Carl von Ossietzky Universität Oldenburg

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

# Ein Verfahren zur zustandsbasierten Abstraktion und Simulation der Leistungsaufnahme von digitalen integrierten Schaltungskomponenten auf Systemebene

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften
der Carl von Ossietzky Universität Oldenburg
zur Erlangung des Grades und Titels eines

**Doktors der Ingenieurwissenschaften (Dr.-Ing.)**

angenommene Dissertation

**von Herrn Daniel Lorenz**

geboren am 23. September 1986 in Nordhausen

Gutachter: Prof. Dr.-Ing. Wolfgang Nebel
Weitere Gutachter: Prof. Dr.-Ing. Andreas Rauh

Tag der Disputation: 03. März 2022

**Zusammenfassung**

Aufgrund der steigenden Komplexität sowie Leistungsfähigkeit und somit auch der Leistungsaufnahme von heutigen digitalen Schaltungssystemen müssen schon früh im Designprozess Techniken für die Energieverwaltung integriert werden. Dazu ist es nötig, die Leistungsaufnahme abhängig von der tatsächlichen Nutzung des Systems in Simulationsläufen auf Systemebene sichtbar und analysierbar zu machen.

Im Rahmen dieser Arbeit werden sogenannte Ein-Chip-Systeme, bestehend aus Prozessorkernen, Bussen, Speichern und Hardwarebeschleunigern, betrachtet. Aus Softwareentwicklungs- und Systemintegrationssicht werden diese IP-Komponenten von Zulieferern auf der *Register Transfer Level* (RTL) entwickelt, getestet und können dort oder auf tieferen Abstraktionsebenen bezüglich ihrer Leistungsaufnahmeeigenschaften untersucht werden. Die dort ermittelten Leistungsaufnahmeeigenschaften müssen nun durch entsprechende Modelle den Entwicklern auf der Systemebene zugänglich gemacht werden. Diese Modelle sollen dafür die folgenden Eigenschaften besitzen: Einfachheit, Ausführbarkeit, Unabhängigkeit vom funktionalen Design, horizontale und vertikale Kombinierbarkeit, Adaptivität, Performanz und Genauigkeit.

Auf der Systemebene gibt es bisher nur wenige Methoden zur Simulation der Leistungsaufnahme. Speziell für IP-Komponenten, bei denen lediglich die äußeren Schnittstellen und nicht der innere Aufbau bekannt sind, werden entsprechende Modelle zur Simulation der Leistungsaufnahme benötigt. Da die energetischen Eigenschaften von Hardwarekomponenten erst auf deutlich niedrigeren Abstraktionsebenen physikalisch bestimmt werden können, müssen diese in geeigneter Weise abstrahiert werden. Hier fehlen grundlegende Methoden, um eine valide Abstraktion der Werte zu garantieren und diese abzusichern.

Diese Arbeit stellt eine Methodik vor, welche die gezielte Modellierung und Simulation der Leistungsaufnahme von Systemkomponenten auf Systemebene erlaubt. Dabei kann die Modellierung im Besonderen in Kombination mit *Black-Box*-IP-Komponenten genutzt werden. Für den Einsatz der Methodik ist keine Änderung der Komponenten oder des Systems erforderlich und es ist leicht integrier-, austausch- und konfigurierbar. Die Methodik erlaubt eine Kopplung mit weiteren extra-funktionalen Modellen wie z.B. Temperaturmodellen, deren Verhalten von der Leistungsaufnahme abhängt. Des Weiteren wird ein teilautomatisiertes Vorgehen vorgestellt, welches eine valide Charakterisierung der kontrollpfad- und datenpfadabhängigen Leistungsaufnahme aus niedrigeren Abstraktionsebenen erlaubt und diese Eigenschaften für einen zuverlässigen Gebrauch absichert.

Die Evaluation zeigt, dass die entwickelte Methodik einen sinnvollen Kompromiss zwischen hoher Genauigkeit und geringem Aufwand erreicht sowie dass das Leistungsaufnahmemodell leicht erstellt und in ein bestehendes Gesamtsystemmodell integriert werden kann.

# Inhaltsverzeichnis

# KAPITEL 1

# Einleitung

In der heutigen Zeit werden einfache mechanische und elektrische Schaltungen durch komplexe digitale integrierte Schaltungen ersetzt, die verschiedenste Steuer und Regelaufgaben übernehmen. Des Weiteren gibt es ein immer größer werdendes Anwendungsfeld an umfangreichen Systemen für die Unterhaltungs- und Informationselektronik. Diese Art der Systeme wird als eingebettetes System bezeichnet. Um bei diesen sowohl Flexibilität als auch Durchsatzfähigkeit zu gewährleisten, bestehen diese Systeme in den meisten Fällen aus flexibler Software, die von einem Prozessor ausgeführt wird, und Hardware, die eine schnelle und energiesparende Bearbeitung der Daten ermöglicht. Durch die zunehmenden Anforderungen bezüglich Datendurchsatz und Anwendungsumfang wächst die Komplexität und Leistungsfähigkeit solcher Systeme stetig. Die aus der Vergangenheit bekannten Programme zum Schaltungsentwurf wurden komplett durch abstrakte Entwurfsmethodiken ersetzt. Angefangen bei einer groben Spezifikation des Systems, geht der Entwurfsfluss über verschiedene Abstraktionsebenen, bei denen das System immer weiter bis hin zur fertigen Implementierung und Umsetzung verfeinert wird.

Viele solcher Systeme werden mobil eingesetzt und somit durch mobile Energieversorgungen betrieben. Diese haben in der Regel einen begrenzten Energievorrat. Die hohe Leistungsfähigkeit führt unweigerlich auch zu einer gesteigerten Leistungsaufnahme, die wiederum zu einer spürbaren Temperaturerhöhung oder unter Umständen zu einer Überhitzung des Systems führt. Aus diesem Grund sollten mobile eingebettete Systeme so sparsam wie möglich sein, um die bereitgestellte Energie effizient auszunutzen. Damit dies umgesetzt werden kann, müssen schon früh im Entwurfsfluss (auf hohen Abstraktionsebenen) Modelle zur Verfügung gestellt werden, die eine Abschätzung über die resultierende Leistungsaufnahme ermöglichen. Untersuchungen haben gezeigt, dass durch Optimierungen des Systems auf höheren Abstraktionsebenen deutlich mehr Energie eingespart werden kann als auf niedrigen Abstraktionsebenen [75].

Mit Hilfe eines solchen Modells kann die Leistungsaufnahme statisch abgeschätzt werden. Da die Leistungsaufnahme in den meisten Fällen stark von der Benutzung des Systems abhängt, ist es notwendig, dass die zu erwartende Leistungsaufnahme von allen Systemkomponenten für realistische Anwendungsfälle analysiert werden kann. Dies wird z.B. in einer Simulation auf *Electronic System Level* (ESL) erreicht. Auf ESL wird ein System soweit abstrahiert, dass es effizient simuliert werden kann.

Diese Simulation beinhaltet die Leistungsaufnahme von Prozessorkernen und seinen integrierten Powermanagementfähigkeiten sowie die Leistungsaufnahme von zusätzlicher Hardware wie z.B. Bussen, Speichern oder dedizierten Hardwarebeschleunigern. Die meisten solcher Komponenten werden nicht von Grund auf neu entwickelt, sondern aus vorherigen Designs wiederverwendet oder von *Intellectual Property* (IP) Anbietern eingekauft. Gewöhnlich beinhalten diese Komponenten keine Informationen über ihre Leistungsaufnahme. In einigen Fällen sind Datenblätter vorhanden, die es erlauben, in Abhängigkeit von Parametern wie Taktfrequenz, Versorgungsspannung, Zieltechnologie und durchschnittlicher Schaltaktivität eine durchschnittliche Leistungsaufnahme abzuschätzen. Jedoch beinhalten diese Werte keine dynamischen Effekte wie z.B. Kontrollfluss- oder Datenabhängigkeiten und führen zu einem nicht vernachlässigbaren Fehler.

Um die Leistungsaufnahme eines Systems zu modellieren und die effiziente Nutzung der Hardwarekomponenten auf Systemebene zu evaluieren, wird ein holistischer Ansatz für die Modellierung der Leistungs- und Energieaufnahme benötigt, der sich nahtlos in virtuelle Plattformen auf ESL integriert.

Der in Abbildung 1.1 gezeigte ESL Entwurfsfluss skizziert das typische Vorgehen zum Erstellen des Designs auf ESL. Hierbei wird mit den Anforderungen an das System begonnen und ein rein funktionales Design auf algorithmischer Ebene entwickelt. Dieses wird darauffolgend in Hardware- und Softwareanteile partitioniert. Der Hardwareentwurf wird als sogenannte virtuelle Plattform umgesetzt, die eine Modellierung und Simulation der Hardwareplattform ermöglicht. Ist dieser fertig, können entsprechende Modelle für die Leistungsaufnahme entwickelt und an die virtuelle Plattform gebunden werden. Auf dieser wird die entwickelte Software nun ausgeführt und verschiedene Aspekte wie korrektes funktionales Verhalten sowie erwartungsgemäße Zeit- und Leistungseigenschaften geprüft. Diese Ergebnisse können genutzt werden, um das System zu optimieren. Die in dieser Entwurfsphase getroffenen Entscheidungen haben den größten Einfluss auf das resultierende Design.

An dieser Stelle zeigt sich, dass die Integration der Leistungsaufnahmemodelle sowohl dazu genutzt werden kann, die Software dahingehend zu optimieren, dass sie die Hardware effizienter nutzt als auch für die Verbesserung der Hardwareauswahl bzw. des Hardwaredesigns, um bei korrekter Funktion und Einhaltung weiterer Randbedingungen die Leistungsaufnahme und Leistungsaufnahmespitzen bzw. die Energieaufnahme und Energieaufnahmespitzen zu verringern.

Für Leistungsaufnahmemodelle haben sich zustandsbasierte Ansätze etabliert, die einem funktionalen Zustand eine definierte Leistungsaufnahme zuordnen. Ein solches Modell muss dabei noch weitere Aspekte betrachten, die eine Modellierung schwierig machen bzw. die Genauigkeit des Modells negativ beeinflussen. Diese werden nachfolgend kurz beschrieben.

Die IP-Komponenten sind in den meisten Fällen sogenannte *Black-Box*-Komponenten, bei denen nur die äußeren Schnittstellen und nicht der innere Aufbau bekannt sind. Das Verbergen der internen Zustände sorgt gleichzeitig dafür, dass bestimmte funktionale

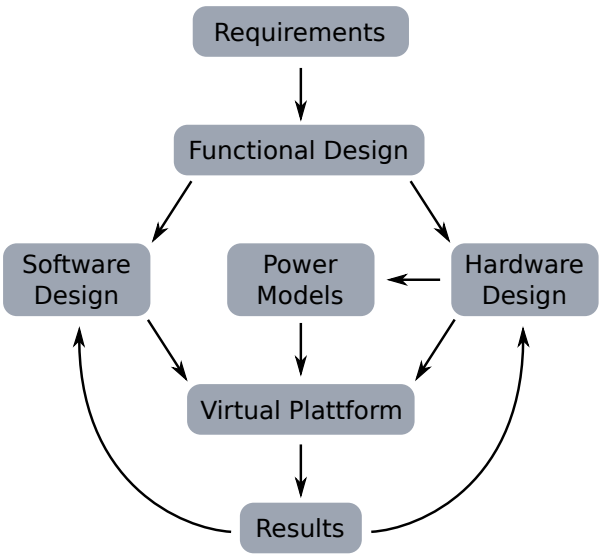



Abbildung 1.1.: ESL Entwurfsfluss

Zustandswechsel, die einen Einfluss auf die Leistungsaufnahme haben, nicht an den äußeren Schnittstellen sichtbar sind. Ein Leistungsaufnahmemodell kann also nicht auf interne Informationen zurückgreifen und muss in der Lage sein, die Leistungsaufnahme aus den Schnittstelleninformationen abzuleiten. Des Weiteren führt dies dazu, dass es vollständig unabhängig vom funktionalen Modell sein muss.

In vielen Fällen haben neben dem aktuellen Zustand auch die Eingangsdaten einen deutlichen Einfluss auf die Leistungsaufnahme. Diese muss ein Modell mit betrachten, um eine akzeptable Genauigkeit zu erreichen.

Konkrete Leistungsaufnahmewerte lassen sich erst durch Simulationen auf niedrigen Abstraktionsebenen erzeugen oder durch Messungen auf einer reellen Hardware. Diese Werte müssen über geeignete Methoden in die abstrakten Modelle auf ESL überführt werden.

In diesem Bereich gibt bereits viele Arbeiten, die sich mit einigen der genannten Aspekte beschäftigen, jedoch nicht alle abdecken. Die Schwierigkeit hierbei ist, sowohl einen guten Kompromiss zwischen Simulationsgeschwindigkeit und Genauigkeit zu finden als auch die Modelle zielgerichtet aus den Leistungsaufnahmewerten zu synthetisieren. Das Ziel dieser Arbeit ist, für alle genannten Aspekte eine Lösung zu finden, die durch ein entsprechendes Modell und Synthesevorgehen umgesetzt wird.

## 1.1. Ziele der Arbeit

Aus den in der Einleitung beschriebenen Problemen ergeben sich für diese Arbeit eine Reihe an Zielen, die hier kurz beschrieben werden. Aus diesen Zielen werden die im nächsten

Abschnitt erläuterten Anforderungen an das Resultat dieser Arbeit abgeleitet.

In dieser Arbeit soll ein Modell zur Simulation der Leistungsaufnahme entwickelt werden, welches leicht in eine Systemebenensimulation von komplexen digitalen integrierten Schaltungen eingebunden werden kann. Dieses soll im Zusammenhang mit *Black-Box*-Komponenten eingesetzt werden können und unabhängig von Versorgungsspannung und Taktfrequenz sein. Es soll die Ankopplung von weiteren Modellen (wie z.B. ein Temperaturmodell) ermöglichen und einen bestmöglichen Kompromiss aus Genauigkeit und Simulationsgeschwindigkeit bieten. Da Leistungsaufnahmewerte erst auf *Gate Level* (GL) zur Verfügung stehen, soll für die Erstellung des Modells ein Verfahren entworfen werden, welches möglichst automatisiert aus Leistungsaufnahme-Traces von GL das entwickelte Modell herleitet.

Um diese Ziele zu erreichen, setzt sich die Arbeit mit Fragestellungen auseinander, die thematisch nachfolgend eingeordnet werden können:

1. Abdeckung der Leistungsaufnahmezustände,
2. Untersuchung von verschiedenen Komponentenklassen,
3. Abhängigkeit zwischen Leistungsaufnahme und Komponenteninteraktion sowie
4. Einfluss der Datenabhängigkeit auf die Leistungsaufnahme.

Diese sollen dabei helfen, die verschiedenen Zusammenhänge zwischen Leistungsaufnahme, Interaktion und Komponentenkategorie zu verdeutlichen und den Vorgang zur Modellerzeugung so weit wie möglich zu automatisieren. Des Weiteren sollen Fehlerquellen erkannt und angezeigt bzw. korrigiert werden.

### 1.1.1. Abdeckung von Leistungsaufnahmezuständen

Für eine genaue Simulation der Leistungsaufnahme ist es notwendig, dass das Verhalten der Leistungsaufnahme möglichst vollständig vom entsprechenden Modell abgebildet wird.

**These 1** *Es gibt eine starke Kopplung zwischen funktionalen Zuständen sowie den Änderungen auf den Datenschnittstellen und Leistungsaufnahmezuständen.*

Um diese These zu belegen, wird bei einer Gate-Level-Simulation die Leistungsaufnahme aufgezeichnet. Diese Simulation soll so aufgebaut sein, dass bei unterschiedlichen Eingangsdaten jeder funktionale Zustand mehrmals aktiv ist und diese regulären Anwendungsszenarien entsprechen. Damit ist gemeint, dass die Eingangsdaten sich nicht *ungewöhnlich* verhalten sollen, wie z.B. sich ändernde Dateneingänge bei einem Zurücksetzen der Komponente.

Es werden dabei unterschiedliche Sets sowohl für das Training als auch für den Test gewählt und die Genauigkeit der Modellausgaben bewertet. Entspricht die Genauigkeit des Tests der

des Trainingssets, kann davon ausgegangen werden, dass alle Leistungsaufnahmezustände abgedeckt wurden.

Bei komplexen Komponenten ist eine vollständige Simulation des funktionalen Verhaltens inklusive aller möglichen Eingangsdatenkombination in allen Zuständen nicht in praktischer Zeit möglich. Da gerade bei Black-Box Komponenten der innere Aufbau nicht bekannt ist, ist die Simulation die einzige Methode zum Charakterisieren der Leistungsaufnahme.

### 1.1.2. Klassifizierung der Komponenten

In der Systementwicklung werden in der Regel viele verschiedene Module mit sehr unterschiedlichen Aufgaben verwendet wie z.B. Prozessorkerne, Speicher, Hardwarebeschleuniger, I/O-Komponenten oder Busse. Diese Komponentenklassen unterscheiden sich durch die grundlegend unterschiedlichen Rollen auch in ihrem Leistungsaufnahmeverhalten. Kann ein generisches Modell diese Eigenschaften abbilden, vereinfacht dies den Modellierungs- und Integrationsaufwand, da nicht für jede Komponentenklasse ein eigenes Modell gewählt werden muss.

**These 2** *Das unterschiedliche Leistungsaufnahmeverhalten von verschiedenen Komponentenklassen lässt sich teilweise durch ein generisches Modell darstellen.*

Eine Untersuchung soll zeigen, dass sich die Leistungsaufnahme von verschiedenen Komponentenklassen unterscheidet, diese aber durch ein generisches Modell dargestellt werden kann.

### 1.1.3. Abhängigkeit zwischen Energieverbrauch und Interaktion

Änderungen an den Ein- bzw. Ausgängen einer Komponente können funktionale Zustandsänderungen auslösen, die die Leistungsaufnahme beeinflussen. Umgekehrt muss eine funktionale Zustandsänderung nicht immer mit einer Änderung an den externen Schnittstellen verbunden sein. Das bedeutet, die Leistungsaufnahme kann sich durch einen funktionalen Zustandsübergang verändern, der nicht durch eine Änderung an den Ein- oder Ausgängen erkennbar ist.

**These 3** *Definierte Aktionen am Ein- bzw. Ausgang führen zu dem immer gleichen Einfluss auf die Leistungsaufnahme. Interne funktionale Zustandsübergänge ohne Änderung der externen Schnittstellen sind deterministisch in Bezug auf ihr zeitliches Auftreten.*

Es soll untersucht werden, inwieweit definierte Ein- und Ausgaben vorhanden sind, die immer den gleichen Einfluss auf die Leistungsaufnahme haben und welche diese sind. Des Weiteren soll untersucht werden, ob nicht extern sichtbare Zustandsänderungen der Komponente vorhersagbar sind. Diese Untersuchung zeigt, ob eine vollständige Modellierung des Leistungsaufnahmeverhaltens möglich ist, auch wenn nicht alle Zustandsübergänge an den externen Schnittstellen sichtbar sind.

### 1.1.4. Datenabhängigkeit auf Systemebene

Die Leistungsaufnahme ist neben dem funktionalen Zustand oft abhängig von Datenänderungen an den Schnittstellen und innerhalb der Komponente.

**These 4** *Auf Systemebene wird der Energieverbrauch so weit abstrahiert, dass sich datenabhängige Energieverbräuche heraus mitteln. Ist dies nicht der Fall, kann das Modell mit minimalem Zusatzaufwand erweitert werden, sodass es die datenabhängige Leistungsaufnahme mit abbildet.*

Es soll untersucht werden, inwieweit die Modellierung eines Durchschnittswerts für die datenabhängige Leistungsaufnahme ausreicht. Für Komponenten, bei denen durch die Modellierung einer durchschnittlichen datenabhängigen Leistungsaufnahme große Ungenauigkeiten entstehen, soll untersucht werden, ob der Zusammenhang charakterisiert und durch eine Erweiterung in das Modell mit eingebracht werden kann.

## 1.2. Anforderungen

Aus der in der Einführung beschriebenen Motivation und Problemstellung ergeben sich einige Anforderungen an das erforderliche Modell sowie die Methodik, die im Folgenden aufgelistet sind. Diese Anforderungen werden im weiteren Verlauf dieser Arbeit für die Konzepterstellung sowie die Evaluation genutzt. Bei den Anforderungen wird unterschieden zwischen funktionalen (ID beginnt mit ’F’) und nichtfunktionalen (ID beginnt mit ’N’) Anforderungen. Funktionale Anforderungen beziehen sich auf die direkte Funktionsfähigkeit des Modells, wobei die nichtfunktionalen Anforderungen Eigenschaften des Modells beschreiben. Alle Anforderungen zeichnen sich aus durch einen eindeutigen Identifikator, eine Beschreibung und die Problemstellung, die das Problem für die Anforderung beschreibt.

### 1.2.1. Funktionale Anforderungen

Nachfolgend sind alle funktionalen Anforderungen definiert. Alle funktionalen Identifikationsnummern beginnen mit einem 'F'.

**F1: Leistungsaufnahme von Black-Box Komponenten**

**Beschreibung:** Das Modell muss in der Lage sein, die Leistungsaufnahme für Black-Box Komponenten zu modellieren.

**Problemstellung:** Bei Black-Box Komponenten sind lediglich die Schnittstellen bekannt. Aus diesem Grund kann die Leistungsaufnahme nicht aus der funktionalen Beschreibung abgeleitet werden und das Leistungsaufnahmemodell kann nicht aus der funktionalen Beschreibung heraus benachrichtigt werden.

**F2: Nichtinvasive Modellierung der Leistungsaufnahme**

**Beschreibung:** Die funktionalen Modelle sollen bei der Annotation der Leistungsaufnahmemodelle nicht verändert werden, damit sichergestellt wird, dass sich das funktionale Verhalten nicht ändert. Zusätzlich darf das Kommunikationsverhalten nicht verändert werden.

**Problemstellung:** Diese Anforderung hat zwei Ursachen. Zum einen gibt es viele Komponenten (z.B. Black-Box IP Komponenten von Drittanbietern), deren innere Strukturen nicht verändert werden können. Zum anderen geht durch eine Annotation von nichtfunktionalen Eigenschaften an das funktionale Modell die Übersicht in der Implementierung verloren und es kann versehentlich das Verhalten der Komponente verändert werden. Dies wird durch eine separate Modellierung verhindert, indem die Modelle unverändert benutzt werden. Da die Ableitung der Leistungsaufnahme durch die Beobachtung der Kommunikation geschieht, muss aber auch hier sichergestellt werden, dass dieser Eingriff keinen Einfluss auf das Kommunikationsverhalten hat. Des Weiteren erlaubt die Trennung der Modelle, dass sie einzeln und unabhängig voneinander verändert und ausgetauscht werden können.

**F3: Leistungsaufnahmemodellierung unabhängig von Versorgungsspannung, Taktfrequenz und Temperatur**

**Beschreibung:** Die Modellierung der Leistungsaufnahme soll unabhängig von der aktuellen Taktfrequenz, Versorgungsspannung und Temperatur erfolgen.

