Formale Verifikation digitaler Systeme mit Petrinetzen

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

vorgelegt dem Rat der Fakultät für Mathematik und Informatik
an der Friedrich-Schiller-Universität Jena

von Diplom-Physiker Torsten Schober
geboren am 13.7.1969 in Halle (Saale)

Gutachter

1. Prof. Dr.-Ing. Werner Erhard
2. Prof. Dr.-Ing. Wolfgang Fengler, TU Ilmenau

Tag der letzten Prüfung des Rigorosums: 22. April 2003

Tag der öffentlichen Verteidigung: 02. Juli 2003

Zusammenfassung

Diese Arbeit beschäftigt sich mit dem Entwurf und der Verifikation digitaler Systeme. Sie liefert Beiträge zur formalen Verifikation digitaler Systeme und zu einem vereinheitlichten Entwurfsablauf für synchrone und asynchrone digitale Systeme. Im ersten Teil wird eine petrinetz-basierte Hardware-Entwurfsmethodik vorgeschlagen. Die Phasen des Entwurfsablaufes sind mit Petrinetzen untersetzt, so dass sowohl synchron getaktete als auch asynchrone digitale Systeme entworfen werden können. Weiter ausgearbeitet werden die Entwurfsabschnitte Modellierung und Verifikation auf funktionaler Ebene. In der Phase der Modellierung wird ein digitales System mit einem Petrinetz-Modell beschrieben. Die Äquivalenz zwischen Markenfluss des Petrinetz-Modells und Signalfluss des digitalen Systems bildet die Grundlage für eine formale Verifikation im folgenden Entwurfsabschnitt. Den Schwerpunkt der Arbeit bildet die formale Verifikation digitaler Systeme mit Petrinetzen. Auf der Basis der angewandten Modellierungsstrategie wird eine Verifikationsmethodik eingeführt. Eine Reihe von Algorithmen zur Eigenschaftsverifikation wird auf komplexe digitale Kontrollpfade angewandt. Die für das Petrinetz-Modell erzielten Eigenschaften werden geeignet als Eigenschaften des digitalen Systems interpretiert und ermöglichen die Ableitung einer Aussage zur formalen Verifikation des digitalen Systems. Für verifizierte Petrinetz-Modelle wurden Optimierungsstrategien erarbeitet und mit Petrinetz-Analysealgorithmen umgesetzt. Ein Petrinetz-Werkzeug für die Modellierung, Analyse, formale Verifikation und Optimierung digitaler Systeme dient dem praktischen Nachweis der Anwendbarkeit der vorgeschlagenen Methodik.

Inhaltsverzeichnis

1 Motivation und Einordnung der Arbeit **1**

2 Petrinetze **4**
2.1 Definitionen und Netzklassen 4
2.2 Petrinetze als Modellierungswerkzeug 7
2.3 Analyse von Petrinetz-Modellen 8
2.3.1 Struktureigenschaften 9
2.3.2 Verhaltenseigenschaften 10

3 Digitale Systeme und Petrinetze **14**
3.1 Interpretierte Petrinetze 14
3.2 Modellierung digitaler Systeme mit Petrinetzen 15
3.3 Hardware-Entwurf mit Petrinetzen 17

4 PNDes - Eine petrinetz-basierte Hardware-Entwurfsmethodik **19**
4.1 Entwurfsmethodik 19
4.1.1 Entwurfsautomatisierung 24
4.2 Modellierung und Verifikation digitaler Systeme mit PNDes 24

5 Modellierung digitaler Systeme mit Interpretierten Petrinetzen **27**
5.1 SIPN-Ansatz 27
5.2 Transformationen zwischen VHDL-Konstrukten und Petrinetz-Strukturen . . . 30
5.3 Hierarchische Petrinetz-Modelle 32
5.4 Eine Anwendung - Das Petrinetz-Modell eines Mikroprozessors 35
5.4.1 Petrinetz-Modell des sequentiellen DLX-Prozessors 36
5.4.2 Petrinetz-Modell des nebenläufigen DLX-Prozessors 42
5.5 Funktionale Simulation und Validierung des Petrinetz-Modells 49
5.5.1 Simulation mit Auswertung äußerer Schaltbedingungen 50
5.5.2 Simulation alternativer Befehlssequenzen 50

6 Verifikation digitaler Systeme mit Interpretierten Petrinetzen **51**
6.1 Verifikationsziele 51

6.2 Verifikationsstrategien für sequentielle Kontrollpfade 53
6.2.1 1-Beschränktheit 53
6.2.2 Lebendigkeit 55
6.2.3 Reversibilität 61
6.2.4 Schleifenfreiheit 61
6.2.5 Statische und dynamische Konflikte 63
6.2.6 Anwendung der Analysestrategien 65
6.3 Verifikationsstrategien für nebenläufige Kontrollpfade 68
6.3.1 Konservativität und Überdeckung mit P-Invarianten 68
6.3.2 1-Beschränktheit und Lebendigkeit 72
6.3.3 Schleifen- und Konfliktfreiheit 74
6.3.4 Anwendung der Analysestrategien 75
6.4 Optimierung verifizierter Petrinetz-Modelle 78
6.4.1 Analyse von Kreisen und Distanzen für sequentielle Kontrollpfade . . 78
6.4.2 Reduktionsanalyse 79
6.5 Eine Anwendung - Verifikation des Petrinetz-Modells eines Mikroprozessors 84
6.5.1 Verifikation der Petrinetz-Modelle des DLX-Prozessors 84

7 Das Werkzeug VeriCon 86
7.1 Anforderungen an das Petrinetz-Werkzeug 86
7.2 Konzeption und Umsetzung 88
7.3 Test und Erweiterung 89

8 Zusammenfassung und Ausblick 91

A Petrinetz-Modell I 93

B Petrinetz-Modell II 98

C Formelzeichen 107

Abbildungsverzeichnis

2.1 Modellierung mit Free-Choice-Netzen 8
2.2 Deadlock-Trap-Eigenschaft 11

4.1 PNDes 21
4.2 Modellierung und Verifikation mit PNDes 25

5.1 SIPN mit Hardware-Interpretation 29
5.2 Strukturelle PN-Umsetzung von VHDL-Konstrukten 31
5.3 Transitionenverfeinerung 32
5.4 Platzverfeinerung durch Transitionenverfeinerung 33
5.5 Parallele und sequentielle Expansion 34
5.6 DLX-Prozessor mit externem Speicher 35
5.7 Befehlsformate des DLX-Prozessors 35
5.8 Kontrollpfad des sequentiellen DLX-Prozessors 36
5.9 Datenpfad des sequentiellen DLX-Prozessors 37
5.10 Reduziertes Petrinetz-Modell des sequentiellen DLX-Prozessors 40
5.11 Optimierte Modellierung von Kontrollanweisungen 41
5.12 DLX-Pipeline 43
5.13 Vereinfachtes Petrinetz-Modell einer Prozessor-Pipeline 44
5.14 Reduziertes Petrinetz-Modell des nebenläufigen DLX-Prozessors (Pipeline-Struktur) 46
5.15 Reduziertes Petrinetz-Modell des nebenläufigen DLX-Prozessors (ALU- und load/store-Befehle) 47
5.16 Reduziertes Petrinetz-Modell des nebenläufigen DLX-Prozessors (Pipeline-RESET) 48

6.1 Nicht stark-zusammenhängende Zustandsmaschinen 56
6.2 Teilnetz mit Markenkonsumtion 58
6.3 Quelle und Senken, die zur Einschränkung der Lebendigkeit führen 60
6.4 Schleifenfreies und schleifenbehaftetes Teilnetz 62
6.5 Konfliktbehaftete Netzstrukturen 64

6.6 Verifikationsschema zur funktionalen Verifikation von Petrinetz-Modellen sequentieller Kontrollpfade 67
6.7 Verifikationsschema zur funktionalen Verifikation von Petrinetz-Modellen nebenläufiger Kontrollpfade 77
6.8 Zusammenfassung äußerer Schaltbedingungen als Optimierung des Petrinetz-Modells (Ausschnitt Abb. A.4) 80
6.9 Netzsymmetrien im Petrinetz-Modell 82

7.1 Komponenten für die Erstellung des Petrinetz-Werkzeuges VeriCon 88

A.1 Petrinetz-Modell des sequentiellen DLX-Prozessors (Teil a) 94
A.2 Petrinetz-Modell des sequentiellen DLX-Prozessors (Teil b) 95
A.3 Petrinetz-Modell des sequentiellen DLX-Prozessors (Teil c) 96
A.4 Petrinetz-Modell des sequentiellen DLX-Prozessors (Teil d) 97

B.1 Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil a - Pipeline-Struktur IF/DEC) 99
B.2 Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil b - Pipeline-Struktur DEC/EXE/MEM/WB) 100
B.3 Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil c - Untersetzung der DEC-STUFE) 101
B.4 Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil d - Ausführung der Sprungbefehle) 102
B.5 Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil e - Ausführung der Speicherbefehle) 103
B.6 Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil f - Auswahl der ALU-Befehle) 104
B.7 Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil g - Ausführung der ALU-Befehle) 105
B.8 Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil h - Pipeline-Kontrolle) 106

Tabellenverzeichnis

5.1 Befehlsumfang des reduzierten DLX-Modells 39
5.2 Ausnahmebehandlungen 48

6.1 Petrinetz-Eigenschaften und deren Interpretation im Petrinetz-Modell . . . 52
6.2 Zusammenfassung der Analysestrategien für Petrinetz-Modelle sequentieller Kontrollpfade 65
6.3 Zusammenfassung der Analysestrategien für Petrinetz-Modelle nebenläufiger Kontrollpfade 76

7.1 Realisierung der Entwurfsphasen Modellierung und Funktionale Analyse . . 87

Kapitel 1

Motivation und Einordnung der Arbeit

Die Entwicklung und Anwendung von Methoden zur formalen Verifikation digitaler Systeme erfährt seit einigen Jahren im akademischen und industriellen Bereich einen starken Aufschwung. Hauptsächliche Gründe hierfür sind Forderungen, die aus dem Entwurf und Einsatz komplexer digitaler Systeme resultieren. Das Interesse an Verfahren und Werkzeugen, welche digitale Hardware automatisiert formal verifizieren, wird mit folgenden Fakten belegt (Eve01):

- Simulationskrise bei konventionellen Verifikationsmethoden durch das exponentielle Ansteigen der Integrationsdichte von Halbleiterbauelementen
- Durchdringung aller, insbesondere sicherheitskritischer Bereiche des Alltagslebens mit digitalen Systemen
- Kostenaufwand von Rückrufaktionen fehlerbehafteter Hardware

Derzeit werden in den Entwicklerteams zwischen 60-80% des Gesamtaufwandes beim Entwurf komplexer digitaler Systeme für die Verifikation benötigt (Ber00). Im Gegensatz zur Verifikation durch Simulation ist die formale Verifikation vollständig. Alle Signalpfade eines Entwurfs werden überprüft. Neben diesem qualitativen Vorteil bieten formale Verfahren auch einen Effizienzvorteil, da sie bei komplexen Entwürfen oft weitaus schneller Ergebnisse liefern als herkömmliche Methoden (Simulationskrise). Komplizierte Testumgebungen werden durch das Formulieren eines Satzes von Eigenschaften ersetzt und führen so zur einfacheren und schnelleren Erstellung und Handhabung einer Testumgebung. Bei fehlgeschlagener Verifikation werden Fehler durch die Generation von Gegenbeispielen oder durch Nichterhaltung einer bestimmten Eigenschaft schnell lokalisiert und sind somit eliminierbar. Die Nachteile formaler Methoden liegen momentan in der Begrenzung der Anwendbarkeit auf Ausschnitte des Gesamtentwurfes, im Eingliederungsaufwand von entsprechenden Werkzeugen in den Entwurfsfluss

und im Einarbeitungsaufwand für Hardwareentwickler. Der optimale Weg, um den Entwurf komplexer digitaler Systeme zu beherrschen, ist die gezielte Nutzung und Weiterentwicklung formaler Verifikationsmethoden und deren Kombination mit konventionellen Simulationsmethoden.

Grundsätzlich werden zwei Arten formaler Verifikationsmethoden unterschieden: Mit einer **Äquivalenzprüfung** wird die funktionale Übereinstimmung von Systemspezifikation und Implementierung nachgewiesen. Ebenso kann die Äquivalenz zweier Systembeschreibungen von aufeinanderfolgenden Entwurfsschritten verifiziert werden.

Die **Eigenschaftsprüfung** untersucht anhand einer Systemspezifikation bestimmte Eigenschaften und prüft damit die Funktionalität eines Entwurfs. Systemrelevante Eigenschaften können zeitabhängig oder zeitunabhängig spezifiziert und entsprechend verifiziert werden.

Diese Arbeit stellt eine Methode der formalen Verifikation als Eigenschaftsprüfung vor und wendet diese auf digitale Systeme an. Dabei wird die Theorie der Petrinetze von C. A. Petri (Pet62) als Grundlage für die Verifikation digitaler Systeme genutzt. Petrinetze sind jedoch nicht nur für die Systemanalyse geeignet. Mithilfe von Netzinterpretationen können Arbeitsprozesse, technische Systeme oder auch Entwurfsabläufe modelliert und validiert werden. Die modernen Erweiterungen der Netztheorie bieten verschiedene Hierarchie- und Zeitkonzepte. Darüber hinaus können modellierte Systeme bei Erhaltung bestimmter wichtiger Verhaltenseigenschaften reduziert und komponiert werden.

Auf der Basis dieser vielfältigen Anwendbarkeit der Netztheorie wird in dieser Arbeit eine Hardware-Entwurfsmethodik vorgeschlagen, die in einen konventionellen Entwurfsablauf eingebettet ist und dennoch eine petrinetz-basierte Methodik darstellt. Da Petrinetze ereignisbasiertes Systemverhalten explizit repräsentieren, ermöglicht diese Hardware-Entwurfsmethodik den Entwurf synchroner *und* asynchroner digitaler Systeme. Asynchrone Systeme besitzen keinen globalen Takt. Die Informationsverarbeitung ist in asynchronen Systemen datenabhängig. Das kann sich positiv auf den Leistungsverbrauch und auf die durchschnittliche Verarbeitungsleistung auswirken (Hau95, Brz95). Bisher ist der Entwurf synchroner Hardware sehr dominant, da Entwurf und Test getakteter Systeme leicht zu handhaben sind. Das Interesse am Entwurf asynchroner digitaler Systeme und Systemkomponenten wächst zunehmend auch im industriellen Bereich, da komplexe digitale Systeme zukünftig mehr Nebenläufigkeit und mehr Asynchronität aufweisen werden (YGL00, ITR01).

Am Lehrstuhl für Rechnerarchitektur und -kommunikation der Friedrich-Schiller-Universität Jena wurde ein Forschungskonzept für ein Entwurfssystem zur petrinetz-basierten Umsetzung asynchroner und synchroner digitaler Systeme in rekonfigurierbare Hardware erarbeitet (RS98, ERS99). Das Entwurfssystem ist in drei Schwerpunkte untergliedert:

Schwerpunkte des Projektes:

1. Modellierung und Analyse digitaler Systeme mit interpretierten Petrinetzen
2. Technologieabhängige Abbildung digitaler Systeme auf rekonfigurierbare Hardware
3. Rekonfigurierbare Hardwarearchitektur für die Implementierung asynchroner digitaler Systeme

Der erste Schwerpunkt ist Gegenstand dieser Arbeit. Folgend werden die einzelnen Kapitel umrissen. **Kapitel 2** gibt eine kurze Einführung zur Theorie der Petrinetze, sowie zu deren Nutzung als Modellierungs- und Analysewerkzeug. In **Kapitel 3** wird ein Abriss zu Methoden des Hardwareentwurfs mit Petrinetzen, basierend auf verschiedenen Netzinterpretationen gegeben. **Kapitel 4** führt die Hardwareentwurfsmethodik PNDes ein und beschreibt einen petrinetz-basierten Entwurfsablauf, der in einem konventionellen Entwurfsablauf eingebettet ist. Danach werden die Entwurfsabschnitte Modellierung und Verifikation in den **Kapiteln 5 und 6** ausgearbeitet. Eine Modellierungsstrategie legt den Modellierungsumfang und die Netzinterpretation fest. Es werden Transformationen zwischen VHDL-Konstrukten und Petrinetz-Strukturen vorgeschlagen, die eine textuelle Modelleingabe und eine Umsetzung verifizierter und optimierter Netzmodelle in synthetisierbare VHDL-Konstrukte auf Register-Transfer-Ebene ermöglichen. Zur Handhabung komplexer Petrinetz-Modelle wird die Anwendung von Hierarchiekonzepten und höheren Netzklassen diskutiert. Schwerpunkt der Arbeit ist das **Kapitel 6**. Nach der Definition von Bedingungen und Zielen der formalen Verifikation mit Petrinetzen wird die Eigenschaftsverifikation digitaler Systeme beschrieben. Die Anwendung einer Reihe von Verifikationsalgorithmen zur Prüfung bestimmter Eigenschaften des Petrinetz-Modells erzielt Aussagen, die geeignet für die formale Verifikation des modellierten digitalen Systems interpretierbar sind. Ziel ist es, eine Gesamtaussage zur formalen Verifikation auf funktionaler Modellebene abzuleiten. Für verifizierte Petrinetzmodelle werden Optimierungsstrategien eingeführt, die sich positiv auf den Hardwareaufwand des zu entwerfenden digitalen Systems auswirken. In den Kapiteln 5 und 6 dient ein Anwendungsbeispiel der Illustration der Entwurfsmethodik. **Kapitel 7** stellt das Petrinetz-Werkzeug VeriCon vor. VeriCon ist eine prototypische Realisierung der Entwurfsmethodik PNDes.

Kapitel 2

Petrinetze

2.1 Definitionen und Netzklassen

Die Theorie der Petri-Netze wurde in der Dissertation von C. A. Petri (Pet62) postuliert. Heute ist der Begriff Petrinetze ein kollektiver Term, der unterschiedliche Modellierungs- und Analysemethoden, basierend auf bestimmten Gesichtspunkten der Informationsverarbeitung vereint. Grundlegend für das Verständnis von Petrinetzen ist die Unterscheidung in *lokale Zustände* und *lokale Aktionen*, sowie deren Wechselwirkung. Lokale Zustände und lokale Aktionen sind gleichwertige Konstituenten eines Petrinetzes. Lokale Aktionen werden im Allgemeinen nur von einer Untermenge aller lokalen Zustände beeinflusst (RR98). In den meisten Fällen wird als grafische Repräsentation eines lokalen Zustandes ein Kreis und als grafische Repräsentation einer lokalen Aktion ein Rechteck gewählt. Die Wechselwirkung der Konstituenten wird grafisch durch Pfeile dargestellt, wobei ein lokaler Zustand ausschließlich durch Wechselwirkung mit einer lokalen Aktion einen anderen oder denselben lokalen Zustand beinflussen kann. Ebenso kann eine lokale Aktion nicht unmittelbar andere lokale Aktionen beeinflussen. Die dynamische Wechselwirkung von Aktionen und Zuständen wird durch Marken repräsentiert, die, dem grafischen Konzept folgend, auf Plätzen des Petrinetzes abgelegt werden. Markierte Plätze eines Petrinetzes symbolisieren aktive Zustände.

Definition 2.1 Petrinetz : Ein Petrinetz N ist ein 4-Tupel $\{P, T, F, m_0\}$ mit

(i) Transitionenmenge T mit $t_i \in T,\ t_i = t_1, t_2, \ldots, t_{|T|}$

(ii) Platzmenge P mit $p_j \in P,\ p_j = p_1, p_2, \ldots, p_{|P|},\ P \cap T = \emptyset,\ P \cup T \neq \emptyset$

(iii) Kantenmenge F mit $f_k \in F,\ f_k = f_1, f_2, \ldots, f_{|F|},\ F \subseteq (P \times T) \cup (T \times P),$
$F \cup (P \times P) = F \cup (T \times T) = \emptyset$

(iv) Vorbereich von $x,\ x \in P \cup T:\ \bullet x = \{y \in P \cup T \mid (y, x) \in F\}$

(v) Nachbereich von $x,\ x \in P \cup T:\ x \bullet = \{y \in P \cup T \mid (x, y) \in F\}$

(vi) Initialmarkierung $m_0 : P \to \mathbb{N},$ mit $m_0, m_1, \ldots m_l \in m.$

Mithilfe von Markierungen kann eine einfache Klassifizierung erfolgen.

Klassifikation von Petrinetzen:

1. Petrinetze mit ununterscheidbaren Marken und
   (a) maximal einer Marke pro Platz
   (b) einer endlichen Anzahl Marken pro Platz
2. Petrinetze mit unterscheidbaren Marken

Fall 1a) beschreibt Bedingungs-Ereignis-Systeme, bei denen Zustände Bedingungen und Aktionen Ereignisse genannt werden (GLT80). Charakteristisch für Fall 1b) sind Platz/Transitionen-Netze (Rei87). Die letzte Klassifikation fasst Netzsysteme zusammen, die auch höhere Petrinetze (High Level Petri Nets = HLPN) genannt werden. Bekannte Varianten höherer Petrinetze sind Prädikat/Transitionen-Netze (Pr/T-Netze) (GL81) und gefärbte Petrinetze (Coloured Petri Nets = CPN) (Jen91). Darüber hinaus existieren viele Netzvarianten, bei denen entweder die Konstituenten nicht gleichwertig sind oder die Flussrelation abweichend definiert wird. In dieser Arbeit werden gewöhnliche Platz/Transitionen-Netze und deren strukturelle Subklassen (Definition 2.8-2.10) angewandt.

Definition 2.2 Platz/Transitionen-Netz : Ein Petrinetz N heißt Platz/Transitionen-Netz (P/T-Netz), wenn

(i) Platzkapazität $K: P \to \mathbb{N} \setminus \{\emptyset\}$

(ii) Kantengewicht $W: F \to \mathbb{N} \setminus \{\emptyset\}$.

Definition 2.3 gewöhnliches Platz/Transitionen-Netz : Ein Platz/Transitionen-Netz N heißt gewöhnlich, wenn

(i) Platzkapazität $K = \infty,\ \forall p_j \in P$

(ii) Kantengewicht $W = 1,\ \forall f_k \in F$.

Definition 2.4 Transitionsaktivierung gewöhnlicher P/T-Netze: Die Transition t_i ist bei Markierung m_l aktiviert $m_l \mid t_i\rangle$, wenn

$$\forall p_j \in {\bullet t_i}: \mid m_l(p_j) \mid > 0.$$

Definition 2.5 Schaltsequenz: Die Sequenz von Transaktionen, auch Schaltsequenz oder Schaltschritt genannt, ist eine Folge von Markierungen

$$m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \ldots \xrightarrow{t_i} m_l = m_0 \mid \sigma\rangle\, m_l, \quad \text{mit } \sigma = t_1, t_2, \ldots, t_i.$$

Definition 2.6 Schaltstrategie: Für beliebige Übergänge $m_l \mid \sigma\rangle\, m_l',\ \ \sigma = t_1, t_2, \ldots, t_i$

(i) schalten alle zum Zeitpunkt t aktivierten Transitionen in einem maximalen Schaltschritt.

(ii) schaltet zum Zeitpunkt t immer nur eine aktivierte Transition.

In dieser Arbeit wird ausschließlich die Schaltstrategie (ii) angewandt.

Definition 2.7 Markierung eines gewöhnlichen P/T-Netzes: Der Übergang zwischen zwei Netzmarkierungen $m_l \mid t_i\rangle\, m_l'$ erfolgt mit

$$\mid m_l'(p_j) \mid \ = \ \mid m_l(p_j) \mid -1, \text{ wenn } [(p_j \in \bullet\, t_i) \ \wedge \ (p_j \neq\in t_i \bullet)]$$
$$\mid m_l'(p_j) \mid \ = \ \mid m_l(p_j) \mid +1, \text{ wenn } [(p_j \in t_i \bullet) \ \wedge \ (p_j \neq\in \bullet\, t_i)]$$
$$\mid m_l'(p_j) \mid \ = \ \mid m_l(p_j) \mid \text{ sonst.}$$

Eine sehr nützliche strukturelle Subklasse der gewöhnlichen P/T-Netze sind die ursprünglich in (Hac72) vorgeschlagenen und in (RT86) verallgemeinerten Free-Choice-Netze (FCPN).

Definition 2.8 Free-Choice-Netz : Ein gewöhnliches Platz/Transitionen-Netz N heißt Free-Choice-Netz, wenn

$$\forall p_j \in P,\ \forall t_i \in T: \ \ [\, p_j \bullet = \{t_i\} \ \vee \ \bullet\ t_i = \{p_j\}\,].$$

Free-Choice-Netze setzen sich aus den beiden strukturell einfachen Petrinetzklassen Synchronisationsgraph (Marked Graph - MG) (CHEP71) und Zustandsmaschine (State Machine = SM) (Hac72) zusammen.

Definition 2.9 Synchronisationsgraph: Ein Free-Choice-Netz N heißt Synchronisationsgraph, wenn

$$\forall p_j \in P: \quad \mid \bullet\, p_j \mid \ = \ \mid p_j \bullet \mid \ = \ 1.$$

Definition 2.10 Zustandsmaschine : Ein Free-Choice-Netz N heißt Zustandsmaschine, wenn

$$\forall t_i \in T: \quad \mid \bullet t_i \mid \ = \ \mid t_i \bullet \mid \ = \ 1.$$

Mithilfe der in diesem Abschnitt angegebenen Definitionen werden nun die Möglichkeiten der Modellierung und Analyse von Systemen und Abläufen mit Petrinetzen aufgezeigt.

2.2 Petrinetze als Modellierungswerkzeug

Der Anwendungsbereich von Petrinetzen als Modellierungswerkzeug erstreckt sich von technischen Systemen (digitale Hardware, Produktions- und Verkehrssysteme) über betriebliche Informationssysteme (Workflow Management Systeme) bis hin zu informatisch/mathematischen Systemen (Software, Prozessalgebren) (RE98). Petrinetz-Modelle werden für folgende Aufgaben eingesetzt:

(a) Validierung: Konzeptprüfung beim Systementwurf

(b) Simulation: Test des Petrinetz-Modells durch sukzessive Ablaufsimulation

(c) Analyse: Untersuchung von Struktur und Verhalten des Petrinetz-Modells

(d) Verifikation: Eigenschaftsprüfung des Petrinetz-Modells.

In den Kapiteln 4 bis 7 wird auf jede der Aufgaben detailliert anhand des Entwurfs digitaler Systeme eingegangen.

Grundsätzlich ist der Prozess der Modellierung ein Abstraktionsprozess, bei dem wesentliche Aspekte des betrachteten Systems und seines Verhaltens in einem Modell widergespiegelt werden. Somit werden die Modellierungmöglichkeiten mit Petrinetzen in zwei Kategorien gefasst. Einerseits können verschiedenste Systeme auf unterschiedlichen Abstraktionsebenen in ihrer Struktur modelliert werden. Des Weiteren ist es möglich, das Systemverhalten abstrakt als Systemablauf (Des98b) in Form von sequentiellen und nebenläufigen Abläufen zu modellieren.

Mit den in Abschnitt 2.1 definierten strukturellen Netzvarianten der Platz/Transitionen-Netze lassen sich hervorragend sequentielle und nebenläufige Abläufe modellieren. Die Zustandsmaschine ermöglicht mit den Strukturen der Verzweigung und Begegnung die einfache Modellierung sequentieller Abläufe (Abbildung 2.1). Gleichermaßen werden nebenläufige Abläufe mit den beiden Strukturen des Markierten Graphen, Aufspaltung und Synchronisation, modelliert. Free-Choice-Netze vereinen beide Netzklassen und folglich beide Modellierungsvarianten. Aus diesem Grund werden Free-Choice-Netze in dieser Arbeit als *Netzspezifikation* zur Modellierung digitaler Systeme ausgewählt.

Neben der Modellierung von Abläufen, die Systemverhalten widerspiegeln, ist eine strukturale Systemmodellierung von großem Nutzen. Ein strukturales Systemmodell beschreibt die direkte Abbildung der Systemspezifikation in ein Petrinetz-Modell. Diese Abbildung leistet eine *Netzinterpretation*, die Zuordnungen zwischen Petrinetz-Strukturen und Spezifikations-Konstrukten definiert. Auf Netzinterpretationen und deren Anwendung wird in den Abschnitten 3.1 und 5.1 detailliert eingegangen.

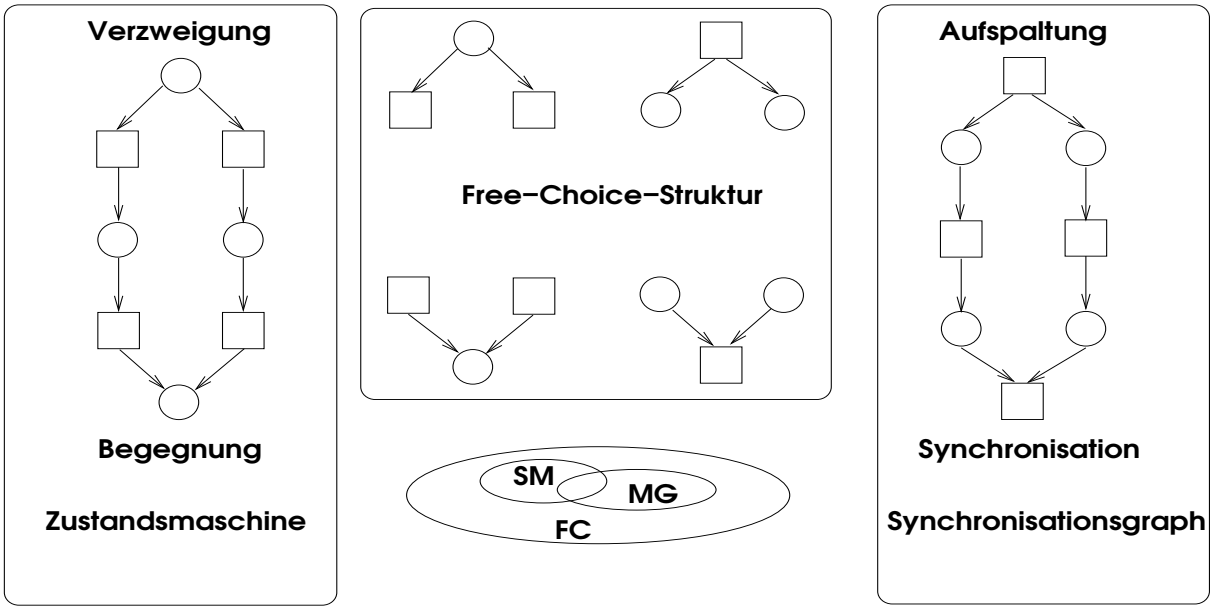

Abbildung 2.1: Modellierung mit Free-Choice-Netzen

Zusammenfassend bieten Petrinetz-Modelle die Möglichkeit, Struktur und Verhalten eines Systems in *einem* Modell wiederzugeben, welches validiert, simuliert, analysiert und verifiziert werden kann.

Im Gegensatz zur Automatentheorie wird der Systemzustand nicht ausschließlich als globaler Zustand beschrieben (Sta90). Durch die lokale Wirkung der Aktionen auf Zustände wird eine ereignisbasierte Modellsicht ermöglicht, wobei kausal abhängige und kausal unabhängige Ereignisse erfasst werden.

2.3 Analyse von Petrinetz-Modellen

Petrinetze offerieren sehr gute Möglichkeiten der Modellanalyse. Die im Petrinetz-Modell enthaltenen Struktur- und Verhaltensmerkmale einer Systemspezifikation können anhand bestimmter Eigenschaften analysiert werden. Diese Eigenschaften sind entweder unabhängig oder abhängig von der Initialmarkierung m_0 und werden entsprechend Stuktureigenschaften oder Verhaltenseigenschaften genannt. Für die Analyse von Petrinetz-Modellen wird die gewählte Netzspezifikation (Definition 2.8) vorausgesetzt. Beweise zu angeführten Sätzen sind in (Mur89, Sta90) enthalten.

2.3.1 Struktureigenschaften

Mit den in Abschnitt 2.1 definierten Netzklassen und einigen einfachen Strukturbegriffen ist die Struktur von Petrinetz-Modellen sehr gut analysierbar (Sta90). Struktureigenschaften von Petrinetz-Modellen haben direkte Auswirkungen auf die Dynamik der Netzmodelle.

Definition 2.11 Starker Zusammenhang : Ein Petrinetz N heißt stark zusammenhängend, wenn

$$\forall x, y \in (P \cup T) : \; x \; F^* \; y.$$

Bemerkung: F^* bezeichnet die reflexiv transitive Hülle der Kantenmenge F.
$F^* = (x, y)$, mit: $x = \; a_1 \; F \; a_2 \; F \; a_3 \; F \; \ldots \; F \; a_n \; = y, \;\; a_1, \ldots, a_n \in (P \cup T)$. F^* beschreibt somit jeden gerichteten Weg in N.

Definition 2.12 Zusammenhang : Ein Petrinetz N heißt zusammenhängend, wenn

$$\forall x, y \in (P \cup T) : \; (x, y) \in (F \cup F^{-1})^*.$$

Bemerkung: $(F \cup F^{-1})^*$ beschreibt jeden Weg in N ungeachtet der Kantenrichtung.

Definition 2.13 Reinheit/Schleifenfreiheit : Ein Petrinetz N heißt schleifenfrei, wenn

$$\forall t_i \in T : \; \bullet \; t_i \; \neq \; t_i \; \bullet.$$

Definition 2.14 Quelle : Ein Petrinetz N besitzt eine

(i) Transitionsquelle $Ft0$, wenn $\exists \, t_i \in T : \; \bullet \; t_i \; = \emptyset$

(ii) Platzquelle $Fp0$, wenn $\exists \, p_j \in P : \; \bullet \; p_j \; = \emptyset$.

Definition 2.15 Senke : Ein Petrinetz N besitzt eine

(i) Transitionssenke $tF0$, wenn $\exists \, t_i \in T : \; t_i \; \bullet \; = \emptyset$

(ii) Platzsenke $pF0$, wenn $\exists \, p_j \in P : \; p_j \; \bullet \; = \emptyset$.

Definition 2.16 Statische Konfliktfreiheit : Ein Petrinetz N heißt statisch konfliktfrei, wenn

$$\forall p_j \in P : \; \mid p_j \; \bullet \mid \; \leq \; 1.$$

Bemerkung: Für $\mid p_j \; \bullet \mid > \; 1$ besteht die Möglichkeit einer Markierung, die mehrere Transitionen im Nachbereich von p_j aktiviert. Diese stehen dann in Konflikt zueinander.

Definition 2.17 Konservativität : Ein gewöhnliches Petrinetz N heißt konservativ, wenn

$$\forall t_i \in T: \ \mid \bullet t_i \mid \ = \ \mid t_i \bullet \mid.$$

Bemerkung: Konservativität beschreibt die Erhaltung der Markenanzahl im Netz N. Bei einem Kantengewicht von $W = 1, \ \forall f_k \in F$ ist die Konservativität durch das Gleichgewicht der Vielfachheiten von Aufspaltung und Synchronisation im Netz bestimmt.

2.3.2 Verhaltenseigenschaften

Verhaltenseigenschaften charakterisieren die Dynamik eines Petrinetz-Modells. Von zentraler Bedeutung sind dabei die Begriffe Beschränktheit und Lebendigkeit, wobei sich Beschränktheit auf Plätze und Lebendigkeit auf Transitionen bezieht. Beide Eigenschaften stehen in engem Zusammenhang mit dem Begriff der Erreichbarkeit von Markierungen.

Definition 2.18 Erreichbarkeit und Erreichbarkeitsgraph: Eine Markierung m_l heißt erreichbar, wenn eine Schaltsequenz

$m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \ \ldots \ \xrightarrow{t_i} m_l \ = \ m_0 \mid \sigma\rangle \, m_l$, mit $\sigma = t_1, t_2 \ldots t_i$ existiert.
Die Menge aller von m_0 erreichbaren Markierungen bildet den Erreichbarkeitsgraphen $R_n(m_0)$.

Bemerkung: $R_n(m_0)$ besteht aus Knoten und gerichteten Kanten, die mit Markierungen und Transitionen korrespondieren.

Definition 2.19 Beschränktheit : Ein Petrinetz N heißt beschränkt, wenn

$$\forall p_j \in P: \ \mid m_l(p_j) \mid \ \leq \ n, \quad n \in \mathbb{N}.$$

Definition 2.20 1-Beschränktheit/Sicherheit : Ein Petrinetz N heißt 1-beschränkt oder sicher, wenn

$$\forall p_j \in P: \ \mid m_l(p_j) \mid \ \leq \ 1.$$

Definition 2.22 Lebendigkeit :

(i) t_i heißt lebendig bei m_l, wenn $\nexists \, m_l'$ mit $m_l \mid \sigma\rangle \, m_l' : \ t_i$ ist nicht aktivierbar bei m_l'

(ii) m_l heißt lebendig, wenn $\forall t_i \in T: \ m_l \mid t_i\rangle$

(iii) N heißt lebendig, wenn $\forall t_i \in T: \ m_0 \mid t_i\rangle$.

Definition 2.23 Reversibilität : Ein Petrinetz N heißt reversibel, wenn

R_n ist stark zusammenhängend.

Definition 2.24 Nebenläufigkeit : Zwei Transitionen t_{i_1}, t_{i_2} eines Petrinetzes N heißen nebenläufig bei einer Markierung m_l, wenn

$$m_l \mid t_{i_1} t_{i_2}\rangle \ \wedge \ m_l \mid t_{i_2} t_{i_1}\rangle.$$

Bemerkung: Nebenläufigkeit charakterisiert die Unabhängigkeit der Schaltreihenfolge mehrerer Transitionen bei gegebener Markierung.

Definition 2.25 Dynamische Konfliktfreiheit : Ein Petrinetz N heißt dynamisch konfliktfrei oder persistent, wenn

$$\forall t_i \in T, \ \forall m_l \in R_n(m_0) : \ m_l \mid t_{i_1} t_{i_2}\rangle \ \wedge \ m_l \mid t_{i_2} t_{i_1}\rangle.$$

Bemerkung: Bei jeder erreichbaren Markierung können zwei aktivierte Transitionen $t_{i_1} t_{i_2}$ in beliebiger Reihenfolge schalten.

Definition 2.26 Deadlock-Trap-Eigenschaft :

(i) Ein Deadlock $P_D, \ P_D \subseteq P$ ist eine Platzmenge mit $\bullet P_D \subseteq P_D \bullet$.

(ii) Ein Trap $P_T, \ P_T \subseteq P$ ist eine Platzmenge mit $P_T \bullet \subseteq \bullet P_T$.

(iii) Ein Free-Choice-Netz N ist lebendig $\Longleftrightarrow$ Bei der Initialmarkierung m_0 enthält jeder Deadlock P_D einen markierten Trap P_T.

Bemerkung: Jede Transition t_i, die der Platzmenge P_D eine Marke hinzufügt, entnimmt auch eine Marke aus P_D. Jede Transition t_i, die eine Marke aus P_T entnimmt, addiert zu P_T eine Marke hinzu (Abbildung 2.2).

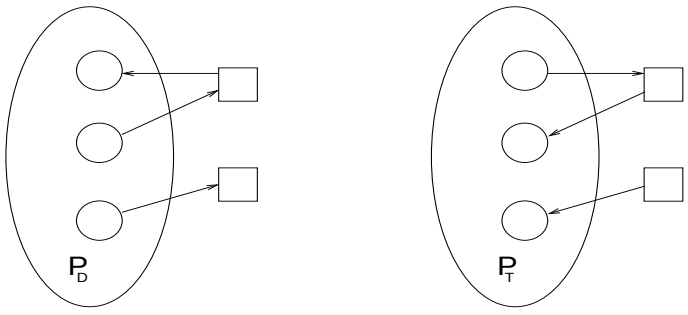

Abbildung 2.2: Deadlock-Trap-Eigenschaft

Für strukturelle Subklassen der Free-Choice-Netze lassen sich aus den Struktureigenschaften und Annahmen zur Initialmarkierung in einfacher Weise die grundlegenden Verhaltenseigenschaften ableiten (Mur89).

Kriterien für Lebendigkeit und Sicherheit bei Subklassen von Free-Choice-Netzen

Satz 2.1 Lebendigkeit bei Zustandsmaschinen : Eine Zustandsmaschine SM heißt lebendig $\Longleftrightarrow$

SM ist stark zusammenhängend und $\mid m_0 \mid > 0$.

Satz 2.2 Sicherheit bei Zustandsmaschinen : Eine Zustandsmaschine SM heißt sicher $\Longleftrightarrow$

SM ist stark zusammenhängend und $\mid m_0 \mid \leq 1$.

Folgerung 2.1 : Eine Zustandsmaschine SM heißt lebendig und sicher $\Longleftrightarrow$

SM ist stark zusammenhängend und $\mid m_0 \mid = 1$.

Satz 2.3 Lebendigkeit bei Synchronisationsgraphen : Ein Synchronisationsgraph MG heißt lebendig $\Longleftrightarrow$

Jeder gerichtete Kreis in MG enthält mindestens eine Marke bei m_0.

Satz 2.4 Sicherheit bei Synchronisationsgraphen : Ein Synchronisationsgraph MG heißt sicher $\Longleftrightarrow$

Jeder gerichtete Kreis in MG enthält höchstens eine Marke bei m_0.

Folgerung 2.2 : Ein Synchronisationsgraph MG heißt lebendig und sicher $\Longleftrightarrow$

Jeder gerichtete Kreis in MG enthält genau eine Marke bei m_0.

Zustandsgleichung und Invariantenanalyse

In Analogie zur Analyse der Dynamik physikalischer Systeme mit Differentialgleichungssystemen ist das Aufstellen der Zustandsgleichung eines Petrinetzes und die Berechnung von Platz- und Transitions-Invarianten eine Möglichkeit der Analyse dynamischer Netzeigenschaften. Netzinvarianten werden mit den Methoden der linearen Algebra

berechnet und lassen Schlussfolgerungen auf Verhaltenseigenschaften zu. Dabei werden ausschließlich Informationen zur Netzstruktur und zur Initialmarkierung benötigt (Des98a).

Definition 2.27 Inzidenzmatrix : Die Inzidenzmatrix eines Petrinetzes wird aus der Folge der Transitionsvektoren $\overrightarrow{t_i}$ gebildet. Für ein gewöhnliches Netz gilt

Inzidenzmatrix: $\mathbf{I} = (\overrightarrow{t_1}, \overrightarrow{t_2}, \ldots, \overrightarrow{t_i})$
$\overrightarrow{t_i} = (t_{i1}, t_{i2}, \ldots, t_{ij}),\ i = 1 \ldots \mid T \mid,\ j = 1 \ldots \mid P \mid$
$t_{ij} = -1$, wenn $(p_j, t_i) \in F$
$t_{ij} = 1$, wenn $(t_i, p_j) \in F$
$t_{ij} = 0$, sonst.

Definition 2.28 Zustandsgleichung :

Initialmarkierungsvektor: $\overrightarrow{m_0} = (m_{0_{p_1}}, m_{0_{p_2}}, \ldots, m_{0_{p_j}})$
Schalthäufigkeitsvektor: $\overrightarrow{k} = (k_{t_1}, k_{t_2}, \ldots, k_{t_i})$
Zustandsgleichung: $\mathbf{m} = \mathbf{m_0} + \mathbf{I} \cdot \mathbf{k}$.

