

Effekte von Stuck-At Faults in Delay-Insensitiver Logik

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Christian Trenkwald

Matrikelnummer e0525307

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao. Univ. Prof. Dipl.-Ing. Dr.techn. Andreas Steininger
Mitwirkung: Univ.-ass. Dipl.-Ing. Robert Navjvirt

Wien, 30. September 2014 _____
(Unterschrift Verfasser)

_____ (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich Trenkwaldner Christian geboren am 20.04.1985 in IT-39049 Sterzing, wohnhaft in AT-1020 Wien Ilgplatz 7/16, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, den 30. September 2014

.....
Unterschrift Verfasser

Abstract

Asynchronous-quasi-delay-insensitive (QDI) logic is more tolerant to transient- and permanent faults than synchronous logic. This results from a delay independent encoding. The occurrence of permanent faults causes a deadlock in the protocol of the considered class of delay-insensitive circuits and therefore they will stop operation. This raises the question whether, after correction of the defect, the circuit can continue to work without errors, or their internal state needs to be reset. After the occurrence of a stuck-at fault deadlock, late detection and premature fire are identified as possible behaviours.

To analyse the behaviour of complex circuits with reasonable effort, we must take a compromise between complexity and level of detail. In this work, an abstract model is developed, which without loss of generality, is applicable to all circuits. A circuit is divided into several blocks at the logical level, which will be analysed independently of each other. The results of the individual analyses are used on the next higher level of abstraction to determine the behaviour of the complete circuit.

To verify the created model it is then compared to a behavioural simulation. We are going to prove the correctness and applicability of the model on a common circuit template, which is then applied on null convention logic, level-encoded-dual-rail and code-alternating logic. At the end we obtain a prediction for the probability of the circuit creating an undesirable output or a faulty internal state.

Kurzfassung

Asynchrone Quasi Delay-Insensitive (QDI) Logik ist im Vergleich zu synchroner Logik toleranter gegenüber transienten Fehlern und permanenten Defekten. Dies ist das Resultat einer delay-unabhängigen Kodierung. Das Auftreten eines permanenten Defektes führt bei der betrachteten Klasse von delay-insensitiven Schaltungen immer zu einem Deadlock im Protokoll und damit zum Anhalten der Operation. Es stellt sich die Frage, ob nach Korrektur des Defekts die Schaltung unmittelbar fehlerfrei in ihrer Operation fortfahren kann, oder ob ihr interner Status als Folge des Defekts fehlerhaft sein kann, zudem stellt sich die Frage, von welchen Faktoren dies abhängt. Diese Fragen sollen in der vorliegenden Arbeit beantwortet werden.

Als mögliches Verhalten der Schaltung nach dem Auftreten eines Defektes werden Deadlock, Late Detection und Premature Fire identifiziert. Um das Verhalten von komplexen Schaltungen mit vertretbarem Aufwand zu analysieren, muss ein Kompromiss aus Komplexität und Detailreichtum geschlossen werden. In dieser Arbeit wird ein abstraktes Modell entwickelt, welches ohne Verlust der Generalität auf alle Schaltungen anwendbar ist. Eine Schaltung wird auf ihrem logischen Level in mehrere Blöcke unterteilt, welche unabhängig voneinander analysiert werden. Die Ergebnisse der Einzelanalysen werden, auf der nächsthöheren Abstraktionsebene verwendet, um auf das Verhalten der Gesamtschaltung zu schließen.

Zur Verifikation des erstellten Modells wird dessen Ergebnis mit dem einer Behavioral-Simulation verglichen. Es wird die Anwendbarkeit und Korrektheit des Modells für ein verbreitetes Template für asynchrone Schaltungen bewiesen. Das Template wird auf Null-Convention-Logic, Level-Encoded-Dual-Rail und Code-Alternating-Logic angewandt. Das Resultat ist eine Vorhersage mit welcher Wahrscheinlichkeit eine Schaltung nach einem Defektes ein unerwünschtes Ergebnis liefert bzw. einen fehlerhaften internen Zustand aufweist.

Inhaltsverzeichnis

Abstract	ii
Kurzfassung	iii
Inhaltsverzeichnis	v
1 Einführung	1
1.1 Problemstellung	1
1.2 Ziel der Arbeit	2
1.3 Aufbau	2
2 Grundlagen	3
2.1 Übersicht	3
2.2 Logik Elemente	3
2.2.1 Basis Elemente	3
2.2.2 Programmable Logic Array	3
2.2.3 C Element	4
2.2.4 SR-Latch	5
2.2.5 Treshold Gates	6
2.3 Petri-Netze	7
2.4 Pipeline	8
2.4.1 Timing	8
2.4.1.1 Synchrone Pipeline	8
2.4.1.2 Asynchrone Pipeline	9
2.4.2 Eigenschaften einer Pipeline	11
2.4.3 Logische Organisation	11
2.4.3.1 Lineare Pipeline	11
2.4.3.2 Nichtlineare Pipeline	12
2.5 Delay Modell	14
2.6 Handshake Protokolle	14
2.6.1 Vier-Phasen-Handshake	14
2.6.2 Zwei-Phasen-Handshake	15
2.7 Dual-Rail Codierung	15
	v

2.7.1	Three State Logic	16
2.7.1.1	TSL Completion Detector (CD)	17
2.7.2	Four State Logic (FSL)	17
2.7.2.1	LEDR CD	18
2.7.2.2	CAL CD	19
2.8	Dual Rail Logik	19
2.8.1	Self Timed Logic Module (STLM)	19
2.8.2	Delay Insensitive Minterm Synthesis (DIMS)	20
2.8.3	Threshold Gate (TG)	20
2.8.4	Programmable Logic Array (PLA)	21
2.9	Logical Effort	21
2.10	Faults	22
2.10.1	Single Stuck-At Fault (SSAF)	23
3	Qualitative Analyse und Erstellung eines Modells	25
3.1	Übersicht	25
3.1.1	Einschränkung	25
3.1.2	Klassifizierung der Effekte	28
3.2	Analyse der Blöcke	31
3.2.1	Verfahren	31
3.2.1.1	Time Petri-Net basierendes Stuck-At Fault Modell	32
3.2.1.2	Verfahren für die qualitative Analyse	33
3.2.1.3	Modell für die quantitative Analyse	35
3.2.2	Completion Detektor	38
3.2.3	Kontrolllogik	42
3.2.4	Register	44
3.2.5	Funktionslogik	46
3.3	Allgemeines Pipeline Template	47
3.3.1	Verfahren für die quantitative Analyse	47
3.3.2	Anwendung auf das Pipeline Template	49
3.4	Zusammenfassung	51
4	Quantitative Analyse und Verifikation durch Simulation	53
4.1	Verfahren und Testaufbau	53
4.2	Schaltungsaufbau	54
4.2.1	Codeerweiterung für Fault-Injection	54
4.2.2	Timing laut Logical-Effort	58
4.3	Stimulus Generator	59
4.4	Auswertung	60
4.4.1	TSL	61
4.4.2	LEDR	61
4.4.3	CAL	63
4.5	Zusammenfassung und Vergleich zu Modell	66

<i>Inhaltsverzeichnis</i>	vii
5 Conclusion	69
A Beweise	71
B Quelltext	73
C Tabellen	75
C.1 Qualitative Analyse	75
D Dokumentation der Simulation	81
D.1 Dateien	81
D.2 Variablen	82
D.3 Funktionen	84
Literaturverzeichnis	87
Glossary	97

Einführung

1.1 Problemstellung

Asynchrone Prozessoren finden seit Anfang der Neunziger-Jahre ein großes Forschungsinteresse, diesem regen Interesse zum Trotz finden sie jedoch keine breite Anwendung. Der Unterschied zu synchronen Prozessoren liegt darin, dass kein globales Taktsignal existiert, sondern die verschiedenen Schaltkreise sich selbst takten. Dies bedingt den Vorteil, dass man nicht an das strikte Zeitraster der synchronen Logik gebunden ist und die Gültigkeit von Daten mittels Handshaking signalisiert wird. Für das Quasi Delay Insensitive (QDI)-Schaltungsmodell, welches hier Anwendung findet, wird üblicherweise auf eine „Dual-Rail“-Codierung gesetzt, wobei alle Zustände durch zwei Leitungen repräsentiert werden (00,01,10,11). Damit kann die Funktion einer Schaltung für beliebige (auch veränderliche) Delays in Gattern und Leitungen garantiert werden. Permanente Fehler (beschrieben durch das sogenannte „Stuck-At“-Fehlermodell) einer QDI-Schaltung führen dazu, dass die, im Protokoll vorgesehene, Abfolge von Zuständen nicht mehr ordnungsgemäß eingehalten werden kann, was wiederum zu einem Fehlverhalten der Schaltung führt.

Ein solches Fehlverhalten kann, wie von [PN95, LM04] beschrieben, drei verschiedene Ausprägungen haben:

Deadlock Es kommt zum Stillstand der Schaltung.

Late Detection Wie Deadlock, aber mit Stillstand in unbestimmter Zukunft.

Premature Fire Vorzeitige Änderung des Signalzustandes am Ausgang einer Schaltung.

Das Fail-Stop-Verhalten von Deadlock und Late Detection ist in vielen Fällen wünschenswert, da es zu keiner falschen Ausgabe führt und die Schaltung nach Entfernung der Fehlerursache ohne explizite Recovery-Maßnahmen weiterarbeiten kann. In der existierenden Literatur ist

bis dato noch nicht erörtert, unter welchen Bedingungen und wie wahrscheinlich (relativ zu den anderen Manifestationen) dieses Verhalten in realen Schaltungen tatsächlich auftritt.

1.2 Ziel der Arbeit

Die Hauptaufgabe dieser Arbeit liegt in der Entwicklung eines Modells, welches zur Vorhersage des Verhaltens von asynchronen Schaltungen dient, welche durch Stuck-At-Faults beeinträchtigt sind. Es wird der Frage nachgegangen, welche möglichen Auswirkungen die auftretenden permanenten Fehler auf asynchrone Logik haben können. Dabei soll geklärt werden in welchem Zusammenhang die einzelnen Parameter des Schaltungsdesigns mit der Auftrittswahrscheinlichkeit der Effekte stehen.

Eventuelle Vor- und Nachteile verschiedener Designtechniken sollen betrachtet werden. Gesucht ist die Antwort darauf, unter welchen Bedingungen eine asynchrone Schaltung möglichst zuverlässig arbeitet. Es sollen Richtwerte zur Auftrittswahrscheinlichkeit von negativen Effekten und Schwankung dieser Effekte ausgearbeitet werden. Vorschläge zur Minimierung unerwünschter Effekte sollen resultieren.

1.3 Aufbau

Die Diplomarbeit ist in Kapitel unterteilt, welche so weit als möglich in sich selbst abgeschlossen sind.

Kapitel 2 führt den Leser in das Themengebiet der asynchronen-Logik ein. Das Wissen über verschiedene Verfahren zur Implementation asynchroner Schaltungen soll aufbauend vermittelt werden.

Kapitel 3 befasst sich mit der Entwicklung eines Modells, welches für die Vorhersage von Effekten durch Stuck-At Faults verwendet wird.

Kapitel 4 beschreibt die Simulation mehrerer QDI-Pipelines, welche von Stuck-At-Faults beeinträchtigt sind. Es beinhaltet eine Detailanalyse der auftretenden Effekte, sowie eine Gegenüberstellung dieser Effekte mit dem in Kapitel 3 erstellten Modell.

Kapitel 5 dient als eine Zusammenfassung der Arbeit und gibt Ausblick auf mögliche weitere Forschungsfelder.

Grundlagen

2.1 Übersicht

Im diesem Kapitel werden wir uns die Zeit nehmen eine fundierte Basis zu schaffen, auf welche die folgenden Kapitel aufbauen. Aufgrund der meist englischsprachigen Fachliteratur, wird die englische Terminologie entsprechend verwendet, dabei jedoch jeweils die entsprechende Erklärung in deutscher Sprache beigefügt.

2.2 Logik Elemente

2.2.1 Basis Elemente

In diesem Dokument werden die nach IEC 60617-12 spezifizierten Gatter verwendet. Diese sind in Tabelle 2.1 gelistet.

2.2.2 Programmable Logic Array

Eine programmierbare logische Anordnung, in englischsprachiger Fachliteratur als Programmable Logic Array oder kurz PLA bezeichnet, ist eine Schaltung, welche aus hintereinander geschalteten UND- und ODER-Matrizen besteht. Durch ein PLA, sind logische Funktionen in disjunktiver Form einfach implementierbar, wie in Gleichung 2.1 dargestellt.

$$\bigvee_i \bigwedge_j (\neg) x_{ij} \quad (2.1)$$

Die UND-Matrix repräsentiert dabei die Konjunktionsterme, die ODER-Matrix die disjunktiven Verknüpfungen dieser.

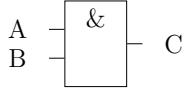
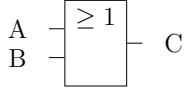
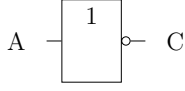
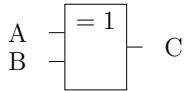
Name	Funktion	Symbol	Wahrheitstabelle															
UND-Gatter	$C = A \wedge B$		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	C	0	0	0	0	1	0	1	0	0	1	1	1
A	B	C																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
ODER-Gatter	$C = A \vee B$		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	C	0	0	0	0	1	1	1	0	1	1	1	1
A	B	C																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NICHT-Gatter	$C = \neg A$		<table border="1"> <thead> <tr> <th>A</th> <th>C</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	C	0	1	1	0									
A	C																	
0	1																	
1	0																	
Exklusiv ODER-Gatter	$C = A \oplus B$		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	C	0	0	0	0	1	1	1	0	1	1	1	0
A	B	C																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

Tabelle 2.1: Logikgatter

2.2.3 C Element

Das C-Element wurde 1959 von David E. Muller im Rahmen seiner Arbeit "A Theory of Asynchronous Circuits" [Mul59] das erste Mal erwähnt, und wird daher auch Muller C-Element genannt. Es handelt sich dabei um ein in asynchronen Schaltungen oft verwendetes Bauteil mit Hysterese. Das Verhalten eines einfachen C-Elementes mit zwei Eingängen a und b , sowie dem Ausgang c und dessen vorherigen Zustand c' , kann wie folgt durch eine boolesche Formel dargestellt werden:

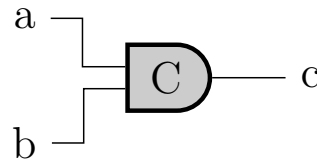
$$c = [c' \wedge (a \vee b)] \vee (a \wedge b) \quad (2.2)$$

Die dazugehörige Wahrheitstabelle und das Symbol für das Gatter sind in Abbildung 2.1 zu sehen.

Es gibt verschiedene Implementierungen welche diverse Vor- und Nachteile haben; zwei davon sind in Abbildung 2.2 zu sehen, zudem gibt es C-Elemente mit mehr als zwei Eingängen [NQA09, WV93]. In dieser Diplomarbeit ist allerdings nur das abstrakte Verständnis für ein C-Element von Bedeutung, deshalb sei dem Interessierten [SEE96] als weiterführende Literatur empfohlen.

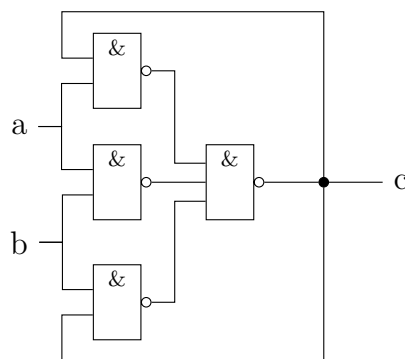
a	b	c
0	0	0
0	1	c'
1	0	c'
1	1	1

(a) Wahrheitstabelle

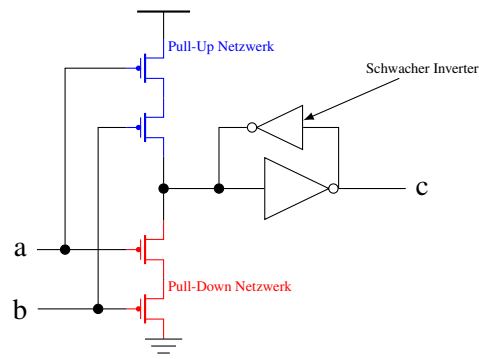


(b) Symbol

Abbildung 2.1: Muller-C-Element



(a) Schaltung aus NAND Gattern



(b) halb statische Schaltung

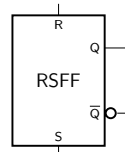
Abbildung 2.2: Muller C-Element Implementierungen

2.2.4 SR-Latch

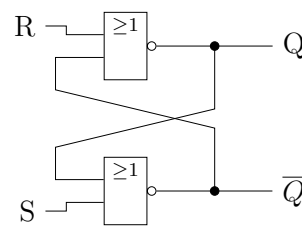
In der Elektronik ist ein Latch eine Schaltung, welche zwei stabile Zustände hat und zum Speichern von Informationen genutzt werden kann. Es gibt verschiedene Arten von Latches, im Bezug auf diese Arbeit, ist jedoch das SR (Set-Reset) Latch relevant. Bei der Verwendung von Basisgattern besteht die Möglichkeit, das Latch mittels NOR-Gatter zu erstellen. Ein RS-Latch

S	R	Q_{next}	Erklärung
0	0	Q	halte Zustand
0	1	0	reset
1	0	1	set
1	1	X	nicht erlaubt

(a) Wahrheitstabelle



(b) Symbol



(c) Aus NOR-Gattern gebaut

Abbildung 2.3: SR-Latch

besitzt zwei stabile Zustände, welche an einem Ausgang "Q" anliegen können. Diese Zustände werden "gesetzt" (set) und "zurückgesetzt" (reset) genannt. Zwischen diesen Zuständen kann

durch Signale an den Eingängen umgeschaltet werden. Mit einem Signal am „Setz“-Eingang, das heißt $S = 1$ und gleichzeitig $R = 0$, wird der Ausgang Q des Latch auf 1 gesetzt. Aktiviert man hingegen den „Rücksetz“-Eingang (also $R = 1$ und gleichzeitig $S = 0$), so wird das Latch zurückgesetzt: Am Ausgang Q liegt dann eine logische 0 an.

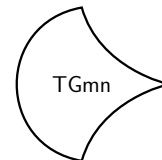
RS-Latches sind, sollte die Reihenfolge der auftretenden Zustandsänderungen am Eingang nicht eindeutig definiert sein, von metastabilen Zuständen betroffen. Es gibt Latches welche dieses Problem nicht haben, hierbei sei auf die Literatur [Rot92, Kapitel: „Latches and Flip-Flops“] verwiesen.

2.2.5 Threshold Gates

Ein m -aus- n -Threshold Gate (TG) mit Hysterese ist eine Generalisierung aus C-Element und ODER-Gatter [SF98]. Ein m -aus- n -TG mit Hysterese wird seinen Ausgang high setzen wenn m Eingänge high sind und seinen Ausgang low setzen, wenn alle seine Eingänge low sind. Hierbei entspricht das n -aus- n -TG einem C-Element und das 1 -aus- n -TG einem ODER-Gatter. Bei der Beschriftung wird die Form TG_{nm} angewandt, wobei für n und m die entsprechenden Werte eingesetzt werden und $1 \leq m \leq n$ gilt. Folglich lautet die Beschreibung für ein 2-aus-3-TG TG23 [SF10].

Ein_1	Ein_2	Ein_3	Aus_{next}	Erklärung
0	0	0	0	reset
0	0	1	Aus	hold
0	1	0	Aus	hold
0	1	1	1	set
1	0	0	Aus	hold
1	0	1	1	set
1	1	0	1	set
1	1	1	1	set

(a) Wahrheitstabelle eines TG23



(b) Symbol

m	n				
	1	2	3	4	5
1	Buffer	OR2	OR3	OR4	OR5
2		C2	TG23	TG24	TG25
3			C3	TG34	TG35
4				C4	TG45
5					C5

(c) m -aus- n Threshold Gatter mit Hysterese ($1 \leq m \leq n | n \in \{1..5\}$)

Abbildung 2.4: Threshold-Gatter

2.3 Petri-Netze

Wir fassen hier die Eigenschaften von Petri-Netzen zusammen, für eine detaillierte Darstellung sei die Lektüre von [Pet77] empfohlen.

Ein Petri-Netz ist ein gerichteter Graph, welcher zwei Arten von Knoten besitzen kann. Diese Knoten werden Stellen (Places) bzw. Transitionen genannt. Kanten innerhalb dieses Graphen können von einer Stelle zu einer Transition und umgekehrt führen. Um den Zustand eines Systems festzuhalten werden Marken (Token) an Stellen gesetzt. Bezogen auf eine Pipeline bezeichnet ein Token Daten (auch Datentoken) die durch eine Pipeline propagieren.

Damit die Dynamik eines Systems dargestellt werden kann, werden diese Marken über die Transitionen zu anderen Stellen entsprechend einem Regelset verschoben. Ein minimales Petri-Netz ist in Abbildung 2.5a dargestellt.

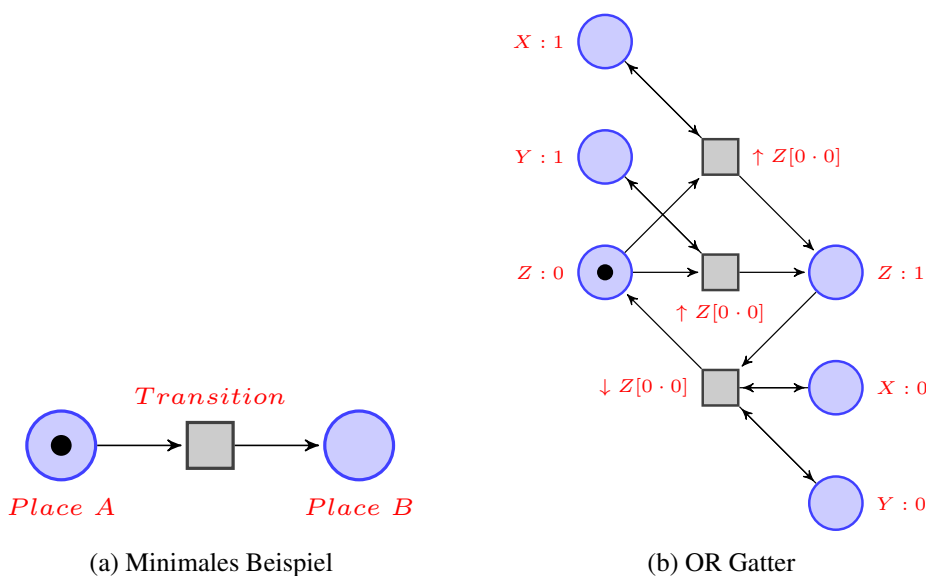


Abbildung 2.5: Petri-Netz

Mit Hilfe von Petri-Netzen ist es möglich logische Schaltungen abzubilden, hierfür wird die Logik entsprechend den von Petersons [Pet81] aufgestellten Regeln in ein Petri-Netz übersetzt. Es gibt eine Vielzahl an Varianten von Petri-Netzen [YKSK96], hier von Interesse ist das zeitbehaftete Petri-Netz (engl. Time Petri Net).

Die Zeit als beschränkenden Faktor in ein einfaches Petri-Netz einzuführen, bringt den Vorteil, das zeitliche Verhalten (engl. Timing) eines Systems besser darstellen zu können. Ein Petri-Netz erhält die Timing Information indem einer Transition ein Zeitbereich $[d(t_i) \cdot D(t_i)]$ zugewiesen wird. Hierbei beschreibt $d(t_i)$ den frühesten (engl. earliest firing time - EFT) und $D(t_i)$ den spätesten Zeitpunkt (engl. latest firing time - LFT) an welchem eine Transition ausgelöst werden kann. Dieser definierte Zeitbereich hat den Zeitpunkt t_0 sobald alle eingehenden Stellen eine Marke besitzen. In Abbildung 2.5b ist ein OR Gatter mit den Eingängen X , Y und

dem Ausgang Z zu sehen. In der Ausgangskonfiguration besitzt dieses eine Marke an der Stelle $Z : 0$, womit der Ausgang Z als Low festgelegt wird. An den Eingängen sind zu diesem Zeitpunkt keine Marken vorhanden. Sollte nun die Auslösebedingung für eine Transition erfüllt werden, so würden diese entsprechend ihrer Definition $\uparrow Z[0 \cdot 0]$ ohne Verzögerung auslösen.

2.4 Pipeline

Einfache logische Systeme deren Ausgänge nur von Eingängen abhängen, können mittels Schaltnetzen (engl.: combinational circuit) implementiert werden. Bei komplizierteren Systemen, d.h. Systeme die nicht nur Basisfunktionen implementieren ist es jedoch oft nötig die Funktionen in mehrere Teilaufgaben aufzuspalten. Diese Teilaufgaben - Pipeline-Stufen genannt - werden nacheinander abgearbeitet. Die einzelnen Pipeline-Stufen arbeiten parallel nebeneinander, wodurch mehrere Aufgaben gleichzeitig abgearbeitet werden können. Damit die einzelnen Stufen unabhängig bleiben, werden dazwischen Buffer-Elemente, auch Statusregister genannt, geschaltet. Das Aufteilen der Funktionen hat mehrere Vorteile:

1. Einzelne Schaltnetze können von mehreren Funktionen überlappend genutzt werden.
2. Das Ergebnis eines Schaltnetzes kann als Eingang für mehrere weitere Funktionen benutzt werden.
3. Dadurch, dass jede Pipeline-Stufe unabhängig ist und mit dem Beginnen einer neuen Berechnung nicht auf das Beenden des vorherigen Zyklus gewartet werden muss, bieten sie einen höheren Durchsatz.

Das Timing einer Pipeline wird durch synchrone oder asynchrone Designmethoden festgelegt [Spa01, Kapitel 1.3]; Die folgenden zwei Kapitel werden auf diese beiden Methoden eingehen.

2.4.1 Timing

Es gibt zwei Möglichkeiten eine Schaltung zu steuern: auf Zeit- oder auf Event-basierend. Bei einer Zeit-basierenden Steuerung gibt es ein globales Taktsignal, welches dafür sorgt, dass die Schaltung synchron arbeitet. Events halten sich nicht an einen vorgegebenen Takt und sind daher asynchron.

2.4.1.1 Synchrone Pipeline

Wie in Abbildung 2.6 zu sehen ist, werden die Register einer synchronen Pipeline von einem globalen Taktsignal getrieben.

Die Zeit zwischen aufeinanderfolgenden aktiven Taktflanken muss größer sein, als das längste Delay zwischen zwei Pipeline-Stufen. Dadurch wird sichergestellt, dass die Berechnungen in allen Pipeline Stufen abgeschlossen sind und die Daten stabil am Eingang der Register anliegen.

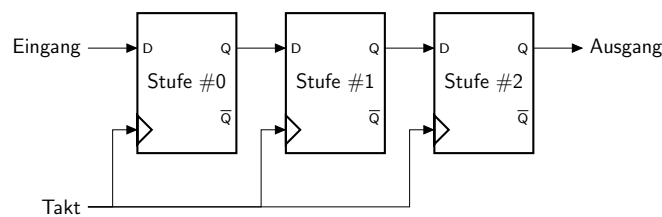


Abbildung 2.6: Synchroner Pipeline mit globalen Taktsignal

Abbildung 2.7 zeigt die Datenverarbeitung innerhalb einer Pipeline, wobei die Daten bei jeder steigenden Taktflanke von der nächsten Pipeline-Stufe übernommen werden. D_0 bis D_3 bezeichnen vier verschiedene Datensets, welche von der Pipeline verarbeitet werden. Jede Stufe benötigt je nach Eingangsdaten unterschiedlich lange für die Verarbeitung, demzufolge variiert auch der Zeitpunkt, an welchem die Daten am Ausgang zur Verfügung stehen. Die nicht beschrifteten Zyklen stellen jene Zeit dar, welche eine Stufe auf das nächste Taktsignal warten muss. Stufe 2 benötigt für die Verarbeitung von D_0 die gesamte Zeitspanne eines Taktes. Bei dieser benötigten Zeitspanne handelt es sich um den sogenannten Kritischen Pfad, der ausschlaggebend für die maximal verwendbare Taktfrequenz ist. Sollte durch Verwendung einer höheren Taktfrequenz die Verarbeitung von D_0 in Stufe 2 noch nicht abgeschlossen sein, kommt es zu einem fehlerhaften Verhalten. Stufe 3 benötigt zur Verarbeitung einen Bruchteil des Taktes, daher ruht die Stufe einen Großteil eines Zykluses. Trotzdem ist eine Erhöhung der Taktfrequenz nicht möglich, da diese dem worst-case Szenario anzupassen ist.

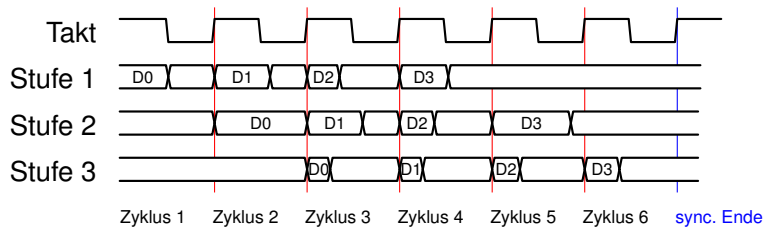


Abbildung 2.7: Datenflussdiagramm einer synchronen Pipeline

2.4.1.2 Asynchrone Pipeline

In asynchronen Pipelines werden, wie der Name schon sagt, die einzelnen Register asynchron getaktet, dies geschieht durch die Erkennung des Ergebnisses an den Ausgängen der Pipeline-stufen. Das globale Taktsignal wird durch lokale, für jede Stufe das Taktsignal erzeugende, Controller ersetzt. Diese Controller verwenden wie in Abbildung 2.8 dargestellt, sogenannte Request/Acknowledge Signale, um einen Handshake zwischen aufeinander folgenden Stufen zu signalisieren und Daten zu übermitteln.

In Abbildung 2.9 werden die entsprechenden Datenströme dargestellt. In der asynchronen Pipeline-Stufe wird das Timing der einzelnen Stufen lokal von jeder Stufe erzeugt, wodurch

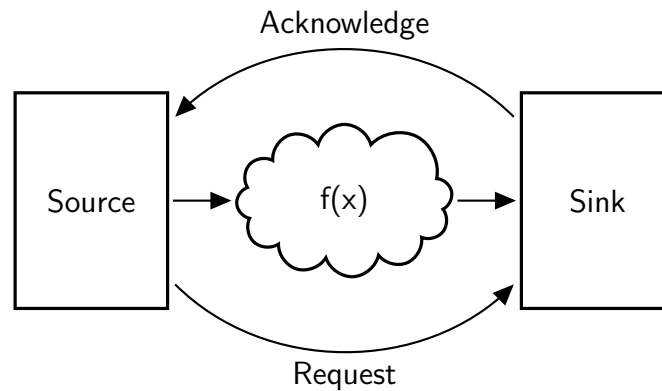


Abbildung 2.8: Asynchrone Pipeline mit globalen Taktsignal

diese Pipeline schneller als die synchrone ist. Das Timing wird dynamisch der Verarbeitungszeit für die Daten in einzelnen Stufen angepasst und hat bei Unausgeglichenheit zur Folge, dass:

1. Eingangsdaten anliegen, jedoch die nächste Stufe noch nicht bereit zur Verarbeitung neuer Daten ist und dadurch die Verarbeitung **blockiert (engl. blocking)** wird.
2. Die Stufe zur Verarbeitung neuer Daten bereit ist, aber keine Eingangsdaten vorliegen, weshalb diese Stufe verhungert (engl. starving).

