Speziell die im Konzeptteil genannten Listen mit den Read-Events müssen vor solchen Auswirkungen geschützt werden. Dies kann mit einfachen Software-Mechanismen erreicht werden, um einen gleichzeitigen Zugriff von zwei Ausführungssträngen zu verhindern. Für die Performance der Parallelisierung kann dies bedeuten, dass einzelne Ausführungsstränge kurzzeitig warten müssen bis der Zugriff auf die Read-Event-Liste eines Speichers wieder freigegeben wird.

Die zuvor beschriebene Optimierung durch Simulation ohne Timing führt dazu, dass alle von einem Taktsignal abhängigen Speicher zum gleichen Zeitpunkt getriggert werden. Die Möglichkeiten zur Parallelisierung mit dem optimierten Algorithmus (Abschnitt 5.1.3) verbessern sich gegenüber dem ursprünglichen Algorithmus. Da große digitale Schaltungen üblicherweise aus wesentlich mehr Speichern bestehen, als dem ausführenden Analyserechner Rechenkerne zur Verfügung stehen, kann der Rechner nahezu maximal ausgelastet werden.

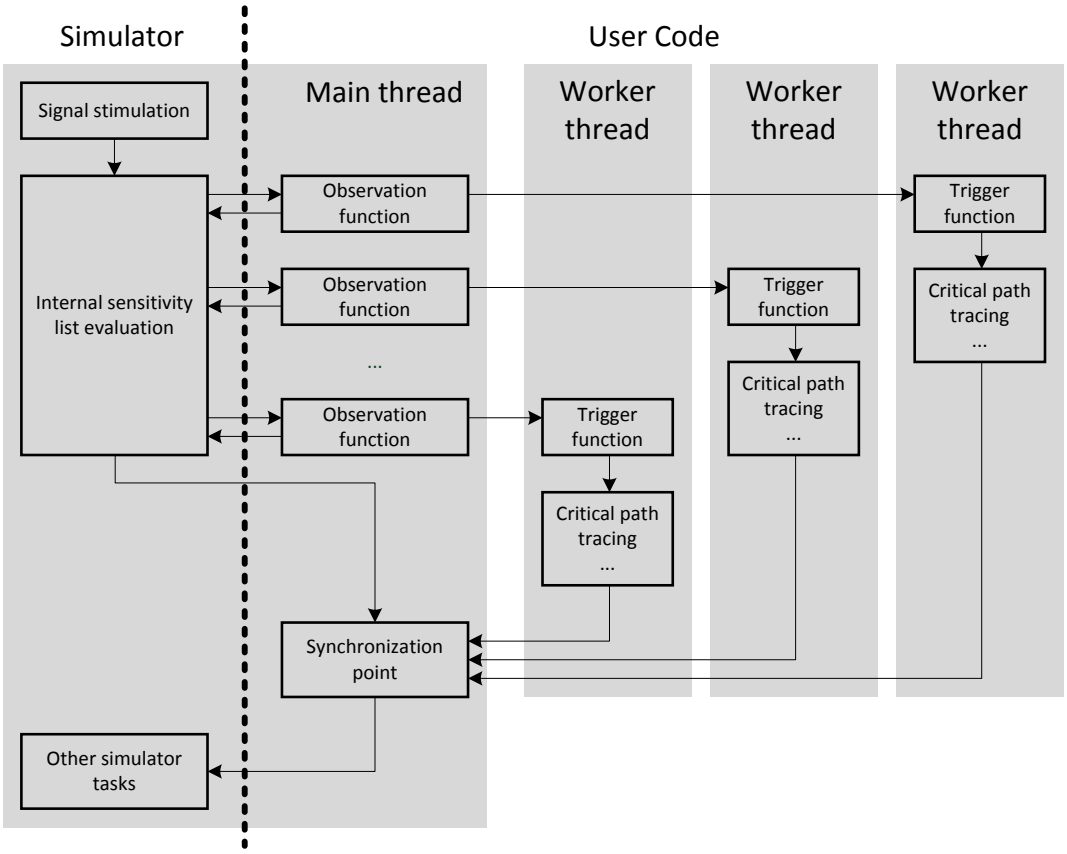



Abbildung 5.7: Software-Ablauf mit parallel arbeitenden Ausführungssträngen für Triggerfunktionen und nachfolgende Funktionen

# Kapitel 6

# Ergebnisse

In diesem Kapitel werden die Eigenschaften der Rückwärts-Analyse mittels Beispielschaltungen nachgewiesen und die vorgestellten Verbesserungen anhand von Vergleichen der Rechenlaufzeit bestätigt.

## 6.1 Verifikation der Ergebnisse anhand konstruierter Testschaltungen

### 6.1.1 Einfache sequentielle Schaltung

Die Basisfunktionalität der Rückwärts-Analyse wird anhand eines einfachen Beispiels demonstriert (Abbildung 6.1). Für diese und für alle nachfolgenden Beispiele wird angenommen, dass alle Ausgänge zu jeder Zeit beobachtbar sind. Ausgangssignale und die vorgeschalteten Ausgangsregister sind daher immer relevant. In diesem Beispiel werden weiterhin alle Triggersignale unabhängig voneinander angesehen, es werden keine Eigenschaften eines Taktbaumes mit in die Untersuchungen einbezogen. Weiterhin wird auch das zwischen den Registern liegende Gatter $f$ auf allen Pfaden zu jedem Zeitpunkt als empfindlich angesehen, so dass sich alle Fehler weiter ausbreiten könnten. Wie im theoretischen Teil erklärt, hat die Reihenfolge der Triggerevents einen großen Einfluss auf das Gesamtergebnis der Analyse. Zwei oder mehr aufeinander folgende Write-Events ohne ein dazwischen liegendes Read-Event führen zu einem Zeitblock mit unempfindlichem Registerinhalt. Nach dem Trigger-Event #4 des Registers R1 tritt kein Read-Event vor dem nächsten Trigger-Event #6 auf. In dem dazwischen liegenden Zeitraum ist der Inhalt von R1 daher sicher irrelevant. Weiterhin ist der Inhalt beider Register R1 und R2 nach den Read-Events ebenfalls irrelevant. Empfindliche Zeiträume ergeben sich daher nur, wenn ein Read- und Write-Event-Paar aufgetreten ist.

Die in Abbildung 6.1 dargestellte Schaltung wurde für ein Xilinx Virtex4-FPGA synthetisiert, platziert und verdrahtet, die Schaltungssimulation mit dem daraus erzeugten Post-Place-and-Route-Modell (PAR) und dem zugehörigen Schaltungstiming durchgeführt. Für die Analyse der Schaltung auf dem gerade betrachteten Niveau spielt aber das Schaltungstiming keine Rolle. Neben den Taktsi-

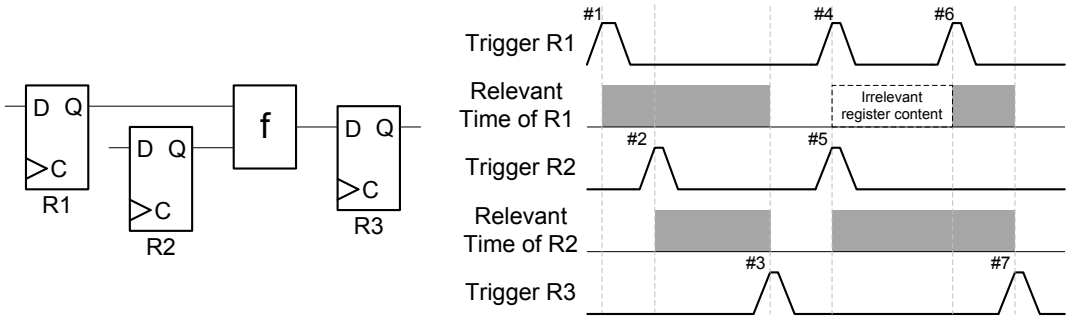



Abbildung 6.1: Grundfunktion der BA: Bestimmung der empfindlichen Zeiträume anhand von Triggerzeitpunkten

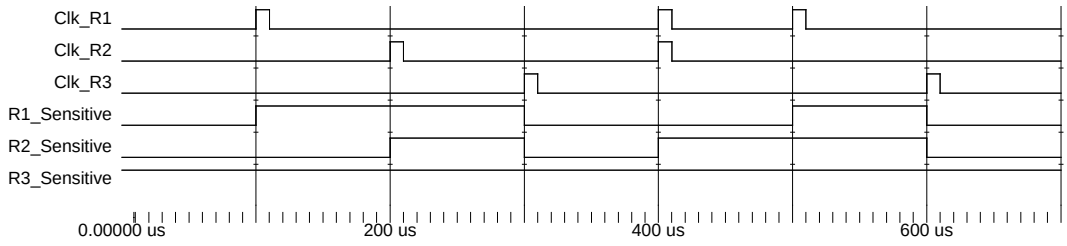



Abbildung 6.2: Ergebnisdarstellung BA in Modelsim: Signalverläufe und empfindliche Zeiträume für die Schaltung aus Abbildung 6.1

gnalen für die einzelnen Register (Clk_R1, Clk_R2, Clk_R3) sind die durch den BA-Algorithmus bestimmten Empfindlichkeitsverläufe dargestellt (R1_Sensitive, R2_Sensitive, R3_Sensitive). Der Vergleich der Abbildungen 6.1 und 6.2 zeigt eine Übereinstimmung bei den empfindlichen Zeiten der Register. Die empfindlichen Zeiträume beginnen jeweils zum Zeitpunkt der Taktflanke am Quellenregister und enden mit einer Taktflanke am Senken-Register. Einzige Ausnahme bildet erwartungsgemäß das Register R1 zwischen den Triggerevents #4 und #6, wo kein empfindlicher Zeitraum erwartet wird. Auch dieses Verhalten wird von der Implementierung des BA-Algorithmus erwartungsgemäß gezeigt.

### 6.1.2 Sequentielle Schaltung mit Verzögerungszeiten

Zur Verfeinerung der Genauigkeit der Analyseergebnisse wird das Schaltungstiming mit einbezogen. Die bereits zuvor verwendete einfache Pipeline wird nun detailliert mit dem Zeitverhalten betrachtet. Die Verzögerungszeiten können dem SDF-File entnommen werden, welches zusammen mit der Schaltungs-Netzliste das PAR-Modell bildet.

Basierend auf Abbildung 6.3 können folgende Verzögerungszeiten entnommen

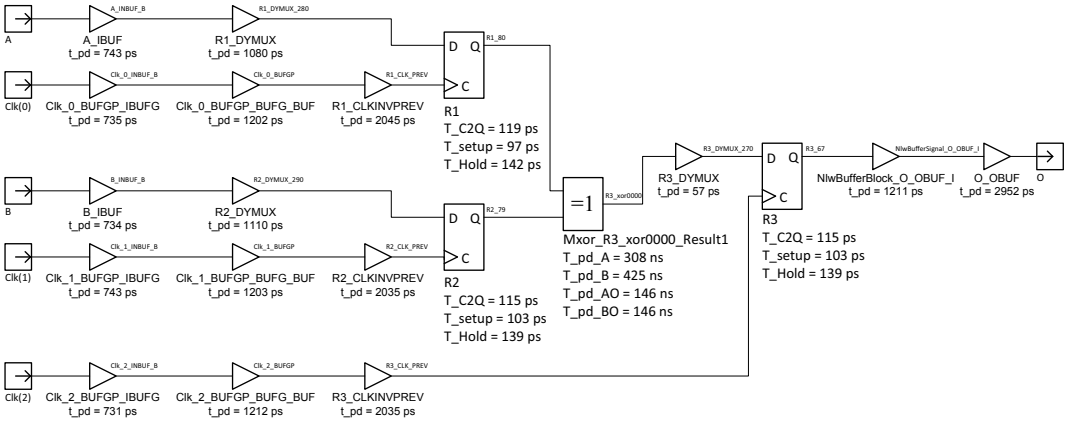



Abbildung 6.3: Einfache Pipeline: Schaltungstiming und Struktur des PAR-Modells

werden:

$$t_{Clk(0)\rightarrow R1\_CLK\_PREV} = 3982ps \quad (6.1)$$
$$t_{Clk(1)\rightarrow R2\_CLK\_PREV} = 3981ps \quad (6.2)$$
$$t_{Clk(2)\rightarrow R3\_CLK\_PREV} = 3978ps \quad (6.3)$$
$$t_{R1\_80\rightarrow R3\_DYMUX\_270} = 511ps \quad (6.4)$$
$$t_{R2\_79\rightarrow R3\_DYMUX\_270} = 628ps \quad (6.5)$$

Mit Hilfe der Gleichungen 4.11 und 4.12 sowie einer Taktperiode von 100 $\mu$s können sowohl Länge als auch die Anfangszeitpunkte der empfindlichen Zeiträume für R1 und R2 rechnerisch bestimmt werden:

$$t_{WriteEvent,R1}(n) = n \cdot 100\mu s + 3982ps + 119ps \quad (6.6)$$
$$= n \cdot 100\mu s + 4101ps \quad (6.7)$$
$$t_{WriteEvent,R2}(n) = n \cdot 100\mu s + 3981ps + 115ps \quad (6.8)$$
$$= n \cdot 100\mu s + 4096ps \quad (6.9)$$
$$t_{ReadEvent,R3\rightarrow R1}(n) = n \cdot 100\mu s + 3978ps + 139ps - 511ps \quad (6.10)$$
$$= n \cdot 100\mu s + 3606ps \quad (6.11)$$
$$t_{ReadEvent,R3\rightarrow R2}(n) = n \cdot 100\mu s + 3978ps + 139ps - 628ps \quad (6.12)$$
$$= n \cdot 100\mu s + 3489ps \quad (6.13)$$

Die Länge der empfindlichen Zeiträume wird durch die Differenz der Zeitpunkte der Read- und Write-Events bestimmt. Die folgende Rechnung stellt das Ergebnis exemplarisch für Triggerevents am Quellen- und Senkenregister im Abstand von 100 $\mu$s dar. Im Beispiel aus Abbildung 6.1 kommt noch ein weiterer empfindlicher Zeitraum pro Register vor, dieser ist exakt um 100 $\mu$s länger (siehe

```
Register output   /testbench/uut/R1/O   Active time:  299999010   Rate Sum          2   ReconvRate    0
   Duration     99999505   Rate   1   Reconv   0   First occurence from      99996394   to     199995899
   Duration    199999505   Rate   1   Reconv   0   First occurence from     399996394   to     599995899

Register output   /testbench/uut/R2/O   Active time:  299998786   Rate Sum          2   ReconvRate    0
   Duration     99999393   Rate   1   Reconv   0   First occurence from     399996511   to     499995904
   Duration    199999393   Rate   1   Reconv   0   First occurence from      99996511   to     299995904
```

Abbildung 6.4: Darstellung des Analyseergebnis in Textform für die einfache Pipeline (Abb. 6.1)

Analyseergebnisse in Abbildung 6.4).

$$
\begin{aligned}
t_{SensitiveTimeR1} &= t_{ReadEvent,R3\rightarrow R1}(n+1) - t_{WriteEvent,R1}(n) && (6.14)\\
&= (n+1)\cdot 100\mu s + 3606ps - (n\cdot 100\mu s + 4101ps) && (6.15)\\
&= 100\mu s - 495ps = 99999595ps && (6.16)\\
t_{SensitiveTimeR2} &= t_{ReadEvent,R3\rightarrow R2}(n+1) - t_{WriteEvent,R2}(n) && (6.17)\\
&= (n+1)\cdot 100\mu s + 3489ps - (n\cdot 100\mu s + 4096ps) && (6.18)\\
&= 100\mu s - 607ps = 99999393ps && (6.19)
\end{aligned}
$$

Die Analyseergebnisse in Abbildung 6.4 zeigen zusätzlich den Einfluss der Verzögerungszeiten auf den Taktpfaden. Aus dem höheren Wert in der Spalte "First occourence from ... to ..." kann der Wert für die individuelle Verzögerungszeit plus die Clock-to-Q-Zeit entnommen werden, wegen der zeitlichen Invertierung des Analyseablaufs verschiebt sich der Zeitpunkt jedoch nach vorne (z.B. für R1: $200\mu s - 4101ps = 199995899ps$).