Bemerkung: Der Schalthäufigkeitsvektor **k**, auch Parikh-Vektor genannt, ist eine Abbildung $T \to \mathbb{N}$, die die Schalthäufigkeit von t_i als k_{t_i} angibt.

Definition 2.29 P-Invariante : Sei $\mathbf{I}$ die Inzidenzmatrix des Petrinetzes N. Dann heißt

(i) jede nicht-triviale ganzzahlige Lösung des Gleichungssystems $\mathbf{y} \cdot \mathbf{I} = 0$ P-Invariante von N.

(ii) $\mathbf{y}$ echte P-Invariante, wenn $\mathbf{y} \geq 0$.

(iii) N von P-Invarianten überdeckt $\iff \forall p_j \in P\ \exists\, \mathbf{y} :\ y(p_j) > 0$.

Definition 2.30 T-Invariante : Sei $\mathbf{I}$ die Inzidenzmatrix des Petrinetzes N. Dann heißt

(i) jede nicht-triviale ganzzahlige Lösung des Gleichungssystems $\mathbf{I} \cdot \mathbf{x} = 0$ T-Invariante von N.

(ii) $\mathbf{x}$ echte T-Invariante, wenn $\mathbf{x} \geq 0$.

(iii) N von T-Invarianten überdeckt $\iff \forall t_i \in T\ \exists\, \mathbf{x} :\ x(t_i) > 0$.

Satz 2.5 Überdeckung mit P-Invarianten und Beschränktheit: N ist mit P-Invarianten überdeckt $\Rightarrow$

N ist bei beliebiger Anfangsmarkierung beschränkt.

Kapitel 3

Digitale Systeme und Petrinetze

3.1 Interpretierte Petrinetze

Werden Petrinetze zur Modellierung von Systemstrukturen und deren dynamischen Verhalten eingesetzt, so ist eine Abbildung von Netzkomponenten auf Systemkomponenten notwendig. Dieser Prozess wird als Netzsynthese bezeichnet, da ein Petrinetz-Modell auf Komponenten eines technischen Systems abgebildet wird. In (Sta80) werden Beispiele solcher Abbildungen für Produktionssysteme und Schaltwerke dargestellt. Entsprechend der gewählten Abstraktionsebene kann die Netzsynthese einer Systemsynthese, einer algorithmischen Synthese oder einer Technologieabbildung entsprechen.

Grundlage jeder Netzsynthese sind wohldefinierte Abbildungen, die unter dem Begriff Netzinterpretation zusammengefasst werden. In einer Netzinterpretation ist festgeschrieben, welche Petrinetz-Konstituenten welchen Systemstrukturen zugeordnet werden. Die Abbildung kann auch zwischen aus Petrinetz-Konstituenten zusammengesetzten Teilnetzen und Systemstrukturen definiert werden. Darüber hinaus legt eine Netzinterpretation fest, ob und wie die Wechselwirkung eines Systems mit seiner Umgebung im Petrinetz-Modell Ausdruck findet.

Beim Vergleich von Netzinterpretationen wird klar, dass mit interpretierten Petrinetzen in erster Linie ein spezifischer anwendungsbezogener Zugang bei der Modellierung von Systemen mit Petrinetzen in verschiedenen Anwendungsbereichen geschaffen werden soll. Einerseits führt das, verglichen mit nicht anwendungsspezifischen Petrinetz-Modellen, zur Verringerung des Abstraktionsvermögens. Interpretierte Petrinetze sind im Allgemeinen auf ein bestimmtes Anwendungsgebiet festgelegt. Genau dieser Punkt ist allerdings von großem Nutzen für die Realisierung einer transparenten Netzsynthese, bei der das Petrinetz-Modell äquivalent eine spezifische Systemstruktur und spezifisches Systemverhalten wiedergeben soll. Für den Entwurf eines technischen Systems ist eine derart genaue Abbildung unabdingbar.

Die Frage, inwiefern ein Petrinetz-Modell Systemstrukturen widerspiegeln muss, stellt sich erneut, wenn das Petrinetz-Modell nicht nur zur Systemvalidation und -simulation, sondern auch zur Systemanalyse und -verifikation eingesetzt werden soll. Netzinterpretationen weichen von reinen Netzspezifikationen (Abschnitt 2.1) ab. Die Methoden zur Analyse von Petrinetz-Modellen können jedoch nicht oder nur sehr eingeschränkt angewandt werden, wenn zum Beispiel Konstituenten des Petrinetzes nicht gleichwertig sind oder wenn für unterschiedliche Konstituenten verschiedene Aktivierungs- und Schaltregeln definiert wurden. Die Gewährleistung von Analysierbarkeit und formaler Verifizierbarkeit mit den Methoden und Eigenschaften aus Kapitel 2 verlangt, dass das Petrinetz-Modell keine die Netzspezifikation übersteigenden Netzerweiterungen enthält. Auf der Seite der Modellierung bedeutet das wiederum eine maximale Abstraktion und eine für den Systementwurf zu geringe Transparenz in der Netzsynthese.

Es ist ein Anliegen dieser Arbeit, den geschilderten Widerspruch von systemnaher formaler Beschreibung und formaler Verifizierbarkeit aufzuheben. Im Kapitel 4 wird dazu eine Methodik vorgestellt, mit deren Hilfe transparente Petrinetz-Modelle adäquat analysiert und formal verifiziert werden können.

3.2 Modellierung digitaler Systeme mit Petrinetzen

Der Entwurf digitaler Systeme ist seit mehreren Jahrzehnten von Beschreibungsmodellen und -sprachen dominiert, die in erster Linie sequentielles, synchron getaktetes Hardware-Verhalten darstellen. Ein bekanntes Beispiel sind endliche Automaten, die als Beschreibungsmodell in Hardware-Beschreibungssprachen zur Verfügung gestellt werden.

Seit einiger Zeit ermöglicht die weiterhin exponentiell ansteigende Integrationsdichte von Bauelementen in CMOS-Technologie den Entwurf von System-on-Chip-Systemen, welche u.a. Prozessoren, rekonfigurierbare Einheiten, Speichereinheiten und Ein-/Ausgabecontroller als Subsysteme besitzen. Der Informationsaustausch zwischen den Subsystemen wird immer aufwendiger, wenn ein globaler Takt zugrunde gelegt wird, der im gesamten SoC-System verteilt werden muss. Darüber hinaus ist die Leistungsaufnahme synchroner Systeme potentiell höher, verglichen mit asynchronen Systemen. Besonders für pervasive, mobile Anwendungen ist dieser Punkt außerordentlich wichtig, da die Entwicklung mobiler Energiespeicher nicht so schnell voranschreitet wie die Komplexität und der Leistungsbedarf integrierter mobiler Systeme. Eine ausschließlich globale Zeitnotation der digitalen Verarbeitung von Informationen ist somit nicht mehr sinnvoll. Der Entwurf von asynchronen und nebenläufigen Systemen ist eine aktuelle Herausforderung, die gemeistert werden kann, wenn Beschreibungsmethoden angewandt werden, die diese beiden Phänomene explizit modellieren.

Wie in Kapitel 2 gezeigt wurde, bieten Petrinetze die Möglichkeit, nebenläufige und sequentielle Prozesse abzubilden. Die ereignisbasierte Modellsicht gewährleistet Asynchronität eines modellierten digitalen Systems als allgemeine Annahme. Es existieren mehrere Ansätze für die Modellierung digitaler Systeme mit Petrinetzen.

Den Anfang machte das Projekt MAC am Massachusetts Institute of Technology. Dort wurde basierend auf der Theorie der geschwindigkeits-unabhängigen Schaltungen von *D.E.Muller* und *W.C.Bartky* (MB59) der petrinetz-basierte Entwurf digitaler Schaltungen beschrieben (Pat70, DP71, PD72). Die Vorschläge von *S.S.Patil* und *J.B.Dennis* weisen den Petrinetz-Konstituenten und Petrinetz-Konstrukten, wie Aufspaltung, Verzweigung, Begegnung und Synchronisation, Schaltungskomponenten zu, um geschwindigkeits-unabhängige Kontrollstrukturen zu modellieren und zu implementieren. Die aus dieser Netzinterpretation folgende Netzsynthese ist auch als Synthese auf elementarem Niveau bekannt. *D.Misunas* führt in (Mis73) diesen Ansatz weiter und definiert komplexe Module, die eine petrinetz-basierte Beschreibung geschwindigkeits-unabhängiger Systeme auf Register-Transfer-Ebene ermöglichen.

Eine Reihe weiterer Arbeiten entstand in den 90er Jahren im Bereich elektronische Entwurfsautomation (Lav92, PCKR98, CKK+00). Ziel ist wiederum der Entwurf asynchroner Schaltungen, wobei als Netzinterpretation die in (RY85, Chu87) vorgeschlagenen Signal Transitionen Graphen (STG) dienen. Die STG-Interpretation modelliert asynchrone Schaltungen als nebenläufige Systeme, so dass Ereignisse als Signalübergänge interpretierbar sind. *L.Lavagno* stellt in (Lav92) eine Methode der Netzsynthese vor, bei der das klassische Huffmann-Modell (Huf54a, Huf54b) beidseitig beschränkter Verzögerungen von Schaltelementen mit der Interpretation der STG zusammengeführt wird. In (PCKR98) wird eine Netzsynthese für geschwindigkeits-unabhängige Systeme mit der STG-Interpretation beschrieben. Ein Überblick zur Anwendung der STG-Interpretation für den Entwurf asynchroner Systeme wird in (CKK+00) gegeben. Im Rahmen der Projekte zum Entwurf asynchroner Hardware mit der STG-Interpretation entstanden die Werkzeuge SIS (LSV93) und Petrify (CKK+97). Diese und weitere Werkzeuge ermöglichen den praktischen Einsatz der STG-Methoden zur petrinetz-basierten Beschreibung und Realisierung digitaler Systeme bis hin zur industriellen Anwendung (CF00).

Ein Modellierungsansatz, der nicht im Bereich des Hardware-Entwurfs entstand, ist die steuerungstechnische Interpretation eines Petrinetzes (SIPN) (KQ88). Dieser Ansatz unterscheidet zwischen der Hardware-Interpretation und der Software-Interpretation eines Petrinetzes für den Entwurf von Steuerungen. Bei der hardware-orientierten Interpretation werden den Netzkonstituenten Schaltelemente der Zielarchitektur zugeordnet. Entsprechend der gewählten Zielarchitektur lassen sich mit der hardware-orientier-

ten Interpretation synchron getaktete oder asynchrone Steuerungen entwerfen. Die software-orientierte Interpretation weist Netzkonstituenten und Netzkonstrukten Sprachelemente einer Fachsprache der Steuerungstechnik zu, so dass speicherprogrammierbare Steuerungen (SPS) auf diesem Wege entwerfbar sind. In (KQ88) wird eine allgemeine Netzinterpretation wie folgt definiert:

Definition 3.1 Interpretiertes Petrinetz nach (KQ88) : Ein interpretiertes Petrinetz IPN ist ein 3-Tupel $\{N, q_T, q_P\}$ mit

(i) q_T als Abbildung der Transitionen des Petrinetzes N in Komponenten des zu modellierenden Systems oder Sachverhaltes.

(ii) q_P als Abbildung der Plätze des Petrinetzes N in Komponenten des zu modellierenden Systems oder Sachverhaltes.

Beim Vergleich von STG-Interpretation und steuerungstechnischer Netzinterpretation fällt auf, dass die Netzinterpretationen in verschiedenen Anwendungsgebieten und für verschiedene Zielsysteme Anwendung finden. Signal Transitionen Graphen werden hauptsächlich für den Entwurf asynchroner digitaler Schaltungen angewandt, während in der Steuerungstechnik zum großen Teil software-basierte, synchrone Systeme entworfen werden. Allerdings haben beide Methoden den Entwurf von Steuerungen oder Steuereinheiten zum Ziel. An dieser Stelle sei zur Vermeidung von Missverständnissen angemerkt, dass SIPN mit den in (DA92) vorgeschlagen CIPN (Control Interpreted Petri Nets) nichts gemein haben.

In dieser Arbeit wird der Vorschlag der Anwendung der steuerungstechnischen Netzinterpretation für den Entwurf digitaler Systeme (RS98, ERS99) ausgearbeitet und untersucht. In Abhängigkeit von der Zielarchitektur soll es möglich sein, mit einer Hardware-Beschreibung synchrone und asynchrone digitale Systeme zu entwerfen.

3.3 Hardware-Entwurf mit Petrinetzen

Der Einsatz von Petrinetzen als Beschreibungs- und Realisierungsgrundlage im industriellen EDA-Bereich (Electronic Design Automation) erfordert die Unterstützung verschiedener Entwurfsschritte mit geeigneten Werkzeugen. Weiterhin ist es erforderlich, Schnittstellen zwischen Petrinetz-Modell und weit verbreiteten Hardware-Beschreibungssprachen, wie Verilog und VHDL (VHSIC[1] Hardware Description Language), zu schaffen, um damit industrielle Anwendbarkeit zu gewährleisten. Dass dieses Thema auf akademischer und industrieller Seite auf Interesse stößt, zeigt ein in den Jahren 1998 und 1999,

[1] Very High Speed Integrated Circuit

im Rahmen der internationalen Konferenz zur Anwendung und Theorie der Petrinetze, durchgeführter Workshop zum Thema „Hardware-Entwurf und Petrinetze" (YGL00). Hier wurden einige interessante Arbeiten vorgestellt, die kurz umrissen werden sollen.

In (BL00) wird ein Ansatz beschrieben, mit dem Verilog-Konstrukte automatisch in Signal Transitionen Graphen umgesetzt und danach mit dem Werkzeug Petrify in asynchrone Kontrolleinheiten abgebildet werden. Diese Methode ist industriell anwendbar.

(MMB00) gibt einen Überblick zum petrinetz-basierten Entwurf synchroner Kontrolleinheiten. Hier existiert eine Entwurfsmethodik, mit der aus hierarchischen Petrinetz-Modellen paralleler Mikrocontroller synthetisierbare Beschreibungen auf Register-Transfer-Ebene in der Hardware-Beschreibungssprache VHDL gewonnen werden (FAP97). Die dafür verwandte Netzinterpretation heißt synchrone Netzinterpretation. (MFES00) erweitert diese Methodik mit der Beschreibungsmöglichkeit von Datenpfaden, mit einem Hardware/Software Co-Design Partitionierungsansatz und mit dem Vorschlag einer rekonfigurierbaren Zielarchitektur.

Ein petrinetz-basierter Entwurfsfluss für den Entwurf synchroner Systeme wird in (RFH00) beschrieben. Beim Entwurfsansatz ESDA (Electronic System Design Automation) wird ausgehend von einer abstrakten Beschreibung mit gefärbten Petrinetzen ein Hardware-Petrinetz erzeugt, welches in eine synthetisierbare VHDL-Beschreibung auf Register-Transfer-Ebene übersetzt wird.

Eine Bewertung jedes genannten Ansatzes soll an dieser Stelle nicht erfolgen. Wird jedoch die gewählte Netzinterpretation als Ausgangspunkt für das Petrinetz-Modell und für die Anwendbarkeit der jeweils vorgeschlagenen Methodik gesehen, so ist folgendes erkennbar:

- Die Netzinterpretationen beschränken den Anwendungsbereich zu stark, so dass mit diesen Petrinetz-Modellen entweder asynchrone oder synchrone Systeme entwerfbar sind.
- Die Netzinterpretationen enthalten selbstdefinierte Netzkonstituenten, so dass auf formale Modellanalyse und Modellverifikation mittels Eigenschaftsprüfung verzichtet werden muss.

Im folgenden Kapitel wird eine Hardware-Entwurfsmethodik eingeführt, mit der synchrone und asynchrone Systeme beschrieben, simuliert, verifiziert, optimiert, synthetisiert und implementiert werden können. Diese Methodik hat in erster Linie die formale Verifikation digitaler Systeme innerhalb eines petrinetz-basierten Entwurfsflusses zum Ziel.

Kapitel 4

PNDes - Eine petrinetz-basierte Entwurfsmethodik für den Entwurf digitaler Systeme

4.1 Entwurfsmethodik

Der konventionelle Hardware-Entwurfsprozess, welcher seit mehreren Jahrzehnten für den Entwurf digitaler Systeme angewandt wird, lässt sich in die Abschnitte Hardware-Beschreibung, funktionale Simulation, technologieunabhängige Synthese, technologieabhängige Synthese und zeitbehaftete Simulation fassen. Im Gegensatz zum konventionellen Hardware-Entwurfsprozess sind bei der Hardware-Entwurfsmethodik PNDes (Petri Net based Design) Entwurfsschritte derart erweitert und modifiziert, dass eine formale Verifikation auf funktionaler Ebene mit Petrinetzen geleistet wird. Darüber hinaus kann mit einer zeitbehafteten Simulation und Leistungsbewertung basierend auf deterministisch und stochastisch zeitbehafteten Petrinetzen ein synthetisierter Entwurf mit formalen Methoden analysiert werden.

Ziel der petrinetz-basierten Hardware-Entwurfsmethodik PNDes ist eine transparente Abbildung der Systemstruktur und des Systemverhaltens in ein einfaches und überschaubares Petrinetz-Modell, das als unbeschriftetes Petrinetz effizient und vollständig analysierbar ist und durch geeignete Interpretation der Analyseergebnisse formal verifiziert, implementiert und getestet werden kann.

Die signifikanten Vorteile der Petrinetze als Beschreibungs- und Verifikationsbasis des Hardware-Entwurfs werden anhand der folgenden Fakten zusammengefasst (ERS01c).

- Die Fähigkeit, sequentielle und nebenläufige Ereignisse korrekt und explizit zu modellieren.
- Die Fähigkeit, spezielle Verhaltenskonstrukte, wie Aufspaltung, Verzweigung, Begegnung und Synchronisation, korrekt und explizit zu modellieren.
- Die Fähigkeit, Systemstruktur und Systemverhalten mit der Netzmodellstruktur und dem dynamischen Netzverhalten wiederzugeben.
- Die Fähigkeit, digitale Systeme auf verschiedenen Abstraktionsstufen mithilfe von Hierarchiekonzepten zu modellieren.
- Die Fähigkeit, funktionale und zeitbehaftete Simulation des Systemverhaltens explizit grafisch zu realisieren.
- Die Fähigkeit, mit der Analyse von Struktur- und Verhaltenseigenschaften ein Petrinetz-Modell formal zu verifizieren.
- Die Fähigkeit, mit Methoden der Modellprüfung ein Petrinetz-Modell formal zu verifizieren.

Um diese Vorteile beim Entwurf von digitalen Systemen zu nutzen und einen petrinetzbasierten Entwurfsfluss zu kreieren, der anwendbar und beherrschbar ist, müssen zwei Bedingungen erfüllt sein. Zum einen soll eine weit verbreitete Hardware-Beschreibungssprache als Modelleingabemöglichkeit anwendbar sein. Hierzu ist eine Schnittstelle erforderlich, die HDL-Konstrukte in Petrinetz-Strukturen umsetzt. Des Weiteren soll ein konventionelles, nicht petrinetz-basiertes Werkzeug für die technologieabhängige Abbildung des Hardware-Modells auf eine variable Zielarchitektur anwendbar sein. Das erfordert eine Schnittstelle, die Petrinetz-Strukturen in HDL-Konstrukte transformiert.

Konventionelle Hardware-Beschreibungssprachen wie VHDL und Verilog, und Synthesewerkzeuge sind im industriellen Bereich weit verbreitet und somit für jeden Hardware-Entwurfsfluss relevant. Es ist aus Gründen der Anwendbarkeit nicht sinnvoll, einen Entwurfsfluss vorzuschlagen, der vollständig petrinetz-basiert ist. Werden die beiden genannten Bedingungen erfüllt, so ist es möglich, petrinetz-basierte Entwurfsabschnitte in einen konventionellen Entwurfsfluss einzubetten. Dieses Ziel wird mit PNDes verfolgt.

Die Hardware-Entwurfsmethodik PNDes ist auf die ersten beiden Entwurfsschritte Modellierung/Beschreibung und Simulation/Verifikation fokussiert. Der petrinetz-basierte Entwurfsprozess ist in Abbildung 4.1 schematisiert.

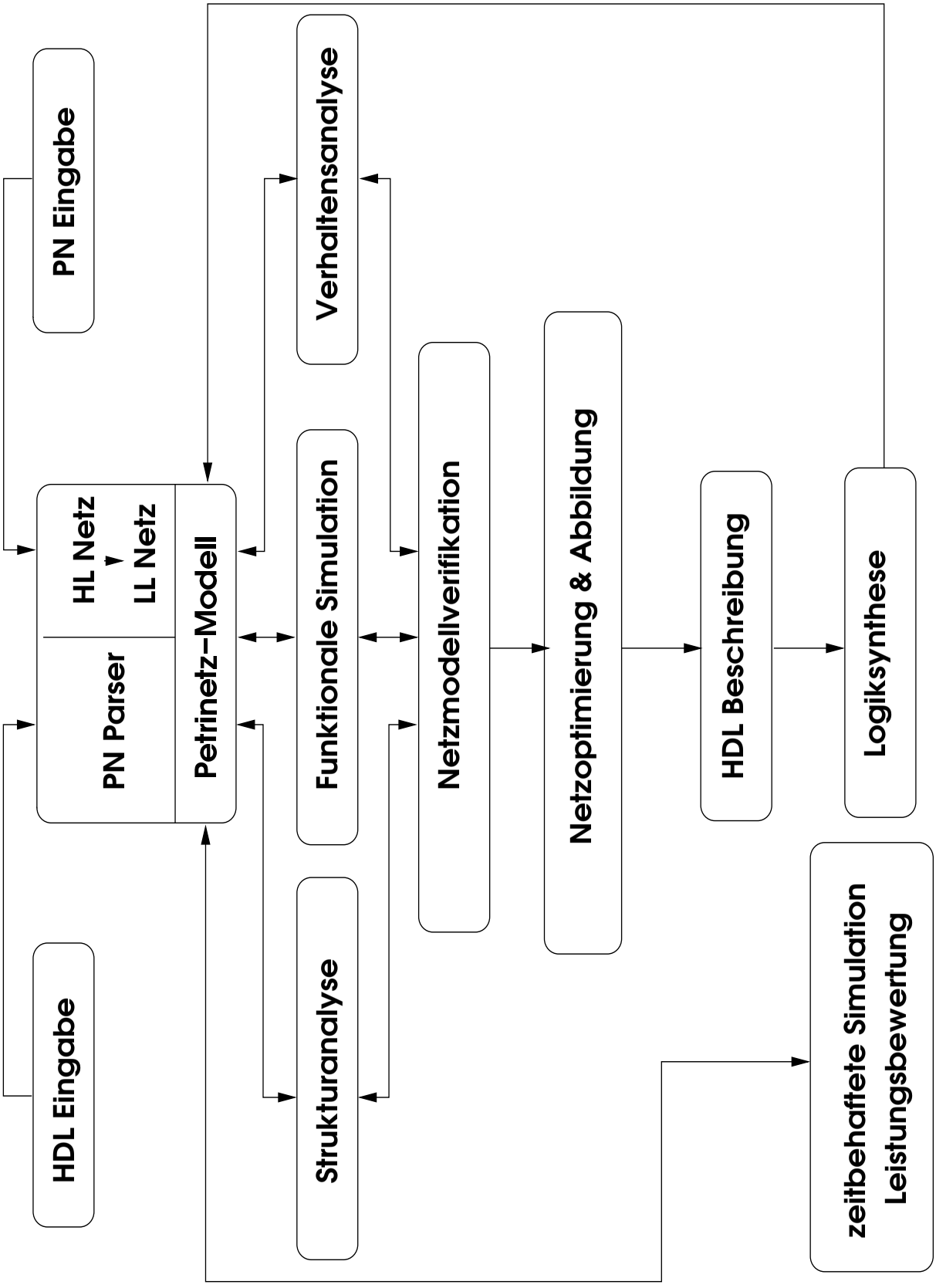

Abbildung 4.1: PNDes

PNDes wird in fünf Entwurfsschritte untergliedert:

1. Modellierung eines digitalen Systems mit Free-Choice-Petrinetzen und der steuerungstechnischen Netzinterpretation
2. Simulation, Validierung, Analyse und Verifikation des Petrinetz-Modells
3. Netzmodelloptimierung und Abbildung auf HDL-Konstrukte
4. Logiksynthese der HDL-Beschreibung mit konventionellen EDA-Werkzeugen
5. Zeitbehaftete Simulation des Petrinetz-Modells und Leisungsbewertung

In der ersten Phase des Entwurfsprozesses wird das digitale System textuell oder grafisch spezifiziert und in ein hardware-nahes Petrinetz-Modell umgesetzt. Vorgegeben sind die Netzspezifikation und die Netzinterpretation als Modellierungsbasis. Die Wahl von FCPN und SIPN wurde in den Abschnitten 2.2 und 3.2 getroffen und begründet.

Die Modellierung eines digitalen Systems mit Free-Choice-Petrinetzen ohne Zeitbehaftung nimmt keinen Bezug auf die Art der Taktung. Die durch funktionale Verifikation erhaltenen Ergebnisse liefern eine Aussage über die vollständige Funktionsfähigkeit ungeachtet des Zeitverhaltens. Ein funktional verifiziertes Petrinetz-Modell ist prinzipiell als asynchrones digitales System *und* als synchron getaktetes digitales System implementierbar.

Höhere Petrinetze sind als Eingabespezifikation dann sinnvoll, wenn diese geeignet in Platz/Transitionen-Netze entfaltet werden können und wenn sich durch deren Anwendung eine Entwurfssicht ergibt, die funktionale Zusammenhänge prägnant wiedergibt und damit überschaubar werden lässt. Eine Anwendung von gefärbten Netzen ermöglicht die Modellierung digitaler Systeme auf höheren als durch die Netzinterpretation festgelegten Abstraktionsstufen. Die Modellierung der Kommunikation zwischen verschiedenen Abstraktionsebenen wird mit hierarchischen Petrinetzen realisiert. Dieses Vorgehen erfolgt analog zum konventionellen Hardware-Entwurf, da auch dieser Entwurf hierarchisch entwickelt wird, bevor der technologieabhängige Syntheseprozess die hierarchische Top-Down-Entwurfssicht in einen „flachen Entwurf“ umwandelt.

Das Petrinetz-Modell wird folgend in mehreren Schritten validiert und analysiert. Bereits während des Modellierungsprozesses wird das Petrinetz-Modell mit Hilfe einer sukzessiven Ablaufsimulation getestet. Grobe Fehler in der Wiedergabe der Systemstruktur und des Systemverhaltens können so erkannt werden. Die funktionale Simulation dient weiterhin dazu, nach abgeschlossenem Modellierungsprozess das Gesamtsystem zu validieren.

Zur formalen Verifikation des Petrinetz-Modells auf funktionaler Ebene wird die Eigenschaftsprüfung angewandt. Entsprechend Abschnitt 2.3 wird hierbei zwischen der Untersuchung von Struktureigenschaften und Verhaltenseigenschaften unterschieden. Eigenschaften, die für das Petrinetz-Modell geprüft wurden, werden für das modellierte digitale System entsprechend einer Verifikationsstrategie interpretiert. Dazu benennt die Verifikationsstrategie eine Menge von Petrinetz-Eigenschaften, die für ein funktional korrektes Systemmodell erfüllt sein müssen. Außerdem wird mit der Verifikationsstrategie jede Modelleigenschaft als eine Eigenschaft des digitalen Systems interpretiert. Wird das Verifikationsziel vollständig erreicht, so kann die formale Verifikation des modellierten digitalen Systems geschlossen werden.

Ein verifiziertes Petrinetz-Modell wird mit Methoden der erweiterten Verhaltensanalyse und mit der Symmetrie- und Reduktionsanalyse optimiert. Ziel ist hierbei die strukturelle Optimierung des Petrinetz-Modells unter Erhaltung der verifizierten Eigenschaften. Ein optimiertes Petrinetz-Modell ermöglicht eine effiziente Abbildung auf Hardware-Komponenten. Dabei werden Hardware-Komponenten eingespart, was sich positiv auf den Flächen- und Leistungsverbrauch des Entwurfs auswirken kann. Weiterhin bewirkt ein verringerter Hardware-Aufwand eine potentielle Steigerung der Verarbeitungsleistung.

Nach erfolgter Verifikation und Optimierung wird das Petrinetz-Modell auf Konstrukte einer Hardware-Beschreibungssprache abgebildet. An dieser Stelle ist die Art der Implementierung im Zusammenhang mit der Zielarchitektur entscheidend. Es wird mit der Wahl der abzubildenden HDL-Konstrukte entschieden, ob ein synchrones digitales System oder ein asynchrones digitales System synthetisiert und implementiert werden soll. Diese Form der Synthese wird als technologie-unabhängige Synthese bezeichnet, obwohl bereits schon auf dieser Synthesestufe eine Zielarchitektur implizit gewählt wurde. Die Vorauswahl einer bestimmten Hardwareplattform ist für eine effiziente Abbildung des optimierten Petrinetz-Modells erforderlich. Liegt eine HDL-Beschreibung des Petrinetz-Modells vor, so wird die technologieabhängige Synthese mit konventionellen Synthesewerkzeugen umgesetzt. Der resultierende Entwurf wird implementiert und getestet.

Der Test und die Leistungsbewertung des implementierten Entwurfes werden mit deterministisch zeitbehafteten Petrinetzen und mit stochastisch zeitbehafteten Petrinetzen realisiert. Dazu werden die nach der Logiksynthese zur Verfügung stehenden realen Verarbeitungszeiten der einzelnen Hardware-Komponenten den Komponenten des Petrinetz-Modells geeignet zugewiesen. Aus dieser Zuweisung entsteht ein zeitbehaftetes Petrinetz-Modell, das simuliert und analysiert werden kann.

Dieser Entwurfsablauf zeigt, dass die Hardware-Entwurfsmethodik PNDes als petrinetzbasierte Entwurfsmethodik in einen konventionellen Entwurfsfluss eingebettet ist. Petrinetze werden in geeigneter Art zur Modellierung, Validierung, Analyse und Verifikation von Hardware-Modellen eingesetzt, während mit der Nutzung konventioneller Hardware-Beschreibungssprachen und Synthesewerkzeuge eine breite Anwendbarkeit bezüglich Modelleingabe und Implementierung gewährleistet ist.

4.1.1 Entwurfsautomatisierung

Für die computergestützte Realisierung der Entwurfmethodik PNDes wird ein petrinetzbasiertes Entwurfswerkzeug benötigt, das ausgehend von einer grafischen oder textuellen Netzeingabe oder einer textuellen HDL-Eingabe die Erstellung und Analyse eines Petrinetz-Modells unterstützt. Die im Sinne eines Hardware-Entwurfsprozesses optimale Ausgabe eines solchen Werkzeuges ist ein aus einem verifizierten Petrinetz-Modell generierter HDL-Entwurf, der mit konventionellen Hardware-Entwurfswerkzeugen synthetisierbar ist.

Die Suche nach einer geeigneten Modellierungs-, Simulations- und Verifikationsumgebung führte über eine vergleichende Bewertung und den Test bestehender Petrinetz-Werkzeuge zum Petrinetz-Kern (PNK) (KW99), der eine Infrastruktur zur Erstellung von Petrinetz-Werkzeugen darstellt und eine maximale Anzahl von Freiheitsgraden bei der Entwurfsunterstützung offeriert. In Kapitel 7 wird das mit Hilfe des PNK erstellte Werkzeug VeriCon vorgestellt. VeriCon realisiert prototypisch die Entwurfsmethodik PNDes.

4.2 Modellierung und Verifikation digitaler Systeme mit PNDes

In diesem Abschnitt wird der Fokus von PNDes, die Modellierung und Verifikation digitaler Systeme, weiter untersetzt. Entscheidend ist hierfür die Idee, ein mit interpretierten Petrinetzen modelliertes System formal zu verifizieren. Abbildung 4.2 stellt diese Idee schematisch anhand von 7 Schritten dar.

In den ersten beiden Schritten werden Netzspezifikation und Netzinterpretation festgelegt. Mit Hilfe dieser Voraussetzung kann der Modellierungsprozess durchgeführt werden, und es entsteht im dritten Schritt ein sehr transparentes Petrinetz-Modell, das mit seinem Markenfluss den Signalfluss des digitalen Systems wiedergibt. Das Petrinetz-Modell wird in einem weiteren Schritt während der Modellierung sukzessiv funktional simuliert und als vollständiges Hardware-Modell validiert. Ein validiertes Petrinetz-Modell wird folgend im fünften Schritt in ein uninterpretiertes Petrinetz transformiert, um somit eine

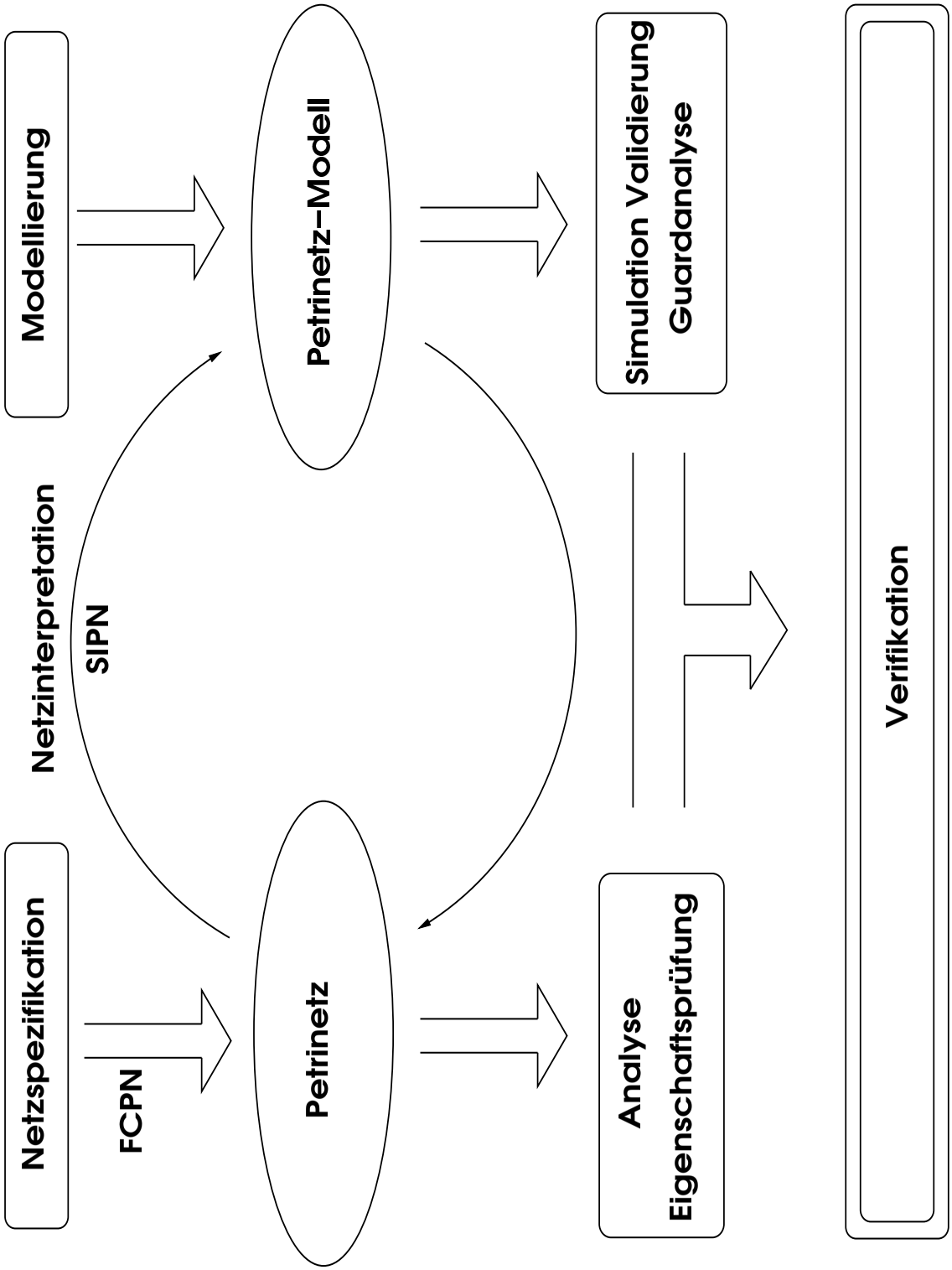

Abbildung 4.2: Modellierung und Verifikation mit PNDes

formale Analyse des Zustandsraumes zu ermöglichen. Die Zustandsräume des Petrinetz-Modells und des uninterpretierten Petrinetzes sind unter Beachtung der Schaltstrategie aus Definition 2.6 (ii) äquivalent. Das ist die Voraussetzung, die im letzten Schritt nach erfolgter Eigenschaftprüfung des uninterpretierten Petrinetzes, die Schlussfolgerungen zur formalen Verifikation des Petrinetz-Modells erlaubt. Zuvor erfolgt im sechsten Schritt die Analyse und Eigenschaftsprüfung auf der Grundlage der in Kapitel 2 vorgestellten Analyse von Petrinetz-Modellen, getrennt nach Struktur- und Verhaltenseigenschaften. Ebenso werden spezielle Schaltbedingungen des Petrinetz-Modells, genannt Guards, in einer Analyse ausgewertet und fließen in die Schlussfolgerung zur formalen Verifikation des Petrinetz-Modells ein.

Für die Modellierung und Verifikation digitaler Systeme mit PNDes werden zwei Ziele formuliert. Ziel des Modellierungsprozesses ist ein einfach strukturiertes, leicht überschaubares Petrinetz-Modell, das in Struktur und Verhalten transparent zum digitalen System ist. Mit Hilfe von geeigneten Modellierungsrichtlinien sollen viele Modellierungsfehler von Anfang an vermieden werden. Ziel der Modellverifikation ist eine vollständige Aussage zur funktionalen Korrektheit des Hardware-Modells. Umgesetzt wird dieses Ziel mit einer Verifikationsstrategie und der Analyse von Petrinetz-Eigenschaften. Dabei soll es möglich sein, Analyseverfahren entsprechend der Größe des Zustandsraumes des Petrinetz-Modells geeignet anzuwenden.

Kapitel 5

Modellierung digitaler Systeme mit Interpretierten Petrinetzen

5.1 SIPN-Ansatz

In Abschnitt 3.2 wurden steuerungstechnisch interpretierte Petrinetze (SIPN) allgemein als Methode einer Netzsynthese eingeführt, die zwischen einer Software-Interpretation und einer Hardware-Interpretation eines Petrinetzes unterscheidet. Da PNDes den petrinetz-basierten Entwurf digitaler Systeme zum Ziel hat, ist die Software-Interpretation eines Petrinetzes für diese Arbeit nicht von Nutzen. Für die Modellierung digitaler Systeme wird folgend ausschließlich die Hardware-Interpretation der SIPN angewandt.

Definition 5.1 Steuerungstechnisch interpretiertes Petrinetz : Ein steuerungstechnisch interpretiertes Petrinetz N_{SIPN} ist ein 10-Tupel $\{P, T, F^{pre}, F^{post}, G, X, Y, Q_t, Q_p, m_0\}$ mit

(i) Transitionenmenge T

(ii) Platzmenge $P,\ p_j = (e_j, a_j)$

(iii) Prekantenmenge $F^{pre},\ f_k^{pre} = (p_j, t_i)$

(iv) Postkantenmenge $F^{post},\ f_k^{post} = (t_i, p_j)$

(v) Guardmenge G

(vi) Eingabemenge X

(vii) Ausgabemenge Y

(viii) Abbildung Q_t zur Transitionsaktivierung $Q_t = G_i a_j = \{0,1\}^{\{X\}}$

(ix) Abbildung Q_p zur Platzmarkierung $Q_p = a_j = \{Y\}^{\{X\}}$

(x) Initialmarkierung m_0

Bei der steuerungstechnischen Interpretation eines Petrinetzes werden kommunizierende endliche Automaten direkt auf Hardware-Module derart abgebildet, dass der Signalfluss eines implementierten Petrinetz-Modells äquivalent als Markenfluss im Petrinetz-Modell interpretierbar und analysierbar ist.

Ein SIPN N_{SIPN} ist in Abbildung 5.1 visualisiert. Bei der gewählten Hardware-Interpretation wird jeder Transition t_i eine UND-Verknüpfung und jedem Platz p_j ein Speicherplatz eines Zustandsspeichers zugeordnet. Eingangssignale x_i können beliebig logisch verknüpft werden und gehen als Schaltbedingungen G_i für Transitionen in das Petrinetz-Modell ein. Die Dynamik eines SIPN wird mit den Abbildungen Q_t für die Transitionsaktivierung und Q_p für die Platzmarkierung festgelegt. Die Markierung $m(p_j) = 1$ entspricht einem gesetzten logischen Wert in der Speicherzelle $p_j = (e_j, a_j)$. Ein Signal $a_j = 1$ kann als Ausgangssignal y_j weiterverarbeitet und für einen Zustandswechsel $p_{j_1} \rightarrow p_{j_2}$ benutzt werden. Bei einem Zustandswechsel wird a_j zu einer frei wählbaren Transition t_i geführt und dort mit einer aus der Eingangssignalmenge $\{X\}$ gebildeten Schaltbedingung G_i konjunktiv verknüpft. Ist die Transition t_i aktiviert, kann ein Schaltvorgang stattfinden. Der Zustand p_{j_2} wird gesetzt, nachdem p_{j_1} rückgesetzt wurde.

Die Art der Kommunikation der endlichen Automaten bestimmt das Verhalten des digitalen Systems. Mit den Verhaltenskonstrukten Verzweigung und Begegnung werden sequentielle Systeme beschrieben. Die Hardware-Interpretation weist in diesem Fall Verbindungen zwischen einer Speicherzelle und mehreren UND-Verknüpfungen oder zwischen mehreren UND-Verknüpfungen und einer Speicherzelle auf. Nebenläufige Systeme entstehen mit den Verhaltenskonstrukten Aufspaltung und Synchronisation. Dabei existieren Verbindungen zwischen einer UND-Verknüpfung und mehreren Speicherzellen oder zwischen mehreren Speicherzellen und einer UND-Verknüpfung. Mit diesen beiden Möglichkeiten können mehrere Speichermodule mit UND-Verknüpfungen sequentiell oder nebenläufig gekoppelt werden.

In dieser Arbeit wird die steuerungstechnische Netzinterpretation mit dem Ziel angewandt, komplexe sequentielle und nebenläufige Kontrollpfade digitaler Systeme, sowie deren Umgebung transparent zu modellieren. Der Modellierungsprozess ist dabei die Basis für die formale Verifizierbarkeit des Hardware-Modells. Als Zielanwendung wird in Abschnitt 5.4 die Modellierung eines Mikroprozessors in verschiedenen Varianten detailliert vorgestellt.