**Problemstellung:** Damit die Versorgungsspannung des Systems gesenkt werden kann, muss gleichzeitig auch die Taktfrequenz gesenkt werden. Dadurch werden weniger Operationen pro Sekunde durchgeführt und die dynamische Leistungsaufnahme sinkt. Das Modell für die Leistungsaufnahme soll unabhängig von diesem Parameter modelliert werden. Weiterhin ist die statische Leistungsaufnahme abhängig von der Temperatur. Auch hier soll die Modellierung diesen Parameter abstrahieren. Die Berechnung der resultierenden Leistungsaufnahme soll in einem nachgelagerten Schritt geschehen.

**F4: Betrachtung von nicht detektierbaren Zustandsänderungen des funktionalen Modells**

**Beschreibung:** Einige Komponenten weisen funktionale Zustandsübergänge auf, die nicht durch eine gleichzeitige Änderung am Eingang ausgelöst werden bzw. keine gleichzeitige Änderung am Ausgang auslösen. Diese sollen jedoch vom resultierenden Leistungsaufnahmemodell betrachtet werden, um die Leistungsaufnahme so genau wie möglich zu simulieren.

**Problemstellung:** Interne funktionale Zustandsübergänge können die Leistungsaufnahme signifikant beeinflussen und müssen deshalb, wenn möglich, mit modelliert werden. Da diese Zustandsübergänge aber nicht durch eine gleichzeitige Änderung am Ein- oder Ausgang detektierbar sind, stellt dies insbesondere für Black-Box IP Komponenten ein Problem dar. In dieser Arbeit soll ein Vorgehen entwickelt werden, diese Zustandsübergänge zu erkennen. Außerdem soll das Leistungsaufnahmemodell in der Lage sein, diese Zustandsübergänge zu modellieren, sobald sich ein deterministisches Verhalten zeigt.

**F5: Berücksichtigung von datenabhängiger Leistungsaufnahme**

**Beschreibung:** Da bei vielen Systemkomponenten die Leistungsaufnahme nicht nur vom aktuellen Kontrollzustand abhängt, sondern auch von den verarbeiteten Daten, soll diese Abhängigkeit im Leistungsaufnahmemodell betrachtet werden, um eine hohe Genauigkeit zu erreichen.

**Problemstellung:** Die Schwierigkeit bei datenabhängiger Leistungsaufnahme besteht darin, die genaue Abhängigkeit zwischen Daten und Leistungsaufnahme zu erkennen und zu modellieren. Je nach funktionalem Verhalten kann Art und Dauer der Beeinflussung sehr verschieden sein. Es gilt, einen Weg für die Charakterisierung dieser Abhängigkeiten zu finden und in das Leistungsaufnahmemodell zu integrieren.

**F6: Modellierung auf Electronic System Level**

**Beschreibung:** Um eine Optimierung von Entscheidungen im Systementwurf durchführen zu können, müssen Systeme als Ganzes modelliert und simuliert werden.

**Problemstellung:** Auf niedrigen Abstraktionsebenen wie GL oder RTL ist der Detailgrad so hoch, dass es nicht effizient möglich ist, gesamte Systeme zu modellieren und zu simulieren. Dies wird erst auf ESL durch eine sehr hohe Abstraktion ermöglicht, die die Simulationszeit immens verkürzt. Daher ist eine Modellierung auf ESL erforderlich.

**F7: Abstraktion der Gate Level-Leistungsaufnahme nach Electronic System Level**

**Beschreibung:** Die Leistungsaufnahme hängt in der Regel von dem funktionalen Verhalten der Komponente ab. Daher wird ein Modell für die Leistungsaufnahme in der Regel

aus dem entsprechenden funktionalen Modell abgeleitet. Die nötigen Leistungsaufnahmewerte stehen erst auf GL zur Verfügung und müssen daher auf GL erzeugt und in ein entsprechendes ESL-Leistungsaufnahmemodell übertragen werden.

**Problemstellung:** Die Herausforderung bei der Übertragung der Leistungsaufnahmewerte auf GL besteht in der korrekten Abstraktion der Werte in das Modell sowie in der Herstellung der Abhängigkeit zwischen Leistungsaufnahme auf GL und funktionalem Verhalten auf ESL.

### 1.2.2. Nichtfunktionale Anforderungen

Nachfolgend sind alle nicht-funktionalen Anforderungen definiert. Alle nicht-funktionalen Identifikationsnummern beginnen mit einem 'N'.

**N1: Leistungsaufnahmemodell lässt sich einfach in Simulation integrieren**

**Beschreibung:** Das entstehende Modell für die Leistungsaufnahme soll möglichst einfach in ein bestehendes funktionales Modell für eine virtuelle Hardwareplattform integriert werden können. Dabei soll der Benutzeraufwand so gering wie möglich gehalten werden. Dies kann erreicht werden durch eine größtmögliche Automatisierung der Erstellung sowie eine verständliche und einfach gehaltene Möglichkeit der Integration und Anbindung an das funktionale Modell. Das funktionale Modell soll dabei so wenig wie möglich verändert werden müssen.

**Problemstellung:** Die in dieser Arbeit entstehende Methodik soll zum Ziel haben, die Leistungsaufnahme auf ESL simulieren zu können und dabei so wenig Zeit wie nötig zu erfordern. Der Grund ist, dass die Alternative eine Simulation auf niedrigeren Abstraktionsebenen ist, die aber deutlich mehr Zeit für die funktionale Simulation benötigen, da mehr Implementationsdetails betrachtet werden. Diese Zeit soll eingespart werden. Ein hoher Zeitaufwand für die Modellierung der Leistungsaufnahme und Integration in das funktionale Modell würde diesem entgegenwirken.

**N2: Leistungsaufnahmemodell ermöglicht einfache Benutzung**

**Beschreibung:** Ein Modell ist nur dann sinnvoll einsetzbar, wenn es eine einfache und schnelle Möglichkeit zur Konfiguration, Anwendung und Auswertung bietet.

**Problemstellung:** Nur wenn ein Modell eine einfache und schnelle Möglichkeit der Konfiguration, Anpassung, Simulation und Auswertung bietet, kann eine Zeiteinsparung für den Benutzer erfolgen und somit eines der Hauptziele dieser Arbeit erreicht werden. Dafür sollen Mechanismen entwickelt werden, die genau diese Zielsetzung verfolgen.

**N3: Syntheseflow zur Modellbildung**

**Beschreibung:** Das Modell bildet nur die Struktur ab, um die Leistungsaufnahme auf ESL zu simulieren und aufzuzeichnen. Damit das Modell benutzbar ist, muss es ein Synthesevorgehen geben, das strukturiert das Vorgehen beschreibt, wie das Modell mit Leistungsaufnahmewerten von GL erstellt werden kann.

**Problemstellung:** Das Problem bei der Synthese besteht in der Bereitstellung von Leistungsaufnahmewerten. Diese werden gewöhnlich auf der deutlich niedrigeren Abstraktionsebene GL bereit gestellt, da erst auf dieser Ebene die Technologiebibliotheken benutzt werden, die die Leistungsaufnahme (und weitere Parameter wie z.B. die zeitliche Verzögerung) für Gatter beschreiben. Da sich die Werte hier aber auf eine wesentlich detailliertere Beschreibung beziehen, müssen diese Werte so abstrahiert werden, dass sie in das entstehende Leistungsaufnahmemodell integriert werden können und die Abweichung zu den Ursprungswerten so gering wie möglich halten.

## 1.3. Aufbau der Arbeit

In diesem Abschnitt wurde das Thema der Arbeit eingeleitet, die Problematik beschrieben und die Zielstellung vorgestellt. Des Weiteren wurden Anforderungen an das erforderliche Modell und die Methodik abgeleitet.

Die Arbeit ist im weiteren Verlauf wie folgt untergliedert. Um einen Überblick über die Grundlagen und den Kontext der Arbeit zu erlangen, werden in Kapitel 2 die nötigen Inhalte erläutert, die in den nachfolgenden Kapitel als bekannt vorausgesetzt werden. Dabei wird zum einen auf die verschiedenen Arten der funktionalen Modellierung und deren Abstraktionsebene und zum anderen auf die Eigenheiten der Leistungsaufnahme eingegangen. Kapitel 3 gibt einen Überblick über verwandte Arbeiten und zeigt die Abgrenzung dieser Arbeit. Kapitel 4 zeigt einen Gesamtüberblick über das entwickelte Konzept und beschreibt sowie formalisiert die Methodik. Dabei werden die Einzelheiten der Methodik erläutert und ihre Notwendigkeit erklärt. Da das resultierende Modell erstellt und mit Werten gefüllt werden muss, wird in Kapitel 5 gezeigt, wie Leistungsaufnahmewerte erzeugt und diese in ein Modell überführt werden können. In Kapitel 6 wird die Methodik für verschiedene Komponentenklassen anhand von ausgewählten Beispielen evaluiert und bewertet. Kapitel 7 schließt die Arbeit ab, zeigt einen Ausblick und fasst sie kurz zusammen.

# KAPITEL 2

# Grundlagen

Wie im letzten Kapitel beschrieben, ist das Ziel dieser Arbeit die Modellierung der Leistungsaufnahme von kompletten Komponenten auf ESL sowie die Herleitung dieses Modells. Aus diesem Grund wird in diesem Kapitel auf den Entwurfsfluss von Komponenten und Systemen eingegangen sowie auf die Modellierung und die Grundlagen der Leistungsaufnahme.

## 2.1. Entwurfsfluss

Die Entwicklung einer digitalen Schaltung passiert in mehreren Entwurfsschritten auf unterschiedlichen Abstraktionsebenen. Diese werden beim Design der Systeme durchlaufen und stellen verschiedene Detailgrade der Modellierung dar. Alle Ebenen trennen dabei verschieden Aspekte wie z.B. die Software und Hardware, erlauben aber eine gesamtheitliche Modellierung und Simulation des Systems. Dies führt dazu, dass Arbeitsschritte beim Design von eingebetteten System parallelisiert werden können und nur die für den Arbeitsschritt nötigen Details betrachtet werden. Dadurch können effiziente und performante Systeme in möglichst kurzer Zeit entworfen und in Bezug auf funktionale sowie nicht-funktionale Anforderungen getestet werden.

Von einer Abstraktion wird dann gesprochen, wenn Details entfernt werden, das grundlegende Verhalten aber erhalten bleibt. Eine Abstraktion kann somit zu einer Verschlechterung der Genauigkeit führen, aber nicht zu einem grundsätzlich falschen Verhalten. Hierzu gibt es unterschiedliche Untersuchungen, die sicherstellen, dass eine Abstraktion valide ist [73].

In der Regel führt der Systementwurf über die Abstraktionsebenen ESL, RTL und GL, die nachfolgend erklärt werden.

#### Electronic System Level

Die Abstraktionsebene ESL ist ein Begriff, der verbreitet genutzt wird, aber bis heute keine eindeutige Definition bekommen hat. Aus diesem Grund stützt sich diese Arbeit auf die Definition aus dem Buch *ESL Design and Verification: A Prescription for Electronic System Level Methodology* [46]. Dieses beschreibt ESL als eine Anwendung einer angemessenen

Abstraktion, um Systemelemente so weit zusammenzufassen, dass die Wahrscheinlichkeit erhöht wird, eine anwendbare Implementierung der Funktionalität mit minimalem Kostenaufwand zu erlangen, während die Randbedingungen an das Design eingehalten werden. Sie erlaubt, eine Architektur festzulegen, das System aus Hardwaresicht zu definieren, die Funktionalität auf Hardwareblöcke abzubilden sowie parallel dazu die benötigten Softwareanteile zu entwickeln und im Zusammenhang mit dem restlichen System zu testen. Dies ermöglicht eine Evaluation und Validierung von verschiedenen Systementscheidungen.

Auf dieser Ebene geschieht eine Betrachtung des Systems komponentenweise. Der innere Aufbau wird in der Regel algorithmisch beschrieben und definiert lediglich die reine Funktionalität ohne Bezug zur konkreten Hardwareumsetzung. Es ist aber bereits entschieden, welcher Teil der Funktionalität von welchen Hardwarekomponenten implementiert werden soll.

Die Softwareausführung wird häufig in einem sogenannten *Instruction Set Simulator* (ISS) simuliert. Da dieser die Ausführung instruktionsgenau abbildet, bietet er das nötige Maß an Genauigkeit und Abstraktionsvermögen auf ESL.

Für die Kommunikation werden abstrakte Schnittstellen benutzt wie reine Funktionsaufrufe oder Busmodelle. Letztere werden häufig durch das sogenannte *Transaction Level Modelling* (TLM) dargestellt, welches die Schreib- und Lesezugriffe als einzelne Transaktionen modelliert. Je nach Entwicklungsfortschritt werden hier entweder generische Modelle eingesetzt oder speziellere Modelle, die beispielsweise das konkrete Arbitrierungsverhalten eines Busses nachbilden.

Auf ESL wird entweder ohne ein konkretes Zeitverhalten gearbeitet oder es wird ein Zeitverhalten genutzt, welches sich an die Zielzeiten annähert. Letzteres beschreibt die Zeitintervalle zwischen definierten Events. Ein Taktsignal gibt es in der Regel nicht und damit auch kein taktgenaues Verhalten. Da auf ESL viele Details fehlen, die benötigt werden, um ein exaktes Zeitverhalten abbilden zu können, ist hier mit Ungenauigkeiten zu rechnen. Diese sind aber so gering, dass sie bei der Evaluation eines Systems vernachlässigt werden können.

#### Register Transfer Level

Ist das Systemdesign fertiggestellt, wird das Hardwaredesign auf RTL fortgeführt. Neue Komponenten werden durch einen Syntheseprozess von ESL nach RTL überführt. In einigen Fällen ist zu Beginn das RTL Design vorhanden und wird dann über eine sogenannte *High-Level Synthese* nach ESL überführt.

Auf RTL wird eine Schaltung von einzelnen Gattern abstrahiert und durch logische Operationen und Register beschrieben. Das Design wird auf dieser Ebene in der Regel in den Sprachen *VHDL* oder *Verilog* beschrieben. Diese Ebene ist die erste Ebene, bei der das Design aus einem Graph aus Registern und kombinatorischen Elementen besteht und wird daher als sogenannten *Netzliste* bezeichnet. Dabei setzt sich die Beschreibung zum einen

aus dem Datenpfad und zum anderen aus der Kontrollschrittreihenfolge zusammen, die bestimmt, in welcher Abfolge der Datenpfad ausgeführt wird [70]. Diese Elemente können leicht durch Schaltungselemente auf GL ersetzt werden und erlauben daher eine schnelle Synthese und Abbildung auf die Zieltechnologie. Das Zeitverhalten wird in der Regel mit einem Taktsignal dargestellt und ermöglicht daher eine taktgenaue Analyse des Zeitverhaltens. Detaillierte Analysen wie die Bestimmung von Schaltverzögerungen ist auf RTL nicht möglich, da hierfür weitere Zeitinformationen wie die Schaltverzögerungen der einzelnen Gatter benötigt werden, die auf RTL nicht verfügbar sind. Auf dieser Ebene ist das Design bereits so detailliert beschrieben, dass die Simulation eines gesamten Systems so viel Zeit in Anspruch nimmt, dass dieses nicht mehr effizient simuliert werden kann. Aus diesem Grund werden hier noch einzelne Komponenten betrachtet, simuliert und optimiert. Eingangsdaten für eine Komponente können aus der Systemsimulation extrahiert und auf die RTL Schnittstellen abgebildet werden. Dies erlaubt die Validierung für anwendungsbezogene Fälle.

#### Gate Level

Nach der Fertigstellung des RTL Designs wird das System nach GL synthetisiert und somit auf die Zieltechnologie abgebildet.

Auf GL wird eine digitale Schaltung auf einem sehr hohen Detaillierungsgrad beschrieben. Diese besteht hier aus einzelnen logischen Gattern, die die Operationen abbilden, und Registern, die als Speicher dienen. Diese Gatter werden durch die genutzte Technologie bereitgestellt. Eine Technologie beschreibt neben den zur Verfügung stehenden Gattern auch weitere Parameter wie das Zeitverhalten oder die Leistungsaufnahme unter verschiedenen Bedingungen. Des Weiteren ist es möglich, die einzelnen Elemente direkt auf einem Chip zu platzieren und den Verlauf der Signalleitungen auf dem Chip zu bestimmen – das sogenannte *Place-and-Route*. Das Zeitverhalten wird hierdurch direkt beeinflusst und kann demnach genauer bestimmt werden. Eine Simulation auf GL gibt Aufschluss darüber, ob sich ein Design entsprechend seiner Vorgaben in Bezug auf seine Funktion, sein zeitliches Verhalten und seine Leistungsaufnahme wie erwartet verhält. Beispielsweise kann analysiert werden, wie lang der kritische Pfad ist (zeitlich längster kombinatorischer Pfad zwischen zwei Registerzellen), um daraus die maximale Taktfrequenz abzuleiten.

## 2.2. Leistungsaufnahme

Wie in der Einleitung beschrieben, werden Modelle für Leistungsaufnahme häufig auf ESL in die virtuelle Plattform integriert, da auf dieser Ebene Designentscheidungen den größten Einfluss haben. Bevor näher beschrieben wird, wie solche Modelle aussehen, werden nachfolgend die Grundlagen der Leistungsaufnahme beschrieben.

Grundlegend beschreibt der Begriff Leistung, wie viel Arbeit in einer bestimmten Zeit verrichtet wird. Bei Strom entspricht dies der elektrischen Energie, die in einer bestimmten Zeit umgesetzt wird. Die elektrische Leistung $P$ ergibt sich aus dem Produkt der elektrischen Versorgungsspannung $V_{\mathrm{dd}}$ und des elektrischen Stroms $I$:

$$P = V_{\mathrm{dd}} \cdot I \quad . \tag{2.1}$$

Um die Leistungsaufnahme zu senken, ergibt sich dadurch die triviale Lösung, einfach Strom, Spannung, oder beides zu senken. Dass das in den meisten Fällen nicht so einfach ist, zeigt sich, wenn man sich elektrische Schaltungen genauer anschaut. In den heutigen Systemen wird die meiste Energie nicht mehr in analogen Schaltungen umgesetzt, sondern in digitalen *Integrated Circuits* (ICs). Der Großteil der Komponenten auf dem IC sind Transistoren. Diese agieren wie kleine Schalter. Durch die Kombination von Transistoren können komplexe Schaltungen aufgebaut werden. In den heutigen ICs werden meist sogenannte *Metal Oxid Semiconductor Field Effect Transistors* (MOSFETs) eingesetzt. Diese steuern den Stromfluss durch das Öffnen und Schließen einer Energiebarriere zwischen *Source* und *Drain* Anschluss. Diese Energiebarriere wird über eine Spannung am *Gate* Anschluss gesteuert [58]. Wird ein öffnender N-Kanal und schließender P-Kanal MOSFET, wie in Abbildung 2.2 gezeigt, in Reihe zusammengeschaltet, bekommt man eine *Complementary Metal-Oxide Semiconductor* (CMOS)-Schaltung, die sich durch eine niedrige Leistungsaufnahme auszeichnet.

Durch den Aufbau von CMOS-Schaltungen ergeben sich unterschiedliche Arten an Strömen, die verschiedene Leistungsaufnahmen hervorrufen. Es kann grob unterschieden werden zwischen der statischen Leistungsaufnahme und der dynamischen Leistungsaufnahme. Die statische Leistungsaufnahme ist immer präsent, sobald eine Versorgungsspannung an der CMOS-Schaltung anliegt. Die dynamische Leistungsaufnahme tritt nur bei Schaltvorgängen auf.

### 2.2.1. Statische Leistungsaufnahme

Die statische Leistungsaufnahme ergibt sich hauptsächlich aus zwei Strömen, die als Leckströme bezeichnet werden. Das sind der Strom zwischen Gate und Drain und zwischen Source und Drain im MOSFET (siehe Abbildung 2.1). Diese Ströme entstehen durch den Tunneleffekt, der dafür sorgt, dass Elektronen auch durch Sperr- und Isolationsschichten fließen. Je größer die Versorgungsspannung ist, desto größer sind auch die Leckströme und somit die statische Leistungsaufnahme. Ein weiterer Faktor bei den Leckströmen ist die verwendete Technologie. Durch die immer kleiner werdenden Strukturgrößen der Transistoren sinkt auch die Dicke der Sperrschicht zwischen Kanal und Gate sowie die Kanallänge. Dadurch wird das Potential, das die Elektronen überwinden müssen, stetig kleiner, was zu immer größer werdenden Leckströmen führt. Dadurch ist neben der dynamischen Leistungsaufnahme auch die statische Leistungsaufnahme ein immer wichtigerer Faktor geworden. Negativ wirkt sich auch die Temperatur auf die Leckströme aus. Denn je höher die Temperatur ist, desto kleiner wird das zu überwindende Potential [57]. Da

die Temperatur auch abhängig von der Leistungsaufnahme ist, kann dies zu einem Dominoeffekt führen, der Temperatur und Leistungsaufnahme deutlich höher treiben kann als angenommen. Dies führt dazu, dass es nicht mehr nur ausreicht, die Leistungsaufnahme korrekt abzuschätzen, sondern auch die daraus resultierenden Temperaturen und diese beim zeitlichen Verlauf der Leistungsaufnahme mit zu betrachten und einzuberechnen.

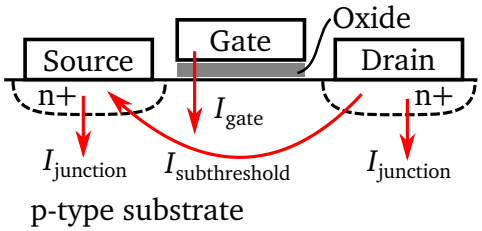



Abbildung 2.1.: Immer persistente Leckströme in einem MOSFET

### 2.2.2. Dynamische Leistungsaufnahme

Die dynamische Leistungsaufnahme tritt im Gegensatz zur statischen Leistungsaufnahme nur dann auf, wenn CMOS-Glieder schalten. Dies soll nachfolgend an einem CMOS-Inverter gezeigt werden. Ein solcher ist in Abbildung 2.2 zu sehen und besteht aus einem P-Kanal MOSFET (oben) und einem N-Kanal MOSFET (unten). Bei dem Schaltvorgang treten zwei Ströme auf: der Kurzschlussstrom $I_{sc}$ und der Schaltstrom $I_{sw}$.

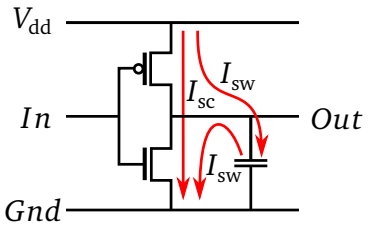



Abbildung 2.2.: Dynamische Ströme in einem CMOS-Inverter, die bei Umschaltvorgängen auftreten

Wie in der Abbildung zu sehen, sind ein P-Kanal und ein N-Kanal MOSFET so in Reihe geschaltet, dass immer einer der beiden Transistoren geschlossen ist und somit kein Stromfluss von $V_{dd}$ nach $Gnd$ ermöglicht wird. Wenn ein Transistor schaltet, benötigt er für diesen Vorgang eine gewisse Zeit. Aus diesem Grund sind bei einem Umschaltvorgang beide Transistoren für einen kurzen Augenblick $t_{sc}$ leitend und ermöglichen den sogenannten Kurzschlussstrom. Nimmt man einen linearen Verlauf bei der Strom- und Spannungsänderung an – dieses Verhalten entspricht nicht dem wirklichen Verhalten, nähert es jedoch sehr gut an [8] – ergibt sich der Kurzschlussstrom wie in Abbildung 2.3. Dort sieht man, dass sich bei etwa der Hälfte des Umschaltvorgangs die Spitze des Kurzschlussstroms befindet, da hier beide Transistoren in etwa die gleiche Leitfähigkeit besitzen. Des Weiteren wird

ersichtlich, dass erst eine gewisse Schwellspannung $V_{\text{th}}$ überschritten werden muss, damit ein Transistor leitend wird.

