

RESEARCH

Christoph Brandau

Modellierung und Transformation digitaler Schaltungen mittels Digital Circuit Petri Nets



Springer Vieweg

Modellierung und Transformation digitaler Schaltungen mittels Digital Circuit Petri Nets

Christoph Brandau

Modellierung und Transformation digitaler Schaltungen mittels Digital Circuit Petri Nets

Mit einem Geleitwort von Prof. Dr.-Ing. Dietmar Tutsch

 Springer Vieweg

Christoph Brandau
Fakultät für Elektrotechnik
Informationstechnik und Medientechnik
Bergische Universität Wuppertal
Wuppertal, Deutschland

Zugl.: Dissertation, Bergische Universität Wuppertal, 2018

ISBN 978-3-658-25243-4 ISBN 978-3-658-25244-1 (eBook)
<https://doi.org/10.1007/978-3-658-25244-1>

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Springer Vieweg

© Springer Fachmedien Wiesbaden GmbH, ein Teil von Springer Nature 2019

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag, noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen. Der Verlag bleibt im Hinblick auf geografische Zuordnungen und Gebietsbezeichnungen in veröffentlichten Karten und Institutionsadressen neutral.

Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature

Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

für Jenny

Geleitwort

Seit der Vorstellung der sogenannten Petri-Netze durch Carl Adam Petri zur Modellierung von Nebenläufigkeit im Jahr 1962 wurde diese Art der Stellen-Transitions-Netze kontinuierlich zur Steigerung des Modellierkomforts aber auch der Modelliermächtigkeit erweitert. Neben der Einführung von zeitbehafteten und stochastischen Petri-Netzen waren dies auch High-Level-Petri-Netze mit ihren teils objektorientierten Strukturen. Waren zu Beginn hauptsächlich die Abläufe bei technischen Prozessen und Computersystemen im Fokus der Modellierung, so dehnte sich der Anwendungsbereich später auch auf davon weiter entfernte Themen wie Prozesse in der Geschäftswelt aus. In Konsequenz wurde ein auf Petri-Netze basierendes Konzept auch für die Unified Modeling Language (UML) übernommen und im Rahmen des Aktivitätsdiagramms als eines der Verhaltensdiagramme realisiert.

Der Erfolg der Petri-Netze ist insbesondere durch die übersichtliche graphische Beschreibung von Nebenläufigkeit begründet. Insofern lag die Idee von Herrn Brandau nahe, eine solche Modellierungsart auch für digitale Schaltungen zu verwenden, in denen der Stromfluss und damit der Logikfluss auch nebenläufig stattfindet. Bisherige Modellierungsarten wie beispielsweise die Hardware-Beschreibungssprachen VHDL oder Verilog HDL erfordern fundierte Kenntnisse von deren Syntax. Zieht man diese Sprachen zum Entwurf von digitalen Schaltungen heran, so ist zusätzlich eine weitreichende Erfahrung im Umgang mit deren Sprachkonstrukten unabdingbar um synthesefähigen Code zu erzeugen. Die in diesem Buch vorliegende Arbeit von Herrn Brandau schaltet mit den Petri-Netzen eine Stufe der Modellierung vor diese Hardware-Beschreibungssprachen, so dass der Entwickler sich nicht um die Eigenheiten dieser Sprachen kümmern muss. Mit den hier vorgestellten Erweiterungen der Petri-Netz-Nomenklatur können digitale Schaltungen direkt auf der graphischen Petri-Netz-Ebene beschrieben werden. Dies betrifft nicht nur Schaltnetze, auch Schaltwerke mit ihrem speichernden Verhalten sind auf diese Weise komfortabel zu entwerfen. Damit ist der Weg eingeschlagen, hin zu einer reinen Verhaltensbeschreibung von digitalen Schaltungen mittels Petri-Netzen, aus denen dann die eigentliche Schaltung generiert werden kann.

Die große Leistung dieses Buches liegt in der hohen Qualität der mathematischen Definition des neuen Petri-Netz-Typs, den Herr Brandau mit Digital Circuit Petri Nets (DCPN) bezeichnet hat. Durch diese umfassende formale Definition werden zusätzliche Erweiterungen in Richtung Verhaltensbeschreibung auf einfache Weise ermöglicht, um das beschriebene Ziel zu erreichen. Zusätzlich wird eine Implementierung der Definitionen in einem Tool namens Logical PetriNet vorgestellt, mit dem entsprechende Schaltungen bereits beschrieben und synthetisiert werden können. Beispielschaltungen und Erläuterungen runden die Arbeit ab. Das Buch ist sehr zu empfehlen für Entwickler digitaler Schaltungen.

Prof. Dr.-Ing. Dietmar Tutsch
Lehrstuhl für Automatisierungstechnik/Informatik
Bergische Universität Wuppertal

Vorwort

Der Ansatz für meine Dissertation ist aus der Verbindung mehrerer meiner Studieninteressen entstanden. Auf der einen Seite haben mich Modellierungen von Systemen immer beschäftigt, wodurch in dieser Arbeit die Petri-Netze eine große Rolle spielen. Zum anderen haben Hardware-Beschreibungssprachen mein Interesse geweckt und ich habe mich intensiv mit diesen beschäftigt. Daraus ist die Frage entstanden, ob beide Themen nicht miteinander verknüpft werden können und wenn ja, in welcher Art und Weise dies geschehen kann. Heraus gekommen ist das vorliegende Buch, indem aus neu definierten Petri-Netzen, durch einen formalisierten Transformationsprozess, Hardware-Beschreibung generiert werden. Die Umsetzung der Forschungsergebnisse in Software zur Verifikation ist ein weiteres Interessengebiet und hat die Arbeit abgerundet und deutlich verbessert.

Ich habe die Arbeit an der Bergischen Universität Wuppertal am Lehrstuhl für Automatisierungstechnik/Informatik geschrieben. Für seine Unterstützung und ein immer offenes Ohr möchte ich meinem Doktorvater Prof. Dr.-Ing. Dietmar Tutsch danken. Ebenfalls hatte mein Zweitgutachter Prof. Dr.-Ing. habil. Carsten Gremzow immer gute Ratschläge und Hinweise, die bei der Entstehung viel beigetragen haben. Ebenfalls möchte ich Prof. Dr.-Ing. habil. Armin Zimmermann für seine Hinweise zur Verbesserung dieser Arbeit danken.