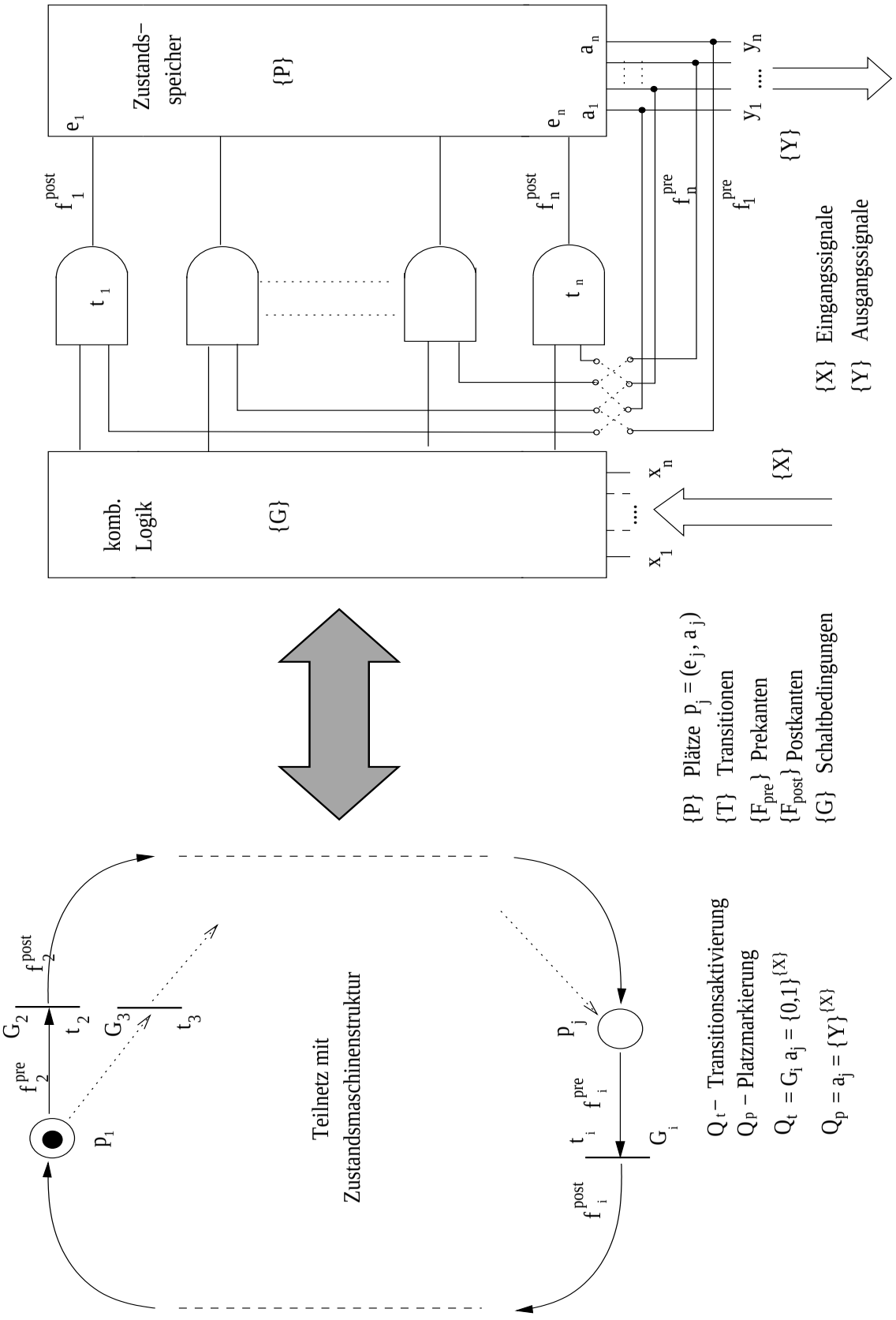

Abbildung 5.1: SIPN mit Hardware-Interpretation

5.2 Transformationen zwischen VHDL-Konstrukten und Petrinetz-Strukturen

Die Forderung der Einbettung petrinetz-basierter Entwurfsschritte in einen konventionellen Entwurfsablauf erfüllt die Hardware-Entwurfsmethodik PNDes mit zwei Schnittstellen, an denen VHDL-Konstrukte in Petrinetz-Strukturen, sowie Petrinetz-Strukturen in VHDL-Konstrukte transformiert werden. Vorschläge für diese Transformationen existieren auf akademischer Seite seit einigen Jahren. Hier soll die Umsetzung von VHDL-Konstrukten in Petrinetz-Strukturen vorgestellt werden, die eine HDL-Eingabe eines Hardware-Modells ermöglicht. Die automatische Generierung einer VHDL-Beschreibung aus einem Petrinetz-Modell nach erfolgter Modellverifikation und -optimierung ist invers dazu ebenso möglich.

Es existieren zwei Möglichkeiten der Definition einer VHDL/PN-Schnittstelle:

(i) Die Erweiterung der Netzspezifikation mit speziellen Kanten, Plätzen oder Transitionen, sowie mit verändertem Schaltverhalten, die eine exakte und transparente Umsetzung von HDL-Konstrukten ermöglicht.

(ii) Die intuitive Umsetzung von HDL-Kontrollflussanweisungen in Petrinetz-Strukturen ohne eine Erweiterung der Netzspezifikation mit Ausnahme von Schaltbedingungen (Guards).

PNDes sieht keine Anwendung erweiterter oder selbst definierter Netzspezifikationen vor, weil dadurch die Verifizierbarkeit des Petrinetz-Modells nicht mehr gewährleistet wäre. Somit ist eine Schnittstelle gemäß (i) nicht akzeptabel. Für eine intuitive Umsetzung, entsprechend (ii), existieren mehrere Vorschläge, die ähnliche Transformationsregeln aufstellen (OC93, Olc95, Idz01).

Zentrales Element bei der Wiedergabe einzelner Kontrollflussanweisungen ist das Verhaltenskonstrukt Verzweigung. Die Anweisungen IF, CASE, FOR, WHILE und LOOP beginnen mit der Verzweigung eines Kontrollzustandes in mögliche Alternativen. Die Bedingungen für eine Verzweigung werden als Guards repräsentiert. Dabei geben die Guards äußere Signale wieder, die eine Interaktion mit dem Datenpfad, sowie mit der Umgebung modellieren.

In Abbildung 5.2 sind strukturelle PN-Umsetzungen einzelner VHDL-Konstrukte illustriert. Hier wurde von der Angabe konkreter Guards abstrahiert. Diese Kontrollflussanweisungen können in einfacher Weise mit anderen Anweisungen, wie NEXT, EXIT oder WAIT, erweitert werden. Eine Kontrollflussanweisung kann mit dem Verhaltenskonstrukt Begegnung beendet werden. Prozesse werden exakt mit Petrinetz-Zustandsmaschinen,

entsprechend Definition 2.10 modelliert. Es besteht die Möglichkeit, Kontrollflussanweisungen mit Transitionen nebenläufig als Aufspaltung und Synchronisation sequentieller Kontrollflussanweisungen zu koppeln.

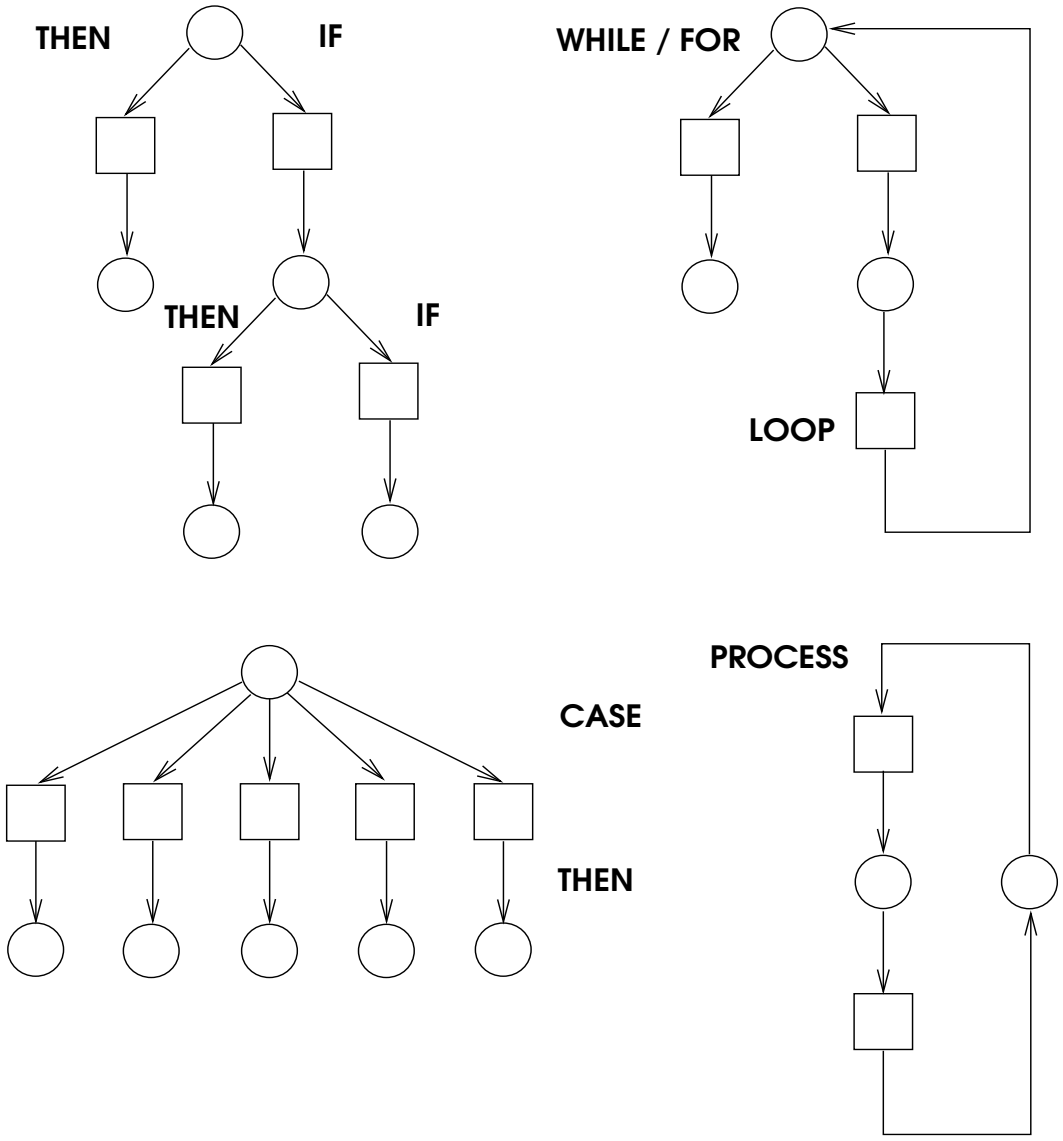

Abbildung 5.2: Strukturelle PN-Umsetzung von VHDL-Konstrukten

5.3 Hierarchische Petrinetz-Modelle

Ziel einer Hierarchisierung beim Entwurf technischer Systeme ist die Strukturierung des Modells in verschiedene Modellebenen. Große Modelle werden dadurch überschaubar und der Entwurf komplexer technischer Systeme wird beherrschbar. Eine formale Definition hierarchischer Petrinetze erfolgt in (Feh93).

Methoden der Hierarchisierung sind Abstraktion und Verfeinerung, ausgehend von einer gegebenen Modellierungsebene. Für Petrinetz-Modelle wurden beide Methoden erstmals in (Val79) und (SM83) derart angewandt, dass wichtige Verhaltenseigenschaften wie Lebendigkeit und Beschränktheit bei der Hierarchisierung erhalten bleiben. Abbildung 5.3 zeigt die Methode der Verfeinerung anhand von Transitionen.

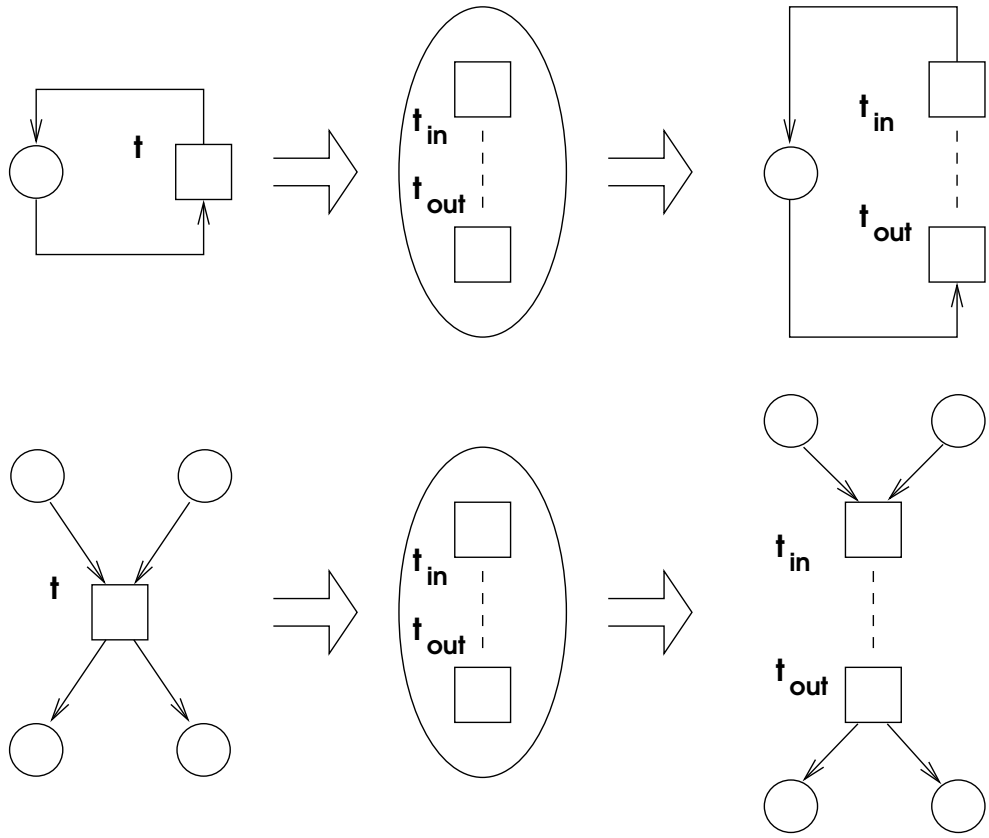

Abbildung 5.3: Transitionenverfeinerung

Die Methode der Verfeinerung wird vorrangig auf Transitionen angewandt, da Transitionen die aktiven Elemente (vgl. Abschnitt 2.1 „lokale Aktionen") des Petrinetzes sind. Eine Transitionenverfeinerung kann mit einer Platzverfeinerung kombiniert werden. Entsprechend Abbildung 5.4 ist dann jede Verfeinerung mit dem Ziel der Hierarchisierung des Petrinetz-Modells als kombinierte Transitionen- und Platzverfeinerung umsetzbar.

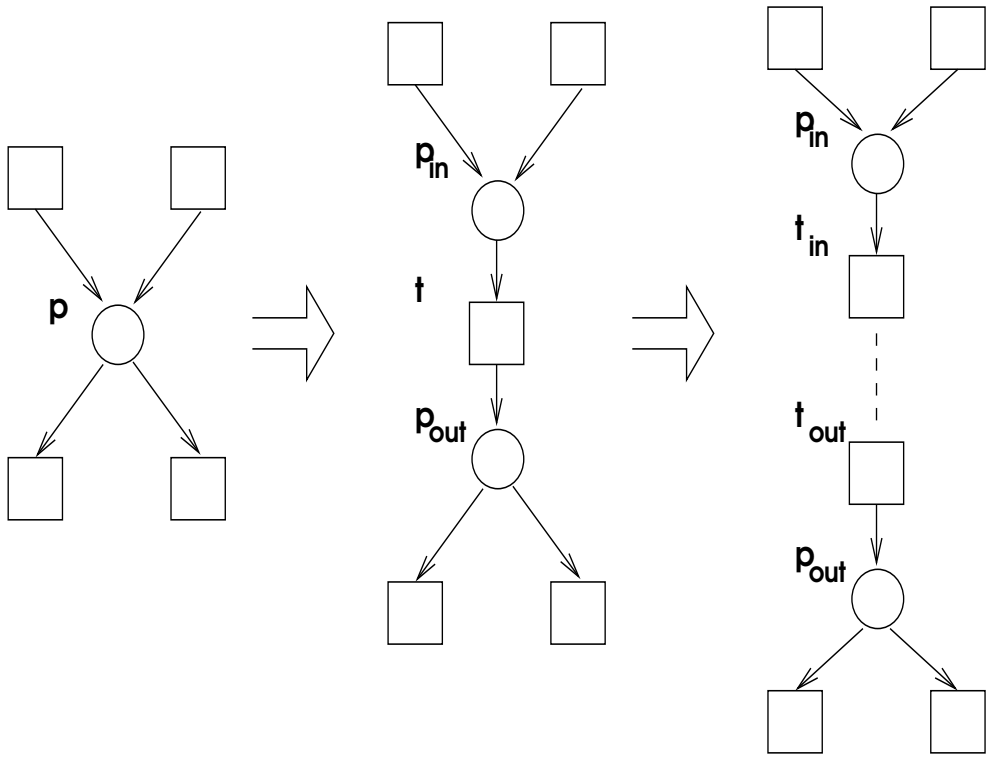

Abbildung 5.4: Platzverfeinerung durch Transitionenverfeinerung

Bei der Verfeinerung können gezielt sequentielle oder parallele Verhaltenskonstrukte eingeführt werden. Abbildung 5.5 illustriert die parallele und die sequentielle Expansion nach erfolgter Transitionenverfeinerung. Innerhalb der Expansion können beliebige Plätze wiederum verfeinert werden.

Eine weitere Anwendung der Methode der Verfeinerung ist die Modellierung durch Verfeinerung mit Netzmodulen. Durch einfache Netzmodule kann ein Petrinetz-Modell entsprechend der Vorschläge der Abbildungen 5.3, 5.4 und 5.5 komponiert werden, das global die Verhaltenseigenschaften der einzelnen Netzmodule aufweist. Idealisiert führt diese Möglichkeit der Modellierung zu einem First-Time-Right-Entwurf, der ad hoc funktionale Korrektheit aufweist.

Die Anwendung hierarchischer Petrinetze wird in Abschnitt 4.1 für die Modellierung der Kommunikation zwischen Systemkomponenten auf unterschiedlichen Abstraktionsebenen vorgeschlagen. Darüber hinaus können hierarchische Petrinetze zur vollständigen Modellierung auf verschiedenen Abstraktionsebenen benutzt werden. Die Hardware-Entwurfsmethodik PNDes gibt mit der gewählten Netzsynthese SIPN eine bestimmte Abstraktionsebene vor. Da bei dieser Netzsynthese Petrinetz-Konstituenten auf Speicherelemente und kombinatorische Elemente abgebildet werden, befindet sich die

gegebene Abstraktionsebene für den Modellierungsprozess auf Register-Transfer-Ebene. Beispiele für die Anwendung der Hierarchisierungsmethoden werden in Abschnitt 5.4.2 vorgestellt.

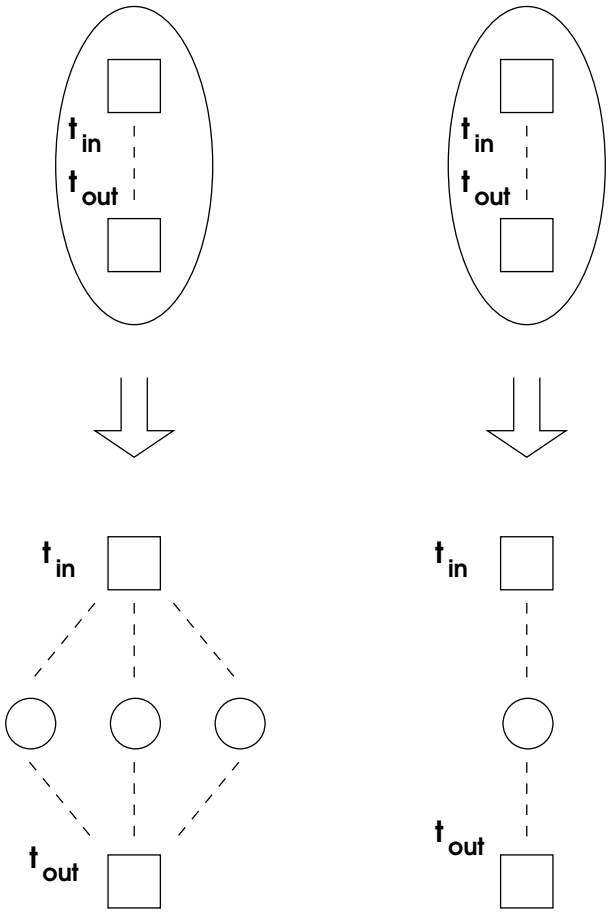

Abbildung 5.5: Parallele und sequentielle Expansion

5.4 Eine Anwendung - Das Petrinetz-Modell eines Mikroprozessors

In diesem Abschnitt wird eine Anwendung petrinetz-basierter Modellierung digitaler Systeme mit PNDes vorgestellt. Es handelt sich um den Entwurf des Kontrollpfades eines RISC-Prozessors in mehreren Varianten. Der DLX-Prozessor nach Hennessy und Patterson (HP90, HP96) dient in der sequentiellen und in der nebenläufigen Variante als Beispiel. Abbildung 5.6 schematisiert das Gesamtsystem, bestehend aus Prozessor-Kern, Speicher und Umgebung.

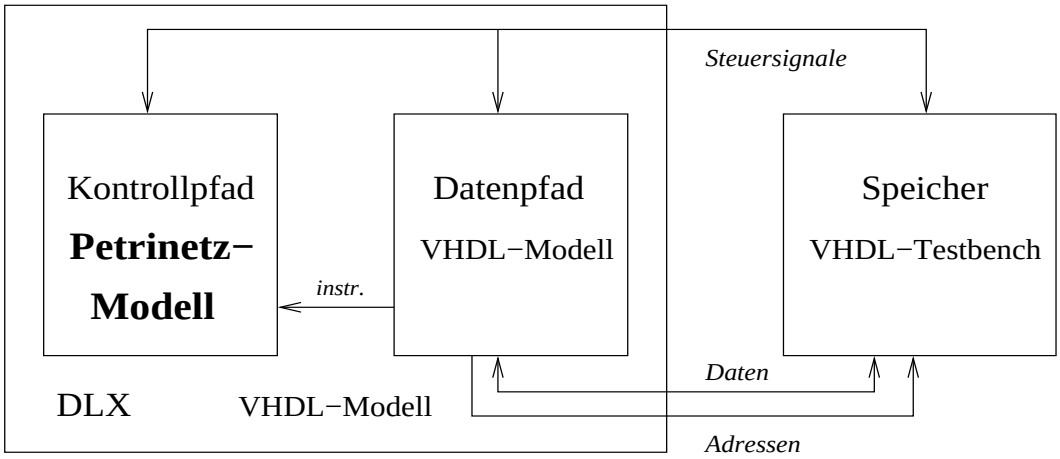

Abbildung 5.6: DLX-Prozessor mit externem Speicher

Der Kontrollpfad umfasst die Steuerung und verschiedene Befehlsdekoder, in denen Befehle unterschiedlicher Formate dekodiert werden. Der DLX-Prozessor unterscheidet zwischen drei Befehlsformaten. In Abbildung 5.7 sind die Befehlsformate des DLX aufgeführt.

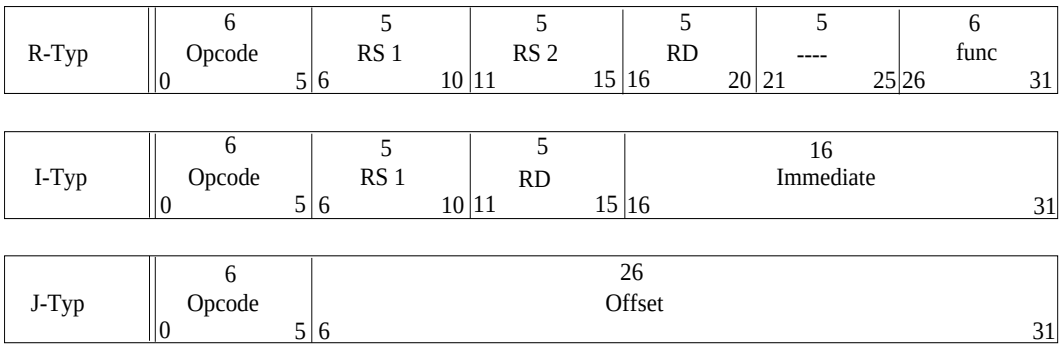

Abbildung 5.7: Befehlsformate des DLX-Prozessors

Um ein reales Petrinetz-Modell des Prozessors zu erstellen, werden synthesefähige Register-Transfer-Beschreibungen des sequentiellen und nebenläufigen DLX-Prozessors in der Hardware-Beschreibungssprache VHDL als Vorlage genutzt (Gum95, Sch98). Die Umsetzung des VHDL-Modells in ein Petrinetz-Modell erfolgte mit den Regeln aus (OC93). Eine Rücktransformation eines verifizierten Petrinetz-Modells in eine VHDL-Beschreibung ist ebenso realisierbar. Der vollständige DLX-Befehlssatz umfasst 118 Befehle. Die hier betrachteten Kontrollpfade realisieren jeweils 52 Befehle aller herkömmlichen Befehlsklassen. Ausnahmen, Unterbrechungen sowie das Anhalten und Rücksetzen des Prozessors wurden im Petrinetz-Modell realisiert. Befehle für Operanden ohne Vorzeichen und Befehle zur Verarbeitung von Fließkommazahlen wurden nicht realisiert.

5.4.1 Petrinetz-Modell des sequentiellen DLX-Prozessors

Das Hardware-Modell des sequentiellen DLX-Prozessors wird in (HP90) ausführlich behandelt. Das Petrinetz-Modell des sequentiellen DLX-Prozessors gibt die Register-Transfer-Ebene des Kontrollpfades des Prozessors wieder, so dass nach einer Rücktransformation in ein VHDL-Modell eine synthesefähige Hardware-Beschreibung vorliegt. Die Abbildungen 5.8 und 5.9 untersetzen die Darstellung des DLX-Gesamtsystems (Abbildung 5.6) mit dem Kontroll- und Datenpfad des Hardware-Modells.

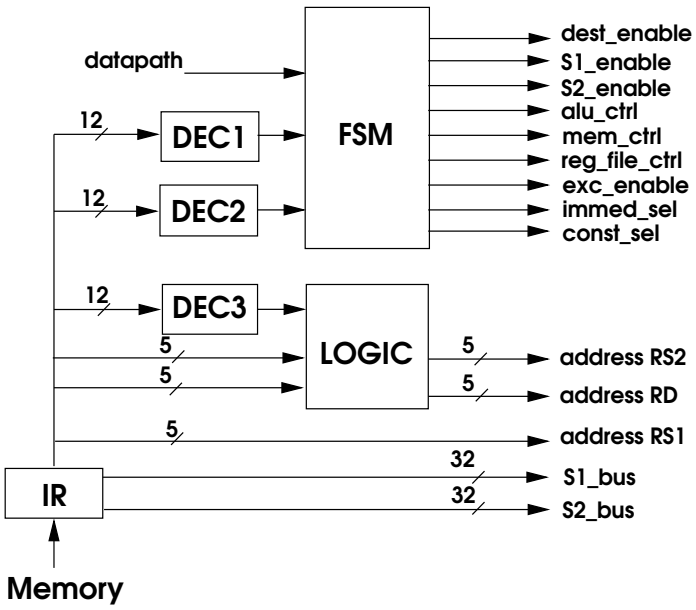

Abbildung 5.8: Kontrollpfad des sequentiellen DLX-Prozessors

Zur Implementierung der 52 DLX-Befehle werden 64 Zustände benötigt (ERS01a). Transitionen modellieren Zustandsübergänge $p_{j_1} \rightarrow p_{j_2}$ und werden teilweise mit äuße-

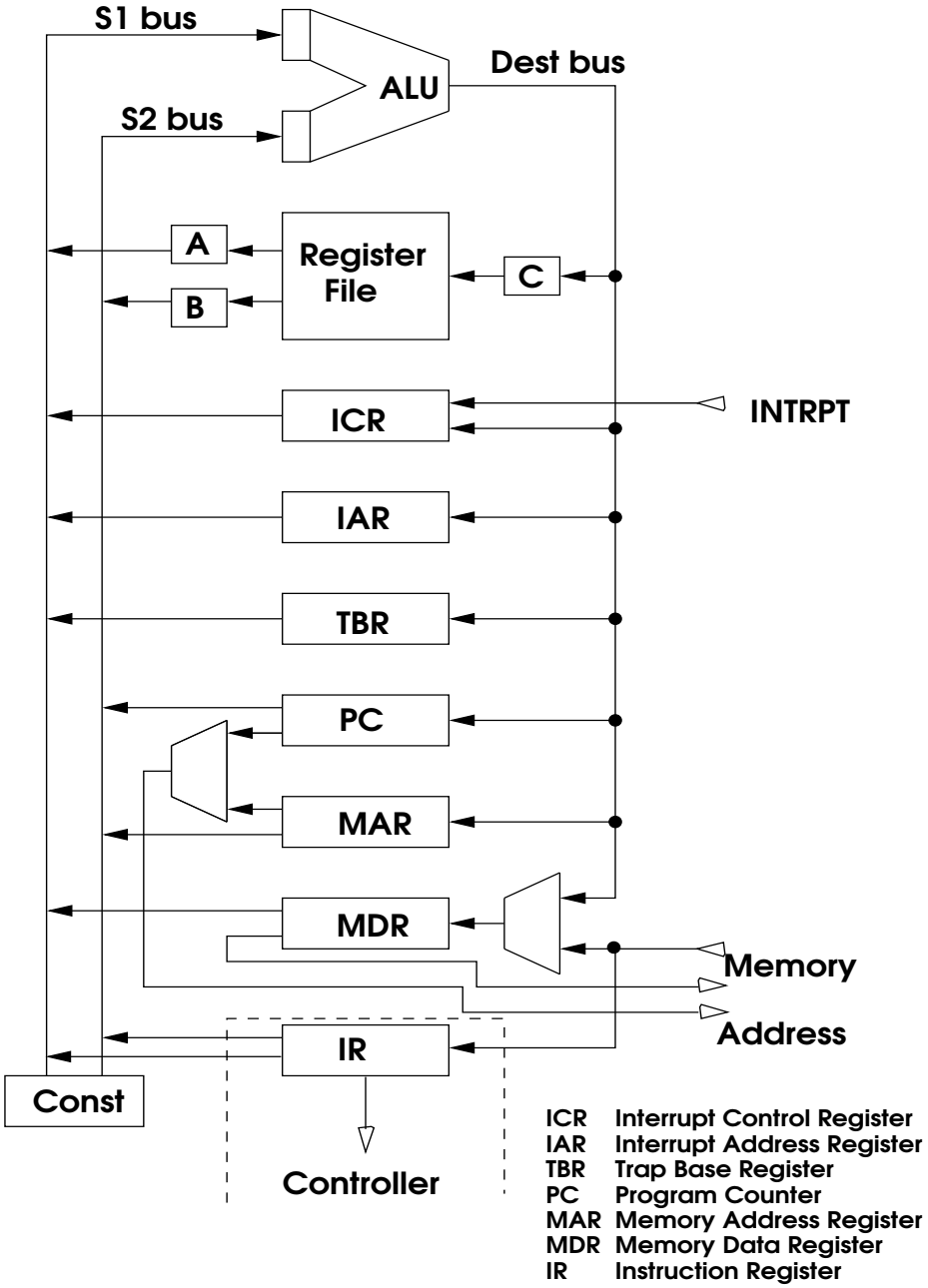

Abbildung 5.9: Datenpfad des sequentiellen DLX-Prozessors

ren Schaltbedingungen G_j belegt. Das Petrinetz-Modell besteht aus 243 Netzknoten, die in 4 Teilnetze untergliedert sind. Im Anhang A ist mit den Abbildungen A.1 bis A.4 das vollständige Kontrollpfad-Modell dargestellt. Fett umrandete Plätze (z.B. p_{decode}) verbinden die 4 Abschnitte, indem sie in verschiedenen Teilnetzen A.1 bis A.4 auftreten.

Mit dem Petrinetz-Modell können beliebige Befehlssequenzen validiert, simuliert, analysiert und verifiziert werden. Bei Übergabe gültiger Steuersignale werden mit den äußeren Schaltbedingungen Alternativen freigeschaltet, deren Abarbeitung eine Befehlssequenz bildet. Die äußeren Schaltbedingungen beschreiben Steuersignale, die mit der Umgebung des Kontrollpfades interagieren. Hierbei existieren drei verschiedene Kategorien äußerer Schaltbedingungen: Befehlscode, Flags und externe Signale. Im Petrinetz-Modell werden die verschiedenen Signalarten nicht unterschieden, da sie bezüglich des Kontrollpfades äußere Signale darstellen.

Da die Erzeugung und Verarbeitung der Steuersignale mit kombinatorischen Schaltungen beschrieben wird, ist jede äußere Schaltbedingung als Boolsche Funktion darstellbar. Alle kombinatorischen Schalteinheiten des Prozessors werden in äußere Schaltbedingungen für Transitionen abgebildet. Die Schnittstelle zwischen textueller Beschreibung mit VHDL und dem Petrinetz-Modell ist hier analog der Trennung zwischen kombinatorischen und sequentiellen Schalteinheiten des Prozessors definiert. Die Kombinatorik wird abstrakt als Menge von Bedingungen zusammengefasst, während die sequentielle Steuerung strukturell in das Petrinetz-Modell abgebildet wurde.

Bei weiterer Modelluntersetzung und zusätzlicher Interpretation der Modellkomponenten sind äußere Schaltbedingungen als Teilnetze modellierbar. Dadurch werden neben sequentiellen auch kombinatorische Schaltungen im Petrinetz-Modell transparent. Das Petrinetz-Modell ist auf diesem Wege erweiterbar und kann den Datenpfad sowie eine Prozessorumgebung beschreiben. Andererseits ist eine Modellierung des Gesamtsystems mit Petrinetzen nicht sinnvoll, da die erweiterte Netzinterpretation das Modellverständnis erschwert und weitere Probleme bei der Analyse und Verifikation des Petrinetz-Modells aufwerfen würde. In dieser Arbeit wird die Modellierung von Datenpfaden digitaler Systeme mit interpretierten Petrinetzen nicht vertieft.

Aus Gründen der besseren Übersicht wird an dieser Stelle ein reduziertes Petrinetz-Modell betrachtet. Die Funktionalität des Prozessors bleibt nach der Reduktion des Befehlssatzes weitgehend erhalten, da jedes der drei Befehlsformate weiterhin unterstützt wird und da Fehlererkennung sowie RESET-Zustand und HALT-Zustand berücksichtigt werden. Der reduzierte Befehlssatz schafft die Basis für einen Vergleich mit (vGvBB98). In Tabelle 5.1 ist der Befehlsumfang des reduzierten DLX-Modells aufgelistet. Abbildung 5.10 zeigt das komplette Kontrollpfad-Modell für 6 Befehle, das insgesamt 49 Netzknoten benötigt.

Beide Petrinetz-Modelle des sequentiellen DLX-Prozessors sind optimierte Modelle bezüglich der Umsetzung der VHDL-Konstrukte in Petrinetz-Strukturen. Ziel der Optimierung ist die transparente Umsetzung mit einer minimalen Anzahl Netzknoten. Die Kom-

Befehl	Wirkung	Befehlsgruppe	Befehlsformat
OP_BEQZ	branch if equal zero	branch	I-Typ
OP_J	jump	branch	J-Typ
OP_SW	store word	load/store	I-Typ
OP_LW	load word	load/store	I-Typ
FUNC_SUB	subtraction	arithmetic	R-Typ
FUNC_SLT	set lower than	test	R-Typ
FUNC_NOP	no operation	arithmetic	R-Typ

Tabelle 5.1: Befehlsumfang des reduzierten DLX-Modells

munikation vom Datenpfad zum Kontrollpfad wird mit Signalen modelliert, die als Guards für Transitionen wirken. Werden entsprechend Abschnitt 5.2 Kontrollflussanweisungen als Petrinetz-Modellstrukturen abgebildet, so kommt es in vielen Fällen zu ineffizienten Abbildungen, die einen erhöhten Hardware-Aufwand verursachen.

Bei genauer Beachtung der Netzinterpretation kann der Hardware-Aufwand optimiert werden. Ziel ist hierbei die Optimierung des Hardware-Aufwands entsprechend der in der SIPN-Definition festgelegten Äquivalenz zwischen Speicherzellen und Plätzen des Petrinetz-Modells. Demnach soll ein Petrinetz-Modell exakt die Anzahl Speicherzellen des digitalen Systems wiederspiegeln, die notwendig ist, um den Steuerpfad mit allen Steuerzuständen zu realisieren.

Abbildung 5.11 verdeutlicht das Problem anhand eines Ausschnitts des vollständigen Petrinetz-Modells. Im oberen Teil der Abbildung wurde der Übergang vom SUB-Zustand in mögliche Folgezustände als typisches IF-THEN-ELSEIF VHDL-Konstrukt umgesetzt. Die Signale ALU_OVFL und SUPER werden nacheinander abgefragt und es entsteht ein Zwischenzustand P1. Der Zwischenzustand P1 ist kein für die Steuerung notwendiger Zustand, der Systemausgaben an die Umgebung oder Signale an den Datenpfad übermitteln muss. Für die Modellierung der drei möglichen Zustandsübergänge werden 9 Netzknoten benötigt, die Hardware-Komponenten entsprechen.

Im unteren Teil der Abbildung 5.11 wurden die Zustandsübergänge ausgehend vom Zustand SUB als optimierte Petrinetz-Modellstruktur dargestellt. Hier konnte das Petrinetz-Modell durch das Zusammenfassen der beiden Signalabfragen optimiert werden. Für die Modellierung der drei möglichen Zustandsübergänge werden nun 7 Netzknoten benötigt und jeder Platz modelliert genau eine Speicherzelle der Steuerung, die einen Steuerzustand wiedergibt. Ein weiteres Beispiel der Optimierung der Petrinetz-Modells ist in Abbildung 5.10 erkennbar. Bei Auswahl eines Lade- oder Speicherbefehls wird nach

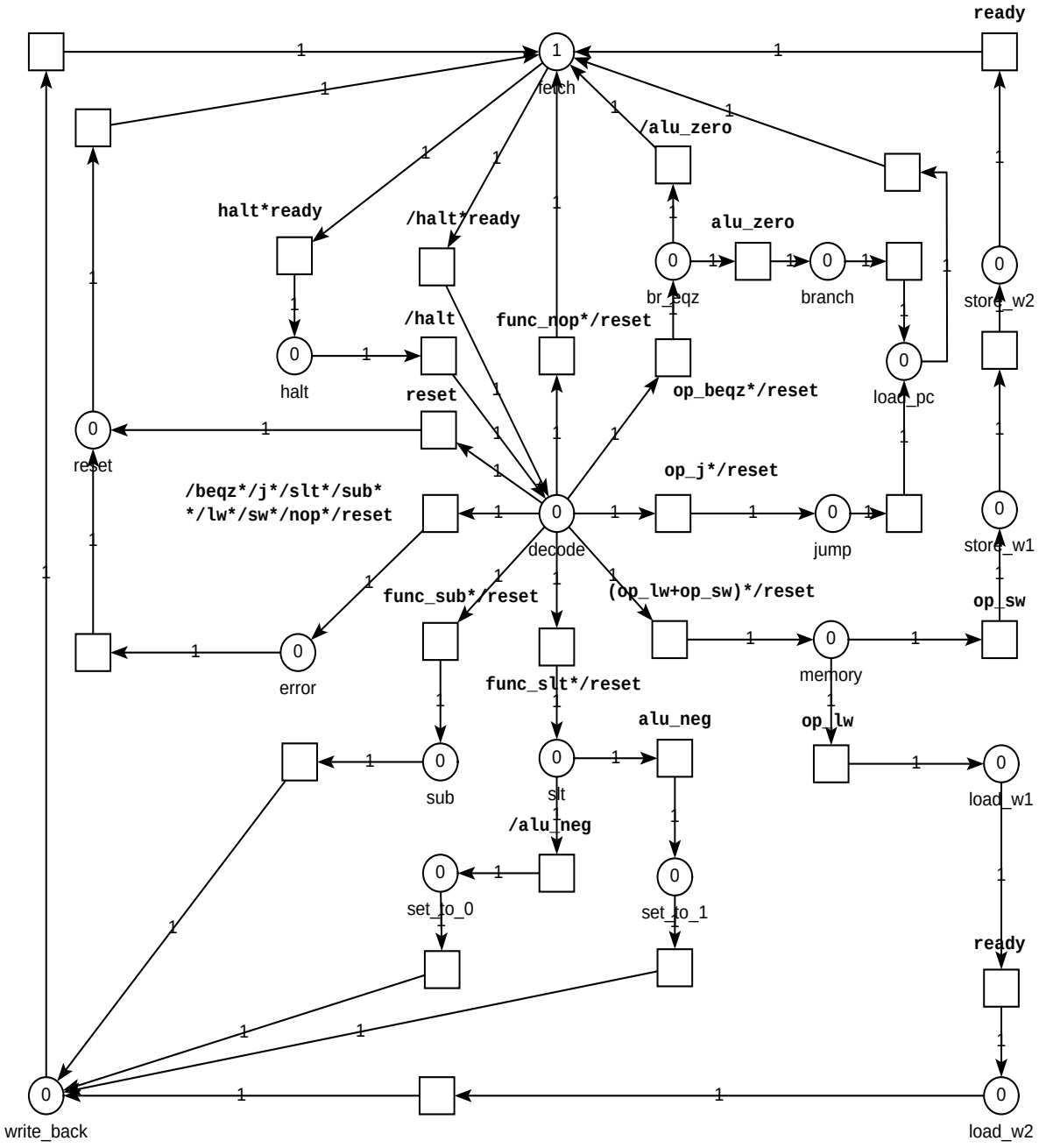

Abbildung 5.10: Reduziertes Petrinetz-Modell des sequentiellen DLX-Prozessors

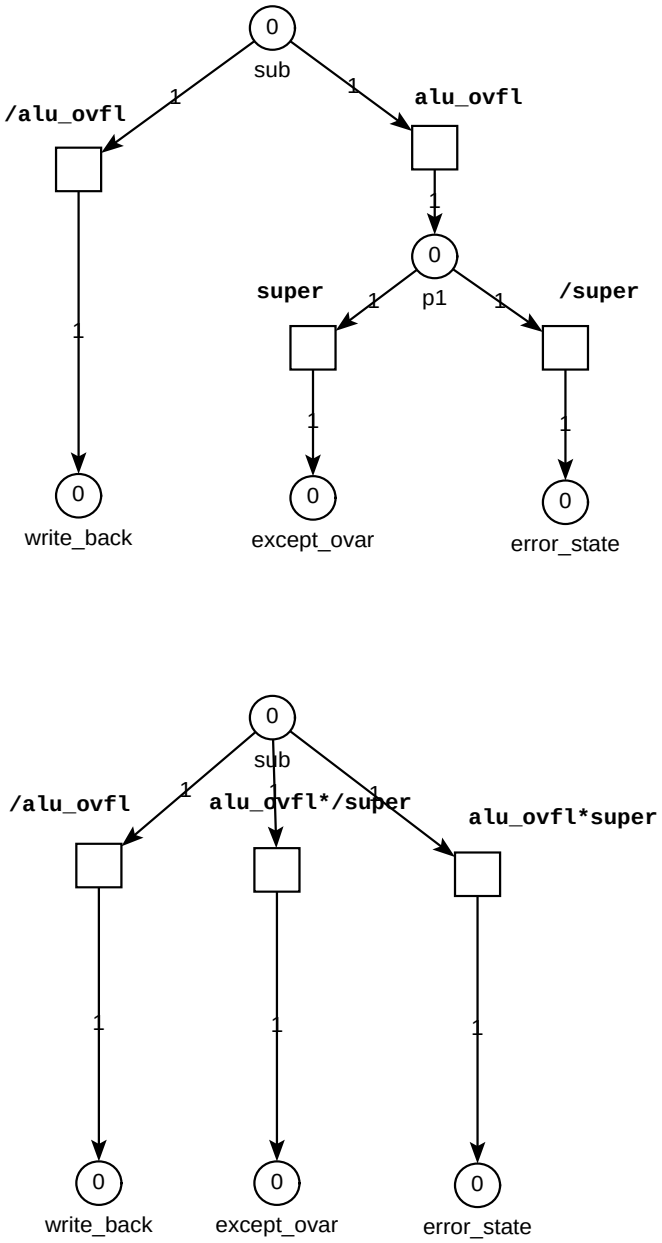

Abbildung 5.11: Optimierte Modellierung von Kontrollanweisungen

dem DECODE-Zustand der MEMORY-Zustand angenommen. Aus diesem Grund werden dort zwei Transitionen mit der Auswahl OP_LW und OP_SW zu einer Transition mit der Auswahl OP_LW+OP_SW zusammengefasst.