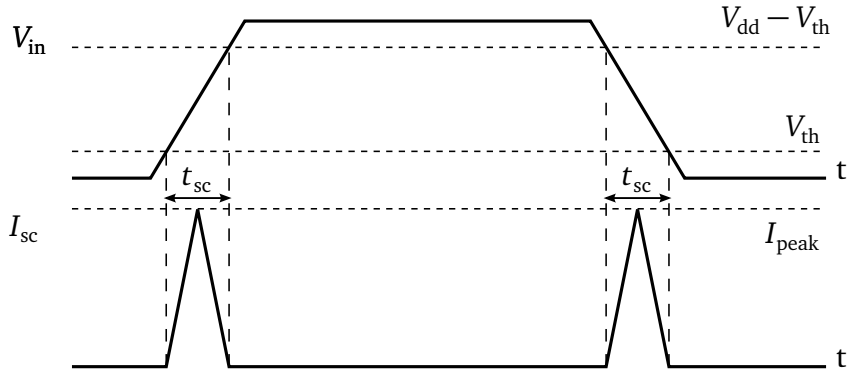



Abbildung 2.3.: Das Verhalten des Kurzschlussstroms in Abhängigkeit der Schaltspannung an den Gates der MOSFETs in einem CMOS-Inverter.

Wenn von einem linearen Verlauf des Stroms ausgegangen wird, ist der Strom bis $\frac{t_{sc}}{2}$ linear steigend bis zu einem Wert von $I_{peak}$ und von $\frac{t_{sc}}{2}$ bis $t_{sc}$ linear sinkend. Damit kann die elektrische Ladung, die durch diesen Strom verschoben wird, folgendermaßen berechnet werden:

$$\begin{aligned} Q_{sc} &= \frac{I_{peak}}{\frac{t_{sc}}{2}} \cdot \frac{t_{sc}}{2} + \left( \frac{-I_{peak}}{\frac{t_{sc}}{2}} + I_{peak} \right) \cdot \frac{t_{sc}}{2} \\ &= I_{peak} \cdot \frac{t_{sc}}{2} \quad . \end{aligned} \tag{2.2}$$

Ein Schaltvorgang kann nur bei der steuernden Flanke des Taktsignals $f$ auftauchen, da dies in den Registern die Übernahme von neuen Werten triggert und diese Änderung dadurch auf die kombinatorische Logik übertragen wird. Mit der Einberechnung der Versorgungsspannung $V_{dd}$ und der durchschnittlichen Schaltaktivität $\alpha$ ergibt sich folgende Formel zur Berechnung der Leistungsaufnahme des Kurschlussstroms:

$$P_{sc} = I_{peak} \cdot \frac{t_{sc}}{2} \cdot V_{dd} \cdot f \cdot \alpha \quad . \tag{2.3}$$

Da die Stromstärke durch die Spannung beeinflusst wird, kann die Formel nach geltenden physischen Regeln wie folgt umgeformt werden:

$$\begin{aligned} P_{sc} &= I_{peak} \cdot \frac{t_{sc}}{2} \cdot V_{dd} \cdot f \cdot \alpha \\ &= \frac{V_{dd}}{R_{sc}} \cdot \frac{t_{sc}}{2} \cdot V_{dd} \cdot f \cdot \alpha \\ &= \frac{1}{2} \cdot C_{sc} \cdot V_{dd}^2 \cdot f \cdot \alpha \quad . \end{aligned} \tag{2.4}$$

Der Parameter $C_{sc}$ beschreibt hierbei eine Ersatzkapazität für die Leistungsaufnahme des Kurzschlussstroms. Damit ergeben sich folgende Möglichkeiten, die Leistungsaufnahme des Kurzschlussstroms zu senken. Wie auch bei der statischen Leistungsaufnahme, kann die Versorgungsspannung gesenkt werden. Diese beeinflusst die Leistungsaufnahme quadratisch. Des Weiteren kann durch die Senkung der Taktfrequenz und Schaltaktivität die Leistungsaufnahme verringert werden. Die Dauer des Kurzschlussstroms sowie die Spitze des Kurzschlussstroms sind abhängig von der verwendeten Technologie sowie der Versorgungsspannung.

Wie auch der Kurzschlussstrom entsteht der Schaltstrom $I_{sw}$ beim Schaltvorgang des CMOS-Inverters. Dieser sorgt für des Laden bzw. Entladen der Ausgangskapazität $C$. Dies beinhaltet die Kapazität des Transistorausgangs, der angebundenen Leitungen und der damit verknüpften Eingangskapazitäten der nachgeschalteten CMOS-Glieder.

Die entsprechende Ladungsverschiebung ergibt sich durch die Einberechnung der Versorgungsspannung:

$$Q_{sw} = C \cdot V_{dd} \quad . \tag{2.5}$$

Bei jedem zweiten Schaltvorgang gibt es nur eine Ladungsverschiebung aus der Ausgangskapazität $C$ nach $Gnd$, wodurch hier keine weitere Ladung von der Spannungsversorgungsleitung verschoben wird. Aus diesem Grund wird dieser Schaltvorgang bei der Leistungsberechnung ignoriert und nur $\frac{f}{2}$ betrachtet. Analog zum Kurzschlussstrom (2.3) kann die daraus resultierende Leistungsaufnahme folgendermaßen berechnet werden:

$$\begin{aligned} P_{sw} &= (C_{sw} \cdot V_{dd}) \cdot V_{dd} \cdot \frac{f}{2} \cdot \alpha \\ &= \frac{1}{2} \cdot C_{sw} \cdot V_{dd}^2 \cdot f \cdot \alpha \quad . \end{aligned} \tag{2.6}$$

Auch hier beeinflusst die Versorgungsspannung die Leistung wieder quadratisch. Da sich die Versorgungsspannung quadratisch auf alle vorgestellten Leistungen auswirkt, kann durch eine Senkung eine größtmögliche Energieeinsparung erreicht werden, wobei die Funktion des CMOS-Inverters erhalten bleibt. Dass dieses Vorgehen aber nicht nur Vorteile bringt, wird in Abschnitt 2.3.2 diskutiert. Weitere Einflussfaktoren für eine Energieeinsparung sind wie bei dem Kurzschlussstrom die Schaltaktivität und die Taktfrequenz. Die Ausgangskapazität ist abhängig von der Technologie sowie von der Leitungslänge, die durch das *Place-and-Route* bestimmt wird. Dieses gibt an, wie Komponenten platziert und miteinander verbunden werden.

Die dynamische Leistungsaufnahme teilt sich wie bereits beschrieben in die durch den Kurzschlussstrom $P_{sc}$ und die durch den Schaltstrom ausgelöste Leistungsaufnahme $P_{sw}$ auf.

Die gesamte dynamische Leistungsaufnahme $P_{dyn}$ ergibt sich aus der Addition der Leistungsaufnahme des Kurzschlussstroms $P_{sc}$ und des Schaltstroms $P_{sw}$ aus (2.6). Diese kann

nun folgendermaßen berechnet werden:

$$\begin{aligned} P_{dyn} &= P_{sw} + P_{sc} \\ &= \frac{1}{2} \cdot (C_{sw} + C_{sc}) \cdot V_{dd}^2 \cdot f \cdot \alpha \\ &= \frac{1}{2} \cdot C_{dyn} \cdot V_{dd}^2 \cdot f \cdot \alpha \qquad \text{mit} \quad C_{dyn} = C_{sw} + C_{sc} \quad . \end{aligned} \tag{2.7}$$

## 2.3. Modellierung der Leistungsaufnahme

Im letzten Abschnitt wurde erklärt, wie sich die elektrische Leistungsaufnahme zusammensetzt und welche Parameter sie beeinflussen. In diesem Abschnitt wird nun erklärt, wie diese Leistungsaufnahme auf ESL modelliert werden kann, um in entsprechenden Simulationen das Leistungsaufnahmeverhalten nachzubilden.

Methoden für die Berechnung der Leistungsaufnahme lassen sich danach unterscheiden, von welchen Parameter sie das Verhalten abbilden und damit in Klassen einteilen. Diese Klassen sind Methoden zum Erfassen von

- Leckwiderstand und geschalteter Kapazität,
- Versorgungsspannung und Taktfrequenz sowie
- Schaltaktivität.

*SystemC* ist heutzutage eine der meist verwendeten Sprachen zur Modellierung von Hardware Schaltungen. Aus diesem Grund werden Modelle für die Leistungs- und Energieaufnahme häufig in dieser Sprache implementiert und an die funktionalen Modelle gebunden [12, 25, 35, 74]. Auch das in dieser Arbeit entwickelte Modell wurde in *SystemC* implementiert und für die Evalation genutzt.

### 2.3.1. Methoden zum Erfassen von Leckwiderstand und geschalteter Kapazität

Der Leckwiderstand und die geschaltete Kapazität werden maßgeblich durch die Synthese des Systems als auch durch die verwendete Technologie festgelegt. Der Leckwiderstand kann sich leicht ändern durch die Temperatur als auch durch den Schaltzustand einzelner Logikgatter.

Eine Erhöhung der Temperatur sorgt dafür, dass der Leckwiderstand leicht sinkt und damit die Leckströme steigen [58]. Die meisten Methoden betrachten Systeme immer bei gleicher Temperatur. Der Grund dafür ist, dass der Anteil der statischen Leistungsaufnahme an der gesamten Leistungsaufnahme bei neueren Technologien wie *Fully Depleted Silicon On Insulator* (FD-SOI) [68], *FinFET* [29] und *high-k* [52] gering ist und damit die Dynamik

dieses Parameters im Bereich des Abstraktionsfehlers der dynamischen Leistungsaufnahme liegt. Dadurch kann diese Abweichung in den meisten Fällen vernachlässigt werden. Einige Modelle bieten eine Schnittstelle zur Eingabe des Leckwiderstands. Dieser kann dann durch ein separates Modell während der Simulation berechnet und dem Leistungsaufnahmemodell zur Verfügung gestellt werden.

Der Leckwiderstand wird zusätzlich leicht durch den Schaltzustand einzelner Logikgatter beeinflusst. Wie in Abschnitt 6.3 gezeigt wird, ist dieser Effekt so gering, dass sein Einfluss vernachlässigt werden kann. Demnach können sowohl Leckwiderstand als auch geschaltete Kapazität als konstant angenommen werden, sobald sie durch Syntheseprozess und Technologieauswahl festgelegt wurden.

Die geschaltete Kapazität kann des Weiteren bei der Systemsynthese optimiert werden. Hier kann unter Betrachtung verschiedener Randbedingungen wie z.B. der Platzverfügbarkeit auf einem *Application-Specific Integrated Circuit* (ASIC), den verfügbaren Schaltelementen auf einem *Field Programmable Gate Array* (FPGA), der erforderlichen Schaltgeschwindigkeit oder geringem Energiebedarf bzw. geringer Leistungsaufnahme eine optimale Lösung gesucht werden, die das gewünschte funktionale Verhalten umsetzt. Heutige Synthese-Programme wie *Xilinx ISE Design Suite* und *Synopsys Design Compiler* bieten hier umfangreiche Einstellungen und bilden das Design nach den Wünschen und Anforderungen des Anwenders auf die Hardware ab.

Diese Arbeit fokussiert sich auf die dynamischen Effekte der Leistungsaufnahme zur Laufzeit. Daher wird im nachfolgenden Verlauf dieser Arbeit nicht näher auf Arbeiten eingegangen, die sich mit der Abbildung der statischen Leistungsaufnahme beschäftigen.

### 2.3.2. Methoden zum Erfassen von Versorgungsspannung und Taktfrequenz

Methoden zum Senken oder Abschalten der Versorgungsspannung sind sehr effektive Maßnahmen zur Reduzierung der Leistungsaufnahme, da sich die Versorgungsspannung zum einen auf die statische und dynamische Leistungsaufnahme auswirkt und zum anderen einen quadratischen Einfluss auf die Leistungsaufnahme hat. Da die Stromstärke abhängig ist von der Versorgungsspannung, kann man die Formel folgendermaßen umstellen, um die Versorgungsspannung als einzige abhängige Größe zu erhalten:

$$P = V_{dd} \cdot I \quad \Leftrightarrow \quad P = \frac{V_{dd}^2}{R} \quad . \tag{2.8}$$

Wird die Taktfrequenz gesenkt oder komplett ausgeschaltet wirkt sich dies linear auf die dynamische Leistungsaufnahme aus.

In dieser Arbeit wird eine Methodik entwickelt, die lediglich die Aktivität einer funktionalen Komponente auf die Leistungsaufnahme abbildet. Demnach wird auf eine Abgrenzung

gegen Arbeiten, die sich mit der Modellierung und Optimierung von Steuerungsalgorithmen für die dynamische Festlegung der Versorgungsspannung und Taktfrequenz beschäftigen, in dieser Arbeit verzichtet.

Da, wie oben dargestellt, die Spannung und Taktfrequenz zur Laufzeit dynamisch verändert werden können und einen großen Einfluss auf die Leistungsaufnahme haben, werden nachfolgend kurz die bekanntesten Mechanismen vorgestellt, die diese Parameter steuern, und wie diese modelliert werden können.

**Power Gating** Beim sogenannten *Power Gating* wird die Versorgungsspannung auf 0 V gesenkt, da hiermit die Leistungsaufnahme größtmöglich minimiert wird und die entsprechenden Systemteile sozusagen abgeschaltet [59].

Dies kann bei Systemteilen eingesetzt werden, die das System über einen längeren Zeitraum nicht benötigt. Wichtig dabei ist, zu beachten, dass das restliche System weiterhin korrekt funktioniert und wichtige Daten nicht verloren gehen. Ersteres wird erreicht, indem z.B. intelligente Busschnittstellen benutzt werden, die erkennen, wenn die angesprochene Komponente ausgeschaltet ist und entsprechende Antworten an die anfragende Komponente geben bzw. die Komponente bei Bedarf wieder anschalten. Speicher sind meistens flüchtig und nicht persistent. Sie verlieren daher ohne Versorgungsspannung die aktuellen Daten. Damit dies beim Abschalten einer Komponente nicht passiert, werden diese Speicher entweder weiterhin mit Strom versorgt oder die Daten werden in sogenannte *Retention Register* geschrieben, die die Daten während der abgeschalteten Zeit behalten. Das An- und Ausschalten von Systemteilen erfordert eine gewisse Zeit und Energie. Dadurch ist es wichtig, dass die Zeit im ausgeschalteten Zustand lang genug ist, damit die eingesparte Energie größer ist als die nötige Energie zum Ein- und Ausschalten. Der Zeitpunkt, an dem diese beide Werte gleich sind, wird *Break Even Point* genannt.

**Clock Gating** Wird eine Komponente oder ein Systemteil nicht gebraucht, muss aber bei Bedarf wieder schnell verfügbar sein, gibt es noch eine andere Methode zur Energieeinsparung. Das Taktsignal verbraucht in den meisten Systemen einen Großteil der Energie, da es einen großen Leitungsbaum besitzt, der zu allen relevanten Systemteilen führt und teilweise noch verstärkt werden muss, damit über die großen Leitungslängen ein ausreichend starkes Signal ankommt. Des Weiteren sorgt das Taktsignal in vielen angeschlossenen Komponenten für Schaltaktivität, auch wenn diese gerade keine Funktionen ausführen. Um dies zu verhindern, kann man Teile des Leitungsbaumes isolieren. Dieser Vorgang wird *Clock Gating* genannt. Dadurch werden nicht genutzte Komponenten vom Taktbaum getrennt und das Schalten von Signalen in diesen Komponenten verhindert. Zusätzlich wird die geschaltete Kapazität der Taktleitung verringert. Im Idealfall können dadurch die einzelnen Verstärker im ungenutzten Leitungsbaum deaktiviert werden, um weitere Energie zu sparen. Werden die abgeschalteten Komponenten wieder benötigt, kann direkt im nächsten Takt mit den Komponenten gearbeitet werden, da diese durch die weiterhin vorhandene Spannungsversorgung ihren aktuellen Zustand nicht verloren haben.

**DVFS** Eine weitere Maßnahme zum Senken der Leistungsaufnahme besteht in der Senkung der Versorgungsspannung. Diese wird im Gegensatz zum *Power Gating* nicht bis auf 0 V gesenkt, sondern nur so weit, dass der Teil der Schaltung noch funktionsfähig bleibt.

Bei der Spannungssenkung gibt es einen negativen Effekt. MOSFETs brauchen eine gewisse Zeit, bis das Ausgangssignal umgeschaltet ist. Diese Zeit bezeichnet man als Schaltverzögerung und ergibt sich dadurch, dass auf der Ausgangsseite des Transistors die Ausgangskapazität geladen bzw. entladen werden muss. Wird die Spannungsversorgung gesenkt, dauert der Ladevorgang dieser Kapazität länger und die Schaltverzögerung verlängert sich. Dies kann dazu führen, dass kombinatorische Logik zwischen zwei Speicherregistern so verzögert umschaltet, dass vor der nächsten positiven Taktflanke des Taktsignals nicht alle Schaltvorgänge erfolgreich durchgeführt wurden. In diesem Fall übernimmt das Register eventuell falsche Werte und die korrekte Funktion kann nicht mehr gewährleistet werden. Dieses Verhalten ist oft auf dem sogenannten *kritischen Pfad* zu sehen, der den Datenpfad mit der größten zeitlichen Verzögerung durch die Gatterlogik angibt. Um ein Fehlverhalten zu verhindern, wird zeitgleich mit der Versorgungsspannung auch die Taktfrequenz gesenkt. Dadurch steigt die Zeit zwischen zwei positiven Taktflanken und somit kann auf dem *kritischen Pfad* die Berechnung vor der nächsten positiven Taktflanke beendet werden. Diese Methode ist das sogenannten *Dynamic Voltage and Frequency Scaling* (DVFS) [24, 44]. Das Problem dabei ist, dass durch die Verringerung der Taktfrequenz die entsprechenden Systemteile auch weniger Operation in der gleichen Zeit ausführen können und dadurch die Durchsatzrate sinkt. Aus diesem Grund gibt es beim DVFS in der Regel mehrere Modi, die verschiedene Paare von Spannungsversorgung und Taktfrequenz angeben. Zur Ausführungszeit wird der Modus so gewählt, dass die Systemteile die Berechnungen in der benötigten Zeit durchführen, aber die Spannung so niedrig wie möglich ist. Dafür gibt es meistens eine oder mehrere Steuereinheiten, die den aktuellen Funktionszustand des Systems überwachen und entsprechend die Modi der Systemteile ändern.

**Modellierung** Beim Modellieren dieser Techniken liegt der Fokus darauf, die Mechanismen möglichst effizient einzusetzen und damit eine größtmögliche Einsparung zu erhalten bei gleichzeitigem Erhalt der Funktionalität [12, 59]. Die Schwierigkeit liegt hierbei zu erkennen, wann für welche Systemteile eine Umschaltung der Versorgungsspannung oder Taktfrequenz durchgeführt werden kann. Denn ist eine Komponente zu langsam oder ausgeschaltet, können eventuell angeforderte Informationen zu spät oder sogar gar nicht zur Verfügung gestellt werden. Dies führt dann zu einer falschen Funktion und fehlerhaften Verhalten des Systems [66, 51].

Eine weitere Herausforderung ist zu erkennen, ob sich ein Umschaltvorgang rentiert oder die Umschaltkosten höher sind als die Einsparung [47, 48].

### 2.3.3. Methoden zum Erfassen von Schaltaktivität

Wie in Abschnitt 2.2.2 beschrieben, hat die Schaltaktivität einen linearen Einfluss auf die Leistungsaufnahme. Daher ist es wichtig, diesen Faktor mit zu betrachten. Beim Modellieren dieses Parameters sind zwei Faktoren entscheidend:

1. das Modell soll möglichst wenig zusätzlichen Aufwand erzeugen, damit eine schnelle Berechnung möglich ist und
2. das Modell soll so genau wie möglich sein, damit korrekte Entscheidungen für eine Systemlösung getroffen werden können.

Das Ziel hierbei ist, neben den funktionalen Anforderungen an das System auch die Anforderungen an die Leistungsaufnahme zu erfüllen. Folgende Aspekte werden hier in der Regel betrachtet:

- eine minimale Energie- bzw. Leistungsaufnahme und
- das Vermeiden von Leistungsaufnahmespitzen.

Die Auswahl von Hardware-Komponenten geschieht in der Regel aus einer Reihe von bestehenden Komponenten. Diese sind entweder zugekauft oder bereits vorentwickelt. In seltenen Fällen werden Komponenten eigenständig neu entwickelt, wenn keine verfügbare Komponente die Anforderungen hinreichend erfüllen kann.

**Analytische Methoden** Leistungsaufnahmewerte können mit Hilfe von Technologiewerten, strukturellen Informationen des Systems und Erfahrungswerten der Systemdynamik statisch berechnet werden. Hierdurch ist eine schnelle Abschätzung früh im Designprozess möglich [49]. Da die dynamische Leistungsaufnahme stark von der Benutzung abhängt, führt diese Abschätzung zu hohen Ungenauigkeiten.

Eine weitere Möglichkeit zur analytischen Auswertung ist die Nutzung von Zustandsautomaten für jede Komponente. Hiermit lässt sich die maximale und minimale Leistungsaufnahme von einem gesamten System analysieren und es können unerwünschte Spitzen identifiziert werden [7]. Des Weiteren können Abfolgen aus einer beispielhaften Ausführung des funktionalen Systems eingespeist werden, um eine sogenannte symbolische Simulation durchzuführen. Da hier aber eine funktionsfähiges System benötigt wird, ähnelt diese Art der Modellierung sehr den dynamischen Modellen.

**Zählerbasierte Methoden** Der einfachste Ansatz von simulationsbasierten Methoden für die Aktivitätsabschätzung besteht darin, bestimmte Aktivitäten im funktionalen Design zu zählen. Dies können z.B. die Anzahl von Funktionsaufrufen oder die Anzahl von Kommunikationsaufrufen sein. Jedem Aufruf wird eine bestimmte Menge an aufgenommener Energie zugeordnet.

Durch die einfache Zuordnung von Events zu definierten Energien können die entsprechenden Events in der Simulationsumgebung einfach und schnell integriert werden. Aus diesem Grund kann diese Modellierung früh im Designprozess eingesetzt werden [75]. Da auf ESL in der Regel keine Daten über die Leistungsaufnahme zur Verfügung stehen, müssen die Werte erst von niedrigeren Abstraktionsebenen auf die funktionalen Events auf ESL abgebildet werden oder die Abfolge dieser Events muss auf niedrigeren Abstraktionsebenen genutzt werden, um die Leistungsaufnahme zu berechnen [2]. Letzteres Vorgehen verbessert die Genauigkeit, sorgt aber für einen erheblichen Mehraufwand für die Transformation und die Berechnung der Leistungsaufnahme.

Ein weiterer Vorteil ist, dass diese Art der Modellierung auch ohne Zeitinformationen möglich ist, da direkt eine aufgenommene Energie ausgegeben wird.