Starving ist die Folge von zu wenigen Datentoken (Erklärung in Kapitel 2.3) in einer Pipeline und führt zu einem geringen Durchsatz. Sind hingegen zu viele Datentoken vorhanden blockieren sich die Stufen gegenseitig.

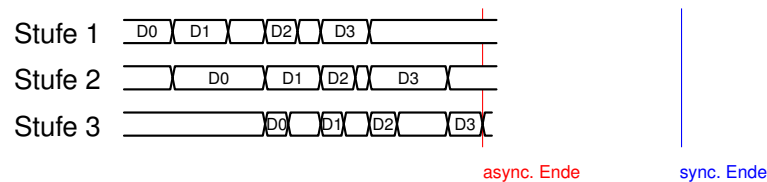


Abbildung 2.9: Datenflussdiagramm einer asynchronen Pipeline

Um den oben beschriebenen Handshake durchzuführen ist es nötig, das Ende der Berechnung einer Pipeline-Stufe zu erkennen. Dies geschieht bei asynchronen QDI-Pipelines durch die Wahl einer geeigneten Codierung. Die Information darüber, welche Daten zusammengehören, wird demzufolge nicht explizit durch ein einzelnes Signal übertragen, sondern implizit über die Informations Domäne. Es ist die Verwendung jeder *m*-aus-*n*-Codierung möglich, wobei ein Symbol durch *m* Transitions auf *n* Leitungen repräsentiert wird. Eine Übersicht über diese Verfahren bietet Tabelle 2.2. Üblicherweise finden *1*-aus-*2*-Codierungen, auch Dual-Rail genannt, Anwendung. Eine repräsentative Auswahl, Null Convention Logic, Level Encoded Dual Rail sowie Code Alternation Logic, wird in Kapitel 2.7 erläutert.

2.4.2 Eigenschaften einer Pipeline

Ausgehend von der Unvollständigkeit der booleschen Logik [Del05, Abschnitt 2.3.2] ergeben sich mehrere fundamentale Eigenschaften, welche bei der Übergabe von Daten (z.B. zwischen mehreren Pipelinestufen) sichergestellt werden müssen. Darunter Validität, Konsistenz und Verlustfreiheit der verarbeiteten Daten.

Man bezeichnet einen Eingangsvektor als **konsistent** zu einem Zeitpunkt t_i , wenn alle Signalzustände dem selben Kontext, an einem Zeitpunkt t_i , zugehörig sind. Dabei wird nicht nur das Datenwort als konsistent bezeichnet, sondern auch die dazugehörigen Signale. Ein Signal wird als **valide** bezeichnet, sobald dieses zu einem Zeitpunkt t_i das Ergebnis einer logischen Operation auf einen konsistenten Eingangsvektor darstellt. Z.B. die Verwendung eines 2 Bit Eingangsvektors $[E_1, E_2]$ mit LEDR als Codierung (Details siehe Abschnitt 2.7.2). LEDR baut auf zwei alternierende Phasen zur Repräsentation logischer Werte, wo zuerst $[E_1(\varphi_0), E_1(\varphi_0)]$ und darauffolgend $[E_1(\varphi_1), E_1(\varphi_1)]$ an den Eingängen erwartet wird. Verändert ein auftretender Fault den Eingangsvektor kommt es zu dem Zustand $[E_1(\varphi_0), E_1(\varphi_1)]$. Das Eingangssignal ist nun inkonsistent und kann daher auch zu keinem validen Ausgangssignal führen.

Um die Konsistenz und damit auch die Validität eines Signales sicherzustellen werden sogenannte Phasen- oder auch Completion Detektoren verwendet, welche in den Abschnitten 2.7.2.2, 2.7.2.1 und 2.7.1.1 beschrieben werden.

Um die **Verlustfreiheit** der Datenverarbeitung zu garantieren, ist es nötig der Datenquelle (engl: Source) mitzuteilen, wann diese dem Datenempfänger (engl: Sink) neue Daten zur Verfügung stellen kann. Umgekehrt muss dem Empfänger übermittelt werden, wann die Daten übernommen (engl. consumed) werden können.

Der zweite Punkt kann realisiert werden, indem die Konsistenz und Validität des Eingangsvektors am Empfänger überprüft werden. Für den Sender ist es hingegen nur dann möglich die Information zu erhalten, wenn diese ihm vom Empfänger, nach erfolgreicher Datenübernahme, explizit mitgeteilt wird.

2.4.3 Logische Organisation

Es gibt eine Vielzahl an Pipeline Typen, diese verschiedenen Typen müssen unterschiedliche Anforderungen erfüllen. Eine fundamentale Eigenschaft ist die Linearität einer Pipeline.

2.4.3.1 Lineare Pipeline

Eine lineare Pipeline ist eine Abfolge von Funktionsblöcken, welche linear angeordnet sind um eine spezielle Funktion auf einen Datenstrom anwenden zu können. Beispielanwendungen von linearen Pipelines sind nichtrekursive digitale Filter (engl. finite impulse response filter), arithmetische Operationen oder auch Speicherzugriffe. Es handelt sich im Prinzip um eine einfache First In First Out Pipeline (FIFO), bei der nur die letzte Stufe eine Ausgabe erzeugt.

Eine Aufgabe A kann wie in Abbildung 2.10 in n Teilaufgaben zerlegt werden. Jede Teilaufgabe befindet sich in einer sogenannten Pipeline Stufe. Dies bedingt das Speichern der Zwi-

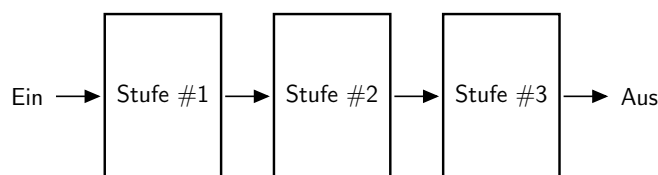


Abbildung 2.10: Lineare Pipeline

schenergebnisse in Registern für deren Steuerung jeweils ein Completion Detector (CD) verwendet wird. Die Aufgabe eines CD liegt darin, die Konsistenz und Validität eines Codewortes sicherzustellen und das Ergebnis der Kontroll-Logik mitzuteilen. Die Pipeline funktioniert wie folgt: Hat die nachfolgende Stufe die Daten übernommen, dann wird mittels Wechsel des ACK-Signals, dies der aktuellen Stufe mitgeteilt. Bei jedem Wechsel des ACK-Signals können auch in der aktuellen Stufe neue Daten übernommen werden. Also wird sichergestellt, dass noch nicht übernommene Daten nicht verloren gehen.

In letzter Zeit wurden viele neue asynchrone Pipeline Templates vorgestellt [FF14]. Diese Strukturen sind oft für einfache FIFO Strukturen ausreichend, aber nicht für komplexe nichtlineare Pipelines, welche Forks und Joins (siehe nachfolgenden Abschnitt) enthalten.

2.4.3.2 Nichtlineare Pipeline

Eine nichtlineare Pipeline, manchmal auch dynamische Pipeline genannt, bietet die Möglichkeit verschiedene Funktionen zu unterschiedlichen Zeitpunkten auf Daten anzuwenden. Im Gegensatz zu einer linearen Pipeline gibt es bei ihr auch Vorwärts- (engl. Feed-Forward) und Rückwärtsverbindungen (engl. Feedback). Die Generierung von Ausgabe ist nicht auf die letzte Stufe beschränkt, sondern kann sich aus mehreren Zwischenergebnissen zusammensetzen. Die dadurch ermöglichte dynamische Nutzung geht mit einer erhöhten Komplexität der Steuerung einher.

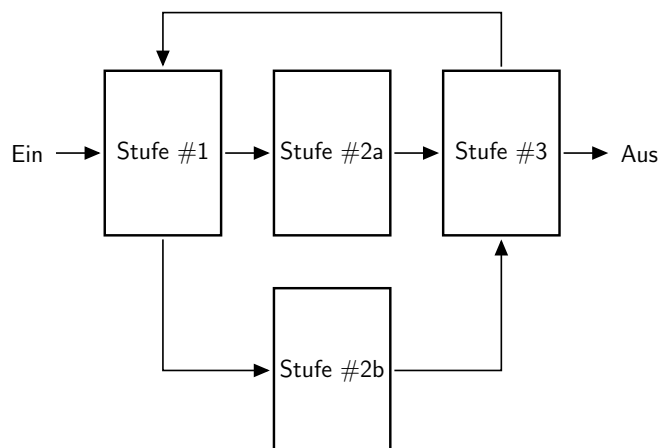


Abbildung 2.11: Nichtlineare Pipeline

Die in Abbildung 2.11 dargestellte Pipeline hat in den Stufen #1 und #3 an den Eingängen ein Join und an den Ausgängen ein Fork.

Bei einem **Fork** handelt es sich um die Aufteilung des Datenstroms auf mehrere aufeinanderfolgenden Stufen. Dies wird in der Regel dadurch erreicht, dass die entsprechenden Signale für mehrere Stufen als Eingänge dienen. Für das Acknowledge ist es nötig, dass die Losslessness-Eigenschaft bei allen Empfängern vorhanden ist. Dadurch, dass die von den Empfängern kommenden ACK Signale mittels eines C-Elements synchronisiert werden, siehe auch Abbildung 2.12a, wird dies erreicht.

In QDI-Schaltungen gibt es eine Sonderform des Forks, das so bezeichnete Isochrone Fork, bei welchem angenommen wird, dass alle Stufen die Daten erhalten haben, sobald eine einzelne dies hat. Dies ist in Abbildung 2.12b ersichtlich.

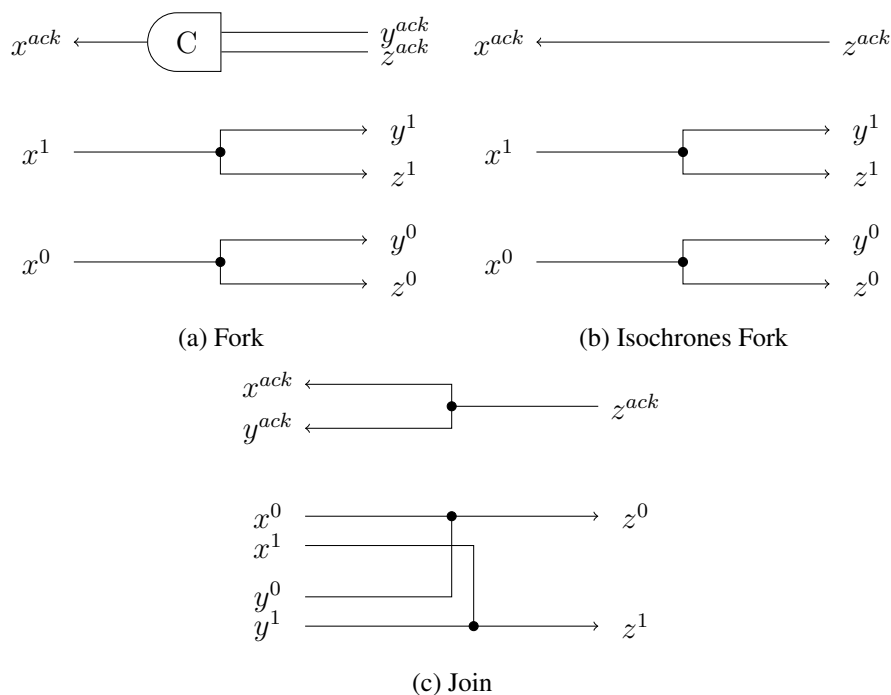


Abbildung 2.12: Fork und Join

Bei einem **Join** handelt es sich um die Vereinigung mehrerer Datenströme zu einem Datenstrom. Dabei werden die Signale von Vorgängerstufen an die Eingänge gelegt. Sobald diese die Konsistenz, Validität und Losslessness-Eigenschaft erfüllen, wird vom Empfänger ein ACK Signal an alle Sender, wie in Abbildung 2.12c ersichtlich, geleitet.

2.5 Delay Modell

Das Delay-Modell spezifiziert die Annahmen über Delays von Gattern und Signalen während des Design-Prozesses. Allgemein gilt, desto restriktiver die Annahmen über ein Delay sind, desto robuster ist auch das Design eine Schaltung gegenüber Variationen des Delays. Dies ist ein essentieller Punkt, da das Delay oft aufgrund unvorhersehbarer Einflüsse stark variiert.

Als das robusteste Delay-Modell gilt das Delay Insensitive (DI) Design [Udd86], bei welchem keine Annahmen über das Delay von Gattern und Signalen getätigt werden. Dies macht DI Schaltungen sehr resistent gegenüber allen Formen von Variationen. Es wurde jedoch gezeigt [Mar90], dass es nicht möglich ist praktisch nutzbare Schaltungen mit vertretbarem Aufwand zu erzeugen.

Als Kompromiss wurde das QDI-Design [Mar86] vorgeschlagen, welches breite Verwendung findet. Generell wird bei QDI wie bei DI, keine Annahme über die Delays von Gattern und Signalen getroffen. Allerdings gibt es eine Ausnahme, diese betrifft bestimmte Forks von Signalen, die sogenannten isochronen Forks. Ein isochrones Fork ist definiert als ein Fork, bei welchem das Delay an allen daraus hervorgehenden Signalen das Selbe sein muss. Da es in der Praxis nicht möglich ist eine solch strikte Forderung umzusetzen, wird sie entsprechend ihrem Sinn interpretiert, welcher darin liegt, mit einer definierten Reihenfolge von Transitionen an den Eingängen der Gatter die Hazard-Freiheit sicherzustellen [SN01b].

2.6 Handshake Protokolle

Um die von einer asynchronen Pipeline benötigte Anforderung und Bestätigung des Datenerhalts zu standardisieren wurden Handshake Protokolle eingeführt. Diese werden entsprechend der vorkommenden Phasen benannt. Vier-Phasen-Handshake-Protokolle sind, durch die geringe Komplexität der zu verwenden Logik, für eine lokale Kommunikation geeignet. Bei der Kommunikation zwischen mehreren Bereichen eines Chips wird oft auf ein Zwei-Phasen-Protokoll gesetzt [MAMN08, DMKG10], da dieses weniger Energie benötigt und höhere Transferraten erreicht.

2.6.1 Vier-Phasen-Handshake

Das Vier-Phasen-Handshake-Protokoll ist ein sogenanntes Return To Zero (RTZ)-Handshake-Protokoll. Das Request (REQ)- und Acknowledge (ACK)-Signal startet im Low-Zustand (Zero) und kehrt am Ende des Handshake-Zyklus zu diesem zurück. Das Vier-Phasen-Protokoll ist in Abbildung 2.13 dargestellt. Wie der Name bereits nahe legt, gibt es auf den REQ- und ACK-Signalen vier Transitionen pro Zyklus. Ein Request wird durch eine steigende Flanke des REQ-Signals signalisiert. Das entsprechende Acknowledge-Signal wird mittels steigender Flanke des ACK-Signals, durch den Empfänger, erzeugt. Sobald ACK und REQ high sind befindet sich das Protokoll in der Arbeitsphase. Wird das Acknowledge-Signal erhalten, so führt dies zu einer fallenden Flanke des REQ-Signals und daraufhin zu einem fallenden ACK-Signal, womit

ein Zyklus abgeschlossen ist. Während die Signale REQ und ACK low sind befindet sich das Protokoll in der Reset Phase.

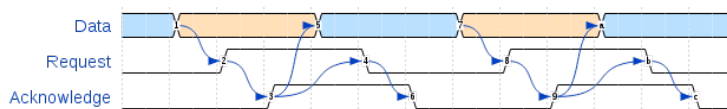


Abbildung 2.13: Vier Phasen Handshake

Durch seine Einfachheit findet das Vier-Phasen-Handshake-Protokoll Verwendung. Diese Einfachheit wird mit einer geringen Effizienz erkauft, da während der Reset-Phase keine Berechnungen durchgeführt werden können.

2.6.2 Zwei-Phasen-Handshake

Bei einem Zwei-Phasen-Handshake Protokoll besteht jeder Zyklus aus zwei Transitions (steigender oder fallender Flanke) durch die REQ- und ACK-Signale (siehe Abbildung 2.14). Es handelt sich hierbei um ein No Return To Zero (NRZ)-Protokoll.

Ein Request wird durch eine Transition auf dem REQ-Signal erzeugt. Der Empfänger bestätigt die Beendigung der Work-Phase mittels Transition auf dem ACK Signal. Die Working-Phase wird als Dauer zwischen Senden des REQ- und Erhalt des ACK-Signals definiert.

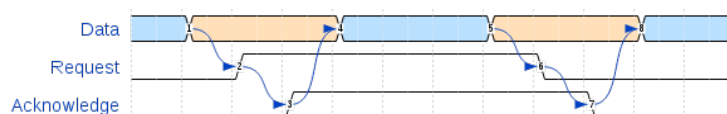


Abbildung 2.14: Zwei Phasen Handshake

Im Gegensatz zum Vier-Phasen-Protokoll kann hier der nächste Zyklus direkt nach der Working-Phase begonnen werden. Dies führt zu Controllern, welche einen höheren Takt und damit einen höheren Durchsatz erreichen, so der MOUSETRAP-Controller [SN01a], welcher mit bis zu 2.4 GHz betrieben werden kann [SN07].

2.7 Dual-Rail Codierung

Da bei asynchronen Schaltungen kein Taktsignal die Gültigkeit der Daten signalisiert, muss diese Information über die Daten auf dem Bus implizit transportiert werden. Dies ist mit normaler boolescher Logik nicht möglich, da es unmöglich festzustellen ist ob eine 0 oder 1 auf der Leitung gültig ist. Von Dual-Rail-Codierung¹ spricht man bei der Verwendung von zwei Leitungen zur Darstellung eines einzelnen Bits. Hierbei kann jede Leitung den Wert 0 oder 1 annehmen. Wie Tabelle 2.2 zeigt gibt es verschiedene Möglichkeiten dies zu Realisieren. Nachfolgend werden die für diese Arbeit relevanten erklärt.

¹Rail wird in diesem Zusammenhang als Bezeichnung für das gesamte Signal verwendet

²Return to Zero

Encoding Scheme		Throughput Metric (RZ vs. NRZ)	Coding Efficiency (bits/wire)	Transition Power Metric (transitions/bit /transaction)	Latency Overhead Metric (due to Completion Detector)	Timing Constraints
RZ ²	1-of-2 (NCL)	RZ	$\frac{1}{2}$	2	OR2/bit into n-input C-element	DI
	1-of-N	RZ	$\frac{\lceil \log N \rceil}{N}$	$\frac{2}{\lceil \log N \rceil}$	ORn/bit into $\lceil \frac{N}{\lceil \log N \rceil} \rceil$ -input C-element	DI
	M-of-N	RZ	$\frac{\lceil \log \binom{n}{m} \rceil}{N}$	$\frac{2}{\lceil \log \binom{n}{m} \rceil}$	M-of-N majority function into Celement	DI
LETS ³	1-of-2 (LE-DR)	NRZ	$\frac{1}{2}$	1	XOR2/bit into n-input C-element	DI
	1-of-N	NRZ	$\frac{\lceil \log N \rceil}{N}$	$\frac{1}{\lceil \log N \rceil}$	XORn/bit into $\lceil \frac{N}{\lceil \log N \rceil} \rceil$ -input C-element	DI
Misc	Transition Signaling	NRZ	$\frac{1}{2}$	1	XOR2/bit into n-input C-element	DI
	Bundled-data	N-/RZ	1	$\frac{1}{2}$	Wire with delay	Bundled path delay
	Pulse-mode	RZ	$\frac{1}{2}$	2	Complex [EFS07]	Pulse with Timing

Tabelle 2.2: Comparison of Asynchronous Global Communication Coding Schemes [MAMN08]

2.7.1 Three State Logic

Three State Logic (TSL) findet in verschiedenen Bereichen Anwendung [KAGEBR82, GM88, FB96a], wobei der Code drei States hat: TRUE, FALSE und einen Null-State. Nach jeder Datenübertragung (TRUE oder FALSE) wird ein Null (N) als Separator übertragen. Im Folgenden wird daher auch von Null Convention Logic (NCL) gesprochen. Daten sind als Werte zwischen zwei aufeinander folgende Separatoren, den Null-Codeworten, definiert. Diese Definition gilt dabei ohne Referenz auf einen Zeitpunkt. In Tabelle 2.15a und 2.15b sind das logische Mapping und die entsprechenden Phasenübergänge zu sehen. Resultierend aus der NULL-Phase, in welcher keine Berechnungen durchgeführt werden, besitzt TSL gegenüber FSL einen um 50% geringeren Datendurchsatz. Mit dem Ziel den Datendurchsatz zu erhöhen, wurden in [Smi06] ein entsprechendes Verfahren, NULL Cycle Reduction (NCR), vorgestellt.

Durch die Verwendung einer solchen Codierung wird verhindert, dass mehrere Signale eine Flanke zum selben Zeitpunkt besitzen. Hierdurch werden unerwünschte Effekte (Hazards) vermieden. Zusätzlich wird durch abwechselnde Null- und Daten-Wellen die Logik vereinfacht, wobei sich jedoch auch der Datendurchsatz verringert, da die Wellen nacheinander übertragen werden müssen [DGY92]. Hierfür gibt es einen Ansatz genannt NCR [Smi06], welcher den Durchsatz bis zu einem Faktor von 1.6 zu steigern vermag.

³Level Encoded Transition Signaling

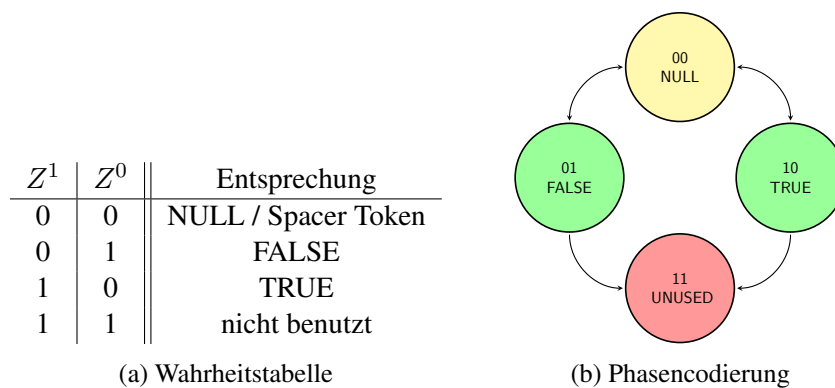


Abbildung 2.15: Three State Logic

2.7.1.1 TSL Completion Detector (CD)

Um das Ende der Null bzw. Data Phase auf einem n -Bit breiten Datenbus zu erkennen, wird ein CD, bestehend aus n OR und einem C-Element verwendet. Der Aufbau dieser Schaltung ist in Abbildung 2.16 zu sehen. Die Null-Phase wird am Ausgang des C-Elements durch logisch 0 definiert und die Daten Phase durch logisch 1. Die OR-Gatter haben als Eingang beide Rails eines Bits, dadurch entsprechen die Ausgänge der OR-Elemente der Null- bzw. Daten-Phase dieses Bits. Durch das C-Element wird der Ausgang geschaltet, wenn alle Werte am Eingang einer Phase entsprechen, und dadurch eine Synchronisation erreicht wurde [DGY92].

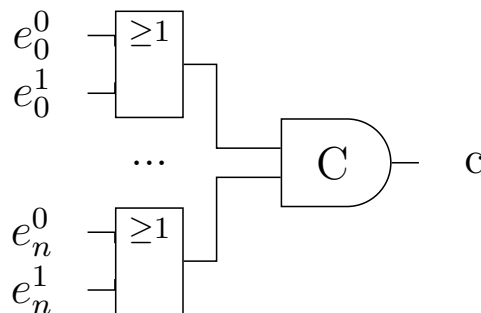


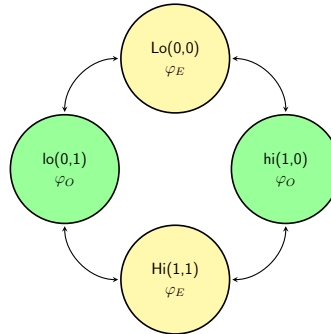
Abbildung 2.16: TSL CD, Spacer state = 0, Data State = 1

2.7.2 Four State Logic (FSL)

Four State Logic (FSL) erlaubt den Daten, wie in Tabelle 2.17a dargestellt, in einem von vier Zuständen zu sein, wobei je zwei einen TRUE (Hi,hi) bzw. FALSE(Lo,lo) Zustand beschreiben. Bei der Übertragung von Daten ist es nötig zwischen den beiden Phasen (φ_0 und φ_1) umzuschalten. Wie in Abbildung 2.17b zu sehen ist, ist damit zwischen jedem Datenwert eine Transition und somit eine Null-Phase nicht mehr notwendig.

Z^1 (Value)	Z^0 (Phase)	Phase	Single Rail Entsprechung
0	1	φ_0 ODD	0
1	0	φ_0 ODD	1
1	1	φ_1 EVEN	1
0	0	φ_1 EVEN	0

(a) Wahrheitstabelle



(b) Phasencodierung

Abbildung 2.17: Four State Logic

Das Alternieren zwischen φ_0 und φ_1 sorgt dafür, dass sich immer nur Z^1 oder Z^0 durch eine Transition ändert. Dies führt dazu, dass die Phase einer Rail mittels XOR festgestellt werden kann. Es gibt flankenorientierte Arten der FSL wie das Level Encoded Dual Rail (LEDR), und zustandsorientierte, wie das Code Alternation Logic (CAL). Diese sind untereinander kompatibel.

2.7.2.1 LEDR CD

Als Eingänge für das XOR werden beide Rails eines Bits genutzt, womit die alternierenden Phasen erkannt werden. Mittels C-Element werden, gleich dem Completion Detektor, für Three-State Coding alle Bits synchronisiert [DWD91].

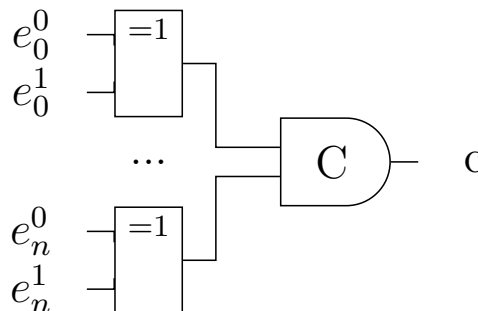


Abbildung 2.18: LEDR CD, Even Phase = 0, Odd Phase = 1

2.7.2.2 CAL CD

Die von Martin Delvai in seiner Dissertation [Del05] vorgeschlagene Code Alternation Logic, kurz CAL, gehört wie LEDR zu der Gruppe der Four-Stage Logic. Es verwendet die selbe Codierung (siehe Abbildung 2.17b) für die Signale.

Wichtigster Unterschied zwischen LEDR und CAL ist die Verwendung eines RS-Latch als Speicherelement (siehe Abbildung 2.19). Anstatt eines C-Elements, wie bei LEDR, wird bei CAL ein RS-Latch zum Erzeugen einer Hysterese verwendet. Je ein Rail wird mittels XOR verknüpft, dessen Ausgang einem NOR bzw. UND-Gatter als Eingang dient. Die Aufgabe des UND-Gatters ist den Set-Eingang des RS-Latches zu triggern, sobald alle Eingänge des CD in Phase φ_0 sind. Umgekehrt triggert das NOR-Gatter den Reset-Eingang, sobald alle Eingänge des CD in Phase φ_1 sind.

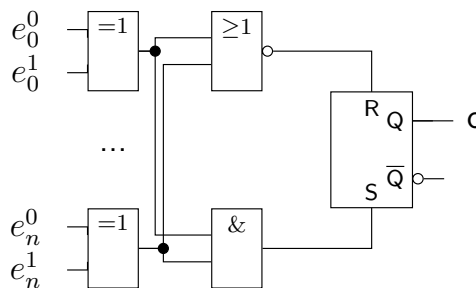


Abbildung 2.19: CAL CD, Even Phase = 0, Odd Phase = 1

2.8 Dual Rail Logik

Bei codierten Datenströmen ist es nötig, logische Funktionen auf spezielle Weise zu implementieren. Nachfolgend soll für die in der Arbeit verwendeten Arten der Dual-Rail Codierung ein Überblick geboten werden.

2.8.1 Self Timed Logic Module (STLM)

Bei einem Self Timed Logic Module (STLM), wie es von [DGY92] vorgestellt wurde, handelt es sich um eine Möglichkeit logische Funktionen für NCL zu implementieren. Die Implementation besteht aus vier Subnetzen, welche so verbunden sind, wie in Abbildung 2.20 dargestellt. Das Subnetz ORN bestimmt, ob der Eingang definiert ist oder nicht. Im Subnetz CEN wird die Synchronisation aller Eingänge aus ORN sichergestellt. Im Subnetz DRN wird die tatsächliche Logik implementiert. Durch das Subnetz OUTN wird sichergestellt, dass die Zustände definiert bleiben.

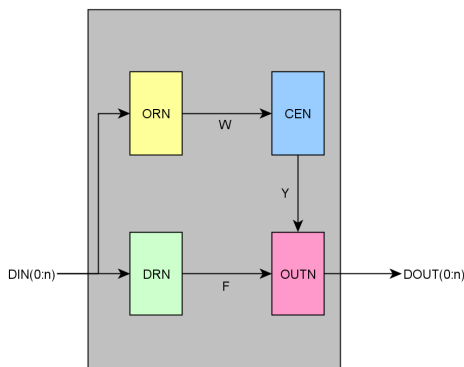


Abbildung 2.20: TSL Element nach [DGY92]

2.8.2 Delay Insensitive Minterm Synthesis (DIMS)

Dieser Ansatz [SSDS92] erhält seinen Namen dadurch, dass die Schaltung Delay insensitiv ist und die C-Elemente alle Minterme der Eingänge erzeugen. Durch erstellen einer Wahrheitstabelle mit drei Gruppen von Zeilen wird die logische Funktion erzeugt. In der ersten Gruppe wird das leere Codewort definiert, indem die Schaltung den Ausgang auf NULL zurück setzt. In der zweiten Gruppe werden jene Codewörter definiert, welche keine Auswirkungen auf den Ausgang haben. In der letzten Gruppe werden die gültigen Codewörter definiert, auf welche die Schaltung mit setzen des korrekten Ausgangswertes reagiert. Entsprechend der Wahrheitstabelle werden die einzelnen Signale verbunden.

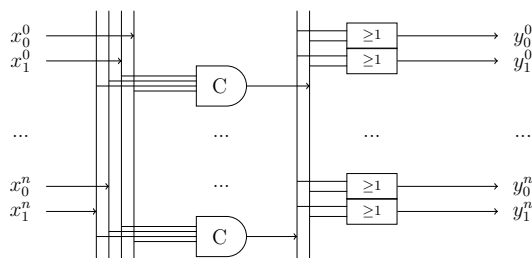


Abbildung 2.21: Aufbau DIMS Gatter, Minterme nicht verbunden

2.8.3 Threshold Gate (TG)

C-Elemente und OR-Gatter aus den vorherigen Abschnitten können als n -aus- n -TG und 1 -aus- n -TG mit Hysterese gesehen werden. Durch das Verwenden von TG ist es möglich, die Komplexität zu verringern. TG können sowohl für NCL [FB96b] als auch für LEDR-Logik [DWD91] verwendet werden. Betrachtet man sie auf einem höheren Abstraktionslevel so gibt es keinen

Unterschied zu DIMS bzw. STLM, demzufolge wird auch auf eine weitere Ausführung verzichtet. Dem interessierten Leser sei die Lektüre von [JGJ⁺07] empfohlen.