Der Vergleich der theoretisch erwarteten Ergebnisse und der vom Algorithmus bestimmten empfindlichen Zeiträume stimmt überein. Es wird daher davon ausgegangen, dass auch die Ergebnisse für andere, beliebig größere Schaltungen richtig bestimmt werden.

### 6.1.3 Rekonvergente Pfade

Die Auswirkungen von rekonvergenten Pfaden werden mittels einer Schaltung mit einem vereinfachten, aber dreifach ausgeführten Datenpfad demonstriert (Abbildung 6.5). Durch die mehrfache Ausführung des Datenpfades sind die dort enthaltenen Register bei Einzelfehlern immer als unempfindlich anzusehen, da der dem Datenpfad nachfolgende Mehrheitsentscheider eine beobachtbare Auswirkung der Fehler verhindert. Zu Demonstrationszwecken wurde nahe dem Schaltungsausgang ein Maskierungsgatter (Und-Funktion) eingefügt, damit der zeitliche Versatz zwischen dem Gate-Signal und den kritischen Zuständen sichtbar wird.

Sofern durch das Maskierungsgatter Read-Events bis zum Mehrheitsentscheider gelangen, werden diese Read-Events von dort aus nur als vermutlich rekonvergente Read-Events weitergeleitet. Die Register R21, R22, R23, R31, R32 und

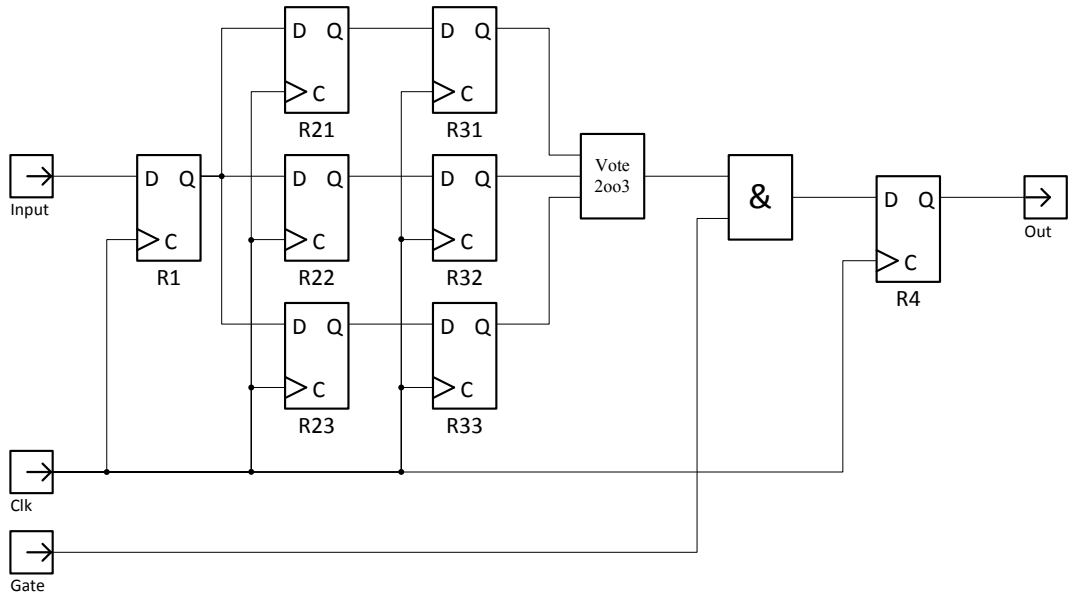



Abbildung 6.5: Einfache TMR-Schaltung mit zwei Pipelinestufen

R33 der beiden Pipeline-Stufen speichern und leiten diese besonders markierten Read-Events weiter, bis sie am Register R1 wieder zusammentreffen. Wird nun das Register R1 getriggert, so werden die zeitgleich vorhandenen Read-Events mit Rekonvergenz-Markierung gefunden und daraus ein empfindlicher Zeitraum generiert, der ohne weitere Einschränkungen weiterpropagiert werden kann.

Der zeitliche Versatz, sichtbar beim Empfindlichkeits-Signal für Register R1, wird durch die Pipelinetiefe verursacht. Der direkte Zusammenhang mit dem Gate-Signal kann durch den Vergleich der beiden Signale direkt abgelesen werden. Wegen der dreifachen Implementierung der Datenpipeline sind die in den parallelen Pfaden enthaltenen Register für sich betrachtet unkritisch, dies kann auch aus der Empfindlichkeits-Wellenform für diese Register abgelesen werden.

In der Ergebnisdarstellung als Text (Abbildung 6.7) sind nun auch in der Spalte mit der Bezeichnung "Reconv" Werte eingetragen. In diesem Beispiel wird gezeigt, dass sieben Blöcke mit empfindlicher Zeit am Register R1 gefunden wurden. Diese sieben Blöcke können in der grafischen Ergebnisdarstellung einfach gefunden werden.

## 6.2 Laufzeit- und Ergebnisvergleiche für Testschaltungen

Wesentliches Ziel bei der Entwicklung der Rückwärts-Analyse waren exakte Analyseergebnisse in Verbindung mit möglichst geringer Laufzeit. Insbesondere die Veränderung der Laufzeit bei wachsender Schaltungsgröße sollte auf ein möglichst

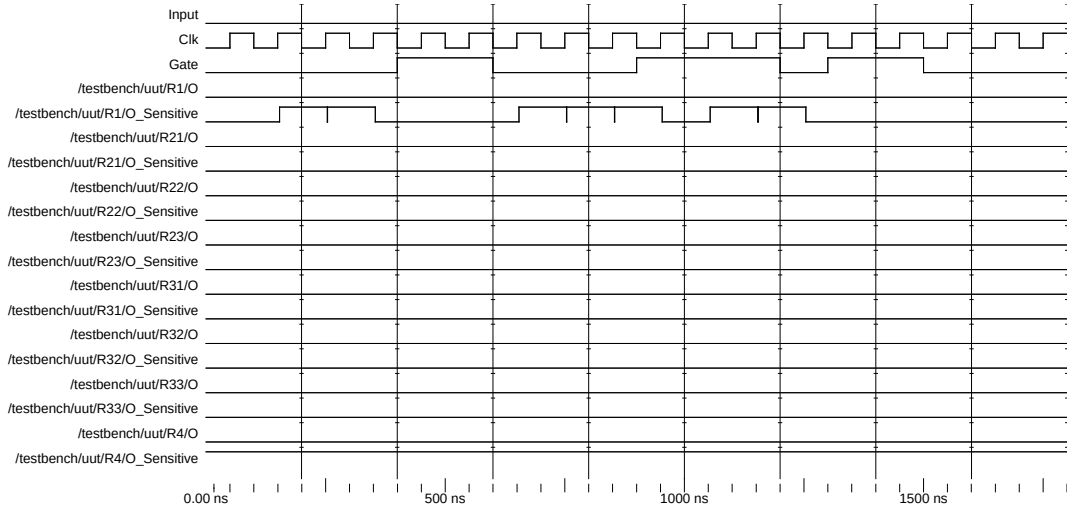



Abbildung 6.6: Signalverläufe und Analyseergebnis für die einfache TMR-Schaltung (Abbildung 6.5)

```
Register output   /testbench/uut/R1/O    Active time:      697417   Rate Sum        7   ReconvRate     7
  Duration        99631   Rate   7   Reconv   7   First occurence from        746242   to          845873

Register output   /testbench/uut/R21/O    Active time:           0   Rate Sum        0   ReconvRate     0
Register output   /testbench/uut/R22/O    Active time:           0   Rate Sum        0   ReconvRate     0
Register output   /testbench/uut/R23/O    Active time:           0   Rate Sum        0   ReconvRate     0
Register output   /testbench/uut/R31/O    Active time:           0   Rate Sum        0   ReconvRate     0
Register output   /testbench/uut/R32/O    Active time:           0   Rate Sum        0   ReconvRate     0
Register output   /testbench/uut/R33/O    Active time:           0   Rate Sum        0   ReconvRate     0
Register output   /testbench/uut/R4/O     Active time:     1945873   Rate Sum       19   ReconvRate     0
  Always relevant
```

Abbildung 6.7: Analyseergebnis für die einfache TMR-Schaltung (Abbildung 6.5)

kleines Maß begrenzt werden.

Für die Laufzeitvergleiche wurde versucht einen möglichst repräsentativen Satz an frei verfügbaren Schaltungen zu benutzen. In die engere Auswahl kamen dabei die ISCAS89-Benchmark-Schaltungen [26] [61] und die ITC99-Benchmark-Schaltungen [23]. Obwohl die ITC99-Benchmarks prinzipiell aktueller sind als die ISCAS89-Schaltungen, sind sie für die Verwendung als Laufzeitvergleich bei der Rückwärts-Analyse nicht geeignet. Dies liegt an der Konzeptionierung und Aufgabenstellung hinter diesen Schaltungen. Die ITC99-Schaltungen dienen als Benchmark für ATPG-Algorithmen [62] [63], da sie viele potentiell schwer zu testende Knoten enthalten. Weiterhin existieren für diese Schaltungen keine Testbenches, die eine sinnvolle Stimulation der Schaltungen ausführen würden, da diese Schaltungen aus unterschiedlich großen Bestandteilen anderer, teilweise kommerzieller Schaltungen zusammengesetzt werden und somit keine nützliche Funktion darstellen. Bei ersten Versuchen zur Erstellung von Testbenches für diese Schaltun-

gen mit pseudo-random Eingangssignalen hat sich herausgestellt, dass viele der Schaltungen in großen Bereichen und über lange Zeiträume nur wenig Aktivität zeigen. Deshalb lässt sich folgern, dass die Ergebnisse der Rückwärts-Analyse für diese Schaltungen nicht für Vergleiche geeignet sind.

Die ISCAS89-Schaltungen stellen eine Zusammenstellung verschiedener funktionstüchtiger Schaltungen dar, die einen großen Bereich unterschiedlicher Gatter- und FlipFlop-Anzahl abdecken. Für diese Schaltungen existieren zwar auch keine standardisierten Testbenches, das Verhalten dieser Schaltungen mit einer Stimulation durch Pseudo-Random Eingangssignale stellte sich jedoch als grundlegend anders heraus als bei den ITC99-Benchmarks.

Da ein direkter Laufzeitvergleich auch auf einen Vergleich der verwendeten Rechner für die Simulations- und Analyse-Aufgaben hinausläuft, wird versucht einen auch auf andere Rechner übertragbaren Vergleich zu finden. In den weiteren Darstellungen werden alle Laufzeitvergleiche im Verhältnis zur Laufzeit der Simulation der betrachteten Schaltung und des Anwendungsfalls gezogen. Diesem Vergleich liegt die Annahme zu Grunde, dass der verwendete Rechner prinzipiell keinen Einfluss auf das Verhältnis der Laufzeit von Schaltungssimulation und Rückwärts-Analyse hat.

Bei der Durchführung von Laufzeitmessungen dient die Systemzeit des Rechners als Zeitbasis, an drei Zeitpunkten findet die Zeitmessung statt:

- Unmittelbar nach dem Aufruf des Skripts zur Automatisierung des BA-Ablaufs
- Nach Abschluss der Initialisierungsphase (Start des Simulators, Laden des Designs, Kompilieren des Designs, Parsen der VHDL- und SDF-Dateien, Aufbau des BA-internen Schaltungsgraph)
- Nach Abschluss der BA-Teilaufgabe (Simulation oder Analyse)

Für Anwendungsfälle mit kurzer Simulationszeit stellt die Initialisierungsphase der Rückwärtsanalyse einen großen zeitlichen Anteil dar, hingegen bei langen Anwendungsfällen wird die Initialisierungsphase vernachlässigbar kurz. Um diesen verfälschenden Einfluss bei der Auswertung der Ergebnisse auszuschließen, wird das Laufzeitverhältnis auf zwei Arten berechnet, d.h. sowohl mit, aber auch ohne die Initialisierungsphase. Aus dem kompletten Satz an ISCAS89-Schaltungen wurden repräsentativ einige Schaltungen mit einer Mindestgröße ausgewählt. Der Vergleich der Schaltungseigenschaften und der benötigten Simulationszeit (Simulation von Testbench und Schaltung für eine Dauer von 10000 Taktzyklen) ist in Tabelle 6.1 dargestellt. Im Abschnitt 5.2 wurde erklärt, wie die Netzliste modifiziert werden kann um eine Beschleunigung der Simulationszeit zu erreichen bzw. den unvorteilhaften Stil des Synthesetools bei der Erstellung des Simulationsmodells auszugleichen. Die hier angegebenen Zeiten beziehen sich immer auf die Simulation der modifizierten Schaltung unter Berücksichtigung des Schaltungstimings.

In Tabelle 6.2 wird aufgezeigt, wie lange die Ausführungszeiten für die Teilschritte der BA für unterschiedliche ISCAS89-Schaltungen sind. Zusätzlich wird

| Schaltung | Eingänge | Ausgänge | Logikgatter | Register | Simulationsdauer in s |
|---|---|---|---|---|---|
| S1196 | 14 | 14 | 609 | 18 | 5 |
| S1238 | 14 | 14 | 632 | 18 | 6 |
| S1423 | 17 | 5 | 664 | 74 | 4 |
| S3330 | 40 | 73 | 1261 | 132 | 13 |
| S6669 | 83 | 55 | 1893 | 239 | 24 |
| S13207 | 62 | 152 | 2887 | 638 | 27 |
| S35932 | 35 | 320 | 9113 | 1728 | 197 |
| S38584 | 38 | 304 | 9937 | 1426 | 170 |

Tabelle 6.1: Untersuchte Beispielschaltungen aus den ISCAS89 Benchmark-Schaltungen

der Einfluss der Verbesserungsmaßnahmen am BA-Algorithmus auf die Laufzeiten dargestellt. Für den Simulationsteil lassen sich je nach Schaltung Verbesserungen zwischen 18% und 69% erzielen im Vergleich zur ursprünglichen BA (m2). Damit kann in der Variante m4 trotz Aufzeichnung der Signalverläufe die Laufzeit geringer als bei einer einfachen Schaltungssimulation mit Timing sein (vgl. Tabelle 6.1).

Für den Analyse-Teil zeichnet sich ein ähnliches Bild ab. Die Laufzeit des Analyse-Teils der BA verringert sich durch die vorgestellten Verbesserungen zwischen 31% und 84%. Die Effektivität der Verbesserungen sowohl auf den Simulationsteil als auch auf den Analyseteil steht in keinem direkten Zusammenhang mit der Größe der Schaltung. Große Verbesserungen konnten sowohl bei den kleinsten als auch den größten untersuchten Schaltungen beobachtet werden. Weiterhin kann bei der Höhe der Verbesserung kein Zusammenhang zwischen dem Simulationsteil und dem Analyseteil festgestellt werden.