Komponenten haben oft einen Grundumsatz, der durch das Taktsignal hervorgerufen wird. Der Nachteil dieser Methode ist, dass sie diese Leistungsaufnahme nicht abbilden kann. Des Weiteren sind einige auslösende Events interne Events, die gerade bei *Black-Box*-Komponenten nicht sichtbar sind und damit nicht modelliert werden können. Diese Faktoren können zu einer hohen Abweichung führen.

**Zustandsbasierte Methoden** Die zustandsbasierten Ansätze gehen davon aus, dass sich die Leistungsaufnahme einem Zustand zuordnen lässt. Übergänge zwischen diesen Zuständen werden durch funktionale Events ausgelöst, wie sie auch bei den aktivitätsbasierten Methoden angewendet werden. Dabei wird der funktionale Zustand entweder direkt mit einem Leistungsaufnahmewert verknüpft [62] oder funktionale Events werden als Eingangssignale für einen separaten Zustandsautomaten genutzt. Hier stellt jeder Zustand eine spezifische Leistungsaufnahme dar. Um einmalig aufgenommene Energie bei einem Zustandsübergang abzubilden, kann dieser mit Kosten in Form von Energieaufnahme und Zeit annotiert werden [6]. Da Leistungsaufnahme nur in Verbindung mit einem Zeitintervall in eine Energieaufnahme umgewandelt werden kann, müssen für diese Art von Modellen die funktionalen Modelle mit Zeitinformationen instrumentiert sein.

In der Regel wird nicht ein Zustandsautomat für das gesamte System erstellt, sondern jeweils ein Zustandsautomat pro Komponente. Damit können mit dem Austausch von Komponenten auch die dazugehörigen Zustandsautomaten für die Leistungsaufnahme ausgetauscht werden [6].

### 2.3.4. Modellerstellung

Die zuvor beschrieben Modelle müssen auf geeignete Weise erstellt und mit Daten befüllt werden. Besonders letzteres ist herausfordernd, da Leistungsaufnahmewerte im Entwurfsprozess frühestens auf GL zur Verfügung stehen.

Leistungsaufnahmedaten können entweder aus Simulationen auf GL gewonnen werden oder aus Messungen auf einer reellen Hardware [54]. Bei den GL-Simulationen werden

die Schaltaktivitäten aufgezeichnet und in Werkzeuge zur Leistungsaufnahmeberechnung eingespeist. Viele der Werkzeuge können auch mit Schaltaktivitäten aus RTL-Simulationen befüllt werden. Dies hat den Vorteil, dass Simulationen auf RTL deutlich weniger Zeit benötigen, kann aber zu Abweichungen führen.

Auf GL gibt es eine direkte Abbildung der Schaltung auf die Hardware. Eine Bibliothek liefert für spezifische Schaltungszustände und die Übergänge zwischen diesen die relevanten Leistungsaufnahmeinformationen. Diese Informationen stammen in der Regel vom Hersteller der Technologie. Die darin enthaltenen Werte sind meistens unter definierten Randbedingungen wie einer festen Temperatur, Spannung und Prozessvariation aufgenommen. Oft gibt es für die verschiedenen Prozessvariationen oder Spannungen einzelne Bibliotheken. Der Vorteil von Simulationen auf GL ist, dass sie schnell und einfach einsetzbar sind. Des Weiteren kann eine Simulation durchgeführt werden, ohne dass das sogenannte *Place-and-Route* durchgeführt wurde, bei dem die Elemente auf der Hardware platziert und die Signalleitung verlegt werden. Hierbei wählt man Durchschnittswerte für die Signallängen auf Basis der Anzahl von Gattern auf dem Chip.

Steht die zu modellierende Hardware bereits zur Verfügung, kann diese für die Erfassung der Leistungsaufnahmewerte genutzt werden [67]. Hierbei entstehen einige Schwierigkeiten. Bei vielen Schaltungen ist es schwer, die Leistungsaufnahme für konkrete Komponenten zu messen, da die direkten Versorgungsleitungen nicht zugreifbar sind oder unterbrochen werden können, um die Stromstärke zu messen. Jene Versorgungsleitungen, an denen eine Messung möglich ist, sitzen oft vor Spannungskonvertern oder Netzteilen. Zum einen haben diese eine eigene Energieaufnahme und zum anderen beeinflussen sie durch bestehende Kapazitäten und Induktivitäten die Stromaufnahme. Dies erschwert die ohnehin schwierige Zuordnung von der Leistungsaufnahme auf die ausgeführten Funktionen.

Hat man die Leistungsaufnahme aus einer echten Messung oder Simulation, können mit diesen Werten durch geeignete Methoden die Leistungsaufnahme-Modelle erstellt werden.

## 2.4. Abstraktion der Leistungsaufnahme

Im letzten Abschnitt wurden mehrere Methoden zur Modellierung der Leistungsaufnahme auf ESL und zur Modellerstellung vorgestellt. Diese Arbeit befasst sich mit der zustandsbasierten Modellierung der Schaltaktivität. Leistungsaufnahmewerte werden im Designprozess erst auf GL taktgenau zur Verfügung gestellt. Diese Werte müssen in geeigneter Weise abstrahiert werden, damit sie in zustandsbasierte ESL-Modelle überführt werden können, die durch funktionale Events gesteuert werden.

Die in Abschnitt 2.2 gezeigten Formeln zur Berechnung der dynamischen und statischen Leistungsaufnahme sind jeweils nur auf ein CMOS-Gatter bzw. auf einen MOSFET bezogen. Diese Werte stellen die Leistungsaufnahmewerte auf Basis eines Taktsignals dar. Auf ESL ist kein Taktsignal vorhanden, sondern nur noch eine eventbasierte Simulation, bei der

nur noch mit Zeitintervallen gearbeitet wird und nicht mit einer Anzahl von einzelnen Takten. Damit eine Leistungsaufnahme auf ESL modelliert werden kann, wird nachfolgend die Abstraktion der Leistungsaufnahme von GL nach ESL gezeigt. Da die ESL-Simulation lediglich auf Events und ganzen Komponenten basiert, ist die taktgenaue Betrachtung von CMOS-Gattern als Grundlage vollkommen ausreichend, um die im Ziel gewünschte Genauigkeit zu erreichen.

Auf Systemebene werden viele CMOS-Gatter zu komplexen Komponenten zusammengefasst. Das bedeutet, dass auch die Kapazität und Schaltaktivität angepasst werden müssen. Dafür wird die Gesamtkapazität aller $m$ CMOS-Gatter zusammengefasst und der Durchschnitt aller Schaltungsaktivitäten berechnet, wobei diese nach den Größen der Kapazitäten gewichtet werden:

$$C'_{dyn} = \sum_{i=1}^{m} C_{dyn,i} \quad \text{und} \tag{2.9}$$

$$\alpha' = \frac{1}{C'_{dyn}} \sum_{i=1}^{m} C_{dyn,i} \cdot \alpha_i \quad . \tag{2.10}$$

Dadurch ergibt sich die dynamische Leistungsaufnahme für eine komplette Komponente folgendermaßen:

$$P'_{dyn} = \frac{1}{2} \cdot C'_{dyn} \cdot V_{dd}^2 \cdot f \cdot \alpha' \quad . \tag{2.11}$$

Da $\alpha$ aus (2.7) die durchschnittliche Schaltaktivität für ein CMOS-Gatter beschreibt, gibt $\alpha'$ hier die durchschnittliche gewichtete Schaltaktivität für eine komplette Komponente an. Das Ziel dieser Arbeit ist ein zeitabhängiges Modell. Aus diesem Grund wird die Leistungsaufnahme und die Schaltaktivität nachfolgend als Funktion der Zeit betrachtet:

$$P'_{dyn}(t) = \frac{1}{2} \cdot C'_{dyn} \cdot V_{dd}^2 \cdot f \cdot \alpha'(t) \quad . \tag{2.12}$$

Demnach beschreibt $\alpha'(t)$ die durchschnittliche Schaltaktivität der Komponente und $P'_{dyn}(t)$ die dynamische Leistungsaufnahme zum Zeitpunkt $t$.

Funktionale Modelle auf ESL und die entsprechenden Modelle für die Leistungsaufnahme können ein so genaues Verhalten der Schaltaktivität nicht darstellen, da hierfür ein taktgenaues Zeitsignal notwendig wäre. Wie bereits am Anfang dieses Kapitels beschrieben, ist auf ESL nur eine eventbasierte Simulation umgesetzt, bei der sich der Zustand bei einem Event ändert. Zwischen zwei Events liegt immer genau ein definierter Zustand an. Zeiten zwischen den Events werden über Zeitintervalle ausgedrückt. Aus diesem Grund müssen sich auch Modelle für die Leistungsaufnahme an diesen Grenzen orientieren und ihre Eigenschaften auf die entsprechenden Simulationszustände des Systems abbilden. Diese Zustände sollen im Folgenden in der Menge $S$ abgebildet werden. Alle möglichen durchschnittlichen Schaltaktivitäten werden durch die Menge $A$ ausgedrückt und die Zeit durch die Menge $T$. Es gibt nun eine Funktion $\varphi : T \rightarrow S$, die jedem diskreten Zeitpunkt

einen spezifischen Systemzustand zuordnet. Des Weiteren gibt es eine Funktion $\overline{\alpha} : S \rightarrow A$ die jedem Zustand eine Schaltaktivität zuordnet. Diese Schaltaktivitäten ist der Durchschnitt aller Schaltaktivität aus dem Zeitraum, zu dem das System in dem entsprechenden Systemzustand war. Wenn $t_i$ alle diskreten Zeitpunkte angibt, in denen der Zustand $s_i \in S$ aktiv ist, so gilt für jeden Zustand:

$$\overline{\alpha}(s_i) = \frac{1}{n_i} \sum_{j=1}^{n_i} \alpha'(t_i(j)) \quad , \tag{2.13}$$

wobei $n_i$ die Anzahl aller Zeitpunkte in $t_i$ angibt. Dadurch kann nun abhängig vom aktuellen Systemzustand – der abhängig von der aktuellen Zeit ist – die durchschnittliche Schaltaktivität abgeleitet werden. Daraus ergibt sich die durchschnittliche Leistungsaufnahme $\overline{P}_{dyn}$ in Abhängigkeit des aktuellen Systemzustands:

$$\overline{P}_{dyn}(t) = \frac{1}{2} \cdot C'_{dyn} \cdot V_{dd}^2 \cdot f \cdot \overline{\alpha}(\varphi(t)) \quad , \tag{2.14}$$

wobei

$$\sum_{t=0}^{o} \overline{P}_{dyn}(t) = \sum_{t=0}^{o} P'_{dyn}(t) \quad \text{mit} \quad o = |T| \quad . \tag{2.15}$$

Ein Beispiel dieser Abstraktion ist in Abbildung 2.4 gezeigt. Hier wird durch die Abstraktion eine Abweichung für die einzelnen Zeitpunkte herbeiführt, die umgesetzte Energie und die durchschnittliche Leistungsaufnahme sind aber identisch, wie in (2.15) gezeigt.

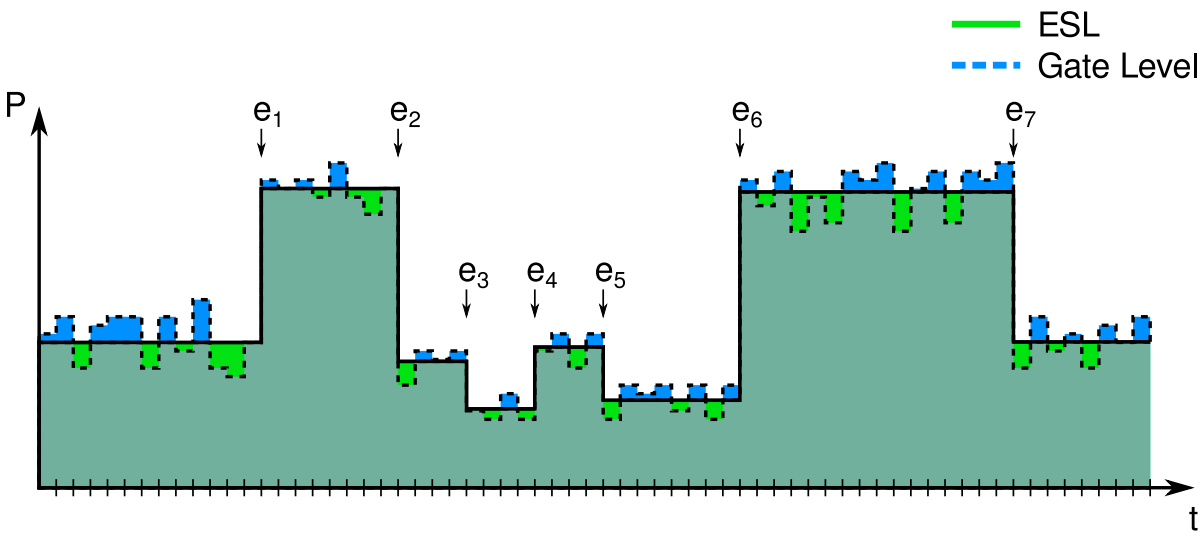



Abbildung 2.4.: Vergleich von einem Leistungsaufnahme-Trace auf ESL und auf GL. Es ist zu sehen, dass Änderungen beim GL-Trace pro Takt auftauchen, beim ESL-Trace aber nur bei Events ($e_1 \ldots e_7$). Die aufgenommene Energie ist bei beiden Traces gleich.

In einigen Fällen reicht es nicht aus, nur die durchschnittliche Leistungsaufnahme für

einen Systemzustand zu bestimmen. Beispiele hierfür sind die Bestimmung der maximalen Leistungsaufnahme eines Systems oder die Variation der Leistungsaufnahme. Hierfür können diese Parameter aus dem GL-Trace extrahiert und anhand der Intervalle an die entsprechenden Zustände annotiert werden.

Wie Helms [28] beschreibt, gibt es auch bei der statischen Leistungsaufnahme Datenabhängigkeiten. Über die große Anzahl an CMOS-Schaltungen gleichen sich diese Unterschiede aber aus [58], wodurch es legitim ist, für alle Glieder einen Durchschnittswert für die Leistungsaufnahme anzugeben:

$$\overline{P}_{stat} \approx \frac{1}{p} \sum_{i=1}^{p} P_{stat,i} \quad , \tag{2.16}$$

wobei $p$ die Anzahl der CMOS-Glieder ist. Des Weiteren ist in den meisten Fällen der Anteil und der Variationsbereich der statischen Leistungsaufnahme deutlich geringer als der der dynamischen Leistungsaufnahme. Dies wird in Abschnitt 6.3 anhand der Evaluationsbeispiele belegt.

## 2.5. Zusammenfassung

In diesem Kapitel wurde der Entwurfsfluss von digitalen Schaltungen vorgestellt und es wurden die Vorgänge erklärt, die auf den Abstraktionsebenen ESL, RTL und GL durchgeführt werden. Es wurde gezeigt, wie sich die Modellierung der Leistungsaufnahme in diesen Entwurfsfluss integriert und es wurde erläutert, dass sich die Leistungsaufnahme aus einer dynamischen und statischen Komponente zusammensetzt. Für den dynamischen Teil wurde vorgestellt, dass es sowohl Methoden zur Modellierung für die Parameter Spannung und Frequenz als auch Methoden zur Modellierung der Schaltaktivität gibt und wie diese erstellt werden können. Am Schluss wurde beschrieben, wie die Leistungsaufnahme von der GL-Ebene so abstrahiert werden kann, dass sie von den ESL-Modelle nutzbar wird. Im nächsten Kapitel werden einige Herausforderungen dieser Modellierungen angesprochen und vorgestellt, wie diese und andere Arbeiten diese Herausforderungen lösen.

# KAPITEL 3

# Verwandte Arbeiten

Im letzten Kapitel wurden verschiedene Methoden vorgestellt, wie die Leistungsaufnahme modelliert werden kann. Diese Arbeit konzentriert sich auf Methoden zur zustandsbasierten Erfassung der Schaltaktivität, die die dynamische Leistungsaufnahme linear beeinflusst. In diesem Kapitel werden verwandte Arbeiten in diesem Bereich vorgestellt, welche Herausforderungen sich in diesem Bereich ergeben und wie die vorgestellten Arbeiten mit diesen umgehen.

Die Herausforderungen lassen sich wie folgt unterteilen:

- Kompromiss zwischen Genauigkeit und Simulationsgeschwindigkeit,
- Umgang mit Datenabhängigkeiten,
- mit Black-Box-IP-Komponenten,
- mit aktive Komponenten und
- mit Signalleitungen,
- Einfluss von Spannung, Frequenz und Temperatur sowie
- die Modellerstellung.

## 3.1. Kompromiss zwischen Genauigkeit und Simulationsgeschwindigkeit

Wie bereits erwähnt, ist ein Modell für die Leistungsaufnahme meist ein Kompromiss zwischen Genauigkeit und Simulationsgeschwindigkeit. In der Regel sinkt die Simulationsgeschwindigkeit, wenn die Genauigkeit erhöht wird und umgekehrt.

Aus diesem Grund hat Bansal [5] ein Modell entwickelt, welches zur Simulationszeit die Genauigkeit des Modells umstellt, um den Mehraufwand zu verringern. Dabei wird die Genauigkeit genau an den Stellen vermindert, an denen sie keinen großen Einfluss hat bzw. die Aktivität gering ist. Diese Umschaltung geschieht durch unterschiedlich komplexe

Berechnungen. Dies führt zu einer deutlich erhöhten Simulationsperformanz bei vernachlässigbar erhöhter Ungenauigkeit. Diese Umschaltlogik erfordert mehr Aufwand und Wissen bei der Modellierung und kann dadurch potentiell zu Fehlern und größerer Ungenauigkeit führen.

Im Gegensatz dazu setzt diese Arbeit auf ein generischeres Modell, welches eine einfachere Erstellung ermöglicht. Wie in Abschnitt 6.4 zu sehen, ist der Mehraufwand des in dieser Arbeit entwickelten Modells bereits sehr gering bei gleichzeitig minimaler Abweichung, wodurch eine Umstellung der Genauigkeit unnötig ist.

## 3.2. Datenabhängigkeiten

Die Leistungsaufnahme kann stark von Datenwerten oder Datenänderungen beeinflusst werden, was häufig bei Bus- und Speicherkomponenten zu beobachten ist. Dies liegt daran, dass diese Komponenten in der Regel nur einen kleinen Teil mit Kontrolllogik besitzen und der Großteil der Leistungsaufnahme in Registern, Speicherzellen und internen Signalleitungen aufgenommen wird. Diese Leistungsaufnahme weißt eine hohe Korrelation zu den übertragenen Daten und Datenänderungen auf.

Um diese Art der Leistungsaufnahme zu modellieren, werden in der Regel die schaltenden Signale zwischen zwei aufeinanderfolgenden Datenvektoren – die *Hamming Distance* (HD) – betrachtet und linear mit einer Leistungsaufnahme korreliert. Der durch den Kontrollfluss dominierte Teil wird hier durch einen zusätzlichen Konstantanteil dargestellt.

Diese Methode findet bisher lediglich auf niedrigen Abstraktionsebenen (z.B. RTL) Anwendung und führt auf Systemebene bei Komponenten mit einer stark datenabhängigen Leistungsaufnahme zu großen Ungenauigkeiten [42].

Gag et al. [22] hat für die Signalleitungen ein Modell entwickelt, welches für die Kommunikation die Aktivitäten auf den Signalleitungen analysiert und die Durchschnittswerte mit Hilfe eines physikalischen Berechnungsmodells in Leistungsaufnahmewerte überführt. Dabei werden Faktoren wie die Kuppel- und Massekapazitäten betrachtet. Diese Berechnung wird statisch vor der Simulation durchgeführt und beschleunigt dadurch die Gesamtsystemsimulation. Auch einige andere Arbeiten setzen darauf, bestimmte Aktivitäten in der Kommunikation anzunehmen und damit die resultierende datenabhängige Leistungsaufnahme zu berechnen [39, 38]. Der Nachteil dabei ist, dass jedes Abweichen der Aktivitäten auf den Signalleitungen zu einem Fehler der Leistungsaufnahme führt. Es kann auch nur auf Signal- und Busleitungen und nicht auf Komponenten angewendet werden. Das in dieser Arbeit entwickelte Modell hingegen kann auf ESL die datenabhängige Leistungsaufnahme von passiven Komponenten vollumfänglich und abhängig von den Simulationsdaten darstellen.

Ein weiterer Ansatz für die Betrachtung der datenabhängigen Leistungsaufnahme wurde von Lee et al.[42] vorgestellt. Dieser stellt ein Modell bereit, welches auf Basis von Eingangs- und Ausgangsdaten einer Bustransaktionen die datenabhängige Leistungsaufnahme darstellt. Das Modell nutzt die Kontrollflusssignale, um einzelne Zustände zu erkennen, die einzeln für die datenabhängige Leistungsaufnahme charakterisiert werden, um ein genaueres Modell zu bekommen. Für die Charakterisierung wird eine Simulation auf TLM und GL mit identische Eingangsdaten durchgeführt. Aus der Simulation auf GL werden die Leistungsaufnahmewerte berechnet, die auf die Eingangsdaten auf TLM abgebildet werden. Dieses Vorgehen entspricht weitestgehend dem Vorgehen aus dieser Arbeit. Diese Arbeit ermöglicht aber eine deutlich bessere Darstellung des funktionalen Verhaltens durch zeitgesteuerte Zustandsübergänge und Zustandsvariablen und ermöglicht damit eine genauere Darstellung der dynamischen Leistungsaufnahme.

## 3.3. Invasive und nichtinvasive Ansätze

Wie in den vorherigen Abschnitten beschrieben, sind für eine Ableitung der Aktivität oder des Zustands im funktionalen Design ein oder mehrere Messpunkte nötig, an denen die relevanten Informationen abgegriffen werden. Dabei werden zwei Arten unterschieden:

1. invasive Modelle und
2. nichtinvasive Modelle.

**Invasiv** bedeutet, dass hier eine Änderung oder Erweiterung des funktionalen Designs nötig ist, um den Messpunkt zu integrieren. Dabei werden entweder konkret Energie- oder Leistungsaufnahmewerte an das funktionalen Modell annotiert, die im Leistungsaufnahmemodell aufgenommen und verarbeitet werden, oder Triggerpunkte gesetzt, die ein Event an das Leistungsaufnahmemodell senden, woraufhin dieses die Leistungsaufnahme oder Energie aus den gegeben Informationen berechnet [12, 17, 35, 41, 62]. Eine weitere Möglichkeit ist die Beobachtung der von internen Ereignissen wie Eingaben und Ausgaben durch das Ersetzen von Ports, die die empfangenen Daten an das Leistungsaufnahmemodell weiterleiten [74].

Der Vorteil dieser Methode ist, dass es eine direkte Kopplung zwischen funktionalem Modell und Leistungsaufnahmemodell gibt. Damit wird die größtmögliche Genauigkeit erreicht. Die Nachteile sind zum einen, dass ein solches Modell einen hohen Integrationsaufwand besitzt, da funktionales Modell und Leistungsaufnahmemodell miteinander verwoben werden, und zum anderen dass diese Modelle nur dann angewendet werden können, wenn eine Änderung des funktionalen Modells möglich ist. Dies wird speziell dann ein Problem, wenn *Black-Box*-IP-Komponenten von Drittanbietern benutzt werden. Bei diesen sind in der Regel nur die Schnittstellen aber nicht der innere Aufbau der Komponente bekannt, da dieser nicht für die Benutzung relevant ist und vor der Öffentlichkeit geschützt werden soll. Das führt einerseits dazu, dass der Quellcode nicht annotiert werden kann – weder mit Leistungseigenschaften noch mit Triggerpunkten für die das Leistungsaufnahmemodell – und anderseits

dazu, dass keine Zuordnung von funktionalem Verhalten zum Leistungsaufnahmeverhalten möglich ist.