2.8.4 Programmable Logic Array (PLA)

Das Verwenden eines PLA ist eine weitere Methode um logische Funktionen zu implementieren. Diese Methode kann bei allen Technologien verwendet werden, da bei ihr lediglich der Nutzung einer technologiespezifischen Art für das Halten des Ausgangs bedarf. Konkrete Beispiele dafür sind LEDR-PLA [DWD91], CAL [DFH⁺05] als auch das bereits vorgestellte STLM (siehe Abschnitt 2.8.1).

2.9 Logical Effort

Das Verfahren des Logical Efforts wurde 1991 von Ivan Sutherland entwickelt und in seinem Buch [SSH99b] vorgestellt. Es handelt sich dabei um eine Technik zum Bestimmen des Delays in einer CMOS Schaltung. Damit ist es möglich, die optimale Größe von Gattern zu berechnen und ein minimales Delay zu erreichen. Der Logical Effort ist nur als Richtwert zu verstehen, da das Delay von mehreren Faktoren abhängig ist [YS10].

Das Delay d wird normalisiert dargestellt und alle Effekte des Fertigungsverfahrens werden durch die Variable τ repräsentiert. τ ist dabei als Delay eines Inverters welchem ein identischer Inverter ohne parasitäres Delay nachgeschaltet ist, definiert. Für ein Fertigungsverfahren mit $0,6\mu m$ ($0,25\mu m$, $0,045\mu m$) Strukturbreite ist τ mit 50ps (20ps, 5ps) definiert. Das absolute Delay ist das Produkt aus normalisiertem Delay und τ .

$$d_{abs} = d * \tau \quad (2.3)$$

Das normalisierte Delay eines Gatters wird als Summe des parasitären Delay p sowie dem Stage Effort f dargestellt. Bei dem parasitären Delay handelt es sich um das Delay welches auftritt, wenn am Ausgang des Gatters keine Last hängt, der Stage Effort hingegen ist abhängig von der Last am Gatter.

$$d = f + p \quad (2.4)$$

Der Stage Effort ist das Produkt aus zwei Komponenten, dem Logical Effort g sowie dem Electrical Effort h . Bei dem Logical Effort handelt es sich um das Verhältnis zwischen Eingangskapazität eines Gatters und der eines Inverters welcher den selben Ausgangsstrom zur Verfügung stellen kann. Der Electrical Effort beschreibt das Verhältnis zwischen Eingangskapazität eines Gatters und der Last des zu betrachtenden Gatters.

$$f = g * h \quad (2.5)$$

Unter Kombination der oben aufgeführten Gleichungen lässt sich das normalisierte Delay eines einfachen Logik Gatters darstellen:

$$d = g * h + p \quad (2.6)$$

Für die in dieser Arbeit verwendeten Gatter sind die berechneten Werte für den Logical Effort in Tabelle 2.3a und für das Parasitic Delay in Tabelle 2.3b zu finden. Zusammenfassend finden sich in Tabelle 2.4 alle Formeln zur Berechnung des Logical Effort und können auch in [SSH99b, Seite 71,77] und [Chu03, S.18] nachgeschlagen werden.

Gatter	Eingänge			
	1	2	3	4
Inverter				
NAND		$\frac{4}{3}$	$\frac{5}{3}$	$\frac{6}{3}$
NOR		$\frac{5}{3}$	$\frac{7}{3}$	$\frac{9}{3}$
XOR,XNOR		4	12	16
C-Element		2	3	4
D-Latch	4			
DETFE	8			

(a) Logical Effort(g)

Gatter	Eingänge			
	1	2	3	4
Inverter	1			
NAND		2	3	4
NOR		2	3	4
XOR,XNOR		4	6	8
C-Element		2	3	4
D-Latch	2			
DETFE	4			

(b) Parasitic Delay(p)

Tabelle 2.3: Übersicht Basiswerte

Begriff	Gatter	Pfad
Logical effort	g (Tabelle 2.3a)	$G = \prod g_i$
Electrical effort	$h = \frac{C_{out}}{C_{in}}$	$H = \frac{C_{out-path}}{C_{in-path}}$
Stage Effort	$f = g * h$	$F = G * B * H = \prod f_i$
Number of stages	1	N
Parasitic delay	p (Tabelle 2.3b)	$P = \sum p_i$
Delay	$d = f + p$	$D = \sum d_i = D_F + P$

Tabelle 2.4: Terme und Formeln des Logical Effort

2.10 Faults

Einige oft genutzte Begriffe in Bezug auf Fehler Toleranz und - Erkennung sollen hier erklärt werden. Dabei werden die englischsprachigen Begriffe verwendet, da es bei Verwendung der

deutschen Begriffe zur Verwechslung kommen könnte. Die Terme Failure, Error und Fault sind in [ALRTCS01] definiert und genauestens beschrieben.

- **Failure** beschreibt das Fehlverhalten eines Systems, wenn dieses nicht mehr in der spezifizierten Art und Weise funktioniert.
- **Error** beschreibt eine Ausnahmesituation im System, welche zu einem Fehlverhalten desselben führt. Eine solche Ausnahmesituation kann durch in einer Abweichung der korrekten Werte zu den berechneten festgestellt werden.
- **Fault** beschreibt die Ursache einer Ausnahmesituation.

Abbildung 2.22 beschreibt die Abhängigkeiten zwischen Faults Errors und Failures.

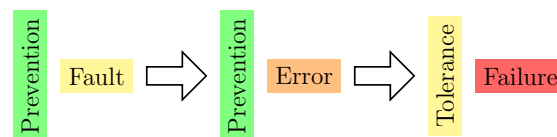


Abbildung 2.22: Abhängigkeit zwischen Fault, Error und Failure

”A fault will generate an error provided the fault is activated and not dormant. The produced error will lead to a failure provided the error changes the intended behavior of the system. Basically, an error must propagate to the system boundaries to trigger a failure. This causal chain is continued on the next higher level. A failure in a subsystem may be regarded as fault on the higher system level.” Zitat [ALRTCS01].

2.10.1 Single Stuck-At Fault (SSAF)

Das Single Stuck-At Fault (SSAF) Modell [GNR61, Joh89] ist eines der populärsten Fault Modelle. SSAF trifft mehrere Annahmen, welche die Verwendung ohne Wissen über benutzte Technologien auf dem logischen Level der Gatter Ebene ermöglichen. Ein Fault wird dabei nur an einem Ein- oder Ausgang eines Gatters zu einem bestimmten Zeitpunkt auftreten. Dieser Fault ist permanent und kann für das Stuck-At 1 (s@0) bzw. Stuck-At 0 (s@1) mit einem Kurzschluss nach V_{DD} bzw. GND beschrieben werden. Weiters wird angenommen dass ein Test der jeden single Fault erkennt, auch mehrfache Faults erkennen kann. Das SSAF Model deckt dabei viele mögliche Herstellungsfehler wie ”missing features”, ”source-drain shorts”, ”diffusion contaminants”, und ”metallization shorts” ab. Dabei wird typisch eine Fehlerabdeckrate von über 70 % erreicht [LM04].

Qualitative Analyse und Erstellung eines Modells

Auf dem Weg zu einer umfassenden Analyse wird in diesem Kapitel ein erster Schritt unternommen. In einer qualitativen Analyse wird der Frage nachgegangen, welche Auswirkungen ein Stuck-At Fault haben kann und von welchen Einflussfaktoren diese abhängig sind. Es werden Kategorien für auftretende Effekte definiert. Diese Kategorien beschreiben ob eine Pipeline blockiert ist, ohne Fehler arbeitet oder inkorrekte Ergebnisse liefert.

3.1 Übersicht

3.1.1 Einschränkung

Um die Arbeit zu beschränken werden einige Annahmen getroffen.

Diese Arbeit beschränkt sich auf NCL, LEDR sowie, das am Institut für Technische Informatik der TU Wien entwickelte, CAL. Durch die Kompatibilität zwischen den CD von LEDR und CAL würde es ausreichen einen CD zu analysieren, dennoch soll die Gelegenheit genutzt werden beide zu vergleichen.

In der Betrachtung beschränken wir uns auf SSAF. Diese ermöglichen es verschiedene Faults, wie Transition-, Transistor- oder Path-Delay-Faults [Sch09, MBAB99] ohne Aufwand abzudecken. Der Grund des Verzichts auf die Betrachtung der Multiple Stuck-At Fault (MSAF) liegt darin, dass sich diese durch SSAF abbilden lassen [Vaa06, LOKS06].

Es ist möglich die Busbreite ohne Verlust der Generalität auf zwei Bit zu beschränken (siehe Theorem A.0.2); dieses Faktum wird dafür genutzt den Aufwand der Analyse zu verringern. Aus dem genannten Theorem geht die Einteilung der Signale eines Bus in das letzte D_{letzte} , in der aktuellen Phase stabil werdende, und alle anderen D_{andere} hervor.

Die diskrete Betrachtung der Schaltung wird durch die Annahme ermöglicht, dass zu einem Zeitpunkt t immer nur eine Flanke auftreten wird [DFH⁺05].

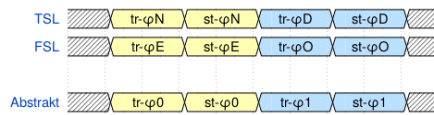
Um aus der qualitativen Aussage quantitative Rückschlüsse ziehen zu können, ist es nötig ein gewichtetes D^N zu erhalten, d.h. die Ergebnisse für D_{andere} und D_{letzte} im entsprechende Verhältnis zu ihrer Gesamtanzahl N zu setzen.

$$D_{andere}^N = D_{andere} * (N - 1) \quad (3.1)$$

$$D_{letzte}^N = D_{letzte} * frac{1}{N} \quad (3.2)$$

Für die Implementierung der Dual-Rail-Logik gibt es mehrere verschiedene Verfahren, z.B. STLM, Delay Insensitive Minterm Synthesis (DIMS), TG oder das Programmable Logic Array (PLA). Auf die detaillierte Betrachtung wird verzichtet, da die Wahl des Verfahrens das Verhältnis verändert, nicht jedoch die Art der auftretenden Faults. Sollte in einer erweiterten Analyse die Funktionslogik miteinbezogen werden, so sei auf die entsprechende Literatur [PA91] verwiesen.

Die in Kapitel 2.6 vorgestellten Handshake-Verfahren bestehen jeweils aus zwei Phasen. Diese Phasen setzen sich wiederum aus zwei Subphasen, tr- φ und st- φ , zusammen. Während bei NCL auf jede *Data-Phase* eine *Null-Phase* folgt, ist letztere bei FSL nicht nötig. Es folgt eine *Data-Phase* auf eine *Data-Phase*. Da für die Analyse die Unterscheidung zwischen *Data-* und *Null-Phase* nicht nötig ist, werden wir die Phasen für NCL und FSL unter gleicher Bezeichnung führen. Durch die in Abbildung 3.1 ersichtliche Konversion wird der Vergleich zwischen NCL und FSL möglich, und somit wird auch die Konsistenz mit anderen Arbeiten gewahrt [PN95, DWD91, FB96a, DFH⁺05]. φ_0 bezeichnet die Phase in welcher *FALSE* am Ausgang des CD anliegt, φ_1 diejenige mit *TRUE* am Ausgang.



(a) Waveform

NCL	FSL	CD	Abstrakt
φ_N	φ_E	0	φ_0
φ_D	φ_O	1	φ_1

(b) Äquivalenzen von Phasen

Abbildung 3.1: Äquivalenz der Phasenbezeichnung

Die Vielzahl existierender Pipelinemplates [Smi02, OB02] macht es nötig ein allgemein gültiges abstraktes Template zu verwenden. Basierend auf [BOF10, Kapitel 11.2] wird das Template verwendet wie es die Abbildung 3.2 zeigt. Zur Datenflusskontrolle verwendet es einen Eingangs- (LCD), Ausgangs-CD (RCD), sowie das Signal (RACK) des RCD der nachfolgenden Pipelinestufe (Sink). Die in der Kontrolllogik (CTRL) verarbeiteten Signale steuern das Register (REG). Sobald durch RACK die Bereitschaft der nächsten Stufe neue Daten zu erhalten signalisiert wird, initiiert LCD die Übernahme neuer Daten aus der vorhergehenden Pipelinestufe (Source). RCD stellt sicher, dass in allen Registern einer Stufe die Daten derselben Phase gespeichert werden. Auch wenn das gewählte Template seine Allgemeingültigkeit selbst nach

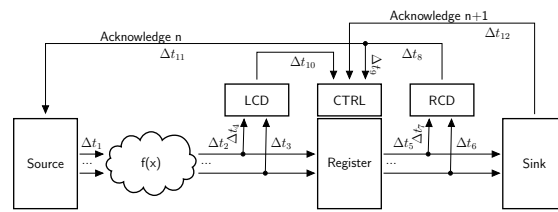


Abbildung 3.2: Abstraktes nichtlineares Pipeline Template

Entfernung von LCD bewahren [BOF10, Kapitel 11.1] würde, wäre aber nur noch für lineare Pipelines geeignet.

Der Aufbau eines CDs ist in dem Kapitel 2.7 beschrieben.

Der CTRL-Block ist für NCL/LEDR und für CAL verschieden. Für NCL/LEDR wird er aus einem einzelnen C-Element mit drei Eingängen gebaut (siehe Abbildung 3.3a), wobei der Eingang für das Signal von LCD negiert ist, sodass folgende Hysteresenfunktion gilt.

$$\uparrow E_n = \neg\varphi_{LCD} \wedge \varphi_{RCD} \wedge \varphi_{RACK} \quad (3.3)$$

$$\downarrow E_n = \varphi_{LCD} \wedge \neg\varphi_{RCD} \wedge \neg\varphi_{RACK} \quad (3.4)$$

Da für CAL mit D-Latches zustandsorientierte Speicherelemente verwendet werden, muss dementsprechend die Kontrollfunktion angepasst werden. Eine Möglichkeit dafür bietet der Aufbau aus Abbildung 3.3b, welcher aus zwei UND-Gattern mit drei Eingängen und einem ODER-Gatter besteht, welche folgende kombinatorische Funktion erfüllen:

$$E_n = (\neg\varphi_{LCD} \wedge \varphi_{RCD} \wedge \varphi_{RACK}) \vee (\varphi_{LCD} \wedge \neg\varphi_{RCD} \wedge \neg\varphi_{RACK}) \quad (3.5)$$

$$(3.6)$$

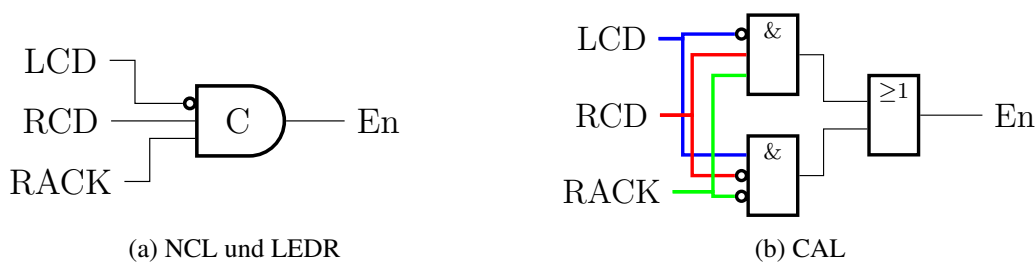


Abbildung 3.3: Kontrolllogik

Der REG-Block enthält für jedes Signal des Datenbus ein Speicherelement. Bei NCL kommt ein C-Element zum Einsatz, bei LEDR ein DET-FF und bei CAL D-Latches. Alle Enable-Signale der Speicherelemente werden durch den CTRL-Block gesteuert.

Die Signale auf verschiedenen Leitungen einer Rail breiten sich unterschiedlich schnell aus. Aufgrund der Synchronisation der Signale an jedem Dual-Rail-Gatter, wird die Differenz des Delays der Signale auf einer Rail nicht separat in das Modell einbezogen, sondern ist in die Laufzeit der Dual-Rail Signale, D_{andere} und D_{letzte} , bereits mit einberechnet.

Das von Fant [FB96a] vorgestellte Verfahren die Pipeline mittels Treshold Gates zu erstellen wird hier nicht berücksichtigt, da dieses laut Smith [YS10] auf dem Synchronisationspfad keine abweichende Funktion und auch Delay aufweist.

Wie in Theorem A.0.1 gezeigt, ist es ausreichend, den nächsten zu erwartenden Zustand nach einer aktiven Flanke zu betrachten. Dieser Zustand gibt Aufschluss darüber, ob und in welcher Weise das Auftreten eines Faults die Funktion beeinträchtigt.

3.1.2 Klassifizierung der Effekte

Aus der Literatur [LM04, PN95, MRL07] geht hervor, dass es eine begrenzte Anzahl an Effekten durch Stuck-At Faults gibt. Diese werden in Deadlock, Late Detection und Premature-Fire unterteilt. Ein Premature-Fire hat weitere Effekte zur Folge; darunter den Transparent Timing Fault, Illegal Symbol Generation, Token Generation, Token Consumption und Isochronic Delay Fault. Bedingt durch die, den verschiedenen Arten der Dual-Rail Kodierungen zugrunde liegende Robustheit, nehmen Deadlock und Late Detection einen Großteil der auftretenden Effekte ein.

Ein **Deadlock** beschreibt eine Verklemmung, bei der es durch das Auftreten eines Faults zum Stillstand der Schaltung kommt. Nach Auftreten eines Stuck-At Faults handelt es sich beim DL um einen wünschenswerten Effekt, da die Datenkonsistenz nicht beeinträchtigt wird. Die Pipeline arbeitet bis zum Auftreten des Fehlers korrekt und bleibt dann stehen. Eine Schaltung nimmt, nach dem Beseitigen eines zu einem Deadlock führenden Faults, ihre ursprüngliche Funktion, ohne etwaige Datenkorruption, wieder auf. Durch ein Timeout ist das Erkennen eines DL auf simple Weise möglich.

Ein einfaches Beispiel (siehe Abbildung 3.5) dafür ist eine lineare Pipeline, bei der es nach Erreichen der Phase φ_O , in einer Stufe n zu einem Stuck-At 1 am Ausgang des CDs kommt. Dadurch dass keine Änderung am Ausgang mehr möglich ist, kann keine nachfolgender Phasenwechsel signalisiert werden. Es wird weder eine vorhergehende Pipelinestufe neue Daten auf dem Bus senden, noch wird eine nachfolgende Daten erhalten.

Late Detection definiert die Menge der Faults, welche die Schaltung in einen Zustand bringen, in welchem nur noch eine eingeschränkte Funktion möglich ist. Das Schaltnetz funktioniert rein für eine Untermenge M_u der Gesamtmenge M der möglichen Eingangszustände. Solange an den Eingängen diese Untermenge auftritt, funktioniert das Schaltnetz weiterhin korrekt und innerhalb der Parameter. Sollte an den Eingängen ein Zustand auftreten, welcher der Menge $M_o = (M \setminus M_u)$ entspricht, kommt es zu einem Deadlock. Durch den SSAF ist das Codewort nicht mehr abbildbar und somit keine Phasenwechsel signalisierbar.

Ein Beispiel dafür (siehe Abbildung 3.5) ist ein NCL Schaltnetz welches das an den Eingängen anliegende Codewort ($e_1 = 1, e_2 = 0$) hat. Tritt ein Stuck-At 1 an e_1 auf, so werden

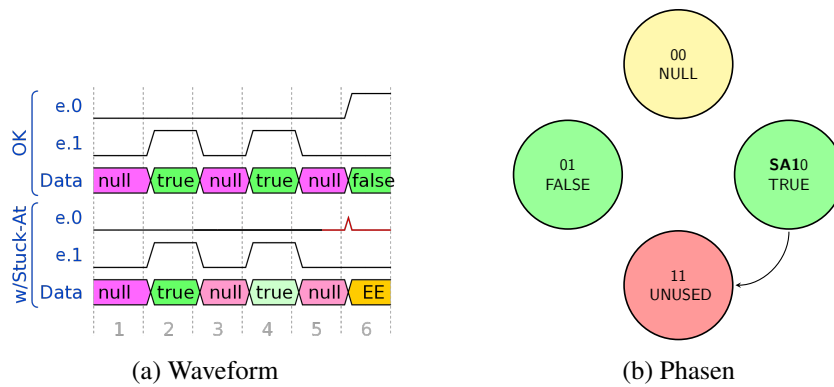


Abbildung 3.4: Deadlock durch SA1 nach einem TRUE am NCL Bus

dadurch die Mengen $M = \{00, 01, 10, 11\}$, $M_u = \{10, 11\}$ und $M_o = \{01, 00\}$ definiert. Solange die nun folgenden Codewörter in der Menge M_u abgebildet sind, funktioniert die Schaltung innerhalb der Parameter. Erreicht die Schaltung ein Codewort aus der Menge M_o , so wird ein Phasenwechsel nicht mehr registriert. Es kommt zum Deadlock, bis dorthin funktioniert die Schaltung allerdings zuverlässig.

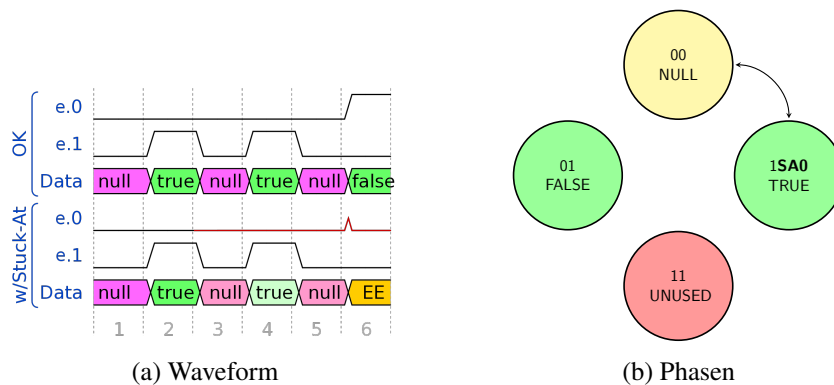


Abbildung 3.5: Late Detection durch SA0 nach einem NULL am NCL Bus

Ein **Premature Fire** (in der Literatur auch *Synchronisation Failure* genannt [LM04]) beschreibt die vorzeitige Änderung des Signalzustandes am Ausgang einer Schaltung. Abhängig von Timing und betroffenem Signal, führt ein PF zu verschiedenen Effekten.

Ist die nachfolgende Pipelinestufe (Sink) nicht für eine Datenübernahme bereit und liegt das nächste Datenwort am Eingang der Register an, so wird die Pipeline blockiert (token-limitiert/blocking). Nach Eintreffen der nächsten Datenwelle sind die Steuersignale als $LCD = \varphi$, $RCD = \neg\varphi$ und des $RACK = \varphi$ definiert. Kommt es, bei einer Token-limitiert-Pipeline, durch ein SSAF zum Signalisieren von $RACK = \neg\varphi$, so tritt ein PF auf. Die Register übernehmen das vorhandene Datenwort. Dieses wird zur Sink propagiert, wo die Daten von der Funktionslogik $f(x)$ verarbeitet werden. Abhängig vom Timing wird es in der Sink zu unterschiedlichen

Verhalten kommen.

- a) Hat Sink bei Eintreffen der neuen Datenwelle das vorherige Datenwort übernommen, kommt es zu einer Late Detection. Durch glückliches Timingverhalten wurde die Validität und Konsistenz der Datenwörter erhalten, eine erneute Datenübernahme aber wird durch den SSAF verhindert. Man spricht hier von einem **Transparent-Timing-Fault**.
- b) Hat Sink bei Eintreffen der neuen Datenwelle das vorherige Datenwort nicht übernommen, kommt es zu einer **Token-Consumption**. Dabei wird ein Datenwort von der nachfolgenden Datenwelle überschrieben. Es wird die Konsistenz der Datenwörter verletzt und die CDs verhindern eine weitere Arbeit der Pipeline. Es tritt ein Deadlock auf, allerdings muss nach Beseitigen des SSAF die Pipeline neu initialisiert werden.

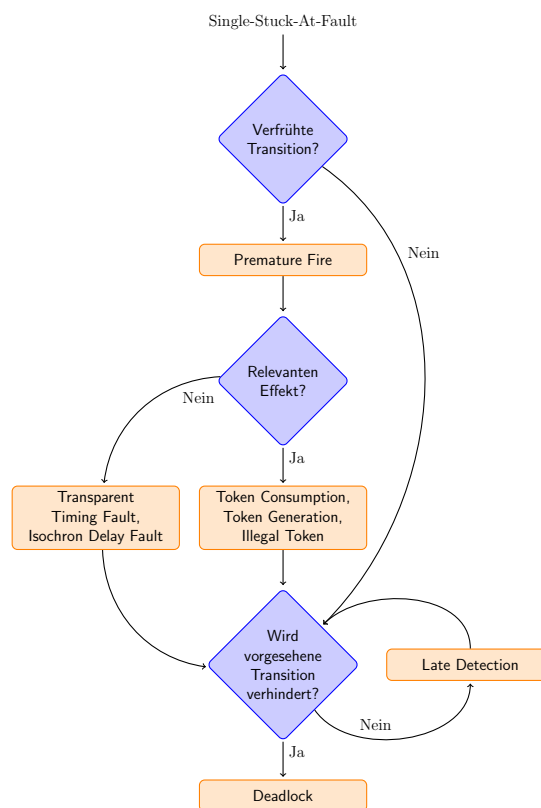


Abbildung 3.6: Zusammenhänge zwischen den Effekten

Ist Sink für eine Datenübertragung bereit und liegt am Eingang der Register kein Datenwort an, so verhungert die Pipeline (bubble-limitiert/starvating). Vor Eintreffen der nächsten Datenwelle sind die Steuersignale als $LCD = \varphi$, $RCD = \varphi$ und $RACK = \varphi$ definiert. Kommt es bei einer bubble-limitiert-Pipeline durch einen SSAF zur Signalisation von $LACK = \neg\varphi$, so tritt ein PF auf. Die Register übernehmen an deren Eingängen den aktuellen Signalfeldzustand.

Abhängig von den übernommenen Signalen sowie dem Timing sind mehrere Unterscheidungen nötig.

- a) Ist das PF innerhalb eines kurzen Zeitfensters $\Delta t \leq T_{CTRL}$ aufgetreten, so wird das korrekte Datenwort aus Source übernommen. Δt bezeichnet hier die Differenz zwischen Auftritt des SSAF und Eintreffen des nächsten Datenwortes. Es handelt sich um ein Late Detection durch PF. Man spricht hier von einem **Isochron-Delay-Fault**.
- b) Wird das PF durch ein SA auf einer Datenleitung hervorgerufen und wird dabei ein Datenwort erzeugt, welches Validität und Konsistenz erfüllt, hingegen vom korrekten Codewort abweicht, so spricht man von einer **Token Generation**, dh. eine zusätzliche Datenwelle wurde generiert. Für Sink ist dieser Fall nicht erkennbar, da die erzeugte Datenwelle von den nachfolgenden Pipelineinstufen verarbeitet wird und am Ende der Pipeline ein falsches Ergebnis am Ausgang erzeugt wird. Dieses Verhalten kann allerdings erst verzögert erkannt werden, da ein nachfolgendes Codewort zu einer Verletzung der Konsistenz führen würde. Der genaue Zeitpunkt hängt von dem Codewort ab; die Pipeline muss nach Beseitigen des SSAF neu initialisiert werden.
- c) Wird das PF durch ein SA auf einer Datenleitung hervorgerufen, und wird dabei ein die Validität verletzendes Datenwort in das Register übernommen, so spricht man von einer **Illegal-Token-Generation**. In dem von uns gewählten Pipeline-Template führt dies zu einem Deadlock. Andere asynchrone Schaltungen werden das Datenwort aber verarbeiten [LM04] werden. Ein resultierendes valides Datenwort wird von der Pipeline weiter verarbeitet. In beiden Fällen muss die Pipeline nach Beseitigung des SSAF neu initialisiert werden.

Die Zusammenhänge zwischen den verschiedenen Effekten sind in Abbildung 3.6 skizziert. Sollte ein Premature Fire ein Token generieren oder auch konsumieren, so ist davon auszugehen, dass der Inhalt der Pipeline kompromittiert wurde. Daher ist eine Neuinitialisierung der Pipeline nach der Bereinigung von solchen Faults notwendig.

3.2 Analyse der Blöcke

Jeder Teil der Schaltung muss separat analysiert werden. Jene Analyse soll, um Duplizität in der Arbeit zu vermeiden, lediglich für den NCL CD detailliert dargestellt werden. Für alle weiteren Blöcke werden nur die Ergebnisse präsentiert und die für die Arbeitsschritte nötigen Informationen aufgelistet.

3.2.1 Verfahren

Das Ziel der qualitativen Analyse ist die Bestimmung der Auswirkungen eines Stuck-At Faults auf eine Schaltung. Händisch wird eine Liste aus allen Kombinationen der Phasen, mit Ein-/Ausgängen und möglichen Stuck-At Faults, erstellt. Der logischen Spezifikation entsprechend

wird, eine der in Abschnitt 3.1.2 gelisteten Folgen, bestimmt. Wenngleich eine formale Beweisführung ermöglichende Methoden existieren, ist die beschriebene Methode für diese Arbeit hinreichend. Nichtsdestotrotz soll hier die formale Beweisführung kurz vorgestellt werden. Um die Berechnung für das Modell durchzuführen wird „Octave“¹ in Version 3.8.1 vom 7. März 2014 verwendet, dabei handelt es sich um eine unter GNU General Public License Version 3 lizenzierte Software, zur numerischen Lösung mathematischer Probleme. Der Quellcode des Modells befindet sich in Anhang B.

3.2.1.1 Time Petri-Net basierendes Stuck-At Fault Modell

Wie wir in Kapitel 2.3 erfahren haben, lässt sich jedes logische Gatter als Petri-Netz darstellen. In [PT05] wird der auf „Time-Petri-Netz“basierende Algorithmus 1, zum Konvertieren von fehlerfreien Schaltungen vorgestellt, sodass sich Stuck-At Faults darstellen lassen. Im Algorithmus entspricht P jenem Place der den Stuck-At Zustand darstellt. T bezeichnet die Stuck-At- und U die von P ausgehenden Transitions.

Eingabe : Die dem Stuck-At entsprechende Transition T

Ergebnis : TPN mit Stuck-At

Erstelle Bogen von T nach P ;

$T[EFT, LFT] = [0, 0]$;

für jedes U tue

 | $[EFT, LFT] = [\infty, \infty]$;

Ende

Algorithmus 1 : TPN Transformation

Bei einem Time Petri Net (TPN) kann durch Analyse des Graphen festgestellt werden, ob es zu einem DL, LD oder PF kommt. Um festzustellen ob ein Punkt des Petri-Netzes noch erreichbar ist, gibt es bereits ausgereifte Algorithmen [WKB04, BK05]. Als großer Nachteil wirkt sich die, bei großen Schaltungen stark zunehmende, Komplexität aus.