Ich danke meinen Kollegen am Lehrstuhl dafür, dass Sie meine Launen ausgehalten und mit Ratschlägen zur Verbesserung der Arbeit beigetragen haben. Ebenso möchte ich mich bei meinen Freunden dafür bedanken, dass sie auch nach einem Jahr wenig Kontakt noch immer uneingeschränkt an meiner Seite stehen und jederzeit für mich da sind. Gleiches gilt für meine Eltern und Schwester, die mit mir viel durchmachen mussten, aber trotzdem immer vorbehaltlos zu mir gestanden haben.

Ich widme meiner Freundin, Seelenverwandten und großen Liebe Jenny diese Dissertation, ohne Sie wäre das vorliegende Werk niemals entstanden. Sie hatte immer aufmunternde Worte und eine Schulter zum Anlehnen, egal wie schwierig die Situation auch war.

Inhaltsverzeichnis

Symbolverzeichnis	XV
Eigenschaften	XVII
Strategien	XIX
Abbildungsverzeichnis	XXI
Tabellenverzeichnis	XXV
Listingsverzeichnis	XXVII
Algorithmenverzeichnis	XXIX
Kurzfassung	XXXI
1 Einleitung	1
1.1 Motivation	2
1.2 Ziel der Arbeit	4
1.3 Aufbau der Arbeit	6
2 Grundlagen und Stand der Technik	7
2.1 Petri-Netze	7
2.1.1 Einsatzgebiete	8
2.1.2 Definition	8
2.1.3 High-Level Petri-Netze	11
2.1.4 Analyse	12
2.2 Digitalschaltungen	17
2.2.1 Kombinatorische Logik	18
2.2.2 Asynchron-Schaltwerke	19
2.2.3 Synchron-Schaltwerke	21
2.2.4 Hazards	22
2.3 Hardwarebeschreibungssprachen	26
2.3.1 VHDL	26

2.3.2	Verilog	30
2.4	Endliche Automaten	32
2.5	Stand der Forschung	34
3	Digital Circuit Petri Nets	41
3.1	Erweiterungen	41
3.1.1	Ein- und Ausgangsstellen	42
3.1.2	Subnetze	43
3.2	Definition	46
3.3	Eigenschaften	49
3.3.1	Erreichbarkeitsgraph	55
3.3.2	Lebendigkeit und Terminierung	63
3.3.3	Markierungen	66
3.3.4	Invarianten	68
3.4	Simulation	69
4	Transformation von DCPN nach VHDL	73
4.1	Schaltelementerzeugung	73
4.1.1	Kombinatorische Logik	73
4.1.2	Takterzeugung für sequentielle Logik	76
4.1.3	Sequentielle Logik	80
4.2	System zur Transformation	85
4.3	Validierung des Netzes	87
4.4	Optimierung des Netzes	95
4.5	Netzanalyse	99
4.6	Synthese	108
4.6.1	Kombinatorische Logik	108
4.6.2	Sequentielle Schaltungen	114
4.7	Erzeugung der VHDL-Beschreibung	125
4.7.1	Erzeugung der Schnittstelle	125
4.7.2	Kombinatorik	130
4.7.3	Sequentiell	133
4.8	Verifikation der Schaltung	142
4.9	Analyse bestehender digitaler Schaltungen	145
4.10	Gesamtüberblick Transformationsprozess	148
5	Implementierung und Validierung	151
5.1	Logical PetriNet	151
5.1.1	Platzierungsoptimierung	153
5.1.2	Universelle Exportschnittstelle	156

5.1.3	Logger	158
5.1.4	Markenspiel	160
5.1.5	Analyse	161
5.1.6	Simulation	161
5.1.7	Transformation	163
5.1.8	Funktionsbibliothek	166
5.1.9	Interner Aufbau	168
5.1.10	Validierung	174
5.2	Regeln zur Modellierung	176
5.3	Transformation exemplarischer DCPN	178
5.3.1	Multiplexer	178
5.3.2	Addierer	184
5.3.3	Lauflicht	192
5.3.4	Register	196
5.3.5	Arithmetisch Logische Einheit	203
6	Fazit und Abgrenzung	217
7	Zusammenfassung und Ausblick	225
7.1	Zusammenfassung	225
7.2	Ausblick	228
	Literaturverzeichnis	231
	Anhang	255

Symbolverzeichnis

Symbol	Beschreibung
A	Menge der Kanten der Erreichbarkeitsgraphen
B	Menge der Bedingungen der Erreichbarkeitsgraphen
E	Menge der Knoten der Erreichbarkeitsgraphen
F	Flussrelation der Kanten des DCPN
H	Matrix der hemmenden Kanten
I	Matrix der Eingangskanten
IRG	Eingabeorientierter Erreichbarkeitsgraph
K	Menge aller Konflikte
M	Markierung des Petri-Netzes
N	Menge der Netzverbinder
O	Matrix der Ausgangskanten
P	Menge der Stellen
P_{sub}	Menge der Substellen
PN	Petri-Netz
\mathcal{P}	Prioritäten
RG	Erreichbarkeitsgraph
RRG	Reduzierter Erreichbarkeitsgraph
S	Schaltvektor
SF	Schaltfolge
SW	Schaltzeiten
T	Menge der Transitionen
T_{sub}	Menge der Subtransitionen
W	Wahrheitstabelle
Z	Menge der Zyklen in einem Petri-Netz