Gelöst werden kann dies durch **nichtinvasive** Modelle, die Informationen nutzen, die das funktionale Modell für die Interaktion mit seiner Umgebung bereitstellt. Dies sind z.B.:

- Eingangs- und Ausgangsdaten,
- Zugriffe aus der Komponente auf Umgebungsfunktionen und -variablen,
- Reaktionen auf Events oder
- öffentlich einsehbare interne Informationen wie Variablenwerte.

Das Abgreifen der Daten erfolgt, indem die entsprechenden Datenquellen beobachtet werden oder mittels eines Adapters, der an den Kommunikationswegen eingeschleust wird [5, 42, 54, 64, 72]. Modelle unterscheiden sich dadurch, ob sie nur die Daten auf nach außen sichtbaren Schnittstellen beobachten oder auch interne Variablen [2]. Letztere können durch den erhöhten Detailgrad an Informationen genauer sein, aber nicht in Verbindung mit *Black-Box*-Komponenten eingesetzt werden, da hier nur die äußeren Schnittstellen sichtbar sind.

Nichtinvasive Modelle sind separate Modelle, die das Verhalten der Leistungsaufnahme beschreiben ohne dieses vom funktionalen Verhalten abzuleiten. Da sie lediglich auf externe Signale vom funktionalen Modell reagieren, bilden sie nur eine Untermenge des funktionalen Verhaltens ab und verlieren dadurch beträchtlich an Genauigkeit. Um dies zu umgehen, müssen die funktionalen Anteile, die einen Einfluss auf die Leistungsaufnahme haben, im Leistungsaufnahmemodell nachgebildet werden. Einige Arbeiten bilden dafür einen Teil des funktionalen Modells als Zustandsautomat ab, wie es auch in dieser Arbeit umgesetzt wird [64, 72].

Keine der verwandten Arbeiten betrachtet dabei die Zustandswechsel, die nicht durch ein externes Event sichtbar gemacht werden. Ein solcher Zustandswechsel passiert häufig bei passiven Komponenten, die die Berechnung der Eingangswerte beendet haben und nun auf eine Abfrage dieser Werte warten. Diese passieren oft nach einer deterministischen Zeitverzögerung. Die in dieser Arbeit entwickelte Methodik ermöglicht die Darstellung genau dieser Zustandsübergänge mittels sogenannter zeitgesteuerter Zustandsübergänge, siehe Abschnitt 4.7.

## 3.4. Prozessoren und aktive Komponenten

Auch für *Black-Box*-Prozessoren oder andere aktive Komponenten können Leistungsaufnahmemodelle entwickelt werden, sobald die nötigen Zugriffe auf relevante Ressourcen sichtbar sind, die über das innere funktionale Verhalten Aufschluss geben. Bei Prozessoren

ist dies neben dem Zugriff auf die Speicher und andere Hardwarekomponente wie Hardwarebeschleuniger oder Eingabe-/Ausgabekomponenten auch die Zugriff auf den Daten- und Instruktions-Cache [63].

Prozessoren und aktive Komponenten wie *Direct Memory Accesss* (DMAs) Controller bieten andere Herausforderungen als passive Komponenten. Passive Komponenten initiieren in der Regel selbstständig keine Berechnung, sondern reagieren auf Anfragen von aktiven Komponenten. Damit hängt ihre Funktion und die damit verbundene Leistungsaufnahme stark von den Eingaben ab. Durch diese Abhängigkeit ist dieses Verhalten – je nach Komplexität des Designs – besser vorhersagbar.

Aktive Komponenten hingegen steuern das System und sind weniger abhängig von ihren Eingangsdaten. In vielen Fällen kommen noch weitere Mechanismen hinzu wie Caches, die Daten weiter vorhalten oder potentielle Zugriffe vorziehen, um die Zugriffsgeschwindigkeit zu erhöhen. Dadurch werden diese Zugriffe nicht extern sichtbar. Das führt zu einer deutlich erschwerten Vorhersage und Ableitung des Verhaltens sowie der damit verbundenen Leistungsaufnahme. Einige Arbeiten versuchen genau dieses Verhalten in ihren Modellen abzubilden und in Bezug auf die Leistungsaufnahme zu charakterisieren [43, 67].

Aus diesen Gründen ist es so gut wie unmöglich, ein generisches Modell zu entwickeln, welches alle aktiven Komponenten modellieren kann. Die existierenden Arbeiten fokussieren sich in der Regel auf bestimmte aktive Komponenten wie z.B. einen speziellen Prozessortyp [19, 43, 54, 63]. Damit werden die Freiheitsgrade dramatisch eingeschränkt und die Modellbildung vereinfacht. Einige Arbeiten modellieren lediglich ein Zustandsautomaten mit wenigen Zuständen [43] oder bilden die einzelnen Instruktionen auf verschiedene Leistungsaufnahmewerte ab [67, 19].

Diese Arbeit bietet ein generisches und unspezifisches Modell, welches die spezifischen funktionalen Eigenschaften nur schwer nachbilden kann. Dadurch ist es für Prozessoren und aktive Komponenten ungeeignet und es sollten spezifische Modelle wie oben beschrieben eingesetzt werden.

## 3.5. Signalleitungen

Neben den funktionalen Komponenten haben auch die Signalleitungen einen großen Einfluss auf die Leistungsaufnahme. Hier wird die Leistung zum einen durch funktionale Komponenten umgesetzt, die den Bus betreiben, und zum anderen durch die Signalleitungen selbst. Zu den funktionalen Komponenten gehören die Bustreiber, die verschiedene Kommunikationsdaten in Busprotokolle übersetzen sowie *Decoder* zur Signalverteilung und *Arbiter* für die Steuerung, welcher Busmaster Zugriff auf den Bus erhält. Die funktionalen Komponenten können ähnlich wie passive Komponenten modelliert werden. Die Schwierigkeit liegt bei den Signalleitungen.

Signalleitungen agieren durch ihre hohe Kapazität als Kondensatoren zwischen Signalleitung und Substrat als auch zwischen den benachbarten Signalleitungen, die sogenannte Kopplungskapazität. Diese Leistungsaufnahme unterscheidet sich dadurch deutlich von der Leistungsaufnahme durch CMOS-Gatter und erfordert dadurch auch eine andere Modellierung. Das Laden und Entladen dieser Kapazitäten verursacht eine nicht unerhebliche Leistungsaufnahme. Diese hängt stark von der Benutzung, Länge und Verlegung der Signalleitungen ab. Die Parameter Länge und Verlegung werden durch das Gesamtsystem festgelegt und können daher nicht wie bei Komponenten einzeln charakterisiert werden. Da auf Systemebene gerade diese Parameter häufig nicht bekannt sind, müssen diese basierend auf verschiedenen Faktoren abgeschätzt und modelliert werden. Dies kann beispielsweise anhand der Komplexität der Schaltung und der Art der Platzierung abgeschätzt werden [18, 69]. Zusätzlich müssen bei vielen Schaltungen Komponenten für die Signalübertragung wie Signalverstärker betrachtet und optimiert werden [33].

Wie auch bei funktionalen Komponenten kann die Bus-Benutzung statisch abgeschätzt oder simuliert werden. Hier kommen die gleichen Vor- und Nachteile zum Tragen. Bei einer statischen Analyse werden die übertragenen Daten aus Testbeispielen charakterisiert. Das bedeutet, es wird klassifiziert, wie die Umschaltcharakteristik auf den einzelnen Signalen und Bussen aussieht. Für diese kann dann ein mittlerer Wert abgeschätzt werden. Zusammen mit weiteren Parametern wie erwartete Chipgröße und Art der Kommunikation (welches Busprotokoll wird benutzt oder ist eine Arbitrierung notwendig) kann abgeschätzt werden, wie ein Syntheseergebnis aussehen würde. Mit diesen Parametern kann eine durchschnittliche Leistungsaufnahme berechnet werden, die auch wie oben beschrieben die Effekte der schaltenden Signalleitungen mit betrachtet [22, 76, 40]. Solche Modelle können auch in einer Simulation mit den wirklich übertragenen Daten befüllt werden, um eine genauere Abschätzung für einen speziellen Anwendungsfall zu erhalten.

Greaves und Yasin [25] haben speziell für die Hardwarebeschreibungssprache *SystemC* mit TLM eine Erweiterung entwickelt, die die Transaktionsdaten so erweitert, dass zur Simulationszeit eine dynamische und performante Abschätzung der Leistungsaufnahme auf den Busleitung möglich wird. Dafür werden die zur Verfügung stehenden Systeminformationen genutzt, um die Leitungslängen abzuschätzen und die Schaltaktivitäten auf den Signalleitungen zu bestimmen.

Wie oben beschrieben, erfordern die speziellen Eigenschaften der Leistungsaufnahme von Signalleitungen und die Abhängigkeit zum Gesamtsystem (Länge und Verlegung der Signalleitungen) spezielle Modelle, wodurch es generischen Modellen für Komponenten nicht möglich ist, diese Leistungsaufnahme abzubilden. Ähnlich wie bei aktiven Komponenten können in Gesamtsystemsimulation die in diesem Abschnitt referenzierten Modelle genutzt werden, um die Leistungsaufnahme von Signalleitungen darzustellen. Aus diesem Grund fokussiert sich diese Arbeit nur auf die Modellierung von passiven Komponenten. Die Leistungsaufnahme von Signalleitungen wird nachfolgend nicht weiter betrachtet.

## 3.6. Abhängigkeit von Spannungsversorgung, Taktfrequenz und Temperatur

Wie in Abschnitt 2.2 beschrieben, hängt die statische und dynamische Leistungsaufnahme von der Temperatur und der Spannung ab. Zusätzlich gibt es noch eine Abhängigkeit zwischen der dynamischen Leistungsaufnahme und der Taktfrequenz.

Aus diesem Grund versuchen einige Arbeiten diese Abhängigkeiten mit zu modellieren, indem sie entweder eine Charakterisierung der verschiedenen Kombinationen durchführen und diese in sogenannten *Look-Up* Tabellen festhalten [19] oder indem sie Formel entwickeln, die mit den entsprechenden Parametern einen Ausgabewert berechnen.

Bei funktionalen Modellen auf hohen Abstraktionsebenen wird die zeitliche Darstellung immer gröber. Dies wirkt sich unmittelbar auf den Verlauf der daraus resultierenden Schaltungstemperatur aus. Daher versuchen andere Arbeiten durch die Verteilung der Leistungsaufnahme auf ein definiertes Zeitintervall einen realistischeren Temperaturverlauf zu erhalten [12].

Neben den dynamischen Parametern wie Versorgungsspannung, Temperatur und Taktfrequenz hängt die Leistungsaufnahme stark von der Prozessecke ab. Prozessecken entstehen durch Prozessvariationen bei der Chipherstellung. Typischerweise wird ein Leistungsaufnahmemodell für eine definierte Prozessecke erstellt. Möchte man also mehrere Prozessecken für ein Design analysieren, werden entsprechend viele Modelle benötigt. Um dem entgegen zu wirken hat Dhanwada et al. [16] ein Vorgehen vorgestellt, welches die verschiedenen Prozessecken in ein Modell – dem sogenannten *Power Contributor Model* – überführt, welches danach von dem Leistungsaufnahmemodell benutzt wird. Neben der Prozessecke erlaubt das Modell auch eine dynamische Betrachtung von Versorgungsspannung und Temperatur für ganze IP-Komponenten. Über die dynamische Hinzugabe der Prozessparameter kann dann die resultierende Leistungsaufnahme berechnet werden.

In vielen Bereichen wird heute auf das maschinelle Lernen gesetzt [34, 65]. Diese Bereiche zeichnen sich häufig dadurch aus, dass die Einflussfaktoren der Leistungsaufnahme und deren Zusammenhänge nur schwer oder gar nicht zu erkennen sind und nicht durch manuell erstellte Modelle abgebildet werden können. Bei der Energieaufnahme von integrierten Schaltungen ist dies hinlänglich bekannt, wodurch maschinelles Lernen bisher nur selten und dann nur in kleinem Umfang Anwendung (z.B. lineare Regression [42]) findet.

Das in dieser Arbeit erstellte Modell abstrahiert weitestgehend von Spannung, Frequenz und Temperatur. Dies wird dadurch ermöglicht, dass der von der Temperatur abhängige Leckwiderstand als Eingabeparameter genutzt werden kann und für die Ausgabe die geschaltete Kapazität genutzt wird, die mit Hilfe von Spannung und Taktfrequenz in die resultierende Leistungsaufnahme umgerechnet werden kann. Hiermit kann dieses Modell nahtlos in alle Umgebungen integriert werden und dies unabhängig davon, ob Abhängigkeiten zu genannten Parametern simuliert werden sollen. Eine Trennung erlaubt hier, die entsprechenden Modelle in der nötigen Genauigkeit zu modellieren. Der Parameter

*Prozessecke* wird bei diesem Modell nicht dynamisch betrachtet, sondern wird statisch vorgegeben durch die eingesetzte Technologie für die Leistungsaufnahmeberechnung auf GL, da dieser zur Laufzeit fest ist und sich nicht ändert.

## 3.7. Modellerstellung

Einige der Arbeiten in diesem Bereich konzentrieren sich lediglich auf die Modelle sowie eine genaue und schnelle Simulation. Ein weiterer wichtiger Punkt ist die Erstellung der Leistungsaufnahmemodelle auf Basis von Leistungsaufnahmewerten aus einer Simulation auf GL oder aus einer Messung der echte Hardware. Können die Modelle nicht leicht erstellt und sinnvoll mit Werten befüllt werden, sind sie für den praktischen Einsatz nicht geeignet.

Eine Möglichkeit der Modellerstellung ist die Kalibrierungen, bei der eine Regression – meistens eine lineare Regression – zu einer Annäherung des Modells an die echten Werte führt [42, 53]. Hierfür werden zum Teil automatisierte Verfahren eingesetzt, die über eine automatische Kalibrierung die korrekten Werte für die Zustände identifizieren [53].

Andere Arbeiten versuchen, definierte Operationsmodi einzelner Komponenten einzustellen und über eine Messung der reellen Schaltung die aufgenommene Leistung zu ermitteln [67].

Eine weitere Möglichkeit ist, bestimmte Events wie z.B. einzelne Berechnungen, Instruktionen oder Lese- und Schreibzugriffe zu simulieren und gezielt zu charakterisieren [38, 64].

Auch das Verfahren aus dieser Arbeit benutzt eine automatische Kalibrierung der Zustände mittels linearer Korrelation, um die datenabhängige Leistungsaufnahme und den Konstantanteil zu charakterisieren. Im Gegensatz zu allen verwandten Arbeiten stellt das Verfahren dieser Arbeit auch eine Charakterisierung der Leistungsaufnahme zur Verfügung sowie eine automatische Synthese der Automatenzustände. Die Charakterisierung der Leistungsaufnahme hilft speziell bei IP-Komponenten dabei, unbekannte aber relevante funktionale Zustandsübergänge zu finden, die einen signifikanten Einfluss auf die Leistungsaufnahme haben. Die automatische Automatensynthese beschleunigt den Modellerstellungsprozess enorm, was eine einfache und schnelle Verwendbarkeit fördert.

Wie bereits erwähnt, hat eine Modellierung auf ESL immer das Problem, dass ein Weg gefunden werden muss, wie die Modelle mit entsprechenden Leistungsaufnahmewerten befüllt werden können. Die Modellerstellung auf RTL ist deutlich einfacher als auf ESL, da das RTL-Design sehr gut auf das GL-Design abgebildet werden kann. Aus diesem Grund hat Hylla [31] ein Modell entwickelt, welches eine abstrakte Beschreibung auf ESL nach RTL synthetisiert und dort charakterisiert. Die Grundidee dabei ist, dass sogenannte Hardwareblöcke identifiziert und charakterisiert werden, die die Logik zwischen zwei Registern beschreiben. Das Ergebnis der Charakterisierung sind Funktionen, die auf Basis der Eingabe und des Zustands eine Leistungsaufnahme berechnen. Diese können in das funktionale

ESL-Modell integriert und simuliert werde. Der Nachteil hierbei ist, dass zum einen die Komponente veränderbar und die Quellcode einsehbar sein muss und somit dieses Vorgehen nicht für *Black-Box*-IP-Komponenten geeignet ist. Durch den hohen Detailgrad der Modellierung wird die Genauigkeit des Modells erhöht, die Simulationsperformanz ist im Vergleich zum Modell dieser Arbeit aber deutlich schlechter.

## 3.8. Vergleich anderer Arbeiten

In diesem Abschnitt sollen die einzelnen referenzierten Arbeiten unter Berücksichtigung der angesprochenen Problematiken bewertet und verglichen werden. Die Bewertungspunkte sind:

- Modellierung der datenabhängigen Leistungsaufnahme,
- nichtinvasive Modellierung,
- Modellierung von aktiven Komponenten,
- Modellierung von Signalleitungen,
- eine von Spannung, Frequenz und Temperatur unabhängige Modellierung,
- Modellierung von Zustandsübergängen, die nicht an den äußeren Schnittstellen sichtbar sind und
- eine festgelegte Methodik zur Modellerstellung.

Die Bewertung ist in Tabelle 3.1 gezeigt.

Bei diesem Vergleich wird sichtbar, dass die wenigsten Arbeiten eine Unterstützung für die datenabhängige Leistungsaufnahme haben und den dynamischen Einfluss von Temperatur, Versorgungsspannung und Frequenz betrachten.

Diese Arbeit zieht alle diese Faktoren in Betracht und ermöglicht dadurch eine sehr genaue Modellierung der Leistungsaufnahme bei einer gleichzeitig hohen Rechenperformanz sowie eine einfache und schnelle Integration in bestehende funktionale Systemmodelle. Des Weiteren ermöglicht es als einzige Arbeit die Abbildung von internen Zustandsübergängen für *Black-Box*-Komponenten auf die Leistungsaufnahme, was sonst nur bei *White-Box*-Komponenten und invasiven Modellen möglich ist.

Viele Arbeiten sind invasiv und können damit keine Leistungsaufnahme von *Black-Box*-Komponenten modellieren. Andernfalls hat eine invasive Modellierung den Vorteil, dass alle internen Zustandsübergänge sichtbar sind und die Leistungsaufnahme von aktiven Komponenten leicht modelliert werden kann. Um sowohl *Black-Box*- als auch *White-Box*-Komponenten modellieren zu können, arbeiten einige Ansätze mit verschiedenen Modellen für *White-Box*- und *Black-Box*-Komponenten [5, 64, 72].

Tabelle 3.1.: Vergleich der verwandten Arbeiten mit Bezug auf die in Abschnitt 1.2 aufgestellten Anforderungen.

| Methode | Black-Box | Datenabhängigkeiten | Unabhängigkeit Spannung, Frequenz und Temperatur | Interne Zustandsänderung | Aktive Komponenten | Signalleitungen | Modellerstellung |
|---|---|---|---|---|---|---|---|
| Diese Arbeit | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ |
| Bansal [5] | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| Benini [6] | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| Dhanwada [17] | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ |
| Ducroux [19] | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ |
| Gag [22] | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| Greaves [25] | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| Hylla [31] | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ |
| Klein [35] | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ |
| Kühnle [38] | ✗ | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ |
| Landman [40] | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| Lebreton [41] | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| D. Lee [42] | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ |
| I. Lee [43] | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ |
| Onnebrink [54] | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| Samei [62] | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| Schürmans [64, 63] | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ |
| Sotiriou-Xanthopoulos [67] | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ |
| Trabelsi [72] | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| Vece [74] | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| Walravens [75] | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Zhong [76] | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |

Nur eine einzige Arbeit ist in der Lage, die Leistungsaufnahme für eine aktive *Black-Box*-Komponente zu modellieren. Dies wird hier dadurch ermöglicht, dass zum einen ein komponentenspezifisches Modell entwickelt wurde und zum anderen die *Caches* des Prozessors als externe Peripherie umgesetzt sind. Damit ist der Zugriff auf diese sichtbar und legt das interne Verhalten eindeutig offen [54]. Dies ist bei aktiven *Black-Box*-Komponenten nur selten der Fall.

Alle Arbeiten teilen die Eigenschaft, dass sie die Leistungsaufnahme entweder nur von Signalleitungen oder nur Komponenten modellieren können, da die Leistungsaufnahme von Signalleitungen durch deutlich andere Parameter beeinflusst wird, die erst nach einer Platzierung der Komponenten bekannt sind und nicht durch generische Leistungsaufnahmemodelle für Komponenten abgebildet werden können.

## 3.9. Zusammenfassung

In diesem Kapitel wurden einige Herausforderungen beschrieben und gezeigt, wie verschiedene Arbeiten diese lösen und wie sich diese Arbeit von den verwandten Arbeiten unterscheidet und abhebt. Dies ist speziell bei der datenabhängige Leistungsaufnahme der Fall, die einen erheblichen Einfluss auf die Leistungsaufnahme haben kann sowie bei der von Versorgungsspannung, Temperatur und Taktfrequenz unabhängigen Modellierung, die ein sehr gute Integration nach sich führt. Zusätzlich gibt es im Gegensatz zu allen anderen Arbeiten speziell für *Black-Box*-Komponenten die Möglichkeit, interne Zustandsübergänge zu modellieren und dadurch eine sehr akkurate Modellierung der Leistungsaufnahme herbeizuführen.

Des Weiteren wurde gezeigt, dass aktive Komponenten wie CPUs sehr spezielle Modelle benötigen, um eine genaue Modellierung zu ermöglichen. Das ist besonders dann der Fall, wenn diese nur als *Black-Box*-Komponente zur Verfügung stehen.

Die Leistungsaufnahme von Signalleitungen kann durch ihre besonderen Eigenschaften nicht durch generische Modelle für Komponenten dargestellt werden und benötigt daher immer gesonderte Modelle.

Diese Arbeit kombiniert im Gegensatz zu allen bisherigen Arbeiten die benannten Schlüsselfaktoren in einem Modell für passive Komponenten und beschreibt einen Weg zur Synthese der Leistungsaufnahme zum Erstellen des entwickelten Modells.

KAPITEL 4

# Konzept zur Leistungsaufnahmemodellierung

Im letzten Kapitel wurden die Herausforderungen für diese Arbeit vorgestellt, wie andere Arbeiten diese lösen und wie sich diese Arbeit dagegen abgrenzt. In diesem Kapitel wird die Gesamtmethodik dieser Arbeit vorgestellt. Diese Methodik teilt sich auf in die Modellierung der Leistungsaufnahme und in die automatisierte Synthese zum Erstellen des Modells.