Für das vollständige Petrinetz-Modell des sequentiellen DLX-Prozessors werden unter Vernachlässigung der Befehlscodes als äußere Signale des Kontrollpfades, 11 verschiedene äußere Signale zu insgesamt 35 unterschiedlichen äußeren Schaltbedingungen kombiniert. Von 179 Transitionen sind 145 Transitionen mit einer dieser äußeren Schaltbedingungen behaftet. Mit der Optimierung durch die konjunktive Verknüpfung äußerer Signale konnte eine transparente Abbildung derart realisiert werden, dass die Anzahl der Plätze des Petrinetz-Modells nicht größer, sondern gleich der Anzahl der Steuerzustände des Kontrollpfades ist. Grundsätzlich kann ein Petrinetz-Modell mit der konjunktiven und disjunktiven Verknüpfung äußerer Signale zu komplexen äußeren Schaltbedingungen optimiert werden. In Abhängigkeit von der gewählten Zielarchitektur kann diese Optimierung den erforderlichen Hardware-Aufwand signifikant senken. Die Optimierung findet innerhalb des Modellierungsprozesses und nach der Modell-Verifikation statt. In Abschnitt 6.4 wird eine Methode vorgestellt, die die Optimierung eines verifizierten Petrinetz-Modells durch disjunktive Signalverknüpfungen vorschlägt.

Die Kommunikation zwischen Kontrollpfad und Datenpfad sowie zwischen Kontrollpfad und Umgebung wird mit dem Petrinetz-Modell nicht explizit modelliert. In Abbildung 5.8 sind die Ausgaben der Steuerung (FSM) dargestellt. Für alle Zustände des Kontrollpfades existieren jeweils 9 Steuersignale, die unterschiedliche Werte annehmen, um Datenpfad und Umgebung zu steuern. Zu jedem Zustand gibt es einen Vektor gültiger Steuersignale. Für kein Paar beliebiger Steuerzustände sind diese Steuersignalvektoren identisch. Deshalb wird das Freischalten von Bussen, die Speicherkontrolle oder die Auswahl von Datenformaten nicht explizit mit dem Petrinetz-Modell erfasst.

5.4.2 Petrinetz-Modell des nebenläufigen DLX-Prozessors

In diesem Abschnitt wird der vorgeschlagene Modellierungsansatz auf nebenläufige, digitale Systeme angewandt. Ausgangspunkt ist das Hardware-Modell der DLX-Pipeline in (HP96), das als Blockschaltbild in Abbildung 5.12 dargestellt ist.

Für die Modellierung von nebenläufigen Mikroprozessoren mit Petrinetzen muss in erster Linie die Struktur der Pipeline im Petrinetz-Modell exakt wiedergegeben werden. Abbildung 5.13 illustriert ein einfaches Petrinetz-Modell einer Pipeline mit fünf Verarbeitungsstufen und vier Verarbeitungspfaden. Die verschiedenen Verarbeitungspfade deuten die Abarbeitung von Befehlen aus unterschiedlichen Befehlsgruppen an. Das Petrinetz-Modell weist Free-Choice-Struktur auf. Entsprechend der gewählten Netzinterpretation werden die einzelnen Verarbeitungsstufen als Transitionen und die Übergänge zwischen den Verarbeitungsstufen als Plätze modelliert. Zur besseren Übersicht wurde bei diesem einfachen Modell auf äußere Signale verzichtet.

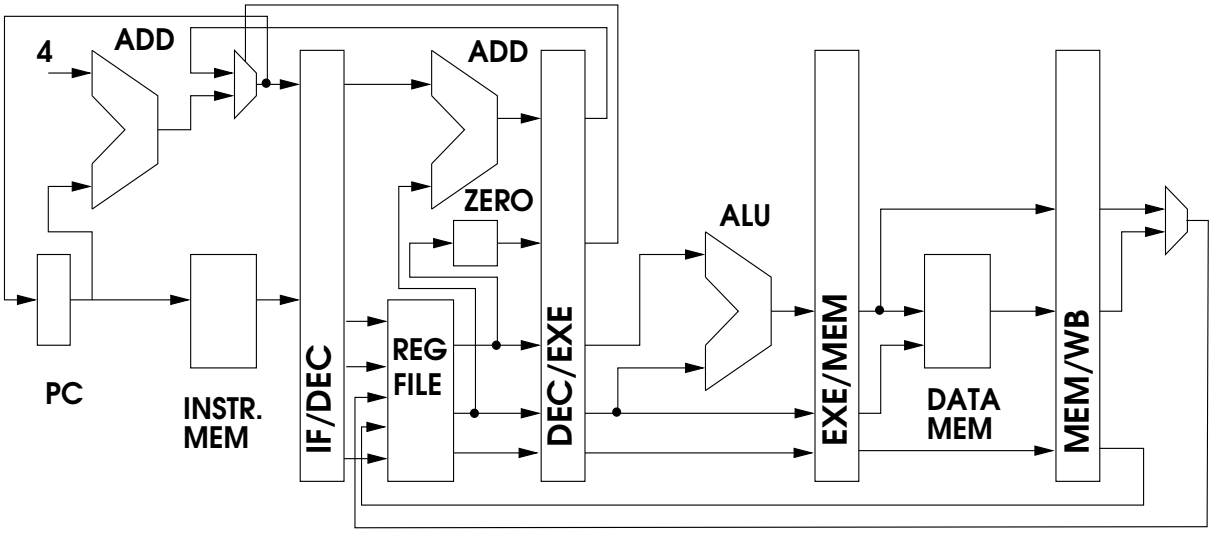

Abbildung 5.12: DLX-Pipeline

Markant sind die Kontrollplätze CP1, ..., CP4, die den gegenseitigen Ausschluss der verschiedenen Verarbeitungspfade auf einer Verarbeitungsstufe realisieren. Pro Pipeline-Stufe soll maximal ein Platz markiert sein. Diese Pipeline-Struktur bleibt auch bei der weiteren Ausarbeitung des nebenläufigen DLX-Prozessors erhalten und bildet die Grundlage für eine korrekte Wiedergabe der überlappenden Befehlsabarbeitung.

Mit der Modellierung der Abarbeitung einzelner Befehle wird dieses einfache Modell nun untersetzt. Analog zum sequentiellen Modell wird hier durch gezielte Reduktion des Befehlssatzes ein gut verständliches reduziertes Modell des nebenläufigen DLX-Prozessors behandelt. Der reduzierte Befehlssatz umfasst die in Tabelle 5.1 aufgeführten 6 Befehle. Das reduzierte Modell des nebenläufigen DLX-Prozessors ist in den Abbildungen 5.14, 5.15 und 5.16 dargestellt.

Die Analogie zwischen den Abbildungen 5.13 und 5.14 ist gut erkennbar. Bei der Untersetzung des abstrakten Modells wurde die Pipeline-Struktur erhalten. Jede Befehlsabarbeitung wird mit einem Kontrollplatz CP1, ..., CP4 synchronisiert, so dass pro Pipeline-Stufe nur ein Befehl abgearbeitet werden kann. Die Anzahl der Verarbeitungspfade in den einzelnen Verarbeitungsstufen ist nun nicht mehr konstant. In der Stufe des Befehlsholens wird nur ein Verarbeitungspfad benötigt (*if_t1*), der beim Übergang in die Dekodierstufe vier Alternativen bietet. Das sind im einzelnen die beiden ALU-Befehle FUNC_SUB und FUNC_SLT (*dec_t1*), die Speicherbefehle OP_LW und OP_SW (*dec_t2*), die Sprung- und Verzweigungsbefehle OP_J und OP_BEQZ (*dec_t3*) und der Befehl NOP (*dec_t4*).

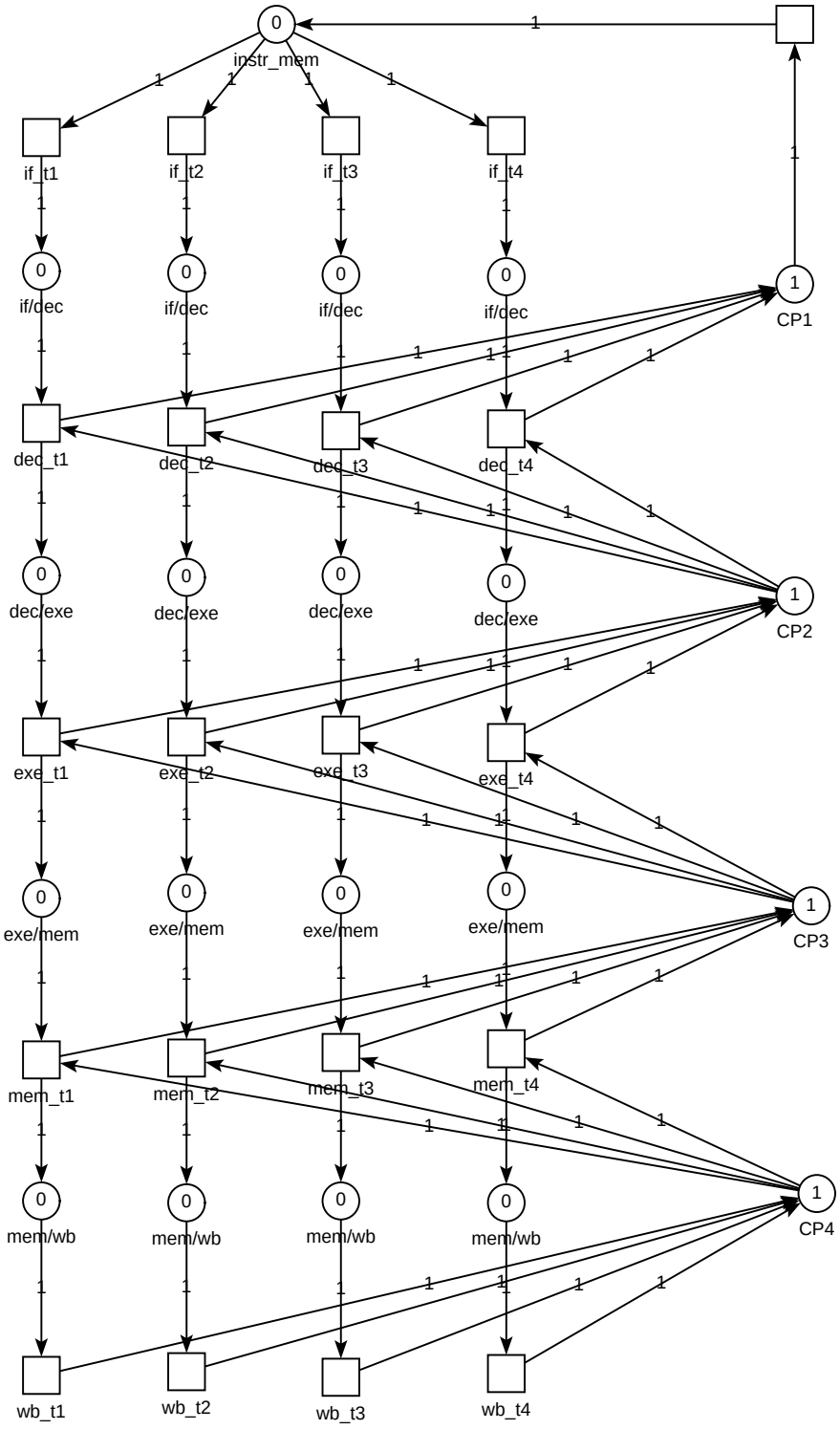

Abbildung 5.13: Vereinfachtes Petrinetz-Modell einer Prozessor-Pipeline

Zur Vermeidung von Kontroll-Hasards werden Sprung- und Verzweigungsbefehle bereits in der Dekodierstufe abgearbeitet. Das Teilnetz, welches die Abarbeitung von OP_J und OP_BEQZ modelliert, ist in Abbildung 5.15 links dargestellt und wird zwischen den beiden Plätzen *jump_dec_out* und *load_npc* eingefügt. Kontroll-Hasards entstehen, wenn ein Befehl, der in seinem Ergebnis den Befehlszähler neu lädt, erst dann ausgeführt wird, wenn weitere Befehle basierend auf dem alten Befehlszählerstand bereits Verarbeitungsstufen durchlaufen haben. Sprung- und Verzweigungsanweisungen müssen so früh wie möglich erkannt und ausgeführt werden, weshalb im Blockschaltbild Abbildung 5.12 in der Dekodierstufe etwas Kombinatorik für die Auswertung der Sprungbedingung und ein Addierer zur Berechnung des Sprungzieles eingegliedert wurde (HP96).

In der Ausführungsstufe wird zwischen den beiden Plätzen *alu_func_exe_in* und *alu_exe_out* die Ausführung der Befehle FUNC_SUB und FUNC_SLT eingebettet. Das entsprechende Teilnetz ist in Abbildung 5.15 dargestellt. Der rechte Teil der Abbildung 5.15 zeigt zwei Teilnetze zur Ausführung der Speicherbefehle OP_LW und OP_SW, die in der Speicherstufe zwischen den Plätzen *load_mem_in*, *store_mem_in* und *load_mem_out* eingebettet sind. In Abbildung 5.16 ist eine einfache Pipeline-Kontrolle modelliert, die das Ereignis RESET behandelt. Das Anhalten der Pipeline wird mit dem Signal HALT realisiert, wobei in diesem Fall durch die Schaltbedingung /HALT an allen synchronisierenden Transitionen (Abbildung 5.14) der Zustand der Pipeline eingefroren wird.

Beim vollständigen Petrinetz-Modell des nebenläufigen DLX-Prozessors wurde analog zum sequentiellen Petrinetz-Modell die Abarbeitung von 52 Befehle modelliert. Es umfasst 267 Netzknoten, die in 8 Teilnetze untergliedert sind. Im Anhang B ist das vollständige Petrinetz-Modell dargestellt. Die Abbildungen B.1 und B.2 stellen die Kontrollpfadstruktur der verschiedenen Verarbeitungsstufen dar.

Alle Teilmodelle zur Abarbeitung von Befehlen unterschiedlicher Befehlsgruppen (Abbildungen B.4- B.7) sind in die entsprechende Verarbeitungsstufe (Abbildungen B.1-B.2) eingebettet. Abbildung B.3 untersetzt die Dekodierstufe und zeigt die Abarbeitung einiger Spezialbefehle. Diese Spezialbefehle tauschen Daten zwischen Spezialregistern und Registern des Registersatzes aus (MOVS2I und MOVI2S) und realisieren die Rückkehr zur normalen Befehlsabarbeitung nach einer Ausnahmebehandlung (RFE). Wie bereits erwähnt, werden Sprung- und Verzweigungsbefehle bereits in der Dekodierstufe ausgeführt. Das entsprechende Teilnetz ist in Abbildung B.4 dargestellt. In der Ausführungsstufe werden alle Test-, Shift-, sowie logische und arithmetische Befehle (Abbildungen B.6 und B.7) abgearbeitet. Die Ausführung von Lade- und Speicherbefehlen ist in Abbildung B.5 dargestellt und wird in die Speicherstufe eingebettet.

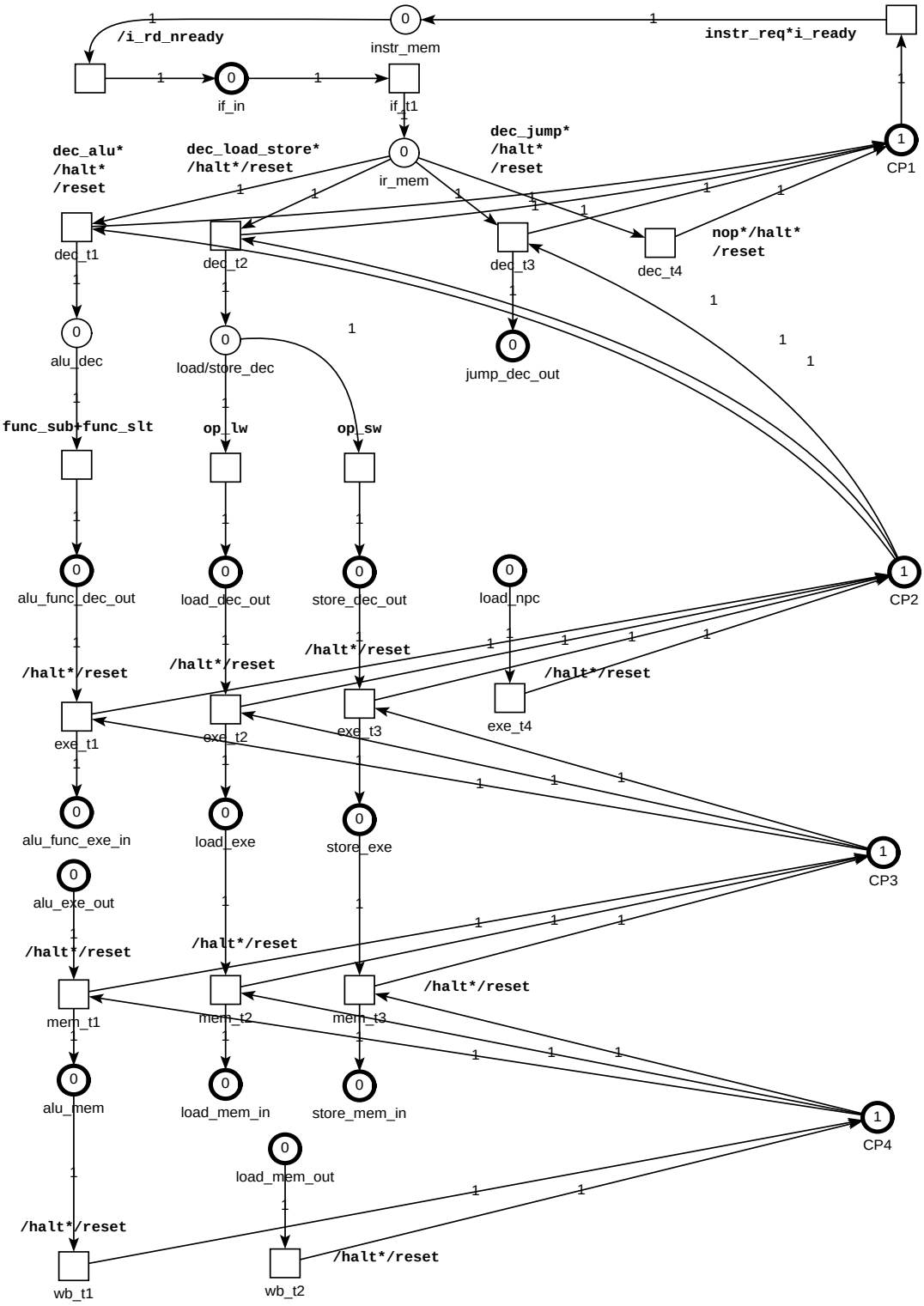

Abbildung 5.14: Reduziertes Petrinetz-Modell des nebenläufigen DLX-Prozessors (Pipeline-Struktur)

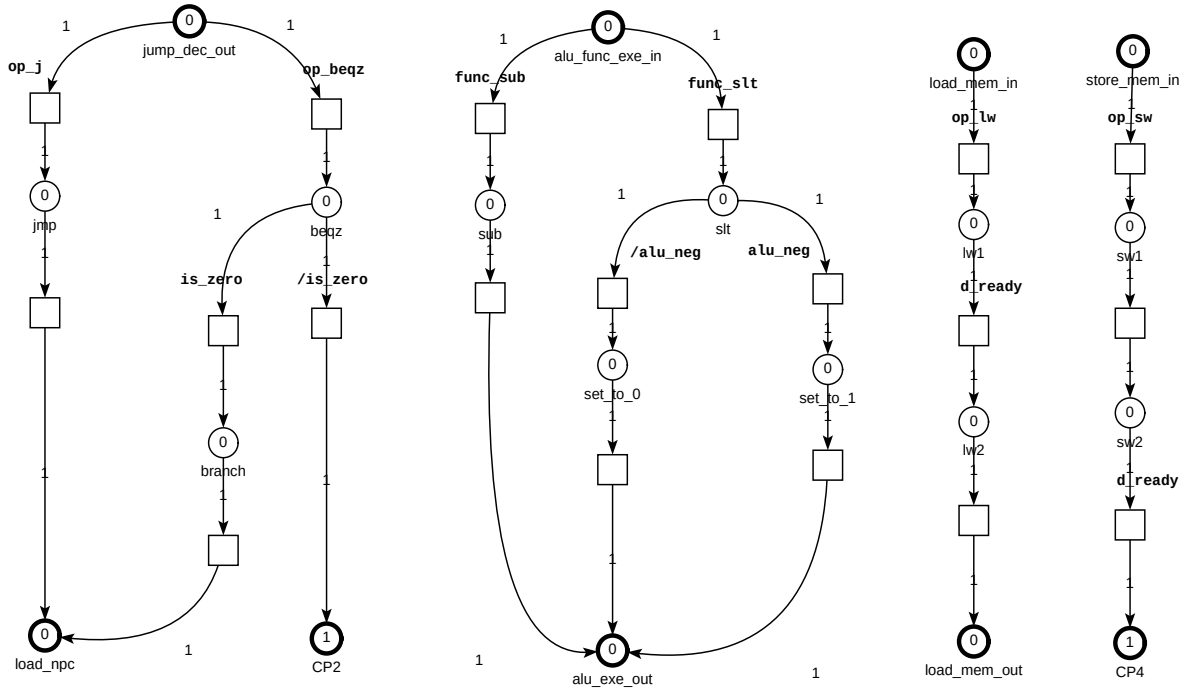

Abbildung 5.15: Reduziertes Petrinetz-Modell des nebenläufigen DLX-Prozessors (ALU- und load/store-Befehle)

Zur Behandlung von Ausnahmen werden die Zustände ERROR_STATE1, ERROR_STATE2 und ERROR_STATE3 eingeführt. Mögliche Ausnahmen sind in Tabelle 5.2 aufgelistet. Die Adressverletzungsausnahme ADR_EXC umfasst hierbei die beiden Ausnahmen ALIGN_EXC und PRIV_VIOL. Im Teilnetz Abbildung B.8 wird die Pipeline-Kontrolle mit der Behandlung von Unterbrechungen, nicht maskierbaren Unterbrechungen, Ausnahmen und dem Rücksetzen des Prozessors erweitert. Höchste Priorität haben die Ereignisse RESET und NMI. Danach folgt die Ausnahmebehandlung, wobei hier alle Ausnahmen aus Tabelle 5.2 zum Signal EXC = ADR_EXC + ILL_OP + ILL_RR_FUNC + ALU_OVFL disjunktiv verknüpft werden. Die Unterbrechung besitzt mit dem Signal INT die geringste Priorität. Die Ergebnisse der Ereignisbehandlungen werden analog zur Befehlsabarbeitung mit den entsprechenden Kontrollplätzen synchronisiert. Das Signal HALT verursacht als weiteres Kontrollsignal das Anhalten der gesamten Pipeline.

Die Einbettung der Teilnetze Abbildungen B.3 - B.8 in die Kontrollpfadstruktur der Verarbeitungsstufen wurde mit den Methoden der Hierarchisierung aus Abschnitt 5.3 erreicht. Ausgehend vom Teilnetz Abbildung B.2 ist die Hierarchisierung als Verfeine-

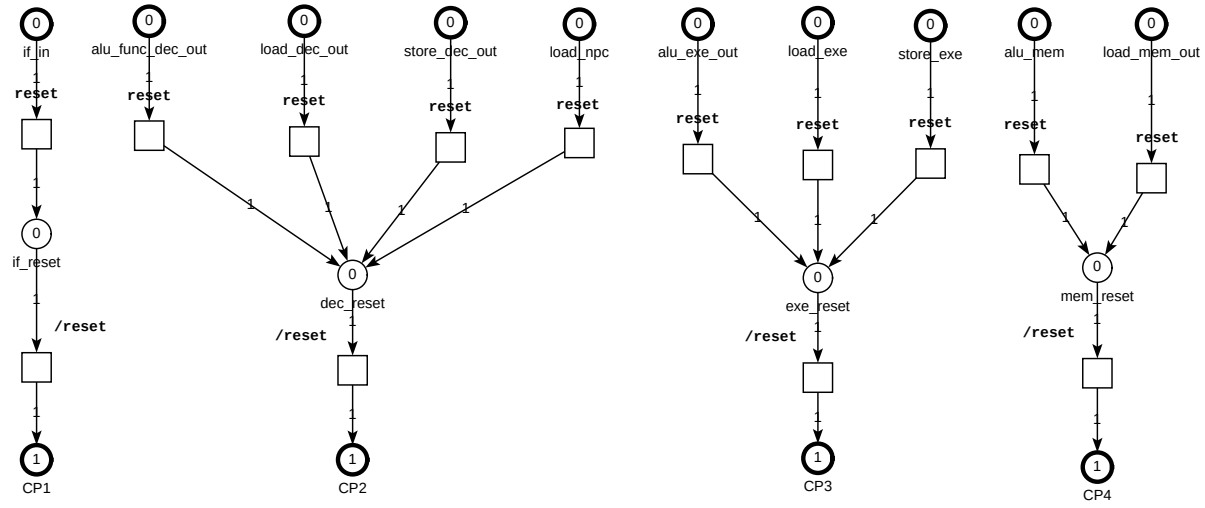

Abbildung 5.16: Reduziertes Petrinetz-Modell des nebenläufigen DLX-Prozessors (Pipeline-RESET)

Ausnahme	Bedeutung	Verarbeitungsstufe	Fehlerzustand
ADR_EXC	Adressverletzung	DEC	ERROR_STATE1
ILL_OP	illegale Befehlskodierung	DEC	ERROR_STATE1
ILL_RR_FUNC	illegale ALU-Funktionskodierung	DEC	ERROR_STATE1
ALU_OVFL	arithmetischer Überlauf	EXE	ERROR_STATE2
ADR_EXC	Adressverletzung	MEM	ERROR_STATE3

Tabelle 5.2: Ausnahmebehandlungen

rung in der Ausführungsstufe und in der Speicherstufe gut erkennbar. Angewandt wurde hierbei die Platzverfeinerung durch Transitionenverfeinerung (Abbildung 5.4) in Kombination mit sequentieller Expansion (Abbildung 5.5).

Daten-Hasards werden mit dem Petrinetz-Modell nicht erkannt. Es handelt sich hierbei um Hasards, die eine überlappende Abarbeitung von Befehlen aufgrund von Abhängigkeiten beim Zugriff auf Operanden verhindern. Das Erkennen und Auflösen von Daten-Hasards erfordert die zusätzliche Information, welche Register bei der Ausführung eines Befehls benötigt werden. Prinzipiell können die 32 Register des DLX-Registersatzes als Plätze des Petrinetz-Modells wiedergegeben werden. Jedoch ist der Grad der Verzweigung bei 32 Registern und 52 möglichen Befehlen als Schaltbedingungen sehr

hoch. Darüber hinaus muss dann mit weiteren Schaltbedingungen festgestellt werden, ob ein Ergebnis der Ausführungsstufe oder der Speicherstufe zur Befehlsausführung benötigt wird, um es gegebenenfalls zur Dekodierstufe weiterzuleiten (forwarding). Die Behandlung von Daten-Hasards kann mit einem weiteren Petrinetz-Modell realisiert werden, das an das Petrinetz-Modell des nebenläufigen DLX-Prozessors gekoppelt wird. Aus Gründen der besseren Übersicht sowie der Analysierbarkeit und Verifizierbarkeit des Gesamtsystems wird das Hardware-Modell an dieser Stelle vereinfacht modelliert.

Der Zustandsraum des Petrinetz-Modells des nebenläufigen DLX-Prozessors ist wesentlich größer als der des modellierten sequentiellen DLX-Prozessors. Die Anzahl der benötigten Netzknoten ist vergleichbar groß (sequentieller DLX: 64 Plätze und 179 Transitionen, nebenläufiger DLX: 80 Plätze und 187 Transitionen). Durch das Verhaltenskonstrukt Aufspaltung, welches in den Kontrollstrukturen der Pipeline benötigt wurde, wächst der Zustandsraum exponentiell zur Anzahl der Knoten. Bereits für das stark vereinfachte Pipeline-Modell aus Abbildung 5.13 beträgt die Anzahl der Zustände 750 bei einer Knotenanzahl von 21 Plätzen und 21 Transitionen. Das reduzierte Petrinetz-Modell des nebenläufigen DLX-Prozessors spannt mit 37 Plätzen und 52 Transitionen bereits einen Zustandsraum mit $6 \cdot 10^3$ Zuständen auf, während das vollständige Petrinetz-Modell einen Zustandsraum mit $O(10^5)$ Zuständen besitzt.

5.5 Funktionale Simulation und Validierung des Petrinetz-Modells

Die funktionale Simulation des Petrinetz-Modells ist eine erste Möglichkeit zum Test des Verhaltens des modellierten Systems bereits während des Modellierungsprozesses. Erreicht wird das durch eine schrittweise oder automatische Markenanimation des Petrinetz-Modells. Nach vollendetem Modellierungsprozess wird das Petrinetz-Modell mit gezielter funktionaler Simulation validiert.

Jeder Befehl des DLX-Prozessors wird als eine Zustandsfolge modelliert, in der erster und letzter Zustand identisch sind. Innerhalb einer solchen Zustandsfolge kann es Alternativen geben, wie zum Beispiel bei BRANCH- oder TEST-Befehlen. Bei einer sukzessiven Modellierung des Kontrollpfades kann mit der schrittweisen Markenanimation systematisch jedes neue Teilnetz, jeder modellierte Befehl durchlaufen und funktional überprüft werden. Der Entwickler kann Modell und Systemspezifikation vergleichen und ist in der Lage, weitere Funktionalität in das Modell einzugliedern. In Abschnitt 6.4.1 werden diese Zustandsfolgen als Kreise im Erreichbarkeitsgraphen quantitativ untersucht, wodurch eine Optimierung des Petrinetz-Modells und folglich des Hardware-Entwurfs erreicht wird.

Die Markenanimation ermöglicht eine grobe Einschätzung der Funktionsfähigkeit sowie das Auffinden grober Modellierungsfehler und wird mit Simulationsanwendungen des Petrinetz-Kerns umgesetzt. Hier wird der Vorteil der hohen Ausdruckskraft eines Petrinetz-Modells deutlich. Das digitale System kann bezüglich Struktur und Verhalten sehr transparent modelliert werden, so dass bereits in dieser Phase Entwurfsfehler wesentlich leichter zu entdecken sind als in einem adäquaten VHDL-Modell. Die funktionale Simulation ist in zwei Stufen durchführbar.

5.5.1 Simulation mit Auswertung äußerer Schaltbedingungen

Die Auswahl eines bestimmten Befehls wird nach SIPN mit äußeren Schaltbedingungen der Transitionen realisiert. Diese äußeren Schaltbedingungen repräsentieren Kombinationen verschiedener Steuersignale, die eine Transition schaltbar machen, wenn ihr Wert wahr ist. So können verschiedene Steuerzustände ausgewählt werden, die fortgesetzt einen Befehlszyklus ergeben. Mehrere Befehlszyklen ergeben ein Programm, dessen sequentieller Ablauf als Petrinetz ohne Alternativen darstellbar ist. Eine spezielle Simulationsanwendung des Petrinetz-Kerns ermöglicht die Darstellung des Programmablaufes im Petrinetz-Modell mit einem Petrinetz, welches die modellierten Prozesse/Befehlszyklen anschaulich wiedergibt.

5.5.2 Simulation alternativer Befehlssequenzen

Diese funktionale Simulation lässt alle äußeren Schaltbedingungen unbeachtet und es wird per Zufall entschieden, welche Zustandsfolgen entstehen. Ein Stop der Markenanimation zu einem beliebigen Zeitpunkt ist gleichbedeutend mit dem Auffinden eines toten erreichbaren Zustandes. In diesem Fall befindet sich das Petrinetz-Modell in einem Zustand, in dem keine Transition aktivierbar ist. Die Fortschaltfähigkeit ist nicht gewährleistet.

Kapitel 6

Verifikation digitaler Systeme mit Interpretierten Petrinetzen

6.1 Verifikationsziele

Das Verifikationsziel der formalen Verifikation digitaler Systeme mit Petrinetzen ist die Ableitung einer Aussage bezüglich der vollständigen Korrektheit des Petrinetz-Modells auf funktionaler Ebene. Dazu werden, wie in Kapitel 4 vorgestellt, Eigenschaften des Petrinetzes untersucht und für das Petrinetz-Modell interpretiert. Die Realisierung des in Abbildung 4.2 schematisierten Verifikationsablaufes erfordert eine Verifikationsstrategie, die festlegt, welche Petrinetz-Eigenschaften in welcher Reihenfolge untersucht werden und wie die Ergebnisse der Petrinetz-Analyse für das Petrinetz-Modell interpretiert werden (ERS01b).

Tabelle 6.1 fasst die für das Petrinetz geforderten Eigenschaften und deren Interpretation bezüglich des Petrinetz-Modells zusammen. Die Festlegung der Eigenschaften basiert auf dem gegebenen Umfang der Analysemethoden für Petrinetze, welche in Abschnitt 2.3 besprochen wurden. Die Interpretation einer solchen Eigenschaft für das Petrinetz-Modell ist stark durch die gewählte Netzinterpretation und somit den technischen Anwendungsbereich aber auch durch die Netzspezifikation bestimmt. In den folgenden beiden Abschnitten werden die einzelnen Interpretationen der Petrinetz-Eigenschaften untersetzt.

Der praktische Schritt nach der Festlegung der Petrinetz-Eigenschaften für die Analyse des Petrinetz-Modells ist das Erstellen von Analysestrategien, mit denen alle Modellierungsfehler, die eine geforderte Eigenschaft verletzen, detektiert, lokalisiert und eliminiert werden. Folgend werden die Analysestrategien zu einem Verifikationsschema zusammengesetzt, das entsprechend der Erfordernisse der Anwendung die Analysestrategien in eine bestimmte Reihenfolge bringt. Damit werden Fallunterscheidungen

Eigenschaften des Petrinetzes	Interpretation für das Petrinetz-Modell
Zustandsmaschinenstruktur (Def. 2.10)	Struktur des sequentiellen Kontrollpfades
Free-Choice-Struktur (Def. 2.8)	Struktur des nebenläufigen Kontrollpfades
Beschränktheit (Def. 2.19)	endlicher Zustandsraum
1-Beschränktheit (Def. 2.20)	korrekte Belegung des Zustandsspeichers, Transparenz des Signalflusses
Lebendigkeit (Def. 2.22, 2.26)	Fortschaltfähigkeit des Kontrollpfades
starker Zusammenhang (Def. 2.11)	strukturelle Voraussetzung für die Fortschaltfähigkeit
Konservativität (Def. 2.17)	keine Einschränkung der Fortschaltfähigkeit und der Transparenz des Signalflusses
keine Quellen und Senken (Def. 2.14, 2.15)	keine Einschränkung der Fortschaltfähigkeit und der Transparenz des Signalflusses
Reversibilität (Def. 2.23)	Rücksetzbarkeit des Kontrollpfades aus einem beliebigen Zustand in den Initialzustand
Schleifenfreiheit (Def. 2.13)	Erkennung und Zuordnung von Zustandsübergängen zwischen identischen Zuständen
Persistenz (Def. 2.25)	Auflösung dynamischer Konflikte mit Schaltbedingungen, Beseitigung von Indeterminismen

Tabelle 6.1: Petrinetz-Eigenschaften und deren Interpretation im Petrinetz-Modell

für alle Modellierungsfehler geschaffen, die eine vollständige funktionale Korrektheit des Petrinetz-Modells verhindern. Das Verifikationsschema ist die Basis für eine automatisierte Verifikation, die mit dem Petrinetz-Werkzeug VeriCon (Kapitel 7) realisiert wird.

Als weiteres Ergebnis der formalen Verifikation wird aus dem Verifikationsschema eine Anzahl von Modellierungsrichtlinien abgeleitet, die den Modellierungsprozess bereits frühzeitig beeinflussen, um Entwurfsfehler zu vermeiden. Durch die Nutzung der Verifikationsergebnisse für den Modellierungsprozess kann der Entwurfszyklus erheblich verkürzt und damit der gesamte Entwurfsablauf effizienter gestaltet werden. Die im Rahmen des Verifikationsprozesses erlangte Aussage zur Korrektheit des Petrinetz-Modells wird durch die nachfolgende Optimierung des Petrinetz-Modells nicht beeinflusst. Die untersuchten Struktur- und Verhaltenseigenschaften sind invariant bezüglich einer Optimierung des verifizierten Petrinetz-Modells.

6.2 Verifikationsstrategien für sequentielle Kontrollpfade

Dieser Abschnitt untersucht anhand eines Petrinetzes eine Reihe von Eigenschaften, die eine Aussage zur funktionalen Verifikation eines sequentiellen Kontrollpfades ermöglichen. Es werden eine Reihe von Analysestrategien vorgeschlagen, die im Abschnitt 6.5 zur Verifikation eines Mikroprozessors angewandt werden (ERS01a).

6.2.1 1-Beschränktheit

Der Nachweis der Beschränktheit (Definition 2.19) des Petrinetzes interpretiert das Petrinetz-Modell als digitales System mit einer endlichen Anzahl von Zuständen. Mit der 1-Beschränktheit (Definition 2.20) des Petrinetzes wird die Transparenz zwischen Markenfluss des Petrinetz-Modells und Signalfluss des Kontrollpfades gewährleistet. Da jede Speicherzelle der Steuerung als Platz modelliert wird, entspricht das Setzen oder Rücksetzen einer Speicherzelle einem markierten oder unmarkierten Platz im Petrinetz-Modell. Die n-Beschränktheit eines Platzes p_j mit $n > 1$ ist somit nicht sinnvoll.

In einem funktionsfähigen Kontrollpfad ist jeder Zustand in einer endlichen Anzahl Zustandsänderungen erreichbar. Es soll ein gerichteter Weg zwischen zwei beliebigen Plätzen des Petrinetz-Modells existieren. Im Petrinetz wird dafür die Struktureigenschaft des starken Zusammenhangs untersucht (Definition 2.11), die diese Bedingung erfüllt.

Um eine optimale Transparenz zwischen Petrinetz-Modell und Hardware-Realisierung zu erreichen, wird als Struktur des Petrinetzes eine Zustandsmaschinenstruktur gefordert. Hierbei befindet sich im Vor- und Nachbereich jeder Transition nur ein Netzknoten (Definition 2.10). Weil dieses Petrinetz genau einen markierten Platz aufweist, entspricht jeder Platz p_j des Petrinetz-Modells nicht nur einem Speicherplatz des Zustandsspeichers, sondern auch genau einem Zustand der Steuerung. Diese Kodierung der Steuerzustände ist als 1-aus-n Kodierung bekannt und lässt sich vorteilhaft in Hardware umsetzen. Des Weiteren wirkt sich die 1-aus-n Kodierung positiv auf eine Erreichbarkeitsanalyse aus, da jede Markierung des Erreichbarkeitsgraphen R_n des Petrinetzes einem Zustand der Steuerung entspricht.

Wenn die beiden Struktureigenschaften starker Zusammenhang und Zustandsmaschinenstruktur für das Petrinetz nachgewiesen werden konnten, ergeben sich in Abhängigkeit von der Anzahl der Marken im Initialzustand drei verschiedene Möglichkeiten der Ableitung der Verhaltenseigenschaften 1-Beschränktheit und Lebendigkeit (Satz 2.1, Satz 2.2, Folgerung 2.1). Ein unerwünschtes Verhalten des Petrinetz-Modells ist auf die Verletzung dieser Struktureigenschaften zurückzuführen.

Wird die Zustandsmaschinenstruktur des Petrinetzes verletzt und es kommt durch das Auftreten geteilter Transitionen zur Produktion $\mid t_i \bullet \mid > 1$ oder Konsumtion $\mid \bullet t_i \mid > 1$ von Marken, dann ist das Petrinetz ist nicht konservativ (Definition 2.17) und die Summe der Markenanzahl im Petrinetz ist nicht invariant. Ebenso wird die erwünschte Netzstruktur mit den nicht konservativen Konstrukten $\mid t_i \bullet \mid = 0$ und $\mid \bullet t_i \mid = 0$ gestört, was eine Beeinträchtigung der Verhaltenseigenschaften des Petrinetzes zur Folge hat.

Verletzung der Beschränktheit durch Produktion von Marken

Wenn die Prekantenmenge einer Transition t_i leer ist, dann existiert im Petrinetz eine Transitionsquelle Ft_i0 (Definition 2.14). Eine solche Transition ist stets aktiviert und feuert ununterbrochen Marken in das Petrinetz, welche zuerst auf deren Nachplatz $p_j = t_i \bullet$ gelangen. Somit ist p_j unbeschränkt und die Eigenschaft der Beschränktheit kann für das Petrinetz nicht verifiziert werden. Die strukturelle Analyse von Transitionsquellen Ft_i0 erfolgt mit der Auswertung der Bedingung $\mid \bullet t_i \mid = 0$. Aus der Erreichbarkeitsanalyse folgt, dass mindestens einer der Nachplätze p_j der Transitionsquelle unbeschränkt ist.

Unbeschränktheit kann auch durch Markenproduktion geteilter Transitionen $\mid t_i \bullet \mid > 1$ verursacht werden. Hierbei wird bei jedem Schaltvorgang von t_i mindestens eine Marke erzeugt. Wenn t_i beliebig oft schalten kann, so ist mindestens einer seiner Nachplätze $p_j = t_i \bullet$ unbeschränkt. Daraus folgt Unbeschränktheit für das gesamte Petrinetz. Strategie $\mathbf{S1}$ lokalisiert Modellierungsfehler, die durch Markenproduktion geteilter Transitionen die Unbeschränktheit des Petrinetzes verursachen.

Strategie $\mathbf{S1}$:

1. Wenn das Petrinetz N unbeschränkt ist, dann
   (a) bestimme alle geteilten Transitionen t_i mit $\mid t_i \bullet \mid > 1$ durch Auswertung von $\bullet p_j \; \forall p_j \in P \Rightarrow$ Bilde Liste $\{t_P\}$.
   (b) Bestimme Ft_i0 mit der Auswertung von $\bullet t_i \; \forall t_i \in T$.
2. Teste ob $\bullet p_j = t_i, \; t_i \in \{t_P\} \vee t_i = Ft_i0. \Rightarrow$ Die Vortransition des Platzes p_j ist eine markenproduzierende Transition $t_i \in \{t_P\}$ oder eine Transitionsquelle Ft_i0. t_i verursacht die Unbeschränktheit des Platzes p_j.