Eigenschaften

1	Vor- und Nachbereich	49
2	Transitionsaktivierung	50
3	Schaltreihenfolge	51
4	Markierungsübergang	51
5	Beschränktheit	52
6	Zusammenhang	52
7	Quelle	52
8	Senke	53
9	Aufspaltung und Synchronisation	53
10	Statische Konfliktfreiheit	53
11	Nebenläufigkeit	54
12	Konservativität	54
13	Free Choice	54
14	Siphon	55
15	Trap	55
16	Erreichbarkeitsgraph	55
17	Suberreichbarkeitsgraph Subtransition	59
18	Suberreichbarkeitsgraph Substelle	61
19	Reduzierter Erreichbarkeitsgraph	62
20	Lebendigkeit Transition	63
21	Lebendigkeit DCPN	64
22	Lebendigkeit Markierung	64
23	Netztyp	65
24	Zyklus	65
25	Terminierung	65
26	Markierungsbereiche	66
27	Zeitliche Eingangsstellenänderung	66
28	Zustandsfreiheit	68
29	Stellen-Invariante	68
30	Transitions-Invariante	69

Strategien

1	Vorhandener Eingang und Ausgang	88
2	Vorhandene Transition	88
3	Kanten bei Eingangs- und Ausgangsstellen	89
4	Transition ohne eingehende/ausgehende Kante	89
5	Stellen ohne eingehende/ausgehende Kante	90
6	Eingehende/ausgehende Kanten bei Subnetzen	91
7	Zusammenhang	92
8	Starker Zusammenhang	93
9	Statische Konflikte	94
10	Entfernung von Elementen ohne Kanten	96
11	Zusammenfassung von parallelen Transitionen	97
12	Zusammenfassung von Transitionen in Reihe	98
13	Entfernung von Stellen ohne eingehende Kante	98
14	Entfernung nicht zusammenhängender Netzbereiche	98
15	Reduktion von Netzsymmetrien	99
16	Bestimmung Netztyp	100
17	Erreichbarkeitsgraphen aufstellen	101
18	Zyklen bestimmen	102
19	Terminierung	103
20	Zeitlose Terminierung	103
21	Eindeutiger Endzustand	104
22	Eindeutig definierter Zustand	105
23	Zustandsfreiheit	106
24	Schaltungstyp bestimmen	107
25	Wahrheitstabelle erzeugen	109
26	Erzeugung DNF/KNF	110
27	Optimierungsziele festlegen	110
28	Technologieunabhängige Optimierung	113
29	Bestimmung des sequentiellen Schaltungstyps	115
30	Berechnung des Takts	116
31	Bestimmung der abweichenden Schaltzeiten	117
32	Erzeugung des IRG	119
33	Bedingungen des IRG optimieren	122

34	Anpassung des IRG an den Takt	123
35	Teilnetze zusammenfügen	125
36	Hinzufügen vorher entfernter Ein- und Ausgangsstellen	126
37	Überprüfung der Namenskonvention	127
38	Erzeugung Schnittstelle Hauptnetz/hierarchielos	127
39	Erzeugung Schnittstelle Subtransition	128
40	Erzeugung Schnittstelle Substelle	129
41	Strukturbeschreibung aus Wahrheitstabelle generieren	131
42	Strukturbeschreibung aus booleschem Ausdruck erzeugen	132
43	Zustände erzeugen	134
44	Synchronen Zustandsübergang erzeugen	135
45	Asynchronen Zustandsübergang erzeugen	136
46	Folgezustände erzeugen	139
47	Ausgänge schreiben	139
48	Komponenten hinzufügen	140
49	Test-Bench erstellen	143
50	Simulationsergebnisse vergleichen	145

Abbildungsverzeichnis

1.1	Productivity Gap	2
2.1	Stellen-/Transitions-Netz	10
2.2	High-Level Petri-Netz	12
2.3	Petri-Netz zur Erläuterung mathematischer Beschreibungen .	13
2.4	Markierungen durch unterschiedliche schaltende Transitionen	14
2.5	Erreichbarkeitsgraph Philosophen-Problem	15
2.6	Gajski Y-Diagramm	17
2.7	Schaltnetz	18
2.8	Schaltkette	19
2.9	Asynchron-Schaltwerk	20
2.10	Synchron-Schaltwerk	21
2.11	Statische und dynamische Hazards	22
2.12	Funktioneller Hazard	23
2.13	Moore-Automat	33
2.14	Mealy-Automat	33
3.1	Digital Circuit Petri Net	41
3.2	Zeitliche Eingangsstellenänderung	42
3.3	Substelle	45
3.4	Subtransition	46
3.5	Hierarchieauflösung eines DCPN	56
3.6	Hierarchieloses DCPN	57
3.7	Erreichbarkeitsgraph aus Abbildung 3.6	58
3.8	Erreichbarkeitsgraph für Subtransition	60
3.9	Zeitliche Eingangsstellen-Veränderung	67
3.10	Vorgabe Eingangsstellen für Simulation	70
3.11	Simulation zeitbehaftetes DCPN	71
4.1	<i>and</i> -Verknüpfung als Petri-Netz	75
4.2	Taktrealisierung im Petri-Netz	76
4.3	Taktrealisierung im Petri-Netz II	77
4.4	Takteinsatz im DCPN	78