In Abbildung 3.7a ist die Repräsentation eines C-Elementes durch ein TPN zu sehen. Das C-Element hat zwei Eingänge A und B sowie einen Ausgang C, dessen Zustand, zum Zeitpunkt des Auftretens des Stuck-At, ist mit $A=1$, $B=1$, $C=1$ gegeben. Tritt ein Stuck-At 1 an Eingang A auf, so handelt es sich bei dem Punkt des Petri-Netzes welcher $A=1$ repräsentiert um den Stuck-At Punkt. Auf diesen Punkt wird von allen verbundenen Transitions, ein Bogen (durchgehend Rot hervorgehoben) zurück auf diesen gesetzt. Das Intervall der Transition $+A?$ wird auf $[EFT, LFT] = [0, 0]$ gesetzt. Die durch das SSAF nicht mehr erreichbare Transition $-A?$ erhält das Intervall $[EFT, LFT] = [\infty, \infty]$ zugewiesen (die betroffenen Bögen sind strichliert markiert). Das Resultat des Algorithmus ist ein TPN, für welches zwar noch die Transitions $\uparrow B?$ und $\downarrow B?$ erreichbar sind, jedoch nicht $-A?$. Ausgehend vom Zustand $\uparrow C?$ am Ausgang ist es nun nicht mehr möglich die Transition $\downarrow C?$ zu erreichen, welche $\downarrow A?$ bedingt.

¹<https://www.gnu.org/software/octave/>

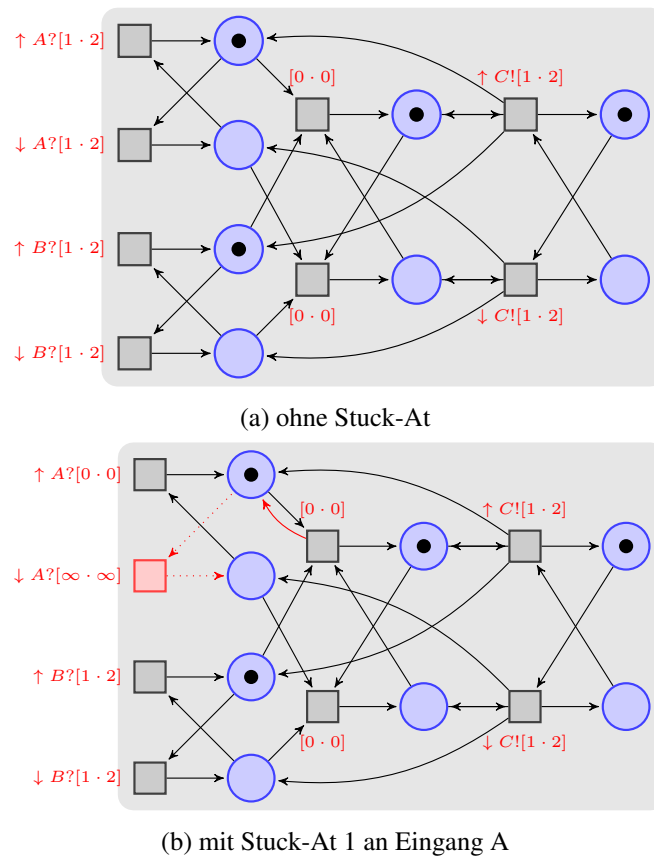


Abbildung 3.7: Anwendung des Algorithmus auf das TPN eines C Elements

3.2.1.2 Verfahren für die qualitative Analyse

Bei dem in dieser Arbeit angewandtem Verfahren, wird ähnlich dem in [PN95] vorgegangen. Der Zustandsraum $M = SA * PH * EA * ZS$ ergibt sich aus folgenden Dimensionen: Aus der Menge der möglichen Stuck-At Faults $SA = \{SA0, SA1\}$, den Phasenzuständen $PH = \{tr-\varphi_0, st-\varphi_0, tr-\varphi_1, st-\varphi_1\}$, den Zuständen, welche ein Signal annehmen kann, $ZS = \{Hi, Lo\}$ und den Ein- und Ausgängen der Gatter der Schaltung

$$EA = \left\{ \bigcup_{k=0}^{k=|E|} E_k \bigcup_{k=0}^{k=|A|} A_k \right\}$$

Während die Erklärung über die Notwendigkeit des Miteinbezugs von SA , EA und ZS trivial ist, bedarf es bei PH einer kurzen Erklärung. Aus der Definition von QDI geht hervor, dass keine Annahmen über das Delay von Gattern sowie von Signalen gemacht werden dürfen, daher erfolgt die Einteilung eines Zyklus in Stable und Transition Phasen. Den jeweiligen Phasen ist ein Set an Signalen zugewiesen, welche auftreten dürfen. Erst durch Betrachtung dieser Signalzustände ist es möglich eine Betrachtung der Wirkung eines Stuck-At Faults zu machen.

Für einen NCL-CD, wie in Abbildung 3.8, setzt sich die Menge EA wie folgt zusammen:

$$EA = \left\{ \bigcup_{k=0}^1 \{OR_k.Ein_2, OR_k.Ein_1, OR_k.Aus\}, C_0.Ein_0, C_0.Ein_1, C_0.Aus \right\}$$

Das Verfahren des Fault-Collapsing [Vaa06, LOKS06] wird dazu angewandt den Aufwand zu minimieren. Dadurch können die zu analysierenden Kombinationen stark eingeschränkt werden. So kann zum Beispiel ein SSAF an $OR_0.Aus$ durch ein anderes an $C_0.Ein_1$ abgebildet werden.

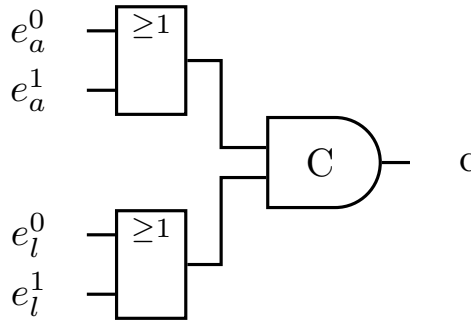


Abbildung 3.8: NCL CD, $e_a = D_{andere}$, $e_l = D_{letzte}$

Um eine vollständige Betrachtung anzufertigen werden alle Signale eines Zyklus aufgelistet. Das Format zur vollständigen Bezeichnung eines Signales lautet:

$$\langle Gatter \rangle . \langle Pinname \rangle . [Zeitpunkt] . \langle erwarteter Zustand \rangle . \quad (3.7)$$

Die eckigen Klammern ($[,]$) entsprechen dabei optionalen Komponenten der Bezeichnung und die Angewinkelten (\langle, \rangle) den obligatorischen. Der Zeitpunkt ist optional, da er für bestimmte E/A nicht relevant ist. Dies trifft z.B. auf den Ausgang des C-Elementes, aus dem Beispiel im vorhergehenden Abschnitt, zu. Durch die Kombination der Phase mit dem nächsten erwarteten Zustand ist die Menge möglicher vorhergehender Zustände implizit definiert. Dies ist deshalb wichtig, da so der Zustand einer Rail aus zwei verschiedenen vorhergehenden Zuständen erreicht werden kann.

Das Resultat ist eine Tabelle, in welcher den jeweiligen Zuständen ein Effekt zugewiesen wird. Hervorzuheben ist schließlich auch, dass, ausgehend von den Ergebnissen dieses Verfahrens, quantitative Rückschlüsse gezogen werden können. Dazu werden die Einflussfaktoren dementsprechend gewichtet. Diese Methode wird in Kapitel 4.4 angewandt um die Ergebnisse der Simulation und der qualitativen Analyse zu vergleichen.

Die aus den Tabellen hervorgehenden Werte entsprechen einer Pipeline-Stufe mit einem zwei-Bit-Dual-Rail-Bus und einer Gleichverteilung der bestimmenden Effekte über alle Blöcke. Um das konkrete Verhalten unter verschiedenen Bedingungen zu analysieren wird nachfolgend ein entsprechendes Modell aufgestellt.

3.2.1.3 Modell für die quantitative Analyse

Um die Komplexität des Modells zu verringern, wird, wie in Kapitel 3.1.1 beschrieben, abstrahiert. Diese Sichtweise kann als Blackbox verstanden werden, dadurch kann die Anzahl der Parameter für das Modell beschränkt werden. Da nur Fälle betrachtet werden in denen ein Stuck-At auftritt, besteht ein Parameter \vec{SA} aus dem Verhältnis zwischen Stuck-At 0 und Stuck-At 1. Der Vektor \vec{SA} besteht aus zwei Komponenten, aus der Wahrscheinlichkeit für das Auftreten eines Stuck-At 0 (sa_0) sowie deren Gegenwahrscheinlichkeit ($1 - sa_0$).

$$\vec{SA} = \begin{pmatrix} sa_0 \\ 1 - sa_0 \end{pmatrix} \quad (3.8)$$

Als weiterer Parameter kommt das Verhältnis zwischen den alternierenden Phasen eines Zyklus, sowie deren Subphasen, hinzu. Das Verhältnis wird mit dem jeweiligen prozentualen Anteil am Gesamtzyklus angegeben. Dies bedeutet, dass der Anteil für eine Phase 0 mit $\varphi_0 \in \{0 \dots 1\}$ angegeben wird. Der verbleibende Anteil für Phase 1 lässt sich somit durch $\varphi_1 = 1 - \varphi_0$ berechnen. Für die Subphasen gilt dasselbe, allerdings bezogen auf den prozentualen Anteil an der Phase φ_0 oder φ_1 . Der Anteil der Transition Phasen kann mit $f_{tr}^\varphi \in \{0 \dots 1\}$ angegeben werden, wobei sich die stabilen Phasen durch $f_{st}^\varphi = 1 - f_{tr}^\varphi$ ergeben. Der Index f^φ entspricht der zu beschreibenden Phase. Die Parameter der Phasen und Subphasen werden als Matrix PH dargestellt, zur besseren Lesbarkeit wird hier die Transponierte PH^T verwendet.

$$PH^T = \begin{pmatrix} \varphi_0 \cdot f_{tr}^0 \\ \varphi_0 \cdot (1 - f_{tr}^0) \\ (1 - \varphi_0) \cdot f_{tr}^1 \\ (1 - \varphi_0) \cdot (1 - f_{tr}^1) \end{pmatrix} \quad (3.9)$$

Für die Berechnung der Wahrscheinlichkeit des Auftretens eines bestimmten Stuck-At in einer bestimmten Phase, wird der Vektor \vec{SA} mit der Matrix PH multipliziert. Die daraus resultierende Matrix wird als PS bezeichnet.

$$PS = \vec{SA} \cdot PH \quad (3.10)$$

Die bisher beschriebenen Parameter sind schaltungsunspezifisch. Um die schaltungsspezifischen Einflüsse in das Modell aufzunehmen, ist es nötig, diese durch eine geeignete Analyse (siehe Kapitel 3.2.1.2) festzuhalten. Die Ergebnisse der qualitativen Analyse werden in Form einer Matrix $V = v_{i=1, \dots, 4; j=1, 2}$ dargestellt. Die zu verwendenden Werte werden aus der jeweiligen Ergebnistabelle, sei es für Deadlock, Late Detection oder Premature Fire, in die Matrix $V_{[DL, LD, PF]}$ übernommen.

$$V = \begin{matrix} & \varphi_{0\text{-tr}} & \varphi_{0\text{-st}} & \varphi_{1\text{-tr}} & \varphi_{1\text{-st}} \\ \text{Stuck-At0} & v_{1,1} & v_{1,2} & v_{1,3} & v_{1,4} \\ \text{Stuck-At1} & v_{2,1} & v_{2,2} & v_{2,3} & v_{2,4} \end{matrix} \quad (3.11)$$

Durch die Elementweise-Multiplikation der Matrix V mit Matrix PS wird M erzeugt.

$$M = (PH \cdot SA) \circ V \quad (3.12)$$

Die Matrix M enthält die Einzelergebnisse für die zur Berechnung verwendeten Parameter. Um ein Gesamtbild zu erstellen, wird die Summe aller Elemente der Matrix erstellt. Womit die Gesamtwahrscheinlichkeit des Auftretens des durch V spezifizierten Effektes berechnet wird.

$$P(\text{Effekt}) = \sum_{i=1}^2 \sum_{j=1}^4 m_{i,j} \quad (3.13)$$

Da das Modell auf unterschiedliche Technologien angewendet wird, ist es nötig eine Methode zu entwickeln, welche es erlaubt diese zu vergleichen. Die aus der Statistik stammende Methode der Standardisierung erlaubt den Vergleich von Messwerten, welche in unterschiedlichen Verteilungen gewonnen wurden. Die Standardisierung wird hier durch die Funktion $norm(x, y)$ umgesetzt.

Um die Standardisierung auf das Modell anzuwenden, wird das Ergebnis der Funktion, entsprechend der Phasen, in einer Matrix abgebildet, welche anschließend mit dem ursprünglichen Modell multipliziert wird.

$$N^T = \begin{pmatrix} 1 - norm(x, y) \\ norm(x, y) \\ 1 - norm(x, y) \\ norm(x, y) \end{pmatrix} \quad (3.14)$$

Für das Modell ergibt sich eine angepasste Formel:

$$M_{Bsp} = (N \cdot PH \cdot SA) \circ V \quad (3.15)$$

Die Funktion erwartet als Übergabeparameter für den CD die Signallaufzeit der Signale D_{andere} als x und D_{letzte} als y (vergleiche Abbildung 3.9a). Als Rückgabewert erhält man den

prozentualen Anteil der Stable-Phase an der Gesamtphasenlänge; Bei $x = y$ ergibt sich wieder ein Wert von $norm(x, y) = 0,5$.

$$norm(x, y) = 1 - \frac{x}{y} \quad (3.16)$$

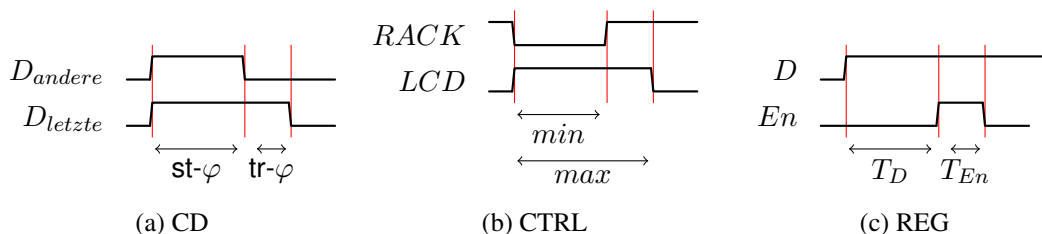


Abbildung 3.9: Timing Diagramme

Die Funktion erwartet als Übergabeparameter für den CTRL die Signallaufzeit der Signale $RACK$ als x und LCD als y (vergleiche Abbildung 3.9b). Da nicht definiert ist, welches der beiden Signale als erstes den Zustand wechselt, wird zur Bestimmung die Min/Max Funktionen verwendet. Als Rückgabewert erhält man den prozentualen Anteil der Stable-Phase an der Gesamtphasenlänge; Bei $x = y$ ergibt sich also ein Wert von $norm(x, y) = 0,5$.

$$norm(x, y) = \frac{min(x, y)}{max(x, y)} \quad (3.17)$$

Die Funktion erwartet als Übergabeparameter für den REG die Signallaufzeit der Signale En als x und D als y (vergleiche Abbildung 3.9c). Die Gesamtphasenlänge ergibt sich aus der Summe von $x + y$. Der Rückgabewert ist der prozentuale Anteil der Transition-Phase an der Gesamtphasenlänge; Bei $x = y$ ergibt sich also ein Wert von $norm(x, y) = 0,5$.

$$norm(x, y) = \frac{x}{x + y} \quad (3.18)$$

Der Signalpfad bestimmt die minimale Signallaufzeit, welche sich aus den Konstanten der Stuck-At Fault betroffenen Blöcke, sowie aus Source und Sink zusammensetzt. Weiters wird angenommen, dass die zu analysierende Pipeline-Stufe das selbe Template verwendet wie die Pipeline-Stufen Source und Sink. Die Signallaufzeit setzt sich aus dem minimalen Signalpfad der Blöcke Source und Sink, sowie der Gatteranzahl in $f(x)$ zusammen. Es wird angenommen, dass Source und Sink baugleich mit der zu analysierenden Pipeline-Stufe sind. Daher setzt sich der minimale Signalpfad innerhalb Source aus dem Delay eines CTRL und REG Block zusammen. Für Sink führt der minimale Signalpfad über LCD, CTRL, REG und RCD. Die Zusammensetzung aller Signallaufzeiten ist in Tabelle 3.1 festgehalten. Mit df wird der Faktor angegeben, um welchen das Datensignal D_{letzte} kleiner ist als D_{andere} .

Block	Bezeichnung	Zusammensetzung
Dual-Rail Gatter	T_G	NCL: als STLM gebaut LEDR/CAL: als PLA gebaut
Source	T_{min_Source} T_{Source}	$T_{CTRL} + T_{REG}$ $T_{min_Source} + nT_G$
Sink	T_{min_Sink} T_{Sink}	$T_{CD} + T_{CTRL} + T_{REG} + T_{CD}$ $T_{min_Sink} + mT_G$
LCD	D_{andere} D_{letzte}	$T_{CTRL} + T_{REG} + \max(T_{Sink}, T_{LCD} + T_{Source})$ $T_{andere} * df$
RCD	D_{andere} D_{letzte}	$T_{CTRL} + T_{REG} + \max(T_{Sink}, T_{LCD} + T_{Source} + T_{REG})$ $T_{andere} * df$
CTRL	LCD RCD	$T_{CTRL} + T_{REG} + \max(T_{Sink}, T_{LCD} + T_{Source} + T_{RCD})$ $T_{CTRL} + T_{REG} + \min(T_{Sink}, T_{LCD} + T_{Source} + T_{RCD})$
REG	En D	$T_{CTRL} + T_{REG} + \max(T_{Sink}, T_{LCD} + T_{Source} + T_{RCD})$ $T_{REG} + T_{Source} + T_{LCD}$

Tabelle 3.1: Timingvariablen

In den folgenden Abschnitten sind die beschriebenen Schritte für die einzelnen Komponenten CD, REG und CTRL für NCL, LEDR und CAL, durchgeführt worden. Um die Übersicht zu wahren und unnötige Wiederholungen zu vermeiden ist hier für die Komponente NCL-CD exemplarisch der komplette Rechenweg festgehalten worden. Die benötigten Informationen zur selbständigen Berechnung weiterer Komponenten sind in Anhang C.1 zu finden.

3.2.2 Completion Detektor

Entsprechend dem in Abschnitt 3.2.1.2 vorgestellten Verfahren, wird für die Ein- und Ausgänge des Blockes die Tabelle 3.2 angelegt. Diese enthält für alle E/A, der Schaltung in Abbildung 3.10b, die bei einem SSAF auftretenden Effekte, um die Kombination aus Signalen, Phasen und Stuck-At zu erstellen. Es werden Deadlocks mit „DL“, Late Detection mit „LD“ und Premature Fire mit „PF“ abgekürzt. Für jeden Fall ist ein formaler Beweis, wie in Kapitel 3.2.1.1 beschrieben, möglich, auf diesen wird aber, wie in Kapitel 3.2.1 begründet, verzichtet.

In Bezug auf Tabelle 3.2 ist festzuhalten, dass eine Gewichtung der Eingänge erfolgt. Im konkreten Fall besitzt der Zustand "OR.Ein.Lo" in den Phasen φ_0 -tr und φ_1 -st eine Gewichtung von zwei, was durch Zuweisung mehrerer Effekte ausgedrückt wird. Dies ist notwendig, da bei NCL in der NULL-Phase beide Rails den selben Signalpegel „00“ aufweisen, und somit der vorhergehende Zustand nicht mehr *eindeutig* impliziert wird. Durch die Gewichtung von E/A ist zudem die Abbildung einer abweichenden Wahrscheinlichkeit für ein SSAF im Modell möglich.

Um die, in Tabelle 3.2 festgehaltenen, Ergebnisse in die, vom Modell geforderte, Form zu bringen, werden sie in die Matrizen V_{DL} , V_{LD} und V_{PF} übertragen. Um die Wahrscheinlichkeit eines Eintrages zu berechnen muss dieser durch die Summe aller Werte dieser Phase geteilt

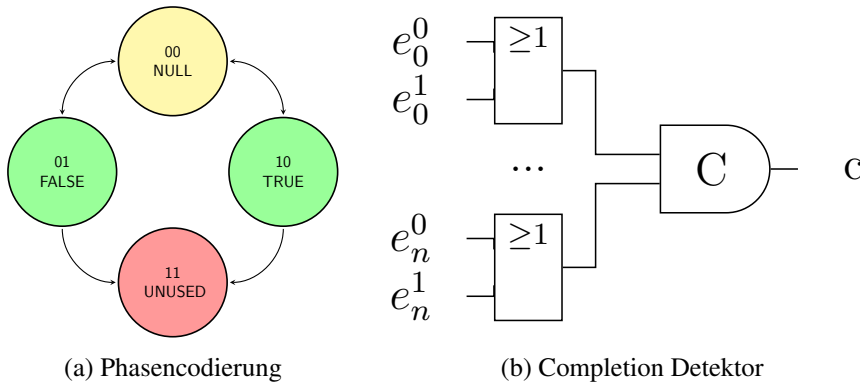


Abbildung 3.10: Three State Logic

Signal	Stuck-At 0				Stuck-At 1			
	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$
OR.Ein(andere).Lo		LD		LD, LD		LD		DL, DL
OR.Ein(andere).Hi		DL				LD		
OR.Ein(letztes).Lo	LD, LD		LD		DL, DL		LD	
OR.Ein(letztes).Hi			DL				LD	
OR.Aus(andere).Lo				LD				DL
OR.Aus(andere).Hi		DL				LD		
OR.Aus(letztes).Lo	PF				DL			
OR.Aus(letztes).Hi			DL				PF	
C.Ein(andere).Lo				LD				DL
C.Ein(andere).Hi		DL				PF		
C.Ein(letztes).Lo	PF				DL			
C.Ein(letztes).Hi			DL				PF	
C.Aus.Lo	PF	LD			DL	PF		
C.Aus.Hi			DL	PF			PF	LD
Summe[DL,LD,PF]	0,3,2	3,2,0	4,1,0	0,4,1	5,0,0	0,3,2	0,2,3	4,1,0

Tabelle 3.2: Qualitative Analyse eines NCL CD

werden. Für das Auftreten eines Stuck-At 0 in der Phase φ_0 -st ergibt sich aus obiger Tabelle die Anzahl von 3 Deadlocks auf eine Gesamtanzahl von 5 Effekten, also $v_{1,2} = \frac{3}{5}$.

$$V_{DL} = \begin{matrix} s@0 \\ s@1 \end{matrix} \begin{matrix} \varphi_0\text{-tr} & \varphi_0\text{-st} & \varphi_1\text{-tr} & \varphi_1\text{-st} \\ \begin{pmatrix} 0 & 3 & 4 & 0 \\ 5 & 5 & 5 & 5 \end{pmatrix} \end{matrix} \quad (3.19)$$

$$V_{LD} = \begin{matrix} s@0 \\ s@1 \end{matrix} \begin{matrix} \varphi_0\text{-tr} & \varphi_0\text{-st} & \varphi_1\text{-tr} & \varphi_1\text{-st} \\ \begin{pmatrix} 3 & 2 & 1 & 4 \\ 5 & 5 & 5 & 5 \end{pmatrix} \end{matrix} \quad (3.20)$$

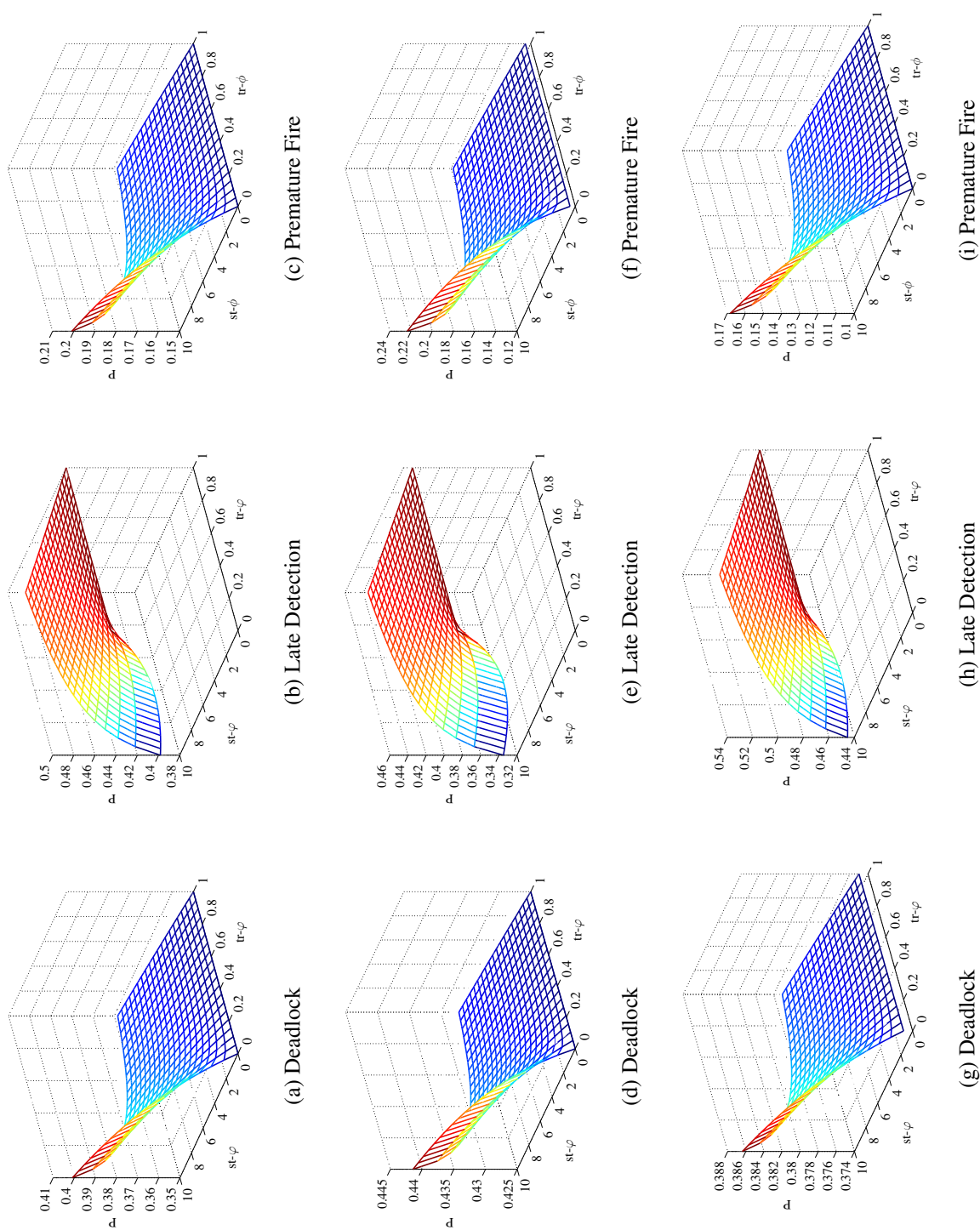
$$V_{PF} = \begin{matrix} s@0 \\ s@1 \end{matrix} \begin{pmatrix} \varphi_{0\text{-tr}} & \varphi_{0\text{-st}} & \varphi_{1\text{-tr}} & \varphi_{1\text{-st}} \\ \frac{2}{5} & \frac{0}{5} & \frac{0}{5} & \frac{1}{5} \\ \frac{0}{5} & \frac{2}{5} & \frac{3}{5} & \frac{0}{5} \end{pmatrix} \quad (3.21)$$

Für die Berechnung des Modells des CD für NCL wurden die, in Tabelle 3.3 aufgelisteten, Werte gewählt. Da für LEDR und CAL die Ergebnisse präsentiert werden, nicht aber der Rechenweg, sei hier auf die sich in den Tabellen 3.3, C.2 für LEDR und C.3 für CAL, befindlichen Werte verwiesen, durch welche eine Berechnung möglich wird.

Variable	Symbol	Wert		
		NCL	LEDR	CAL
Verhältnis $s@0$ zu $s@1$	sa_0	0,5		
Verhältnis zwischen Phase φ_0/φ_1	φ_0	0,5		
Internes Delay	T_{CD}	483ps	600ps	1216ps
Delay eines Dual-Rail-Gatter	T_G	500ps	1150ps	1016ps
Signallaufzeit D_0	T_x	$T_{CD} + nT_G$		
Signallaufzeit D_1	T_y	$T_x + mT_G$		
Zählvariable	m, n	$m, n \in \{0 \dots 10\}$		

Tabelle 3.3: Variablen

Abbildung 3.11: Resultate des Modells für CD NCL(a-c), LEDR(d-f), CAL(g-i)



In Abbildung 3.11 sind die Ergebnisse des Modells für NCL als 3D Graphen für DL (3.11a), LD (3.11b) und PF (3.11c) dargestellt. Die Graphen besitzen als x- und y-Achse die absolute Dauer der Transition- bzw. Stable-Phase. Um den späteren Vergleich zwischen verschiedenen Technologien zu ermöglichen, wurde die Dauer auf die Zeiteinheit eines Dual-Rail Gatter standardisiert und ist auf $T_G = 500ps$ normiert. Auf der z-Achse sieht man die Auftretswahrscheinlichkeit der Effekte, die Werte aller Effekte variieren in einem Bereich von 5-10%. Die Anzahl der DL korreliert mit jenen der PF und in umgekehrter Weise auch mit jenen der LD. Somit, steigt die Anzahl an DL und PF, sobald jene der LD sinkt, und umgekehrt. Für alle Effekte gilt, dass sich die Verlaufskurven nach einer Signallaufzeit von vier Gattern, bereits ihrem Extremwert sehr stark angenähert haben. Die Anzahl der unerwünschten Effekte steigt mit Länge des Zyklus, sobald ein Delay zwischen D_{andere} und D_{letzte} auftritt, sinkt sie hingegen schnell auf ihr Minimum. Je höher dieses Delay ist desto robuster ist der CD.

Für alle drei Technologien bietet sich dasselbe Bild: Sie erreichen ihr Maximum von 90% gutartiger Fälle bei einem hohen Delay zwischen D_{andere} und D_{letzte} . Die Minima und Maxima aller Technologien sind in Tabelle 3.4 zusammengefasst dargestellt. Während LEDR und CAL die Verfahren sind, welche die höchsten Schwankungen bei den PF aufweisen, ist gegenüber NCL eine um 5% geringere minimale Wahrscheinlichkeit für ein PF vorhanden.

[%]	NCL	LEDR	CAL
DL	35-45	42,5-45	37,5-39,5
LD	32-50	27,5-45	37,5-52,5
PF	15-24	12,5-27,5	10-23

Tabelle 3.4: Übersicht CD

Es ist insgesamt zu beobachten, dass die Anzahl der DL in gering variiert. Der CD für CAL erscheint am robustesten gegenüber SSAF.

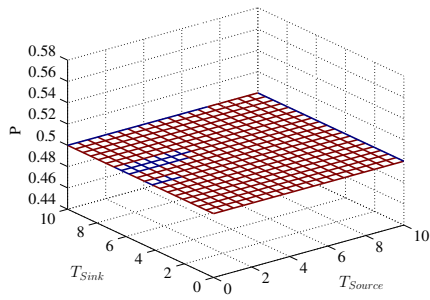
Die hier gewonnenen Erkenntnisse korrelieren mit jenen aus [PN95], in welchem die Analyse der STLM-Dual-Rail-Logik-Gatter, zum selben Ergebnis kommt.