Im Fall eines unbeschränkten Petrinetzes ist jegliche weitere Analyse zur Verifikation des Petrinetz-Modells nicht sinnvoll.

6.2.2 Lebendigkeit

Ist das Petrinetz lebendig, dann kann im Petrinetz-Modell ein Zustandswechsel $p_{j_1} \rightarrow p_{j_2}$ erfolgen (Definition 2.22). Die Lebendigkeit des Petrinetzes wird als Fortschaltfähigkeit des Kontrollpfades interpretiert. Bei Vorhandensein toter Transitionen ermöglicht die Erreichbarkeitsanalyse eine Aussage zur abgeschwächten Lebendigkeit. Folgend werden Analysestrategien zum Auffinden der Modellierungsfehler vorgestellt, die eine Einschränkung der Lebendigkeit des Petrinetzes und damit eine Einschränkung der Fortschaltfähigkeit des Kontrollpfades verursachen. Für die Anwendbarkeit der Analysestrategien wird gefordert, dass für das untersuchte Petrinetz 1-Beschränktheit verifiziert wurde.

Einschränkung der Lebendigkeit durch fehlenden starken Zusammenhang

Ist ein System gekoppelter Zustandsmaschinen nicht stark-zusammenhängend (Definition 2.11), dann ist seine Lebendigkeit eingeschränkt. Das System ist in diesem Fall mit Komponenten mit Zustandsmaschinenstruktur (SM-Komponenten) überdeckbar, aber nicht in SM-Komponenten dekomponierbar. Um Modellierungsfehler aufzufinden, die eine eingeschränkte Fortschaltfähigkeit des Kontrollpfades verursachen, müssen alle toten Transitionen erkannt werden.

Mit der Zuordnung der toten Transitionen zu SM-Komponenten werden Aussagen zur Lebendigkeit einzelner Zustandsmaschinen getroffen. Des Weiteren müssen tote Transitionen erkannt werden, in deren Vor- und Nachbereich sich geteilte Plätze befinden, die zu verschiedenen nicht stark-zusammenhängenden SM-Komponenten gehören. Gesucht sind tote Transitionen t_i mit folgenden Eigenschaften:

(i) $\exists\, p_{j_1} :\ \bullet\, t = p_{j_1} \ \Rightarrow\ |\, p_{j_1} \bullet\,| > 1,\ p_{j_1} \in SM1.$
(ii) $\exists\, p_{j_2} :\ t \bullet = p_{j_2} \ \Rightarrow\ |\, \bullet\, p_{j_2}\,| > 1,\ p_{j_2} \in SM2.$
(iii) $SM1 \cap SM2 = \emptyset.$

Abbildung 6.1 zeigt ein sicheres Teilnetz, das aus zwei Zustandsmaschinen SM1 und SM2 besteht. Da über t_5 eine Verbindung von SM1 zu SM2 besteht, aber nicht umgekehrt, ist das Teilnetz nicht stark-zusammenhängend. Bevor eine Erreichbarkeitsanalyse durchgeführt wird, erfolgt eine Berechnung der minimalen SM-Komponenten des Netzes. Dadurch wird entschieden, ob das Petrinetz mit SM-Komponenten überdeckbar ist. Bei der Erreichbarkeitsanalyse werden je nach Lage der Anfangsmarkierung m_0 zwei Fälle unterschieden.

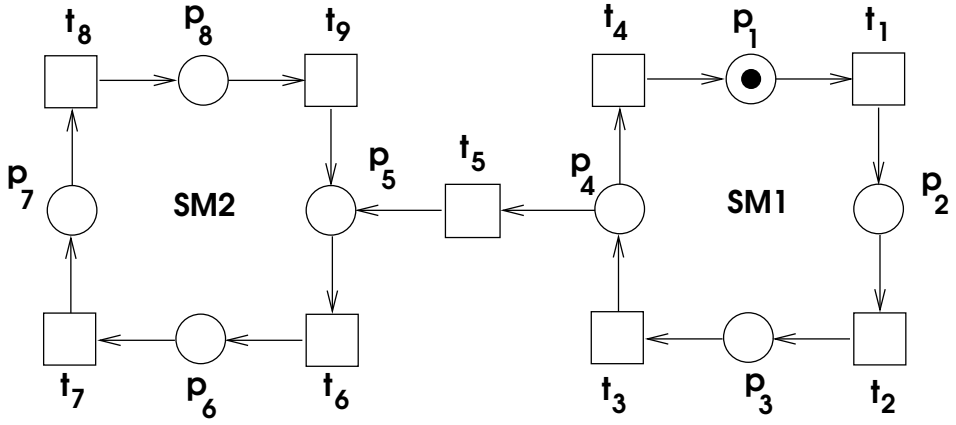

Abbildung 6.1: Nicht stark-zusammenhängende Zustandsmaschinen

a) SM1 ist bei m_0 markiert (p_1, p_2, p_3 oder p_4)

b) SM2 ist bei m_0 markiert (p_5, p_6, p_7 oder p_8)

Im Fall (a) wird bei der Berechnung des Erreichbarkeitsgraphen R_n festgestellt, dass keine toten Transitionen bei m_0 existieren. Ausgehend von m_0 kann jede Transition eine Schaltkonzession erhalten, aber nicht beliebig oft. Wird bei der Markierung $m(p_4) = 1$ der Konflikt zwischen t_4 und t_5 zugunsten von t_5 entschieden, so erhalten die Transitionen t_1 bis t_4 bei keiner folgenden Markierung eine Schaltkonzession und sind somit tote Transitionen. Auch m_0 kann nicht wieder erreicht werden, was als Nichtrücksetzbarkeit des Kontrollpfades interpretiert wird. Weiterhin wird mit R_n festgestellt, dass keine toten erreichbaren Zustände auftreten. Das heißt, es wird kein Zustand erreicht, von dem aus ein Fortschalten nicht möglich ist. Das Netz verklemmt nicht, aber es ist nicht lebendig. Übertragen auf einen beliebigen digitalen Kontrollpfad bedeutet das ein Nichterreichen einer bestimmten Anzahl von Steuerzuständen. In jedem Fall müssen tote Transitionen lokalisiert werden, damit eine geeignete Zuordnung zwischen Modellverhalten und Systemverhalten vorgenommen werden kann. Tote Transitionen und tote Zustandsmaschinen werden nach der folgenden Strategie erkannt.

Strategie $\mathbf{S2}$:

1. Teste Lebendigkeit aller Transitionen t_i $\Rightarrow$ Bilde Liste lebendiger Transitionen $\{t_L\}$.
2. Berechne stark-zusammenhängende Komponenten (SCC) in R_n $\Rightarrow$ Bilde Tupel $\{SCC_i, p_j\}$.
3. Bestimme geteilte Plätze und teilende Transitionen in jeder SCC mit der Auswertung der Vor- und Nachbereiche aller p_j: $\mid \bullet p_j \mid > 1 \vee \mid p_j \bullet \mid > 1$ $\Rightarrow$ Bilde Tupel $\{SCC_i, p_j, t_k\}$.

4. Vergleiche alle Tupel $\{SCC_i, p_j, t_k\}$. Mehrfach auftretende t_k bilden Übergänge $SCC_i \rightarrow SCC_j$ $(SM_i \rightarrow SM_j)$.
5. Vergleiche alle t_k mit den Transitionen $t_i \in \{t_L\}$. Wenn $t_k \in \{t_L\} \Rightarrow t_k$ ist lebendig. Im anderen Fall ist t_k identifiziert als tote Transition zwischen zwei SM-Komponenten.
6. Erkenne lebendige SM-Komponenten:
   $(\bullet\, p_j \in \{t_L\}\ \forall p_j \in SCC_i) \vee (p_j \bullet \in \{t_L\}\ \forall p_j \in SCC_i) \Rightarrow SCC_i$ ist lebendig.
   Nichtlebendige SM-Komponenten besitzen im Vor- oder Nachbereich mindestens einer ihrer Plätze eine tote Transition.

In Abbildung 6.1 sind die Transitionen $\{t_6, t_7, t_8, t_9\}$ lebendig und es existieren zwei stark-zusammenhängende Komponenten: $\{SCC_1, p_1, p_2, p_3, p_4\}$ und $\{SCC_2, p_5, p_6, p_7, p_8\}$. Bei Auswertung der Vor- und Nachbereiche alle Plätze der erhaltenen SCC ergeben sich die Tupel $\{SCC_1, p_4, t_4, t_5\}$ und $\{SCC_2, p_5, t_5, t_6\}$. Das heißt $p_4 \bullet = t_4, t_5,\ p_4 \in SCC_1;\ \bullet\, p_5 = t_5, t_9,\ p_5 \in SCC_2$. Ein Vergleich der beiden Tupel zeigt, dass t_5 den Übergang $SCC_1 \rightarrow SCC_2$ mit den Kanten f_{p_4,t_5}, f_{t_5,p_5} realisiert. t_5 ist eine tote Transition, da t_5 nicht in $\{t_L\} = \{t_6, t_7, t_8, t_9\}$ enthalten ist. Im Nachbereich aller Plätze der Komponente SCC_2 befinden sich ausschließlich lebendige Transitionen. Für SCC_2 gilt: $p_{j_{|j=5..8}} \bullet = \{t_6, t_7, t_8, t_9\} = \{t_L\}$. Somit ist die Komponente SCC_2 lebendig und die Kante f_{t_5,p_5} bildet den Übergang zwischen toten und lebendigen Netzbereichen.

Im Fall (b) treten bereits bei m_0 tote Transitionen auf. Es sind die Transitionen t_1, t_2, t_3 und t_4 aus SM1, die bei keiner von m_0 erreichbaren Markierung eine Schaltkonzession erhalten. Das Petrinetz ist verklemmungsfrei, da immer eine Transition aus SM2 eine Schaltkonzession besitzt. Die Strategie **S2** kann wie im Fall (a) angewandt werden und liefert neben der Erkennung toter und lebendiger Netzbereiche genau die tote Transition, die den Übergang zwischen beiden Bereichen kennzeichnet. Die Liste lebendiger Transitionen enthält die Elemente $\{t_L\} = \{t_6, t_7, t_8, t_9\}$. SM1 ist eine tote und SM2 eine lebendige Netzkomponente. Der Übergang wird mit Transition t_5 realisiert.

Diese Analyse kann für n SM-Komponenten in einem gewöhnlichen, SM-überdeckbaren und sicheren Petrinetz mit fehlendem starken Zusammenhang verallgemeinert werden.

Einschränkung der Lebendigkeit durch Konsumtion von Marken

Eine Konsumtion von Marken erfordert die Existenz geteilter Transitionen. Transitionen mit mehreren Plätzen in ihrem Vorbereich sind in der Lage, Marken zu konsumieren.

Beim Auftreten geteilter Transitionen ist die Anzahl der Marken im Petrinetz keine Erhaltungsgröße. Für eine transparente Abbildung eines sequentiellen Kontrollpfades in ein Petrinetz-Modell ist die Erhaltung der Markenanzahl im Petrinetz beim Schalten beliebiger Transitionen eine notwendige Bedingung. In einem nicht konservativen Petrinetz-Modell kann die Anzahl der gleichzeitig gesetzten Zustände variieren, was zu einer Fehlfunktion des Kontrollpfades führt. Zum Auffinden dieses Modellierungsfehlers müssen alle geteilten Transitionen t_i mit $\mid \bullet t_i \mid > 1$ erkannt werden. Die Transitionen werden mit den Ergebnissen der Erreichbarkeitsanalyse verglichen, und die Ursache der Einschränkung der Lebendigkeit wird bestimmt.

Strategie $\mathbf{S3}$:

1. Bestimme geteilte Transitionen t_i mit $\mid \bullet t_i \mid > 1$ mit der Auswertung von $p_j \bullet \forall p_j \in P \Rightarrow$ Bilde Liste $\{t_K\}$.
2. Bestimme den Nachbereich der Plätze, die durch tote erreichbare Zustände gekennzeichnet sind. Befindet sich im Nachbereich eine tote Transition, die in $\{t_K\}$ enthalten ist, so ist diese Transition die Ursache für das Auftreten des toten erreichbaren Zustandes.
3. Vergleiche $\{t_K\}$ mit den Transitionen, die bei m_0 tot sind. Sind in $\{t_K\}$ tote Transitionen enthalten, dann ist die eingeschränkte Lebendigkeit auf diese geteilten Transitionen zurückzuführen.

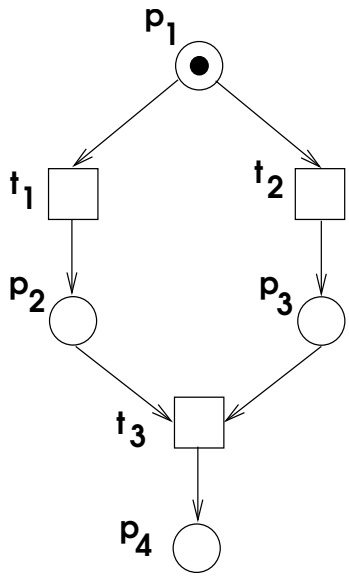

Abbildung 6.2: Teilnetz mit Markenkonsumtion

Die Einschränkung der Lebendigkeit durch Markenkonsumtion ist in Abbildung 6.2 illustriert. Für die Transition t_1 gilt $\mid \bullet t_3 \mid > 1$. Diese Transition kann Marken konsumieren, die sich in p_2 und p_3 befinden und gibt eine Marke in p_4 aus. Somit ist das Petrinetz nicht konservativ und die Zustandsmaschinenstruktur ist nicht gewährleistet, da die Bedingung $\mid t \bullet \mid = \mid \bullet t \mid = 1 \; \forall t \in T$ nicht erfüllt ist. t_3 wird bei der Analyse der Struktur des Petrinetzes als geteilte Transition erkannt, so dass $\{t_K\} = t_3$. Mit der Erreichbarkeitsanalyse wird verifiziert, an welcher Stelle des Petrinetzes die Transition t_3 die Lebendigkeit einschränkt.

Die Berechnung von R_n ergibt, dass zwei tote erreichbare Zustände existieren und dass eine Transition bei m_0 tot ist. Die toten erreichbaren Zustände sind mit den Markierungen $m(p_2) = 1$ und $m(p_3) = 1$ gekennzeichnet. Für den Nachbereich von p_2 und p_3 gilt: $p_2 \bullet = p_3 \bullet = t_3$. Es folgt, dass t_3 die Ursache für das Auftreten der beiden toten Zustände ist. Weiterhin ist t_3 eine bei der Initialmarkierung $m_0 = m(p_1)$ tote Transition. Weil im Petrinetz keine Markenproduktion ($\mid t_i \bullet \mid > 1$) stattfindet, ist in jedem von m_0 erreichbaren Zustand genau ein Platz markiert. Deswegen erhält t_3 bei keiner von m_0 erreichbaren Markierung eine Schaltkonzession und $\mid \bullet t_3 \mid > 1$ wird als Modellierungsfehler erkannt.

Tritt neben Markenkonsumtion auch Markenproduktion auf, so existiert ein Spezialfall, bei dem ein nicht konservatives Petrinetz 1-beschränkt und lebendig ist. Das Petrinetz-Modell besitzt somit die erforderlichen Verhaltenseigenschaften, aber nicht die korrekte Struktur einer Zustandsmaschine. Für ein lebendiges und sicheres Petrinetz, das nicht konservativ ist, wird dieser Modellierungsfehler mit der Strategie $\mathbf{S4}$ erkannt. Die mit $\mid t_i \bullet \mid > 1$ produzierten Marken werden nach einer endlichen Anzahl Zustandsänderungen von allen $\mid \bullet t_i \mid > 1$ konsumiert, was mit Punkt 4 der Analysestrategie überprüft wird.

Strategie $\mathbf{S4}$:

1. Verifiziere 1-Beschränktheit und Lebendigkeit für das Petrinetz N.
2. Bestimme geteilte Transitionen t_i mit $\mid t_i \bullet \mid > 1$ mit der Auswertung von $\bullet p_j \; \forall p_j \in P \; \Rightarrow$ Bilde Liste $\{t_a\}$.
3. Bestimme geteilte Transitionen t_i mit $\mid \bullet t_i \mid > 1$ mit der Auswertung von $p_j \bullet \; \forall p_j \in P \; \Rightarrow$ Bilde Liste $\{t_b\}$.
4. Überprüfe, ob: $\sum_{i=1}^{n}(\mid t_{a_i} \bullet \mid -1) = \sum_{i=1}^{n}(\mid \bullet t_{b_i} \mid -1)$.

Einschränkung der Lebendigkeit durch Quellen und Senken

Alle Möglichkeiten des Auftretens von Quellen und Senken (Definitionen 2.14, 2.15), auch einseitige Netzknoten genannt, beeinträchtigen die Lebendigkeit oder die Beschränktheit des Petrinetzes. Die Transitionssenke sowie die Platzquelle und -senke beeinträchtigen die Lebendigkeit des Petrinetzes. In Abbildung 6.3 sind die drei Fälle dargestellt. Das Auftreten einer Transitionsquelle $Ft0$ beeinträchtigt die Beschränktheit der Petrinetzes und wurde in Abschnitt 6.2.1 analysiert.

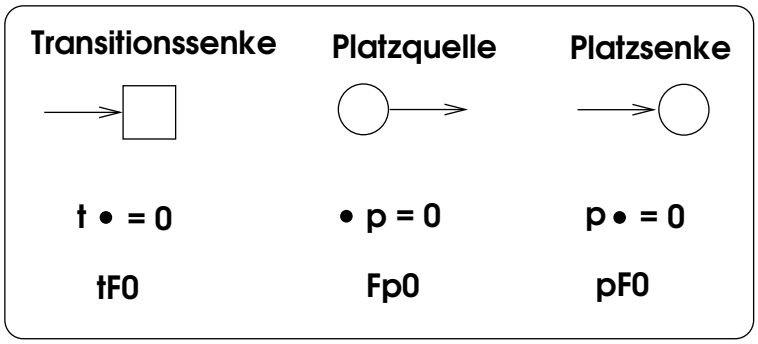

Abbildung 6.3: Quelle und Senken, die zur Einschränkung der Lebendigkeit führen

Transitionssenke: Existiert eine Transitionssenke t_iF0 im Petrinetz, so ist die Zustandsmaschinenstruktur des Petrinetzes gestört, da $\mid t_i \bullet \mid = 1$ nicht erfüllt ist. Wenn der Vorplatz der Transitionssenke markiert ist, dann wird nach dem Schalten von t_i ein toter Zustand eingenommen, weil sich keine Marke im Petrinetz befindet und der Fortschaltvorgang gestoppt wird. Das Petrinetz ist aufgrund der Markenkonsumtion bei t_i nicht konservativ. Die Lokalisierung des Modellierungsfehlers und die Analyse der durch t_iF0 hervorgerufenen eingeschränkten Lebendigkeit wird mit den folgenden Schritten realisiert.

Strategie $\mathbf{S5}$:

1. Bestimme t_iF0 mit der Auswertung von $t_i \bullet \; \forall t_i \in T$.
2. Existiert ein toter erreichbarer Zustand mit $m(p_j) = 0, \; \forall p_j \in P$, dann ist die eingeschränkte Lebendigkeit auf t_iF0 zurückzuführen.

Platzquelle: Eine Platzquelle Fp_j0 verursacht tote Transitionen t_i, die sich in deren Nachbereich befinden. Bezüglich der Initialmarkierung sind für t_i zwei Fälle zu unterscheiden: Wenn die Platzquelle mit $m_0 = m(p_j) = 1$ im initialen Zustand markiert ist, so kann t_i genau einmal schalten. Wenn die Platzquelle im initialen Zustand nicht markiert ist, dann kann t_i nie schalten.

Strategie $\mathbf{S6}$:

1. Bestimme Fp_j0 mit der Auswertung von $\bullet p_j \; \forall p_j \in P$.
2. Alle $t_i = p_j \bullet$, $p_j = Fp_j0$ sind tote Transitionen.
3. Existiert eine bei m_0 tote Transition im Nachbereich von Fp_j0, dann ist die eingeschränkte Lebendigkeit auf Fp_j0 zurückzuführen.

Platzsenke: Eine Platzsenke p_jF0 verursacht einen toten erreichbaren Zustand, von dem aus der Kontrollpfad keinen weiteren Zustandsübergang realisieren kann. p_jF0 wird strukturell mit der Bedingung $\mid p_j \bullet \mid \; = 0$ erkannt.

Strategie $\mathbf{S7}$:

1. Bestimme p_jF0 mit der Auswertung von $p_j \bullet \; \forall p_j \in P$.
2. Existiert ein toter erreichbarer Zustand mit $m(p_j) = 1$, dann ist die eingeschränkte Lebendigkeit auf p_jF0 zurückzuführen.

Das gleichzeitige Auftreten aller drei Fälle t_iF0, Fp_j0 und p_jF0 verursacht eine Überlagerung verschiedener Beeinträchtigungen der Lebendigkeit, so dass diese mit einfacher Strukturanalyse und der Auswertung des Erreichbarkeitsgraphen bezüglich toter erreichbarer Zustände und toter Transitionen erkannt und behoben werden.

6.2.3 Reversibilität

In den meisten Fällen ist es sinnvoll, einen rücksetzbaren Kontrollpfad zu modellieren. Ein rücksetzbarer Kontrollpfad kann aus jedem Zustand in endlich vielen Zustandsübergängen in den Anfangszustand gelangen. Im Petrinetz-Modell wird dafür die Eigenschaft der Reversibilität (Definition 2.23) untersucht. Reversibilität erfordert einen starkzusammenhängenden Erreichbarkeitsgraphen.

6.2.4 Schleifenfreiheit

Mit der Eigenschaft der Schleifenfreiheit (Definition 2.13) kann entschieden werden, ob strukturell modellierte Zustandsübergänge zwischen identischen Zuständen $p_j \rightarrow p_j$ des Kontrollpfades erwünscht sind.

Eine Modellierung unter Verwendung von Schleifen erscheint sinnvoll, wenn der qualitative Unterschied zwischen verschiedenen äußeren Schaltbedingungen strukturell hervorgehoben werden soll. Folgendes Beispiel verdeutlicht die Situation. Bei der Modellierung eines Kontrollpfades wird die Entscheidung zur Darstellung der äußeren Schaltbedingungen G_1 und G_2 getroffen. Wie Abbildung 6.4 zeigt, existieren dabei unterschiedliche Sichtweisen zur Verwendung von Schleifen.

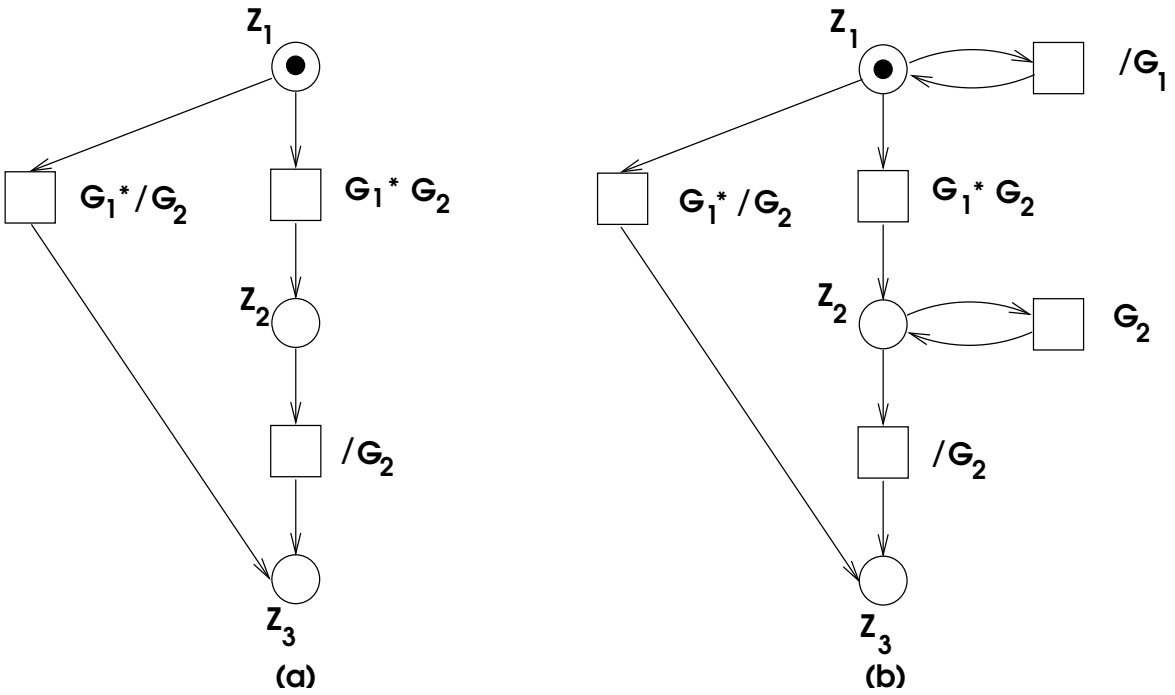

Abbildung 6.4: Schleifenfreies und schleifenbehaftetes Teilnetz

Fall (a): Ein Zustandsübergang $p_j \rightarrow p_j$ ist innerhalb des Kontrollpfades nicht sinnvoll, und die Schleifenfreiheit des Petrinetzes folglich ist eine für die funktionale Verifikation des Petrinetz-Modells erforderliche Eigenschaft.

Fall (b): Die äußeren Schaltbedingungen $G_1 = 0$ und $G_2 = 1$ werden mit Schleifen modelliert. Im Petrinetz-Modell wird mit diesen Strukturen zum Ausdruck gebracht, dass der Zustandswechsel in verschiedenen Fällen vom Wert einer äußeren Schaltbedingung abhängt. In Abbildung 6.4 wird für Fall (b) die Schaltbedingung $G_1 = 0$ mit einer Schleife modelliert. Ungeachtet des Wertes für G_2 kann vom Zustand Z_1 erst ein Übergang zum Zustand Z_2 oder Z_3 erfolgen, wenn $G_1 = 1$ ist. Gelangt der Kontrollpfad in den Zustand Z_2 durch die Schaltbedingung $G_2 * G_1 = 1$, so kann dieser Zustand erst verlassen werden, wenn die Schaltbedingung G_2 nicht mehr aktiv ist. Deshalb wird die

äußere Schaltbedingung G_2 an dieser Stelle mit einer Schleife zum Zustand Z_2 modelliert.

Bei einer Modellierung entsprechend Fall (a) ist jede im Petrinetz auftretende Schleife unerwünscht. Wird bei der Modellierung Fall (b) gewählt, so wird Analysestrategie **S8** angewandt, um eine Unterscheidung zwischen erwünschten und unerwünschten Schleifen zu ermöglichen. Bevor die Schleifen erkannt und zugeordnet werden, erfolgt die Verifikation der bisher betrachteten Verhaltens- und Struktureigenschaften des Petrinetzes (Punkt 1 und 2 der Strategie **S8**).

Strategie S8:

1. Verifiziere $\mid t_i \bullet \mid \; = \; \mid \bullet \, t_i \mid \; = 1, \forall t_i \in T$.
2. Verifiziere 1-Beschränktheit und Lebendigkeit für das Petrinetz N.
3. Bestimme die Transitionen t_i mit $t_i = p_j \bullet = \bullet \, p_j, \; \forall p_j \in P \; \Rightarrow$ Bilde Liste $\{t_s\}$.
4. Vergleiche jedes t_{s_i} mit dem Petrinetz-Modell nach den Kriterien:
   (a) Kann der Transition $t_i \in \{t_s\}$ eine äußere Schaltbedingung zugeordnet werden?
   (b) Unterscheidet sich diese äußere Schaltbedingung qualitativ von anderen äußeren Schaltbedingungen bezüglich des Modellinhaltes?
5. Wird eines der beiden Kriterien für eine Transition $t_i \in \{t_s\}$ nicht erfüllt, so ist die Schleife $t_i = p_j \bullet = \bullet \, p_j$ unerwünscht und wird als Modellierungsfehler erkannt.
6. Werden die Kriterien 4a und 4b für eine Transition $t_i \in \{t_s\}$ erfüllt, dann besteht die Möglichkeit einer sinnvollen Schleifenmodellierung.

Beim Punkt 4b wird subjektiv anhand des Modellgegenstandes entschieden, ob qualitativ unterschiedliche Kategorien von Eingangssignalen x_i als äußere Schaltbedingungen G_i strukturell mit Schleifen modelliert werden.

6.2.5 Statische und dynamische Konflikte

Statische Konflikte geben Aufschluss über Alternativen im Petrinetz ungeachtet deren Erreichbarkeit, während dynamische Konflikte als erreichbare Alternativen charakterisiert werden (Definition 2.25). Dynamische Konflikte bezeichnen konfliktbehaftete Transitionen, die zu erreichbaren Markierungen führen. Dabei entzieht das Schalten einer

Transition t_{i_1} einer anderen Transition t_{i_2} die Schaltkonzession. Unter der Annahme, dass dynamische Konflikte unerwünscht sind, ist ein Kontrollpfad mit dynamischen Konflikten nicht funktionsfähig, da in diesem Fall Zustandsübergänge nicht deterministisch erfolgen können.

Für ein sicheres, lebendiges und markiertes Petrinetz mit Zustandsmaschinenstruktur sind alle statisch konfliktbehafteten Transitionen auch dynamisch konfliktbehaftet.

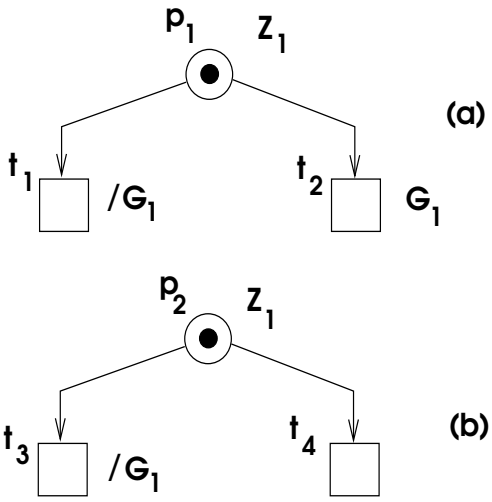

Abbildung 6.5: Konfliktbehaftete Netzstrukturen

Werden bei der Struktur- und Verhaltensanalyse des Petrinetzes Konflikte erkannt, so kann mit der Priorisierung von Transitionen mit äußeren Schaltbedingungen eine geeignete Interpretation der Konflikte erfolgen, die zu deren Auflösung führt. In Abbildung 6.5 ist im Teil (a) ein Konflikt zwischen t_1 und t_2 dargestellt, der mit den äußeren Schaltbedingungen G_1 und $/G_1$ aufgelöst wird. Das Schaltbedingung G_1 kann die beiden Werte 0 und 1 annehmen. Im ersten Fall erhält t_1 eine Schaltkonzession, im zweiten t_2. Bei (b) kann der Konflikt nicht aufgelöst werden, da bei Anliegen der Schaltbedingung $/G_1$ nicht entschieden werden kann, welche Transition schalten soll. Die Transition t_4 besitzt bei $G_1 = 0$ und $G_1 = 1$ eine Schaltkonzession. Die Priorisierung einer der beiden konfliktbehafteten Transitionen ist somit nicht ausreichend für eine deterministische Konfliktlösung. Die in Teil (b) dargestellte Situation ist deshalb als Modellierungsfehler zu interpretieren. Strategie $\mathbf{S9}$ dient der Erkennung von Modellierungsfehlern dieser Art. Die Auflösung von Konflikten erfolgt nach der Verifikation der bisher betrachteten Verhaltens- und Struktureigenschaften des Petrinetzes (Punkt 1 und 2 der Strategie $\mathbf{S9}$).

Strategie $\mathbf{S9}$:

1. Verifiziere $\mid t_i \bullet \mid \; = \; \mid \bullet \, t_i \mid \; = 1, \forall t_i \in T$.
2. Verifiziere 1-Beschränktheit und Lebendigkeit für das Petrinetz N.
3. Bestimme geteilte Plätze p_j mit $\mid p_j \bullet \mid \; > 1$ mit der Auswertung von $\bullet \, t_i \; \forall t_i \in T$.
4. Bestimme für jeden geteilten Platz p_j die Menge der Nachtransitionen $t_i = p_j \bullet \; \Rightarrow$ Bilde Tupel $\{p_j, t_i\}$.
5. Teste für jedes $\{p_j, t_i\}$,
   (a) ob alle konfliktbehafteten Transitionen t_i mit äußeren Schaltbedingungen priorisiert sind.
   (b) ob identische äußere Schaltbedingungen auftreten.
6. Wird Test 5a verifiziert und Test 5b falsifiziert $\Rightarrow$ Auflösung aller Konflikte mit äußeren Schaltbedingungen. Das Petrinetz-Modell ist persistent.

6.2.6 Anwendung der Analysestrategien

Ziel der funktionalen Verifikation des Petrinetz-Modells eines sequentiellen Kontrollpfades ist ein 1-beschränktes, lebendiges Petrinetz mit Zustandsmaschinenstruktur. Für das Petrinetz-Modell sollen alle Konflikte mit äußeren Schaltbedingungen lösbar sein und die Modellierung etwaiger Schleifen soll eindeutig interpretierbar sein. In Tabelle 6.2 sind die vorgeschlagenen Analysestrategien zum Auffinden und Beheben von Modellierungsfehlern zusammengefasst.

Strategie	Modellierungsfehler
$\mathbf{S1}$	Markenproduktion oder Transitionsquellen
$\mathbf{S2}$	fehlender starker Zusammenhang
$\mathbf{S3}$	Markenkonsumtion
$\mathbf{S4}$	Markenproduktion und -konsumtion
$\mathbf{S5}$	Transitionssenken
$\mathbf{S6}$	Platzquellen
$\mathbf{S7}$	Platzsenken
$\mathbf{S8}$	Schleifenerkennung und -zuordnung
$\mathbf{S9}$	Auftreten von Konflikten

Tabelle 6.2: Zusammenfassung der Analysestrategien für Petrinetz-Modelle sequentieller Kontrollpfade

Das Verifikationsschema für Petrinetz-Modelle sequentieller Kontrollpfade ist in Abbildung 6.6 dargestellt. Am Startpunkt wird der Erreichbarkeitsgraph konstruiert. Dieser dient als Ausgangspunkt für die Untersuchung der Eigenschaften Beschränktheit und Lebendigkeit. Weiterhin wird die Struktur des Petrinetzes untersucht, um Zustandsmaschinenstruktur und starken Zusammenhang nachzuweisen. Die Analysestrategien $\mathbf{S1}$...$\mathbf{S9}$ detektieren und lokalisieren alle Modellierungsfehler, welche die geforderten Eigenschaften nicht erhalten. Der Analyseprozess wird iterativ angewandt, bis die Analysestrategien $\mathbf{S8}$ und $\mathbf{S9}$ erfolgreich abgearbeitet wurden und somit das Ziel der Analyse erreicht ist. Die Verifikation des Petrinetz-Modells wird aus der korrekten Interpretation der Analyseergebnisse abgeleitet.

Anhand der diskutierten Modellierungsfehler und der erstellten Analysestrategien können Forderungen für den Modellierungsprozess abgeleitet werden, um bereits dort einem funktional korrekten Petrinetz-Modell nahe zu kommen.

Modellierungsrichtlinien für sequentielle Kontrollpfade:

1. Vermeidung geteilter Transitionen.
2. Vermeidung von Quellen und Senken.
3. Gewährleistung des starken Zusammenhanges.
4. Zur Konfliktlösung mit äußeren Schaltbedingungen wird gefordert, dass
   (a) alle Nachtransitionen eines geteilten Platzes mit äußeren Schaltbedingungen priorisiert sind.
   (b) die äußeren Schaltbedingungen im Nachbereich eines geteilten Platzes nicht identisch sind, und somit jeder geteilte Platz genau eine Nachtransition aktiviert.

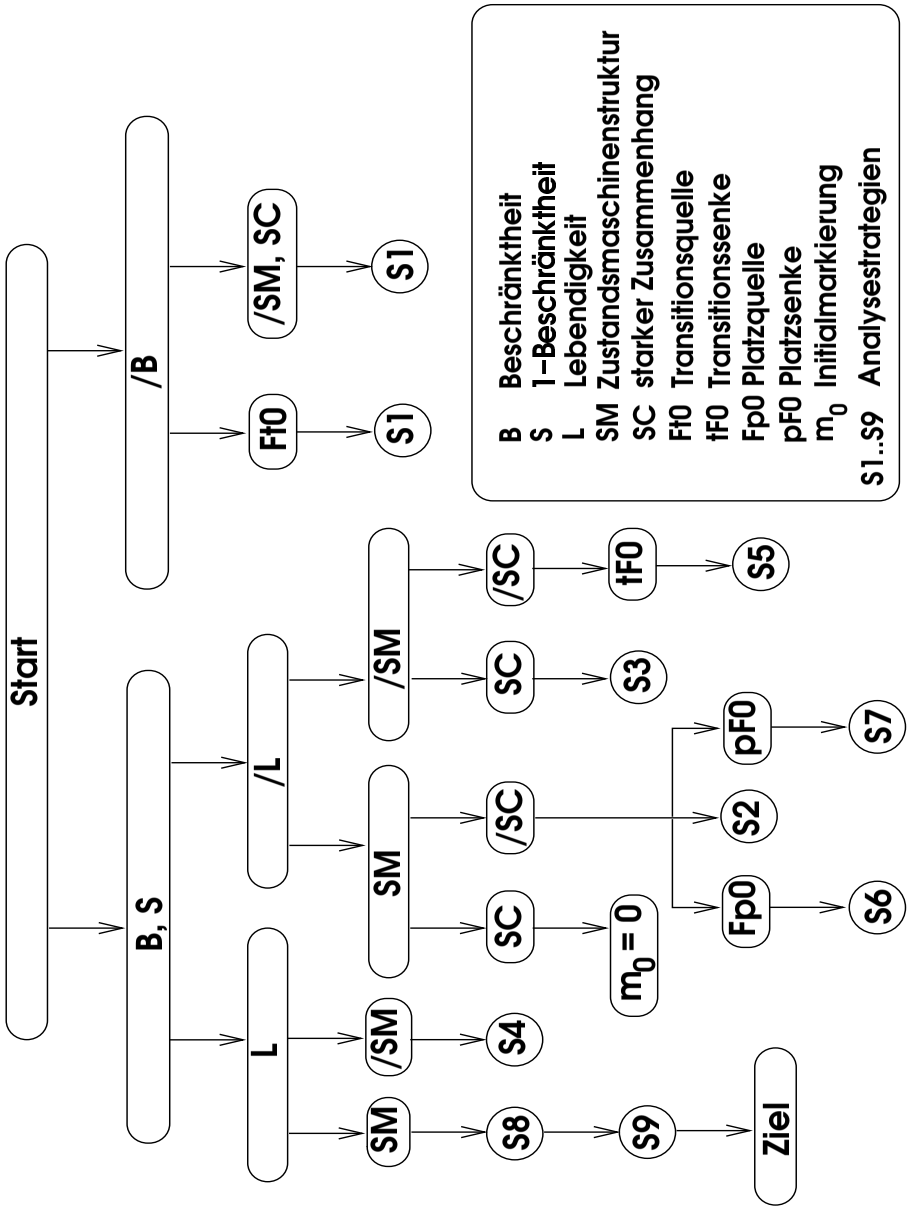

Abbildung 6.6: Verifikationsschema zur funktionalen Verifikation von Petrinetz-Modellen sequentieller Kontrollpfade

6.3 Verifikationsstrategien für nebenläufige Kontrollpfade

In diesem Abschnitt werden Verifikationsstrategien für die vollständige Verifikation nebenläufiger Kontrollpfade vorgestellt. Analog zur Darstellung der Verifikation sequentieller Kontrollpfade in Abschnitt 6.2 werden Strategien zur Detektion und Lokalisierung von Modellierungsfehlern aufgestellt, welche eine Erhaltung bestimmter notwendiger Struktur- und Verhaltenseigenschaften eines Petrinetzes verhindern. Zuerst wird geklärt, welche Petrinetz-Eigenschaften in diesem Fall erhalten bleiben müssen. Bei der Erstellung der Verifikationsstrategien sollen die bereits entwickelten Strategien (Tabelle 6.2) meistmöglich genutzt werden. Ziel ist wiederum die Ableitung eines Verifikationsschemas als Grundlage der automatisierten Verifikation sowie die Ableitung von Modellierungsrichtlinien.

Wie in den Abschnitten 5.4.1 und 5.4.2 deutlich wurde, fokussiert diese Arbeit auf Anwendungen aus dem Bereich des Entwurfes von Mikroprozessoren. Aus diesem Grund wird für die Verifikation nebenläufiger Kontrollpfade eine abstrakte Modellstruktur zugrunde gelegt, welche die Pipeline-Struktur eines Mikroprozessors vereinfacht widerspiegelt. Die Struktur des Petrinetz-Modells aus Abbildung 5.13 weist die für Prozessor-Pipelines erforderliche Pipeline-Struktur auf. In Abschnitt 5.4.2 konnte das abstrakte Petrinetz-Modell unter Erhaltung dieser Struktur untersetzt werden, bis das Petrinetz-Modell des Kontrollpfades des nebenläufigen DLX-Prozessors vorlag. Ausgehend von der erwünschten Pipeline-Struktur werden nun Petrinetz-Eigenschaften festgelegt, welche für die Verifikation des Petrinetz-Modells erhalten sein müssen.

6.3.1 Konservativität und Überdeckung mit P-Invarianten

Mit den beiden Eigenschaften Konservativität (Definition 2.17) und Überdeckung des Netzes mit P-Invarianten (Definition 2.29 und Satz 2.5) wird die Erhaltung der Markenzahl eines Petrinetzes bei beliebigen Schaltvorgängen beschrieben. Für ein konservatives Petrinetz ist für jede Transition die Summe der Markenaufnahme aus dem Vorbereich gleich der Summe der Markenabgabe an den Nachbereich. Es werden keine Marken produziert oder konsumiert, was sich günstig auf die dynamischen Netzeigenschaften auswirkt. Denn prinzipiell kann eine Markenproduktion Unbeschränktheit und eine Markenkomsumtion Nichtlebendigkeit hervorrufen.

Das Petrinetz-Modell nebenläufiger Kontrollpfade mit Pipelinestruktur benötigt keine nichtkonservativen Transitionen. Die Nebenläufigkeit, welche eine überlappende Befehlsabarbeitung ermöglicht, wird mit Transitionen eingeführt, die im Vor- und Nachbereich je zwei Plätze aufweisen. Jeweils ein Platz aus dem Vor- und Nachbereich ist

ein Kontrollplatz CP1, ..., CP4, wobei die Kanten zwischen der Transition und den Kontrollplätzen nicht gleichgerichtet sind, und die beiden Kontrollplätze zwei benachbarte Pipeline-Stufen synchronisieren. Für eine abweichende Netzstruktur mit nichtkonservativen Transitionen existieren verschiedene Möglichkeiten und Auswirkungen. Diese Modellierungsfehler werden im Zusammenhang mit der zweiten Petrinetz-Eigenschaft, die eine konstante Markensumme beschreibt, betrachtet.