## 4.1. Gesamtmethodik

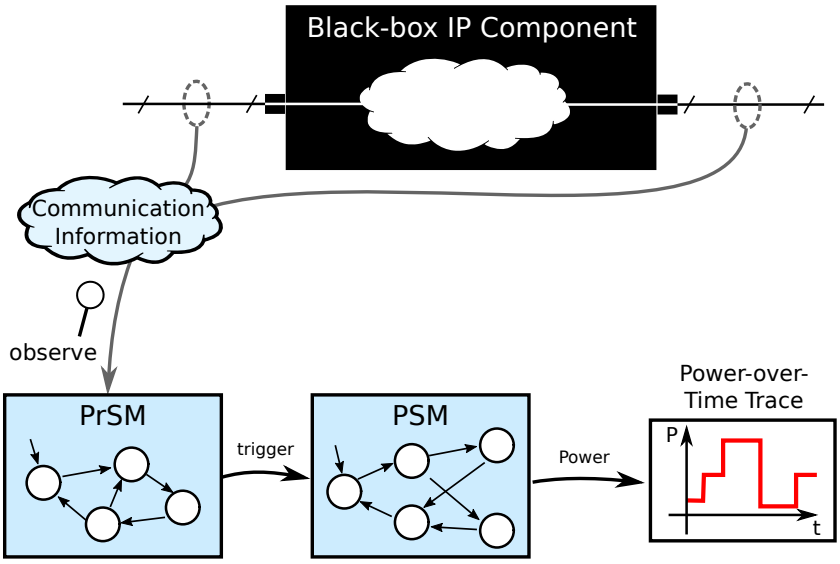



Abbildung 4.1.: PSM-Modell zur Ableitung des Leistungsaufnahmeverhaltens vom Eingabe- und Ausgabeverhalten der entsprechenden Komponente.

Das Modell ist in zwei separate Zustandsautomaten aufgeteilt, wie in Abbildung 4.1 zu sehen ist. Die *Protocol State Machine* (PrSM) beobachtet das Eingabe- und Ausgabeverhalten und bildet in abstrakter Weise den funktionalen Zustand der Komponente nach. Damit wird der interne funktionale Zustand zugreifbar, ohne die Komponente zu verändern oder

im Detail zu kennen. Die PrSM sendet ein Event an den zweiten Zustandsautomaten, die *Power State Machine* (PSM). Diese nutzt dieses Event, um einen entsprechenden Zustand einzustellen, der eine bestimmte Leistungsaufnahme repräsentiert. Des Weiteren kann jeder Zustand basierend auf einer Änderungsmetrik für die Eingangs- und Ausgangsdaten eine definierte Formel zur Berechnung der Leistungsaufnahme bereitstellen. Dies sorgt dafür, dass auch die datenflussdominierte und nicht nur kontrollflussdominierte Leistungsaufnahme berechnet werden kann. Der Leistungsaufnahmewert wird bei jeder Änderung ausgegeben. Die Ausgabe beschreibt eine Wertfolge mit Zeitstempeln. Eine genaue Beschreibung des Modells folgt in Abschnitt 4.2.

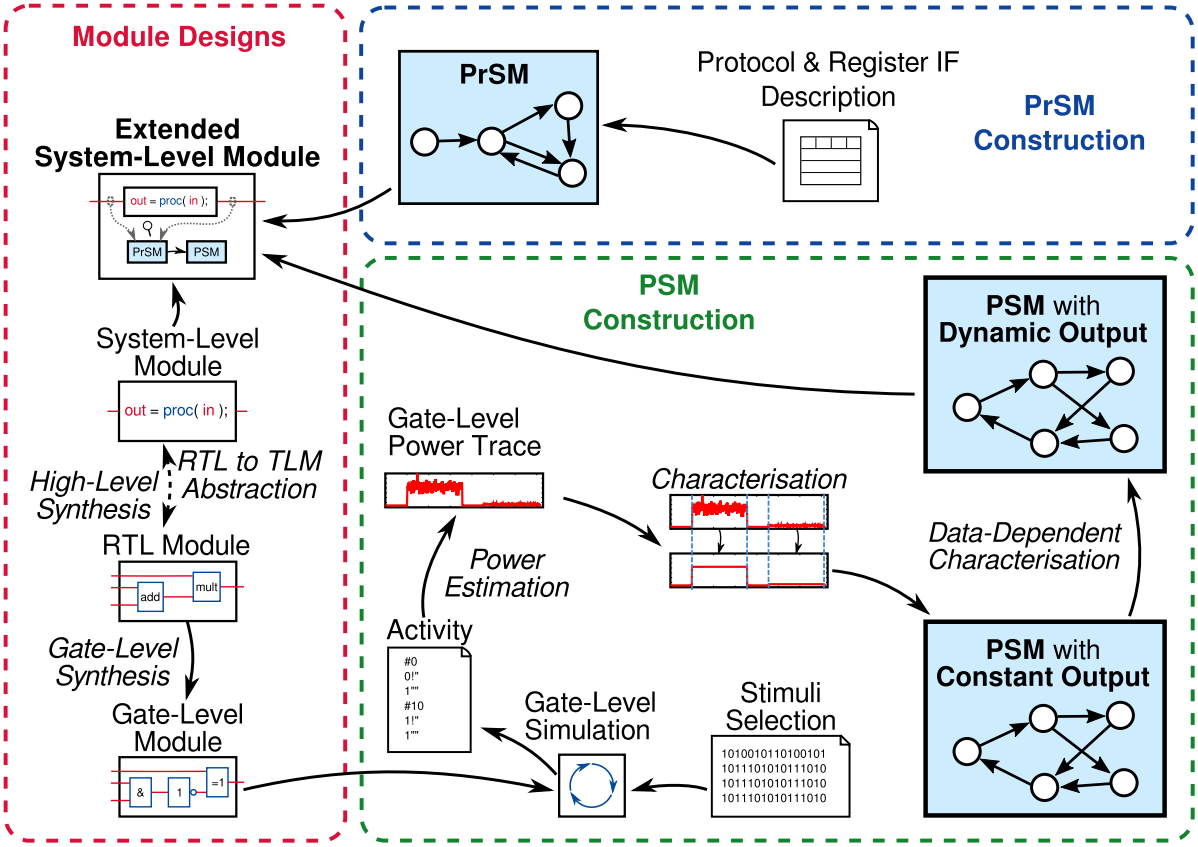



Abbildung 4.2.: Modellsynthese

Wie im letzten Kapitel beschrieben, ist die Modellerstellung ein wichtiger Aspekt. Die in dieser Arbeit entwickelte Modellsynthese ist in Abbildung 4.2 gezeigt. Da das Modell aus zwei Automaten besteht, ist der Syntheseprozess in zwei Schritte unterteilt, die jeweils die PrSM und die PSM erzeugen.

Die PrSM wird aus den funktionalen Informationen erzeugt, die dem Entwickler zur Verfügung stehen, um eine abstrahierte Variante des internen Zustandsautomats der Komponente nachzubauen. Dies ist zum einen die Schnittstellenbeschreibung und zum anderen Funktionsbeschreibung der Komponente.

Für die Erzeugung der PSM wird die Komponente nach GL synthetisiert und mit geeigneten Anwendungsfällen simuliert. Bei diesen werden taktgenaue Wertverläufe erzeugt, die als *Trace* bezeichnet werden. Diese können mit Hilfe einer Technologiedatenbank in einen Leistungsaufnahme-Trace überführt werden. Auf Basis dieser kann die PSM erstellt und nachfolgend mit einer dynamischen Ausgabe erweitert werden, wenn eine datenabhängige Leistungsaufnahme modelliert werden soll.

Im Anschluss wird das funktionale Modell mit dem Leistungsaufnahmemodell erweitert. Eine detaillierte Beschreibung der Synthese folgt in Kapitel 5.

## 4.2. Einführung in das Leistungsaufnahmemodell

Der Hauptbeitrag dieser Arbeit ist die Modellierung der Leistungsaufnahme von Systemebenenkomponenten basierend auf den in Kapitel 1 beschriebenen Anforderungen. Zusammengefasst soll also ein Modell entstehen, welches es ermöglicht, auf ESL die Leistungsaufnahme aufzuzeichnen, wobei das Modell nicht invasiv sein soll, nicht erkennbare Zustandswechsel modellieren kann, unabhängig von Versorgungsspannung, Taktfrequenz und Temperatur ist sowie Datenabhängigkeiten in der Leistungsaufnahme darstellen soll, falls welche vorhanden sind. Dies ermöglicht eine schnelle und ausreichend genaue Simulation der Leistungsaufnahme von hochkomplexen Systemen.

Nachfolgend wird nun das Modell beschrieben, welches es ermöglicht, die Leistungsaufnahme von Systemkomponenten auf ESL zu beschreiben. Dabei soll es die in Abschnitt 1.2 beschriebenen Anforderungen erfüllen. Dadurch wird ermöglicht, dass die Leistungsaufnahme von Black-Box Komponenten und benutzerdefinierten Komponenten ohne Änderung des funktionalen Modells simuliert werden kann, eine leichte Integration des Leistungsaufnahmemodells möglich ist, es mit anderen extra-funktionalen Modellen kombiniert werden kann und dabei das Verhalten der Leistungsaufnahme so genau wie möglich darstellt. Der Einfluss auf die Performanz bei einer Simulation des Modells soll dabei minimal bleiben. Des Weiteren werden die einzelnen Bestandteile des Leistungsaufnahmemodells erklärt, um die Abhängigkeiten zwischen funktionalem und nicht-funktionalem Modell sowie die semantischen Eigenschaften zu verstehen. Das Modell wird formal definiert und die Anwendung des Modells anhand von Beispielen verdeutlicht.

Wie in Abschnitt 2.4 beschrieben, ist die ESL-Simulation eine eventbasierte Simulation. Dadurch ergeben sich Zeitintervalle, in denen das System in einem spezifischen Zustand ist. Bereits Benini hat 1998 erkannt, dass Zustandsautomaten sich gut eignen, um das Verhalten der Leistungsaufnahme zu simulieren [6]. Dafür entwickelte er die sogenannte *Power State Machine* (PSM), wie in Abbildung 4.3 zu sehen. Jede Ressource des Systems bekommt dafür eine eigene PSM, die den Zustand der gegenwärtigen Leistungsaufnahme darstellt. Die PSM ist ein Zustandsautomat, bei dem jeder Zustand eine spezifische Leistungsaufnahme darstellt, wobei die Zustandsübergänge mit einer definierten Übergangszeit annotiert werden, um deren Kosten abzubilden. Er hat erkannt, dass die Leistungsaufnahme

stark von der Schaltaktivität der Komponente abhängt und führte dafür Zustände ein, die abhängig von der maximalen Leistungsaufnahme und der Schaltaktivität die resultierende Leistungsaufnahme für jede Komponente berechnen.

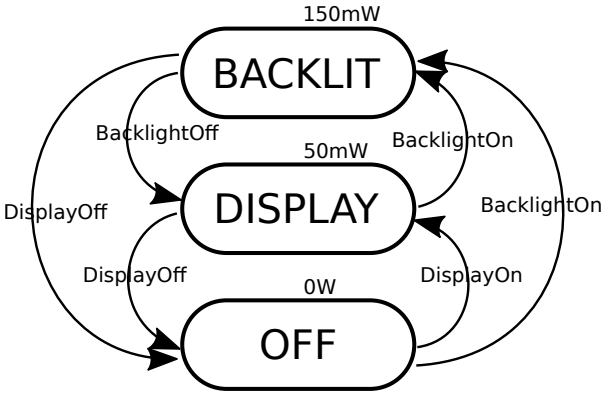



Abbildung 4.3.: Systemmodell der Leistungsaufnahme von Benini [6].

Benini abstrahierte dabei vom kompletten funktionalen System und setzte als Steuerlogik für die einzelnen PSMs einen globalen *Power Manager* ein, wie in Abbildung 4.4 gezeigt. Dieser übernimmt die Kommunikation mit der Außenwelt und setzt Anfragen in Befehle an die einzelnen PSMs um, die daraufhin ihren Zustand anpassen. Er übernimmt dabei die Aufgabe eines abstrakten Kontrollmechanismus, der von dem realen System abgeleitet ist.

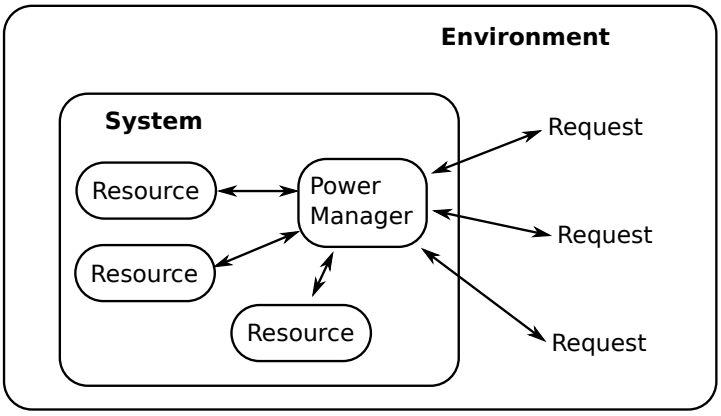



Abbildung 4.4.: Beispiel *Power State Machine* (PSM) von Benini [6].

In seiner Arbeit beschreibt Benini, dass anstatt eines globalen *Power Managers* zu verwenden auch jede Ressource selber eine Logik beinhalten kann, die den Zustand seiner PSM einstellt. Dieser Ansatz wird in dieser Arbeit verfolgt, da primär die Komponenten im einzelnen betrachtet werden anstatt das ganze System. Daher dient dieses Modell als Grundlage für diese Arbeit.

Jedoch kommen mit den in Abschnitt 1.2 beschriebenen Anforderungen neue Herausforderungen hinzu, die eine Änderung des Modells erfordern, um die Anforderungen in vollem

Umfang zu erfüllen. Nachfolgend wird beschrieben, wie das Modell inkrementell geändert und erweitert wird, um alle Anforderungen zu erfüllen.

Eine weitere Möglichkeit der Modellierung wäre der Einsatz von Markov-Modellen, die eine stochastische Betrachtung der Leistungsaufnahme ermöglichen würden. Diese basieren aber auf einer statischen Betrachtung des Systems. Dadurch, dass sich das System aber von Anwendungsfall zu Anwendungsfall verschieden verhält, ist eine dynamische Betrachtung des Systems nötig, um hinreichend genaue Werte für die Leistungsaufnahme zu erlangen.

## 4.3. Power State Machine Modell

Wie im vorherigen Abschnitt beschrieben, soll die PSM von Benini als Grundlage genommen werden, da sich dieses Modell als zuverlässig für eine Modellierung der Leistungsaufnahme auf Systemebene erwiesen hat. Im Unterschied zu Beninis Arbeit soll die PSM in dieser Arbeit in ein bestehendes funktionales System eingebettet werden. Dadurch können die zur Verfügung stehenden funktionalen Blöcke genutzt werden, um die PSMs zu triggern. Durch die zusammenhängende Betrachtung von Leistungsaufnahme und Funktion können Effekte der Leistungsaufnahme, die einen Einfluss auf die Funktion haben, zusammenhängend analysiert werden. Beispielsweise ist es wichtig, ob sich das System funktional richtig verhält, wenn Versorgungsspannung und Taktfrequenz durch DVFS heruntergesetzt werden und sich dadurch das zeitliche Verhalten verändert.

Anstatt einen globalen oder lokalen *Power Manager* zu verwenden, kann die Funktion so annotiert werden, dass sie beim Wechsel des Zustands einen Befehl an die zugehörige PSM aussendet, die daraufhin entsprechend ihren Zustand ändert. Dieses Modell ist in Abbildung 4.5 gezeigt und wird als Ausgangsbasis genutzt.

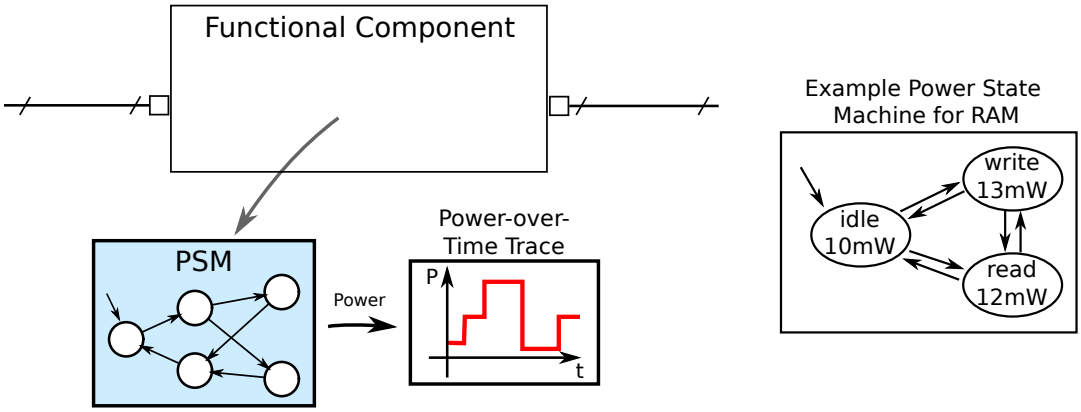



Abbildung 4.5.: Einfache PSM, die durch den funktionalen Block getriggert wird mit Beispiel PSM für einen RAM-Speicherblock.

Die Idee bei dieser PSM ist, abhängig vom aktuellen funktionalen Verhalten der Komponente das Verhalten der Leistungsaufnahme abzuleiten. Wie in Abschnitt 2.2.2 beschrieben,

führt Schaltaktivität zu dynamischer Leistungsaufnahme. Daher wird versucht, aus dem aktuellen funktionalen Verhalten die resultierende Schaltaktivität abzuleiten, aus der die entsprechende Leistungsaufnahme berechnet werden kann.

Für die Synchronisation zwischen funktionalem Modell und PSM soll eine Menge $E$ von $n$ Events $e$ definiert werden. Jedes Event besteht aus einem eineindeutigen Namen. Es kann immer nur genau ein Event aktiv sein:

$$e := (\text{Name}) \quad \text{mit} \quad e \in E \quad . \tag{4.1}$$

Die Menge $E$ wird nachfolgend als PSM-Events bezeichnet.

Die PSM kann demnach analog zu einem einfachen deterministischen Zustandsautomat folgendermaßen dargestellt werden:

$$psm := (S_{psm}, s_{0,psm}, \Sigma_{psm}, \Gamma_{psm}, E_{psm}) \quad , \tag{4.2}$$

wobei $S_{psm}$ die Menge der Zustände darstellt und $s_{0,psm}$ den Startzustand. Die Ausgabe erfolgt in Abhängigkeit vom Zustand, wodurch ein Zustand $s_{psm} \in S_{psm}$ wie folgt definiert ist:

$$s_{psm} := (\text{Name}, \gamma_{psm}) \quad \text{mit} \quad \gamma_{psm} \in \Gamma_{psm} \quad . \tag{4.3}$$

$\Sigma_{psm}$ ist das Eingabealphabet und somit entspricht:

$$\Sigma_{psm} = E \quad . \tag{4.4}$$

$\Gamma_{psm}$ ist das Ausgabealphabet, eine endliche Menge von Werten für die durchschnittlichen Leistungsaufnahme:

$$\Gamma_{psm} := \{\gamma | \gamma \in \mathbb{R}_{\geq 0}\}^n, n \in \mathbb{N} \quad . \tag{4.5}$$

$E_{psm}$ beschreibt die Zustandsübergänge:

$$E_{psm} := S_{psm} \times \Sigma_{psm} \times S_{psm} \quad . \tag{4.6}$$

Ein Problem mit diesem Modell ergibt sich schnell, sobald man Black-Box Komponenten einsetzt oder Komponenten, die nicht verändert werden können, um sie mit entsprechenden Trigger-Befehlen zu annotieren.

## 4.4. Black-Box Komponenten

In heutigen Systemen werden Komponenten oftmals nicht neu entworfen, sondern als sogenannte *Intellectual Property* (IP) zugeliefert. Diese haben in der Regel den Vorteil, dass sie einen hohen Reife- und Qualitätsgrad aufweisen und kein erneuter Entwicklungsaufwand

investiert werden muss. Um Know-How Schutz zu betreiben, werden diese Komponenten oftmals als Black-Box Komponenten ausgeliefert. Das bedeutet, es sind lediglich die Schnittstellen sichtbar, aber nicht der innere Aufbau und das funktionale Verhalten. Des Weiteren ist in den meisten Fällen auch keine Änderung des funktionalen Designs möglich, wodurch das Modell aus Abbildung 4.5 so nicht angewendet werden kann. Auch viele andere Modelle ändern aktiv das funktionale Design, wie in Kapitel 3 beschrieben, um die Anbindung eines Leistungsaufnahmemodells zu ermöglichen. Des Weiteren sind viele Modelle auf ein spezifisches und proprietäres Ökosystem angewiesen, welches die Integration erschwert. Wenige der bestehenden Modelle ermöglichen eine nahtlose Integration mit anderen extra-funktionalen Modellen, wie z.B. die Modellierung der Wärmeentwicklung, welche in der heutigen Zeit eine immer wichtigere Rolle spielen. Im Folgenden wird nun beschrieben, wie das Modell aus Abbildung 4.5 abgeändert wird, um eine Detektion des funktionalen Verhaltens zu ermöglichen, ohne das Ökosystem zu verändern.

Wie oben erwähnt, sind bei Black-Box-IP-Komponenten nur die Schnittstellen sichtbar. Demnach kann nur das Schnittstellenverhalten genutzt werden, um einen Rückschluss auf das funktionale Verhalten zu erlangen. Aus diesem Grund werden als Eingabe für das PSM-Modell die Eingabe- und Ausgabewerte der jeweiligen Komponente genutzt. Es werden die Änderungen an allen Ports betrachtet, die zu einem spezifischen Event (z.B. steigende Taktflanke oder Beginn einer Transaktion) stattfinden. Diese werden dann als Eingabe- und Ausgabesymbol bezeichnet.

Die Menge der Eingabe- und Ausgabesymbole $V_P$ haben einen zeitlichen Verlauf, wodurch jedes Eingabe- und Ausgabesymbol $v_P$ mit einem Zeitstempel $t$ versehen ist. Dadurch ergibt sich für jede Eingabe $i \in I$:

$$i = (t, v_P) \quad \text{mit} \quad v_P \in V_P \quad , \tag{4.7}$$

wobei $V_P$ den gesamten Wertebereich und deren Kombinationen der Symbole über allen Ports $P$ darstellt.

Der Typ von $V_P$ ist ein nicht näher definierter Datentyp. Dieser kann einem primitiven Datentyp wie z.B. `bool` oder `double` entsprechen aber auch komplexen Datentypen auf ESL, die mehrere Felder wie die Adresse und übertragene Daten enthalten, wie es für TLM genutzt wird.

Die Menge der Ein- und Ausgabesymbole $I$ ist demnach wie folgt definiert:

$$I := \{i_1, \ldots, i_w\} = \left\{(t_1, v_{p,1}), \ldots, (t_w, v_{p,w})\right\} \quad , \tag{4.8}$$

wobei $w$ eine endliche Anzahl an betrachteten Eingaben ist und die Zeiten der Eingaben $t_1, \ldots, t_w$ streng monoton steigend sind.

Das Modell wird parallel zur funktionalen Komponente erstellt, da es zum einen bei Black-Box-Komponenten nicht integriert werden kann und zum anderen eine gute Austauschbarkeit, Integration, Wartung und Erweiterung ermöglicht.

Bei dieser Änderung ergeben sich nun zwei Probleme:

1. die PSM kann nur PSM-Events verarbeiten und keine Ein- oder Ausgabestrom und
2. es wird eine Indirektion des funktionalen Verhaltens herbei geführt.

Das heißt, es muss ein Mechanismus gefunden werden, der die Indirektion wieder auflöst und den Ein- und Ausgabestrom wieder in die PSM-Events übersetzt.