Der Vergleich der Laufzeit einer kompletten BA (Simulation und Analyse zusammen) gegenüber einer einfachen Schaltungssimulation mit Timing zeigt, dass die Laufzeit für die BA um einen Faktor zwischen 2 und 18 höher ist. Abbildung 6.8 stellt die Laufzeit der Teilschritte der BA für verschiedene Schaltungen und Optimierungsstufen des Algorithmus dar. Alle dargestellten Werte für eine Schaltung sind dabei auf die Simulationszeit der optimierten Schaltungsvariante normiert (siehe Tabelle 6.1) um einen einfachen Vergleich des Rechenaufwands zu ermöglichen. Die gemessenen und berechneten Werte bestätigen die theoretischen Überlegungen zur Laufzeit des Algorithmus. Unter der Annahme, dass die Simulationszeit der Schaltung ein geeignetes Maß für die Problemgröße (Größe der Schaltung, Länge des untersuchten Usecases und "Schwierigkeitsgrad" der Simulation) ist, so bestätigt dies den mit der Problemgröße linear wachsenden Rechenaufwand (vgl. Abschnitt 4.10). Bei einer Variation der Schaltungsgröße annähernd um den Faktor 100 (bezogen auf die Anzahl der Register einer Schaltung) bleibt

| Schaltung | BA Simulation in s | | | BA Analyse in s | | |
|---|---|---|---|---|---|---|
| | m2 | m3 | m4 | m2 | m3 | m4 |
| S1196 | 9 | 6 | 4 | 56 | 34 | 9 |
| S1196_opt | 7 | 4 | 3 | 26 | 15 | 6 |
| S1238 | 10 | 6 | 5 | 57 | 36 | 9 |
| S1238_opt | 7 | 5 | 4 | 28 | 16 | 8 |
| S1423 | 9 | 7 | 4 | 53 | 39 | 10 |
| S1423_opt | 7 | 4 | 4 | 22 | 15 | 7 |
| S3330 | 25 | 16 | 12 | 189 | 121 | 40 |
| S3330_opt | 16 | 10 | 9 | 75 | 42 | 28 |
| S6669 | 49 | 25 | 16 | 375 | 159 | 73 |
| S6669_opt | 32 | 14 | 10 | 197 | 74 | 50 |
| S13207 | 53 | 53 | 42 | 335 | 302 | 121 |
| S13207_opt | 31 | 26 | 23 | 104 | 86 | 72 |
| S35932 | 321 | 278 | 246 | 1492 | 1287 | 512 |
| S35932_opt | 211 | 183 | 172 | 615 | 465 | 357 |
| S38584 | 192 | 161 | 146 | 591 | 453 | 310 |
| S38584_opt | 187 | 157 | 145 | 603 | 432 | 301 |

m2: Ursprüngliche BA
m3: Verbesserte BA ohne Simulation des Schaltungstimings
m4: Verbesserte BA wie m3 und reduzierter Anzahl stimulierter Signale

Tabelle 6.2: Ausführungszeiten für die Rückwärts-Analyse

der relative Aufwand für die BA nahezu konstant. Für die BA-Implementierung mit den meisten Optimierungen (m4 opt) pendelt sich der relative Aufwand für Simulations- und Analyseteil bei einem Faktor zwischen 2 und 4 ein.

Für eine praktische Anwendung sind die eigentlichen Analyseergebnisse jedoch von größerem Interesse als die Laufzeit des Algorithmus. Eine Aussage über die Empfindlichkeit eines einzelnen Registers bietet das Verhältnis von empfindlicher Zeit zur gesamten Zeit des betrachteten Anwendungsfalls. Dies entspricht der anfangs vorgestellten Interpretation des AVF (vgl. Formel 2.11 im Abschnitt 2.3.5). Je höher dieser Quotient, desto höher die Wahrscheinlichkeit dass ein auftretender Fehler an einem der primären Ausgänge beobachtbar wird. Damit wird die Empfindlichkeit eines jeden einzelnen Registers auf Basis eines zeitlichen Mittelwerts bewertbar. Aus Sicht der Anwendung besonders kritische Register können damit eingestuft werden und passende Schutzmaßnahmen eingefügt werden.

Sofern nur die Gesamt-Empfindlichkeit der Schaltung bewertet werden soll, bietet sich eine andere Darstellung an. In einem Diagramm werden die AVFs aller Register in sortierter Reihenfolge dargestellt (Abbildung 6.9 und Abbildung 6.10).

Diese Grafiken können dabei in vielfältiger Weise interpretiert werden. Bei

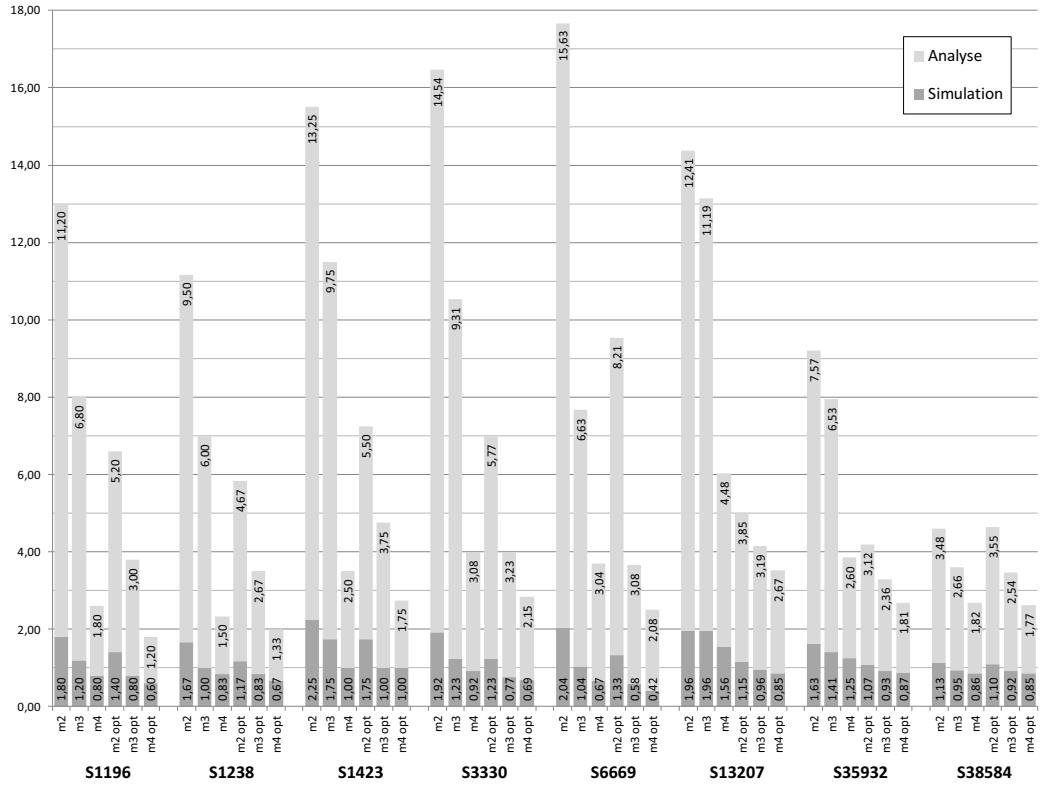



Abbildung 6.8: Laufzeit des BA-Algorithmus in normierter Darstellung

entsprechender Skalierung der Achsen stellt die Fläche unter dem Graphen den Mittelwert des AVFs für die gesamte Schaltung dar (siehe Formel 2.11). Von links nach rechts gelesen, je früher der Graph von AVF 0% nach oben ansteigt, desto empfindlicher ist die Schaltung im Mittelwert. Durch die flächenhafte Darstellung können auch Vergleiche innerhalb einer Schaltung gemacht werden. Am Beispiel der Schaltung S1196 (Abbildung 6.11) kann abgelesen werden, dass ca. 80% der Register einen AVF kleiner als 20% haben. Der mittlere AVF aller Register der Schaltung ist 0,142. Anteilig entfallen 25% (0,035) davon auf die 80% der Register mit AVF kleiner als 20%, umgekehrt entfallen 75%(0,107) auf die 20% der Register mit hohem AVF. Durch diese Betrachtungsweise kann sehr einfach der Aufwand und das Potential bei einer "Härtung" der Schaltung abgeschätzt werden. Unter den bisher vorgestellten Ergebnissen bietet also die Schaltung S6669 das geringste Potential für eine kostengünstige Härtung der Schaltung, da hier für sehr viele Register ein relativ hoher AVF festgestellt wurde.

Die Abbildungen 6.9 und 6.10 sowie die Tabelle 6.2 liefern einen Zusammenhang für die Laufzeit der BA. Schaltungen, für die ein höherer mittlerer AVF festgestellt wurde benötigen im Allgemeinen auch einen größeren Aufwand zur Analyse. Dies hängt direkt mit der Anzahl der empfindlichen Zeitblöcke zusammen bzw. wie viele Maskierungseffekte in der Schaltung wirksam sind.

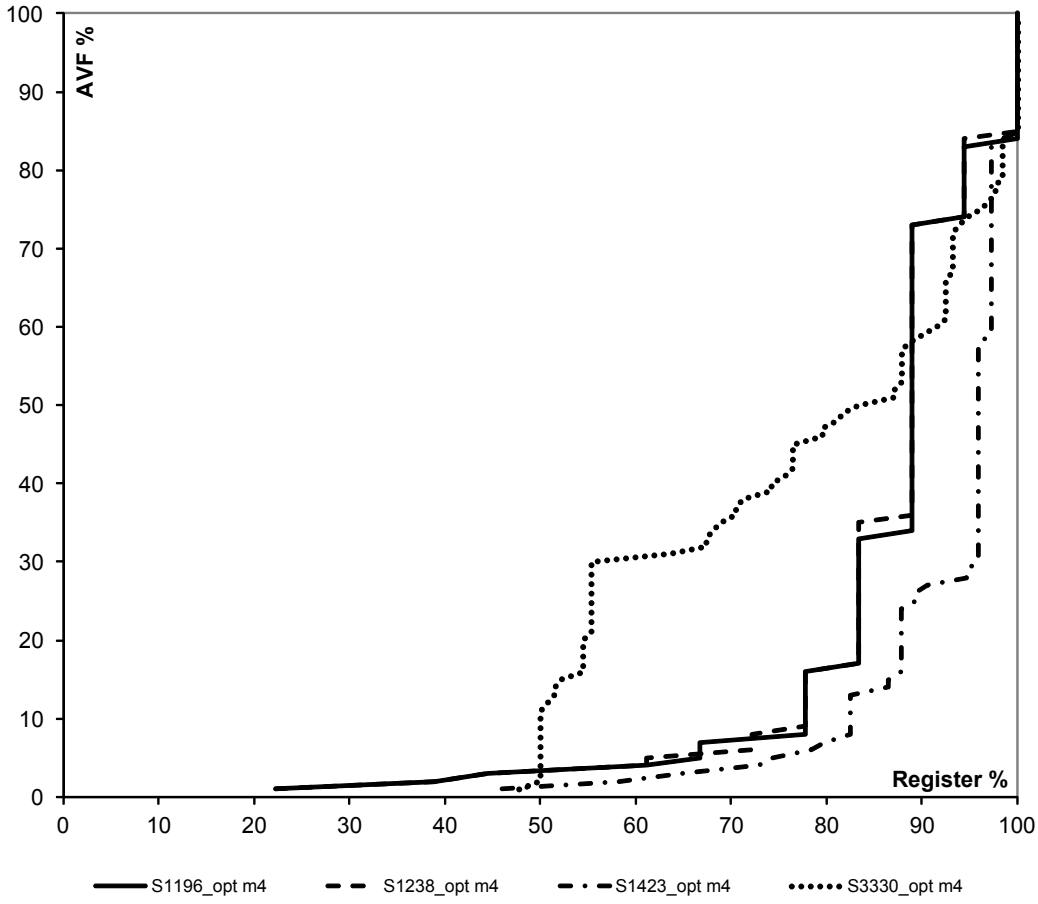



Abbildung 6.9: Verteilungsfunktion für AVF-Werte verschiedener kleiner ISCAS89-Schaltungen

Besonders detaillierte Ergebnisse können für jede Schaltung durch eine spezielle Vorgabe bei der Durchführung des Algorithmus erhalten werden. Für die bisher vorgestellten Ergebnisse wurden in jeder Schaltung immer alle primären Ausgänge als beobachtbar angesehen. Jedoch kann die BA auch durchgeführt werden, wenn nur eine Teilmenge der Ausgänge oder auch interne Signale als beobachtbar angesehen wird. Dadurch können die individuellen Abhängigkeiten auf jedes dieser Signale bestimmt werden. Die Ergebnisse eines möglichen Anwendungsszenarios sind in Abbildung 6.12 dargestellt. Die BA wurde für jeden Ausgang separat (insgesamt 14 Durchläufe) und zusätzlich für alle Ausgänge zusammen durchgeführt. In dieser Grafik ist einfach erkennbar, dass Fehler in einzelnen Registern meist nur an wenigen Ausgängen beobachtbar sind. Die größten Auswirkungen haben beispielsweise das Register dff12 auf den Ausgang outp00 und Register dff16 auf den Ausgang outp09.

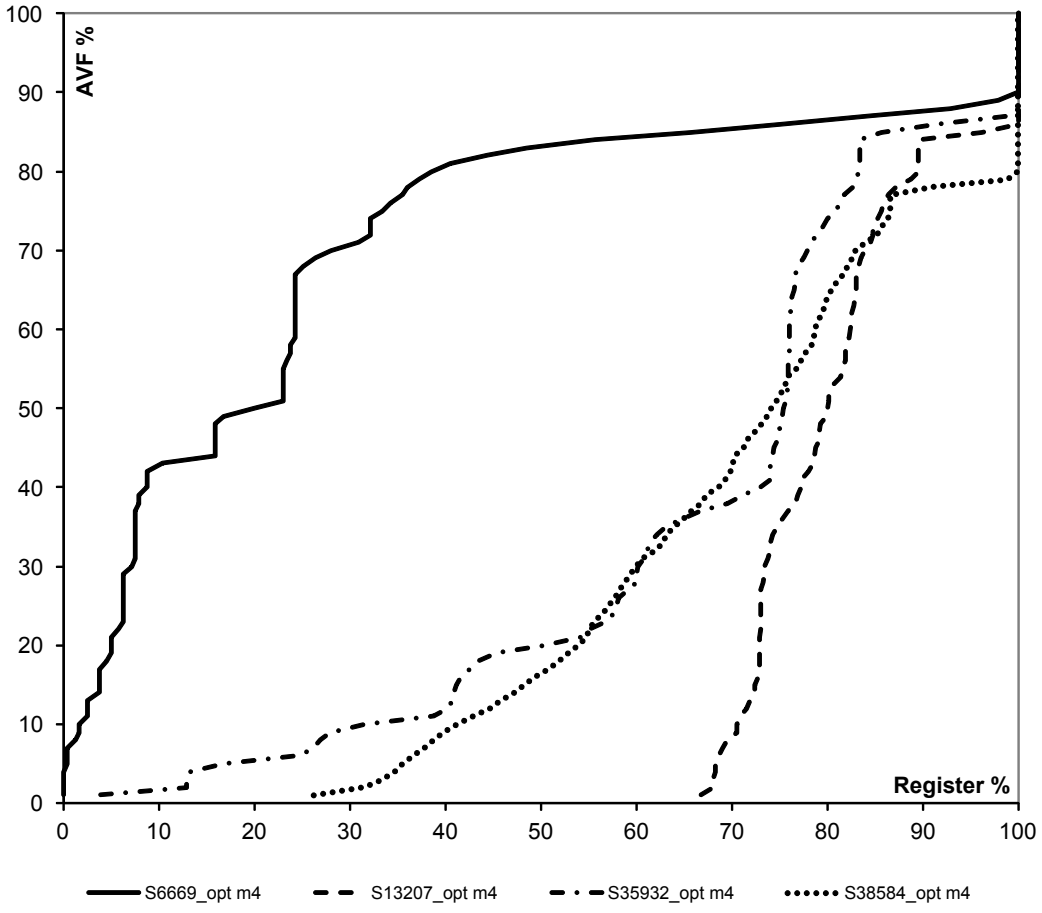



Abbildung 6.10: Verteilungsfunktion für AVF-Werte verschiedener großer ISCAS89-Schaltungen

Fehler an dff05 sind an insgesamt vier Ausgängen beobachtbar, jedoch sind nicht immer alle vier Ausgänge bei einem Fehler betroffen. Durch den Vergleich mit dem Analyseergebnis für alle Ausgänge wird klar, dass Fehler an dff05 meistens nicht an allen Ausgängen gleichzeitig beobachtbar sind, da der AVF-Wert für einzelne Ausgänge wesentlich kleiner als für alle Ausgänge zusammen ist. Eine entgegengesetzte Situation kann an dff04 ausgemacht werden. Hier sind die AVF-Werte für die einzelnen Ausgänge nahezu gleich dem Wert für alle Ausgänge zusammen. Daher kann ohne weitere Untersuchung angenommen werden, dass Fehler hier fast immer gleichzeitig beobachtet werden können. Diese Art der BA mit speziellen Gruppen von beobachtbaren Signalen kann wichtige Ergebnisse liefern, insbesondere dann, wenn aus der Sicht einer Anwendung eben nicht alle Ausgangssignale gleich wichtig sind, sondern zentrale Funktionen eines Systems steuern (z.B. Reset für externe Komponenten).