4.5	Taktbeschreibung an zeitbehafteten Transitionen	79
4.6	Externe Takterzeugung für DCPN	80
4.7	RS-Flip-Flop aus NAND-Gattern	81
4.8	RS-Flip-Flop	82
4.9	Taktzustandsgesteuertes D-Flip-Flop	82
4.10	Taktflankengesteuertes JK-Flip-Flop als DCPN	83
4.11	Taktzustandsgesteuertes RS-Flip-Flop als DCPN	84
4.12	Taktflankengesteuerte T-Flip-Flop als DCPN	85
4.13	Transformationsprozess	85
4.14	Zusammenhang im DCPN	92
4.15	Statische Konfliktmöglichkeiten	93
4.16	Validierungsmethoden	94
4.17	Netzoptimierung paralleler Transitionen	96
4.18	Netzoptimierung sequentieller Transitionen	97
4.19	Netzoptimierung	99
4.20	Eindeutiger Endzustand im DCPN	104
4.21	Eindeutiger Zustand im DCPN	105
4.22	Mögliche Eingangstellenbelegungen	106
4.23	Strategien der Netzanalyse	107
4.24	Optimierungsmöglichkeiten	111
4.25	Minimierung mittels Espresso-Algorithmus	112
4.26	Phasen der Logiksynthese	114
4.27	Taktbestimmung im Petri-Netz	116
4.28	Ringzähler und eingabeorientierter Erreichbarkeitsgraph	120
4.29	Expandierter eingabeorientierter Erreichbarkeitsgraph	123
4.30	Übersicht sequentielle Synthese	124
4.31	DCPN eines Halbaddierers mit erzeugter Wahrheitstabelle	129
4.32	HDL Schnittstellenerzeugung	130
4.33	Strukturbeschreibung kombinatorische Logik	132
4.34	Asynchrones Netz	137
4.35	Integration Komponente im DCPN	140
4.36	Übersicht sequentielle Hardwarebeschreibung	141
4.37	Simulationsergebnisse Xilinx ISE	144
4.38	Simulationsergebnisse Logical PetriNet	144
4.39	Verifikation der erzeugten Schaltung	145
4.40	Transformation Schaltung in Wahrheitstabelle	147
4.41	Simulation der Schaltung aus Abbildung 4.40 (a)	147
4.42	Simulation der Schaltung aus Abbildung 4.40 (b)	148
4.43	Überblick Transformation Teil 1	149
4.44	Überblick Transformation Teil 2	150

5.1	<i>Logical PetriNet</i> GUI	152
5.2	Interner Aufbau LPN	153
5.3	Platzierungsoptimierung verschiedener Graphen	153
5.4	Vergleich automatisierte Platzierung	156
5.5	Simulation Exportmöglichkeiten	158
5.6	Logger in LPN	159
5.7	Multithreading Ablauf	162
5.8	Transformationsprozess in <i>Logical PetriNet</i>	163
5.9	Validierung der Simulation im LPN	166
5.10	Bibliothekselemente	167
5.11	Kante zu Bibliothekselement	168
5.12	Paket-Diagramm von Logical PetriNet	169
5.13	Paketdiagramm Sortierung und Parser	170
5.14	Paketdiagramm action, print, export, bib	171
5.15	Paketdiagramm GUI	171
5.16	Paketdiagramm Simulation	172
5.17	Paketdiagramm Petri-Netz	173
5.18	Paketdiagramm Transformation	174
5.19	Multiplexer als DCPN	179
5.20	Hardwarebeschreibung des Multiplexers	181
5.21	Hardwarebeschreibung der DNF des Multiplexers	182
5.22	Hardwarebeschreibung mittels Quine McCluskey	183
5.23	Subnetz eines Halbaddierers	184
5.24	Hierarchieloser Volladdierer	185
5.25	Volladdierer als synthetisierte Hardware	187
5.26	Hardwarebeschreibung des Volladdierers mittels DNF	187
5.27	Optimierte Hardwarebeschreibung mit Quine McCluskey	188
5.28	4 Bit Volladdierer	189
5.29	Synthetisierte Hardware aus dem DCPN in Abbildung 5.28	190
5.30	4 Bit-Volladdierer Schaltung mit beibehaltener Hierarchie	192
5.31	Laufflicht DCPN mit zwei Eingängen	193
5.32	Eingabeorientierter Erreichbarkeitsgraph des Laufflichts	193
5.33	Schaltung des Laufflichts	194
5.34	Register als DCPN	196
5.35	Schaltung des Registers	197
5.36	Simulationsergebnisse des Registers	199
5.37	Register mit enable-Eingang	200
5.38	Schaltungsbeschreibung des Registers mittels der Xilinx ISE	201
5.39	Simulation des Registers enable	202
5.40	Aufbau und Befehlssatz der ALU	204

5.41	Detailansicht der ALU	205
5.42	DCPN für die Eingänge des Addierers	208
5.43	Flags des Addierers als DCPN	209
5.44	Addierer als DCPN	209
5.45	DCPN des Shifters	210
5.46	Logikeinheit der ALU als DCPN	211
5.47	Flags der ALU als DCPN	212
5.48	Komplette ALU als DCPN	213
5.49	Testsignale zur Simulation der ALU	214
5.50	Simulation der Ausgangssignale der ALU	215
5.51	Simulationsergebnisse der Flags	215
A.1	Register als Schaltung aus Vivado	257
A.2	DCPN des Addierereingangs der ALU	258
A.3	DCPN des Zweiten Eingangs des Addierers	258
A.4	Schaltung der Flags des Addierers	260
A.5	Schaltung des Addierers	261
A.6	Schaltung des Shifters	264
A.7	Schaltung der Logikeinheit der ALU	265
A.8	Schaltung der Flags der ALU	268
A.9	DCPN des Splitters der ALU	269
A.10	DCPN Multiplexer der ALU	270
A.11	Schaltung des Multiplexers	271
A.12	Schaltung der kompletten ALU	272
A.13	PNML-Schema Beschreibung Graphik- und Objekttyp	278
A.14	PNML-Schema des Petri-Netz	279

Tabellenverzeichnis

2.1	Grundlegende Logikverknüpfungen (Gatter)	18
2.2	Fünfwertige Logik zur Erkennung von Hazards	24
2.3	Hazardbeschreibung der fünfwertigen Logik	25
3.1	Markierungstabelle	58
4.1	Gatter mit beliebig vielen Eingängen	75
4.2	Strategien zur Validierung des Netzes	88
4.3	Strategien zur Netzoptimierung	95
4.4	Strategien zur Netzanalyse	100
4.5	Strategien zur Synthese kombinatorischer Netze	109
4.6	Entwurfsverfahren	111
4.7	Strategien zur Logiksynthese sequentieller Netze	115
4.8	Markierungen des eingabeorientierten Erreichbarkeitsgraphen	121
4.9	Bedingungen der Kanten des IRG	121
4.10	Strategien zur Schnittstellenbeschreibung	126
4.11	Strukturbeschreibung kombinatorischer Logik	131
4.12	Strukturbeschreibung sequentieller Logik	133
4.13	Markierung des IRG	137
4.14	Bedingungen der Kanten des IRG	137
4.15	Strategien zur Verifikation der Schaltung	142
5.1	Laufzeitanalyse Platzierungsoptimierung	155
5.2	Übersicht Exportfunktionalität	157
5.3	IRG Markierungen des Lauflichts	193
5.4	IRG Übergangsbedingungen des Lauflichts	193
5.5	IRG Markierung des Registers	200
5.6	IRG Übergangsbedingungen des Registers	200
5.7	Eingangsbelegungen des Addierers	206
5.8	Testinstruktionen für die ALU	214
6.1	Laufzeitvergleich Transformation	223