Eine Möglichkeit wäre, diesen Mechanismus mit in der PSM zu integrieren und das Eingabealphabet entsprechend anzupassen. Dies würde jedoch dazu führen, dass ein Teil des funktionalen Verhaltens in der PSM nachgebildet wird und dadurch eine orthogonale Modellierung von funktionalen Verhalten und Leistungsaufnahmeverhalten nicht mehr möglich ist. Aus diesem Grund wird in dem PSM-Modell ein weiterer endlicher deterministischer Zustandsautomat eingeführt, der diese Indirektion auflöst. Das bedeutet, er stellt aus den Eingabe- und Ausgabewerten ein abstrahiertes funktionales Verhalten nach. Die Ausgaben dieses Automaten sind die in (4.1) PSM-Events. Dadurch wird hier gleichzeitig eine Übersetzung von Ein- und Ausgaben zu PSM-Events durchgeführt und es wird keine Änderung der PSM notwendig, die weiterhin PSM-Events als Eingabe bekommt.

Nicht alle Änderungen des funktionalen Zustands sind an den Schnittstellen sichtbar. Das führt dazu, dass dieser Automat also lediglich das Protokollverhalten abbildet, welches eine Untermenge des funktionalen Verhaltens darstellt. Daher wird er nachfolgend als sogenannte *Protocol State Machine* (PrSM) bezeichnet. Die PrSM bekommt als Eingabe die Eingabe- und Ausgabewerte der funktionalen Komponente, also die Symbole aus (4.7).

Die PrSM lässt sich wie folgt formal darstellen:

$$prsm := (S_{prsm}, s_{0,prsm}, \Sigma_{prsm}, \Gamma_{prsm}, E_{prsm}) \quad , \tag{4.9}$$

wobei die Darstellung hier analog zur PSM ist mit Unterschieden beim Eingabe- und Ausgabealphabet und der Ausgabe der Symbole. Das Eingabealphabet $\Sigma_{prsm}$ der PrSM ist die Menge der Eingabe- und Ausgabesymbole der funktionalen Komponente wie in (4.7) definiert und das Ausgabealphabet $\Gamma_{prsm}$ ist die Menge der PSM-Events $E$ inklusive dem leeren Wort:

$$\Sigma_{prsm} = V_P \quad \text{und} \tag{4.10}$$

$$\Gamma_{prsm} = E \cup \varepsilon \quad . \tag{4.11}$$

Das leere Wort für $\Gamma_{prsm}$ wird benötigt, da nicht bei jedem Zustandsübergang zwingend eine Synchronisation und damit eine Ausgabe der PrSM benötigt wird. Der Grund dafür ist, dass nicht zwingend bei jedem funktionalen Zustandswechsel auch der Zustand der Leistungsaufnahme sich ändert.

In der PrSM ist die Ausgabe nicht abhängig von einem Zustand sondern von einem Zustandsübergang. Der Grund dafür ist, dass auch in dem funktionalen Design nicht bei dem Betreten eines bestimmten Zustands sondern bei dem Ausführen eines definierten

Zustandsübergangs ein PSM-Event erzeugt werden würde. Demnach ist ein Zustand der PrSM wie folgt definiert:

$$s_{prsm} := (\text{Name}) \tag{4.12}$$

und der Zustandsübergang entspricht:

$$E_{prsm} := S_{prsm} \times \Sigma_{prsm} \times \Gamma_{prsm} \times S_{prsm} \quad . \tag{4.13}$$

Durch diese Modellierung erhält man synchron kommunizierende Zustandsautomaten, wie sie Holzmann definiert hat [30]. Die Kommunikation ist synchron, da die PSM immer direkt auf die Ausgabe der PrSM reagiert. Der Grund hierfür ist, dass versucht wird, ein Modell der Leistungsaufnahme zu erstellen, welches einen Trace der Leistungsaufnahme über die Zeit erstellt. Da die PSM keine eigene Uhr besitzt, muss sie also synchron zum funktionalen Design ausgeführt werden. Das nun entstandene Modell ist in Abbildung 4.6 zu sehen und ein Beispiel für einen Speicherblock ist in Abbildung 4.7 gezeigt.

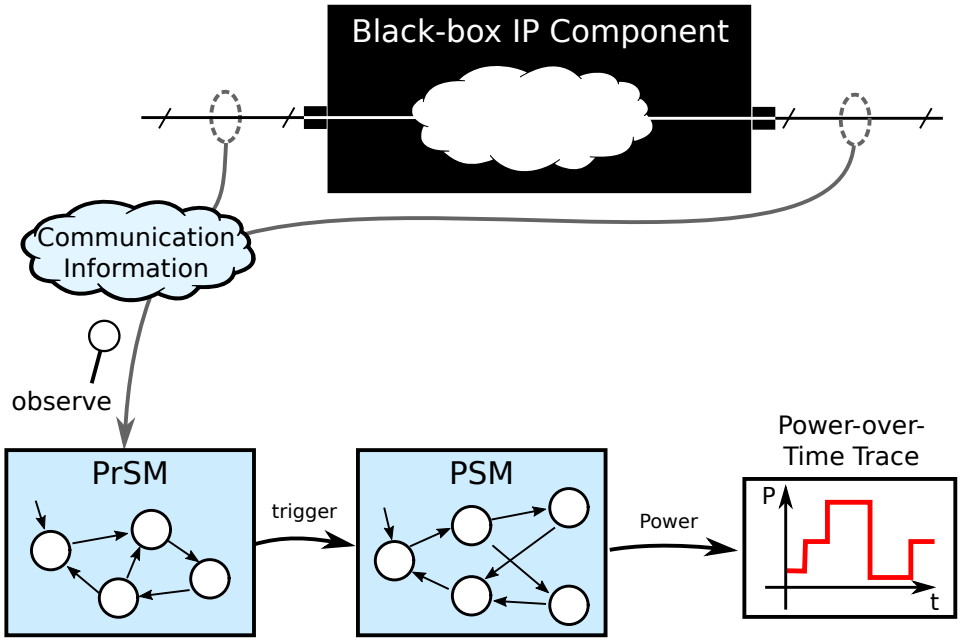



Abbildung 4.6.: PSM-Modell erweitert mit einer PrSM zur Nachbildung des funktionalen Verhaltens aus den Ein- und Ausgabewerten der funktionalen Komponente.

Die PrSM kann ähnlich wie ein lokaler *Power Manager*, so wie er von Benini vorgeschlagen wurde, gesehen werden, der den Kontrollfluss in Befehle übersetzt, welche von der PSM verarbeitet werden.

Durch die Auflösung der Indirektion des funktionalen Verhaltens führt die PrSM auch eine Abstraktion der Eingabe- und Ausgabewerte durch. Zusätzlich ermöglicht sie, dass bei Änderung der Schnittstellenmodellierung (z.B. von TLM auf pinakkurate Modellierung) lediglich die PrSM getauscht werden muss und die PSM unverändert beibehalten werden kann.

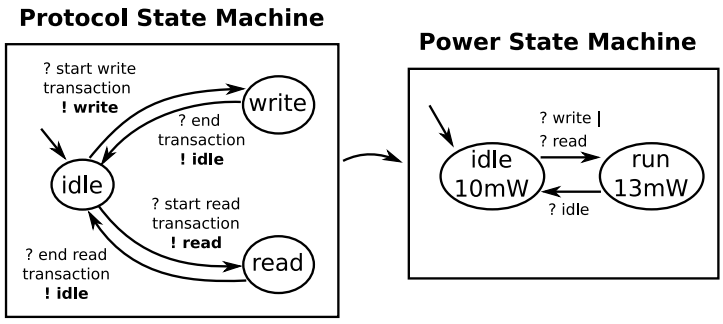



Abbildung 4.7.: Beispiel von einer PSM und PrSM für einen Speicherblock auf ESL. Die PrSM wird durch die Lese- und Schreibtransaktionen der Komponentenschnittstelle gesteuert.

Ändert sich im Gegenzug die Implementierung einer Komponente und die Schnittstelle und deren Verhalten bleiben gleich, muss nur die PSM ausgetauscht werden. In den meisten Fällen werden an die PSM-Zustände lediglich neue Werte für die Leistungsaufnahme annotiert, der Automat bleibt ansonsten unverändert.

Diese Art der Modellierung eignet sich nur für *Slave*-Komponenten, da eine Ableitung des funktionalen Verhaltens von *Master*-Komponenten aus den Ein- und Ausgaben nahezu unmöglich ist. *Master*-Komponenten sind aktive Komponente und führen die meisten Zustandsübergänge ohne Kommunikation an den Schnittstellen durch. Für eine ausreichend genaue Modellierung der Leistungsaufnahme von *Master*-Komponenten muss diese einzeln detailliert analysiert werden. Ein generisches Modell wie die PSM führt hier mit großer Wahrscheinlichkeit zu hohen Abweichungen.

## 4.5. Power Modell

Bei heutigen Systemen hängt die Leistungsaufnahme neben der Schaltaktivität auch von der Temperatur, der Versorgungsspannung und Prozessparametern der Fertigung ab. Diese Faktoren werden von dem bisherigen Modell noch nicht betrachtet.

Prozessparameter sind statisch und ändern sich nicht zur Laufzeit, sondern sind unterschiedlich zwischen mehreren Chips oder zwischen mehreren Bereichen auf einem Chip. Aus diesem Grund können diese als konstant angesehen werden und müssen nur mittels Veränderung der PSM-Ausgabewerte angepasst werden. Hat man beispielsweise einen schnelleren Chip, hat diese typischerweise eine höhere Leistungsaufnahme und umgekehrt ist die Leistungsaufnahme bei einem langsamen Chip niedriger. Prozessparameter dürfen aber keineswegs ignoriert werden, da sie eine hohe Variation bis zu einem Faktor von 25 für die Leckströme und Zeitverhalten für den kritischen Pfad herbeiführen können [58]. Auch die dynamische Leistungsaufnahme kann dadurch beeinflusst werden, mittelt sich aber über eine abstrakte Betrachtung heraus.

Die Spannung $V$ hingegen hat einen direkten quadratischen Einfluss auf die Leistungsaufnahme $P_{dyn}$, wie aus der Formel für die dynamische Leistungsaufnahme hervorgeht, vgl. Gleichung (2.7):

$$P_{dyn} = \frac{1}{2} \cdot C_{dyn} \cdot V_{dd}^2 \cdot f \cdot \alpha \quad .$$

Des Weiteren gibt es in heutigen System fast immer Mechanismen wie DVFS (siehe Abschnitt 2.3.2), die mittels Anpassung der Versorgungsspannung eine möglichst niedrige Leistungsaufnahme einstellen. Hierbei muss in der Regel auch die Taktfrequenz mit angepasst werden, damit eine Senkung der Versorgungsspannung möglich ist. Um die PSM unabhängig von Versorgungsspannung und Taktfrequenz zu machen, gibt es die Möglichkeit, anstatt die Zustände der PSM mit Leistungsaufnahmewerten zu annotieren, die mittlere geschaltete Kapazität $\overline{C}$ zu nehmen. Diese berechnet sich durch

$$\overline{C} = \frac{P_{dyn}}{V_{dd}^2 \cdot f} = \frac{1}{2} C \cdot \overline{a} \quad , \tag{4.14}$$

wobei $C$ die Gesamtkapazität und $\overline{a}$ die durchschnittliche Schaltaktivität über Zeit ist, in der der entsprechende Zustand aktiv ist.

Auch die statische Leistungsaufnahme hängt direkt von der Versorgungsspannung ab. Diese kann analog zur dynamische Leistungsaufnahme behandelt werden. Jedoch berechnet man hier den Leckwiderstand $R_{leak}$:

$$R_{leak} = \frac{V_{dd}^2}{P_{stat}} \quad . \tag{4.15}$$

Ein nachgelagertes *Power Modell* kann nun aus der mittleren geschalteten Kapazität, dem Leckwiderstand, der Versorgungsspannung und Taktfrequenz die resultierende dynamische und statische Leistungsaufnahme berechnen:

$$P_{dyn} = \overline{C} \cdot V_{dd}^2 \cdot f \quad \text{und} \tag{4.16}$$

$$P_{stat} = \frac{V_{dd}^2}{R_{leak}} \quad . \tag{4.17}$$

Wie die PSM aussieht, wenn man ein nachgelagertes *Power Modell* benutzt, ist in Abbildung 4.8 gezeigt. Mit der Einführung des *Power Modells* ändert sich bei der PSM der Typ das Ausgabealphabets $\Gamma_{psm}$ von Watt zu Farad, der Wertebereich bleibt jedoch gleich. Dementsprechend ist an dieser Stelle keine Neudefinition nötig.

Die Versorgungsspannung und Taktfrequenz wird häufig von einem sogenannten *Power Manager* durchgeführt, der die funktionalen Komponenten überwacht und je nach Systemzustand entsprechend die Leistungsaufnahme anpasst. Des Weiteren informiert er die funktionalen Komponenten über diese Änderung, damit sie ihre Funktion entsprechend anpassen. Wird das Modell in einer Simulation ausgeführt, werden die Ausführungszeiten und Funktionen der Komponenten entsprechend angepasst. Des Weiteren sendet der *Power*

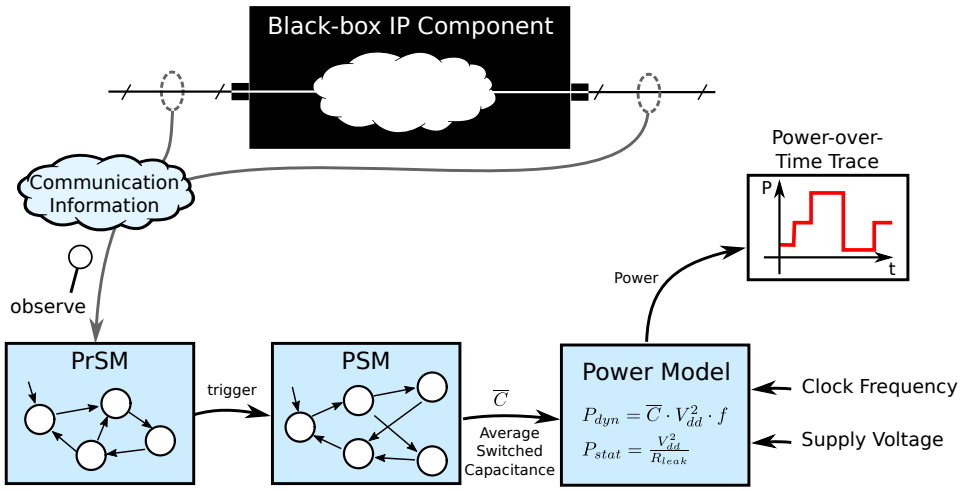



Abbildung 4.8.: PSM mit *Power Modell* zur Abstraktion von Versorgungsspannung und Taktfrequenz.

*Manager* die aktuelle Versorgungsspannung und Taktfrequenz an das *Power Modell* der PSM, welches daraufhin die Ausgabewerte für dynamische und statische Leistungsaufnahme anpasst.

Die Temperatur beeinflusst maßgeblich die statische Leistungsaufnahme und hat nahezu keinen Einfluss auf die dynamische Leistungsaufnahme. Das PSM-Modell betrachtet nur die dynamische Leistungsaufnahme und nimmt die statische Leistungsaufnahme als konstant an. Die Begründung hierfür ist in Abschnitt 2.4 nachzulesen. Damit ist die PSM implizit unabhängig von der Temperatur. Möchte man die statische Leistungsaufnahme abhängig von der Temperatur modellieren, benötigt man neben einem Temperaturmodell auch ein Modell für die statische Leistungsaufnahme, welches die Temperatur als Eingangswert verwendet, um die statische Leistungsaufnahme entsprechend anzupassen. Ein Beispiel für ein solches Modell ist in Abbildung 4.9 zu sehen.

Hierbei nimmt das Temperaturmodell den Wert der gesamten Leistungsaufnahme entgegen und berechnet mit Hilfe von weiteren Parametern, wie Package-Informationen, vorherige Temperatur, Außentemperatur und dem Bereich, in dem die Leistungsaufnahme entstanden ist, die resultierende Temperatur. Diese Temperatur kann nun in einem Modell für die statische Leistungsaufnahme – hier das *Leaking Resistance Modell* – genutzt werden, um den Leckwiderstand zu berechnen. Dieser wird dann wie auch in Abbildung 4.8 für eine neue Berechnung der statischen Leistungsaufnahme genutzt. Dieses Modell wird an dieser Stelle nur beispielhaft gezeigt und es wird nicht weiter vertieft auf das Temperaturmodell eingegangen, da der Fokus dieser Arbeit auf der Modellierung der Leistungsaufnahme liegt. Jedoch ermöglicht der modulare Aufbau des PSM-Modells eine Anbindung an beliebige Temperaturmodelle.

Um verschiedene Modelle, wie eine PSM und ein Temperaturmodell, miteinander verknüpfen zu können, bedarf es eines modularen und flexiblen Konzepts, um sowohl die

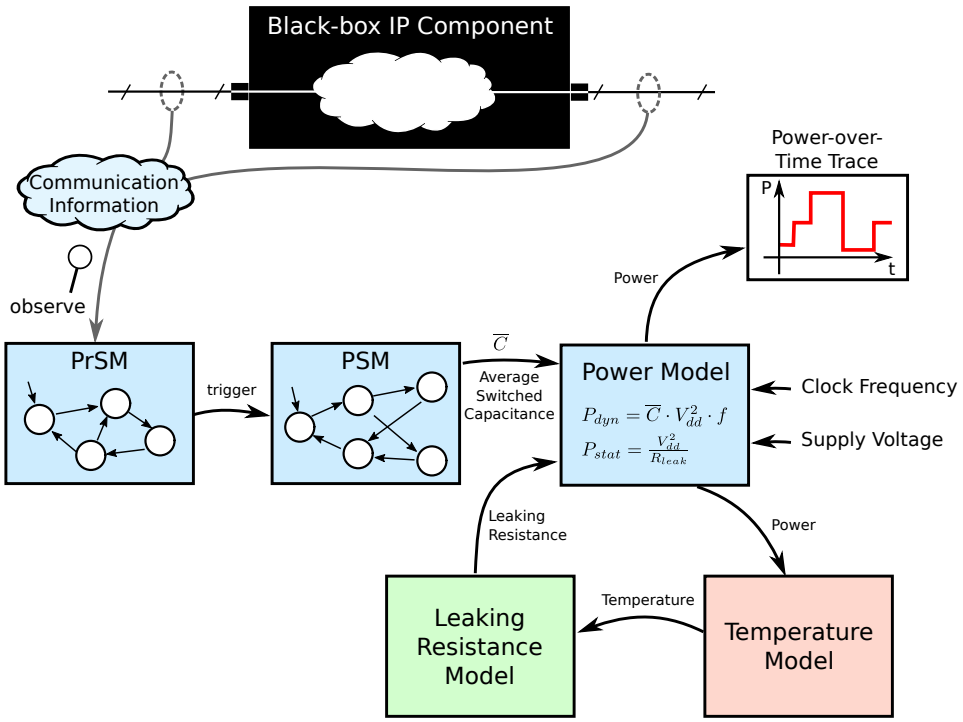



Abbildung 4.9.: PSM erweitert mit einem Modell zur Temperaturberechnung und zur Berechnung vom Leckwiderstand, dessen Ausgabe für eine korrekte Berechnung der statischen Leistungsaufnahme genutzt wird.

Daten als auch die dazugehörigen Zeitstempel zwischen den Modellen auszutauschen. Ein solches Modell wurde von Hartmann et al. [27] entwickelt. Hierbei werden Daten mit einem Gültigkeitszeitintervall annotiert. Durch eine Verkettung dieser Tupel ergibt sich der absolute Anfangs- und Endzeitpunkt aus den Vorgängerwerten. Dadurch wird keine synchrone Zeitbasis mit globaler Uhr gebraucht und ermöglicht eine asynchrone Ausführung der Modelle. Eine Implementierung dieses Modells ist bei *GitHub* [21] verfügbar.

Mit den in diesem Abschnitt erläuterten Änderungen ist die Anforderung F3 zur Unabhängigkeit von Taktfrequenz, Versorgungsspannung und Temperatur erfüllt.

## 4.6. Zustandsvariablen

Die Ausdruckskraft des bisherigen PSM-Modells ist nicht immer ausreichend. Dieses kann bisher nur Funktionen ohne Seiteneffekte darstellen. Ein Beispiel für einen Seiteneffekt ist, wenn nach einer bestimmten Anzahl von Aktionen ein Zustandswechsel im funktionalen Verhalten oder dem Verhalten der Leistungsaufnahme stattfindet. Dies deutet auf einen internen, nicht sichtbaren, funktionalen Zustandswechsel hin. Dieser Punkt wird in Abschnitt 4.7 betrachtet. Häufig ist diese Anzahl nicht fest, sondern hängt von einer bestimmten Konfiguration der Komponente ab, die dynamisch zur Laufzeit geändert werden

kann. Ein weiteres Beispiel ist, wenn die Leistungsaufnahme von spezifischen Daten abhängt, die nicht durch unterschiedliche Zustände in der PrSM unterschieden werden. Hier sprechen wir von einer datenabhängigen Leistungsaufnahme, auf die näher in Abschnitt 4.8 eingegangen wird.

Um dieses Verhalten modellieren zu können, werden im Folgenden sogenannte *Zustandsvariablen* eingeführt. Dafür wird das Konzept der *Extended Finite State Machine* (EFSM) [30] aufgegriffen. Hierbei wird der Zustandsraum um den Wertebereich der Zustandsvariablen vergrößert. Dadurch kann mit wenig Aufwand eine Erweiterung des Zustandsraums durchgeführt und gleichzeitig die Komplexität gering gehalten werden.

Für die oben beschriebenen Beispiele sind zwei verschiedene Arten von Zustandsvariablen nötig:

1. Zählvariablen, die das Vorkommen von bestimmten Aktionen oder Events zählen, und
2. geteilte Zustandsvariablen, die spezifische Werte aus dem Eingabestrom oder anderen Berechnungen speichern, um die Ausführung der beiden Automaten zu steuern.

Ein typisches Beispiel für das Zählen von Aktionen ist eine Komponente, die einen kontinuierlichen Eingangswertestrom verarbeitet. Solche Komponenten beginnen in der Regel erst mit der Verarbeitung der Daten, wenn eine gewisse Anzahl an Daten vorliegt, um einen durchgängigen Ausgangswertestrom garantieren zu können, auch wenn es leichte Verzögerungen im Eingangswertestrom gibt. Sind diese Werte aus Datenblättern bekannt oder können aus der Konfiguration im Eingabe- oder Ausgabewerten extrahiert werden (z.B. beim Einstellen der Puffergröße in dem oben genannten Beispiel), können diese Informationen in Form von den oben genannten **Zählvariablen** umgesetzt werden. Für diese gibt es für die PrSM und PSM einen Container, der diese Variablen verwaltet. Für die Verwaltung eines Zählers werden immer zwei Variablen benötigt. Eine speichert den aktuellen Konfigurationswert und die andere speichert den Zählwert. Der Konfigurationswert bekommt abhängig von der funktionalen Komponente einen entsprechenden Initialwert zugewiesen, der durch die PrSM geändert werden kann, sobald diese eine Änderung des Konfigurationswertes in den Eingabe- und Ausgabewerten der funktionalen Komponente feststellt. Die Zählvariable wird auf den Wert 0 initialisiert und inkrementiert, sobald die verknüpfte Aktion oder das verknüpfte Event auftritt. Erreicht die Zählvariable den Wert der Konfigurationsvariable, wird eine damit verknüpfte Zustandsänderung ausgeführt und die Zählvariable wieder auf den Wert 0 gesetzt.