Vorausblickend auf die Analyse einer kompletten Pipeline kann gesagt werden, dass durch die dort vorherrschende hohe Zyklusdauer bereits nahe einem Punkt $x = y \geq 5$ operiert wird.

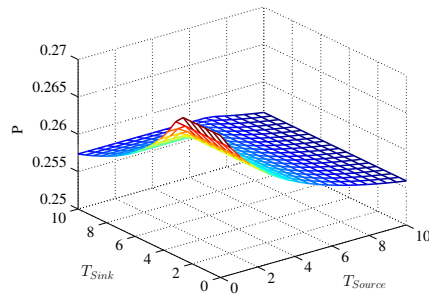
3.2.3 Kontrolllogik

Für die Berechnung des Modells des CTRL wurden die, in Tabelle 3.5 aufgelisteten, Werte gewählt. Da nur die Ergebnisse präsentiert werden, nicht aber der Rechenweg, sei hier auf die in den Tabellen 3.5, C.4 für NCL/LEDR und C.5 befindlichen Werte verwiesen, mit welchen eine Berechnung möglich wird.

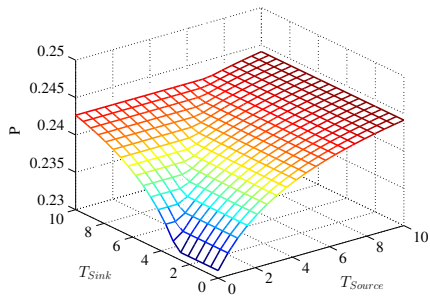
In Abbildung 3.12 sind die Ergebnisse des Modells für NCL/LEDR als 3D Graphen für DL (3.12a), LD (3.12b) und PF (3.12c) dargestellt. Die Graphen besitzen als x- und y-Achse die absolute Dauer der Transition- bzw. Stable-Phase. Um den späteren Vergleich zwischen verschiedenen Technologien zu ermöglichen, wurde die Dauer auf die Zeiteinheit eines Dual-Rail



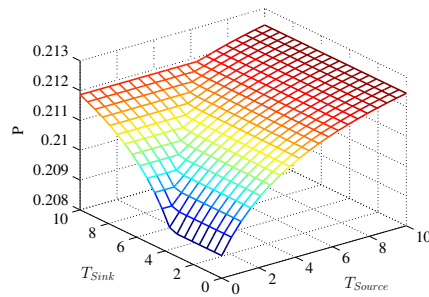
(a) Deadlock



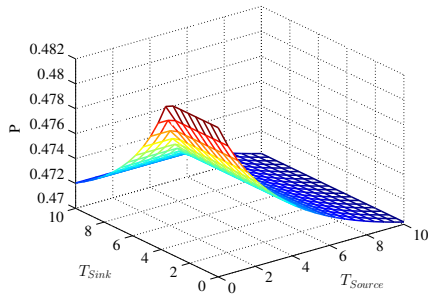
(b) Late Detection



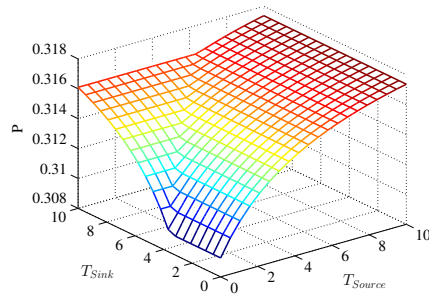
(c) Premature Fire



(d) Deadlock



(e) Late Detection



(f) Premature Fire

Abbildung 3.12: Resultate des Modells für CTRL NCL/LEDR(a-c), CAL(d-f)

Variable	Symbol	Wert	
		NCL	CAL
Verhältnis $s@0$ zu $s@1$	sa_0		0,5
Verhältnis zwischen Phase φ_0/φ_1	φ_0		0,5
Internes Delay	T_{CD}	300ps	617ps
Delay eines Dual-Rail-Gatter	T_G	500ps	1016ps
Signallaufzeit LCD	T_x	$T_{CTRL} + T_{Source} + nT_G$	
Signallaufzeit $RACK$	T_y	$T_{CTRL} + T_{Sink} + mT_G$	
Zählvariable	m, n	$m, n \in \{0 \dots 10\}$	

Tabelle 3.5: Variablen

Gatter standardisiert und ist auf $T_G = 500ps$ normiert. Auf der z-Achse sieht man die Auftretswahrscheinlichkeit der Effekte, die Werte aller Effekte variieren in einem Bereich von 0-2%. Die Anzahl der LD korreliert in umgekehrter Weise auch mit jenen der PF. Somit, steigt die Anzahl an LD, sobald jene der PF sinkt und umgekehrt. Die Anzahl der DL befindet sich konstant bei 50%. LD und PF sind mit 27-25% und 23-25% ungefähr gleich häufig.

Die Ergebnisse des Modells sind für CAL als 3D Graphen für DL, LD und PF in den Abbildungen 3.12d, 3.12e und 3.12f dargestellt. Die Anzahl der DL und LD korreliert in umgekehrter Weise auch mit jenen der LD. Daher, steigt die Anzahl an DL und LD sinkt jene der PF, und umgekehrt. Im Gegensatz zu NCL/LEDR liegt die Wahrscheinlichkeit für ein DL mit 21-21,5% bei ungefähr der Hälfte. Die geringere Anzahl an DL verteilt sich zu einem Drittel auf die der PF und zu zwei Drittel auf jene der LD.

Für alle beide Implementierungen der Kontrolllogik bietet sich dasselbe Bild: steigt die Anzahl der unerwünschten Effekte mit Länge des Zyklus, dabei ist es unerheblich ob die Pipeline token- oder bubble-limitiert ist. Die Grenze zwischen diesen Zuständen ist um ein Delay von $4T_G$ an der y-Achse verschoben, was der Differenz der Minimalwerte für T_{Source} und T_{Sink} entspricht. Die Minima und Maxima aller Technologien sind in Tabelle 3.6 zusammengefasst dargestellt.

[%]	NCL/LEDR	CAL
DL	50	21,5-24,5
LD	25,5-31,5	46,5-47,5
PF	18,5-24,5	28-32

Tabelle 3.6: Übersicht CTRL-Block

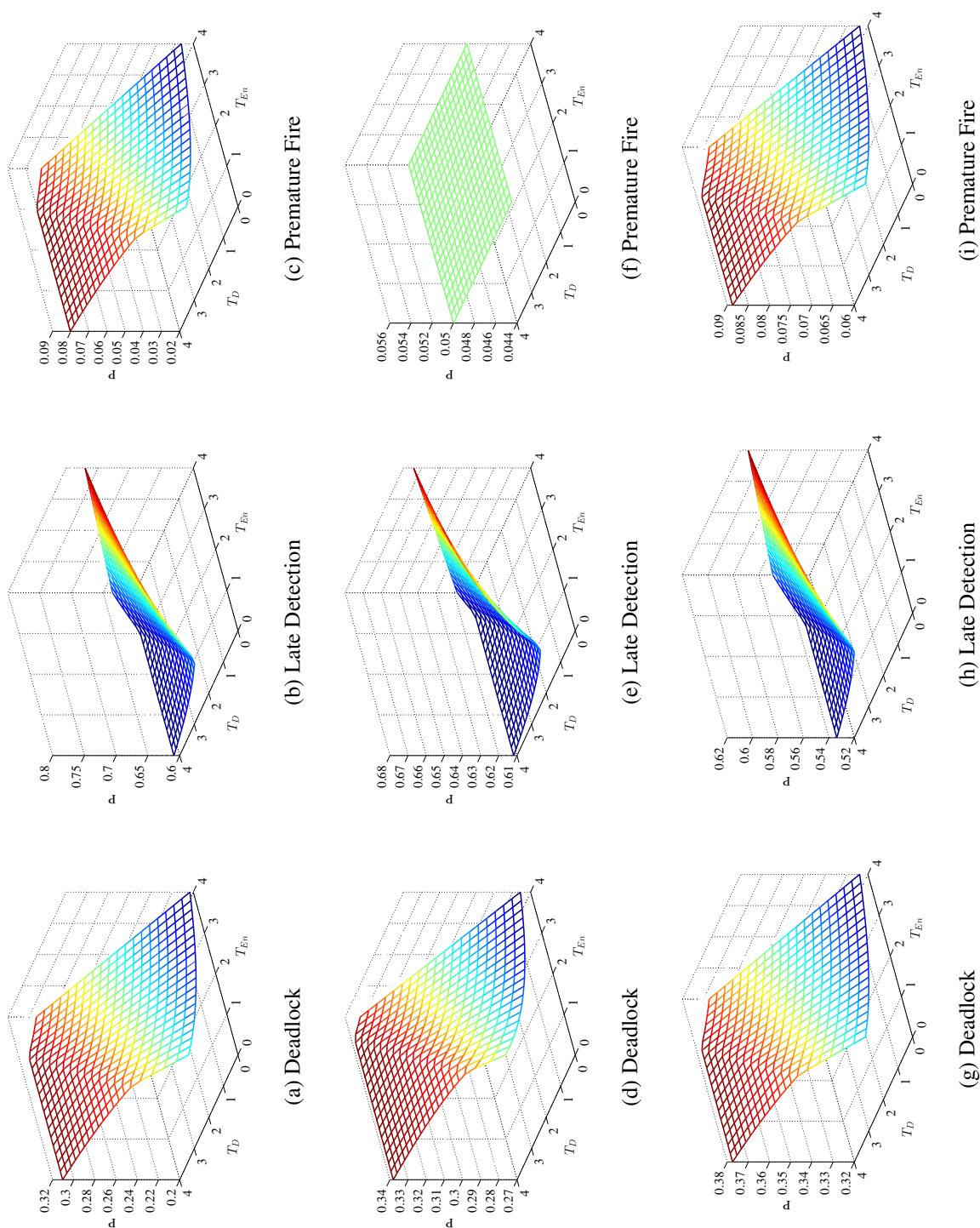
Es ist zu beobachten, dass die Anzahl der Effekte im vernachlässigbaren Maße variiert. Die Kontrolllogik für NCL/LEDR, bestehend aus einem C-Element, ist robuster als jene für CAL.

3.2.4 Register

Der Block mit den Registern verwendet technologiespezifische Speicherelemente. Für NCL werden C-Elemente, für LEDR werden DET-FF und für CAL werden D-Latches verwendet.

Für die Berechnung des Modells des REG wurden die, in Tabelle 3.7 aufgelisteten, Werte gewählt. Da die Ergebnisse präsentiert werden, nicht aber der Rechenweg, sei hier auf die sich in den Tabellen 3.7, C.6 für NCL, C.7 für LEDR und C.8 für CAL, befindlichen Werte verwiesen, durch welche eine Berechnung möglich wird.

Abbildung 3.13: Resultate des Modells für REG NCL(a-c), LEDR(d-f), CAL(g-i)



Variable	Symbol	Wert		
		NCL	LEDR	CAL
Verhältnis $s@0$ zu $s@1$	sa_0	0,5		
Verhältnis zwischen Phase φ_0/φ_1	φ_0	0,5		
Internes Delay	T_{REG}	200ps	600ps	200ps
Delay eines Dual-Rail-Gatter	T_G	500ps	1150ps	1016ps
Signallaufzeit T_D	T_y	$T_{REG} + T_{Source} + mT_G$		
Signallaufzeit T_{En}	T_x	$T_{REG} + \max(T_{Source} + nT_G, T_D)$		
Zählvariable	m, n	$m, n \in \{0 \dots 10\}$		

Tabelle 3.7: Variablen

In Abbildung 3.13 sind die Ergebnisse des Modells für das C-Element als 3D Graphen für DL (3.13a), LD (3.13b) und PF (3.13c) dargestellt. Die Graphen besitzen als x- und y-Achse die absolute Dauer der Transition- bzw. Stable-Phase. Um den späteren Vergleich zwischen verschiedenen Technologien zu ermöglichen, wurde die Dauer auf die Zeiteinheit eines Dual-Rail Gatter wieder standardisiert und ist auf $T_G = 500ps$ normiert. Auf der z-Achse sieht man die Auftrittswahrscheinlichkeit der Effekte, die Werte aller Effekte variieren in einem Bereich von 6-20%. Die Anzahl der DL und PF korreliert in umgekehrter Weise mit jenen der LD. Somit, steigt die Anzahl an DL/PF, sobald jene an LD sinkt, und umgekehrt. Die Anzahl der DL wird bei 20% minimal mit einem hohen Wert für T_{Sink} und nähert sich bei $T_{Sink} = T_{Source}$ ihren Maximum von 32% mit 31% stark an. Die Menge der PF verhält sich äquivalent zu jenen der DL, allerdings liegen die markanten Punkte bei 2% für das Minimum und 8% für das Maximum sowie 7,5% für $T_{Source} = T_{Sink}$. Umgekehrt zu DL und PF verhalten sich die LD, sie erreichen ihr Minimum von 60% bei $T_{Source} \geq T_{Sink}$. Das Maximum von 80% für die auftretenden LD befindet sich bei einem möglichst hohen T_{Sink} .

Das selbe Bild wie bei einem C-Element bietet sich für das D-Latch (Abbildungen 3.13g - 3.13i) als Speicherelement. Abweichend davon verhält sich das DET-FF (Abbildungen 3.13d - 3.13f) welches bei den PF eine Wahrscheinlichkeit von konstant 5% hat. Die theoretischen Maxima und Minima aller Register sind in Tabelle 3.8 gegenübergestellt. Die Register profitieren von einer bubble-limitiert Pipeline.

[%]	NCL	LEDR	CAL
DL	15-27,5	25-35	30-40
LD	57,5-85	60-70	50,5-65
PF	0-15	5	5-9,5

Tabelle 3.8: Übersicht Register

3.2.5 Funktionslogik

Bei logischen Funktionen einer Pipeline-Stufe ist es, wie in Kapitel 2.8 beschrieben, nötig eine speziell auf die Codierung zugeschnittene Art der Implementierung zu verwenden. In der Literatur gibt es bereits vollständige qualitative Analysen hierfür, eine Übersicht dieser bietet Tabelle 3.9.

Bezeichnung	Verwendung	Literatur
Self Timed Combinational Logic Module	NCL	[PN95, LM04]
Delay insensitive Minterm Synthesis	NCL	[SSDS92]
Threshold Gates	NCL, LEDR-EVAL	[FB96b, DWD91]
Programmable Logical Array	CAL, LEDR-PLA	[SDH04, DWD91]

Tabelle 3.9: Übersicht qualitative Analysen

Durch die Abhängigkeit der Verteilung der auftretenden Effekte eines Stuck-At von der zu implementierenden Funktion, ist es nicht möglich, eine allgemeingültige Aussage (wie beim PD in Kapitel 3.2.2) zu treffen. Es ist jedoch möglich, aus der Literatur eine Liste an Äquivalenzen abzuleiten, um festzuhalten, in welcher Phase, welches Stuck-At auftritt. Die möglichen Effekte am Ausgang der Logik entsprechen jenen in Abschnitt 3.1.2 beschrieben.

Mit welcher Wahrscheinlichkeit bzw. ob ein Effekt auftritt, muss anhand der zu implementierenden Funktion ermittelt werden. Die in Tabelle 3.9 festgehaltene Literatur vermag dabei zu helfen. Hervorzuheben ist jedoch, dass durch die Hysterese der Logik, dh. die Logik liefert nur Ergebnisse wenn eine Phase stabil anliegt, ein Propagieren von illegalen Codewörtern verhindert wird.

3.3 Allgemeines Pipeline Template

Im vorhergehenden Abschnitt wurden die Komponenten einer Pipeline-Stufe separat betrachtet. Ausgehend von den Einzelergebnissen soll, nun hier das Modell aus Abschnitt 3.2.1.3 erweiternd, eine komplette Pipeline-Stufe analysiert werden.

3.3.1 Verfahren für die quantitative Analyse

Um das Verfahren zu veranschaulichen, wird auf das Pipeline-Template aus Abschnitt 3.1.1 zurückgegriffen. Das Template (siehe Abbildung 3.14) besteht aus den sieben Blöcken *Source*, *Sink*, $f(x)$, *LCD*, *RCD*, *CTRL* und *REG*. Jeder diese Blöcke besitzt ein bestimmtes Delay T_{Block} . Im Modell wird angenommen, dass ein Stuck-At-Fault nur in den Blöcken *LCD*, *RCD*, *CTRL* sowie *REG* auftritt; *Source*, *Sink*, $f(x)$ und die umgebenden Pipeline-Stufen damit fehlerfrei funktionieren .

Um die Auftrittswahrscheinlichkeit eines SSAF für einen Block zu definieren, wird eine Variable g_{Block} eingeführt. Blöcke werden durch ihren internen Aufbau unterschiedlich gewichtet. Für die Gewichtung G_{Block} eines Blockes werden die Ein- und Ausgänge der zugrundeliegenden Schaltung entsprechend ihrer Busbreite b gezählt (siehe Tabelle 3.10). Um G_{Block} in Relation zur Gesamtschaltung zu setzen, wird der Wert durch die Anzahl G_s aller Ein- und Ausgänge einer Pipeline-Stufe dividiert.

$$G_s = G_{CTRL} + G_{REG} + 2G_{CD}$$

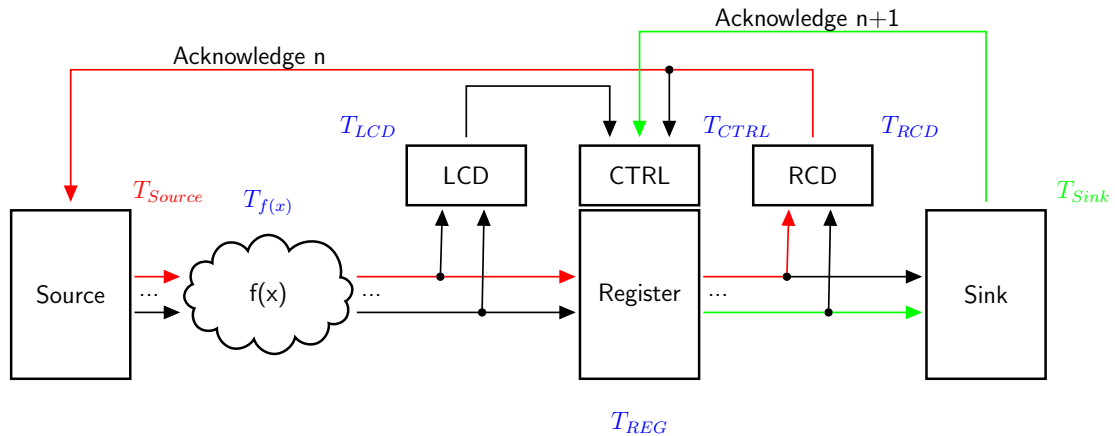


Abbildung 3.14: Abstraktes Pipeline Template

Die gewichtete Auftretswahrscheinlichkeit eines Stuck-At ist durch die Funktion

$$P_{Block} = \frac{G_{Block}}{G_S} * g_{Block}$$

gegeben.

Variable	Zusammensetzung			$G(b = 2)$		
	NCL	LEDR	CAL	NCL	LEDR	CAL
G_{CTRL}	4		11	4	11	
G_{REG}	6 * b			12		
G_{LCD}/G_{RCD}	4 * b + 1		5 * b + 5	9		15

Tabelle 3.10: Gewichtung

Soll nun das Verhalten einer gesamten Pipeline-Stufe vorhergesagt werden, so geschieht dies aufbauend auf den Ergebnissen aus Kapitel 3.2.

Unter Anwendung des Verfahrens aus Abschnitt 3.2.1.2 muss nun analysiert werden, welche Auswirkung ein, in einem Block auftretender Effekt auf die folgenden Blöcke hat. Denn, tritt ein Premature Fire am Ausgang eines CD auf, muss daraus nicht auch in ein solches am Ausgang es CTRL resultieren. Im Gegensatz zur Analyse der einzelnen Blöcke, ist hier nun keine Unterscheidung zwischen den einzelnen Phasen mehr nötig. Da sich φ_0 und φ_1 gleich verhalten, wird nur mehr zwischen tr- φ und st- φ differenziert. Kommt es zum Beispiel in der Phase st- φ am Ausgang des CTRL zu einem DL, so ist keine neue Datenübernahme durch das Register möglich. Die aktuell vorhandenen Daten werden nicht verändert und eine Übernahme eines neuen Datenworts ist nicht mehr möglich, dadurch kommt es zu einer Late Detection. Die Menge der an CTRL auftretenden Deadlocks ist also der Menge an Late Detection zuzurechnen.

Die Ergebnisse aus der Anwendung des Verfahrens sind in der Tabelle 3.11 aufgelistet und werden verwendet um das Endergebnis mittels Algorithmus 2 zu erzeugen. Anschließend wird

die Gewichtung G mit der Verteilung M jedes Blockes multipliziert. Die nun in den Matrizen der Blöcke gewichteten Werte werden elementweise addiert und anschließend durch die Summe aller E/A dividiert.

$$Z = \frac{\sum_{i=0}^{|B|} M_i * G_i}{\sum G} \quad B = \{LCD, RCD, CTRL, REG\} \quad (3.22)$$

Signal	$Tr-\varphi$	$St-\varphi$	Signal	$Tr-\varphi$	$St-\varphi$
LCD.DL	DL,LD	LD	CTRL.DL	DL	LD
LCD.LD	LD	LD	CTRL.LD	LD	LD
LCD.PF	LD,PF	LD	CTRL.PF	PF	LD
RCD.DL	DL,LD	LD	REG.DL	DL	DL
RCD.LD	LD	LD	REG.LD	LD	LD
RCD.PF	LD,PF	LD	REG.PF	PF	PF

Tabelle 3.11: Qualitative Analyse der Pipeline

Daten : Quantitative Analyse T und Matrix M

Ergebnis : Gewichtete Matrix M_G

[ydl,yld,ypf] = M;

für alle Signal von T **tue**

unterscheide φ **tue**

Fall DL

 | $M_G.ydl += M[\text{Signal}];$

Fall LD

 | $M_G.yld += M[\text{Signal}];$

Fall PF

 | $M_G.ypf += M[\text{Signal}];$

Ende

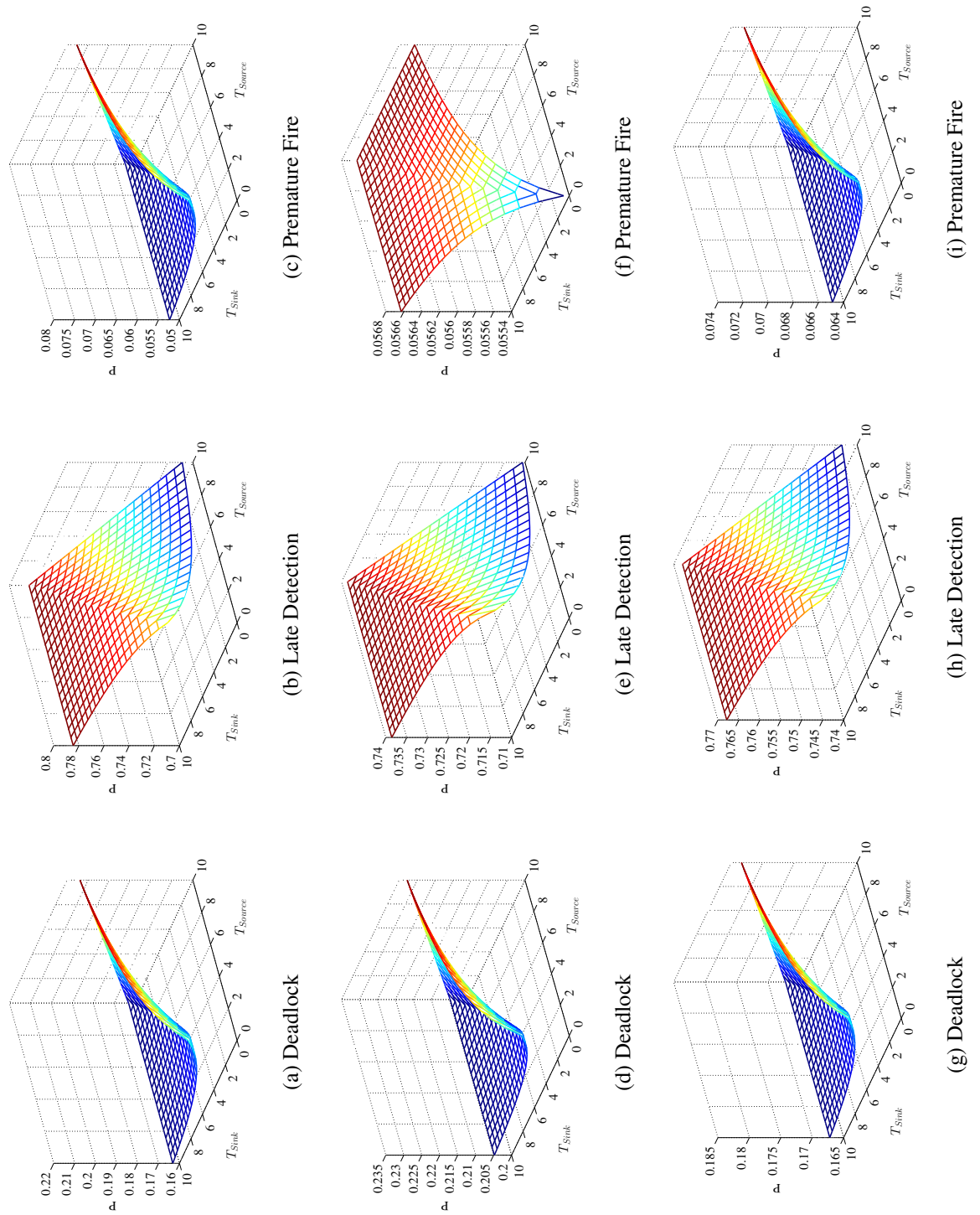
Ende

Algorithmus 2 : Gewichtung der Blöcke

3.3.2 Anwendung auf das Pipeline Template

Aufbauend auf den Zwischenergebnissen aus den vorhergehenden Abschnitten wurde das Modell, für jede der verwendeten Technologien NCL, LEDR und CAL, berechnet.

Abbildung 3.15: Resultate des Modells für eine Pipelinestufe NCL(a-c), LEDR(d-f), CAL(g-i)



In Abbildung 3.15 sind die Ergebnisse des Modells für NCL als 3D Graphen für DL (3.15a), LD (3.15b) und PF (3.15c) dargestellt. Die Graphen besitzen als x- und y-Achse die absolute Dauer der Transition- bzw. Stable-Phase. Um den späteren Vergleich zwischen verschiedenen Technologien zu ermöglichen, wurde die Dauer auf die Zeiteinheit eines Dual-Rail Gatter standardisiert und ist auf $T_G = 500ps$ normiert. Auf der z-Achse sieht man die Auftretswahrscheinlichkeit der Effekte, die Werte aller Effekte variieren in einem Bereich von 5-10%. Die Anzahl der DL korreliert mit jenen der PF und in umgekehrter Weise auch mit jenen der LD. Daher, steigt die Anzahl an DL und PF, sobald jene der LD sinkt, und umgekehrt. Die Anzahl der DL nimmt mit zunehmender Länge von T_{Source} zu und erreicht ein Maximum von 22%. Jenseits der Geraden $T_{Sink} \geq T_{Sink}$ erreicht die Wahrscheinlichkeit für ein DL ihr Minimum von 16%. Dasselbe Verhalten zeigen auch die PF, welche Werte von 5-8% annehmen. In umgekehrter Weise trifft dieses Verhalten auch auf die Anzahl der LD zu, welche ihr Maximum von 78% bei einem hohen T_{Sink} erreichen.

Für alle drei Technologien (NCL, LEDR und CAL), bietet sich dasselbe Bild, so profitiert man von einer kurzen Pipelinestufe und einer Pipeline welche bubble-limitiert ist. Einzige Ausnahme ist die Auftretswahrscheinlichkeit für PF bei LEDR, diese nimmt unabhängig davon ob eine Pipeline bubble- oder token-limitiert ist bei einer höheren Zyklusdauer zu. Die Minima und Maxima aller Technologien sind in Tabelle 3.12 zusammengefasst dargestellt.

[%]	NCL	LEDR	CAL
DL	16-22	20 -24	16 -19
LD	70-80	71-74	74 -77
PF	5-8	5,5- 5,6	6-8

Tabelle 3.12: Übersicht Pipeline Stufe

Es ist zu beobachten, dass die Anzahl der bösartigen Effekte (PF) im mittleren einstelligen Bereich sind, und kaum variiert. Für alle Technologien gilt, dass sie zuverlässiger arbeiten sobald sie bubble-limitiert sind.

Weiters zeigt sich, dass der Anteil der PF bei Betrachtung der gesamten Pipeline geringer ist als bei den einzelnen Komponenten. Dies liegt daran, dass DL und LD dominante Effekte sind: Ein Deadlock in einer Komponente kann in der nächsten nur wieder einen DL bewirken; dies liegt in der Definition von Delay Insensitivity: Die folgende Komponente MUSS auf die (wegen des SSAF niemals ankommende) nächste Flanke warten und kann daher nur mit DL reagieren. Umgekehrt führt ein PF einer Komponente in nicht wenigen Fällen zu einem DL oder LD in einer anderen Komponente, womit unter Umständen ein möglicher bösartiger Effekt verhindert wird.

3.4 Zusammenfassung

In diesem Kapitel wurde das Verhalten asynchroner Pipelines gegenüber SSAF diskutiert. Es wurden Deadlock, Late Detection, Premature Fire als mögliche Effekte identifiziert. Für die

Menge der Premature Fire wurde weiters festgestellt, dass sie Ursache für von der Spezifikation abweichendes Verhalten ist. Dies geschieht in der Form von Token-Consumption, Token-Generation und Illegal-Token-Generation. Ein Pipeline-Template wurde abstrahiert um ein Vorhersagemodell, zu den oben genannten Kategorien, zu schaffen. Um die Basis für die Analyse der Gesamtpipeline zu schaffen, wurden, basierend auf den Leitsatz „Divide et impera“, zuerst einzelne Komponenten einer Pipeline betrachtet. Die Anzahl der „erwünschten“ Fälle nach Eintreten eines SSAF wurden mit 92-98% bestimmt. Das Modell ist in der Hinsicht limitiert, als dass es nicht möglich ist zwischen den durch Premature-Fire verursachten abweichenden Verhalten zu differenzieren. Der berechnete Wert von 2-8% erhält dadurch die Gewichtung eines Worst-Case-Szenarios, da die gutartigen Effekte, nämlich Isochron-Delay- und Transparent-Timing-Fault, nicht subtrahiert werden können.

Quantitative Analyse und Verifikation durch Simulation

In diesem Kapitel wird die Analyse aus dem vorhergehenden Kapitel mittels Verhaltenssimulation einer in VHDL umgesetzten Pipeline-Stufe fortgesetzt. Das Verhalten der Gatter bezüglich ihres Timings wird mittels Logical-Effort bestimmt. Für den Interconnect wird die Isochronität der Schaltung und damit ein Zero-Delay angenommen [LM04, SN01b]. In einem ersten Schritt wird geklärt, wie eine Schaltung aussehen kann und welche Komponenten dabei essenziell sind.