Listingsverzeichnis

2.1	Schnittstellenbeschreibung eines 4 zu 1 Multiplexers	27
2.2	Verhaltensbeschreibung eines 4 zu 1 Multiplexers	27
2.3	Sequentielle Beschreibung Multiplexer	29
2.4	VHDL-Beschreibung eines D-Flipflop	29
2.5	VHDL-entity-Beschreibung eines Registers mittels generic . .	30
2.6	Verhaltensbeschreibung eines 4 zu 1 Multiplexers in Verilog .	31
4.1	Schnittstelle Hauptnetz in VHDL	128
4.2	Schnittstelle Subtransition in VHDL	128
4.3	Schnittstelle Substelle in VHDL	129
4.4	Schnittstellenbeschreibung des generierten Halbaddierers . . .	130
4.5	Beschreibung von Kombinatorik aus Wahrheitstabelle	132
4.6	Beschreibung von Kombinatorik aus booleschem Ausdruck . .	132
4.7	Zustandsbeschreibung IRG	134
4.8	Zustandsspeicher synchron IRG	135
4.9	Zustandsspeicher asynchron IRG	135
4.10	Zustandsübergänge IRG	136
4.11	Berechnung Folgezustand IRG	138
4.12	Ausgangsbeschreibung IRG	139
4.13	Beschreibung Hauptnetz aus Abbildung 4.35	141
4.14	Beschreibung der Test-Bench für das Netz des Halbaddierers	143
5.1	Erweiterung der Test-Bench um TEXTIO	164
5.2	Schnittstelle Multiplexer	180
5.3	Beschreibung Multiplexer Wahrheitstabelle	181
5.4	Beschreibung Multiplexer DNF	182
5.5	Beschreibung Multiplexer Quine McCluskey	183
5.6	Schnittstelle Volladdierer	185
5.7	Beschreibung Volladdierer Wahrheitstabelle	186
5.8	Beschreibung Volladdierer DNF	187
5.9	Beschreibung Volladdierer Quine McCluskey	187
5.10	Schnittstelle 4-bit Volladdierer	189

5.11	Verhaltensbeschreibung 4-bit Volladdierer	190
5.12	Beschreibung 4-bit Volladdierer mit Hierarchie	191
5.13	Schnittstellenbeschreibung Lauflicht	194
5.14	Verhaltensbeschreibung Lauflicht	194
5.15	Schnittstellenbeschreibung Register	197
5.16	Verhaltensbeschreibung Register	198
5.17	Schnittstellenbeschreibung Register mit enable Eingang	201
5.18	Verhaltensbeschreibung Register mit enable Eingang	201
5.19	Beschreibung erster Eingang des Addierers	207
A.1	AND-Gatter	255
A.2	OR-Gatter	256
A.3	NOT-Gatter	256
A.4	Beschreibung Eingang zwei des Addierers	259
A.5	Beschreibung der Flags des Addierers	260
A.6	Beschreibung Addierer der ALU	261
A.7	Beschreibung Shifter der ALU	263
A.8	Beschreibung Logikeinheit der ALU	266
A.9	Beschreibung Flags der ALU	267
A.10	Beschreibung Splitter der ALU	269
A.11	Beschreibung Multiplexer der ALU	269
A.12	Beschreibung der ALU	272
A.13	Testbench zur ALU	274
A.14	PNML Elternelement	278
A.15	PNML UUID	278
A.16	PNML Elementtyp	280
A.17	PNML Markenanzahl	280
A.18	PNML Stellentyp	280
A.19	PNML Zeiteinheiten Transitionen	280
A.20	PNML Transitionstypen	281
A.21	PNML Bibliothekselement	281
A.22	PNML Netzverbindertyp	281
A.23	PNML Netzverbinder-Verbindung	281
A.24	PNML Kantentyp	281

Algorithmenverzeichnis

3.1	Aufstellen des Erreichbarkeitsgraphen	59
3.2	Erreichbarkeitsgraph Subtransition	61
3.3	Erreichbarkeitsgraph Substelle	62
3.4	Aufstellen des reduzierten Erreichbarkeitsgraphen	63
4.1	Eingabeorientierter Erreichbarkeitsgraph	118
4.2	Weitere zeitlose Transitionen feuern	119
4.3	Füge Kante zum IRG hinzu	120
4.4	Aufsplittung Knoten des IRG für Takt	122
4.5	Erweiterung um notwendige Kanten bei der Expandierung . .	124

Kurzfassung

In dieser Arbeit wird der Entwurf und die Verifikation von digitalen Schaltungen mittels Petri-Netzen behandelt. Dabei wird die Definition der Bedingungs-Ereignis-Netze erweitert, woraus die *Digital Circuit Petri Nets* entstanden sind. Im ersten Teil der Arbeit sind die Eigenschaften dieses Netztyps aufgezeigt, welche bei der späteren Transformation in digitale Schaltungen zum Einsatz kommen.

Der Hauptteil der Arbeit beruht auf einer Petri-Netz-basierten Entwurfsmethodik zum Erzeugen von digitalen Schaltungen. Der hier vorgestellte Transformationsprozess besteht aus den Teilschritten Modellierung des Netzes, Validierung des Netzes, Netzoptimierung, Strukturanalyse, Logiksynthese, Überführung in eine Hardwarebeschreibungssprache und einer abschließenden Verifikation der erzeugten Schaltung. Schwerpunkte der Transformation sind hierbei die Synthese des Netzes und die vorherige Strukturanalyse, da aus dieser der Schaltungstyp abgeleitet wird. Hierbei werden die Schaltungen in kombinatorische und sequentielle Schaltungen aufgesplittet, um für den jeweiligen Schaltungstyp optimierte Algorithmen zur Synthese anwenden zu können.