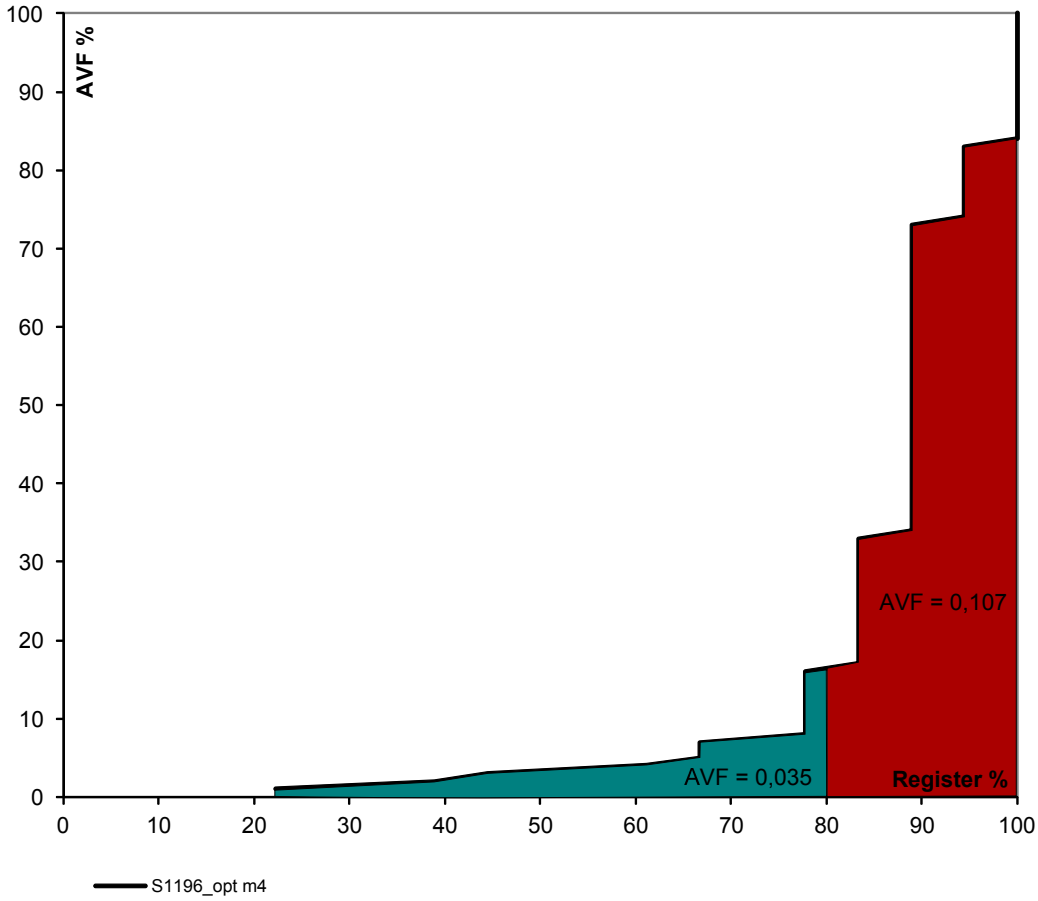



Abbildung 6.11: Vergleich der Anteile am gesamten AVF der Schaltung S1196

## 6.3 Erweiterung für Block-Speicher

Bisher wurden mit der BA ausschließlich Schaltungen untersucht, die als Speicherelemente einzelne Register enthalten. Diese Register sind als Grundbaustein für jede sequentielle Schaltung unerlässlich, jedoch werden in modernen Schaltungen Speicher auch häufig in großen Blöcken eingesetzt. Selbstverständlich könnten große Speicher-Arrays auch aus einzelnen Registern synthetisiert werden, häufig sind diese Speicherblöcke aber von Hand hochgradig optimiert und daher nicht mit Primitiv-Elementen sinnvoll darstellbar. Bei FPGAs werden die Speicherblöcke für vielfältige Aufgaben in synthetisierten Schaltungen verwendet (z.B. für FIFOs, Caches, Dual-Port-RAMs) und sind in vielen Teilaspekten konfigurierbar (z.B. Speichertiefe, Busbreite, Eingangs- und Ausgangsregister), der eigentliche Speicherblock ist aber unveränderlich.

Ein weiterer Aspekt ergibt sich durch die Größe der Block-Speicher. In einem Mikroprozessor oder DSP-Schaltung können leicht mehrere Hundert kbit Block-

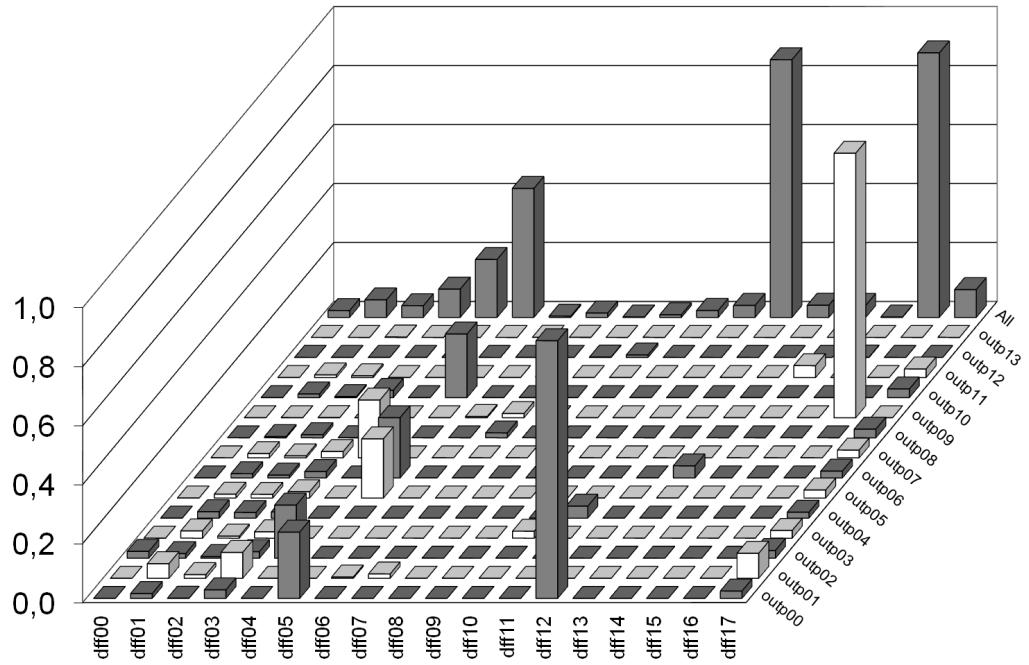



Abbildung 6.12: BA-Ergebnisse für jeweils einen einzelnen beobachtbaren Ausgang am Beispiel S1196

Speicher verwendet werden, wohingegen die Anzahl der einzelnen Register auf mehrere Tausend beschränkt bleibt (siehe Tabelle 3.1). Die Register haben häufig eigene Takt- und Enable-Signale, bei einem Block-Speicher werden häufig aber nur wenige Takt- und Enable-Signale für den ganzen Block verwendet. Zudem führt die Adressierung bei Block-Speichern dazu, dass zu einem Zeitpunkt immer nur wenige Speicherzellen gelesen oder beschrieben werden können und der Großteil der Speicherzellen lediglich seinen Zustand hält. Schon aus Effizienz-Gründen erscheint es daher sinnvoll die Blockspeicher auch als Ganzes in der BA zu behandeln.

Um eine möglichst universelle und flexible Beschreibung von Block-Speichern für die BA zu ermöglichen, müssen die Grundbausteine dieser Speicher zuerst detailliert untersucht werden.

Bei Block-Speichern lassen sich mehrere funktionale Gruppen einfach erkennen:

- Eingangsports
- Ausgangsports
- Trigger-Funktion: Takt- und Enable-Ports
- Daten-Adressierung: Adress-Ports
- Zentraler Speicher

Bei der Integration der Block-Speicher in die BA sollen nun alle Funktionen, die bisher in einem einzelnen Register implementiert wurden, auf die oben genannten

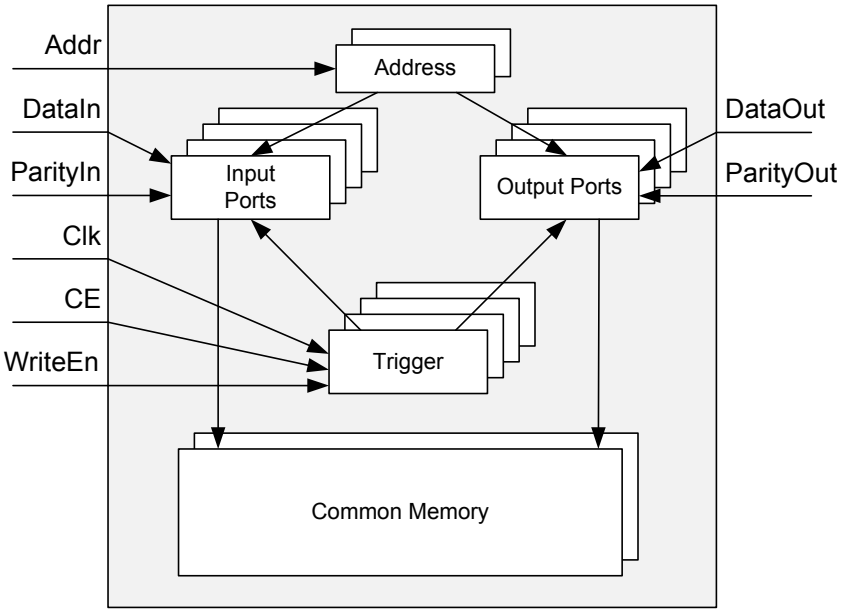



Abbildung 6.13: Funktionsblöcke zur Integration eines Blockspeichers in die BA

Funktionsgruppen verteilt werden (Abbildung 6.13) und wenn nötig zusätzliche Funktionen für die BA implementiert werden.

Wesentliches Merkmal sind die komplexeren Trigger-Funktionen der Block-Speicher. Trigger-Funktionen können separat für die Eingangs- und Ausgangsseite sein und können auch nur Teilen der gesamten verfügbaren Eingangs- und Ausgangsports zugewiesen sein (z.B. bei einem 32 bit breiten Speicher 4 mal 8 bit). Neben dem Taktsignal und dem CE-Signal müssen häufig noch zusätzliche Signale zur Bestimmung einer Trigger-Bedingung untersucht werden (z.B. Write-Enable WE). Daher wurde hier ein flexibles Modell unter Verwendung von konfigurierbaren LUTs implementiert mit dem beliebige Logikfunktionen zur Generierung von Trigger-Events definiert werden können. Von einem Trigger-Block können beliebig viele Eingangs- und Ausgangsports abhängig sein.

Die zusätzlich implementierten Adress-Blöcke dienen dazu die einzelnen Werte aller spezifizierten Adress-Signale zusammenzufassen und für die weiteren Funktionen eine Adresse als Dezimalwert zur Verfügung zu stellen. Ein Adress-Block kann ebenfalls mit beliebig vielen Eingangs- und Ausgangs-Blöcken verknüpft werden. Abhängig vom Speicher können auch mehrere unabhängige Gruppen von Adress-Signalen vorhanden sein (z.B. bei Dual-Port-RAMs), was auch mehrere Adress-Blöcke erforderlich macht.

Wie bei einem einfachen Register enthält der zentrale Speicher Listen für die Read- und Write-Events. Diese Listen sind jedoch für jede Speicherzelle separat ausgeführt, damit eine Zuordnung auf das jeweilige Speicherbit erfolgen kann. Da die Adressierung des Speichers mit binären Schaltungssignalen erfolgt, muss

die Speichergröße eine Potenz von zwei sein. Block-Speicher mit unterschiedlicher Größe (z.B. 2k x 9 bit) müssen daher in zwei oder mehr zentrale Speicherblöcke aufgeteilt werden.

Die Ein- und Ausgangsports haben bei den Blockspeichern die wesentliche Aufgabe die Read- und Write-Events den jeweiligen Speicherzellen zuzuordnen. Jedem Port wird ein Offset-Wert zugewiesen mit dessen Hilfe und der dezimalen Adresse die betroffene Speicherzelle im zentralen Speicherblock bestimmt werden kann. Ein wortweiser Zugriff kann somit in einzelne Bit-weise Zugriffe umgewandelt werden.

Die Implementierung der Block-Speicher-Analyse wurde an einfachen Testschaltungen (z.B. FIFO) getestet um zuverlässige Ergebnisse auch für komplexe Schaltungen zu ermöglichen.

Als anspruchsvolle zu analysierende Schaltung wurde ein komplettes Mikroprozessorsystem (CPU, Cache, Peripherie) basierend auf dem LEON3-Core [64] ausgewählt. In der verwendeten Konfiguration benötigt dieses System nach der Synthese für ein Virtex4-FPGA folgende Ressourcen:

- 8301 Look-Up-Tables mit vier Eingängen, insgesamt 25649 Logikelemente
- 2568 Register
- 15 kleine Schieberegister (16 bit)
- 18 DualPort BlockRAMs mit jeweils 18 kbit Größe

Zusammen mit den Registern in den Eingangs- und Ausgangsports des Systems sind in dieser Schaltung 330944 bit Speicher vorhanden. Mit dieser Schaltung soll die Funktion der BA für wesentlich größere Schaltungen als die bisher vorgestellten demonstriert werden. Die gewonnenen Ergebnisse sollen sowohl die Skalierbarkeit des Algorithmus als auch für eine Detail-Analyse herangezogen werden. Da es sich um eine Schaltung mit CPU handelt, wird der Anwendungsfall auch durch das auszuführende Programm im Speicher der Schaltung definiert.

Für diesen Test wurde ein Programm entworfen, das auf dem simulierten Leon3-System ausgeführt wird und sich besonders zu Demonstrationszwecken eignet (Abbildung 6.14). Das Programm wurde in der Programmiersprache C geschrieben und mit dem gcc-4.4.2-Cross-Compiler für SPARC V8 Architekturen kompiliert und gelinked. Die erzeugte ausführbare Datei wurde zur Initialisierung der in der Testbench vorhandenen Speicher (SRAM und SDRAM) verwendet.