4.1 Verfahren und Testaufbau

Für den Testaufbau wird das in [Lal12] vorgeschlagene Verfahren verwendet. Die Testbench (siehe Abbildung 4.1) ist eine virtuelle Umgebung, die dazu dient eine Schaltung auf Korrektheit und Zuverlässigkeit zu prüfen. Durch sie werden die, eine Pipeline-Stufe umgebenden Stufen, (Source und Sink) abgebildet, sodass eine Simulation der Schaltung möglich ist. Die zu testende Schaltung wird auch Device Under Test (DUT) genannt. Der Source-Block stellt, die aus einer Datei gelesenen, Daten entsprechend der Schaltungsdefinition bereit. In der Testbench gibt es zwei Instanzen des *DUT*, die erste dient als Referenz (DUT_{ref}), die zweite Instanz (DUT_{tst}) hingegen ist jene, in welche im Laufe der Simulation ein Stuck-At injiziert wird. DUT_{tst} wird mit einem 1-Bit-Hot-Bus verbunden, über welchen das Fehlverhalten gesteuert wird. Hierzu ist die Anpassung des VHDL Codes nötig. Die Ausgabe von DUT_{ref} und DUT_{tst} wird mittels einem Komparator verglichen. Die Testbench kommuniziert bidirektional mit einem Python-Skript, welches den Stimulus Generator und die Auswertung der Daten implementiert.

Um die Behavioral-Simulation der zu testenden Schaltung durchzuführen wird zamiaCAD¹ in Version 0.11.3 vom 25. Feber 2014 verwendet. Dabei handelt es sich um eine unter GNU

¹<http://zamiacad.sourceforge.net/>

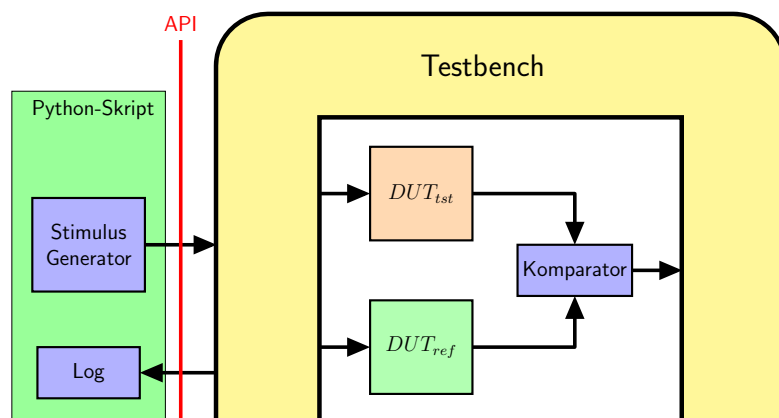


Abbildung 4.1: Aufbau Testbench

General Public License Version 3 lizenzierte, modulare und erweiterbare Plattform. ZamiaCAD wird als Plugin für die Eclipse Plattform (siehe Abbildung 4.2) angeboten und bietet einen vollständigen Parser für VHDL 2003, sowie einen integrierten Simulator, welcher über eine API für die Skriptsprache Python ansprechbar ist. Entwickelt wird die Plattform von der Technischen Universität Tallinn in Kooperation mit IBM.

4.2 Schaltungsaufbau

Für die Delays der Schaltung (siehe Abbildung 4.3) können die im vorherigen Abschnitt berechneten, Werte verwendet werden. Es bleiben T_{Source} und T_{Sink} als Variablen.

Aufgrund der Resultate aus Abschnitt 3.2.3 ist bekannt, dass das Endergebnis durch die Differenz aus T_{Source} und T_{Sink} beeinflusst wird, nicht jedoch durch ihre absoluten Werte. Es wird jeweils für die Technologien NCL, LEDR und CAL eine, der Abbildung 4.3 entsprechende, Pipeline erstellt. Die als $f(x)$ dargestellte Logik wird als abstrakter Teil von *Source* gesehen und nicht implementiert.

4.2.1 Codeerweiterung für Fault-Injection

Um in den Schaltungen an allen Ein- und Ausgängen eine Fault-Injection betreiben zu können, ist es nötig, spezielle Anpassungen für alle zu testenden Gatter durchzuführen. Die nötigen Anpassungen für die Simulation sollen anhand eines OR Gatters demonstriert werden. In Auflistung 1 ist ein OR Gatter mit 2 Eingängen und einem nicht invertierten Ausgang als VHDL-Code dargestellt.

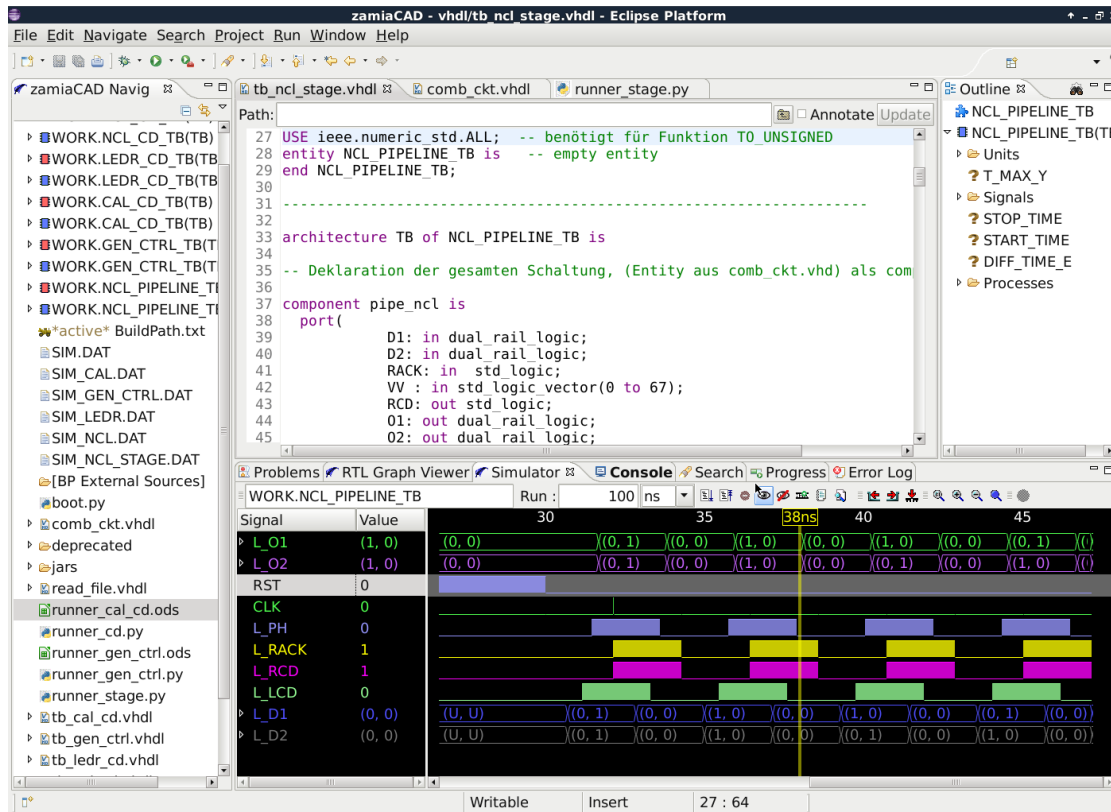


Abbildung 4.2: zamiaCAD mit aktivem Simulator

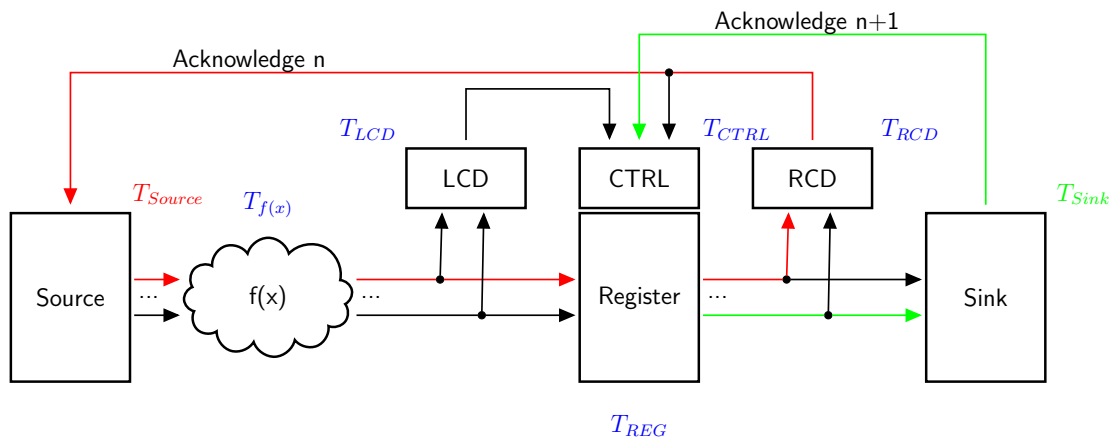


Abbildung 4.3: Pipeline Template

```

1 entity OR_GATE is
2 generic (
3   constant delay : time := 4 ns;
4 );
5 port (   X:         in std_logic;
6         Y:         in std_logic;
7         Z:         out std_logic;
8 );
9 end OR_GATE;
10
11 process
12 begin
13     Z <= (X or Y) after delay; -- logische funktion
14 end process;
15 end behv;

```

Aufistung 1: VHDL Code für Oder Gatter

Um den, für die Testbench notwendigen, 1-Hot-Bus zu integrieren, ist es nötig, für jeden E/A zwei Bit zur Verfügung zu stellen. Für ein OR-Gatter ist eine Busbreite von 6 Bit nötig. Diese wird folgendermaßen realisiert:

```

1 STUCK_AT: in std_logic_vector(0 to 5)

```

Um das Verhalten von SSAF zu implementieren, werden die lokalen Signale X_t , Y_t erzeugt. Die logische Funktion muss dem entsprechend angepasst werden. Durch die Verwendung eines *SWITCH*-Konstrukts wird das dynamische Routing der Eingänge realisiert.

```

1 case stuck_at is
2   when "100000" => -- A SA0
3     X_t <= '0';
4     Y_t <= Y;

```

Aufistung 2: Code für Fault Injection

Für ein Stuck-At 0 am Eingang X wird das Codewort 100000 reserviert. Der Eingang Y_t wird zum lokalen Signal geroutet und das Signal X_t permanent auf 0 gesetzt. Das Ergebnis für das OR-Gatter ist in Auflistung 3 zu sehen.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity OR_GATE is
5 generic (
6     constant delay : time := 4 ns;
7 );
8 port (
9     X:          in std_logic;
10    Y:          in std_logic;
11    STUCK_AT:   in std_logic_vector(0 to 5); -- 1 Hot Bus
12    Z:          out std_logic;
13 );
14
15 architecture behv of OR_GATE is
16     signal X_t, Y_t : std_logic;
17     -- 0: disabled
18     -- 1-3 enabled fuer x,y,z sa0
19     -- 4-6 enabled fuer x,y,z sa1
20 begin
21
22 FAULTINJ: process
23     begin
24     case stuck_at is
25         when "100000" => -- A SA0
26             X_t <= '0';
27             Y_t <= Y;
28         when "010000" => -- B SA0
29             X_t <= X;
30             Y_t <= '0';
31         when "001000" => -- C SA0
32             X_t <= '0';
```

```

33     Y_t <= '0';
34     when "000100" => -- A SA1
35         X_t <= '1';
36         Y_t <= Y;
37     when "000010" => -- B SA1
38         X_t <= X;
39         Y_t <= '1';
40     when "000001" => -- C SA1
41         X_t <= '1';
42         Y_t <= '1';
43     when others => -- normale Funktion
44         X_t <= X;
45         Y_t <= Y;
46 end case;
47 end process;
48
49 process
50 begin
51     Z <= (X_t or Y_t) after delay; -- logische Funktion
52 end process;
53 end behv;

```

Auflistung 3: OR Gatter mit Fault Injection

4.2.2 Timing laut Logical-Effort

Die Gatterlaufzeit hängt von verschiedenen Faktoren, welche durch das Fertigungsverfahren bestimmt werden, ab. Für die Simulation wird eine Strukturbreite von $0.6 \mu m$ angenommen. Basierend auf den Grundlagen aus Kapitel 2.9, soll hier zum besseren Verständnis der Logical Effort für ein AND2-Gatter berechnet werden. Ein AND2-Gatter wird durch ein NAND2 mit angeschlossenem Inverter gebaut; es ist also nötig, den Pfad D zu berechnen. Die Formel dazu lautet:

$$D = \sum d_i \quad (4.1)$$

$$d = g * h + p \quad (4.2)$$

Die Basiswerte für g und p werden der Tabelle 2.3a, bzw. 2.3b entnommen. Der Electrical-Effort h ist das Verhältnis zwischen Ein- und Ausgängen und wird mit $h_{NAND2,INV} = 1$ angegeben. Es ergibt sich also:

$$d_{NAND2} = g_{NAND2} * h_{NAND2} + p_{NAND2} = \frac{4}{3} * 1 + 2 = \frac{10}{3} \quad (4.3)$$

$$d_{INV} = g_{INV} * h_{INV} + p_{INV} = 1 * 1 + 1 = 2 \quad (4.4)$$

$$D_{AND2} = \sum d_i = d_{INV} + d_{NAND2} = 2 + \frac{10}{3} = \frac{16}{3} = 5\frac{1}{3} \quad (4.5)$$

Der für D_{AND2} berechnete Wert ist mit τ zu multiplizieren, welches für $0.6\mu m$ mit $\tau = 50ps$ angegeben ist.

$$D = D_{AND2} * \tau = 5\frac{1}{3} * 50ps = 267ps \quad (4.6)$$

Das Delay für ein AND2-Gatter, bei einer Strukturbreite von $0.6\mu m$, ist $267ps$. Für alle weiteren benötigten Gatter sind die Ergebnisse in Tabelle 4.1 festgehalten.

Gatter	d	d_{abs} [ps]
NAND2	$\frac{10}{3}$	167
AND2 (NAND+INV)	$\frac{16}{3}$	267
NAND4	6	300
NOR2	$\frac{14}{3}$	233
OR2 (NOR+INV)	$\frac{17}{3}$	283
NOR4	7	350
XOR2	8	400
MC2	4	200
MC3	6	300
SRFF	$\frac{28}{3}$	466
DETFE	12	600
D-LATCH	4	200

(a) Gatter

Block	D_{abs} [ps]		
	NCL	LEDR	CAL
CD	483	600	1216
CTRL	300	300	617
REG	200	600	200
GATE	500	1150	1016

(b) Komponenten

Tabelle 4.1: Logical Effort bei $0.6\mu m$ ($\tau = 50ps$)

4.3 Stimulus Generator

Um ein Automatisieren der Simulation zu ermöglichen, wird ein externes Python Skript verwendet. Dieses benutzt das Application Programming Interface (API) des Simulators, um Signale der Testbench zu verändern. Konkret werden dabei der Zeitpunkt T und Ort L eines auftretenden

Stuck-At festgelegt, was zufällig oder innerhalb fest definierter Grenzen geschehen kann. Das Skript startet den Simulator über die, von der API bereitgestellten, Funktionen.

```

1 sim = openSim(0) #oeffne Testbench Nr.
2 sim.reset() #Zuruecksetzen der Simulation
3 run(sim, str(time)) #Lasse Simulation fuer time laufen

```

Nach Ablauf der Zeit T , welche durch die Variable *time* repräsentiert wird, greift die API auf den Fault-Injection-Vektor des DUT zu, um am Pin L ein Stuck-At zu injizieren. Danach läuft die Simulation bis zu ihrem Ende weiter, um anschließend den festgestellten Fehler auszulesen. Die Resultate werden in eine Datei im Comma-separated values (CSV) Format geschrieben, um dadurch eine weitere Auswertung zu ermöglichen. Es wird der prozentuale Anteil an Deadlock, Late-Detection und Premature-Fire festgehalten, damit ist es möglich, eine anschließende Analyse mit geringem Aufwand durchzuführen.

4.4 Auswertung

Um ein abweichendes Verhalten des DUT_{tst} festzustellen, werden die Ausgänge mit jenen des DUT_{ref} verglichen. Je nach vorhandener Abweichung wird dies als Auftreten eines Deadlock, Late-Detection oder Premature-Fire gewertet. Das erfasste Resultat wird dann in eine Variable geschrieben:

```

1 signal faulttype : integer := 0;

```

Für den Datenpfad wird eine erweiterte Analyse vorgenommen. Die auftretenden Effekte werden in die folgenden sieben Kategorien eingeteilt:

Ein **Deadlock** (faulttype=1) tritt dann auf, wenn am Ausgang von DUT_{ref} eine Zustandsänderung auftritt, bei DUT_{tst} hingegen keine.

Ein **Late Detection Typ 1** (faulttype=2) tritt auf, wenn nach dem Injizieren des Stuck-At die Ausgänge von DUT_{ref} und DUT_{tst} weiterhin übereinstimmen.

Ein **Late Detection Typ 2** (faulttype=3) tritt auf, wenn nach dem Injizieren des Stuck-At die Ausgänge von DUT_{ref} und DUT_{tst} nicht mehr übereinstimmen. Der Komparator der Testbench erhält beide Signale jedoch zum selben Zeitpunkt. Es handelt sich um eine token-Consumption.

Ein **Premature Fire Typ 1** (faulttype=4) tritt auf, wenn am Ausgang von DUT_{tst} vor dem des DUT_{ref} eine Zustandsänderung auftritt, die Ausgänge von DUT_{ref} und DUT_{tst} aber übereinstimmen.

Ein **Premature Fire Typ 2** (faulttype=5) tritt bei NCL auf, wenn am Ausgang von DUT_{tst} vor dem des DUT_{ref} eine Zustandsänderung auftritt. Am Ausgang DUT_{tst} liegt der illegale, aber für den CD nicht als solches erkennbare, Zustand $D_n.Rail0 = 1$ und $D_n.Rail1 = 1$ vor.

Ein **Premature Fire Typ 3** (faulttype=6) tritt auf, wenn am Ausgang von DUT_{tst} vor dem des DUT_{ref} eine Zustandsänderung auftritt. Es ist ein gültiges Codewort vorhanden, wobei eines oder mehrere Bits auf einem Rail vertauscht (Bit-Flip) sind. Es handelt sich um eine token-Generation.

Ein **Premature Fire Typ 4** (faulttype=7) tritt auf, sobald am Ausgang von DUT_{tst} vor dem des DUT_{ref} eine Zustandsänderung auftritt. Dabei ist das eigentlich zuletzt stabil werdende Bit betroffen, sodass nach dem Stabilwerden des vorletzten ein gültiges, aber falsches Codewort übernommen wird. Es handelt sich um eine Illegal-token-Generation.

Für eine Übersicht über alle Variablen der Testbench empfiehlt sich die Lektüre von Anhang D.

4.4.1 TSL

In Abbildung 4.4 sind die Ergebnisse der Simulation für die Stufe einer NCL Pipeline als 3D Graphen für DL (4.4a), LD (4.4b) und PF (4.4c) dargestellt. Zusätzlich sind LD weiter in LD1 (4.4d), LD2 (4.4e) und PF in PF1 (4.4f) sowie PF2 (4.4g) aufgeschlüsselt.

Die Graphen besitzen als x- und y-Achse die absolute Dauer der Source- bzw. Sink-Pipelinestufe. Um den späteren Vergleich zwischen verschiedenen Technologien zu ermöglichen, wurde die Dauer auf die Zeiteinheit eines Dual-Rail Gatter standardisiert und ist mit $T_G = 500ps$ aufgenormt. Auf der z-Achse sieht man die Auftretswahrscheinlichkeit der Effekte. Die Werte aller Effekte variieren in einem Bereich von 1-8%. Die Anzahl der DL nimmt mit zunehmender Dauer eines Zyklus zu und erreicht, startend vom Minimum (18%), ein Maximum von 24%. Die Anzahl der LD nimmt mit zunehmender Dauer eines Zyklus minimal ab, jedoch gibt es bei dem Punkt $T_{Source} \geq T_{Sink}$ einen Abfall um 5%. Dieser Abfall bewirkt eine Zunahme von PF von 2 auf 7%. Daraus lässt sich schließen, dass die Pipeline blockierend robuster ist als verhungert/ Starvation.

Die Menge der LD setzt sich aus einem Großteil nicht-bösartiger Effekte zusammen, und einem Anteil, von bis zu 4%, des Typs LD2. Dieser tritt auf, wenn durch ein SA ein valides Codewort erzeugt wird, und das gültige überschrieben wurde (Token Generation).

Die Menge der PF setzt sich aus einem Großteil nicht bösartiger zusammen (PF1 6%), es kommt durch ein PF zu einem LD. Mit 4% ist PF2 ein bösartiges PF, es wird ein Illegal-Token ($D.rail0 = 1, D.rail1 = 1$) erzeugt. Ohne zusätzlichen Fault Detector wie in [PN95], kommt es zu einer fehlerhaften Berechnung. PF3 und PF4 kommen nicht vor.

Die Anzahl der unerwünschten Effekte steigt mit Länge des Zyklus und sie sinkt rasch auf ihr Minimum, sobald die Pipeline bubble-limitiert ist.

Die Ergebnisse decken sich mit denjenigen des Modells in Kapitel 3.3.2. Das Modell gibt die Tendenzen an, ist jedoch nicht punktgenau; z.B wurde die Stufe bei $x = y$ als Kurve mit geringer Steigung dargestellt.

4.4.2 LEDR

In Abbildung 4.5 sind die Ergebnisse der Simulation für die Stufe einer LEDR Pipeline als 3D Graphen für DL (4.5a), LD (4.5b) und PF (4.5c) dargestellt. Zusätzlich sind LD weiter in LD1 (4.5d), LD2 (4.5e) und PF in PF1 (4.5f) sowie PF2 (4.5g) aufgeschlüsselt.

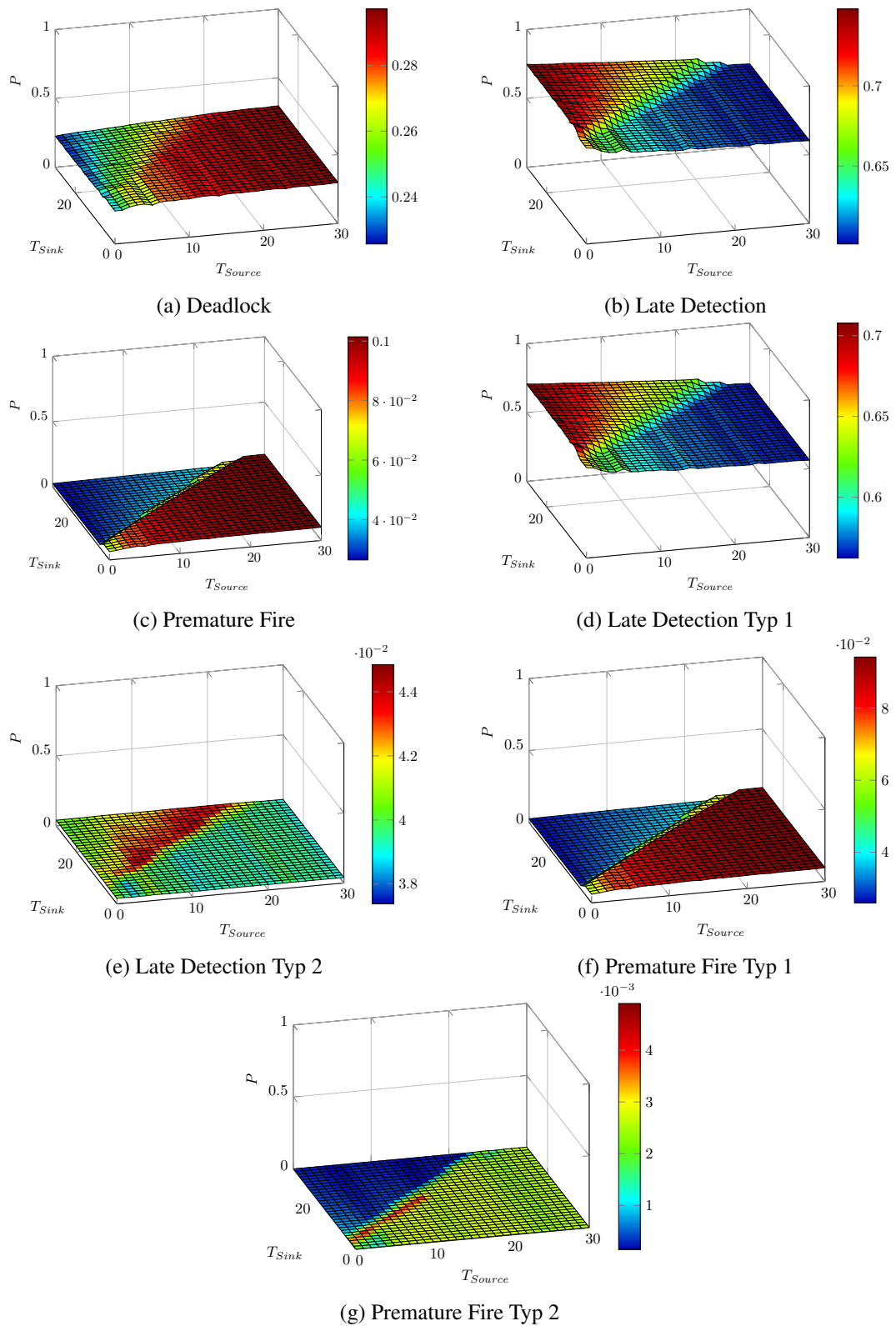


Abbildung 4.4: TSL Pipeline

Die Graphen besitzen als x- und y-Achse die absolute Dauer der Source- bzw. Sink-Pipelinestufe. Um den späteren Vergleich zwischen verschiedenen Technologien zu ermöglichen, wurde die Dauer auf die Zeiteinheit eines Dual-Rail Gatter standardisiert und ist mit $T_G = 1150ps$ aufgenormt. Auf der z-Achse sieht man die Auftrittswahrscheinlichkeit der Effekte.

Die Anzahl der Deadlock steigt mit Länge des Zyklus, und in besonderer Weise sobald eine Pipelinestufe token-limited ist. Dies stellt sich im Graphen als eine Kante mit einer Differenz von 5% bei $T_{Source} = T_{Sink}$ dar. Die Anzahl der DL steigt von 15 auf 30 Prozent. Umgekehrt zur Anzahl der DL verhält sich jene der Late Detection. Ihre Anzahl nimmt bei einer steigenden Dauer von T_{Source} ab und sie sinken von 80% auf 65%. Wiederum gibt es bei $T_{Source} = T_{Sink}$ eine Kante, welche einen Sprung von -5% hat sobald eine Pipeline token-limitiert ist. Die Gesamtanzahl der LD setzt sich aus der Menge an LD des Typ 1 und Typ 2 zusammen. Late Detection des Typs 2 haben ihr Maximum, von einem halben Prozentpunkt, im Ursprung des Graphen. Bei Zunahme der Zyklusdauer nimmt ihre Anzahl ab und tendiert gegen 0. Die Menge der Premature Fire setzt sich aus PF Typ 1 und PF Typ 3 zusammen, die Anzahl variiert zwischen zwei und sechs Prozentpunkten. Der Anteil der PF3 ist von geringer Bedeutung und steigt im Bereich $T_{Source} \geq T_{Sink} \wedge T_{Source} \geq 2T_G$ von 0 auf 1% der Gesamtanzahl aller Effekte. Die Anzahl der PF1 macht einen Großteil aller PF aus und erreicht ihr Maximum von 5% entlang der Geraden $T_{Source} = T_{Sink}$. Bis zu einem Delay von $T_{Source} = 3T_G$ treten PF1 auch in einer token-limitiert Pipeline auf.

Zusammenfassend kann gesagt werden, dass eine mittels LEDR gebaute Pipeline zu 94%-iger Wahrscheinlichkeit nach Auftreten eines SSAF, ohne negative Auswirkungen auf die Datenkonsistenz, einfach stehenbleibt. Der Wert kann weiter Optimiert werden indem bei Design einer Pipeline darauf geachtet wird, dass diese bubble-limitiert ist.

4.4.3 CAL

In Abbildung 4.6 sind die Ergebnisse der Simulation für die Stufe einer LEDR Pipeline als 3D Graphen für DL (4.6a), LD (4.6b) und PF (4.6c) dargestellt. Zusätzlich sind LD weiter in LD1 (4.6d), LD2 (4.6e) und PF in PF1 (4.6f), PF3 (4.6g) sowie PF4 (4.6h) aufgeschlüsselt.

Die Graphen besitzen als x- und y-Achse die absolute Dauer der Source- bzw. Sink-Pipelinestufe. Um den späteren Vergleich zwischen verschiedenen Technologien zu ermöglichen, wurde die Dauer auf die Zeiteinheit eines Dual-Rail Gatter standardisiert und ist mit $T_G = 1016ps$ gewichtet. Auf der z-Achse sieht man die Auftrittswahrscheinlichkeit der Effekte. Die Anzahl der Deadlocks steigt mit der Länge des Delays T_{Source} . Es ist eine Zunahme von 10% von 12 auf 22 Prozent sichtbar. Umgekehrt zur Anzahl der DL verhält sich jene der Late Detection. Ihre Anzahl nimmt bei einer steigenden Dauer von T_{Source} ab und sinkt von 85% auf 70%. Wiederum gibt es bei $T_{Source} = T_{Sink}$ eine Kante, welche einen Sprung von -2% macht, sobald die Pipeline token-limitiert ist. Die Gesamtanzahl der LD setzt sich aus der Menge der LD des Typ 1 und Typ 2 zusammen. Late Detection des Typs 2 haben ihr Maximum, von einem 1,5%, im Ursprung des Graphen. Bei Zunahme der Zyklusdauer nimmt ihre Anzahl ab und tendiert bereits nach $\max(T_{Source}, T_{Sink}) \geq 3$ gegen 0%.

Die Menge der Premature Fire setzt sich aus PF Typ 1,3 und 4 zusammen, die Anzahl variiert

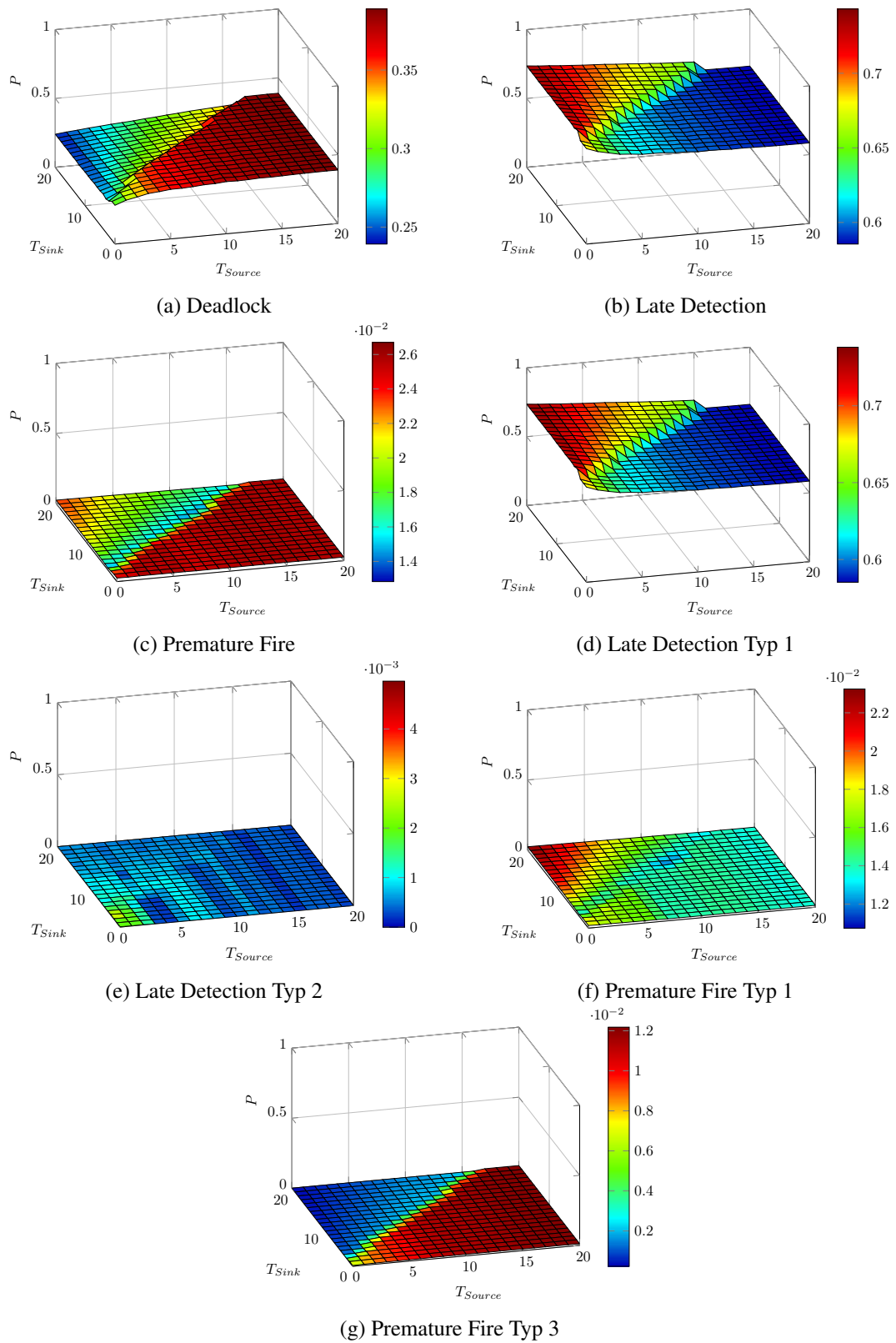
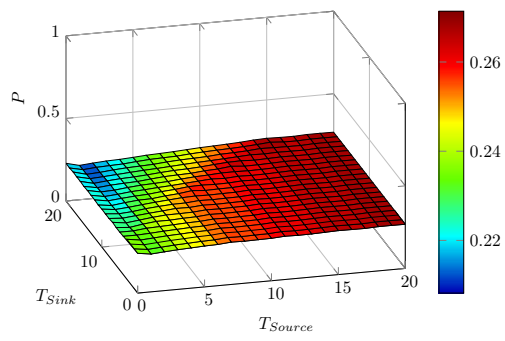
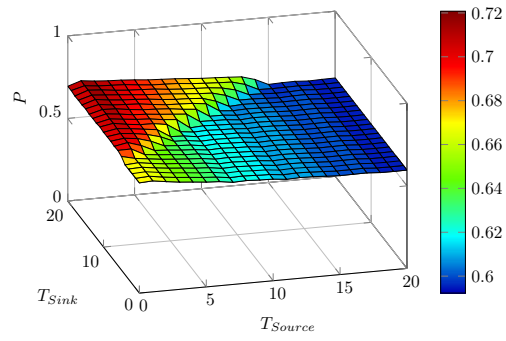


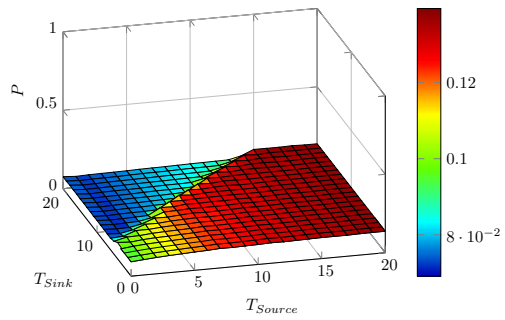
Abbildung 4.5: LEDR Pipeline



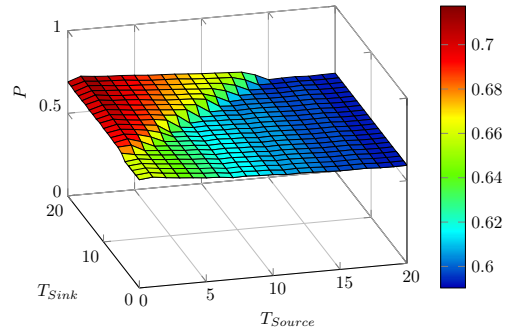
(a) Deadlock



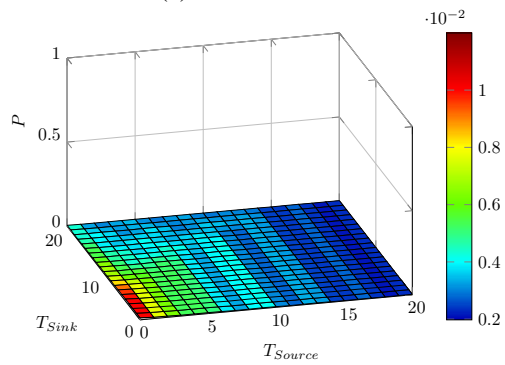
(b) Late Detection



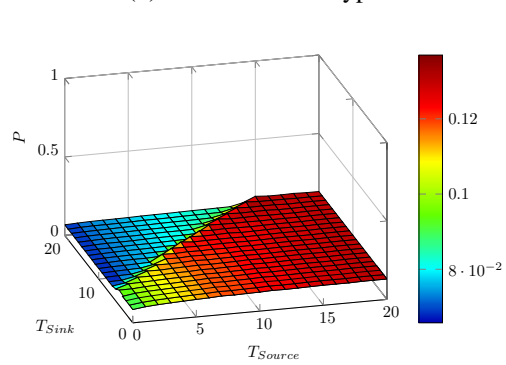
(c) Premature Fire



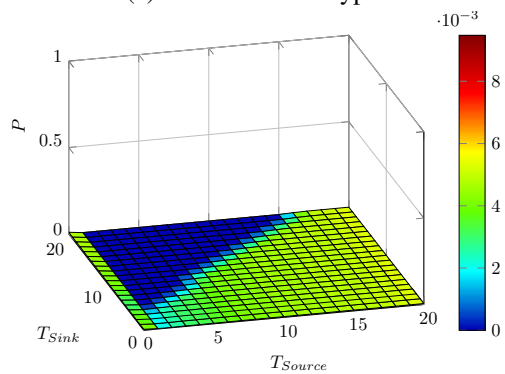
(d) Late Detection Typ 1



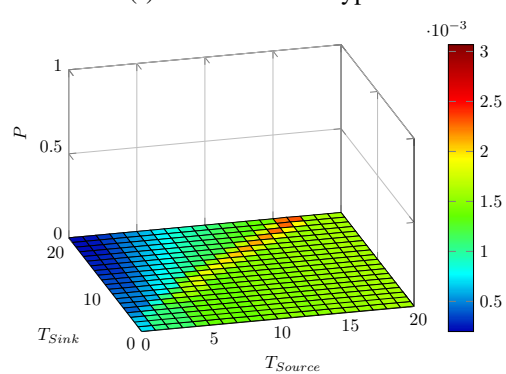
(e) Late Detection Typ 2



(f) Premature Fire Typ 1



(g) Premature Fire Typ 3



(h) Premature Fire Typ 4

Abbildung 4.6: CAL Pipeline

zwischen zwei und acht Prozentpunkten. PF3 und PF4 treten nur punktuell und vernachlässigbar auf. Für PF3 und PF4 ist die Auftretswahrscheinlichkeit mit 0,3% respektive 0,04% auf den Punkt $T_{Source} = T_{Sink} = 0,3T_G$ begrenzt. PF4 ist zusätzlich auch an der Geraden $T_{Source} = 1,8T_G \wedge T_{Sink} \geq 2T_G$ mit einem Maximum von 0,02% vorhanden. Die Anzahl der PF1 macht einen Großteil aller PF aus und erreicht ihr Maximum von 8% bei Zunahme von T_{Source} . An der Grenze zwischen token- und bubble-limitiert ist bei letzterem eine Zunahme von rund 5% zu sehen. In den Ergebnissen der Simulation sieht man dass die Pipeline am zuverlässigsten arbeitet sobald sie bubble-limitiert und die Zyklusdauer möglichst kurz ist. Es entfallen zwischen 94-96% der durch SSAF hervorgerufenen Effekte auf DL und LD.

4.5 Zusammenfassung und Vergleich zu Modell

In diesem Kapitel wurde eine Simulation des gewählten asynchronen Pipeline-Templates für NCL, LEDR und CAL durchgeführt. Es wurde eine vollständige Auswertung der durch Stuck-At Faults generierten Effekte erstellt. Durch den Detailgrad der durchgeführten Simulation war es möglich eine umfassendere Betrachtung als durch das in Kapitel 3.1.2 vorgestellte Modell zu tätigen. Dabei wurde Late Detection und Premature Fire weiter aufgeschlüsselt.

Insgesamt wurde festgestellt, dass alle begutachteten asynchronen QDI Pipelines bei bubble-limitierten kurzen Zyklen weniger fehleranfällig sind. Dies steht im Gegensatz zu synchronen Schaltungen, welche mit steigender Taktfrequenz, an ihre Grenzen stoßen. Weiters wurde beobachtet, dass es bei Schaltungen welche bubble-limitiert sind zu einer geringeren Anzahl an token-Consumption, Token Generation und Illegal Token Generation kommt. Es fällt auf, dass es zwar bei allen Technologien zu einer token-Generation (LD2) kommen kann; zu einer Illegal-token-Generation (PF3) kommt es nur bei LEDR und CAL, letzteres ist auch das einzige welches anfällig gegenüber token-Consumption (PF4) ist.

Betrachtet man nur die böartigen Fälle (Abbildung 4.7), so ist die Wahrscheinlichkeit dafür mit maximal 0,2 bis 4,5% nochmals geringer als jene für Premature Fire. Es stellt sich ein Vorteil für die FSL verwendenden Verfahren heraus, welche ihr Maximum mit 1,4% erreichen und damit um $\frac{1}{3}$ weniger böartige Fälle als NCL aufweisen. Der hohe Anteil an LD2 bei NCL kann durch die Anfälligkeit des Null-Zustandes gegenüber Stuck-At-1 erklärt werden. Hieraus ergibt sich auch die Diskrepanz zu dem erstellten Modell, welches zwischen gut- und böartigen Premature Fire nicht unterscheiden kann.

In Tabelle 4.2 sind alle Ergebnisse aus der Simulation denen des Modells gegenübergestellt. Man erkennt eine recht gute Übereinstimmung der Resultate aus Simulation und Modell; damit ist die Anwendbarkeit als Worst-Case-Modell bewiesen.

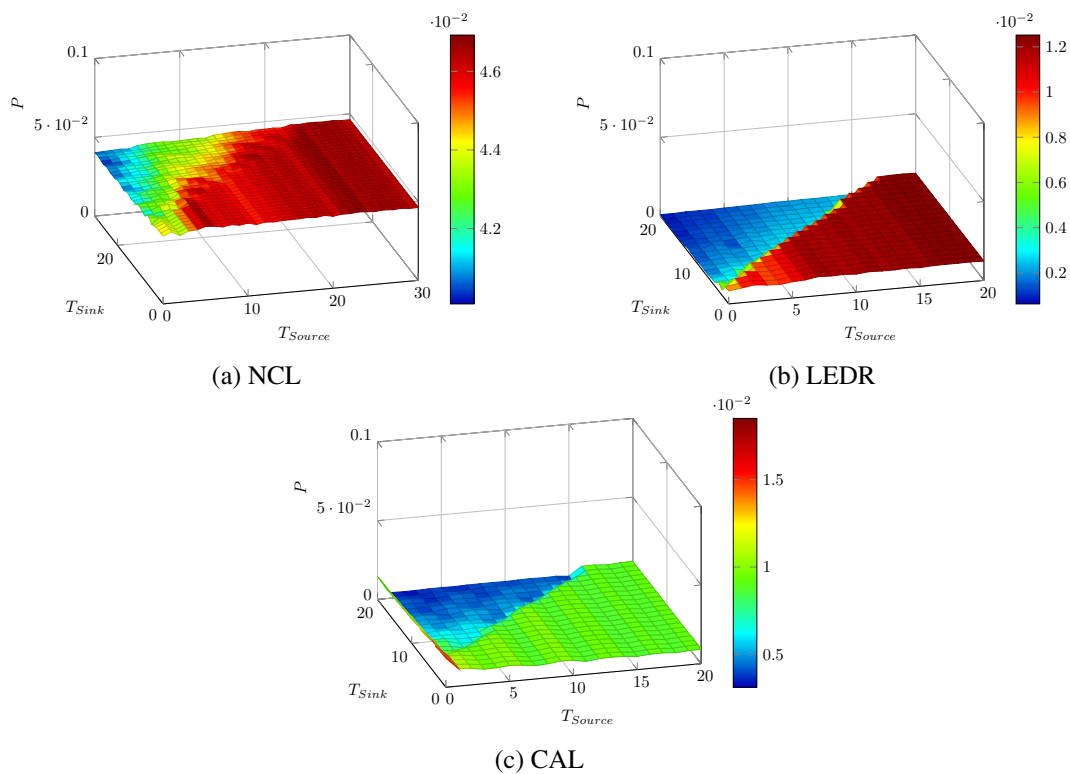


Abbildung 4.7: Verteilung bösartiger Effekte

	Simulation				Modell		
	NCL	LEDR	CAL		NCL	LEDR	CAL
DL	18-24	15-30	12-22	DL	16-22	20-24	16-19
LD	68-76	65-80	70-85	LD	70-80	70-74	74-77
LD1	64-76	65-80	70-85	LD1	-	-	-
LD2	4-4,5	0,2-0,5	0,8-1,4	LD2	-	-	-
PF	2-7	2-6	2-8	PF	5-8	5-6	6-7
PF1	2-6	2-5	2-8	PF1	-	-	-
PF2	0-0,4	0	0	PF2	-	-	-
PF3	0	0,2-0,8	0-0,4	PF3	-	-	-
PF4	0	0	0-0,06	PF4	-	-	-

Tabelle 4.2: Gegenüberstellung (Angaben in Prozent)

Conclusion

Es wurde ein Modell zur Vorhersage von Effekten durch Single Stuck-At Fault (SSAF) erstellt und auf ein gewähltes Pipeline Template unter Verwendung verschiedener Kodierverfahren angewendet. Die Ergebnisse wurden mit einer Behavioral-Simulation verglichen, wobei nur eine geringe Abweichung festgestellt wurde. Dieses Resultat bestätigt die Eignung des entwickelten Verfahrens. Die Simulation ermöglicht eine detailreiche Analyse, ist aber auch extrem zeitaufwendig, da sie bereits für dieses simple Template Tage bis Wochen benötigt. Das erstellte Modell bietet die Möglichkeit einen schnellen Überblick über die Eigenschaften einer Schaltung zu erhalten, verzichtet dabei jedoch auf eine Detailanalyse. Zusätzlich, zur in der Arbeit angestrebten Bestätigung des Modells, wurden die, durch die Simulation erhaltenen, konkreten Zahlen für Null Convention Logic (NCL), Level Encoded Dual Rail (LEDR) und Code Alternation Logic (CAL), genauer betrachtet. Bestätigt wurde auch die, in der Literatur beschriebene, Robustheit von Quasi Delay Insensitive (QDI) Kodierverfahren, deren unerwünschtes Verhalten, bei einem Auftritt eines SSAF, zwischen 0,2 und 4,5 Prozent liegt.

Auch wenn diese Diplomarbeit versucht, einen möglichst großen Teil des Fachbereichs der asynchronen Logik einzubeziehen, war es unmöglich alle Themen abzudecken. Es ergeben sich viele weitere Forschungsfelder:

Die Realisierung eines *komplexeren Modells* wäre durch das Verwenden von Petri-Netzen als Ausgangspunkt der Analyse vorstellbar. Dieses könnte, durch die Anwendung bereits vorhandener Methoden, zu einem automatisierten Verfahren führen.

In der Arbeit wurde der, die Funktionslogik implementierende, Teil der Schaltung nicht betrachtet, und das Augenmerk auf die Synchronisationslogik gelenkt. Die Auswirkungen des ersteren auf die Datenkonsistenz könnte weiter erforscht werden.

Ein weiterer Schritt wäre es, die Analysen auf komplexere Schaltungen anzuwenden, z.B. auf eine mehrstufige Pipeline, sie in Hardware zu gießen und dadurch das gesamte Verfahren zu verifizieren.

Beweise

Theorem A.0.1. *Es ist nötig den nächsten zu erwartenden Zustand zu berücksichtigen.*

Beweis. Geht man davon aus dass der nächste zu erwartende Zustand nicht zu berücksichtigen wäre würde dies bedeuten, dass für alle 4 Phasen jeweils die selben Effekte gelten würden. Dies kann jedoch widerlegt werden indem man für einen CD ein Stuck-At 0 am Ausgang angenommen wird. Tritt dies während $tr - \varphi_0$ auf so kommt es zu einer Late Detection, da in der nachfolgenden Phase $st - \varphi_0$ eben dieser Zustand auch erwartet wird. Kommt es hingegen während der $tr - \varphi_1$ zu einem solchen Stuck-At so ist Phasenwechsel nicht mehr möglich und es kommt zu einem Deadlock. Dies steht im Widerspruch zur getätigten Annahme, daher ist diese widerlegt. \square

Theorem A.0.2. *Wenn zwischen dem zuletzt stabil werdenden Rail(<letztes>) und den restlichen Rails(<andere>) unterschieden wird, so sind alle Zustände der Datenbits abgedeckt.*

Beweis. Ein CD besitzt eine Hysterese, welche an den Punkten

$$CD = \begin{cases} D_{0..N-1} = \varphi_1 & :\uparrow \\ D_{0..N-1} = \varphi_0 & :\downarrow \end{cases} \quad (\text{A.1})$$

definiert ist. Nehmen wir an, dass für einen Bus B mit den Datenbits D_0 bis D_{N-1} , die Busbreite $N = 2$, beträgt. Der Einfachheit halber wird angenommen, dass die einzelnen Bit des Bus entsprechend ihrer Nummerierung zu Zeitpunkt t_n stabil werden (D_0 vor D_1 usw.) und die aktuelle stabile Phase φ_0 anliegt. Um einen Phasenwechsel zu vollziehen ist also das Wechseln aller Datenbit in die neue Phase nötig.

Nehmen wir nun an, man kann diejenigen Datenbits des Bus nach deren Phasenwechsel kein Zustandswechsel am CD Zustande kommt zu einer Gruppe D_{andere} zusammenfassen und das Datenbit welches den Zustandswechsel auslöst $D_{letztes}$ nennen.

Für $N = 3$ würde die Abstraktion also lauten:

$$D_{andere} = \bigcup_{k=0}^1 D_k \quad D_{letztes} = D_2 \quad D_{andere} \cup D_{letztes} = \bigcup_{k=0}^{3-1} D_k \quad (\text{A.2})$$

Daraus lässt sich eine Formel für einen Bus der Breite N ableiten.

$$D_{andere} = \bigcup_{k=0}^{N-2} D_k \quad D_{letztes} = D_{N-1} \quad D_{andere} \cup D_{letztes} = \bigcup_{k=0}^{N-1} D_k \quad (\text{A.3})$$

Um die Korrektheit der obigen Formel zu beweisen, wird hier nun eine vollständige Induktion für $N = N + 1$ durchgeführt:

$$D_{andere} = \bigcup_{k=0}^{(N+1)-2} D_k \quad D_{letztes} = D_{(N+1)-1} \quad D_{andere} \cup D_{letztes} = \bigcup_{k=0}^{(N+1)-1} D_k \quad (\text{A.4})$$

Dies führt zu folgendem Ergebnis

$$D_{andere} = \bigcup_{k=0}^{N-1} D_k \quad D_{letztes} = D_N \quad D_{andere} \cup D_{letztes} = \bigcup_{k=0}^N D_k \quad (\text{A.5})$$

Aus welchem man durch herausheben von D_{N-1} , als nun vorletztes Glied, folgendes erhält:

$$D_{andere} = D_{N-1} \cup \bigcup_{k=0}^{N-2} D_k \quad D_{letztes} = D_N \quad D_{andere} \cup D_{letztes} = D_N \cup \bigcup_{k=0}^{N-1} D_k \quad (\text{A.6})$$

Diese Gleichung lässt nun erkennen dass der Induktionsschluss und somit die ursprüngliche Annahme korrekt war. Das durch Induktion hinzugekommene Signal D_{N+1} nimmt den Platz desjenigen ein, welches $D_{letztes}$ für N war und nun zu D_{andere} gehört. Es ist also möglich jeden Datenbus als abstrakten zwei Bit breiten Bus darzustellen, mit den Datenbit $D_{letztes}$ für das letzte eine Phase erreichende Bit und D_{andere} welches alle anderen Bit repräsentiert. \square

Quelltext

Da es Unsinnig ist den gesamten Quelltext von mehreren tausend Zeilen Code im Anhang aufzulisten, befindet sich dieser auf einem der Druckversion beiliegendem Speichermedium. Sollten Sie dieses Dokument ohne beiliegenden Quellcode erhalten haben, so können Sie diesen anfordern¹.

Verzeichnis	Beschreibung
pdf	Die Diplomarbeit im Portable Document Format
octave	Modell in GNU Octave implementiert
vhdl	Testbench
zamia	zamiaCAD in Version 0.10.3 inkl. Testbench

Tabelle B.1: Inhalt der Ordner auf dem Speichermedium

¹christian.trenkwalder@alumni.tuwien.ac.at

Tabellen

In den nachfolgenden Tabellen werden folgende Abkürzungen verwendet:

- **DL**: Deadlock
- **LD**: Late Detection
- **PF**: Premature Fire
- **e1/e2**: Effekt 1 oder Effekt 2, Gewichtung = 0,5
- **e1,e2**: Effekt 1 und Effekt 2, Gewichtung = 2

C.1 Qualitative Analyse

Signal	Stuck-At 0				Stuck-At 1			
	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$
OR.Ein(andere).Lo		LD		LD, LD		LD		DL, DL
OR.Ein(andere).Hi		DL				LD		
OR.Ein(letztes).Lo	LD, LD		LD		DL, DL		LD	
OR.Ein(letztes).Hi			DL				LD	
OR.Aus(andere).Lo				LD				DL
OR.Aus(andere).Hi		DL				LD		
OR.Aus(letztes).Lo	PF				DL			
OR.Aus(letztes).Hi			DL				PF	
C.Ein(andere).Lo				LD				DL
C.Ein(andere).Hi		DL				PF		
C.Ein(letztes).Lo	PF				DL			
C.Ein(letztes).Hi			DL				PF	
C.Aus.Lo	PF	LD			DL	PF		
C.Aus.Hi			DL	PF			PF	LD
Summe[DL,LD,PF]	0,3,2	3,2,0	4,1,0	0,4,1	5,0,0	0,3,2	0,2,3	4,1,0

Tabelle C.1: Qualitative Analyse eines NCL CD

Signal	Stuck-At 0				Stuck-At 1			
	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$
XOR.Ein(andere).Lo		LD		LD		DL		DL
XOR.Ein(andere).Hi		DL		DL		LD		LD
XOR.Ein(letztes).Lo	LD		LD		DL		DL	
XOR.Ein(letztes).Hi	DL		DL		LD		LD	
XOR.Aus(andere).Lo				LD				DL
XOR.Aus(andere).Hi		DL				LD		
XOR.Aus(letztes).Lo	PF				DL			
XOR.Aus(letztes).Hi			DL				PF	
C.Ein(andere).Lo				LD				DL
C.Ein(andere).Hi		DL				LD		
C.Ein(letztes).Lo	PF				DL			
C.Ein(letztes).Hi			DL				PF	
C.Aus.Lo	PF/LD	LD			LD	PF		
C.Aus.Hi			DL	PF			PF	DL/PF
Summe[DL,LD,PF]	1,1,5,2,5	3,2,0	4,1,0	1,3,1	3,2,0	1,3,1	1,1,3	3,5,1,0,5

Tabelle C.2: Qualitative Analyse eines LEDR CD

Signal	Stuck-At 0				Stuck-At 1			
	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$
XOR.Ein(andere).Lo	LD		LD		LD		LD	
XOR.Ein(andere).Hi	DL		DL		DL		DL	
XOR.Ein(letztes).Lo		LD		LD		DL		DL
XOR.Ein(letztes).Hi		DL		DL		LD		LD
XOR.Aus(andere).Lo				LD				DL
XOR.Aus(andere).Hi		DL				LD		
XOR.Aus(letztes).Lo	PF				DL			
XOR.Aus(letztes).Hi			DL				PF	
NOR.Ein(andere).Lo				LD				LD
NOR.Ein(andere).Hi		DL				LD		
NOR.Ein(letztes).Lo	PF				DL			
NOR.Ein(letztes).Hi			DL				DL	
NOR.Aus.Lo	LD		LD	LD	PF		DL	PF
NOR.Aus.Hi		DL				LD		
AND.Ein(andere).Lo				LD				LD
AND.Ein(andere).Hi		DL				DL		
AND.Ein(letztes).Lo	DL				DL			
AND.Ein(letztes).Hi			DL				PF	
AND.Aus.Lo	LD	LD	LD		DL	PF	PF	
AND.Aus.Hi				DL				LD
RSL.Set.Lo	LD	LD	LD		DL	PF	PF	
RSL.Set.Hi				DL				LD
RSL.Reset.Lo	LD		LD	LD	PF		DL	PF
RSL.Reset.Hi		DL				LD		
RSL.Aus.Lo		LD	LD			DL	PF	
RSL.Aus.Hi	PF			DL	LD			LD
Summe[DL,LD,PF]	2,5,3	6,4,0	4,6,0	4,6,0	6,2,2	3,5,2	4,1,5	2,6,2

Tabelle C.3: Qualitative Analyse eines CAL CD

Signal	Stuck-At 0				Stuck-At 1			
	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$
C.Ein(LCD).Lo			DL	PF			PF	LD
C.Ein(RCD).Lo	LD			LD	LD			DL
C.Ein(RACK).Lo	LD	DL			DL	DL		
C.Ein(LCD).Hi	DL	LD			PF	PF		
C.Ein(RCD).Hi		DL	LD			LD	DL	
C.Ein(RACK).Hi			DL	DL			LD	DL
C.Aus.Lo		PF	DL			DL	PF	
C.Aus.Hi	PF			LD	LD			LD
Summe[DL,LD,PF]	1,2,1	2,1,1	3,1,0	2,1,1	3,0,1	2,1,1	1,1,2	2,2,0

Tabelle C.4: Qualitative Analyse eines NCL/LEDR CTRL

Signal	Stuck-At 0				Stuck-At 1			
	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$
AND3a.Ein(LCD).Lo		LD		LD		LD		LD
AND3a.Ein(LCD).Hi		LD				LD		
AND3a.Ein(RCD).Lo	LD		LD		LD		LD	LD
AND3a.Ein(RCD).Hi			LD				LD	
AND3a.Ein(RACK).Lo				PF				PF
AND3a.Ein(RACK).Hi		LD				LD		
AND3a.Aus(PH0).Lo	LD		LD	PF	PF		DL	PF
AND3a.Aus(PH0).Hi		LD				DL		
AND3b.Ein(LCD).Lo				LD				
AND3b.Ein(LCD).Hi		LD				LD		
AND3b.Ein(RCD).Lo	LD				LD			LD
AND3b.Ein(RCD).Hi			LD				LD	
AND3b.Ein(RACK).Lo				PF				PF
AND3b.Ein(RACK).Hi		LD				LD		
AND3b.Aus(PH1).Lo	DL	DL	PF		DL	PF	PF	
AND3b.Aus(PH1).Hi				DL				DL
OR.Ein(PH0).Lo	LD		LD	PF	PF		DL	PF
OR.Ein(PH0).Hi		LD				DL		
OR.Ein(PH1).Lo	DL	DL	PF		DL	PF	PF	
OR.Ein(PH1).Hi				DL				DL
OR.Aus.Lo	PF		LD		PF		PF	
OR.Aus.Hi		PF		LD		LD		LD
Summe[DL,LD,PF]	2,4,1	2,6,1	0,5,2	2,3,4	2,2,3	2,5,2	2,2,3	2,3,3

Tabelle C.5: Qualitative Analyse eines CAL CTRL

Signal	Stuck-At 0				Stuck-At 1			
	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$
C.Ein(En).Lo		2	1			2	2	
C.Ein(En).Hi	2			1	1			2
C.Ein(D).Lo	2,2	2,2	2	2	1,2	2,2	2	2
C.Ein(D).Hi			3	2			1	2
C.Aus.Lo	2,1	2,1	2	2	2,3	2,2	2	2
C.Aus.Hi			3	2			1	2
Summe[DL,LD,PF]	1,4,0	2,3,0	1,2,2	1,4,0	2,2,1	0,5,0	2,3,0	0,5,0

Tabelle C.6: Qualitative Analyse eines NCL Registers

Signal	Stuck-At 0				Stuck-At 1			
	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$
DETFE.D.Lo	2	2	2	2	3	2	3	3
DETFE.D.Hi	3	2	2	3	2	3	2	2
DETFE.En.Lo	1			2	3			3
DETFE.En.Hi		3	3			2	1	
DETFE.Q.Lo	2	3	3	2	3	2	3	3
DETFE.Q.Hi	3	2	3	3	2	3	2	2
Summe[DL,LD,PF]	1,2,2	0,3,2	0,2,3	0,3,2	0,2,3	0,3,2	1,2,2	0,3,2

Tabelle C.7: Qualitative Analyse eines LEDR Registers

Signal	Stuck-At 0				Stuck-At 1			
	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$	$Tr-\varphi_0$	$St-\varphi_0$	$Tr-\varphi_1$	$St-\varphi_1$
DLatch.D.Lo	3	2	2	3	2	2	2	2
DLatch.D.Hi	2	2	2	3	3	3	3	3
DLatch.En.Lo	3		3		3		3	
DLatch.En.Hi		1		1		3		3
DLatch.Q.Lo	2	3	3	2	2	3	2	2
DLatch.Q.Hi	2	3	2	2	3	2	3	2
Summe[DL,LD,PF]	0,3,2	1,2,2	0,3,2	1,2,2	0,2,3	0,3,2	0,2,3	0,3,2

Tabelle C.8: Modell eines CAL Registers, Deadlock = 1, Late Detection = 2, Illegal = 4, Premature Fire = 3

Dokumentation der Simulation

Dieser Anhang enthält eine detaillierte Aufzählung aller vom Tester änderbaren Variablen der Simulationsumgebung. Die Angabe des Datentyp erfolgt dabei in den Standardbibliotheken(ieee.¹) definierten Datentypen.

D.1 Dateien

Dateiname	Beschreibung
boot.py	Python Skript zum Initialisieren der zamiaCAD API
runner_<name>.py	Python Skript zur automatisierten Simulation der <name> Testbenches
tsl_signals.vhdl	Definition der Dual Rail Signale ²
read_file.vhdl	Logik zum Auslesen von Dateien ³
txt_util.vhdl	Hilfsfunktionen zum arbeiten mit String ⁴
comb_ckt.vhdl	Definition aller verwendeten Logik Bausteine
tb_<tech>_<name>.vhdl	Testbench für die Schaltung <name> und mit Verwendung von Technologie <tech>

Tabelle D.1: Beschreibung der Dateien im Projektverzeichnis

¹<http://www.eda.org/rassp/vhdl/models/standards/>

²Scott C. Smith http://www.ndsu.edu/pubweb/~scotsmit/CCLI_async.html

³Basierend auf Stefan Dolls Tutorial http://www.stefanvhdl.com/vhdl/html/file_read.html

⁴TU Darmstadt http://www.iss.tu-darmstadt.de/student_area/vhdl/uart_example/txt_util.vhd

D.2 Variablen

1. base

Datentyp Array of Positive

Einheit ps

in Datei runner_.py

Beschreibung Minimales Delay auf der Datenleitung, die Länge des Array entspricht dem doppelten der Gesamtanzahl an Rails. Wobei die erste Hälfte Phase0 und die Zweite Phase1 entspricht.

2. range

Datentyp Array of Natural

Einheit ps

in Datei runner_.py

Beschreibung Wertebereich des Delay auf der Datenleitung, die Länge des Array entspricht dem doppelten der Gesamtanzahl an Rails. Wobei die erste Hälfte Phase0 und die Zweite Phase1 entspricht.

3. simnum