Es werden Richtlinien zur Modellierung von Petri-Netzen für digitale Schaltungen gegeben, um die Erstellung von kombinatorischer, sowie getakteter Logik zu vereinheitlichen. Bei getakteter Logik wird zwischen synchronen und asynchronen Schaltungen unterschieden. Die Erzeugung erfolgt über einen eingabeorientierten Erreichbarkeitsgraphen, der in einen Zustandsautomaten überführt wird. Zur Verifikation des erarbeiteten Prozesses der Transformation ist das Tool *Logical PetriNet* entwickelt worden, welches dem praktischen Nachweis der vorgeschlagenen Methodik dient. Die erarbeiteten Ergebnisse der Transformation werden anhand einiger komplexer Schaltungen verifiziert.



1 Einleitung

Zum Verständnis der Wichtigkeit neue Methoden und Werkzeuge für den Entwurf digitaler Schaltungen zu geben, beginnt diese Arbeit mit dem zeitlichen Verlauf des digitalen Schaltungsentwurfs. Ein besonderes Augenmerk wird auf den Transistor gelegt, da er das grundlegende Element dieser Schaltung ist. Im Jahre 1947 wurde in den *Bell Laboratories* der erste funktionierende Transistor entwickelt. Etwa zehn Jahre später entwickelte *Texas Instruments* die Technik, um mehrere Transistoren als integrierte Schaltung in einem Silizium-Chip zu realisieren. Kurz darauf hat Jack Kilby das erste integrierte Flip-Flop, bestehend aus zwei Transistoren, erstellt [BL10].

1963 entwickelt Frank Wanlass [WS63] die ersten Logikgatter basierend auf Metall-Oxid-Halbleiter-Feldeffekttransistoren (*metal-oxide-semiconductor field-effect transistor*, *MOSFET*). Hierbei kommen *nMOS*- und *pMOS*-Transistoren zum Einsatz, wobei immer ein n- und ein p-Transistor zu einem Element zusammen geschaltet werden [Bea68]. Die Verwendung von beiden Transistortypen auf einem Chip wird auch als sich ergänzender Metall-Oxid-Halbleiter (*complementary metal-oxide-semiconductor*, *CMOS*) bezeichnet.

Transistoren sind zunächst in analogen Schaltungen, wie beispielsweise in Transistor-Radios, verbaut worden, bis sie für digitale Schaltungen, wie Prozessoren, zum Einsatz kommen [WH10], [MC79]. Der Transistor ist heutzutage das am häufigsten produzierte elektrische Bauteil der Welt. Die Transistoranzahl pro Chip wurde von Gordon Moore [Moo06] untersucht und er bestätigt eine Verdopplung alle 18 Monate. Daraus ist das *Moore'sche Gesetz* abgeleitet, welches bis heute seine Gültigkeit behält, wobei eine Verdopplung etwa alle 26 Monate stattfindet. Gründe hierfür sind immer kleiner werdende Transistoren und teilweise größer werdende Chipflächen. Die Integration von Chips lässt sich in die Klassen *Small-Scale Integration (SSI)* mit ca. 10 Gattern auf einem Chip, *Medium-Scale Integration (MSI)* mit bis zu 1.000 Gattern, *Large-Scale Integration (LSI)* mit bis zu 10.000 Gattern und *Very-Large-Scale Integration (VLSI)* mit mehr als 10.000 Gattern pro Chip aufteilen.

1.1 Motivation

Seit 1981 können die Entwurfsverfahren zur Erzeugung digitaler Schaltungen immer weniger mit den technischen Möglichkeiten der Halbleiterfertigung Schritt halten. Während aus technischer Sicht die mögliche Anzahl der Transistoren pro Chip jährlich um 58% ansteigt, liegt der Anstieg der verwendeten Transistoren auf Entwurfsebene bei 21% pro Jahr. Die aufgrund dessen stetig wachsende Lücke zwischen diesen beiden Aspekten wird als *Design Productivity Gap* bezeichnet (siehe Abbildung 1.1). Um der technischen Entwicklung folgen zu können, müssen immer abstraktere Methoden zum Entwurf digitaler Schaltungen entwickelt werden. Je nach Modellierungsvorhaben kann die Beschreibung eines Systems auf verschiedenen Ebenen vonstatten gehen. So kann beispielsweise auf der Systemschicht oder auf der Register-Transfer-Ebene modelliert werden.

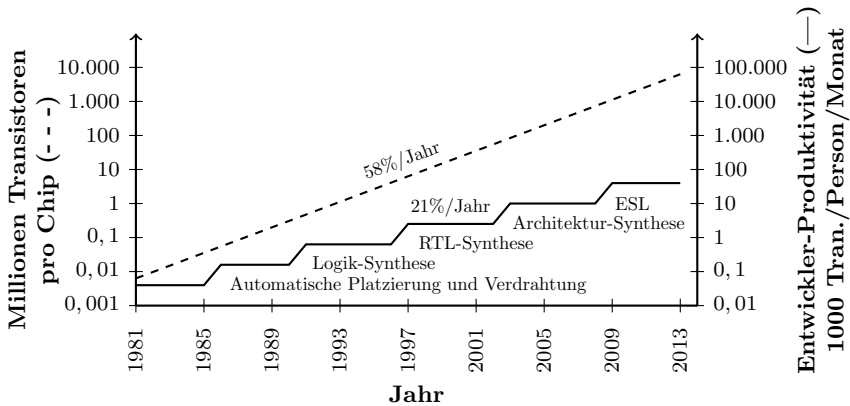


Abbildung 1.1: Entwicklung der Transistoranzahl pro Chip (gestrichelte Linie) im Verhältnis zur Entwicklerproduktivität (durchgezogene Linie) nach [RU01] und [Rei13]

Ziel dieser und folgender Arbeiten ist es, das Verhalten und die Struktur digitaler Schaltungen mittels einer grafischen Repräsentation zu modellieren und die daraus resultierenden Vorteile bezüglich der Verifikation und Validierung zu nutzen. Aus diesem Grund werden in der vorliegenden Arbeit Möglichkeiten zur Beschreibung von Strukturen digitaler Schaltungen unter Zuhilfenahme von Petri-Netzen vorgestellt. Petri-Netze sind formale

Beschreibungen von Systemen, die aus den vier Elementen Stellen, Marken, Transitionen und gerichteten Kanten bestehen. Für diese Netze existieren viele Methoden zur Validierung und Analyse, wie beispielsweise die Markov-Ketten [Mar13]. Zur Verwendung von Petri-Netzen werden in dieser Arbeit zusätzliche Elemente und Eigenschaften eingeführt, die eine Vereinheitlichung der Transformation eines Netzes in eine Schaltung ermöglichen.