Die Leon3-Hardware wurde zusammen mit dieser Software für die Dauer von 600 $\mu$s (30000 Takte bei 50 MHz Taktfrequenz) simuliert und analysiert. Wegen der Größe der Schaltung wurde bei der BA die Optimierungsstufe m4 verwendet. Zusätzlich dazu wurde zu Vergleichszwecken eine Simulation mit Einfluss des Schaltungstimings durchgeführt. Diese Simulation wurde in 756 Sekunden durchgeführt. Im Vergleich dazu benötigte der Simulationsteil (m4: ohne Schaltungstiming) 758 Sekunden und der Analyseteil 2161 Sekunden. Ähnlich wie in Abbildung 6.8 stellt sich ein Verhältnis zwischen der Ausführungszeit der BA und der einfachen Schaltungssimulation von 3,86 zu 1 heraus. Die BA ist damit auch in Schaltungen mit Blockspeichern effizient einsetzbar.

```
int main()
{
    GPIO_Init();
    int cnt = 0;
    while(1)
    {
        int Sum = 0;
        int i = 0;
        for(; i < 100; ++i)
        {
            Sum += i;
        }
        if(Cnt % 2)
        {
            save(&GPIO->Data, Sum);
        }
        else
        {
            save(&GPIO->Data, Cnt);
        }
        Cnt++;
    }
    return 0;
}
```

Abbildung 6.14: Einfaches Testprogramm für das Leon3-System

Ein wesentlicher Grund für den geringen Einfluss der großen Speichermenge ist die begrenzte Zugriffsmöglichkeit der Schaltung darauf. In einer Schaltung in der ausschließlich einzelne Register verwendet werden, kann auf einen großen Teil dieser Register jeden Taktzyklus lesend und schreibend zugegriffen werden. Die Logikmaskierung bezogen auf die Anzahl der unempfindlichen Pfade ist relativ niedrig. Ein Blockspeicher wie im vorgestellten Leon3-Design hat dagegen eine Speicherorganisation von z.B. 512 x 36 bit. Damit ist automatisch nur eine von 512 Speicherzellen zu einem Zeitpunkt zugreifbar. Dementsprechend gering fällt der Rechenaufwand zur Analyse dieser Speicher im Verhältnis zur Speichermenge aus.

Abbildung 6.15 stellt die Analyseergebnisse in gewohnter Form dar. Der Graph ”Leon3 Complete” zeigt, dass der allergrößte Teil der vorhandenen Speicherzellen für die Funktion der Schaltung irrelevant ist. Dies kann auf zwei Arten begründet sein. Erstens kann es bei der Schaltungssynthese vorkommen, dass nicht der komplette Speicher verwendet werden kann. Dies zeigt sich durch nicht verwendete Ein- und Ausgangspins und Einschränkungen bei den Adresssignalen. Da das Leon3-System für ein FPGA synthetisiert wurde, kann hier die Größe des integrierten Speichers nicht genau dem Bedarf angepasst werden, sondern es müssen die vorgefertigten Blöcke verwendet werden. Der zweite Grund für die große

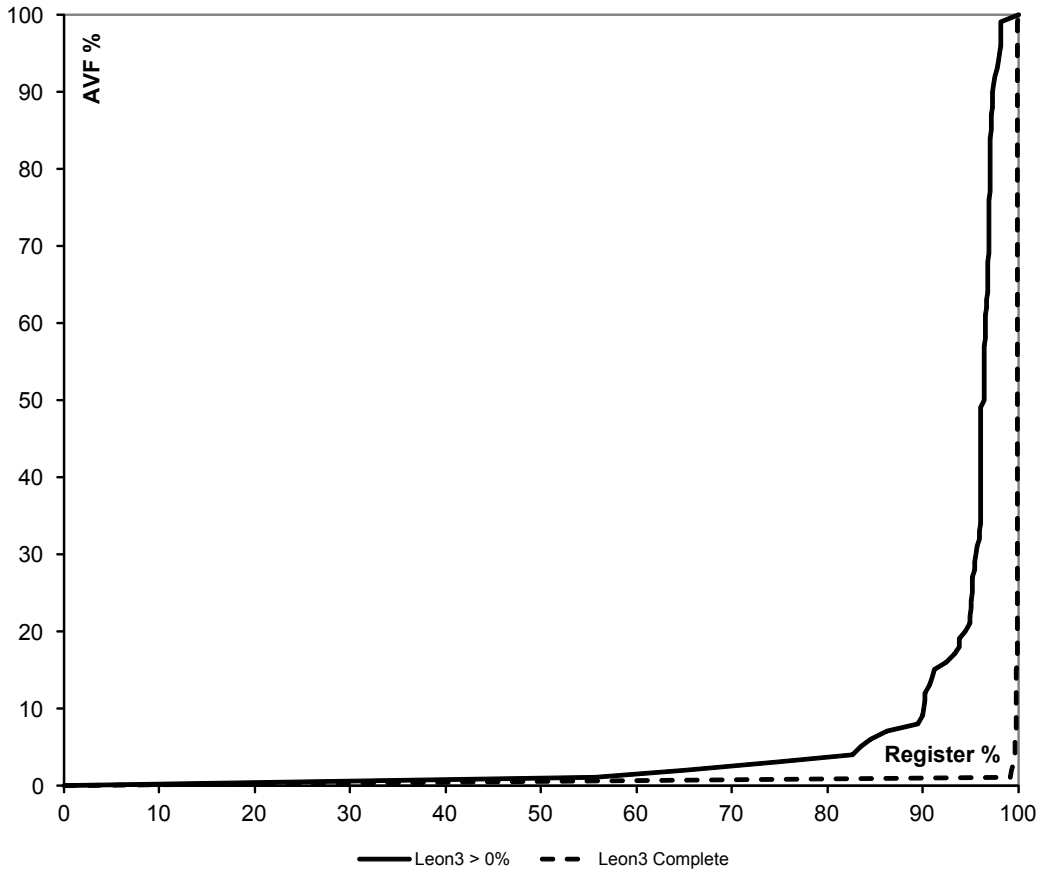



Abbildung 6.15: Verteilungsfunktion der AVF-Werte für das Leon3-System

Anzahl der unempfindlichen Speicherzellen ist das jeweilige Anwendungsszenario. Da das ausgeführte Programm natürlich nur einen Bruchteil der gesamten Funktionalität verwendet und auch der Programmcode relativ klein ist, d.h. der CPU-Cache wird nicht vollständig benötigt, werden die allermeisten Speicherzellen nicht verwendet. Die Analyse findet für diesen Anwendungsfall für 317754 Speicherzellen (ca. 96% des gesamten Speichers in der Schaltung) einen AVF von Null.

Zur besseren Visualisierung wurde in die Abbildung 6.15 ein zweiter Graph "Leon3 $>$ 0%" eingefügt. Diese Verteilungsfunktion der AVF-Werte stellt alle 13190 Speicher mit einem AVF größer als Null dar. Innerhalb dieser reduzierten Speichermenge haben 10% der Speicher einen AVF von größer 10%, 5 % der Speicher einen AVF größer als 22% und 3% der Speicher einen AVF größer als 80%.

## 6.4 Verifikation mittels Fehlerinjektion

Für kleine Schaltungen können die Ergebnisse der Rückwärts-Analyse durch einfache Überlegungen hergeleitet werden, bei größeren Schaltungen ist dies jedoch auf Grund der wachsenden Komplexität nicht mehr möglich. Um aber gerade die Ergebnisse für größere Schaltungen verifizieren zu können, wurde im Rahmen einer Bachelor-Arbeit ein Fehlerinjektions-Framework entwickelt und Vergleichsversuche durchgeführt [65]. Im Abschnitt 2.3.1 wurden die Voraussetzungen hierfür erklärt.

Ein direkter Vergleich zwischen Ergebnissen der BA und der Fehlerinjektion wurde für die ISCAS89-Schaltung S1196 durchgeführt. Dabei wurde ein Anwendungsfall mit einer Taktrate der Schaltung von 100 MHz simuliert. Über einen Zeitraum von 500 ns wurden im Abstand von 1 ns auf jedes der Register Fehler injiziert, die Beobachtungszeit wurde zu 50 ns gewählt. Mit den insgesamt 18 Registern wurden damit für diese Schaltung 9000 Tests durchgeführt. Der Testabstand wurde in diesem Fall sehr klein gewählt, um mit der BA vergleichbare Ergebnisse zu erhalten. Die gesamte Testzeit betrug 913 Minuten.

Bei einer Vergleichsrechnung auf Basis der untersuchten Taktzyklen zeigt sich ein sehr großer Vorteil zugunsten der BA. Die Analysezeit für einen Taktzyklus mit der Fehlerinjektion beträgt 1095 s, mit der BA kann ein Taktzyklus je nach Optimierungsgrad zwischen 6,5 ms (m2: 56 s + 9 s für 10000 Takte) und 1,3 ms (m4: 4 s + 9 s für 10000 Takte) verarbeitet werden. Die Ausführungszeit für beide Analysemethoden unterscheidet sich damit um etwa fünf bis sechs Größenordnungen. Es muss jedoch hinzugefügt werden, dass bei der Fehlerinjektion zusätzliche, nicht aufgeführte 14 Ausgangsregister getestet wurden und als Zusatzergebnis die Fehlerlatenz sowie die betroffenen Ausgänge bestimmt werden könnten.

In diesem Testszenario wurde der Testabstand so klein gewählt, dass innerhalb einer Taktperiode zehn Testpunkte liegen. Durch diesen kurzen Abtastabstand ist aus den Ergebnissen der Fehlerinjektion ein Empfindlichkeitsverlauf rekonstruierbar, der dem durch die BA bestimmten Verlauf recht nahe kommt, d.h. prinzipiell sollten alle Blöcke mit empfindlicher Zeit auch durch die Fehlerinjektion erkannt werden. Bei einem direkten Vergleich zwischen Fehlerinjektion und BA wurden unter Berücksichtigung des Testabstandes identische Ergebnisse bestimmt. Die Fehlerinjektion bestätigt für dieses Testszenario damit die Richtigkeit der Ergebnisse der BA für kleinere Schaltungen.

Das Analyse-Ergebnis für das Leon3-System wurde mittels der Fehlerinjektion ebenfalls verifiziert. Da in der verwendeten Testbench im Zeitraum vom Simulationsstart (0 $\mu$s) und 500 $\mu$s nur die Initialisierung der CPU durchgeführt wird, wurde der Beginn der Fehlerinjektion auf 500 $\mu$s gesetzt. Von diesem Zeitpunkt an wurden auf sechs ausgewählte Register in der Pipeline der CPU im Abstand von 11 ns Fehler injiziert, da hier die Ausführung des zuvor erwähnten Testprogramms beginnt. Die Beobachtungszeit nach dem Zeitpunkt der Fehlereinspeisung betrug

dabei 20 $\mu$s, was einer max. Fehlerlatenz von 1000 Taktzyklen entspricht. In dieser Fehlerinjektions-Kampagne wurden Fehler an 9424 Zeitpunkten eingespeist, damit wurden insgesamt 56544 Tests durchgeführt.

In den Arbeiten von Mukherjee [15] [66] und Wang [17] werden für ein CPU-System maximale Fehlerlatenzen von mehreren 10000 Zyklen angenommen. Auf Grund der Kenntnis des ausgeführten Programms auf der CPU und durch vorangegangene Tests hat sich in diesem Fall die wesentlich geringere Fehlerlatenz von 1000 Taktzyklen als ausreichend herausgestellt. Eine vergrößerte Fehlerlatenz hätte auf Grund des höheren Simulationsaufwands annähernd linearen Einfluss auf die Ausführungszeit der Fehlerinjektion.

Obwohl großen Wert auf Stabilität und Zuverlässigkeit des Fehlerinjektions-Framework gelegt wurde, kann es nach einer großen Anzahl an Tests zu einer Fehlfunktion des Simulators kommen. Ohne zusätzliche Vorkehrungen würde dies zu einem Verlust von Testergebnissen führen, was aber durch besondere Vorkehrungen bei der Implementierung verhindert wurde. Das Gesamtergebnis setzt sich aus mehreren Teilabschnitten zusammen und die Gesamtzeit ist daher nur schwer zu bestimmen. Ein Teilabschnitt mit 9084 Tests im Zeitraum von ca. 503 $\mu$s bis 520 $\mu$s wurde in ca. 160 Stunden bearbeitet. Damit ergibt sich ein Mittelwert von 63 Sekunden pro Test.

Wegen der stark unterschiedlichen Qualität der Ergebnisse im Vergleich zwischen Fehlerinjektion und BA kann hier nicht der identische Vergleich wie bei Schaltung S1196 gezogen werden. Mit der BA wurden in 2919 Sekunden 10761268 Blöcke mit empfindlicher Zeit gefunden, jedoch auch ca. $9,9 \cdot 10^9$ Taktzyklen für alle Speicher einzeln gesehen (330944 Speicher für je 30000 Takte) untersucht. Diese sehr große Zahl wäre vergleichbar mit der nötigen Anzahl an Test mittels Fehlerinjektion um ein vergleichbares Ergebnis zu erhalten. Folgende mittlere Zeitaufwände ergeben sich für beide Ansätze bei der BA:

- 0,27 ms pro gefundenen Block empfindlicher Zeit
- 0,29 $\mu$s pro untersuchtem Taktzyklus und Speicher

Nachdem mit der BA auch ein genauer Empfindlichkeitsverlauf bestimmt werden kann, soll dieser Verlauf auch durch die einzelnen Tests bei der Fehlerinjektion verifiziert werden. Die Testergebnisse der Fehlerinjektion wurden in eine Wellenform-Datei umgewandelt und von Hand direkt mit den BA Ergebnissen verglichen. Ein direkter Vergleich kann natürlich nur bei den Signalen gemacht werden, für die Testergebnisse mit der Fehlerinjektion vorhanden sind.

In Abbildung 6.16 und 6.17 sind Signalverläufe aus der Simulation des Systems zusammen mit den Analyseergebnissen aus der BA dargestellt. Die sechs ausgewählten Register, die mit der Fehlerinjektion getestet wurden, sind jeweils ein Bit des PC-Registers (Program Counter) in den Pipeline-Stufen des Leon3 (Abbildung 6.18). Die Register sind von oben nach unten in der gleichen Reihenfolge angeordnet, wie sie in der Pipeline durchlaufen werden.