Datentyp Positive

Einheit 1

in Datei runner_.py

Beschreibung Dient der Auswahl aus der in der Datei „BuildPath.txt“definierten Toplevel Einträge.

4. tech

Datentyp String

in Datei runner_.py

Beschreibung Dient zur Auswahl von vordefinierten Werten für Δt_{Source} , Δt_{RCD} und Δt_{Sink} . Zulässig sind „ncl“, „ledr“, „cal“ und „fixval“.

5. fixval

Datentyp Array of Positive

Einheit ps

in Datei runner_.py

Beschreibung Setzen von vordefinierten Werten für das Delay an den Eingängen von GEN-CTRL.

6. **step****Datentyp** Positive**Einheit** ps**in Datei** runner_.py**Beschreibung** Setzen der Schrittweite in der Simulation. Darf die Hälfte der kleinsten Laufzeit in der Schaltung nicht überschreiten. Siehe auch Nyquist-Shannon-Abtasttheorem.7. **resetp****Datentyp** Positive**Einheit** ps**in Datei** runner_.py**Beschreibung** Länge der Reset Phase vor der Simulation.8. **VV****Datentyp** std_logic_vector**in Datei** tb_.vhdl**Beschreibung** 1-Hot Bus für das injizieren von Stuck-At Faults. Die Länge des Vektors ergibt sich aus der doppelten Anzahl an vorhandenen E/A in der Schaltung. z.B OR2 welches 2 Eingänge und 1 Ausgang besitzt hat dieser eine Breite von 6. Die Anordnung ergibt sich aus der Reihenfolge $VV_{OR2} = EEAE EA$ wobei die ersten 3 Bit für ein Stuck-At 0 stehen und die letzten 3 für ein Stuck-At 1. Sollte eine Komponente aus mehreren Gattern bestehen, so ergibt sich der Vektor durch Aneinanderreihung der Vektoren einzelner Komponenten. Für einen 2 Wort Completion Detektor für NCL bestehend aus 2 OR und einem C-Element ergibt sich ein 18 Bit breiter Vektor $VV_{CD} = VV_{OR2}VV_{OR2}VV_{C2}$.9. **placetobe****Datentyp** Positive**in Datei** runner_.py**Beschreibung** Bit des *std_logic_vector* VV welcher gesetzt werden soll.10. **stim_file****Datentyp** String**in Datei** tb_.vhdl**Beschreibung** Name der zu Verwendenden Datei, welche die zu verwendenden Codewörter enthält. Wird in der Komponente *FILE_READ* gesetzt und ist standardmäßig „stim.dat“.

11. **T_MAX_Y****Datentyp** Positive**in Datei** tb_.vhdl**Beschreibung** Breite der in *stim_file* gespeicherten Codewörter.12. **difftime****Datentyp** Positive**Einheit** ps**in Datei** tb_.vhdl und runner_.py**Beschreibung** Gibt die Laufzeit der Referenzimplementation an, und wird automatisch ermittelt.13. **durchlaufe****Datentyp** Positive**in Datei** runner_.py**Beschreibung** Anzahl der Wiederholungen einer Simulation. Wird standardmäßig automatisch ermittelt und ergibt sich aus $durchlaufe = \frac{difftime}{step}$ 14. **faulttype****Datentyp** Positive**in Datei** tb_.vhdl und runner_.py**Beschreibung** Enthält den von der Testbench festgestellten Fehlertyp, wobei 0:Keiner, 1:Deadlock, 2:Late Detection und 3:Premature Fire.

D.3 Funktionen

1. **InitSeedValues(sim,range)****in Datei** runner_.py**@sim** Objekt eines Simulator**@range** Array mit Werten des Datentyps Positive**Beschreibung** Setzt innerhalb eines Wertebereiches *@range* zufällige Seed Werte zur Generierung von Zufallswerten für die Delays der Testbench.2. **InitRangeValues(sim,range)****in Datei** runner_.py**@sim** Objekt eines Simulator

@range Array mit Werten des Datentyps Positive

Beschreibung Definiert den Wertebereich *@range* in welchem die Testbench zufällige Delays berechnet.

3. **InitBaseValues(sim,base)**

in Datei runner_.py

@sim Objekt eines Simulator

@base Array mit Werten des Datentyps Positive

Beschreibung Definiert die Basiswerte *@base* für die Delays der Testbench.

Literaturverzeichnis

- [AKLO09] ABRAMOV, B.B. ; KEREN, O. ; LEVIN, I.S. ; OSTROVSKII, V.I.: Constructing self-testing circuits with the use of step-by-step (cascade) control. In: *Automation and Remote Control* 70 (2009), 1217-1227. – DOI 10.1134/S0005117909070121. – ISSN 0005-1179
- [ALRTCS01] AVIŽIENIS, A. ; LAPRIE, J.C. ; RANDELL, B. ; TYNE. COMPUTING SCIENCE, University of Newcastle u.: *Fundamental Concepts of Dependability*. University of Newcastle upon Tyne, Computing Science, 2001 (Technical report series)
- [ALV07] AKGUN, O.C. ; LEBLEBICI, Y. ; VITTOZ, E.A.: Current sensing completion detection for subthreshold asynchronous circuits. In: *Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on*, 2007, S. 376-379
- [And71] ANDERSON, Douglas A.: *Design of self-checking digital networks using coding techniques*. Dissertation. Champaign, IL, USA, University of Illinois at Urbana-Champaign, 1971. – AAI7212065
- [Bes90] BEST, E.: *Characterisation of home states in free choice systems*. Hochschule, 1990 (Hildesheimer Informatikberichte)
- [BK05] BIEGANOWSKI, J. ; KARATKEVICH, A.: Heuristics for Thelen's prime implicant method. In: *Schedae Informaticae* Vol. 14 (2005), S. 125-135. – Jagiellonian University Press, Kraków
- [BM92] BARAKAOU, Kamel ; MINOUX, Michel: A Polynomial-Time Graph Algorithm to Decide Liveness of Some Basic Classes of Bounded Petri Nets. In: JENSEN, Kurt (Hrsg.): *Application and Theory of Petri Nets* Bd. 616, Springer, 1992 (Lecture Notes in Computer Science). – ISBN 3-540-55676-1, 62-75
- [BOF10] BEEREL, P.A. ; OZDAG, R.O. ; FERRETTI, M.: *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, 2010 (A Designer's Guide to Asynchronous VLSI). – 204- S. – ISBN 9780521872447
- [BS09] BAINBRIDGE, W. J. ; SALISBURY, S. J.: Glitch Sensitivity and Defense of Quasi Delay-Insensitive Network-on-Chip Links. In: *Symposium on Asynchronous Circuits and Systems*, 2009, S. 35-44

- [Che98] CHENG, Fu-Chiung: Practical design and performance evaluation of completion detection circuits. In: *ICCD*, 1998, 354-359
- [Chu03] CHUNG, Wai M.: *The Usage of Dual Edge Triggered Flip-flops in Low Power, Low Voltage Applications*, University of Waterloo, Diplomarbeit, 2003
- [CS68] CARTER, William C. ; SCHNEIDER, Peter R.: Design of dynamically checked computers. In: *IFIP Congress (2)*, 1968, S. 878–883
- [DBL12] *2012 IEEE International Symposium on Circuits and Systems, ISCAS 2012, Seoul, Korea (South), May 20-23, 2012*. IEEE, 2012 . – ISBN 978–1–4673–0218–0
- [Del05] DELVAI, Martin: *Design of an Asynchronous Processor Based on Code Alternation Logic - Treatment of Non-Linear Data Paths*. Dissertation. Treitlstr. 3/3/182-1, 1040 Vienna, Austria, Technische Universität Wien, Institut für Technische Informatik, 2005
- [DF96] DONALDSON, Val ; FERRANTE, Jeanne: Determining Asynchronous Acyclic Pipeline Execution Times. In: *Proceedings of the 10th International Parallel Processing Symposium*. Washington, DC, USA : IEEE Computer Society, 1996 (IPPS '96). – ISBN 0–8186–7255–2, 568–572
- [DFH⁺05] DELVAI, Martin ; FUCHS, Gottfried ; HANDL, Thomas ; HUBER, Wolfgang ; STEININGER, Andreas: Design of an Asynchronous Microprocessor with Four-State Logic. In: *Austrochip 2005*, 2005, S. 105–112. – talk: Austrochip, Wien; 2005-10-06
- [DGY92] DAVID, Ilana ; GINOSAR, Ran ; YOELI, Michael: An Efficient Implementation of Boolean Functions as Self-Timed Circuits. In: *IEEE Trans. Comput.* 41 (1992), Januar, Nr. 1, 2–11. – DOI 10.1109/12.123377. – ISSN 0018–9340
- [DGY95] DAVID, Ilana ; GINOSAR, Ran ; YOELI, Michael: Self-timed is self-checking. In: *J. Electron. Test.* 6 (1995), April, Nr. 2, 219–228. – DOI 10.1007/BF00993088. – ISSN 0923–8174
- [DHS05] DELVAI, Martin ; HUBER, Wolfgang ; STEININGER, Andreas: The Limitations of Delay Insensitive Asynchronous Pipeline, Implementation at the Example of Four State Logic. In: *Austrochip 2005*, 2005
- [DLV05] DI, J. ; LALA, P.K. ; VASUDEVAN, D.: On the Effect of Stuck-at Faults on Delay-insensitive Nanoscale Circuits. In: *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems 0* (2005), 371-379. ISBN 0–7695–2464–8

- [DMKG10] DOBKIN, R. ; MOYAL, M. ; KOLODNY, A ; GINOSAR, R.: Asynchronous Current Mode Serial Communication. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 18 (2010), July, Nr. 7, S. 1107–1117. – DOI 10.1109/TVLSI.2009.2020859. – ISSN 1063–8210
- [DWD91] DEAN, Mark E. ; WILLIAMS, Ted E. ; DILL, David L.: Efficient self-timing with level-encoded 2-phase dual-rail (LEDR). In: *Proceedings of the 1991 University of California/Santa Cruz conference on Advanced research in VLSI*. Cambridge, MA, USA : MIT Press, 1991. – ISBN 0–262–19308–6, 55–70
- [EB97] EBERGEN, Jo ; BERKS, Robert: Response Time Properties of Some Asynchronous Circuits. In: *Proceedings of the 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems*. Washington, DC, USA : IEEE Computer Society, 1997 (ASYNC '97). – ISBN 0–8186–7922–0, 76–
- [EFS07] EBERGEN, Jo ; FURBER, Steve ; SAIFHASHEMI, Arash: Notes On Pulse Signaling. In: *Proceedings of the 13th IEEE International Symposium on Asynchronous Circuits and Systems*. Washington, DC, USA : IEEE Computer Society, 2007 (ASYNC '07). – ISBN 0–7695–2771–X, 15–24
- [FB96a] FANT, K. M. ; BRANDT, S. A.: NULL Convention Logic/sup TM/: A Complete And Consistent Logic For Asynchronous Digital Circuit Synthesis. In: *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*. Washington, DC, USA : IEEE Computer Society, 1996 (ASAP '96). – ISBN 0–8186–7542–X, 261–
- [FB96b] FANT, K. M. ; BRANDT, S. A.: NULL Convention Logic/sup TM/: A Complete And Consistent Logic For Asynchronous Digital Circuit Synthesis. In: *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*. Washington, DC, USA : IEEE Computer Society, 1996 (ASAP '96). – ISBN 0–8186–7542–X, 261–
- [FB02] FERRETTI, M. ; BEEREL, P.: Single-Track Asynchronous Pipeline Templates Using 1-of-N Encoding. In: *Proceedings of the conference on Design, automation and test in Europe*. Washington, DC, USA : IEEE Computer Society, 2002 (DATE '02). – ISBN 0–7695–1471–5, 1008–
- [FD96] FURBER, S.B. ; DAY, P.: Four-phase micropipeline latch control circuits. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 4 (1996), June, Nr. 2, S. 247–253. – DOI 10.1109/92.502196. – ISSN 1063–8210
- [Fer06] FERRINGER, Markus: *An Asynchronous Hardware Design for Distributed Tick Generation*. Treitlstr. 3/3/182-1, 1040 Vienna, Austria, Technische Universität Wien, Institut für Technische Informatik, Diplomarbeit, 2006
- [FF14] FRISCHENSCHLAGER, Albin ; FRÜHWIRTH, Thomas: Classification Of Static Asynchronous Pipelines. – Forschungsbericht. 2014

- [Fri12] FRIESENBICHLER, Werner: *Effects and Mitigation of Transient Faults in Quasi Delay-Insensitive Logic*. Dissertation. Treitlstr. 3/3/182-1, 1040 Vienna, Austria, Technische Universität Wien, Institut für Technische Informatik, 2012
- [GM88] GOODMAN, R.M. ; MCAULEY, A.J.: An efficient asynchronous multiplier. In: *Systolic Arrays, 1988., Proceedings of the International Conference on*, 1988, S. 593–599
- [GNR61] GALEY, J. M. ; NORBY, R. E. ; ROTH, J.P.: Techniques for the diagnosis of switching circuit failures. In: *Switching Circuit Theory and Logical Design, 1961. SWCT 1961. Proceedings of the Second Annual Symposium on*, 1961, S. 152–160
- [Hau95] HAUCK, Scott: Asynchronous Design Methodologies: An Overview. In: *PROCEEDINGS OF THE IEEE*, IEEE, 1995, 69–93
- [HL00] HARDEN, James J. ; LINDER, Daniel H.: LEDR Phased Logic Gate Primitives. (2000), 22
- [HSD04] HUBER, Wolfgang ; STEININGER, Andreas ; DELVAI, Martin: Delay Insensitive Asynchronous Pipeline Implementation for Code Alternation Logic. – Forschungsbericht / Technische Universität Wien, Institut für Technische Informatik. Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2004 (85/2004)
- [JCL02] JIAO, Li ; CHEUNG, To-Yat ; LU, Weiming: Characterizing Liveness of Petri Nets in Terms of Siphons. In: ESPARZA, Javier (Hrsg.) ; LAKOS, Charles (Hrsg.): *ICATPN Bd. 2360*, Springer, 2002 (Lecture Notes in Computer Science). – ISBN 3–540–43787–8, 203-216
- [JGJ⁺07] JOSHI, M. V. ; GOSAVI, S. ; JEGADEESAN, V. ; BASU, A ; JAISWAL, S. ; AL-ASSADI, W. K. ; SMITH, S.C.: NCL Implementation of Dual-Rail 2S Complement 8x8 Booth2 Multiplier using Static and Semi-Static Primitives. In: *Region 5 Technical Conference, 2007 IEEE*, 2007, S. 59–64
- [JL09] JEITLER, Marcus ; LECHNER, Jakob: Towards Comparing the Robustness of Synchronous and Asynchronous Circuits by Fault Injection. In: *MEMICS*, 2009, S. 8
- [JN08] JEONG, Cheoljoo ; NOWICK, Steven M.: Block-Level Relaxation for Timing-Robust Asynchronous Circuits Based on Eager Evaluation. In: *2012 IEEE 18th International Symposium on Asynchronous Circuits and Systems 0* (2008), S. 95–104. – DOI <http://doi.ieeecomputersociety.org/10.1109/ASYNC.2008.25>. – ISSN 1522–8681
- [Joh89] JOHNSON, B.W.: *Design and Analysis of Fault-tolerant Digital Systems*. Addison-Wesley Publishing Company, 1989 (Addison-Wesley series in electrical and computer engineering). – ISBN 9780201075700

- [JR12] JAYANTHI, D. ; RAJARAM, M.: A Quasi Delay Insensitive Reduced Stack Pre-Charged Half Buffer based High Speed Adder using pipeline templates for Asynchronous Circuits. In: *Journal of Computer Science* 8 (7): 1114-1122, 2012 ISSN 1549-3636 (2012)
- [KAGEBR82] KUNG, Sun-Yuan ; ARUN, K. S. ; GAL-EZER, R.J. ; BHASKAR RAO, D.V.: Wavefront Array Processor: Language, Architecture, and Applications. In: *Computers, IEEE Transactions on* C-31 (1982), Nov, Nr. 11, S. 1054–1066. – DOI 10.1109/TC.1982.1675922. – ISSN 0018–9340
- [KNR⁺02] KONDRATYEV, A. ; NEUKOM, L. ; ROIG, O. ; TAUBIN, A. ; FANT, K.: Checking Delay-Insensitivity: 104 Gates and Beyond. In: *ASYNC*, 2002, 149
- [Lal12] LALA, P.: Transient and Permanent Fault Injection in VHDL Description of Digital Circuits. In: *Circuits and Systems* (2012)
- [LHM10] LAFRIEDA, Christopher ; HILL, Benjamin ; MANOHAR, Rajit: An Asynchronous FPGA with Two-Phase Enable-Scaled Routing, IEEE Computer Society, 2010. – ISBN 978–0–7695–4032–0, 141-150
- [LM04] LAFRIEDA, Christopher ; MANOHAR, Rajit: Fault Detection and Isolation Techniques for Quasi Delay-Insensitive Circuits. In: *2004 International Conference on Dependable Systems and Networks (DSN 2004), 28 June - 1 July 2004, Florence, Italy, Proceedings*, IEEE Computer Society, 2004. – ISBN 0–7695–2052–9, 41-50
- [LOKS06] LEVIN, Ilya ; OSTROVSKY, Vladimir ; KEREN, Osnat ; SINELNIKOV, Vladimir: Cascade Scheme for Concurrent Errors Detection. In: *Proceedings of the 9th EUROMICRO Conference on Digital System Design*. Washington, DC, USA : IEEE Computer Society, 2006 (DSD '06). – ISBN 0–7695–2609–8, 359–368
- [MAMN08] MCGEE, Peggy B. ; AGYEKUM, Melinda Y. ; MOHAMED, Moustafa A. ; NOWICK, Steven M.: A level-encoded transition signaling protocol for high-throughput global communication. In: *14th IEEE International Symposium on Asynchronous Circuits and Systems*, 2008, 116 – 127
- [Mar86] MARTIN, Alain J.: Compiling Communicating Processes into Delay-Insensitive VLSI Circuits. – Forschungsbericht. Pasadena, CA, USA : California Institute of Technology, 1986
- [Mar90] MARTIN, Alain J.: The Limitations to Delay-insensitivity in Asynchronous Circuits. In: *Proceedings of the Sixth MIT Conference on Advanced Research in VLSI*. Cambridge, MA, USA : MIT Press, 1990 (AUSCRYPT '90). – ISBN 0–262–04109–X, 263–278
- [MBAB99] MAJUMDER, S. ; BHATTACHARYA, B.B. ; AGRAWAL, V.D. ; BUSHNELL, M.L.: A complete characterization of path delay faults through stuck-at faults. In: *VLSI*

- Design, 1999. Proceedings. Twelfth International Conference On, 1999.* – ISSN 1063–9667, S. 492–497
- [MMN07] MITRA, Amitava ; MCLAUGHLIN, William F. ; NOWICK, Steven M.: Efficient Asynchronous Protocol Converters for Two-Phase Delay-Insensitive Global Communication. In: *13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC 2007), 12-14 March 2006, Berkeley, California, USA*, IEEE Computer Society, 2007. – ISBN 978–0–7695–2771–0, 186-195
- [MRL07] MONNET, Y. ; RENAUDIN, M. ; LEVEUGLE, R.: Formal Analysis of Quasi Delay Insensitive Circuits Behavior in the Presence of SEUs. In: *On-Line Testing Symposium, 2007. IOLTS 07. 13th IEEE International, 2007*, S. 113–120
- [Mul59] MULLER, W. S. D. E.; Bartky B. D. E.; Bartky: A Theory of Asynchronous Circuits, Harvard Univ. Press, 1959, S. 204–243
- [Nan98] NANYA, Takashi: Asynchronous VLSI System Design. In: *ASP-DAC'98 Tutorials*. Yokohama, 1998, 21
- [NQA09] NUNEZ, J. ; QUINTANA, J.M. ; AVEDILLO, M.J.: Fast and area efficient multi-input Muller C-element based on MOS-NDR. In: *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on, 2009*, S. 1811–1814
- [OB02] OZDAG, R.O. ; BEEREL, P.A.: High-speed QDI asynchronous pipelines. In: *Asynchronous Circuits and Systems, 2002. Proceedings. Eighth International Symposium on, 2002.* – ISSN 1522–8681, S. 13–22
- [OL05] OSTROVSKY, Vladimir ; LEVIN, Ilya: Implementation of Concurrent Checking Circuits by Independent Sub-circuits. In: *Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. Washington, DC, USA : IEEE Computer Society, 2005 (DFT '05). – ISBN 0–7695–2464–8, 343–351
- [PA91] PHILLIPS, Benny P. ; AFB, Tinker: On computing the detection probability of stuck-at faults in a combinational circuit. In: *AUTOTESTCON, International Automatic Testing Conference, 1991*
- [Pet77] PETERSON, James L.: Petri Nets. In: *ACM Comput. Surv.* 9 (1977), September, Nr. 3, 223–252. – DOI 10.1145/356698.356702. – ISSN 0360–0300
- [Pet81] PETERSON, James L.: *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1981. – ISBN 0136619835
- [PM05] PENG, Song ; MANOHAR, Rajit: Efficient failure detection in pipelined asynchronous circuits. In: *In Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005.* – Fig 2b

- [PN95] PIESTRAK, S. J. ; NANYA, T.: Towards Totally Self-Checking Delay-Insensitive Systems. In: *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*. Washington, DC, USA : IEEE Computer Society, 1995 (FTCS '95), 228–
- [PT05] PHURAT, Tawan ; THONGTAK, Arthit: Time Petri-Net based Stuck-at Fault Model for Asynchronous Circuits Testing. (2005)
- [PU98] PLANA, Luis A. ; UNGER, Stephen H.: Pulse-mode macromodular systems. In: *ICCD*, 1998, S. 348–353
- [Rot92] ROTH, Charles H.: *Fundamentals of Logic Design*. 4th. Boston, MA, USA : PWS Publishing Co., 1992. – ISBN 0534954731
- [Sch09] SCHAT, Jan: On the relationship between stuck-at fault coverage and transition fault coverage. In: *DATE*, 2009, S. 1218–1221
- [SDH04] STEININGER, Andreas ; DELVAI, Martin ; HUBER, Wolfgang: Code Alternation Logic (CAL): A Novel Efficient Design Approach for Delay-Insensitive Asynchronous Circuits. – Forschungsbericht / Technische Universität Wien, Institut für Technische Informatik. Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2004 (87/2004)
- [SDY⁺04] SMITH, S. C. ; DEMARA, R. F. ; YUAN, J. S. ; FERGUSON, D. ; LAMB, D.: Optimization of NULL convention self-timed circuits. In: *Integr. VLSI J.* 37 (2004), August, Nr. 3, 135–165. – DOI 10.1016/j.vlsi.2003.12.004. – ISSN 0167–9260
- [SEE96] SHAMS, Maitham ; EBERGEN, Jo C. ; ELMASRY, Mohamed I.: A comparison of CMOS implementations of an asynchronous circuits primitive: the C-element. In: HOROWITZ, Mark (Hrsg.) ; RABAEY, Jan M. (Hrsg.) ; BARTON, Brock (Hrsg.) ; PEDRAM, Massoud (Hrsg.): *ISLPED*, IEEE, 1996. – ISBN 0–7803–3571–6, 93-96
- [SF98] SOBELMAN, G.E. ; FANT, K.: CMOS circuit design of threshold gates with hysteresis. In: *Proceedings of the 1998 IEEE International Symposium* (1998), 61-64
- [SF10] SPARS, Jens ; FURBER, Steve: *Principles of Asynchronous Circuit Design: A Systems Perspective*. 1st. Springer Publishing Company, Incorporated, 2010. – ISBN 1441949364, 9781441949363
- [Smi02] SMITH, Scott C.: Speedup of Self-Timed Digital Systems Using Early Completion”, The. In: *IEEE Computer Society Annual Symposium on VLSI*, 2002, S. 107–113

- [Smi06] SMITH, S. C.: Speedup of NULL Convention Digital Circuits Using NULL Cycle Reduction. In: *J. Syst. Archit.* 52 (2006), Juli, Nr. 7, S. 411–422. – ISSN 1383–7621
- [SN01a] SINGH, M. ; NOWICK, S.M.: MOUSETRAP: ultra-high-speed transition-signaling asynchronous pipelines. In: *Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference on*, 2001. – ISSN 1063–6404, S. 9–17
- [SN01b] SRETASEREEKUL, N. ; NANYA, T.: Eliminating isochronic-fork constraints in quasi-delay-insensitive circuits. In: *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific*, 2001, S. 437–442
- [SN07] SINGH, M. ; NOWICK, S.M.: MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 15 (2007), June, Nr. 6, S. 684–698. – DOI 10.1109/TVLSI.2007.898732. – ISSN 1063–8210
- [SNC99] STROLLO, A.G.M. ; NAPOLI, E. ; CIMINO, C.: Low power double edge-triggered flip-flop using one latch. In: *Electronics Letters* 35 (1999), Feb, Nr. 3, S. 187–188. – DOI 10.1049/el:19990164. – ISSN 0013–5194
- [Spa01] SPARSØ, J.: Asynchronous circuit design - A tutorial. Version: dec 2001. In: *Chapters 1-8 in "Principles of asynchronous circuit design - A systems Perspective"*. Boston / Dordrecht / London : Kluwer Academic Publishers, dec 2001, 1-152
- [SS93] SPARSØ, Jens ; STAUNSTRUP, Jørgen: Delay-insensitive multi-ring structures. In: *Integr. VLSI J.* 15 (1993), Oktober, Nr. 3, 313–340. – DOI 10.1016/0167–9260(93)90035–B. – ISSN 0167–9260
- [SSDS92] SPARSO, J. ; STAUNSTRUP, J. ; DANTZER-SORENSEN, M.: Design of delay insensitive circuits using multi-ring structures. In: *Design Automation Conference, 1992., EURO-VHDL '92, EURO-DAC '92. European*, 1992, S. 15–20
- [SSH99a] SUTHERLAND, I.E. ; SPROULL, R.F. ; HARRIS, D.F.: *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers, 1999 (The Morgan Kaufmann Series in Computer Architecture and Design Series). – ISBN 9781558605572
- [SSH99b] SUTHERLAND, Ivan ; SPROULL, Bob ; HARRIS, David: *Logical Effort: Designing Fast CMOS Circuits*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999. – ISBN 1–55860–557–6
- [Sut89] SUTHERLAND, Ivan E.: Micropipelines. In: *Commun. ACM* 32 (1989), Nr. 6, 720-738
- [The81] THELEN, B.: *Untersuchungen von Algorithmen für den rechnergestützten logischen Entwurf digitaler Baugruppen*. 1981

- [TYP11] TYANEV, Dimitar S. ; YANEV, Dragomir V. ; POPOVA, Stefka I.: Non-linear asynchronous micro-pipelines. In: *Proceedings of the 12th International Conference on Computer Systems and Technologies*. New York, NY, USA : ACM, 2011 (CompSysTech '11). – ISBN 978-1-4503-0917-2, 38-44
- [Udd86] UDDING, JanTijmen: A formal model for defining and classifying delay-insensitive circuits and systems. In: *Distributed Computing* 1 (1986), Nr. 4, 197-204. – DOI 10.1007/BF01660032. – ISSN 0178-2770
- [Vaa06] VAAJE, Audhild: Theorems for Fault Collapsing in Combinational Circuits. In: *J. Electron. Test.* 22 (2006), Februar, Nr. 1, 23-36. – DOI 10.1007/s10836-006-6222-1. – ISSN 0923-8174
- [Wil90] WILLIAMS, Ted: Latency and throughput tradeoffs in self-timed speed-independent pipelines and rings. – Forschungsbericht / Stanford University. Version: 1990. Stanford, CA, USA, 1990
- [WKB04] WEGRZYN, A. ; KARATKEVICH, A. ; BIEGANOWSKI, J.: Detection of deadlocks and traps in Petri nets by means of Thelen's prime implicant method. In: *International Journal of Applied Mathematics and Computer Science* Vol. 14 (2004), Nr. no 1, S. 113-121
- [WV93] WUU, T.-Y. ; VRUDHULA, S.B.K.: A design of a fast and area efficient multi-input Muller C-element. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 1 (1993), Nr. 2, S. 215-219. – DOI 10.1109/92.238414. – ISSN 1063-8210
- [XIHK12] XIA, Zhengfan ; ISHIHARA, Shota ; HARIYAMA, Masanori ; KAMEYAMA, Michitaka: Dual-rail/single-rail hybrid logic design for high-performance asynchronous circuit. In: *ISCAS[DBL12]*, S. 3017-3020
- [YKSK96] YAKOVLEV, A. V. ; KOELMANS, A. M. ; SEMENOV, A. ; KINNIMENT, D. J.: Modelling, Analysis and Synthesis of Asynchronous Control Circuits Using Petri Nets. In: *INTEGRATION: the VLSI Journal* 21 (1996), S. 143-170
- [YS10] YANCEY, S. ; SMITH, S.C.: A differential design for C-elements and NCL gates. In: *Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on*, 2010. – ISSN 1548-3746, S. 632-635

Glossary

- API** Application Programming Interface. 59, 98
- CAL** Code Alternation Logic. 18, 25, 49, 69, 98
- CD** Completion Detector. vi, 12, 17, 25, 28, 31, 98
- CSV** Comma-separated values. 60, 98
- DI** Delay Insensitive. 14, 98
- DIMS** Delay Insensitive Minterm Synthesis. vi, 20, 26, 98
- DUT** Device Under Test. 53, 98
- FIFO** First In First Out Pipeline. 11, 98
- FSL** Four State Logic. vi, 17, 18, 98
- LEDR** Level Encoded Dual Rail. 18, 25, 49, 69, 98
- MSAF** Multiple Stuck-At Fault. 25, 98
- NCL** Null Convention Logic. 16, 25, 31, 38, 49, 69, 98
- NCR** NULL Cycle Reduction. 16, 98
- NRZ** No Return To Zero. 15, 98
- PLA** Programmable Logic Array. vi, 21, 26, 38, 98
- PN** Petri Net. 98
- QDI** Quasi Delay Insensitive. 1, 14, 33, 66, 69, 98
- RTZ** Return To Zero. 14, 98

SSAF Single Stuck-At Fault. vi, 23, 25, 31, 38, 42, 47, 52, 56, 69, 98

STLM Self Timed Logic Module. vi, 19, 26, 38, 42, 98

TG Threshold Gate. vi, 6, 20, 26, 98

TPN Time Petri Net. 32, 98

TSL Three State Logic. 16, 98