Petri-Netze sind dabei für den Modellierer greifbarer als die Entwicklung von digitalen Schaltungen in VHDL oder anderen Hardware-Beschreibungssprachen, da bei den Petri-Netzen mittels grafischer Methoden modelliert werden kann. Die Motivation ist ein Verfahren zu entwickeln, bei dem die Abstraktionsebene frei gewählt werden kann. Mittels Petri-Netzen soll also eine hierarchische Anordnung der Elemente des Netzes erfolgen. Die erste Sicht auf das modellierte System kann eine sehr abstrakte Sicht sein, indem das Gesamtsystem mit einzelnen Komponenten abgebildet wird. Durch diese Art der Modellierung wird eine übersichtliche Form der Beschreibung des Systems ermöglicht. Hierzu soll in einem zu Implementierenden Werkzeug die Option bestehen, die Elemente von Petri-Netzen mittels Algorithmen optimiert zu platzieren. Hierzu können Ansätze aus der automatischen Verteilung von Graphen eingesetzt werden. Bisher bieten die Petri-Netze vor allem im Bereich der Steuerwerke und besonders bei asynchronen Schaltungen schon jetzt Analyse-Methoden, die im Bereich des reinen Schaltungsentwurf derzeit nicht verfügbar sind. Hier sind die Methoden der Petri-Netze viel tiefergehend und eignen sich zur Validierung der modellierten Schaltungen [Lie05].

Damit die Durchführung der Transformation von Petri-Netzen in digitale Schaltungen weitestgehend automatisiert ablaufen kann, muss zunächst ein Formalismus aufgestellt werden. Zur Lösung nicht-trivialer Probleme und zur Unterstützung des Anwenders sind entsprechende Methoden zur Transformation in eine geeignete Software zu integrieren. Durch den aufgestellten Formalismus soll es möglich sein sequentielle und nebenläufige Ereignisse zu modellieren. Weiterhin muss die Systemstruktur und das Systemverhalten mit dem Modell wiedergegeben werden. Durch eine Hierarchiebildung ist es möglich das System auf verschiedenen Abstraktionsebenen zu betrachten. Darüber hinaus muss die Möglichkeit bestehen das System funktional und zeitbehaftet zu simulieren. Durch die formale Petri-Netz-Spezifikation ist eine Analyse auf vielfältige Arten durchführbar, da die schon bestehenden Methoden mit Anpassungen für den neuen Petri-Netz-Typ verwendet werden können.

Die meisten der in dieser Arbeit vorgestellten Verfahren lassen sich auch auf die Rücktransformation von digitalen Schaltungen in Petri-Netze an-

wenden, was ein nützliches Nebenprodukt der Arbeit darstellt. Eine solche Rücktransformation bietet die Möglichkeit, die in dieser Arbeit beschriebenen Analysemethoden von Petri-Netzen verwenden zu können, um digitale Schaltungen zu untersuchen. Daraus folgt eine formale Validierung des Netzes und Verifikation der modellierten Schaltungen. Der noch zu erledigende Arbeitsschritt zur Rücktransformation von Schaltungen in Petri-Netze ist das Anlegen von Netzen für alle benötigten Schaltelemente.

Es stellt sich die Frage warum ausgerechnet Petri-Netze aus den bekannten und verbreiteten grafischen Sprachen zur Modellierung von Systemen zum Einsatz kommen sollten. Für die Modellierung könnten auch Zustandsautomaten eingesetzt werden, bei denen aber vor allem bei komplexeren Systemen bei der Analyse Zustandsraumexplosionen auftreten. Beim Aufteilen des Systems in mehrere kleinere Automaten tritt das Problem der Synchronisierung und das Problem einer adäquaten Abbildung der Parallelität auf [YK98]. Die Vorteile von Petri-Netzen sind die gute Verifizierbarkeit der erzeugten Netze und die einfache Erstellung von Netzen über Werkzeuge zur Modellierung, die vor allem grafischer Natur sind. Mit Petri-Netzen können asynchrone Systeme beschrieben werden, wobei die synchronen einen Spezialfall dieser Systeme darstellen, bei denen der Takt ein zusätzliches Signal zur Generierung eines Events ist. Nach Möglichkeit sollte jedoch zwischen den beiden Typen unterschieden werden, da so bessere Optimierungsmethoden für jeden Systemtyp angewandt werden können. Als zusätzlicher Netztyp existiert noch reine Kombinatorik, die ebenfalls abgebildet werden sollte.

Wünschenswert sind die folgenden Charakteristiken bei der Modellierung mittels Petri-Netzen: Zum einen soll eine präzise Syntax und Semantik des Netzes vorhanden sein. Zum anderen ist eine lesbare Modellierung, am besten mit gleichzeitiger Verifikation des Modells, und die Abbildung von Parallelität und Synchronisierung interessant. Ebenfalls wünschenswert wäre eine der Modellierungen *top-down* oder *bottom-up*. Bei Petri-Netzen bietet sich durch die Bildung von Hierarchien vor allem die *top-down*-Methode an, da so das Netz in der obersten Ebene betrachtet werden kann und durch weitere hierarchische Elemente des Netzes mehr Details angezeigt werden können. Dabei wird auch von einer Verfeinerung des Systems gesprochen.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist ein Verfahren zu erarbeiten und zu formalisieren, welches die Transformation von Petri-Netzen in digitale Schaltungen ermöglicht. Dabei ist ein modularer Aufbau anzustreben, um spätere Er-

weiterungen und Optimierungen mit möglichst wenig Aufwand implementieren und hinzufügen zu können. Bisherige Ansätze und Definitionen von Petri-Netzen sind auf ihre sinnvolle Anwendung im genannten Umfeld zu überprüfen, wobei der Aspekt der Modellierung ohne Vorkenntnisse von digitalen Schaltungen zu berücksichtigen ist.