Ein Signal mit dem Suffix _Sensitive stellt das Ergebnis der BA dar, Ergebnisse aus der Fehlerinjektion haben das Suffix _FI. Da die Fehlerinjektion nur

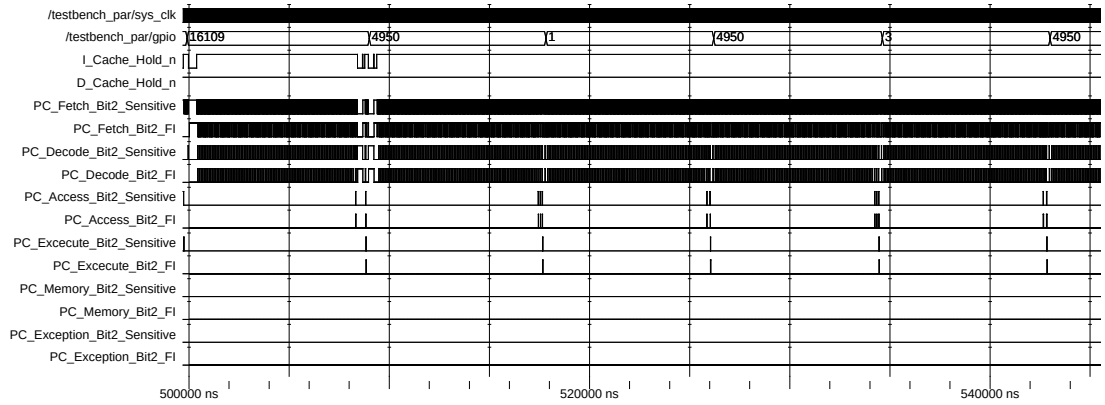



Abbildung 6.16: Signalverläufe für das Leon3-Testsystem, Ergebnisse der BA und Vergleichswerte aus der Fehlerinjektion

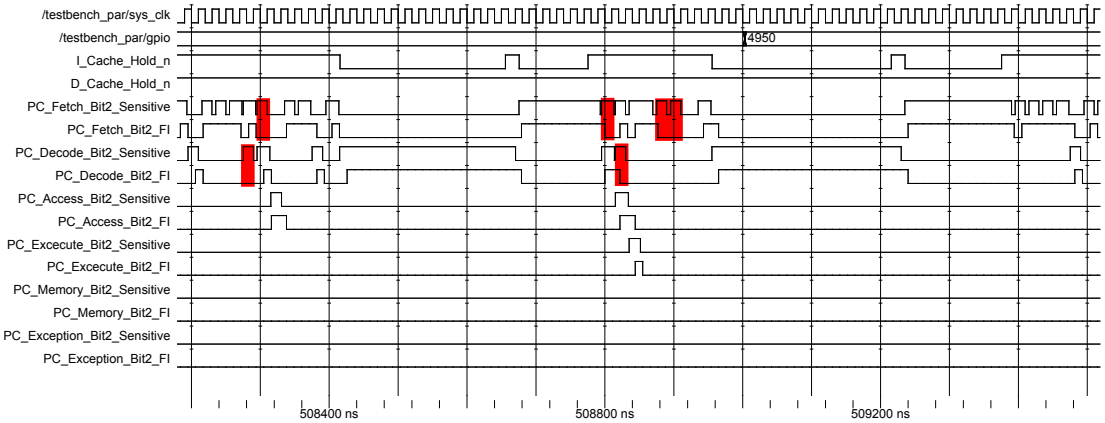



Abbildung 6.17: Detaillierter Vergleich BA/Fehlerinjektion für die PC-Register im Zeitbereich 508,2$\mu$s bis 509,7$\mu$s

Ergebnisse für einzelne, diskrete Zeitpunkte liefert, nicht jedoch Ergebnisse für kontinuierliche Zeiträume, muss bei der Betrachtung der Ergebnisse hier besondere Sorgfalt gezeigt werden. In einer vcd-Datei wird für jeden Datenpunkt nur der zukünftige Signalwert definiert. Für die Darstellung der Ergebnisse der Fehlerinjektion bedeutet dies, dass das Ergebnis für einen Testpunkt durch den Signalwert unmittelbar nach einem Pegelwechsel dargestellt wird. Sofern zwei oder mehr aufeinanderfolgende Testpunkte das gleiche Ergebnis liefern bedeutet dies nicht notwendigerweise, das es sich um einen durchgehenden empfindlichen Zustand handelt. Durch diese Eigenschaft der punktuellen Abtastung des Empfindlichkeitsverlaufs erscheint der Verlauf aus der FI immer zu späteren Zeitpunkten hin verschoben. Entsprechende Details sind in Abbildung 6.17 erkennbar. Eindeutiger Vorteil der BA im Vergleich zur Fehlerinjektion ist die exakte Abgrenzung der

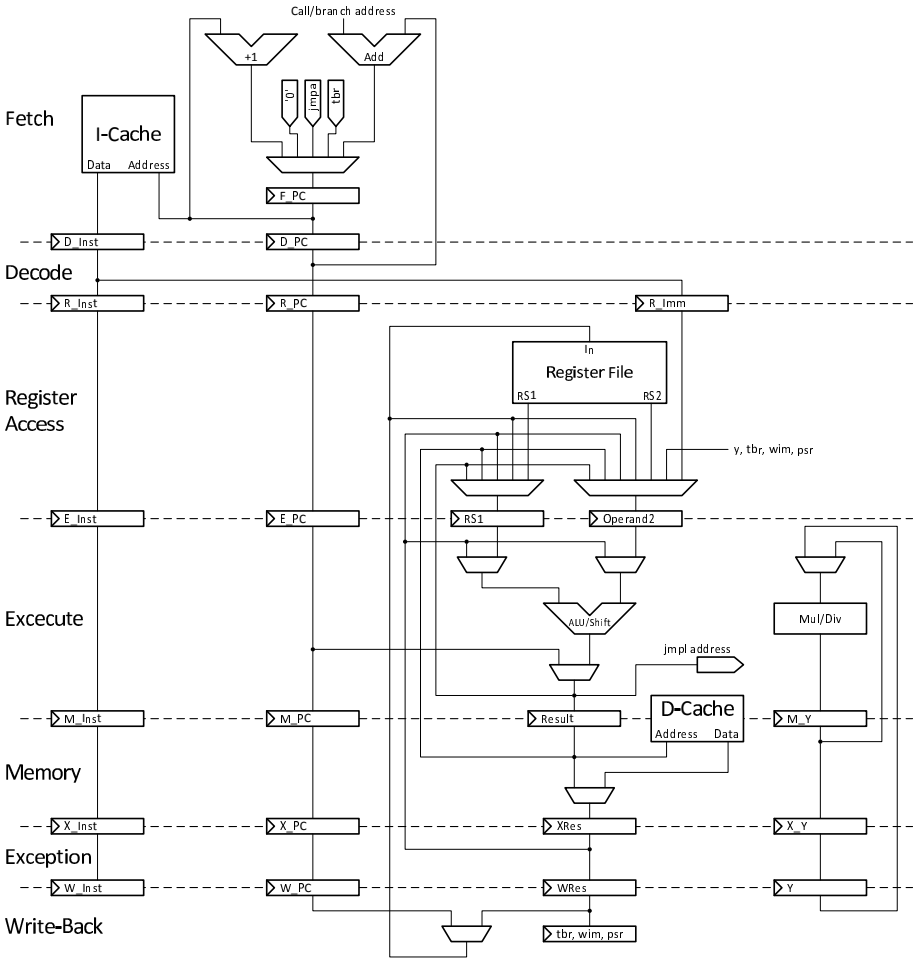



Abbildung 6.18: Struktur der CPU-Pipeline des Leon3-Prozessors [67]

empfindlichen Zeitbereiche, da selbst Blöcke kürzer als eine Taktperiode (siehe Signal sys_clk als Referenz) bzw. kürzer als der Abtastabstand eindeutig erkannt werden.

Der direkte Vergleich zeigt aber auch echte Unterschiede zwischen den Analyse-Ergebnissen. In Abbildung 6.17 sind diese Abweichungen rot eingefärbt dargestellt. Diese Abweichungen wiederholen sich periodisch gegen Ende eines Durchlaufs der While-Schleife im Testprogramm (Abbildung 6.14). Die beiden ersten CPU-Pipeline-Stufen Instruction Fetch und Instruction Decode haben bei ihren Registern für den Program Counter (PC) neben dem Ausgang zum PC-Register der nächsten Pipeline-Stufe noch mindestens einen zusätzlichen Ausgang (z.B. zum Cache und zur Einheit zur Berechnung von Sprungadressen). Nachdem an allen Zeitpunkten mit einer Abweichung im Analyseergebnis das PC-Register der nachfolgenden Pipeline-Stufe unempfindlich ist, müssen die fehlerhaften Read-Events über die beschriebenen zusätzlichen Pfade eintreffen.

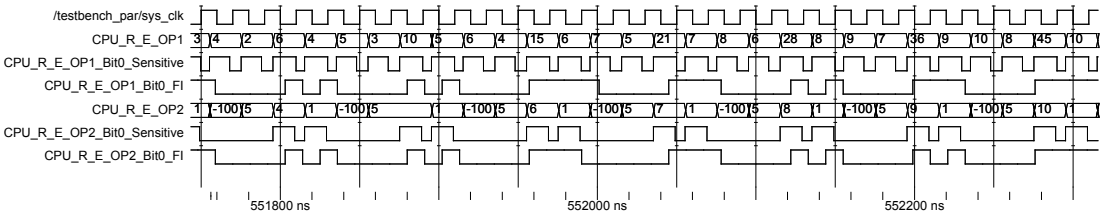



Abbildung 6.19: Detaillierter Vergleich BA/Fehlerinjektion für die ALU-Register OP1 und OP2 im Zeitbereich 551,8$\mu$s bis 552,2$\mu$s

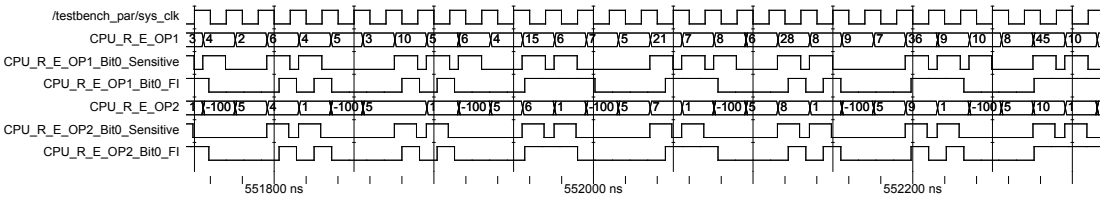



Abbildung 6.20: Ergebnisdarstellung wie in Abbildung 6.19, jedoch mit manuell modifiziertem Zustand eines Ausgangsregisters

Der Vergleich der Ergebnisse zeigt für die BA lediglich eine Überabschätzung der Empfindlichkeit. Als Grund dafür kann entweder die nicht berücksichtigte Selbstmaskierung bei rekonvergenten Pfaden oder andere Maskierungseffekte z.B. in Blockspeichern herangezogen werden. Außerdem wurde in dieser Schaltung festgestellt, dass entgegen der Annahme aus Abschnitt 4.4 hier nicht alle Ausgangsregister immer empfindlich sein müssen. Im Speziellen sind dies hier die Anschlüsse zum Datenbus, der einen externen SRAM-Speicher und ein PROM anbindet. Die Datenleitungen des Bus sind über Tri-State-Treiber mit den externen Komponenten verbunden, können also sowohl Eingang als auch Ausgang sein. Da die CPU natürlich auch direkt Daten in den externen SRAM schreiben kann, sind nur zwei sequentielle Stufen zwischen den ALU-Operand-Registern und den Datenausgängen. Über diesen Pfad treffen Read-Events an einem Operand-Register der ALU ein und führen zu einer Abweichung gegenüber dem Analyse-Ergebnis mit der Fehlerinjektion (Abbildung 6.19). Die hier vorhandene Ausgangslogik in Form von Tristate-Treibern verhindert bei diesem Programm empfindliche Pfade zu den I/O-Ports, daher dürften die Ausgangsregister nicht dauerhaft empfindlich sein.

In einem zweiten Analyse-Durchlauf wurde manuell eines der Ausgangsregister als eben nicht dauerhaft relevant markiert. Damit konnte der Einfluss dieser Fehlerquelle auf das BA-Analyseergebnis weitestgehend vermieden werden. In Abbildung 6.20 kann das verbesserte Analyseergebnis verglichen werden. Da nur ein einzelnes Ausgangsregister durch die manuelle Modifikation betroffen war, ist noch keine vollständige Übereinstimmung mit den Ergebnissen aus der Fehlerinjektion hergestellt. Durch eine Modifikation aller betroffener Ausgangsregister

würde dies aber höchstwahrscheinlich erreicht. Ein besserer Ansatz wäre, durch eine zusätzliche Überwachung der Ausgangssignale die Empfindlichkeit der Ausgangsregister genauer herzuleiten. Unabhängig von möglichen Verbesserungen bestätigen die Ergebnisse der Fehlerinjektion den Empfindlichkeitsverlauf, der durch die BA bestimmt wurde.

## 6.5 Interpretation der BA-Ergebnisse

Aus den Analyseergebnissen der BA für das Leon3-Testsystem können einige Schlussfolgerungen gezogen werden. Für die betrachteten PC-Register ergibt sich, dass zum Ende der Pipeline die Wahrscheinlichkeit eines unkritischen Inhalts steigt. Aus der Pipelinestruktur (Abbildung 6.18) ist ersichtlich, dass die PC-Register nur an wenigen Stellen im späteren Teil der Pipeline verwendet werden bzw. ein kompletter Durchlauf durch alle Stufen nur notwendig ist, wenn der Registerinhalt in das Register-File gelangen soll. Da dies nur bei wenigen Befehlen z.B. zum Laden und anschließenden Modifizieren einer Adresse für den Zugriff in einen externen Speicher notwendig ist, ist die Empfindlichkeit dieser Register dementsprechend niedrig. Die PC-Register (jeweils 30 bit) der Leon3-Pipeline haben für diese Anwendung folgende AVF-Mittelwerte:

- Instruction Fetch Stage: 27%
- Instruction Decode Stage: 8%
- Register Access Stage: 1,5%
- Excecute Stage: 0,5%
- Memory Stage, Exception Stage, Write Back Stage: 0%

Im direkten Vergleich des Empfindlichkeitsverlaufs der einzelnen PC-Register ist auch die Struktur ähnlich einem Schieberegister klar zu erkennen. Ein empfindlicher Zustand in einer Stufe bedingt einen empfindlichen Zustand einen Taktzyklus davor in der vorherigen Stufe.

Dass der Empfindlichkeitsverlauf auch von übergeordneten Steuersignalen abhängig ist, kann am Zusammenhang zum Signal "I_Cache_Hold_n" gesehen werden. Wenn auf Grund eines Cache-Misses die Pipeline angehalten wird, so bleibt auch der Zustand dort in allen Registern unverändert. Dies kann für einzelne Register sowohl eine Verringerung aber auch eine Erhöhung der mittleren Empfindlichkeit bedeuten. Eine Abschätzung der Empfindlichkeit alleine anhand der Ausführungsgeschwindigkeit der CPU und der auszuführenden Anweisungen kann daher zu keinen präzisen Ergebnissen führen.

Ein unerwarteter Verlauf der Empfindlichkeit ist durch die spezielle Implementierung des Register-Files begründet. Wie in den meisten Mikroprozessoren hat das Register-File drei Ports, einen Schreib-Port zum Speichern der berechneten Ergebnisse und zwei Lese-Ports zum Bereitstellen von Operanden für die ALU. Da das Testsystem für ein FPGA synthetisiert wurde und auf dieser Zielplatform nur Dual-Port-RAMs zur Verfügung stehen, wurde das Register File durch

Zusammenschaltung zweier Dual-Port-RAMs realisiert. Bei jedem dieser RAMs wird ein Write-Port identisch angesteuert, damit ist der Speicherinhalt in jedem dieser RAMs zu jedem Zeitpunkt identisch. Der zweite Port jedes Speichers kann unabhängig vom Anderen angesteuert werden, was die Bereitstellung von zwei Operanden gleichzeitig ermöglicht. Da es sich aber in diesem Konstrukt um zwei unabhängige Speicher handelt, muss deren Empfindlichkeitsverlauf nicht identisch sein.

Abbildung 6.21 zeigt einen detaillierten Verlauf der Simulations- und Analyse-Ergebnisse für die im Testprogramm verwendeten Variablen. Die Zählervariable "Cnt" liegt an Adresse 112 im Register File (RF), die Variable "i" an Adresse 129 und Variable "Sum" liegt an Adresse 105. Die Variable "Cnt" wird auf drei unterschiedliche Weisen in dem Programm verwendet, beim Vergleich ob Gerade oder Ungerade, bei der Ausgabe auf die GPIO-Ports und beim Inkrementieren. Im Empfindlichkeitsverlauf für den Speicher A (RF_A...) kann im Zusammenhang mit dem Inkrement der Variable ein kurzer unkritischer Bereich erkannt werden. Dieser Bereich ist durch die Länge der Pipeline definiert. Vom Auslesen eines Wertes aus dem Register File bis zum Schreiben eines Ergebnisses zurück in das Register File werden 4 Taktzyklen benötigt, dies wird durch den Verlauf der Empfindlichkeit für den Speicher RF_A_112 bestätigt. Die Inkrement-Operation verwendet den ALU-Operanden OP1, daher ist der Inhalt im RF_A nahezu immer empfindlich. Dieser Wert wird zu Beginn der äußeren Programmschleife geschrieben und erst am Ende wieder gelesen, um ihn zu inkrementieren. Der Inhalt des RF_B an der gleichen Adresse ist dagegen nur ca. 50% empfindlich. Nur wenn "Cnt" einen ungeraden Wert trägt, wird vom RF_B dieser Wert durch die ALU auf den Datenbus geleitet, wo er letztendlich dann an den GPIO-Pins erscheint.