Eine P-Invariante $\mathbf{y}$ des Petrinetzes N beschreibt eine Platzmenge, deren gewichtete Markensumme bei beliebigen Schaltvorgängen erhalten bleibt. Anhand von Abbildung 5.13 lässt sich dieser Sachverhalt anschaulich verdeutlichen. Das abstrakte Pipeline-Modell besitzt fünf Pipeline-Stufen. Vier Kontrollplätze sind zwischen die fünf Stufen geschaltet und im Initialzustand mit je einer Marke belegt. Das Petrinetz besitzt die folgenden vier P-Invarianten:

$$\begin{aligned}
\mathbf{y_1} &= \{p_{instr_mem}\ \ p^1_{if/dec}\ \ p^2_{if/dec}\ \ p^3_{if/dec}\ \ p^4_{if/dec}\ \ p_{cp1}\} \\
\mathbf{y_2} &= \{p^1_{dec/exe}\ \ p^2_{dec/exe}\ \ p^3_{dec/exe}\ \ p^4_{dec/exe}\ \ p_{cp2}\} \\
\mathbf{y_3} &= \{p^1_{exe/mem}\ \ p^2_{exe/mem}\ \ p^3_{exe/mem}\ \ p^4_{exe/mem}\ \ p_{cp3}\} \\
\mathbf{y_4} &= \{p^1_{mem/wb}\ \ p^2_{mem/wb}\ \ p^3_{mem/wb}\ \ p^4_{mem/wb}\ \ p_{cp4}\}
\end{aligned}$$

Für jede dieser vier Invarianten beträgt die Markensumme eines beliebigen Zustandes gleich Eins. Jede der P-Invarianten ist eine echte Invariante. Entsprechend Satz 2.5 ist das Petrinetz beschränkt. Die Invariantenanalyse kann somit bei geeigneter Netzstruktur eine Schlussfolgerung zur Beschränktheit des Petrinetzes ziehen. Auch Anzahl und Verteilung von P-Invarianten sind für die vorgeschlagene Modellstruktur charakteristisch. Modellierungsfehler, die ein nichtkonservatives Petrinetz zur Ursache haben, und die eine Überdeckung mit P-Invarianten verhindern, können die Unbeschränktheit des Zustandsraumes hervorrufen. Analog zur Verifikation sequentieller Kontrollpfade soll diese Fehlerquelle zuerst ausgeschlossen werden.

Ist das Petrinetz nichtkonservativ und nicht mit P-Invarianten überdeckbar, so werden dessen Zusammenhangseigenschaften (Definitionen 2.11 und 2.12) untersucht. Ein nicht stark-zusammenhängendes Petrinetz weist einseitige Knoten (Definitionen 2.14 und 2.15) auf oder es existiert ein Pfad im Petrinetz von dem aus keine Marke entweichen kann (geschlossener Pfad). Geschlossene Pfade sind ausschließlich in der Lage, Marken aufzunehmen und werden auch Trap genannt (Definition 2.26). Einseitige Knoten werden in Abschnitt 6.2 mit den Analysestrategien $\mathbf{S1}$, $\mathbf{S5}$, $\mathbf{S6}$ und $\mathbf{S7}$ lokalisiert. Während bei sequentiellen Kontrollpfadmodellen nur die Transitionsquelle Unbeschränktheit hevorruft, verursacht eine Platzsenke mit einer nichtkonservativen Vortransition in einem nebenläufigen Kontrollpfadmodell ebenfalls Unbeschränktheit. Strategie $\mathbf{S7}$ wird des-

halb wie folgt angepasst.

Strategie $\mathbf{S7'}$:

1. Bestimme p_jF0 mit der Auswertung von $p_j \bullet \; \forall p_j \in P$.
2. Bestimme $\mid t_i \bullet \mid \; > \; \mid \bullet t_i \mid \; \forall t_i \in T \; \Rightarrow$ Bilde Liste $\{t_{/C}\}$.
3. Teste ob $t_i \bullet = p_jF0, \; t_i \in t_{/C} \; \Rightarrow \; t_i$ ist die Ursache der unbeschränkten Platzsenke p_jF0.

Dieses Beispiel zeigt, dass die bereits hergeleiteten Analysestrategien mit wenigen Veränderungen für die Verifikation nebenläufiger Kontrollpfade genutzt werden können. Für den Fall des geschlossenen Pfades, bei dem Marken zwischen stark zusammenhängenden Netzkomponenten nur in eine Richtung fließen können, wird Strategie $\mathbf{S2}$ unverändert angewandt. Platzquellen und Transitionssenken verursachen nichtlebendige Transitionen, haben aber keinen Einfluss auf die Beschränktheit des Petrinetzes. Die Strategien $\mathbf{S5}$ (Transitionssenken) und $\mathbf{S6}$ (Platzquellen) kommen unverändert zur Anwendung. Um Transitionsquellen zu analysieren, wird ein Teil der Strategie $\mathbf{S1}$ benötigt, welcher als Analysestrategie $\mathbf{S1'}$ formuliert wird.

Strategie $\mathbf{S1'}$:

1. Bestimme Ft_i0 mit der Auswertung von $\bullet t_i \; \forall t_i \in T$.
2. Teste ob $\bullet p_j = Ft_i0. \; \Rightarrow \; Ft_i0$ verursacht die Unbeschränktheit des Platzes p_j.

Liegt ein stark zusammenhängendes Petrinetz vor, so müssen die nichtkonservativen Transitionen gefunden werden, welche Marken produzieren oder konsumieren. Mit dem zweiten Teil der Strategie $\mathbf{S1}$ werden markenproduzierende Transitionen lokalisiert, die für ein stark zusammenhängendes Petrinetz unbeschränkte Plätze zur Folge haben können.

Strategie $\mathbf{S1''}$:

1. Bestimme alle geteilten Transitionen t_i mit $\mid t_i \bullet \mid \; > 1$ durch Auswertung von $\bullet p_j \; \forall p_j \in P \; \Rightarrow$ Bilde Liste $\{t_{/C}\}$.
2. Teste ob $\bullet p_j = t_i, \; t_i \in \{t_{/C}\}. \; \Rightarrow \; t_i$ verursacht die Unbeschränktheit des Platzes p_j.

Ursache für markenproduzierende Transitionen können irrtümlich eingefügte Plätze zwischen verschiedenen Verarbeitungsstufen oder Verarbeitungspfaden sein. Darüber

hinaus bewirken fehlerhaft modellierte Kanten im Vor- und Nachbereich von Transitionen eine Markenproduktion. Die Strategie $\mathbf{S3}$ findet markenkonsumierende Transitionen und vergleicht diese mit den toten Transitionen des Petrinetzes. Als Beispiele solcher Modellierungsfehler kommen ebenfalls fehlerhaft modellierte Kanten im Vor- und Nachbereich von Transitionen in Frage, welche hierbei eine Markenkonsumtion bewirken. $\mathbf{S3}$ kann unverändert zur Verifikation nebenläufiger Kontrollpfadmodelle angewandt werden.

Im Fall eines nichtkonservativen Petrinetzes, welches nicht mit P-Invarianten übedeckbar ist, existieren somit 7 Modellierungsfehler, die nichtkonservative Transitionen zur Ursache haben und in ihrer Wirkung die Beschränktheit oder Lebendigkeit des Petrinetzes nicht erhalten. Daneben existiert eine weitere Möglichkeit, bei der ein Petrinetz mit nichtkonservativen Transitionen mit P-Invarianten übedeckbar ist. Ein Beispiel hierfür ist das Auftreten einer parallelen Expansion (Abbildung 5.5) im Petrinetz, die als Möglichkeit der Transitionenverfeinerung in Abschnitt 5.3 angegeben wurde. Es wird festgestellt, dass eine Transitionenverfeinerung durch parallele Expansion innerhalb oder zwischen verschiedenen Verarbeitungsstufen nicht wünschenswert ist für ein nebenläufiges Kontrollpfadmodell. Die Situation ist vergleichbar mit dem Modellierungsfehler, der mit Analysestrategie $\mathbf{S4}$ detektiert wird. Aufspaltungen produzieren n Marken, die nach einer Reihe von Schaltvorgängen alle mit Synchronisationen wieder konsumiert werden. Zum Auffinden des Modellierungsfehlers werden die P-Invarianten genutzt.

Strategie $\mathbf{S4'}$:

1. Bestimme die P-Invarianten des Petrinetzes N.
2. Bestimme nichtkonservative Transitionen t_i mit $\mid t_i \bullet \mid \neq \mid \bullet t_i \mid \ \forall t_i \in T$ $\Rightarrow$ Bilde Liste $\{t_{/C}\}$.
3. Bestimme aufspaltende nichtkonservative Transitionen t_i mit $\mid t_i \bullet \mid > 1 \ \forall t_i \in \{t_{/C}\} \ \Rightarrow$ Bilde Liste $\{t_a\}$.
4. Bestimme synchronisierende nichtkonservative Transitionen t_i mit $\mid \bullet t_i \mid > 1 \ \forall t_i \in \{t_{/C}\} \ \Rightarrow$ Bilde Liste $\{t_b\}$.
5. Teste $\forall \mathbf{y}, \ \forall t_i \in \{t_a\}, \ \forall t_i \in \{t_b\} : \ (t_{i_1} \bullet \in \mathbf{y}) \wedge (\bullet t_{i_2} \in \mathbf{y})$ mit $t_{i_1} \in \{t_a\}$ und $t_{i_2} \in \{t_b\}$.

Eine P-Invariante überdeckt den Nachbereich der Aufspaltungen und den Vorbereich der Synchronisationen. Die Markensumme dieser Platzmenge ist eine Erhaltungsgröße und verletzt nicht die Beschränktheit des Petrinetzes. Alle Modellierungsfehler, die nichtkonservative Transitionen zur Ursache haben, können mit der Anwendung der vorgeschlagenen Analysestrategien $\mathbf{S1'}, \mathbf{S1''}, \mathbf{S2}, \mathbf{S3}, \mathbf{S4'}, \mathbf{S5}, \mathbf{S6}$ oder $\mathbf{S7'}$ detektiert und

lokalisiert werden. Wenn nach der Herstellung der Konservativität alle Plätze des Netzes mit positiven P-Invarianten überdeckbar sind, dann weist das Petrinetz Beschränktheit auf (Satz 2.5). Nachdem ein endlicher Zustandsraum für das Petrinetz sichergestellt ist, werden im nächsten Schritt der Verifikation Analysestrategien für Modellierungsfehler gesucht, welche trotz konstanter, vom Schaltverhalten unabhängiger Markensumme die Dynamik des Netzes negativ beeinflussen.

Wiederholt werden für diesen Fall die Zusammenhangseigenschaften des Petrinetzes abgefragt. Mögliche einseitige Knoten in einem konservativen Petrinetz bestehen in der Platzquelle und der Platzsenke. Diese Modellierungsfehler beeinflussen die Lebendigkeit des Petrinetzes und werden mit den Analysestrategien $\mathbf{S6}$ und $\mathbf{S7}$ behandelt, welche unverändert aus Abschnitt 6.2 übernommen werden. Wenn das konservative Petrinetz keine einseitigen Knoten und keinen starken Zusammenhang aufweist, so existiert ein geschlossener Pfad, dessen nichtlebendige Transitionen analog zur Fehlersituation in den Abschnitten 6.3.1 und 6.2 mit Strategie $\mathbf{S2}$ lokalisiert werden.

6.3.2 1-Beschränktheit und Lebendigkeit

1-Beschränktheit ist eine Petrinetz-Eigenschaft, die für die Interpretation des nebenläufigen Kontrollpfadmodells als digitales System erhalten sein muss. In der Situation, in welcher das Petrinetz auf 1-Beschränktheit untersucht wird, weist es bereits Konservativität, Beschränktheit und einen starken Zusammenhang auf. Nach der Bestimmung der 1-Beschränkheit wird die Lebendigkeit des Petrinetzes untersucht. Aus der Kombination der beiden Analysen ergeben sich vier Möglichkeiten, von denen drei aufgrund von Modellierungsfehlern nicht erwünscht sind. Nur im Fall der Erhaltung der 1-Beschränkheit und der Lebendigkeit weist das Petrinetz keine Modellierungsfehler auf.

Ein unsicheres, aber lebendiges Petrinetz besitzt fehlerhafte Markierungen. Im Initialzustand sind die Kontrollplätze des nebenläufigen Kontrollpfadmodells mit je einer Marke belegt (Abbildungen 5.13, 5.14, B.1 und B.2). Die 1-Beschränktheit ist dann verletzt, wenn die Summe der Marken im Netz größer als die Anzahl der Kontrollplätze des Petrinetz-Modells ist.

Strategie $\mathbf{S10}$:

1. Voraussetzungen: Das Petrinetz N ist konservativ, mit P-Invarianten überdeckbar und somit beschränkt, stark zusammenhängend, unsicher und lebendig.
2. Bestimme die Anzahl der Marken im Initialzustand $M_I = \sum_{j=1}^{|P|} m_0(p_j)$.

3. Vergleiche M_I mit der Anzahl der Kontrollplätze $\Longrightarrow$ Existieren mehr Marken als Kontrollplätze, dann ist die Verletzung der 1-Beschränktheit auf diesen Modellierungsfehler zurückzuführen.

Ein unsicheres und nichtlebendiges Petrinetz besitzt Transitionen, die verschiedene Verarbeitungsstufen miteinander verbinden, ohne die Verarbeitungsstufen über Kontrollplätze zu synchronisieren. Eine solche Transition zieht aus der Verarbeitungsstufe n eine Marke ab. Da jetzt keine Marke über einen Kontrollplatz an n zurückgeliefert werden kann, verursacht diese Markenentnahme mindestens eine tote Transition in der Verarbeitungsstufe n. Es existiert ein toter erreichbarer Zustand, bei dem kein Platz der Verarbeitungsstufe n markiert ist. Die aus n entnommene Marke wird auf einem Platz abgelegt, der zu einer beliebigen Verarbeitungsstufe $\neq n$ gehört. Dieser Platz kann nun prinzipiell mit zwei Marken belegt werden. Es existiert ein erreichbarer Zustand, der die Unsicherheit für diesen Platz und für alle anderen Plätze derselben Verarbeitungsstufe nachweist. Und obwohl das Petrinetz beim Auftreten dieses Modellierungsfehlers mit P-Invarianten überdeckbar ist, existiert eine Besonderheit bezüglich der Anzahl und Verteilung der P-Invarianten. Im Idealfall existieren für ein nebenläufiges Kontrollpfadmodell mit n Verarbeitungsstufen $n-1$ P-Invarianten, was ebenfalls der Anzahl der Kontrollplätze p_{cp} entspricht. Die Invarianten sind derart angeordnet, dass jedes $\mathbf{y}$ genau eine Komponente y_{p_j}, mit $p_j = p_{cp}$, enthält. Der vorliegende Modellierungsfehler bewirkt die Vereinigung zweier P-Invarianten, so dass eine Invariante existiert, die zwei Kontrollplätze überdeckt. Die Störung des Modellstruktur wird ebenfalls mit der Anzahl und Verteilung der P-Invarianten widerspiegelt.

Strategie $\mathrm{S11}$:

1. Bestimme die P-Invarianten des Petrinetzes N.
2. Existieren für n Verarbeitungsstufen weniger als $n-1$ P-Invarianten und werden zwei Kontrollplätze von einer Invariante überdeckt:
   $\exists\, y_{p_{j_1}}, y_{p_{j_2}} \in \mathbf{y} : p_{j_1} = p_{cp_1},\ p_{j_2} = p_{cp_2}$, dann bestimme die Transitionen mit $(t_i = \bullet\, p_j) \vee (t_i = p_j\, \bullet)\ \forall p_j \in \mathbf{y}$ für diese Invariante. $\Longrightarrow$ Bilde Liste t_d.
3. Bestimme alle $t_i \in t_d$ mit
   (a) $(\bullet\, t_i \neq p_{cp}) \vee (t_i\, \bullet \neq p_{cp}) \Longrightarrow$ Bilde Liste $t_{d'}$.
   (b) Überprüfe, ob die $t_i \in t_{d'}$ Transitionen innerhalb einer sequentiellen Expansion sind.
   (c) Transitionen $t_i \in t_{d'}$ außerhalb sequentieller Expansionen verursachen die Verletzung der 1-Beschränktheit und der Lebendigkeit des Petrinetzes N.

Ein sicheres, aber nichtlebendiges Petrinetz besitzt fehlerhafte Markierungen. In diesem Fall ist die Summe der Marken im Netz kleiner als die Anzahl der Kontrollplätze.

Strategie $\mathbf{S12}$:

1. Voraussetzungen: Das Petrinetz N ist konservativ, mit P-Invarianten überdeckbar und somit beschränkt, stark zusammenhängend, sicher und nichtlebendig.
2. Bestimme die Anzahl der Marken im Initialzustand $M_I = \sum_{j=1}^{|P|} m_0(p_j)$.
3. Vergleiche M_I mit der Anzahl der Kontrollplätze $\Longrightarrow$ Existieren weniger Marken als Kontrollplätze, dann ist die Verletzung der Lebendigkeit auf diesen Modellierungsfehler zurückzuführen.

Wurden 1-Beschränktheit und Lebendigkeit für das Petrinetz sichergestellt, so wird der Erreichbarkeitsgraph auf seinen Zusammenhang überprüft. Ein stark zusammenhängender Erreichbarkeitsgraph modelliert den Zustandsraum eines rücksetzbaren digitalen Systems. Deshalb ist Reversibilität (Definition 2.23) als Eigenschaft für das Petrinetz unverzichtbar.

6.3.3 Schleifen- und Konfliktfreiheit

Die Untersuchungen der Abschnitte 6.2.5 und 6.2.4 zum Übergang zwischen zwei identischen Zuständen durch Schleifenmodellierung und zur Auflösung dynamischer Konflikte mit Schaltbedingungen sind ebenfalls auf nebenläufige Kontrollpfadmodelle anwendbar. Die Analysestrategien $\mathbf{S8}$ und $\mathbf{S9}$ müssen nur in einem Punkt angepasst werden, da die Bedingung $|\, t_i \bullet \,| = |\, \bullet\, t_i \,| = 1$ nebenläufige Aktionen ausschließt.

Strategie $\mathbf{S8'}$:

1. Verifiziere $|\, t_i \bullet \,| = |\, \bullet\, t_i \,| \;\; \forall t_i \in T$.
2. Verifiziere 1-Beschränktheit und Lebendigkeit für das Petrinetz N.
3. Bestimme die Transitionen t_i mit $t_i = p_j \bullet = \bullet\, p_j, \; \forall p_j \in P \;\Rightarrow$ Bilde Liste $\{t_s\}$.
4. Vergleiche jedes t_{s_i} mit dem Petrinetz-Modell nach den Kriterien:
   (a) Kann der Transition $t_i \in \{t_s\}$ eine äußere Schaltbedingung zugeordnet werden?
   (b) Unterscheidet sich diese äußere Schaltbedingung qualitativ von anderen äußeren Schaltbedingungen bezüglich des Modellinhaltes?

5. Wird eines der beiden Kriterien für eine Transition $t_i \in \{t_s\}$ nicht erfüllt, so ist die Schleife $t_i = p_j \bullet = \bullet\, p_j$ unerwünscht und wird als Modellierungsfehler erkannt.
6. Werden die Kriterien 4a und 4b für eine Transition $t_i \in \{t_s\}$ erfüllt, dann besteht die Möglichkeit einer sinnvollen Schleifenmodellierung.

Strategie $\mathbf{S9'}$:

1. Verifiziere $\mid t_i \bullet \mid \; = \; \mid \bullet\, t_i \mid \;\; \forall t_i \in T$.
2. Verifiziere 1-Beschränktheit und Lebendigkeit für das Petrinetz N.
3. Bestimme geteilte Plätze p_j mit $\mid p_j \bullet \mid \; > 1$ mit der Auswertung von $\bullet\, t_i \; \forall t_i \in T$.
4. Bestimme für jeden geteilten Platz p_j die Menge der Nachtransitionen $t_i = p_j \bullet \; \Rightarrow$ Bilde Tupel $\{p_j, t_i\}$.
5. Teste für jedes $\{p_j, t_i\}$,
   (a) ob alle konfliktbehafteten Transitionen t_i mit äußeren Schaltbedingungen priorisiert sind.
   (b) ob identische äußere Schaltbedingungen auftreten.
6. Wird Test 5a verifiziert und Test 5b falsifiziert $\Rightarrow$ Auflösung aller Konflikte mit äußeren Schaltbedingungen. Das Petrinetz-Modell ist persistent.

6.3.4 Anwendung der Analysestrategien

Ziel der funktionalen Verifikation des Petrinetz-Modells eines nebenläufigen Kontrollpfades ist ein konservatives, stark zusammenhängendes, 1-beschränktes, lebendiges Petrinetz. Für das Petrinetz-Modell sollen darüber hinaus alle Konflikte mit äußeren Schaltbedingungen lösbar sein, und die Modellierung etwaiger Schleifen soll eindeutig interpretierbar sein. In Tabelle 6.3 sind die vorgeschlagenen Analysestrategien zum Auffinden und Beheben von Modellierungsfehlern zusammengefasst.

Abbildung 6.7 schematisiert die Vorgehensweise bei der Anwendung der Analysestrategien. Ausgehend vom Startpunkt der Analyse werden die Eigenschaften Konservativität, Beschränktheit (mit der Berechnung der P-Invarianten), starker Zusammenhang, 1-Beschränktheit und Lebendigkeit untersucht. Entsprechend der erhaltenen Ergebnisse kommen die Analysestrategien $\mathbf{S1'}$...$\mathbf{S12}$ zum Einsatz. Dieser Vorgang wird fortgesetzt, bis die gewünschten Eigenschaften des Petrinetz-Modells gewährleistet sind. Die korrekte Interpretation der Ergebnisse des Analyseprozesses ermöglicht die Verifikation des Petrinetz-Modells. Analog zur Verifikation sequentieller Kontrollpfadmodelle

Strategie	Modellierungsfehler
$\mathbf{S1'}$	Transitionsquellen
$\mathbf{S1''}$	Aufspaltung
$\mathbf{S2}$	fehlender starker Zusammenhang
$\mathbf{S3}$	Synchronisation
$\mathbf{S4'}$	Aufspaltung und Synchronisation
$\mathbf{S5}$	Transitionssenken
$\mathbf{S6}$	Platzquellen
$\mathbf{S7}$	Platzsenken
$\mathbf{S7'}$	Platzsenken mit nichtkonservativen Vortransitionen
$\mathbf{S8'}$	Schleifenerkennung und -zuordnung
$\mathbf{S9'}$	Auftreten von Konflikten
$\mathbf{S10}$	Markensumme zu groß
$\mathbf{S11}$	nichtsync. Transitionen zwischen Verarbeitungsstufen
$\mathbf{S12}$	Markensumme zu klein

Tabelle 6.3: Zusammenfassung der Analysestrategien für Petrinetz-Modelle nebenläufiger Kontrollpfade

besteht die Möglichkeit, aus den Ergebnissen des Analyseprozesses allgemeine Modellierungsrichtlinien für den petrinetz-basierten Entwurf nebenläufiger Kontrollpfade abzuleiten.

Modellierungsrichtlinien für nebenläufige Kontrollpfade:

1. Vermeidung nichtkonservativer Transitionen.
2. Vermeidung von Quellen und Senken.
3. Gewährleistung des starken Zusammenhanges.
4. Zur Konfliktlösung mit äußeren Schaltbedingungen wird gefordert, dass
   (a) alle Nachtransitionen eines geteilten Platzes mit äußeren Schaltbedingungen priorisiert sind.
   (b) die äußeren Schaltbedingungen im Nachbereich eines geteilten Platzes nicht identisch sind, und somit jeder geteilte Platz genau eine Nachtransition aktiviert.

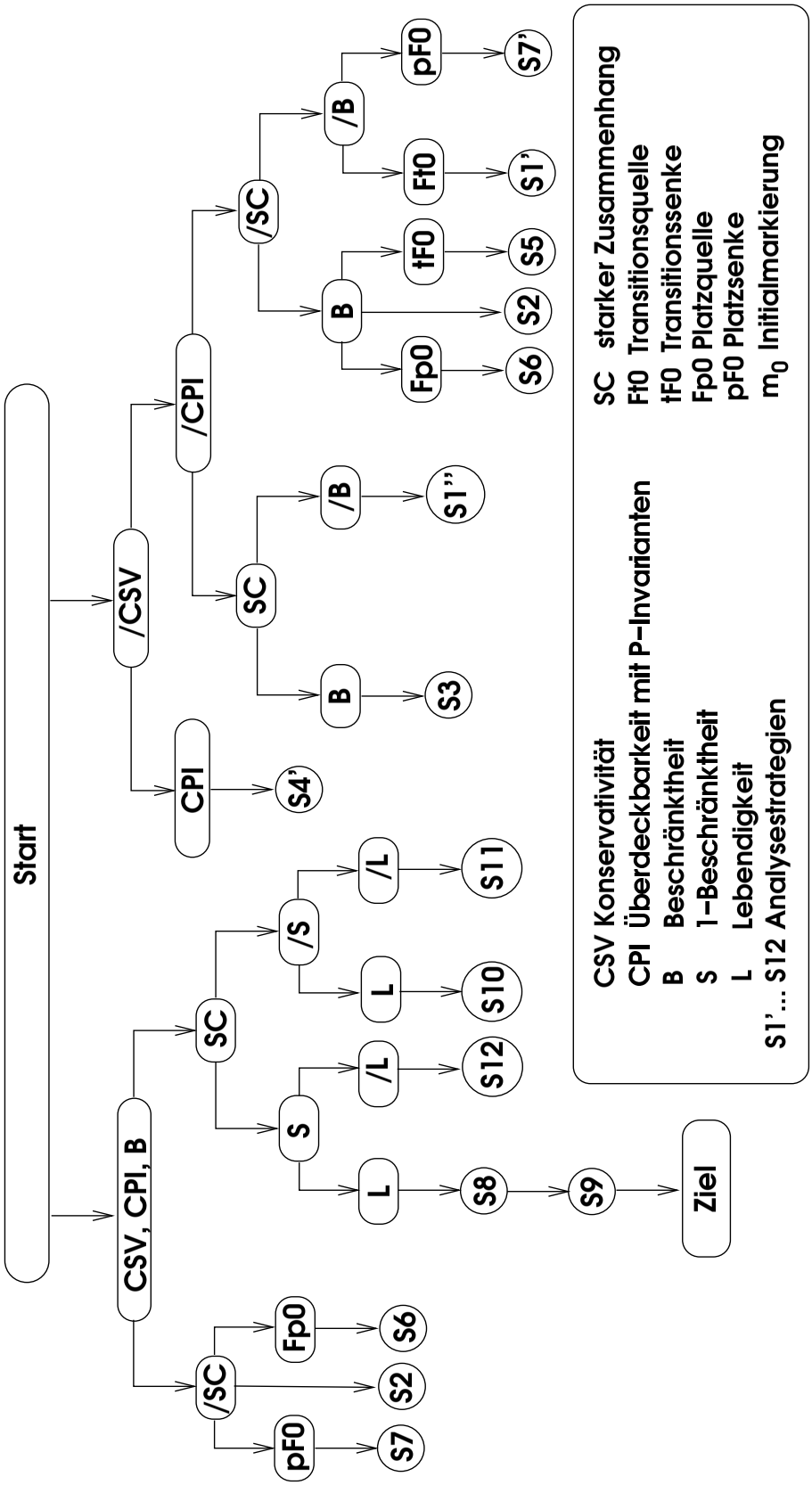

Abbildung 6.7: Verifikationsschema zur funktionalen Verifikation von Petrinetz-Modellen nebenläufiger Kontrollpfade

6.4 Optimierung verifizierter Petrinetz-Modelle

Die Optimierung verifizierter Petrinetz-Modelle hat die Optimierung des Hardware-Aufwandes unter Einhaltung der Systemspezifikation und des Verifikationsergebnisses zum Ziel. Genutzt werden hierbei einerseits Analysemethoden zur Untersuchung von Verhaltenseigenschaften mit erweiterter Erreichbarkeitsanalyse. Weiterhin kommen strukturelle Analysemethoden zum Einsatz, die Struktursymmetrien erkennen und eine strukturelle Reduktionsanalyse ermöglichen.

6.4.1 Analyse von Kreisen und Distanzen für sequentielle Kontrollpfade

Mit der Berechnung von Kreisen innerhalb des Erreichbarkeitsgraphen werden Anzahl und Länge von Befehlszyklen im Petrinetz-Modell abgeleitet. Jeder Kreis entspricht einem Befehlszyklus, beginnend beim Zustand p_{fetch}. Geteilte Plätze generieren neue Kreise und folglich alternative Befehlszyklen. Jeder Kreis beginnt und endet mit einem geteilten Platz. Weil zuvor die gewünschte Zustandsmaschinenstruktur für das gesamte Petrinetz-Modell verifiziert wurde, besitzt auch jeder Befehlszyklus im Petrinetz-Modell die Struktur einer Zustandsmaschine.

Die Zustände innerhalb eines Befehlszyklus werden als Pfad erkennbar. Die Pfadlänge ist gleich der Anzahl der Zustandsänderungen. Im reduzierten Kontrollpfad-Modell des sequentiellen DLX-Prozessors existieren 22 Kreise und damit 22 mögliche Befehlszyklen, die mindestens 2 Schritte, jedoch maximal 7 Schritte zur Abarbeitung benötigen. Das vollständige Petrinetz-Modell des DLX-Kontrollpfades weist 1140 Kreise auf, die wie folgt verteilt sind: Abb. A.1 (Teil a): 56 Kreise; Abb. A.2 (Teil b): 52 Kreise; Abb. A.3 (Teil c): 72 Kreise; Abb. A.4 (Teil d): 960 Kreise. Die Anzahl der Zyklen für Speicherbefehle (Teil d) ist um einen Faktor 15 größer im Vergleich zu anderen Befehlsgruppen.

Grund hierfür ist die Aufspaltung des Zustandes p_{decode} in 12 Alternativen (Teil d) und eine ebenso vielfache Aufspaltung des einzigen Folgezustandes p_{memory}. Es existieren nach p_{decode} und p_{memory} jeweils 12 Prekanten, die eine Verzweigung auf 144 Befehlszyklen hervorrufen. Jedoch, aufgrund der im Petrinetz-Modell angegebenen äußeren Schaltbedingungen, sind nur 12 Befehlszyklen innerhalb der Zustandsübergänge $p_{decode} \rightarrow p_{memory} \rightarrow p_{load/store}$ sinnvoll, und die Anzahl der Befehlszyklen sinkt von 1140 auf 258.

In Abhängigkeit von der gewählten Zielarchitektur erscheint es geeignet, die äußeren Schaltbedingungen zwischen p_{decode} und p_{memory} disjunktiv zu verknüpfen, um unnötige Alternativen und daraus entstehenden Verbindungsaufwand zu vermeiden.

Dadurch werden 12 Transitionen zu einer Transition zusammengefasst. In Abbildung 6.8 ist der betrachtete Modellausschnitt aus Abb A.4 vor und nach der Minimierung der Kreisanzahl dargestellt.

Das Zusammenfassen der 12 Transitionen hat eine Auswirkung auf die Implementierung des Petrinetz-Modells. Während im oberen Beispiel der Abbildung 6.8 jede Schaltbedingung einzeln mit einer Transition verarbeitet wird, so werden im unteren Teil alle Schaltbedingungen vor der Verarbeitung disjunktiv verknüpft. Im ersten Fall werden die 12 Schaltbedingungen mit 12 AND-Gattern einstufig verarbeitet. Durch das disjunktive Verknüpfen der Schaltbedingungen entsteht im unteren Beispiel eine zweistufige OR-AND Logik. So kann bereits bei der Erstellung des Petrinetz-Modells Bezug auf die Zielarchitektur genommen werden, um eine hohe Effizienz bei deren Ausnutzung zu erreichen.

Die Optimierung der Verbindungsstruktur des Hardware-Entwurfs bezüglich geeigneter Anwendungsalgorithmen kann mit einer Simulation dieser Algorithmen durchgeführt werden. Hierfür wird der Algorithmus in Befehlssequenzen und weiter in Befehlszyklen zerlegt. Aus der fortgesetzten Simulation der Befehlszyklen, die, wie in Abschnitt 5.5 beschrieben, visualisierbar ist, kann eine Häufigkeitsverteilung der Befehlssequenzen erstellt werden. Das entstehende Histogramm zeigt häufig aufeinanderfolgende Befehlszyklen, die, im Zustandsspeicher als benachbarte Zustandsmengen implementiert, einen minimierten Verbindungsaufwand ermöglichen.

Mit der Berechnung minimaler und maximaler Distanzen zwischen beliebigen Knoten des Erreichbarkeitsgraphen werden lange Zustandssequenzen erkannt und sind dadurch optimierbar. Der Entwickler kann an dieser Stelle mit dem Petrinetz-Modell die Schaltungskomponenten des Kontrollpfades optimieren und folgend wiederholt die Implementierbarkeit verifizieren.

6.4.2 Reduktionsanalyse

Ein gegenüber der Systemspezifikation redundantes Petrinetz-Modell erfordert einen erhöhten Hardware-Aufwand mit allen Konsequenzen (erhöhter Flächenbedarf, längere Verarbeitungszeiten, erhöhter Leistungsverbrauch und erhöhte Fertigungskosten). Reduktionsalgorithmen dienen dazu, redundante Netzstrukturen zu erkennen und Netzkomponenten zu eliminieren, ohne dabei die Verhaltenseigenschaften des Petrinetzes zu beeinflussen und ohne den Modellinhalt des Petrinetz-Modells zu verändern. Eine Reduktionsregel, deren Anwendung auf ein Petrinetz N weder dessen Lebendigkeit, noch dessen Beschränktheit verändert, heißt konsistent. Daraus folgend werden zwei

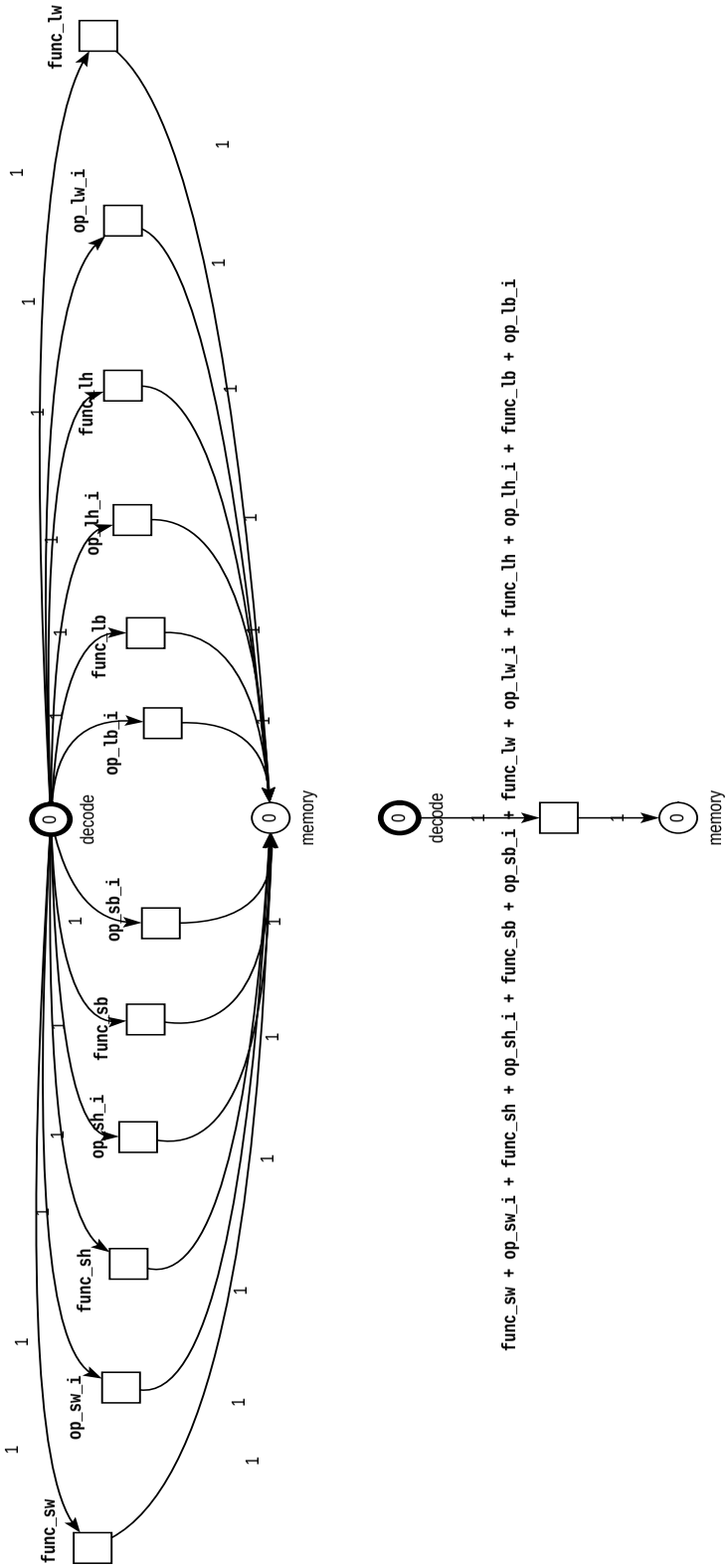

Abbildung 6.8: Zusammenfassung äußerer Schaltbedingungen als Optimierung des Petrinetz-Modells (Ausschnitt Abb. A.4)

Forderungen an eine Reduktionsanalyse gestellt.

Forderung 6.1:

Bei der Netzmodellreduktion sind ausschließlich konsistente Reduktionsregeln anwendbar.

Forderung 6.2:

Es werden keine Netzkomponenten eliminiert, die im digitalen System Zustände oder die Erzeugung und Verarbeitung von Signalen beschreiben. Das Petrinetz-Modell würde bei der Reduktion solcher Netzmodellkomponenten die Systemspezifikation im allgemeinen nicht korrekt wiedergeben.

Reduktion von Netzsymmetrien

Ungeachtet des Modellierungsgegenstandes des Petrinetz-Modells können aus strukturellen Symmetrien eines Modells Symmetrien im Netzverhalten abgeleitet werden. Symmetrien im Verhalten eines Petrinetzes werden äquivalente Markierungen genannt. Das Vorhandensein äquivalenter Markierungen ist ein Hinweis auf eine mögliche Redundanz im Petrinetz-Modell, die als symmetrische Struktur erkennbar und unter bestimmten Bedingungen reduzierbar ist. Netzsymmetrien wurden erstmals für gefärbte Netze eingeführt (HJJJ85). Eine Netzsymmetrie eines Platz/Transitionen-Netzes ist eine eineindeutige Abbildung zwischen zwei Knotenmengen, bei der die Knotenart, die Kanten und das Kantengewicht erhalten bleiben.

Definition 6.1:

Die eineindeutige Abbildung ω auf $P \cup T$ wird Symmetrie des Netzes N genannt, wenn

(i) $p \in P \Leftrightarrow \omega(p) \in P,\ \forall p \in P$ (Erhaltung der Knotenart)

(ii) $t \in T \Leftrightarrow \omega(t) \in T,\ \forall t \in T$ (Erhaltung der Knotenart)

(iii) $[p,t] \in F \Leftrightarrow [\omega(p),\omega(t)] \in F,\ \forall p,t \in P \cup T$ (Erhaltung der Kantenart)

(iv) $[t,p] \in F \Leftrightarrow [\omega(t),\omega(p)] \in F,\ \forall p,t \in P \cup T$ (Erhaltung der Kantenart)

(v) $V(\omega(f)) = V(f)\ \forall f \in F.$ (Erhaltung des Kantengewichtes)

Hier werden ausschließlich gewöhnliche P/T-Netze verwandt, weshalb Bedingung (v) immer gegeben ist ($V(f) = 1\ \forall f \in F$). Mit dem Begriff der Netzsymmetrie können

äquivalente Markierungen und somit Verhaltenssymmetrien eines Platz/Transitionen-Netzes beschrieben werden.

Definition 6.2:

(i) Die Markierungen m, m' heißen *äquivalent*, $m \approx m'$, wenn es eine Netzsymmetrie ω gibt mit $\omega[m] = m'$.

(ii) Die Transitionen t, t' heißen *äquivalent*, $t \approx t'$, wenn es eine Netzsymmetrie ω gibt mit $\omega[t] = t'$.

Bevor eine Symmetrieanalyse erfolgt, sollen folgende Eigenschaften des Petrinetzes sichergestellt werden. Das Petrinetz ist 1-beschränkt und lebendig und weist im Fall eines sequentiellen Kontrollpfadmodells eine stark-zusammenhängende Zustandsmaschinenstruktur auf. Abbildung 6.9 illustriert beispielhaft zwei Teilnetze des Petrinetz-Modells aus Abbildung 5.10.

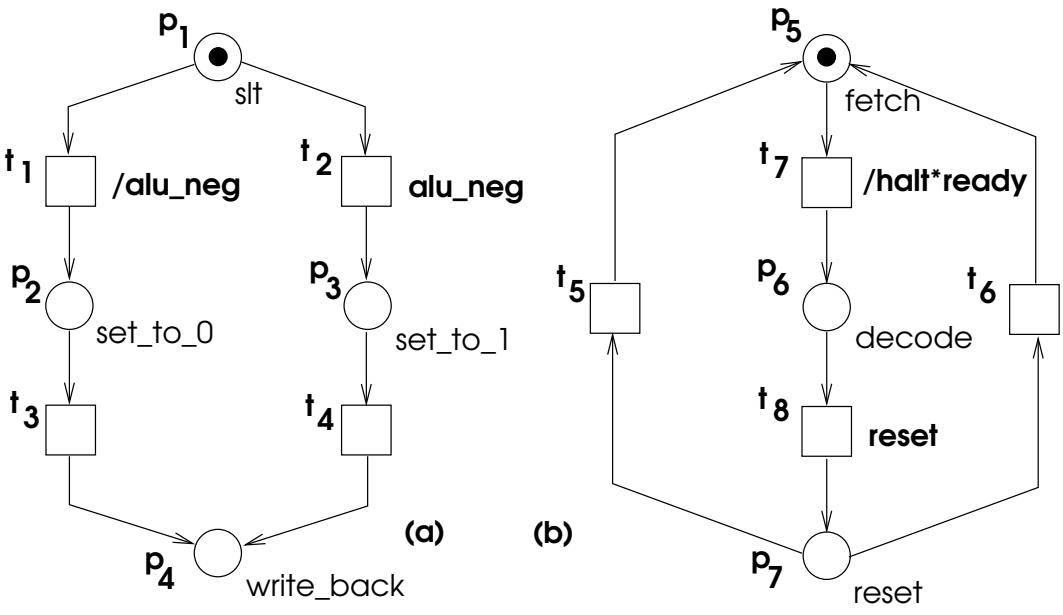

Abbildung 6.9: Netzsymmetrien im Petrinetz-Modell

Im Fall (a) handelt es sich um ein Teilnetz, das den Ablauf des Testbefehls SLT mo-

delliert. Die Markierungen $m(p_2) = 1$ und $m(p_3) = 1$ ($m(p_j) = 0,\ \forall p_j \neq p_2, p_3$) sowie die Transitionenpaare (t_1, t_2) und (t_3, t_4) sind äquivalent. Dennoch ist weder eine Transition der Transitionenpaare (t_1, t_2) und (t_3, t_4), noch einer der beiden Plätze (p_2, p_3) reduzierbar. Eine solche Reduktion würde die Systemspezifikation nicht korrekt wiedergeben.

Fall (b) zeigt ein modifiziertes Teilnetz zur Modellierung des Rücksetzzustandes RESET. Es existiert eine Netzsymmetrie ω mit $\omega(t_5) = t_6$. Die Transitionen (t_5, t_6) sind somit äquivalent. Sie realisieren beide den Übergang vom Rücksetzzustand in den Zustand FETCH. Mit der Reduktion einer der beiden Transitionen t_5 oder t_6 kann diese Redundanz vermieden werden. Die Funktionalität der Systemspezifikation wird dabei nicht beeinträchtigt und der Zustandsübergang RESET $\rightarrow$ FETCH ist eindeutig. Ein effizientes Auffinden und Beseitigen dieser Redundanz ist wie folgt realisierbar.