Die Transformation soll, soweit dies möglich ist, auf bestehende Hardwarebeschreibungssprachen zurückgreifen, um die vorhandenen Tools im Bereich der Hardwarebeschreibung zur Optimierung und Platzierung zu verwendbar zu machen. Eine erneute Implementierung dieser Werkzeuge in einem eigenen Programm übersteigt den Rahmen dieser Arbeit, da die entwickelten Werkzeuge teilweise schon mehrere Jahrzehnte in Entwicklung von namhaften Herstellern wie Xilinx, Altera oder Mentor Graphics sind. Diese Werkzeuge haben sich bewährt und eine Neuimplementierung ist daher nicht notwendig.

Die Modellierung von Petri-Netzen soll für den späteren Entwickler von digitalen Schaltungen möglich sein, ohne dass dieser Kenntnisse vom Entwurf von digitalen Schaltungen besitzen muss. Um dies zu bewerkstelligen, ist ein Tool zu implementieren, welches zum einen eine handliche Modellierung eines Petri-Netzes zur Verfügung stellt und auf der anderen Seite den Transformationsprozess abbilden kann und zur Verifizierung der erarbeiteten Methoden und Strategien dient.

Weiterhin soll der hier erarbeitete Transformationsprozess, anders als in bisherigen Forschungsarbeiten üblich, das gesamte Netz analysieren und die klassische und bisher übliche Transformation von Struktur in Struktur (auch Komponentenweise genannt) aufbrechen. Im Abschnitt zum aktuellen Stand der Technik werden existierende Ansätze zur Transformation von Petri-Netzen in digitale Schaltungen vorgestellt und analysiert, welche alle auf dem Ansatz der Strukturüberführung basieren. Vorteile dieser Verfahren sind nach Möglichkeit in die zu entwickelnden Methoden und Strategien aufzunehmen.

Eine hierarchische Darstellung innerhalb der Petri-Netze wäre wünschenswert, um die Übersichtlichkeit der modellierten Netze zu gewährleisten. Hierzu sind vorhandene Methoden zur Hierarchiebildung zu untersuchen beziehungsweise eigene Subnetze zu erarbeiten und im entstehenden Programm zu integrieren. Weiterhin sollen die Subnetze dazu dienen, um schon modellierte Petri-Netze in neu zu erstellenden Netzen als Bibliotheks-Elemente zu verwenden.

Eine Verifizierung der beschriebenen Methoden anhand von Test-Netzen und deren Transformation soll die Arbeit abschließen.

1.3 Aufbau der Arbeit

Die Arbeit beginnt mit dem Ziel und der Motivation in diesem Kapitel, woraufhin im zweiten Kapitel eine Einführung in die Theorie der Petri-Netze und eine Beschreibung der Einsatzgebiete dieser Netze folgt. Daran anknüpfend folgt die Vorstellung verschiedener Analysemethoden, die bei der Untersuchung von Petri-Netzen zur Anwendung kommen können. Nachfolgend wird eine grundlegende Einführung in den Bereich der digitalen Schaltungen und die unterschiedlichen Strukturen von Schaltungen gegeben. Daran anschließend erfolgt die Erläuterung der beiden Hardwarebeschreibungssprachen Verilog und VHDL. Der aktuelle Stand der Forschung rundet dieses Kapitel ab.

Im dritten Kapitel erfolgt die Erweiterung der im vorhergehenden Kapitel eingeführten Petri-Netze zur eigenen Klasse der *Digital Circuit Petri Nets* (DCPN) mit den Erweiterungen um Eingangs- und Ausgangsstellen und Subnetzen in unterschiedlichen Ausprägungen. Die Definition der DCPN vollendet die Erweiterungen des neuen Netztyps. Abgeschlossen wird das Kapitel mit den Eigenschaften der DCPN, die für eine Transformation in eine Hardwarebeschreibungssprache benötigt werden.

In Kapitel Vier wird die Transformation von Petri-Netzen in VHDL beschrieben. Zunächst werden verschiedene Elemente der Digitaltechnik sowie ihre Abbildung als Petri-Netz vorgestellt. Dadurch stehen Netze mit bekannten Eingangs- und Ausgangskombinationen zur Verfügung. Anschließend werden verschiedene Beschreibungsmöglichkeiten eines Taktes innerhalb eines Petri-Netzes diskutiert. Darauf folgt der Entwurfsprozess, der die einzelnen Schritte erläutert um von einem Petri-Netz zu einer Hardwarebeschreibung zu gelangen. Dabei wird in einigen Schritten der Transformation zwischen zwei möglichen Verfahren unterschieden, je nachdem ob es sich um kombinatorische oder sequentielle Logik handelt.

Eine mögliche Implementierung der hier vorgestellten Verfahren erfolgt in der Software *Logical PetriNet*, die im fünften Kapitel vorgestellt wird. Einige nicht triviale Beispiele zeigen die Methoden des Tools und verifizieren die Ergebnisse der vorhergehenden Kapitel. Ebenfalls in diesem Kapitel werden hilfreiche Empfehlungen zur Modellierung von Petri-Netzen gegeben, um diese in digitale Schaltungen überführen zu können.

Im folgenden Kapitel wird ein Fazit dieser Arbeit und eine Abgrenzung zu den Veröffentlichungen aus dem Stand der Forschung gegeben und zeigt Unterschiede zu diesen auf. Abgeschlossen wird die Arbeit mit einer Zusammenfassung und dem Ausblick auf mögliche weitere Forschungsarbeiten im Bereich der Transformation von Petri-Netzen in digitale Schaltungen.