Ein ähnliches Verhalten wird auch für die Variable "Sum" festgestellt. Diese Variable wird zu Beginn der Schleife direkt in der Pipeline initialisiert. In den weiteren Schleifeniterationen wird der Wert der Variablen "i" aufaddiert. Der Wert der Variablen "Sum" wird dabei kontinuierlich im Register File aktualisiert. Wegen der geringen Anzahl an Instruktionen für einen Schleifendurchlauf ist es möglich, den aktuellen Wert der Variablen "Sum" nicht aus dem Register File zu lesen, sondern durch Operand Forwarding direkt aus der Pipeline abzugreifen. Dies bedeutet, dass die im Register File vorhandene Variable "Sum" einzig und alleine am Ende der Schleife relevant ist, da sie zu diesem Zeitpunkt über den Datenbus an die GPIO-Pins weitergegeben wird. In den Fällen, in denen ein ungerades "Cnt" ausgegeben wird, bleibt "Sum" komplett irrelevant.

Ein nochmals verschärftes Verhalten lässt sich für die Variable "i" beobachten. Diese wird ebenfalls direkt initialisiert und dann ausschließlich innerhalb der Pipeline verwendet, obwohl die entsprechende Adresse im Register File ebenfalls kontinuierlich aktualisiert wird. Zu keinem Zeitpunkt ist der im Register File gespeicherte Wert der Variablen "i" relevant für die Anwendung.

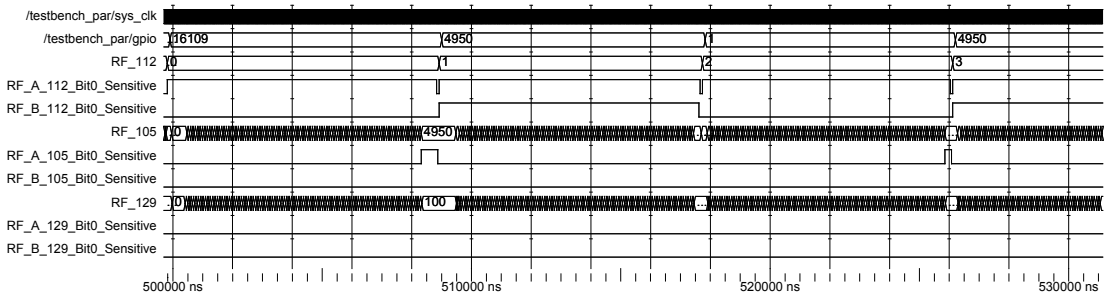



Abbildung 6.21: Analyseergebnisse für das Register File in der Beispielanwendung für den Zeitraum 500 $\mu$s bis 530 $\mu$s

Obwohl die betrachtete Beispielanwendung nur ein sehr kurzes Programm ist und auch nur wenige Ressourcen der gesamten CPU benutzt werden, zeigen sich bereits hier sehr komplexe Zusammenhänge, die nur in seltenen Fällen einfach nachvollzogen werden können. Bei größeren Programmen, mehreren auszuführenden Tasks oder Unterbrechungen durch Interrupt-Aufrufe kann die BA ein gutes Hilfsmittel zum Verständnis der Schaltungen und Programme sein.

## 6.6 Verwendung der Ergebnisse

Die gewonnenen Ergebnisse mittels BA können auf vielfältige Weise verwendet werden. Eine naheliegende Verwendung ist zur Bestimmung der Fehlerrate einer Schaltung. Der mittlere Empfindlichkeitswert eines einzelnen Speichers bzw. der gesamten Schaltung gibt eine Aussage über die Zuverlässigkeit. Wenn bereits Daten für die SEU-Fehlerrate für Schaltungen in einer bestimmten Technologie vorliegen, kann zusammen mit den Ergebnissen der BA daraus die absolute SEU-Fehlerrate abgeleitet werden. Im Bezug auf die IEC61508 findet eine Unterscheidung zwischen gefährlichen und ungefährlichen Fehlern statt. Da der bestimmte Empfindlichkeitswert gleichbedeutend ist mit der Menge an Fehlern, die eine beobachtbare Abweichung im Schaltungsverhalten verursachen, kann dieser Wert als Wahrscheinlichkeit für einen gefährlichen Fehler ($\lambda_D$) angenommen werden. Der Anteil der ungefährlichen Fehler ist folglich der komplementäre Anteil ($\lambda_S = 1 - \lambda_D$). In einem weiteren Schritt kann auch die SFF (Safe Failure Fraction, siehe Abschnitt 2.4.2) bestimmt werden.

Wie im Kapitel 6.2 gezeigt, kann für jede Schaltung ein Empfindlichkeitsprofil erstellt werden. Jeweils die empfindlichsten Speicher aus der Gesamtmenge sind eindeutig identifizierbar. Diese Speicher könnten dann bei einem Redesign der Schaltung speziellen Härtungsmaßnamen unterzogen werden, um eine größere Robustheit der Schaltung zu erreichen. Dies kann, wenn ein Wechsel der Fertigungstechnologie nicht möglich ist, z.B. durch die Verwendung von Speicherzellen mit größerer Fläche erfolgen. Ebenfalls kann ein Ansatz mit partieller TMR ver-

folgt werden. Egal welches Verfahren angewandt wird, allen gemein ist, dass durch die Ergebnisse der BA mit geringstem Aufwand die größtmögliche Verbesserung der Robustheit erreicht werden kann.

Bezogen auf das Beispiel von Abbildung 6.15, könnte durch Härtung von 4% der gesamten Speichermenge der AVF der gesamten Schaltung deutlich reduziert werden. Unter der Annahme, dass eine komplett mit TMR gehärtete Schaltung maximal ca. den siebenfachen Aufwand erfordert [46], ergibt sich für die Schaltung mit partiellem TMR ein Aufwand vom 1,24-fachen der ursprünglichen Schaltung ($96\% + 7 \cdot 4\% = 124\%$). Der Aufwand für die mit partiellem TMR gehärteten Schaltung ist daher nur geringfügig höher als in der ursprünglichen Schaltung. Zusätzlich ist durch die wenigen modifizierten Schaltungsteile die Wahrscheinlichkeit geringer, dass ein Schaltungsteil mit kritischem Timing betroffen ist.

# Kapitel 7

# Abschluss

## 7.1 Zusammenfassung

In dieser Arbeit wurden die Auswirkungen von SEEs auf digitale Schaltungen untersucht. Ausgehend von den physikalischen Vorgängen wurden die resultierenden Effekte auf einzelne Transistoren, Logikgatter, DRAM- und SRAM-Speicherzellen vorgestellt. Auf Grund der stetig sinkenden Strukturgrößen in integrierten Schaltungen können schon einzelne ionisierte Partikel in dotierten Halbleiterbereichen ausreichend viele freie Ladungsträger erzeugen, die den Ausgangspegel eines Logikgatters temporär oder den gespeicherten Wert in einer Speicherzelle permanent verändern können. Die ionisierten Partikel entstehen dabei beim Zerfall radioaktiver Isotope in unmittelbarer Nähe des Halbleiters oder werden durch kosmische Strahlung verursacht. Durch Verbesserungen an den verwendeten Materialien und dem Aufbau der Transistoren konnte die Fehlerrate annähernd konstant gehalten werden. Da jedoch die Anzahl der integrierten Transistoren stetig weiterwächst, steigt auch die Fehlerrate einer integrierten Schaltung insgesamt.

Da aber auch die Logikfunktion und der Anwendungsfall der Schaltung Einfluss auf die Fehlerrate haben, wurde genauer untersucht, welche Effekte tatsächlich aufgetretene Fehler unwirksam werden lassen. Die vorgestellten Fehlermaskierungseffekte konnten systematisch klar voneinander getrennt werden. Im weiteren Verlauf wurden verschiedene Verfahren zur Bestimmung der Fehlermaskierung genauer untersucht. Jedes der betrachteten Verfahren hat Vorteile für einzelne Bewertungskriterien wie z.B. Genauigkeit der Ergebnisse, Rechenaufwand oder Aufwand zur Modellierung, jedoch konnte keines der Verfahren in allen Bewertungskriterien umfassend überzeugen. Die identifizierten konzeptionellen Schwachpunkte und Stärken wurden für weitere Überlegungen zusammengefasst.

Im nächsten Abschnitt wird auf die besonderen Fehlermechanismen bei FPGAs eingegangen, da diese besonders viel Speicher in unterschiedlichen Funktionen enthalten. In einem ersten Teil wurden der typische Aufbau von FPGAs und die möglichen Fehlermechanismen beschrieben. Im Weiteren wurden die Anteile für die unterschiedlichen Funktionsbereiche am gesamten enthaltenen Speicher exemplarisch für eine FPGA-Familie bestimmt. In Zusammenhang mit den gezeigten Fehlererkennungsmaßnahmen kann das Gefährdungspotential durch SEUs

für FPGAs abgeschätzt werden. Insbesondere für den Anteil an Speicher, der in einer synthetisierten Schaltung realisiert ist, sind keine integrierten Schutzmechanismen pauschal anwendbar. Eine wirksame, aber kostenintensive Methode zum Schutz vor den Auswirkungen von SEUs in FPGAs ist die Anwendung von TMR.

Für die Bestimmung der Fehlermaskierungsfaktoren wurde ein eigenes Konzept, die sogenannte Rückwärts-Analyse (Backwards Analysis BA), vorgeschlagen. Anhand der bekannten Schwachstellen anderer Verfahren wurde der Schwerpunkt zuerst auf die Behandlung des transitiven Maskierungseffekts gelegt. Durch die Betrachtung eines Simulationsablaufs in zeitlich umgekehrter Reihenfolge konnte ein einfaches Schema zur Modellierung der logischen und transitiven Fehlermaskierung implementiert werden. In einem weiteren Schritt wurde mittels einer simulationsgetriebenen Erkennung der Schreib- und Lesezugriffe auf die Speicher der betrachteten Schaltung ein weiterer Maskierungseffekt integriert. Das Zeitverhalten der Schaltung wurde durch Modifikation der Zeitpunkte für Schreib- und Lesezugriffe ebenfalls integriert, so dass eine umfassende Betrachtung aller benannten Maskierungseffekte möglich ist. Auf mögliche Sonderfälle im Schaltungsdesign wurde eingegangen und die Konsequenzen für die Analyse betrachtet. Als letzter Punkt in der Konzeptvorstellung wurde die theoretische Laufzeit des Verfahrens betrachtet. Durch den besonderen Ansatz mit dem zeitlich umgekehrt betrachteten Anwendungsfall konnte für die Ausführungszeit ein lineares Verhalten gegenüber der Problemgröße festgestellt werden. Mit Hilfe dieses Verfahrens kann in einem Durchlauf für alle Speicher einer Schaltung ein exakter zeitlicher Empfindlichkeitsverlauf bestimmt werden.

Darüber hinaus wurden weitere Verbesserungen gezeigt, mit denen der Anwendungsbereich dieses Verfahrens vergrößert werden kann. Als Nebeneffekt wird zudem die Laufzeit und der Speicherbedarf weiter verringert. Wesentlicher Ansatz war hier das Weglassen des Schaltungstimings während der Simulation in Verbindung mit der rechnerischen Berücksichtigung des Schaltungstimings im Analyseschritt. Als Folge dieser Verbesserung wurde eine parallelisierbare Implementierung des Verfahrens ermöglicht. Bedingt durch die vorteilhafte Struktur des Verfahrens können im Analyseschritt viele der Teilaufgaben in unabhängiger Reihenfolge und gleichzeitig ausgeführt werden. Auf Grund mangelnder Unterstützung durch den verwendeten Simulator war diese Verbesserung jedoch nahezu wirkungslos. Für Sonderfälle in der Schaltungsstruktur wurde eine Verringerung des Rechenzeit- und Speicherbedarfs durch eine Optimierung der Schaltungsbeschreibung erreicht.

Im verbleibenden Abschnitt wurde das vorgestellte Verfahren und die benannten Verbesserungen durch praktische Versuche hinterlegt. Konstruierte Testschaltungen wurden verwendet, um die korrekte Analyse der grundlegenden Maskierungseffekte zu überprüfen. Diese Ergebnisse wurden für diese sehr kleinen Schaltungen anhand theoretischer Überlegungen verifiziert. Ein Vergleich der verschiedenen Optimierungsstufen der BA wurde mit den ISCAS89-Benchmark-Schaltungen durchgeführt. Die gemessenen Ergebnisse für die Laufzeit der BA

lagen dabei im erwarteten Bereich, speziell der Laufzeitvergleich zwischen reiner Schaltungssimulation und kompletter BA bestätigte einen linearen Zusammenhang. Die ermittelten Empfindlichkeitsverläufe für größere Schaltungen wurden mittels Fehlerinjektion überprüft. Durch einen sehr kleinen Testabstand konnte für einzelne Signale mit der Fehlerinjektion ebenfalls ein Empfindlichkeitsverlauf bestimmt werden. Der Vergleich zeigte hier weitestgehend Übereinstimmung, für den untersuchten Leon3-Prozessor waren kleine Abweichungen festzustellen. Weitere Untersuchungen zeigten, dass diese Abweichungen durch die nicht betrachtete Ausgangslogik verursacht wurden. Überlegungen anhand der Pipeline-Struktur des Leon3-Prozessors und des darauf ausgeführten Programms konnten ebenfalls den mit der BA bestimmten Empfindlichkeitsverlauf bestätigen. Die gewonnenen Ergebnisse belegen, dass in einer Schaltung häufig ein kleiner Teil der enthaltenen Speicher für einen großen Anteil der gesamten SEU-Empfindlichkeit verantwortlich ist. Bei einem direkten Laufzeitvergleich wurde ein Geschwindigkeitsvorteil für die BA um mehr als fünf Größenordnungen gegenüber der Fehlerinjektion festgestellt.

## 7.2 Weiterführende Ideen basierend auf der Rückwärts-Analyse

Eine der wesentlichen Eingenschaften der Rückwärts-Analyse ist die vollständige Abdeckung des Anwendungsfalls und der betrachteten Schaltung. Für jeden Schaltungszustand wird bestimmt, ob dieser innerhalb des betrachteten Anwendungsfalls beobachtbar ist. Im Bezug auf einen Schaltungstest bzw. einen speziellen Testmustersatz könnte die Rückwärts-Analyse zur Bestimmung der Testabdeckung verwendet werden. Jeder Speicher in der Schaltung, für den die Rückwärts-Analyse mindestens einen Block empfindlicher Zeit findet, ist zumindest teilweise testbar durch den angewandten Testmustersatz. Die Art des getesteten Fehlers (s-a-0 oder s-a-1) wird durch den gespeicherten Wert festgelegt, wobei der getestete Fehler dem invertierten gespeicherten Wert entspricht. Für einen vollständigen Test müssen daher mindestens zwei empfindliche Zeitblöcke erkannt werden, jeweils ein Block für beide Logikpegel. Doch auch für die Logikgatter lässt sich eine Testabdeckung ableiten. Dadurch, dass die Blöcke mit empfindlicher Zeit die Beobachtbarkeit einzelner Speicher beschreiben, vereinfacht sich das Testproblem wesentlich. Beim Nachverfolgen der empfindlichen Pfade vom Senken-Speicher zum Quellen-Speicher werden alle beobachtbaren Logikgatter durchwandert. Für die Auswertung der Testabdeckung gilt das gleiche wie bei den Speichern, jeder beobachtbare Logikpegel in diesen empfindlichen Pfaden stellt einen Test für den invertierten Signalwert dar. Für eine Auswertung der Testabdeckung muss lediglich eine Liste geführt werden, welche Logikpegel an welchen Gattern anliegen, während ein empfindlicher Pfad erkannt wurde. Das Fault Grading, d.h.

die Bestimmung der Testabdeckung eines Testmustersatzes, kann genauso wie die Rückwärts-Analyse selbst mit linearer Laufzeit durchgeführt werden. Ein direkter Schluss von einem getesteten Fehler auf ein notwendiges Testmuster ist aber nicht möglich.