Definition 6.3:

Zwei Transitionen t_1 und t_2 heißen parallel im gewöhnlichen Petrinetz N, wenn sie denselben Vor- und Nachbereich besitzen: $(t_1\bullet = t_2\bullet) \wedge (\bullet t_1 = \bullet t_2)$.

Folgerung 6.1:

Parallele Transitionen können fusioniert werden.

Die beiden Transitionen t_5 und t_6 aus Fall (b), Abbildung 6.9, sind mit allen anderen Knoten des Netzes auf die gleiche Art und Weise verbunden: $t_5\bullet = t_6\bullet = p_5$, $\bullet t_5 = \bullet t_6 = p_7$. Es macht demnach keinen Unterschied, ob Transition t_5 oder t_6 schaltet, um den Zustandsübergang RESET $\rightarrow$ FETCH zu realisieren. Deshalb kann die Reduktionsregel der Fusion paralleler Knoten auf die Transitionen t_5 und t_6 angewandt werden. Die Reduktionsregel der Fusionierung paralleler Knoten ist konsistent (Sta90). Die beiden Transitionen t_5 und t_6 werden durch eine Transition t_6' mit $\bullet t_6' = p_7$ und $t_6'\bullet = p_5$ ersetzt.

Netzmodellreduktion

Eine weitere Anwendung der Netzreduktionsregeln kann zur gezielten Veränderung der Systemspezifikation führen. Entspricht das Petrinetz-Modell der Systemspezifikation, und der Entwickler stellt mit dem Petrinetz-Modell eine unerwünschte Redundanz der Systemspezifikation fest, so können auch Netzknoten eliminiert werden, die eine Verletzung der Forderung 6.2 verursachen. Werden bei dieser Netzreduktion nur konsistente Reduktionsregeln angewandt, so sind die dynamischen Eigenschaften des Petrinetzes

invariant. Andererseits kann ein wiederholtes Anwenden eines Satzes konsistenter Reduktionsregeln dazu führen, dass das ursprüngliche Petrinetz-Modell zu einem Petrinetz-Modell bestehend aus einem Platz p und einer Transition t mit $t\bullet = \bullet t = p$ reduziert wird. Dieses Petrinetz-Modell ist sicher und lebendig, gibt aber den ursprünglichen Modellinhalt nicht wieder.

Aus diesem Grund benötigt der Entwickler für die manuelle Reduktion der Netzknoten genaue Kenntnis der Auswirkung einer solchen Netzreduktion auf das Petrinetz-Modell und folgend auf die Äquivalenz zur Systemspezifikation.

6.5 Eine Anwendung - Verifikation des Petrinetz-Modells eines Mikroprozessors

Die vorgeschlagene Verifikationsstrategie wird nun am Beispiel des in Abschnitt 5.4 modellierten Mikroprozessors angewandt. Dabei werden beide Varianten des DLX-Prozessors entsprechend der jeweils erstellten Analysestrategien und der daraus gewonnenen Verifikationsschemata vorgestellt.

6.5.1 Verifikation der Petrinetz-Modelle des DLX-Prozessors

Mit der Anwendung der in den Abschnitten 6.2 und erstellten 6.3 Analysestrategien (Tabellen 6.2 und 6.3) konnten Modellierungsfehler in den Petrinetz-Modellen (Anhang A und B) erkannt, lokalisiert und beseitigt werden. Die in der Struktur- und Verhaltensanalyse untersuchten Eigenschaften wurden für die beiden Kontrollpfadmodelle des DLX-Prozessors nachgewiesen und entsprechend Tabelle 6.1 interpretiert. Diese Interpretation ermöglicht eine Aussage zur formalen Verifikation der Petrinetz-Modelle.

Verifikation des sequentiellen Modells

Die Zustandsmaschinenstruktur wurde für den untersuchten Kontrollpfad nachgewiesen. Das Petrinetz-Modell des DLX-Kontrollpfades besitzt die Struktur stark zusammenhängender Zustandsmaschinen. Die Anzahl der Zustände des DLX-Kontrollpfades ist endlich. Der Erreichbarkeitsgraph des vollständigen Kontrollpfad-Modells umfasst 64 Zustände. Jeder Zustand des Erreichbarkeitsgraphen kann einem Zustand des DLX-Kontrollpfades eindeutig zugeordnet werden. Das Kontrollpfad-Modell des DLX-Prozessors ist in jedem Zustand fortschaltfähig. Für das Kontrollpfad-Modell des DLX-Prozessors kann aus einem beliebigen Zustand p_j ein Übergang zu $m_0 = m(p_{fetch}) = 1$ erfolgen.

Es wurde verifiziert, dass eine Modellierung äußerer Signale mit Schleifen im Petrinetz-Modell nicht realisiert ist. Statische und dynamische Konflikte konnten mit äußeren Schaltbedingungen aufgelöst werden. Somit existiert für den untersuchten Kontrollpfad kein Indeterminismus. Aus den Ergebnissen konnte das Petrinetz-Modell des Kontrollpfades des sequentiellen DLX-Prozessors formal verifiziert werden. Eine weitere Analyse des Erreichbarkeitsgraphen ermöglichte die Optimierung des Petrinetz-Modells.

Verifikation des nebenläufigen Modells

Für den nebenläufigen Kontrollpfad konnte mit einer Strukturanalyse Konservativität und mit der Analyse von P-Invarianten ein endlicher Zustandsraum nachgewiesen werden. Als weitere Struktureigenschaft wurde der starke Zusammenhang untersucht und nachgewiesen. Aus dem danach konstruierten Erreichbarkeitsgraphen konnten mit der Analyse von Verhaltenseigenschaften 1-Beschränktheit, Lebendigkeit und Reversibilität sichergestellt werden. Mit dem Petrinetz-Modell kann die Signalverarbeitung digitaler Systeme äquivalent in einem formalen Modell dargestellt werden. Das modellierte digitale System ist in jeder Situation fortschaltfähig und rücksetzbar. Letztlich wurde das Petrinetz-Modell hinsichtlich vorhandener Konflikte und Schleifen untersucht, wobei alle Konflikte mit äußeren Schaltbedingungen aufgelöst wurden und sinnvoll modellierte Schleifen verifiziert werden konnten. Für das modellierte digitale System existieren ausschließlich deterministische Zustandsübergänge. Die Ergebnisse der Analyse relevanter Petrinetz-Eigenschaften ermöglichte die formale Verifikation des Petrinetz-Modells des Kontrollpfades des nebenläufigen DLX-Prozessors.

Kapitel 7

VeriCon - ein konfigurierbares Werkzeug zur Verifikation digitaler Kontrollpfade

7.1 Anforderungen an das Petrinetz-Werkzeug

Das Werkzeug VeriCon (Verification of Control Paths) ist eine prototypische Umsetzung der Hardware-Entwurfsmethodik PNDes zur Modellierung und Verifikation von Petrinetz-Modellen. Das Petrinetz-Werkzeug leistet eine automatisierte Verifikation digitaler Systeme auf funktionaler Ebene. Darüber hinaus wird mit VeriCon der Modellierungsprozess, die funktionale Simulation und Validierung der Petrinetz-Modelle sowie die Optimierung verifizierter Petrinetz-Modelle realisiert. Bei der Realisierung von PNDes als Petrinetz-Werkzeug müssen aufgabenspezifische und realisierungsbezogene Anforderungen beachtet werden.

Die wichtigste aufgabenspezifische Anforderung besteht in der Umsetzung aller Anwendungen von Petrinetz-Modellen, entsprechend Abschnitt 2.2, und deren Verknüpfung mit dem Modellierungsprozess. Für die Simulation und Validierung von Petrinetz-Modellen werden die grafischen Konzepte der Petrinetze genutzt. Beide Aufgaben können mit dem Modellierungsprozess gekoppelt werden, wenn ein Petrinetz-Editor existiert, der mit verschiedenen Simulationsanwendungen erweitert werden kann. Hierbei sind schrittweise und automatische Simulation ebenso von Bedeutung, wie die Simulation und Validierung interpretierter Petrinetze. Die letzte Forderung zielt auf die Fähigkeit, mit dem Petrinetz-Werkzeug eigene Netzspezifikationen zu definieren, um auf diesem Weg Petrinetz-Modelle mit interpretierten Petrinetzen zu erstellen und zu validieren.

Für die Analyse und Verifikation von Petrinetz-Modellen ist es sinnvoll, ein oder mehrere bereits bestehende Petrinetz-Analysewerkzeuge vollständig oder teilweise zu nut-

zen und mit der Modellierungsumgebung zu koppeln. Voraussetzung hierfür ist, dass mindestens eine der beiden Seiten, Modellierungs- oder Analyseumgebung, erweiterbar sein muss. Es muss außerdem ein manipulierbarer Austausch zwischen Modellierungs- und Analysewerkzeug geschaffen werden.

In Tabelle 7.1 sind die zu lösenden Aufgaben der entsprechenden Entwurfsphase zugeordnet. In der dritten Spalte ist angegeben, für welche der Aufgaben der Petrinetz-Kern und der Integrierte Netz-Analysator genutzt werden. Aus den erstellten Anwendungen des Petrinetz-Kerns und der selektiven Einbeziehung des Analysewerkzeuges INA (SR01) entsteht das Petrinetz-Werkzeug VeriCon.

Aufgabe	Entwurfsphase	Umsetzungsmittel
Modellierung	Modellierung	Petrinetz-Kern
Simulation	Modellierung/ Funktionale Analyse	Petrinetz-Kern
Validierung	Modellierung/ Funktionale Analyse	Petrinetz-Kern
Analyse	Funktionale Analyse	Integrierter Netz-Analysator
Verifikation	Funktionale Analyse	Petrinetz-Kern

Tabelle 7.1: Realisierung der Entwurfsphasen Modellierung und Funktionale Analyse

Die realisierungsbezogenen Anforderungen basieren auf dem Anspruch der Entwurfsautomatisierung. Sie entstehen hauptsächlich aus der Forderung nach Erweiterbarkeit und Kopplung einzelner Komponenten und Anwendungen, um so die petrinetzbasierten Entwurfsschritte miteinander zu verbinden. Diese Anforderungen lauten im Einzelnen:

(i) schnelle Implementierbarkeit von Analysealgorithmen

(ii) einfache Erweiterbarkeit und Adaption an andere Werkzeuge

(iii) erweiterbare grafische Anbindung

(iv) gut lesbares und konvertierbares Dateiaustauschformat

(v) freie Verfügbarkeit der Quellen

(vi) kostenfreie Verfügbarkeit

(vii) weitere Entwicklung

Der Petrinetz-Kern wird als Realisierungsgrundlage für das Werkzeug VeriCon verwendet. Er bietet eine Infrastruktur zur Erstellung von Petrinetz-Werkzeugen mit größtmöglicher Entwurfsflexibilität und erfüllt alle realisierungstechnischen Anforderungen. Die Entwicklungsumgebung PNK ist in der objektorientierten Programmiersprache Python implementiert. Netzkomponenten (Platz, Transition, Kante) und Netzspezifikatio-

nen (Transitionsaktivierung, Schaltregel, Platzmarkierung, diverse Erweiterungen) werden in den Klassenstrukturen strikt getrennt, was eine Erweiterung sehr einfach macht. Da Python eine interpretierte Programmiersprache ist, wird eine schnelle Implementierung von Anwendungen möglich.

7.2 Konzeption und Umsetzung

Abbildung 7.1 illustriert die bei der Konzeption und Erstellung von VeriCon verwendeten Software-Komponenten.

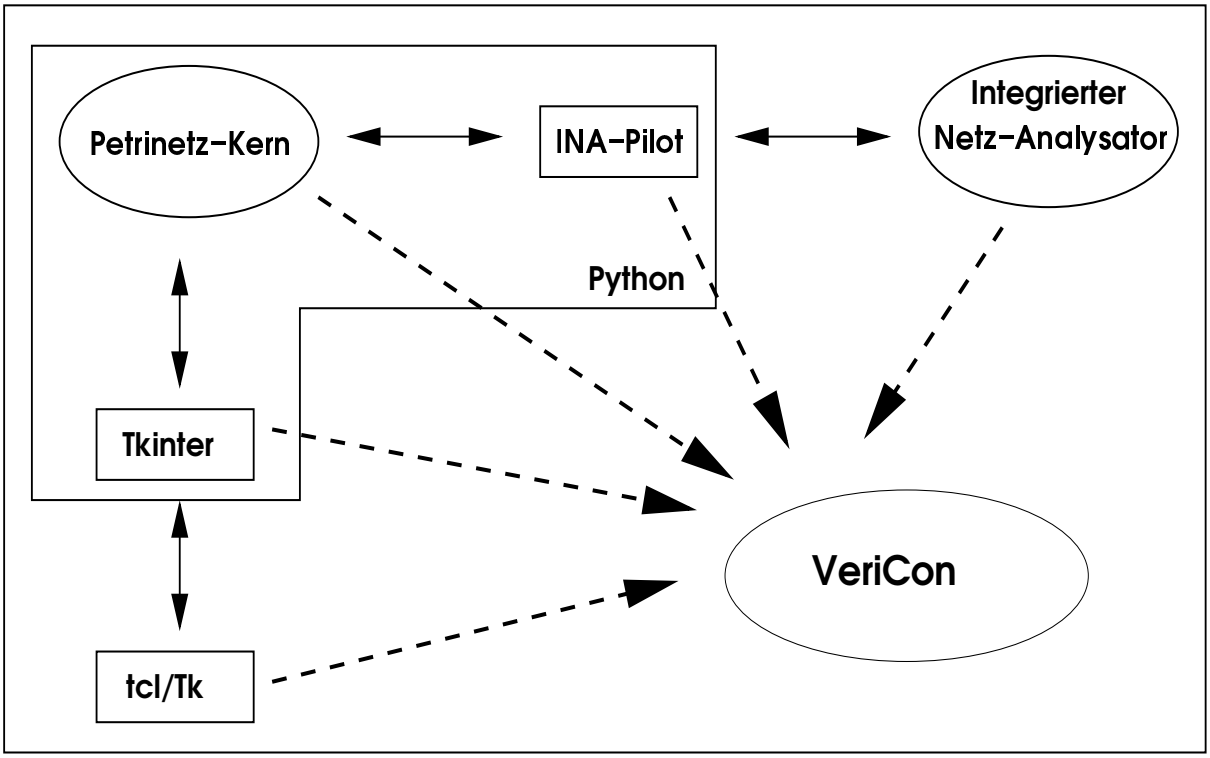

Abbildung 7.1: Komponenten für die Erstellung des Petrinetz-Werkzeuges VeriCon

Der Petrinetz-Kern besitzt einen erweiterbaren grafischen Editor. Dieser Editor kann in einfacher Weise über das Python-Grafikmodul Tkinter mit Simulations- und Analyseanwendungen gekoppelt werden. Darüber hinaus bietet der Petrinetz-Kern die Möglichkeit, eigene Netzspezifikationen zu definieren. Diese können von bestehenden Spezifikationen abgeleitet werden. Somit ist es möglich, Petrinetz-Modelle mit steuerungstechnischer Netzinterpretation mit einer selbstdefinierten Netzspezifikation, welche den SIPN entspricht, zu modellieren. Die Kopplung von Editor und Simulationsanwendungen

erlaubt dann außerdem, Simulationsanwendungen speziell für die selbstdefinierte Netzspezifikation zu erstellen.

Die Analyse von Petrinetz-Modellen basiert auf der Nutzung des Petrinetz-Kerns und des Integrierten Netz-Analysators. Aus der Modellierungsumgebung heraus wird das Petrinetz-Modell vorher festgelegten Analysemethoden unterworfen. Diese werden vom Integrierten Netz-Analysator geleistet. Die Analyseergebnisse werden zur Modellierungsumgebung rückgeführt und am Petrinetz-Modell visualisiert.

Ausgangspunkt für die automatisierte Analyse von Petrinetz-Modellen ist ein Ansatz zur Kopplung des Petrinetz-Kerns mit dem Integrierten Netz-Analysator, der INA-Pilot (KO98, Osc99). Der INA-Pilot kann derart erweitert werden, dass gezielt Analysealgorithmen in gewünschter Reihenfolge auf das unbeschriftete Petrinetz angewandt werden und geeignet für das Petrinetz-Modell auswertbar sind. Die Auswertung der Analyseergebnisse erlaubt Schlussfolgerungen zur Verifikation des Petrinetz-Modells. Auf diesem Weg werden die in Abschnitt 6.2 erstellten Analysestrategien als Verifikationsschema für sequentielle Kontrollpfade implementiert (Wag02). Mit wenigen Modifikationen wird VeriCon für die Verifikation nebenläufiger Kontrollpfade angepasst.

VeriCon ist ein konfigurierbares Petrinetz-Werkzeug. Die verschiedenen Analysestrategien können leicht zu neuen Verifikationsschemata formiert werden und die Verifikation ist somit in sehr kurzer Zeit an neue Aufgabenstellungen anpassbar.

7.3 Test und Erweiterung

VeriCon wurde mit den Petrinetz-Modellen des sequentiellen und des nebenläufigen DLX-Prozessors getestet. Die verschiedenen Aufgaben aus Tabelle 7.1 lassen sich mit VeriCon hervorragend in einem Werkzeug vereinen. Alle Analysestrategien für das sequentielle Kontrollpfadmodell konnten problemlos umgesetzt werden. Die Anordnung einzelner Strategien ist variabel, so dass das Verifikationsschema in sehr kurzer Zeit verändert und an variierende Aufgaben angepasst werden konnte.

Im sequentiellen Fall ist der Rechenaufwand für die Auswertung des Erreichbarkeitsgraphen wegen des relativ kleinen Zustandsraumes $O(10^2)$ gering. Deshalb wurde die Erreichbarkeitsanalyse bei der Analyse sequentieller Kontrollpfadmodelle in den Vordergrund gestellt. Für nebenläufige Kontrollpfadmodelle ist der Rechenaufwand aufgrund eines Zustandsraumes mit $O(10^5)$ Zuständen erheblich größer, bleibt aber mit Analysezeiten zwischen 3-5 Minuten in einem praktikablen Bereich. Um möglichst schnell zu Schlussfolgerungen zu gelangen, wurde bei der Modellanalyse zuerst die Untersuchung

von Struktureigenschaften und die Invariantenanalyse durchgeführt.

Die schnelle Konfigurierbarkeit des Petrinetz-Werkzeuges zeigt sich nicht nur in der Variabilität der Anordnung der Analysestrategien. Die Verifikation des nebenläufigen Kontrollpfadmodells konnte mit wenigen Modifikationen der Analysestrategien des sequentiellen Kontrollpfadmodells ($\mathbf{S1'}, \mathbf{S1''}, \mathbf{S4'}$ und $\mathbf{S7'}$) und mit drei neuen Strategien ($\mathbf{S10}, \mathbf{S11}$ und $\mathbf{S12}$) umgesetzt werden.

Erweiterungsmöglichkeiten bestehen in der Erstellung von Ein- und Ausgabeschnittstellen, die eine direkte HDL-Eingabe und die Ausgabe eines verifizierten HDL-Modells, entsprechend PNDes ermöglichen (Abblildung 4.1). Damit wird die in Abschnitt 5.2 vorgeschlagene Transformation zwischen Petrinetz-Strukturen und HDL-Konstrukten entsprechend des Anspruchs der Entwurfsautomatisierung optimiert.

Darüber hinaus besteht die Möglichkeit, eine weitere Verifikationsmethodik in VeriCon zu einzubringen. Nach erfolgter Technologieabbildung kann im Rahmen eines konventionellen Hardware-Entwurfs eine zeitbehaftete Simulation durchgeführt werden. Dabei wird überprüft, ob der funktional korrekte Entwurf auch unter den zeitlichen Zwangsbedingungen der Zielarchitektur funktionale Korrektheit aufweist. Die Überprüfung kann, wie in Abschnitt 4.1 angesprochen, petrinetz-basiert mit zeitbehafteten Netzspezifikationen realisiert werden. Darüber hinaus ist es möglich, eine petrinetz-basierte Leistungsbewertung des Entwurfs mit deterministisch und stochastisch zeitbehafteten Petrinetzen zu realisieren.

Kapitel 8

Zusammenfassung und Ausblick

Das Ziel dieser Arbeit ist die formale Verifikation digitaler Systeme auf funktionaler Ebene. Dabei werden Petrinetze als Beschreibungsmittel für digitale Systeme und als Werkzeug für deren Simulation, Validierung, Analyse und Verifikation genutzt. Die formale Verifikation eines Petrinetz-Modells wird mit der Methode der Eigenschaftsprüfung realisiert.

Im ersten Teil der Arbeit wird eine Hardware-Entwurfsmethodik vorgeschlagen. Die Hardware-Entwurfsmethodik PNDes ermöglicht den petrinetz-basierten Entwurf digitaler Systeme. Durch die ereignisbasierte Modellierung mit Petrinetzen können digitale Systeme ungeachtet der Taktung des Systems mit Petrinetzen beschrieben und auf funktionaler Ebene vollständig verifiziert werden. PNDes ist in einen konventionellen Hardware-Entwurfsfluss eingebettet. Damit wird sichergestellt, dass herkömmliche Hardware-Beschreibungssprachen und Synthesewerkzeuge zum Einsatz kommen können. Im Fokus von PNDes stehen die Entwurfsabschnitte Modellierung und Verifikation.

Der Entwurfsabschnitt Modellierung wird derart untersetzt, dass mit der Netzspezifikation der Free-Choice-Netze und der Netzinterpretation der steuerungstechnisch interpretierten Petrinetze eine transparente Abbildung eines digitalen Systems in ein Petrinetz-Modell realisierbar ist. Die Transparenz der Abbildung spiegelt sich in der Äquivalenz zwischen der Dynamik des digitalen Systems und der Dynamik des Petrinetz-Modells wider.

Das Petrinetz-Modell wird im folgenden Entwurfsschritt geeignet simuliert, validiert, analysiert und verifiziert. Dazu wurde eine Verifikationsmethodik ausgearbeitet, mit welcher Petrinetz-Modelle mit Netzerweiterungen (Schaltbedingungen) analysierbar und verifizierbar sind. Ziel der formalen Verifikation ist die Ableitung einer Aussage bezüglich der vollständigen Korrektheit des Petrinetz-Modells auf funktionaler Ebene. Die Analyse des Petrinetzes prüft Petrinetz-Eigenschaften, welche für das Petrinetz-Modell geeignet

interpretiert werden. Die Verifikationsstrategie legt fest, welche Eigenschaften in welcher Reihenfolge untersucht werden und wie die Ergebnisse der Petrinetz-Analyse für das Petrinetz-Modell interpretiert werden. Folgend werden Analysestrategien erstellt, mit denen alle Modellierungsfehler, die geforderte Petrinetz-Eigenschaften verletzen, detektiert und lokalisiert werden. Die Analysestrategien werden zu einem Verifikationsschema zusammengesetzt, das die Grundlage für eine automatisierte Verifikation bildet. Aus der Anwendung des Verifikationsschemas wird eine Anzahl von Modellierungsrichtlinien abgeleitet, die den Modellierungsprozess bereits frühzeitig beeinflussen, um Entwurfsfehler zu vermeiden. Die Nutzung der Verifikationsergebnisse für den Modellierungsprozess kann den Entwurfszyklus effizienter gestalten.

Verifizierte Petrinetz-Modelle werden mit Methoden der Reduktionsanalyse und der erweiterten Erreichbarkeitsanalyse optimiert. Diese Optimierung erfolgt im Hinblick auf die Zielarchitektur. Die im Rahmen des Verifikationsprozesses erlangte Aussage zur Korrektheit des Petrinetz-Modells wird durch die nachfolgende Optimierung des Petrinetz-Modells nicht beeinflusst.

Beispielhaft wurde die vorgeschlagene Hardware-Entwurfsmethodik auf ein sequentielles und ein nebenläufiges Modell eines Mikroprozessors angewandt. Synthetisierbare VHDL-Modelle des Mikroprozessors dienten dabei als Referenzmodell.

Die Hardware-Entwurfsmethodik wurde prototypisch als Petrinetz-Werkzeug realisiert. Das dient dem Nachweis der Anwendbarkeit von PNDes und stellt einen ersten Schritt zur Automatisierung eines Entwurfes dar. Das Werkzeug VeriCon ist in einfacher Weise konfigurierbar und kann in kurzer Zeit an unterschiedliche Verifikationsaufgaben angepasst werden. VeriCon wurde mit den Petrinetz-Modellen der verschiedenen Kontrollpfade eines Mikroprozessors getestet.

Zukünftige Arbeiten können die Hardware-Entwurfsmethodik PNDes um eine weitere Modellierungs- und Verifikationsmethodik erweitern. Hierbei wird ein zeitbehaftetes Petrinetz-Modell eines digitalen Systems der vollständigen Verifikation und Leistungsbewertung unterworfen. Diese Methodik wird mit der Anwendung deterministisch zeitbehafteter und stochastisch zeitbehafteter Netzspezifikationen umgesetzt. Die Zeitbehaftung der einzelnen Netzkonstituenten muss dabei aus dem synthetisierten digitalen System gewonnen werden, damit das daraus entstehende Petrinetz-Modell eine transparente Abbildung des synthetisierten Hardware-Entwurfs darstellt.

Anhang A

Petrinetz-Modell I

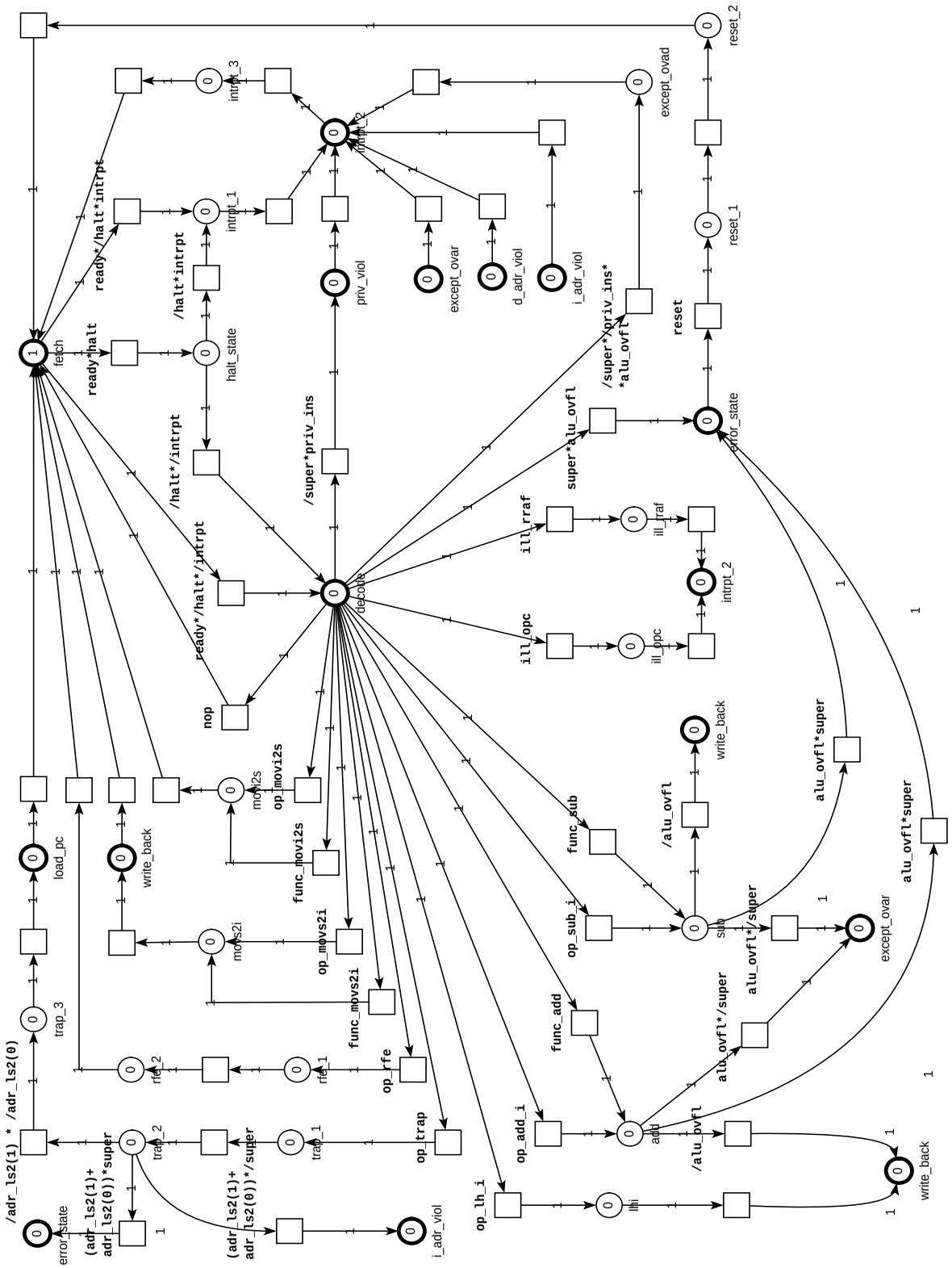

Abbildung A.1: Petrinetz-Modell des sequentiellen DLX-Prozessors (Teil a)

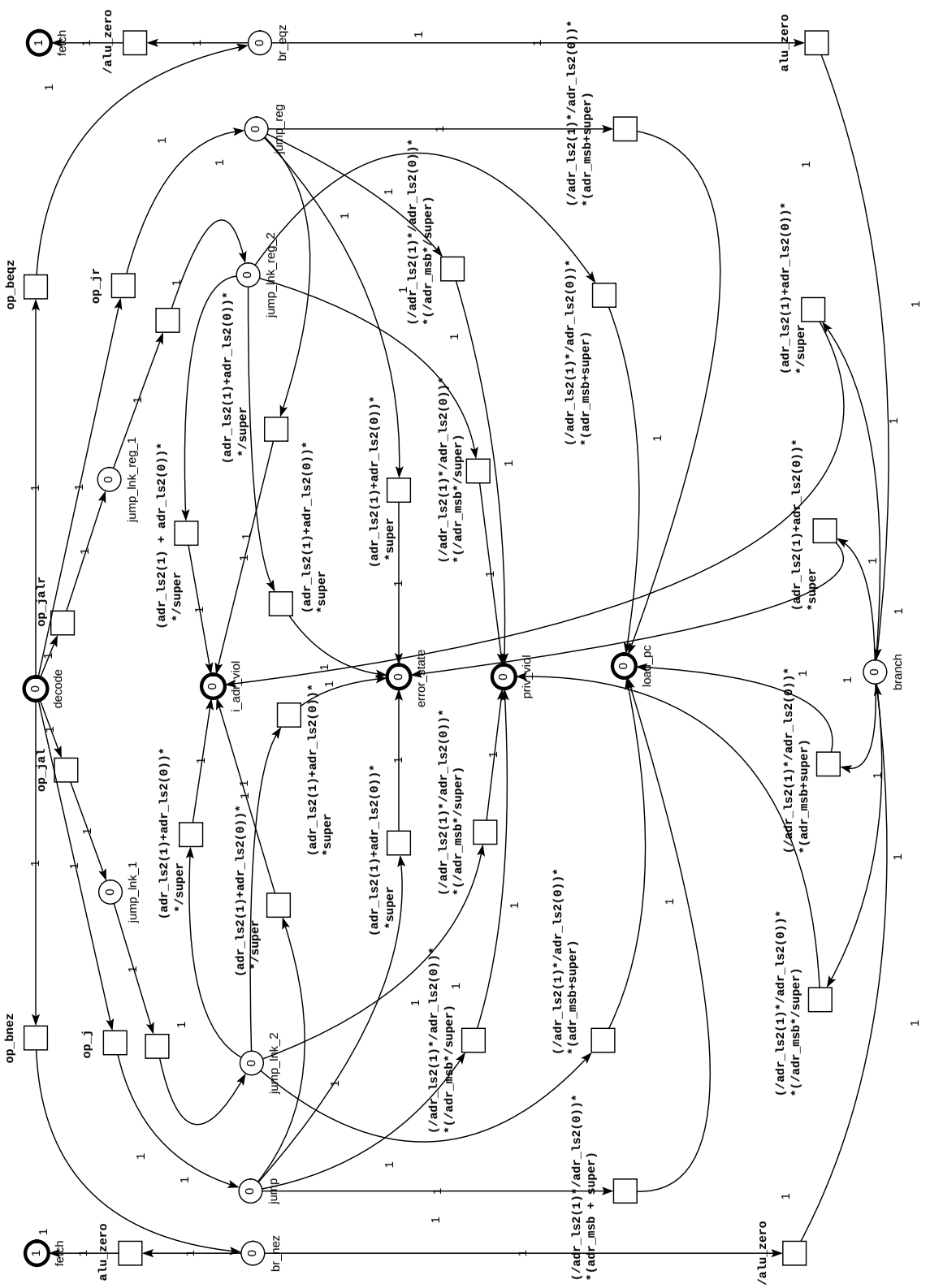

Abbildung A.2: Petrinetz-Modell des sequentiellen DLX-Prozessors (Teil b)

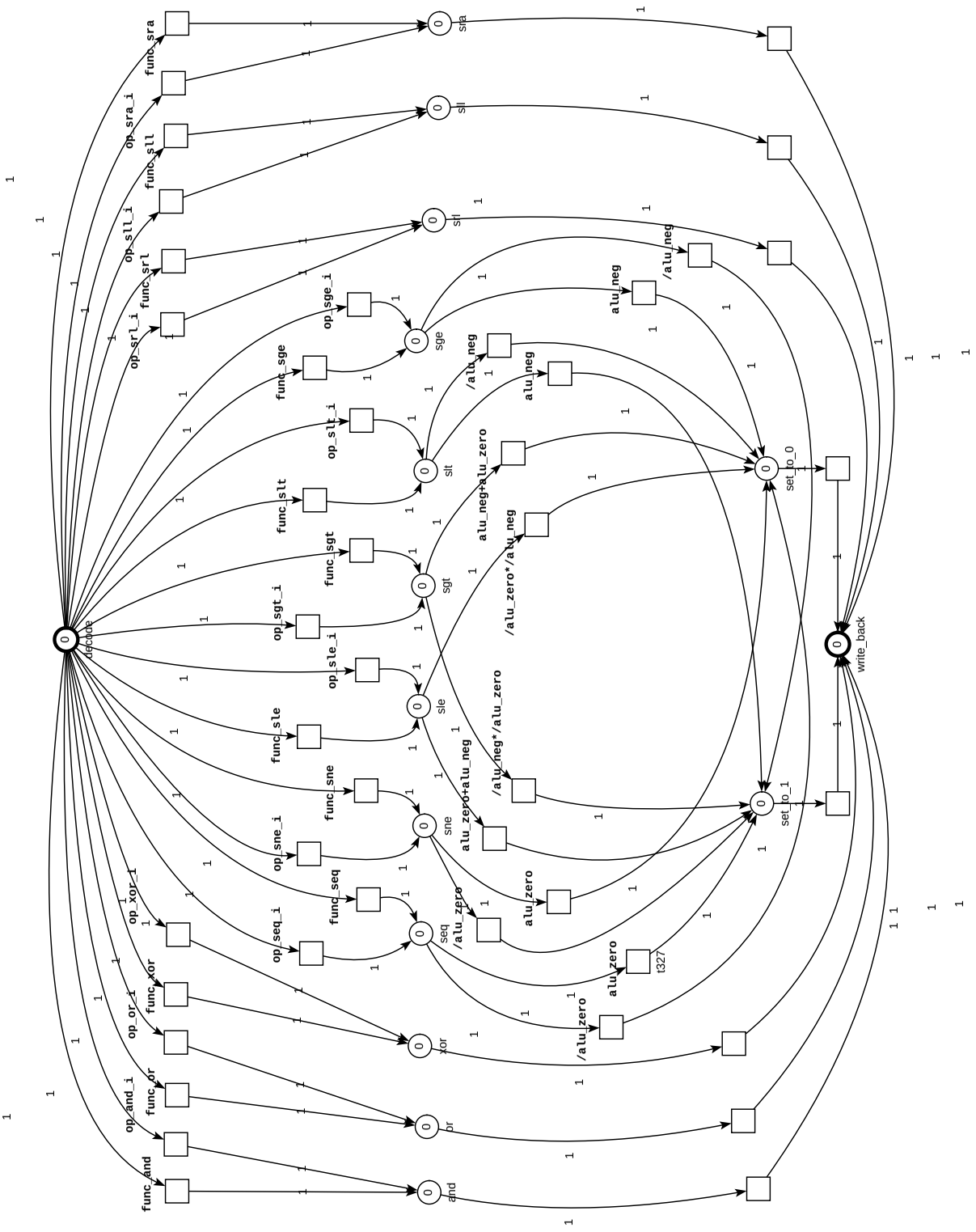

Abbildung A.3: Petrinetz-Modell des sequentiellen DLX-Prozessors (Teil c)

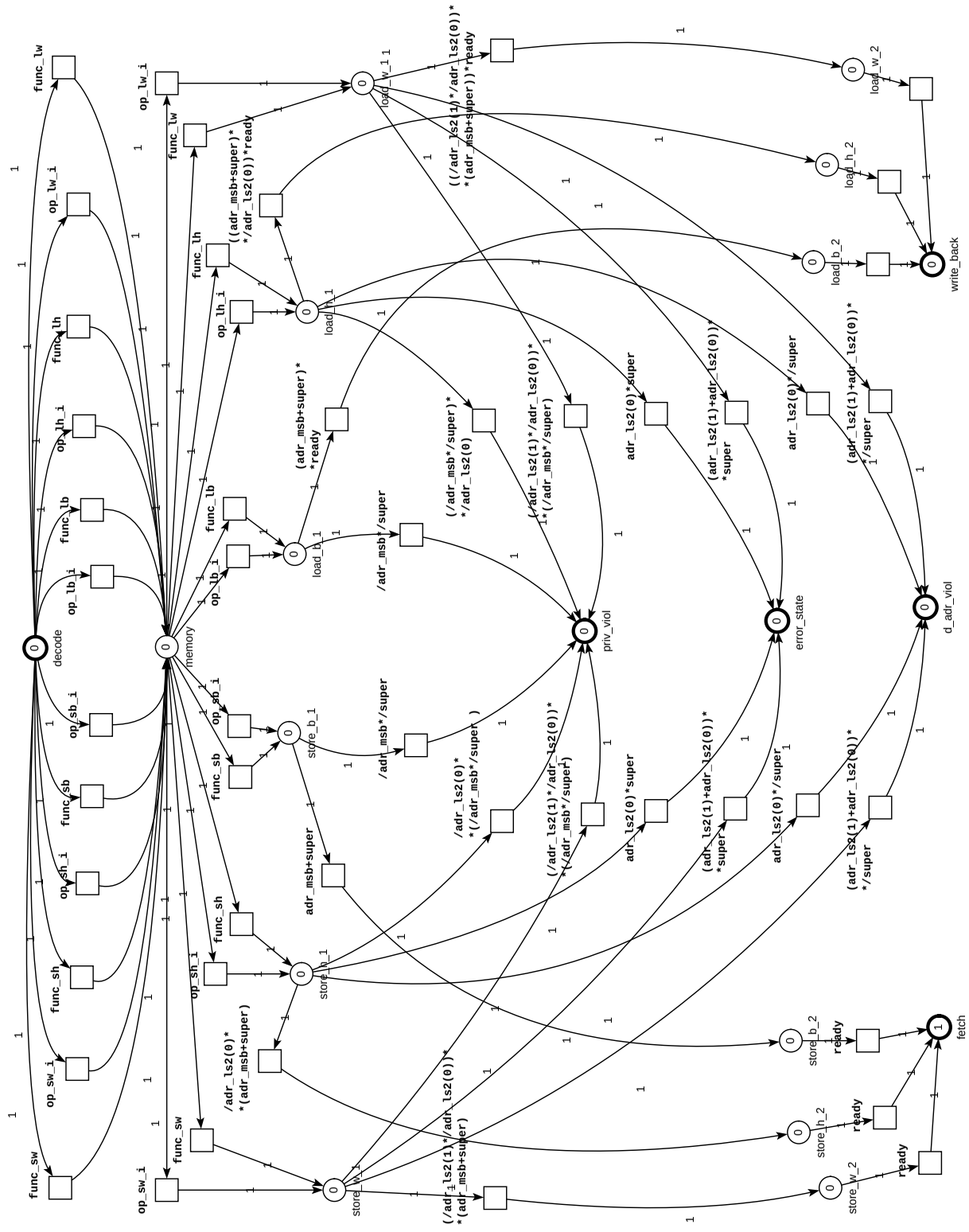

Abbildung A.4: Petrinetz-Modell des sequentiellen DLX-Prozessors (Teil d)

Anhang B

Petrinetz-Modell II

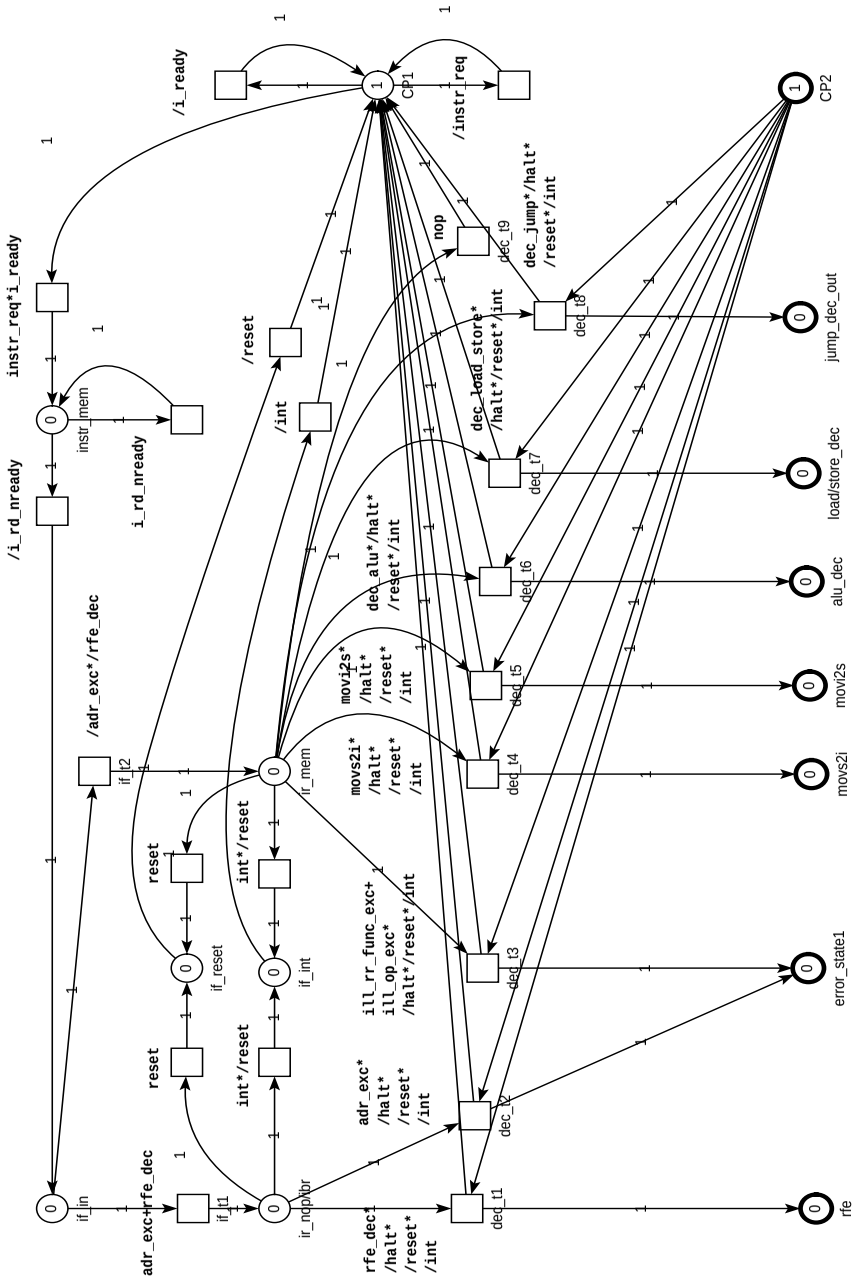

Abbildung B.1: Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil a - Pipeline-Struktur IF/DEC)