Einer der Kern-Mechanismen der Rückwärts-Analyse, die Überwachung von möglichen Trigger-Signalen, ließe sich auch auf andere Abstraktionsebenen einer Simulation übertragen. Hierfür prinzipiell in Frage kämen Simulationen auf Register-Transfer-Level oder Transaction Level Modeling Simulationen z.B. mit SystemC. Generell könnte mit dem Ansatz der Rückwärts-Analyse jede Art von Simulation untersucht werden, für die ein Schaltungsgraph und eine Aufzeichnung der Speicherzugriffe erstellt werden kann. Effekte wie Logikmaskierung und Signallaufzeit können sicher nicht in der gleichen Art und Weise wie für die vorgestellte Implementierung der Rückwärts-Analyse berücksichtigt werden, ähnliche Betrachtungen sind aber auf jeden Fall denkbar. Wesentliches Hindernis stellt für diese High-Level-Systembeschreibungen prinzipbedingt der niedrigere Detaillierungsgrad der Implementierung dar. Effekte wie die Verweildauer von Informationen in einzelnen Speichern oder die Auslastung von Block-Speichern oder FIFOs könnten dennoch einfach berücksichtigt werden. Damit könnte die Rückwärts-Analyse auch schon in früheren Stufen eines Design-Zyklus angewandt werden, wo sich einfacher und schneller Verbesserungen des Gesamt-Systems erreichen lassen als auf Gatter-Ebene.

## 7.3 Schlusswort

Ausgangssituation für diese Arbeit war die Tatsache, dass moderne hochintegrierte digitale Schaltungen nicht mehr zu 100 Prozent fehlerfrei arbeiten und stattdessen sporadisch falsche Rechenergebnisse liefern oder unbegründete Systemausfälle auftreten. Ursache hierfür sind unvermeidbare Strahlungseffekte, die in digitalen Speichern zu SEUs führen können. Für eine Anwendung dieser Schaltungen in sicherheitskritischen Systemen, wie z.B. im Automobil als Fahrerassistenzsystem, in der Automatisierungstechnik im Zusammenspiel mit Menschen oder auch in der Medizintechnik oder Luft- und Raumfahrt, muss eine ausreichend geringe Fehlerrate auf stichhaltige Weise nachgewiesen werden. Die als Stand der Technik bekannten Verfahren zur Abschätzung der Fehlermaskierung haben ihre Stärken für spezielle Anwendungsbereiche. Ein möglichst universelles und hinsichtlich der Qualität der erzeugten Ergebnisse zuverlässiges Verfahren wurde nicht erkannt.

Aus diesem Grund wurde mit einem systematisch konstruierten Verfahren unter einem grundsätzlich anderen Ansatz versucht die Nachteile der bekannten Verfahren zu vermeiden, um eine möglichst detaillierte Analyse der Fehlermaskierung zu erreichen. Das Verfahren beruht auf dem Ansatz, dass der Folgezustand in einem digitalen Automaten eindeutig aus dem gegenwärtigen Zustand und den

Eingangssignalen bestimmbar ist. Dieser Ansatz wurde auf die Fehlerbeobachtbarkeit übertragen mit dem Unterschied, dass hier die Schaltungszustände in zeitlich umgekehrter Reihenfolge durchlaufen werden müssen. In Folge dessen kann die Beobachtbarkeit zu einem bestimmten Zeitpunkt eindeutig aus der Beobachtbarkeit zu einem späteren Zeitpunkt und den Eingangssignalen bestimmt werden. Dieses Vorgehen ermöglichte es einen vollständigen zeitlichen Verlauf der Fehlermaskierung mit relativ geringem Rechenaufwand zu bestimmen. Das Verfahren eignet sich sowohl für ASICs als auch für in FPGAs synthetisierte Schaltungen, da hier die Designzyklen kürzer sind und wesentlich häufiger auch Änderungen durchgeführt werden. Bei einem langwierigen und schwer anzuwendenden Verfahren würde häufig wohl auf eine genaue Analyse der Fehlermaskierung verzichtet werden.

In der Anwendung auf praktische Schaltungen wurden bei der Durchführung des Verfahrens dennoch einige Probleme ausgemacht, die systematisch zu falschen Ergebnissen führen würden. Durch weitere Optimierungen konnten diese Sonderfälle erfolgreich durch das Analyseschema abgedeckt werden. Als Nebeneffekt konnte die Rechenzeit und der Speicherbedarf weiter verringert werden. Sogar eine Parallelisierung des Verfahrens für Mehrkern-Prozessoren wurde ermöglicht, wenngleich dies mit dem zugrunde liegenden Simulator zu keiner Verbesserung führte.

Bei der Verifikation des Verfahrens und der damit erzeugten Ergebnisse wurde die erwartete Funktion und Leistungsfähigkeit bestätigt. Eine einfache quantitative Interpretation der Ergebnisse in Form einer Verteilungsfunktion der Fehlerwahrscheinlichkeit ist möglich. Punktgenaue Erhöhung der Robustheit der Schaltungen wird dadurch ermöglicht. Darüber hinaus gehende detaillierte Betrachtungen sind auch durchführbar, jedoch erfordert dies im Allgemeinen eine genaue Kenntnis der Schaltung, um aus den dargestellten Abläufen sinnvolle Rückschlüsse ziehen zu können.

Zu jetzigen Zeitpunkt ist es für viele Halbleiterhersteller ausreichend, Fehlerraten als Mittelwerte für ihre Schaltungen anzugeben. Sobald aber, z.B. durch Änderung der Gesetzgebung oder Normen, präzise ermittelte Fehlerraten auch für komplexe Systeme bekannt sein müssen, wächst der Bedarf nach einem Verfahren wie der Rückwärts-Analyse. Unabhängig davon besteht der Wunsch, dass transiente Fehler in elektronischen Schaltungen, die Leib und Leben gefährden, so weit wie möglich reduziert werden. Einen Beitrag dazu kann die Rückwärts-Analyse leisten.

# Literaturverzeichnis


[1] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Trans. Electron Devices*, vol. 26, no. 1, Jan. 1979.

[2] C. S. Guenzer, E. A. Wolicki, and R. G. Allas, "Single event upsets of dynamic ram's by neutrons and protons," *IEEE Trans. Nucl. Sci.*, vol. 26, no. 6, Dec. 1979.

[3] J. F. Ziegler, "The effect of concrete shielding on cosmic ray induced soft fails in electronic systems," *IEEE Trans. Electron Devices*, vol. 28, no. 5, May 1981.

[4] A. Taber and E. Normand, "Single event upset in avionics," *IEEE Trans. Nucl. Sci.*, vol. 40, no. 2, Apr. 1993.

[5] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson, "On latching probability of particle induced transients in combinational networks," in *24th International Symposium on Fault-Tolerant Computing FTCS-24*, 1994.

[6] National Aeronatics and Space Administration, "Single Events Effects Specification," http://radhome.gsfc.nasa.gov/radhome/papers/seespec.htm, 2009.

[7] H. Kobayashi, N. Kawamoto, J. Kase, and K. Shiraish, "Alpha particle and neutron-induced soft error rates and scaling trends in sram," in *IEEE Int. Reliability Physics Symposium (IRPS 2009)*, 2009.

[8] J. F. Ziegler, M. E. Nelson, J. D. Shell, R. J. Peterson, C. J. Gelderloos, H. P. Muhlfeld, and C. J. Montrose, "Cosmic ray soft error rates of 16-mb dram memory chips," *IEEE J. Solid-State Circuits*, vol. 33, no. 2, Feb. 1998.

[9] J. Fabula, J. Moore, and A. Ware, "Understanding Neutron single-event phenomena in FPGAs," http://mil-embedded.com/PDFs/Xilinx.Mar07.pdf, 2007.

[10] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, Apr. 1965.

[11] Wikipedia, "Redundanz (Technik)," http://de.wikipedia.org/wiki/Redundanz_(Technik), 2011.

[12] C. E. Shannon, "A mathematical theroy of communication," *Bell System Technical Journal*, 1948.

[13] N. Seifert and N. Tam, "Timing vulnerability factors of sequentials," *IEEE Trans. Device Mater. Rel.*, vol. 4, no. 3, Sep. 2004.

[14] H. T. Nguyen and Y. Yagil, "A systematic approach to SER estimation and solutions," in *IEEE 41st Annual International Reliability Physics Symposium*, 2003.

[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 *Proc. 36th IEEE Int. Symposium on Microarchitecture*, 2003.

[16] H. T. Nguyen, Y. Yagil, N. Seifert, and M. Reitsma, "Chip-level soft error estimation method," *IEEE Trans. Device Mater. Rel.*, vol. 5, no. 3, Sep. 2005.

[17] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," in *IEEE Int. Conference on Dependable Systems and Networks (DSN04)*, 2004.

[18] M. Sauer, V. Tomashevich, J. Müller, M. Lewis, A. Spilla, I. Polian, B. Becker, and W. Burgard, "An fpga-based framework for run-time injection and analysis of soft errors in microprocessors," in *IEEE 17th International On-Line Testing Symposium*, 2011.

[19] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Softarch: An architecture-level tool for modeling and analyzing soft errors," in *IEEE Int. Conference on Dependable Systems and Networks (DSN05)*, 2005.

[20] L. Barnes, "Performance modeling and analysis at amd: A guided tour," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2007.

[21] M. Moudgill, J.-D. Wellman, and J. H. Moreno, "Environment for powerpc microarchitecture exploration," *IEEE Micro*, vol. 19, no. 3, pp. 15–25, 1999.

[22] G. Fey, A. Suelflow, and R. Drechsler, "Computing bounds for fault tolerance using formal techniques," in *46th ACM/IEEE Design Automation Conference (DAC09)*, 2009.

[23] F. Corno, M. S. Reorda, and G. Squilero, "Rt-level itc'99 benchmarks and first atpg results," *IEEE Des. Test. Comput.*, vol. 17, no. 3, Jul. 2000.

[24] G. Asadi and M. B. Tahoori, "An analytical approach for soft error rate estimation in digital circuits," in *IEEE Int. Symposium on Circuits and Systems (ISCAS05)*, 2005.

[25] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM Journal of Research and Development*, vol. 10, no. 4, pp. 278–291, 1966.

[26] Y. Herve and F. Pecheux, "ISCAS85 / ISCAS89 Translation," http://jerry.c-lab.de/~wolfgang/VHDL/models/ISCAS, 1994.

[27] H. Asadi and M. B. Tahoori, "Soft error modeling and protection for sequential elements," in *Proc. 20th IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems (DFT05)*, 2005.

[28] *Intel Itanium Architecture Software Developers Manual Volume 3: Intel Itanium Instruction Set Reference Revision 2.3.* Intel Corporation, 2010.

[29] E. D. Lazowska, J. Zahorjan, S. G. Graham, and K. C. Sevcik, "Quantitative system performance," *Prentice Hall*, 1984.

[30] Standard Performance Evaluation Corporation, "SPEC CPU2000 V1.0," http://www.spec.org/cpu2000, 1999.

[31] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *Proc. 32nd International Symposium on Computer Architecture (ISCA05)*, 2005.

[32] Wikipedia, "Failure In Time," http://de.wikipedia.org/wiki/Failure_In_Time, 2012.

[33] ITEM Software, "IEC62380 Reliability Prediction," http://www.reliabilityeducation.com/intro_iec62380.html, 2012.

[34] *ISO 26262: Road Vehicles: Functional Safety.* ISO, 2011.

[35] *ISO 26262-2: Road Vehicles: Functional Safety, Part 2: Management of functional safety.* ISO, 2011.

[36] *ISO 26262-3: Road Vehicles: Functional Safety, Part 3: Concept phase.* ISO, 2011.

[37] *ISO 26262-10: Road Vehicles: Functional Safety, Part 10: Guideline on ISO 26262.* ISO, 2011.

[38] *ISO 26262-5: Road Vehicles: Functional Safety, Part 5: Product development at the hardware level.* ISO, 2011.

[39] Wikipedia, "IEC 61508," http://de.wikipedia.org/wiki/IEC_61508, 2012.

[40] Xilinx Inc., "UG474: 7 Series FPGAs Configurable Logic Block," http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf, 2012.

[41] ——, "UG470: 7 Series FPGAs Configuration," http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf, 2012.

[42] *IEEE1149.1.* IEEE, 1990.

[43] *M68HC11 Reference Manual.* Freescale Semiconductor, Inc., 2002.

[44] Xilinx Inc., "DS112: Virtex4 Familiy Overview," http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf, 2012.

[45] ——, "UG071: Virtex4 FPGA Configuration User Guide," http://www.xilinx.com/support/documentation/user_guides/ug071.pdf, 2012.

[46] K. S. Morgan, D. L. McMurtrey, B. H. Pratt, and M. J. Wirthlin, "A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs," *IEEE Trans. Nucl. Sci.*

[47] Xilinx Inc., "XAPP216: Correcting Single-Event Upsets Through Virtex Partial Configuration," http://www.xilinx.com/support/documentation/application_notes/xapp216.pdf, 2012.

[48] Altera Corporation, "SEU Mitigation in Stratix V Devices," http://www.altera.com/literature/hb/stratix-v/stx5_51011.pdf, 2012.

[49] S. D'Angelo, C. Metra, S. Pastore, A. Pogutz, and G. Sechi, "Fault-tolerant voting mechanism and recovery scheme for tmr fpga-based systems," in *IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems*, 1998.

[50] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving fpga design robustness with partial tmr," in *44th Annual International Reliability Physics Symposium*, 2006.

[51] R. Hartl, A. Rohatschek, W. Stechele, and A. Herkersdorf, "Architectural vulnerability factor estimation with backwards analysis," in *2010 13th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD)*, 2010.

[52] Robert Hartl, Robert Bosch GmbH, "Verbesserungen der Rückwärts-Analyse zur Bestimmung von Fehlermaskierungsfaktoren," Germany Patent DE201 010 040 035, Aug. 31, 2010.

[53] IEEE, *IEEE1076 Very High Speed Integrated Circuit Hardware Description Language*, 2008.

[54] ——, *IEEE1364 Verilog Hardware Description Language*, 2005.

[55] P. Goel and B. C. Rosales, "PODEM-X: An Automated Test Generation System for VLSI Logic Structures," in *18th Design Automation Conference*, 1981.

[56] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithm," *IEEE Trans. Comput.*, no. 12, pp. 1137–1144, 1983.

[57] *IEEE1164*. IEEE, 1993.

[58] IEEE, *IEEE1364-1995 Value Change Dump file format*, 1995.

[59] J. A. Clark and D. K. Pradhan, "Fault injection: A method for validating computer-system dependability," *IEEE Computer*, vol. 28, no. 6, Jun. 1995.

[60] R. Hartl, A. Rohatschek, W. Stechele, and A. Herkersdorf, "Improved backwards analysis for architectural vulnerability factor estimation," in *Semiconductor Conference Dresden (SCD)*, 2011.

[61] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE Int. Symposium on Circuits and Systems (ISCAS89)*, 1989.

[62] F. Corno, G. Cumani, M. S. Reorda, and G. Squillero, "Effective Techniques for High-Level ATPG," in *18th Asian test Symposium*, 2001.

[63] L. Zhang, I. Ghosh, and M. Hsiao, "Efficient Sequential ATPG for Functional RTL Circuits," in *International Test Conference 2003 (ITC2003)*, 2003.

[64] Jiri Gaisler, Aeroflex Gaisler AB, "GRLIB VHDL IP LIBRARY," http://www.gaisler.com, 2010.

[65] Sebastian Humin, "Schaltungssimulation und Vergleich von Fehlerraten-Abschätzungsverfahren," Lehrstuhl für Integrierte Systeme, Technische Universität München, 2011.

[66] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th Intl. Symposium on High-Performance Computer Architecture (HPCA-11)*, 2005.

[67] Jiri Gaisler, Aeroflex Gaisler AB, "GRLIB IP Core User's Manual," http://www.gaisler.com, 2013.