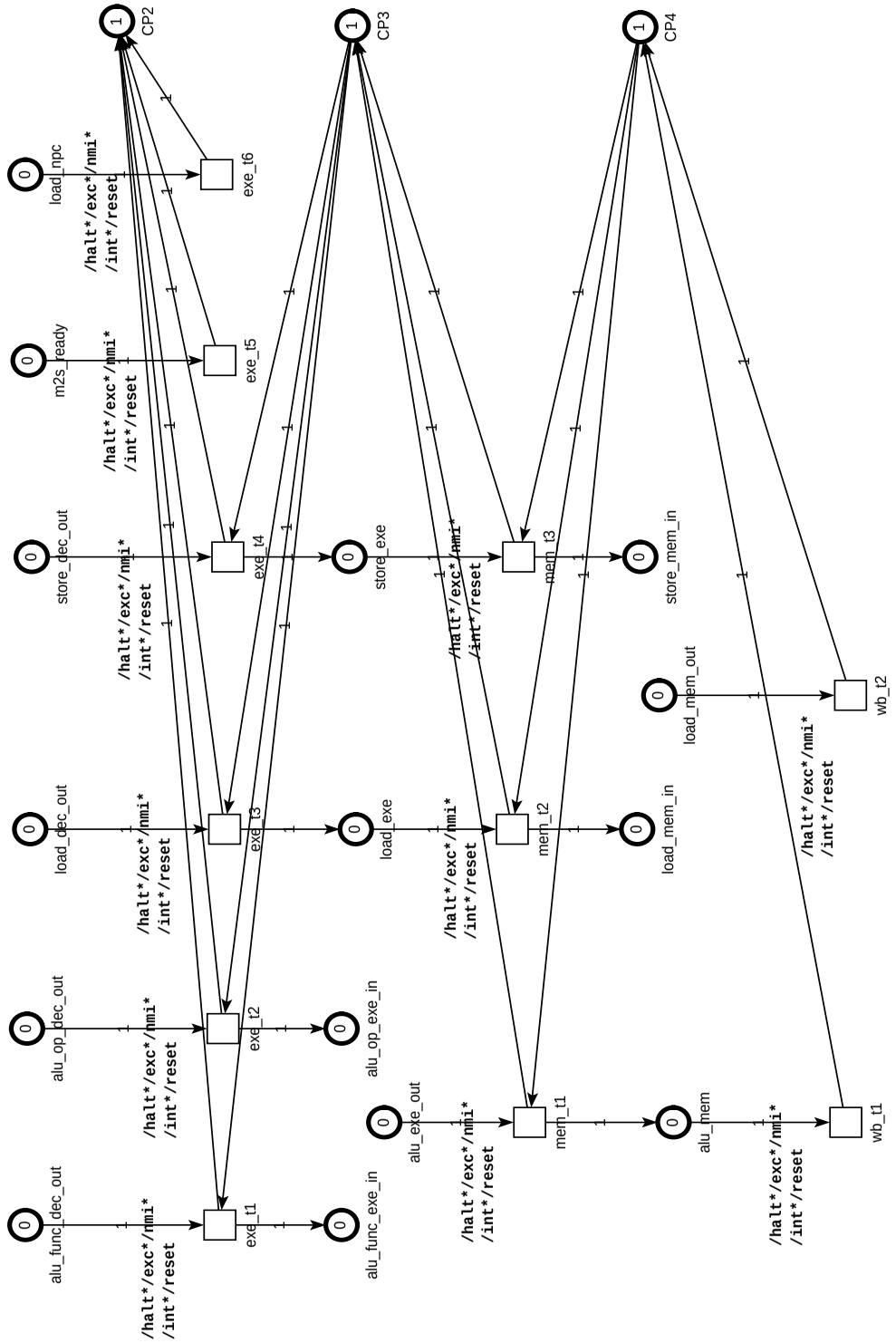

Abbildung B.2: Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil b - Pipeline-Struktur DEC/EXE/MEM/WB)

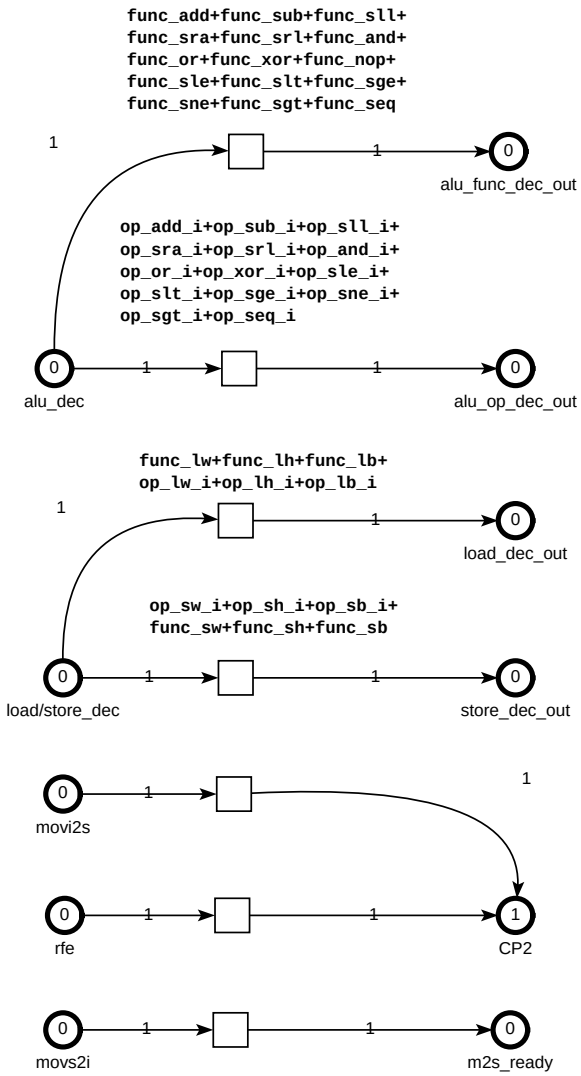

Abbildung B.3: Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil c - Untersetzung der DEC-STUFE)

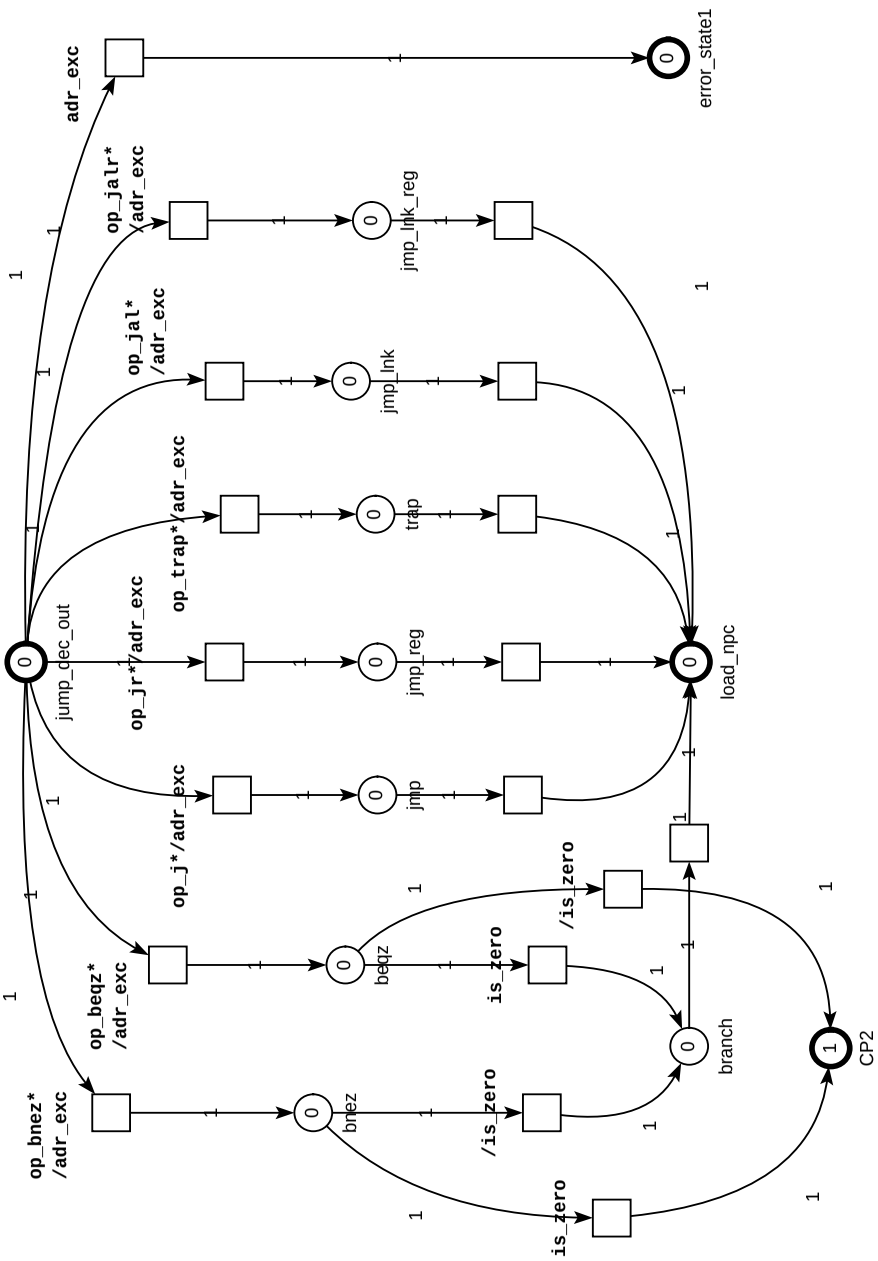

Abbildung B.4: Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil d - Ausführung der Sprungbefehle)

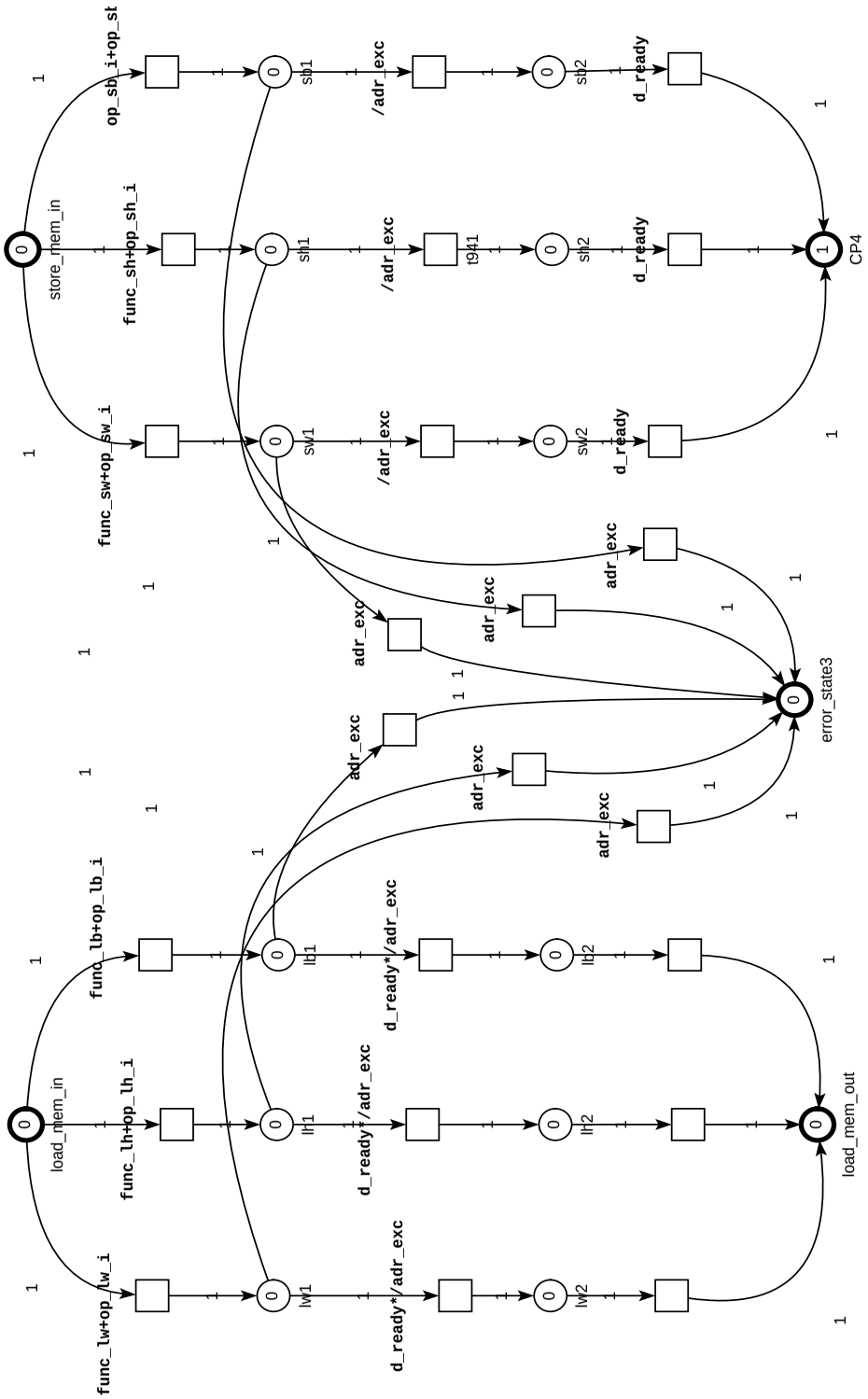

Abbildung B.5: Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil e - Ausführung der Speicherbefehle)

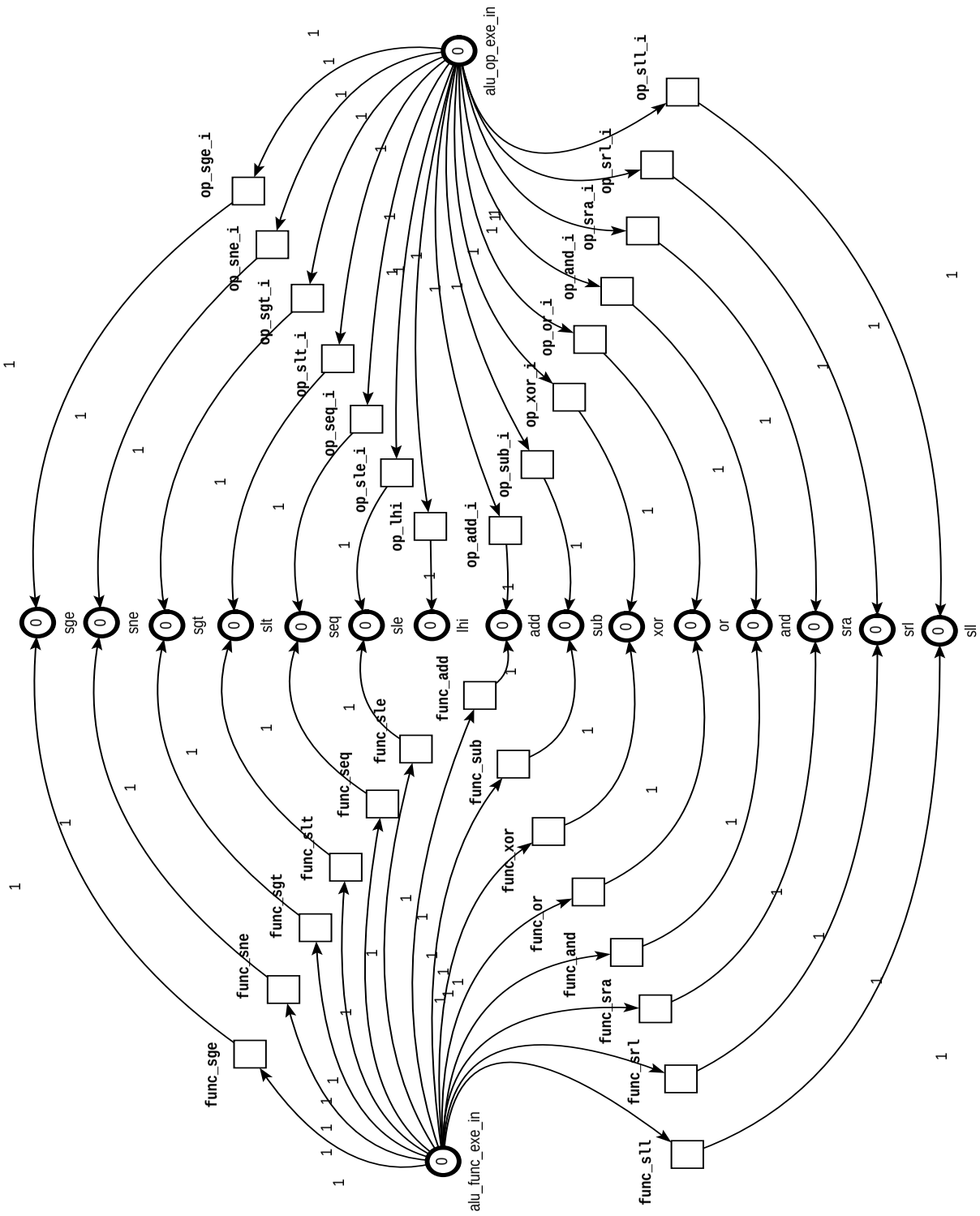

Abbildung B.6: Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil f - Auswahl der ALU-Befehle)

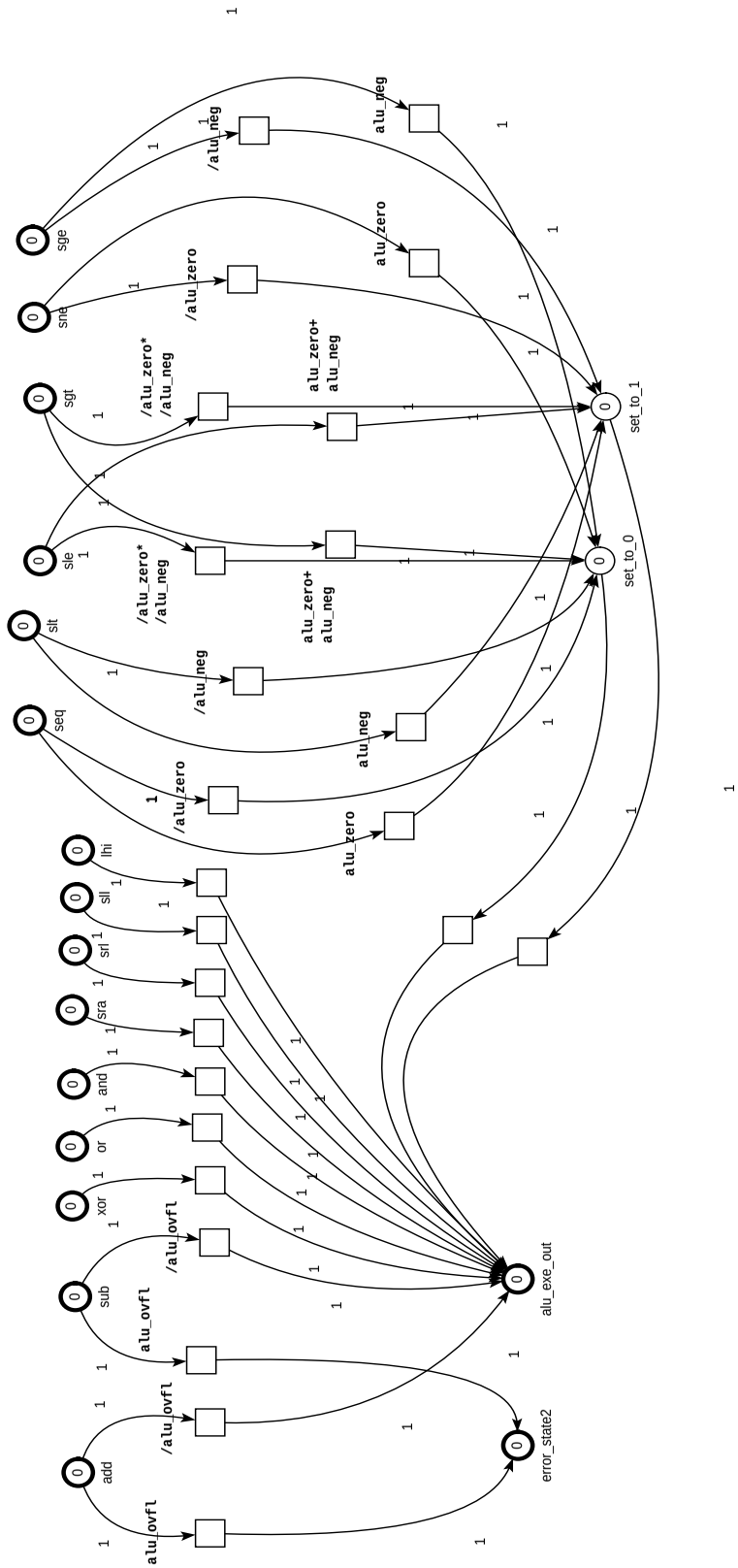

Abbildung B.7: Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil g - Ausführung der ALU-Befehle)

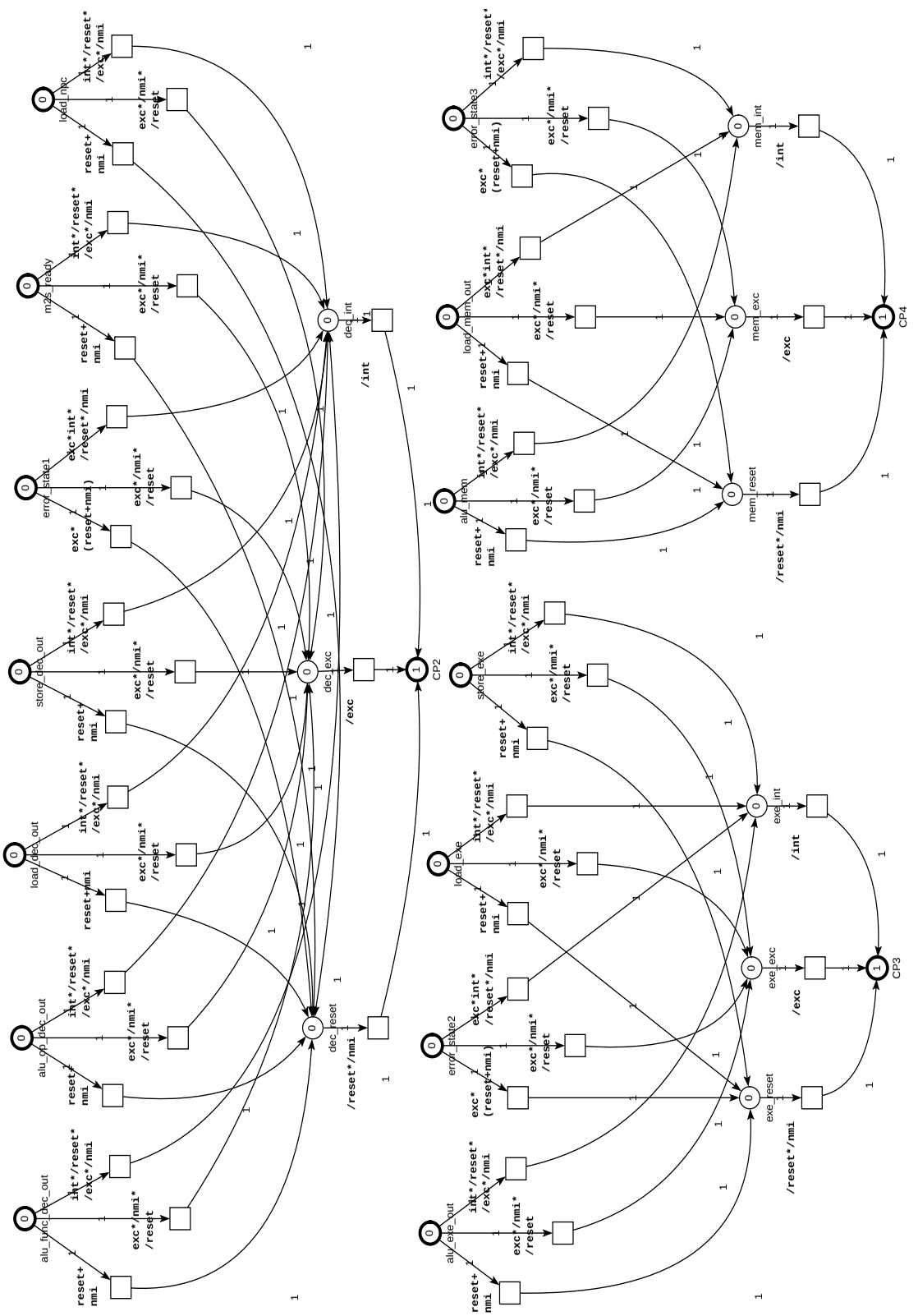

Abbildung B.8: Petrinetz-Modell des nebenläufigen DLX-Prozessors (Teil h - Pipeline-Kontrolle)

Anhang C

Formelzeichen

Petrinetze

N	Petrinetz
$T,\ t_i$	Transitionenmenge, Transition
$P,\ p_j$	Platzmenge, Platz
$F,\ f_k$	Kantenmenge, Kante
$m,\ m_l$	Markierungsmenge, Markierung
m_0	Initialmarkierung
$K,\ W$	Platzkapazität, Kantengewicht
$x\bullet,\ \mid x\bullet\mid$	Nachbereich des Knotens x, Knotenzahl im Nachbereich des Knotens x
$\bullet x,\ \mid \bullet x\mid$	Vorbereich des Knotens x, Knotenzahl im Vorbereich des Knotens x
$m_l\|t_i\rangle$	Aktivierung der Transition t_i bei Markierung m_l
$\|m_l(p_j)\|$	Markenzahl auf Platz p_j bei Markierung m_l
σ	Schaltsequenz
$m_0\|\sigma\rangle m_l$	Schaltsequenz von der Initialmarkierung m_0 zur Markierung m_l
R_n	Erreichbarkeitsgraph des Petrinetzes N
P_D	Platzmenge Deadlock
P_T	Platzmenge Trap
I	Inzidenzmatrix
$\mathbf{k}$	Schalthäufigkeitsvektor
$\mathbf{x},\ \mathbf{y}$	T-Invariante, P-Invariante
ω	Netzsymmetrie

SIPN - steuerungstechnisch interpretierte Petrinetze

$P,\ p_j(e_j, a_j)$	Platzmenge, Platz mit Eingangs- und Ausgangssignal
$F^{pre},\ f_k^{pre} = (p_j, t_i)$	Prekantenmenge, Prekante
$F^{post},\ f_k^{post} = (t_i, p_j)$	Postkantenmenge, Postkante
$G,\ G_i$	Menge der Schaltbedingungen (Guardmenge), Schaltbedingung
$X,\ x_i$	Eingabemenge, Eingangssignal
$Y,\ y_j$	Ausgabemenge, Ausgangssignal
Q_t	Abbildung zur Transitionsaktivierung
Q_p	Abbildung zur Platzmarkierung

Mengen, Aussagelogik

$\emptyset$	leere Menge
$\mathbb{N}$	Menge der natürlichen Zahlen
$X \to \mathbb{N}$	Abbildung von X auf die natürlichen Zahlen
$\cup,\ \cap$	Vereinigungsmenge, Schnittmenge
$\subseteq,\ \subset$	Teilmenge, echte Teilmenge
$\in$	Element von
$\times,\ \setminus$	Produktmenge, Differenzmenge
$\exists$	Existenzquantor
$\forall$	Allquantor
$\Rightarrow$	Implikation
$\Leftrightarrow$	Äquivalenz
$\vee,\ \wedge$	Disjunktion, Konjunktion zur Verknüpfung von Mengenelementen
$+,\ *,\ /$	Disjunktion, Konjunktion, Negation zur Verknüpfung von Schaltbedingungen im Petrinetz-Modell

Literaturverzeichnis

(Ber00) Janick Bergeron. *Writing Testbenches - Functional verification of HDL Models*. Boston: Kluwer Academic Publisher, 2000.

(BL00) Ivan Blunno and Luciano Lavagno. Deriving signal transition graphs from behavioral verilog HDL. In Yakovlev, A., Gomes, L., and Lavagno, L., editors, *Hardware Design and Petri Nets*, pages 151–170. Boston: Kluwer Academic Publishers, 2000.

(Brz95) Janusz Brzozowski. *Asynchronous Circuits.* New York: Springer-Verlag, 1995.

(CF00) Suck-Heui Chung and Steve Furber. The design of the control circuits for an asynchronous instruction prefetch unit using signal transition graphs. In A. Yakovlev, L. Gomes, and L. Lavagno, editors, *Hardware Design and Petri Nets*, pages 171–190. Boston: Kluwer Academic Publishers, 2000.

(CHEP71) F. Commoner, Anatol W. Holt, Shimon Even, and Amir Pnueli. Marked directed graphs. *Journ. Computer and System Science 5*, pages 511–523, 1971.

(Chu87) T. A. Chu. Synthesis of self-timed VLSI circuits from graph-theoretic specifications. In *Proc. of the IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors (ICCD'87)*, pages 220–223. IEEE Computer Society Press, 1987.

(CKK+97) Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alex Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.

(CKK+00) Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alex Yakovlev. Hardware and Petri nets: Application to asynchronous circuit design. In Nielsen, M. and Simpson, D., editors, *Lecture Notes in Computer Science: 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000), Aarhus, Denmark, June 2000*, volume 1825, pages 1–15. Springer-Verlag, 2000.

(DA92) Rene David and Hassane Alla. *Petri nets and Grafcet. Tools for modelling discrete event systems .* New York: Prentice Hall, 1992.

(Des98a) Jörg Desel. Basic linear algebraic techniques for place/transition nets. In Reisig, W. and Rozenberg, G., editors, *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, volume 1491, pages 257–308. Berlin: Springer-Verlag, 1998.

(Des98b) Jörg Desel. Petrinetze als Grundlage der Ablaufmodellierung. In *Tagungsband Modellierung '98. Bericht Nr. 6/98-I.*, pages 55–57. Institut AIBF, Universität Karlsruhe, 1998.

(DP71) Jack B. Dennis and Suhas S. Patil. Speed-independent asynchronous circuits. In *Proc. of the 4th Hawaii International Conference on Systems Sciences HICCS, Honolulu*, pages 55–58, 1971.

(ERS99) Werner Erhard, Andreas Reinsch, and Torsten Schober. Petrinetz-basierter Entwurf asynchroner rekonfigurbarer Systeme. In *Tagungsband 15. GI/ITG-Fachtagung Architektur von Rechensystemen ARCS'99*, pages 121–132, Oct 1999.

(ERS01a) Werner Erhard, Andreas Reinsch, and Torsten Schober. Formale Verifikation sequentieller Kontrollpfade mit Petrinetzen. *Berichte zur Rechnerarchitektur*, 7(2):1–42, Feb 2001.

(ERS01b) Werner Erhard, Andreas Reinsch, and Torsten Schober. Modelling and verification of sequential control paths using Petri nets. In *Proc. of International Workshop on Discrete Event System Design DESDes'01*, pages 41–46, Jun 2001.

(ERS01c) Werner Erhard, Andreas Reinsch, and Torsten Schober. Verification of digital control paths using Petri nets. In *Proc. of IEEE International Conference on Systems, Man and Cybernetics SMC'01*, pages 2694–2699, Oct 2001.

(Eve01) Hans Eveking. Formale Verifikationsverfahren. *Informationstechnik und Technische Informatik*, 43(1):3–5, Jan 2001.

(FAP97) Joao M. Fernandes, Marian Adamski, and Alberto J. Proenca. VHDL generation from hierarchical Petri net specifications of parallel controllers. In *IEE Proceedings: Computers and Digital Techniques*, pages 127–137, Mar 1997.

(Feh93) Rainer Fehling. A concept of hierarchical Petri nets with building blocks. In Rozenberg, G., editor, *Lecture Notes in Computer Science; Advances in Petri Nets 1993*, volume 674, pages 148–168. Berlin: Springer-Verlag, 1993.

(GL81) Hartmann J. Genrich and Kurt Lautenbach. System modelling with high-level Petri nets. *Theoretical Computer Science 13*, pages 109–136, 1981.

(GLT80) Hartmann J. Genrich, Kurt Lautenbach, and P. S. Thiagarajan. Elements of general net theory. In Brauer, W., editor, *Lecture Notes in Computer Science: Net Theory and Applications, Proc. of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979*, volume 84, pages 21–163. Berlin: Springer-Verlag, 1980.

(Gum95) Martin Gumm. VHDL - modelling and synthesis of the DLXS RISC processor. In *VLSI Design Course, University of Stuttgart, Germany*, 1995.

(Hac72) Michel H.T. Hack. *Analysis of Production Schemata by Petri Nets.* Cambridge, Mass.: MIT, Project MAC, TR-94, Feb 1972.

(Hau95) Scott Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, Jan 1995.

(HJJJ85) Peter Huber, Arne M. Jensen, Leif O. Jepsen, and Kurt Jensen. Towards reachability trees for high-level Petri nets. *Lecture Notes in Computer Science: Advances in Petri Nets 1984*, 188:215–233, 1985.

(HP90) John H. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach.* Morgan Kaufmann Publisher Inc., 1990.

(HP96) John H. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach.* Morgan Kaufmann Publisher Inc., 1996.

(Huf54a) David A. Huffman. The synthesis of sequential switching circuits. *IRE Transactions on Electronic Computers*, 257(3):161–190, 1954.

(Huf54b) David A. Huffman. The synthesis of sequential switching circuits. *IRE Transactions on Electronic Computers*, 257(4):275–303, 1954.

(Idz01) Ewa Idzikowska. Petri net models of VHDL control statements. In *Proc. of International Workshop on Discrete Event System Design DESDes'01*, pages 195–202, Jun 2001.

(ITR01) ITRS. *International Technology Roadmap for Semiconductors, 2001 Edition.* http://public.itrs.net, 2001.

(Jen91) Kurt Jensen. Coloured Petri nets: A high level language for system design and analysis. In Rozenberg, G., editor, *Lecture Notes in Computer Science, Advances in Petri Nets 1990*, volume 483, pages 342–416. Berlin: Springer-Verlag, 1991.

(KO98) Ekkart Kindler and Frank Oschmann. The Petri Net Kernel - An INA-Pilot. In *Tagungsband GI-Workshop Algorithmen und Werkzeuge für Petrinetze AWPN'98*, 1998.

(KQ88) Rainer König and Lothar Quäck. *Petri-Netze in der Steuerungstechnik.* München: Oldenbourg, 1988.

(KW99) Ekkart Kindler and Michael Weber. The Petri Net Kernel: An infrastructure for building Petri net tools. In *21st International Conference on Application and Theory of Petri Nets (ICATPN 1999), Petri Net Tool Presentations, Williamsburg, USA, June 1999*, pages 10–19, 1999.

(Lav92) Luciano Lavagno. *Synthesis and Testing of Bounded Wire Delay Asynchronous Circuits from Signal Transition Graphs*. PhD thesis, U.C. Berkeley, Nov 1992.

(LSV93) Luciano Lavagno and Alberto Sangiovanni-Vincentelli. *Algorithms for Synthesis and Testing of Asynchronous Circuits*. Boston: Kluwer Academic Publisher, 1993.

(MB59) David E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, April 1959.

(MFES00) Ricardo J. Machado, Joao M. Fernandes, Antonio J. Esteves, and Henrique D. Santos. An evolutionary approach to the use of Petri net based models – from parallel controllers to HW/SW codesign. In Yakovlev, A., Gomes, L., and Lavagno, L., editors, *Hardware Design and Petri Nets*, pages 205–222. Boston: Kluwer Academic Publishers, 2000.

(Mis73) David Misunas. Petri nets and speed independent design. *Communications of ths ACM*, 16(8):474–481, Aug 1973.

(MMB00) Norian Marranghello, Jaroslav Mirkowski, and Krzysztof Bilinski. Synthesis of synchronous digital systems specified by Petri nets. In Yakovlev, A., Gomes, L., and Lavagno, L., editors, *Hardware Design and Petri Nets*, pages 129–150. Boston: Kluwer Academic Publishers, 2000.

(Mur89) Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

(OC93) Serafin Olcoz and Jose-Manuel Colom. A Petri net approach for the analysis of VHDL descriptions. In *Lecture Notes in Computer Science, Correct Hardware Design and Verification Methods (CHARME93)*, volume 683, pages 15–26. Berlin: Springer-Verlag, 1993.

(Olc95) Serafin Olcoz. A formal Petri net model of VHDL using colored Petri nets. In *Formal Semantics for VHDL*, pages 140–169. Boston: Kluwer Academic Publisher, 1995.

(Osc99) Frank Oschmann. Erweiterung des INApilot. In *Technischer Bericht, LS Theorie der Programmierung, Institut für Informatik, Humboldt Universität zu Berlin*, Apr 1999.

(Pat70) Suhas S. Patil. *Coordination of Asynchronous Events.* PhD thesis, MIT, Dept. Electrical Engineering, May 1970.

(PCKR98) Enric Pastor, Jordi Cortadella, Alexander Kondratyev, and Oriol Roig. Structural methods for the synthesis of speed-independent circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1108–1129, 1998.

(PD72) Suhas S. Patil and Jack B. Dennis. The description and realization of digital systems. *Innovative Architecture*, pages 223–226, 1972.

(Pet62) Carl Adam Petri. *Kommunikation mit Automaten.* Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

(RE98) Wolfgang Reisig and Grzegorz Rozenberg (Eds.). *Lectures on Petri nets II: Applications, LNCS 1492*. Berlin: Springer-Verlag, 1998.

(Rei87) Wolfgang Reisig. Place/Transition systems. In Brauer, W., Reisig, W., and Rozenberg, G., editors, *Lecture Notes in Computer Science: Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, September 1986*, volume 254, pages 117–141. Springer-Verlag, 1987.

(RFH00) Patrick Rokyta, Wolfgang Fengler, and Thorsten Hummel. Electronic system design automation using high level Petri nets. In Yakovlev, A., Gomes, L., and Lavagno, L., editors, *Hardware Design and Petri Nets*, pages 193–204. Boston: Kluwer Academic Publishers, 2000.

(RR98) Wolfgang Reisig and Grzegorz Rozenberg. Informal introduction to Petri nets. In *Lecture Notes in Computer Science: Advances in Petri Nets. Lectures on Petri Nets I*, volume 1491, pages 1–11. Berlin: Springer-Verlag, 1998.

(RS98) Andreas Reinsch and Torsten Schober. Entwicklung eines Systems für die Umsetzung asynchroner Schaltungen in rekonfigurierbare Hardware. *Universität Jena, Institut für Informatik, unveröffentlicht*, Okt 1998.

(RT86) Grzegorz Rozenberg and P. S. Thiagarajan. Petri nets: Basic notions, structure, behaviour. In de Bakker, J.W., de Roever, W.P., and Rozenberg, G., editors, *Lecture Notes in Computer Science: Current Trends in Concurrency*, volume 224, pages 585–668. Springer-Verlag, 1986.

(RY85) L. Y. Rozenblum and Alexandre V. Yakovlev. Signal graphs: From self-timed to timed ones. In *International Workshop on Timed Petri Nets, Torino*, pages 199–206. IEEE Computer Society Press, Jul 1985.

(Sch98) Thomas Schlenker. Redesign eines DLX-Prozessormodells mit Pipeline. *Diplomarbeit, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner*, Apr 1998.

(SM83) Ichiro Suzuki and Tadao Murata. A method for stepwise refinement and abstraction of Petri nets. *Journal of Computer and System Science*, 27(1):51–76, Aug 1983.

(SR01) Peter H. Starke and Stephan Roch. Integrated Net Analyzer INA, Version 2.2., LS Theorie der Programmierung, Institut für Informatik, Humboldt Universität zu Berlin. Mar 2001.

(Sta80) Peter H. Starke. *Petrinetze.* Berlin: Deutscher Verlag der Wissenschaften, 1980.

(Sta90) Peter H. Starke. *Analyse von Petri-Netz-Modellen.* Stuttgart: Teubner, 1990.

(Val79) Robert Valette. Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Science*, 18:35–46, 1979.

(vGvBB98) Hans van Gageldonk, Kees van Berkel, and Arjen Bink. VLSI programming of DLX microprocessor cores. In *Proc. of Asynchronous Circuit Design (ACiD) WG Workshop, Turin.*, Jan 1998.

(Wag02) Katia Wagner. Petri-Netz-basierte Implementierung einer formalen Verifikation sequentieller Kontrollpfade. *Studienarbeit, Universität Jena, Institut für Informatik*, pages 1–22, Aug 2002.

(YGL00) Alex Yakovlev, Luis Gomes, and Luciano Lavagno. *Hardware Design and Petri Nets.* Boston: Kluwer Academic Publisher, 2000.

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel und Literatur angefertigt habe.

Jena, den 21. August 2002

Danksagung

Mein Dank geht an Prof. Dr. Erhard für die umfassende Betreuung dieser Arbeit. Der Thüringer Graduiertenförderung danke ich für die Gewährung eines Graduiertenstipendiums. Für das gute Arbeitsklima und die Hilfsbereitschaft danke ich allen Mitarbeitern des Lehrstuhls für Rechnerarchitektur und -kommunikation.

Ich danke Michael für gute Ratschläge. Ganz besonders danke ich Andrea und Paul Maximilian für Inspiration und die für Engelsgeduld während der letzten Monate. Mein letzter Dank gilt meinen Eltern, für all die moralische und praktische Unterstützung.

Jena, den 21. August 2002

Lebenslauf

Torsten Schober
Diplomphysiker
*13.7.1969, Halle (Saale)

1976 - 1988	Polytechnische und Erweiterte Oberschule mit Abitur in Halle
1988 - 1989	Wehrdienst
1989 - 1991	Grundstudium der Physik an der Martin-Luther-Universität Halle-Wittenberg
1991 - 1996	Hauptstudium der Physik an der Humboldt Universität zu Berlin
1995 - 1996	Diplomarbeit am Institut für Hochenegiephysik DESY Zeuthen
1997 - 1998	Wissenschaftlicher Mitarbeiter an der Friedrich-Schiller-Universität Jena Lehrstuhl für Rechnerarchitektur und -kommunikation Projekt am Europäischen Kernforschungszentrum CERN
1999 - 2001	Graduiertenstipendiat des Landes Thüringen
2001 - 2002	Wissenschaftlicher Mitarbeiter an der Friedrich-Schiller-Universität Jena Lehrstuhl für Rechnerarchitektur und -kommunikation

Jena, den 21. August 2002