Im Gegensatz zu den Konfigurationsvariablen, die nur durch die PrSM geändert werden können, weil diese abhängig vom Eingabe-/Ausgabestrom sind, werden die Zählvariablen von dem Zustandsautomaten geändert, der diese verwendet.

Es wird eine Menge von Zählvariablen $Z_{cn}$ und eine Menge von Konfigurationsvariablen $Z_{cf}$ definiert, wobei gilt:

$$\forall z_{cn,i} \in Z_{cn} \Rightarrow \exists z_{cf,i} \in Z_{cf} \quad \wedge \quad \forall z_{cf,i} \in Z_{cf} \Rightarrow \exists z_{cn,i} \in Z_{cn} \tag{4.18}$$

mit $i \in \{1, \ldots, n_{cn}\}$ und $n_{cn}$ der Anzahl der Zählvariablen.

Da sowohl eine Menge von Zähl- und Konfigurationsvariablen für die PrSM als auch für die PSM existiert, definieren wir entsprechend $Z_{cn,prsm}$ und $Z_{cf,prsm}$ für die PrSM und $Z_{cn,psm}$ und $Z_{cf,psm}$ für die PSM.

Ist innerhalb einer funktionalen Komponente der Kontrollpfad abhängig von Eingabedaten, kann dies Einfluss auf die Leistungsaufnahme haben. Des Weiteren kann die Leistungsaufnahme auch von dem Datenpfad abhängen. Dieses Verhalten wird in der Regel nicht in der PrSM modelliert. Damit diese Abhängigkeit in der PSM dargestellt werden kann, werden die **geteilten Zustandsvariablen** genutzt. Dies sind Variablen mit unterschiedlichen Typen, die nur von der PrSM beschrieben werden können, aber von PrSM und PSM gelesen werden können. So ist es möglich, entsprechendes Verhalten sowohl in der PrSM als auch in der PSM darzustellen. Wird z.B. ein datenabhängiger Kontrollpfad modelliert, wird dies in der PrSM umgesetzt, da diese den funktionalen Part des Modells darstellt. Müssen jedoch datenabhängige Leistungsaufnahmen modelliert werden, wird dies in der PSM dargestellt, da hier keine Unterscheidung im Kontrollpfad der Komponente stattfindet und somit dies nur den extra-funktionalen Part – die Leistungsaufnahme – betrifft. Es wird eine Menge für die geteilten Variablen $Z_{sh}$ definiert.

Da die Automaten aus den Definitionen in (4.9) und (4.2) keine Möglichkeit haben, auf die Variablen zuzugreifen oder diese zu verändern, werden diese entsprechend der Definition der EFSMs angepasst. Das bedeutet, dass die Zustandsübergänge durch sogenannte *Aktivierungsfunktionen* $\phi$ und *Aktualisierungsfunktionen* $\alpha$ erweitert werden [30]. Die Aktivierungsfunktionen definieren Prüfbedingungen auf den verschiedenen Zustandsvariablen. Erst wenn diese Prüfbedingung wahr ist, wird der entsprechende Zustandsübergang ausgeführt. Die Aktualisierungsfunktionen hingegen wird ausgeführt, sobald der verknüpfte Zustandsübergang ausgeführt wird. Diese verändert dann Zustandsvariablen – z.B. wird eine Zählvariable inkrementiert oder ein Wert der Eingabe des funktionalen Designs zwischengespeichert.

Die Aktivierungsfunktion für einen Zustandsübergang $e$ ist folgendermaßen definiert:

$$\phi_e : Z \rightarrow \{\texttt{true}, \texttt{false}\} \quad \text{mit} \quad \phi_e \in \Phi \quad , \tag{4.19}$$

wobei $\Phi$ die Menge aller Aktivierungsfunktionen ist und $Z$ die Menge aller Zustandsvariablen. Das bedeutet, in Abhängigkeit der aktuellen Belegung der Zustandsvariablen gibt diese Funktion entweder `true` oder `false` zurück. $Z$ entspricht dabei sowohl den geteilten Variablen $Z_{sh}$ als auch den Zählvariablen des entsprechenden Zustandsautomaten $Z_{cn,prsm}$ oder $Z_{cn,psm}$.

Da die Aktualisierung der Zustandsvariablen abhängig ist von den Eingaben und der vorhandenen Variablenbelegung, werden die Aktualisierungsfunktionen für die PrSM $\alpha_{prsm} \in A_{prsm}$

und für die PSM $\alpha_{psm} \in A_{psm}$ folgendermaßen definiert:

$$\alpha_{prsm} : \Sigma_{prsm} \times Z \rightarrow Z_{prsm} \quad \text{mit} \quad Z_{prsm} = Z_{cn,prsm} \cup Z_{cf,prsm} \cup Z_{cf,psm} \cup Z_{sh} \quad \text{und} \tag{4.20}$$

$$\alpha_{psm} : \Sigma_{psm} \times Z \rightarrow Z_{cn,psm} \quad . \tag{4.21}$$

Bei der PSM werden nur die Zählvariablen $Z_{cn,psm}$ geändert, da diese keine Änderung auf den geteilten Variablen $Z_{sh}$ durchführen kann. Die PrSM hingegen, kann ihre Zählvariablen $Z_{cn,prsm}$, alle Konfigurationsvariablen $Z_{cf,prsm}, Z_{cf,psm}$ und die geteilten Variablen $Z_{sh}$ ändern.

Dadurch ändert sich die Übergangsfunktion der PrSM und analog für die PSM wie folgt:

$$E_{prsm} = S_{prsm} \times \Sigma_{prsm} \times \Phi_{prsm} \times A_{prsm} \times S_{prsm} \quad \text{und} \tag{4.22}$$

$$E_{psm} = S_{psm} \times \Sigma_{psm} \times \Phi_{psm} \times A_{psm} \times S_{psm} \quad . \tag{4.23}$$

Die Automatendefinitionen erweitern sich wie folgt:

$$prsm := (S_{prsm}, s_{0,prsm}, \Sigma_{prsm}, \Gamma_{prsm}, E_{prsm}, Z) \quad \text{und} \tag{4.24}$$

$$psm := (S_{psm}, s_{0,psm}, \Sigma_{psm}, \Gamma_{psm}, E_{psm}, Z) \quad . \tag{4.25}$$

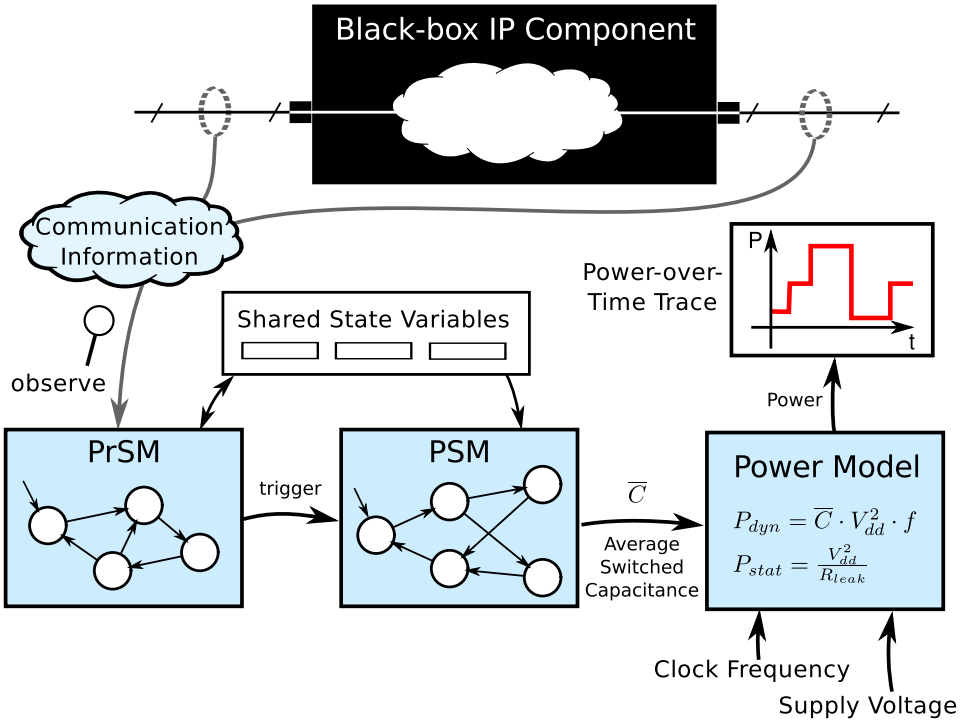



Abbildung 4.10.: PSM-Modell mit Erweiterung von Zustandsvariablen.

Dadurch ergibt sich das in Abbildung 4.10 gezeigte Modell für die Modellierung der Leis-

tungsaufnahme. Dort ist zu sehen, dass es einen Container für die Verwaltung der geteilten Variablen gibt. Dieser speichert auch die Konfigurationsvariablen und regelt den Zugriff auf die Zustandsvariablen, die nur von der PrSM verändert, aber von beiden Zustandsautomaten gelesen werden können. Ein Beispiel für den Einsatz von Zustandsvariablen wird später in diesem Kapitel in Abbildung 4.16 und Abbildung 4.17 gezeigt.

## 4.7. Zeitgesteuerte Zustandsübergänge

In den vergangenen Abschnitten wurde ein Modell entwickelt, welches es ermöglicht, durch Beobachtung der Schnittstelleninteraktionen einer funktionalen Komponente, eine Leistungsaufnahme abzuschätzen. Eine Änderung der Leistungsaufnahme in der PSM erfolgt nur bei einer Aktion auf den Schnittstellen der funktionalen Komponente. Einige Komponenten sind in der Lage, ohne Schnittstellenaktion ihren funktionalen Zustand zu ändern. Führt diese Änderung auch zu einer Änderung der Leistungsaufnahme, ist die PSM nicht in der Lage dies zu erkennen und es wird eine abweichende Leistungsaufnahme modelliert.

Oftmals passieren diese Änderungen nicht sporadisch, sondern entweder nach einer definierten Zeit oder nach einer definierten Anzahl an Aktionen, die fest konfiguriert oder durch eine veränderbare Konfiguration der Komponente festgelegt ist. Dies entspricht einem deterministischen zeitlichen Verhalten und kann mit einer Erweiterung des Modells entsprechend dargestellt werden.

Zur Erweiterung wurden in dieser Arbeit dafür die sogenannten *Event-Clock Automata* (ECAs) [3] gewählt. Diese sind *Timed Automata* mit der Einschränkung, dass es zu jedem Eingangssymbol genau eine Uhr gibt und damit jede Uhr eine feste Assoziation mit genau einem Eingabesymbol hat. Die ECAs sind durch ihre Eigenschaften ein oft eingesetztes und gut geprüftes Modell zur Darstellung von Echtzeitsystemen [13, 23]. Sie unterteilen sich in zwei Kategorien: *Event-Recording Automata* (ERAs) und *Event-Predicting Automata* (EPAs). Bei ERAs werden die Uhren bei dem Empfang des zugehörigen Symbols auf 0 gesetzt und in den Zustandsübergängen auf einen bestimmten Wert geprüft. Dahingegen wird bei EPAs der Wert auf einen nichtdeterministischen negativen Wert gesetzt und bei dem nächsten Auftreten dieses Symbols auf den Wert 0 geprüft.

Daraus wird ersichtlich, dass EPAs nur dann genutzt werden können, wenn die Eingabesprache inklusive ihres zeitlichen Verhaltens vollständig bekannt ist. Dies trifft nicht zu für die Synchronisation zwischen PrSM und PSM mittels der PSM-Events, da das zeitliche Verhalten aus der funktionalen Ausführung abgeleitet wird, bei dem nicht alle Eingaben mit einer speziellen Zeitvorgabe versehen sind. Aus diesem Grund werden in dieser Arbeit nur die ERAs benutzt.

Ein Event-Recording Automaton ist ein Zustandsautomat, bei dem für jedes Eingangssymbol $a$ eine eigene *Event-Recording* Uhr $x_a \in C$ existiert, die die Zeit seit dem letzten Empfang des Symbols $a$ zählt. Gegeben sei das mit Zeit annotierte Eingabewort

$\overline{w} = (a_0, t_0)(a_1, t_1)\ldots(a_n, t_n)$, dann gibt der Wert von $x_a$ an der Position $j$ vom Wort $\overline{w}$ die Differenz $a_j - a_i$ an, wobei $i$ die höchste Position aller Vorgänger von $j$ angibt. Gibt es keinen Vorgänger, so ist der Wert *undefiniert*.

ECAs sind nicht-deterministische Zustandsautomaten, deren Zustandsübergänge mit einem Eingabesymbol und einer Bedingung über die *Event-Predicting* und *Event-Recording* Uhren annotiert sind. Des Weiteren sind die Zustände mit Invarianten versehen, die anzeigen, ob ein Zustandswechsel erfolgen muss oder der Zustand weiterhin aktiv bleiben kann.

In dieser Arbeit wird der ERA-Mechanismus nur auf die PSM angewendet, da die PrSM weiterhin nur das auf den Schnittstellen sichtbare Verhalten darstellen soll. So bleibt weiterhin eine strikte Trennung zwischen Funktion und funktionaler Modellierung der Leistungsaufnahme erhalten.

Dadurch ändert sich die Menge der Aktivierungsfunktionen $\Phi_{psm}$ in der PSM, die nun neben der Belegung der Zustandsvariablen auch die der Uhren prüft. Für jeden Zustandsübergang $e \in E$ gibt es eine Aktivierungsfunktion $\phi_e$ mit folgender Eigenschaft:

$$\phi_e : Z \times C \rightarrow \{\texttt{true}, \texttt{false}\} \quad . \tag{4.26}$$

Für jeden Zustand gibt es eine Invariante aus $Inv$, die bestimmt, wie lange ein Zustand aktiv bleiben kann. Dies passiert in Abhängigkeit der Uhren. Damit erweitert sich die Beschreibung der PSM auf:

$$psm := (S_{psm}, s_{0,psm}, \Sigma_{psm}, \Gamma_{psm}, E_{psm}, Z, Inv) \quad . \tag{4.27}$$

Die Modellierung als ERA kann nun dafür genutzt werden, abhängig von den Werten der Uhren unterschiedliche Zustandsübergänge in der PSM zu modellieren oder Zustandsübergänge nach einer definierten Zeit zu erzwingen. Wichtig dabei ist, dass stets ein deterministischer Automat erzeugt wird oder der nicht-deterministische Automat bereits in einen deterministischen überführt wurde.

Dies wird ermöglicht, indem ein Zustandsübergang $e$ gebildet wird mit dem leeren Wort $\epsilon$ für das Eingabesymbol und einer Aktivierung bei einem bestimmten Zeitwert $t$:

$$s_i \times \epsilon \times \phi_e \times s_j \quad . \tag{4.28}$$

Gegeben sei die Aktivierungsfunktion $\phi_e \Leftrightarrow x = t$ mit $x \in C$ und $t$ der Zeit des Zustandswechsels, dann muss folgende Bedingung erfüllt sein:

$$Inv(s_i) \Leftrightarrow x \leq t \quad . \tag{4.29}$$

Das bedeutet, zu dem Zeitpunkt des zeitgesteuerten Zustandswechsels muss die Invariante des Zustands ungültig werden, damit ein Zustandswechsel erzwungen wird. Wichtig ist,

dass es zu jedem Zustand maximal einen zeitgesteuerter Zustandsübergang gibt, da der Automat sonst nicht deterministisch ist.

Benini [6] hat bei seiner vorgestellten PSM die Möglichkeit vorgesehen, dass für Zustandsübergänge eine Energieaufnahme definiert werden kann. Die hier vorgestellte PSM bietet diese Möglichkeit bisher nicht. Mit den zeitgesteuerten Zustandsübergängen kann dieses Verhalten jedoch nachgebildet werden. Dafür wird zusätzlich ein Zustand mit einer Invariante und ein zeitgesteuerter Zustandsübergang benötigt. Der Grund dafür ist, dass eine Energieaufnahme nur mit einem Zeitintervall gekoppelt werden kann. Ohne Zeitintervall wäre die Leistungsaufnahme unendlich hoch. Des Weiteren wird der Zielzustand des ursprünglichen Zustandsübergangs auf den neuen Zustand geändert und ein neuer zeitgesteuerter Zustandsübergang eingefügt. Dies soll an einem Beispiel gezeigt werden:

Gegeben sei ein Zustandsübergang $(s_i, a, \texttt{true}, \epsilon, s_j)$, der eine Energieaufnahme modellieren soll. Der Zustandsübergang soll ein Zeitintervall $t$ haben.

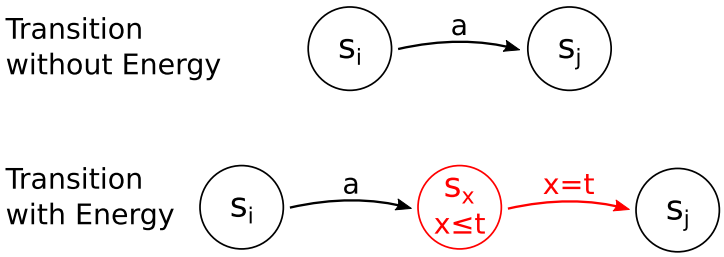



Abbildung 4.11.: Änderung eines Zustandsübergangs zur Modellierung einer Energieaufnahme mit Hilfe des ERA-Mechanismus.

Es wird der neue Zustand $s_x$ mit der Invariante $Inv(s_x) \Leftrightarrow x \leq t$ eingefügt und der bisherige Zustandsübergang wie folgt geändert: $(s_i, a, \texttt{true}, \epsilon, s_x)$ und der neue Zustandsübergang angelegt: $(s_x, \epsilon, x = t, \epsilon, s_j)$. Die Änderung ist in Abbildung 4.11 gezeigt. Ein Beispiel für einen zeitgesteuerten Zustandsübergang ist in Abbildung 4.12 gezeigt. Bei diesem kann der Übergang von `run` zu `idle` nicht aus der Schnittstellenkommunikation abgeleitet werden. Er weißt aber ein deterministisches, verzögertes Verhalten von 0,3 ms zum Übergang von `idle` zu `run` auf und wird entsprechend durch einen zeitgesteuerten Zustandsübergang abgebildet. Die Energie $W$ des Zustandsübergangs berechnet sich dann durch die Leistungsaufnahme des zusätzlichen Zustands $P$ und die Zeit $t$:

$$W = P \cdot t \quad . \tag{4.30}$$

ECAs haben im Gegensatz zu *Timed Automata* den Vorteil, dass sie in deterministische Automaten überführbar sind, da ihr komplettes Verhalten von dem Eingabestrom abhängt. Diese Eigenschaft wird benötigt, weil das Modell in einer Simulation ausführbar sein soll. Dies ist nur möglich, wenn die Automaten im PSM-Modell deterministisch sind.

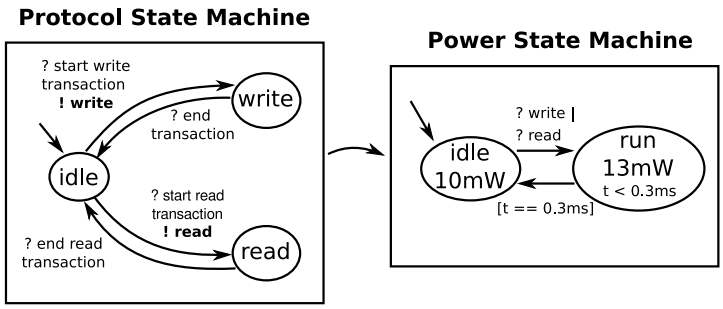



Abbildung 4.12.: Beispiel für zeitgesteuerten Zustandsübergang anhand von einer Speicherkomponente auf ESL. Das Ende einer Operation ist eine deterministische zeitlich verzögerte Reaktion auf den Beginn der Operation von 0,3 ms.

Mit der Modellierung der PSM als ERA erfüllt das Modell Anforderung F4 zur Betrachtung von nicht detektierbaren Zustandsänderungen des funktionalen Modells.

## 4.8. Datenabhängige Leistungsaufnahme

Das in den vorherigen Abschnitten vorgestellte Modell ist in der Lage, die durchschnittliche Leistungsaufnahmen abhängig vom gegenwärtigen Kontrollzustand der funktionalen Komponente zu simulieren. Dies führt in vielen Fällen zu einem ausreichend geringen Fehler, um einen Vergleich von verschiedenen Systemkonfigurationen in Bezug auf die Leistungsaufnahme durchführen zu können. Einige Komponenten weisen auch eine stark datenabhängige Leistungsaufnahme auf. Der Grund dafür ist, dass die Schaltaktivität innerhalb der Komponente größtenteils von den Änderungen der Datenleitungen an den Eingängen abhängt. Siehe dafür die Evaluation der Speicherkomponente in Abschnitt 6.2.2. Weiterhin können diese Änderungen am Eingang die Leistungsaufnahme für einen längeren Zeitraum beeinflussen, da sie in mehreren Schritten verarbeitet oder zwischengespeichert werden. Das bedeutet, dass mehrere Änderungen hintereinander in einem gemeinsamen Zeitraum Einfluss auf die Leistungsaufnahme haben können. Wird dieses Verhalten nicht betrachtet, kann dies zu einem signifikanten Fehler führen. Wie in Anforderung F5 gefordert, wird in diesem Abschnitt beschrieben, wie eine Betrachtung der datenabhängigen Leistungsaufnahme mit in das bestehende PSM-Modell integriert werden kann. Dafür wird nachfolgend definiert, was unter der datenabhängigen Leistungsaufnahme zu verstehen ist:

**Datenabhängige Leistungsaufnahme** Die datenabhängiger Leistungsaufnahme beschreibt den Teil der dynamischen Leistungsaufnahme, der entsteht, wenn Signal- oder Datenleitungen die Nutzdaten weiterleiten und nicht verantwortlich sind für Kontrollzustandswechsel einer Komponente.

Es gibt verschiedene Möglichkeiten, datenabhängige Leistungsaufnahme zu modellieren, die im Folgenden vorgestellt werden sollen. In Abschnitt 5.4.2 wird beschrieben, wie die Parameter für die Modellierung der datenabhängigen Leistungsaufnahme charakterisiert werden, um ein möglichst genaues Modell zu erstellen.

Einige der vorgestellten Möglichkeiten nutzen eine Auswertung der datenabhängigen Leistungsaufnahme. Da dynamische Leistung immer dann aufgenommen wird, wenn Signale ihren Wert wechseln, entsteht datenabhängige Leistungsaufnahme nur bei einer Änderung der Werte auf den Eingangs- oder Ausgangssignale der Nutzdaten. Aus diesem Grund werden für die datenabhängige Leistungsaufnahme immer zwei aufeinander folgende Datenvektoren betrachtet. Durch eine spezielle Funktion, die im folgenden als *Differenzmetrik* bezeichnet wird, wird der Unterschied zwischen beiden Vektoren bewertet. Die für diesen Anwendungsfall am häufigsten eingesetzte Metrik ist die sogenannte *Hamming Distance* (HD), da sie genau die Anzahl der schaltenden Signale beschreibt.

**Hamming Distance** Die Hamming Distance beschreibt für zwei Vektoren der gleichen Größe die Anzahl der Positionen, an denen sich das entsprechende Symbol zwischen den beiden Vektoren unterscheidet.

Der Wertebereich der Differenzmetrik entspricht also dem ganzzahligen Bereich zwischen 0 und der Anzahl der Signalwerte (die Bitbreite) für einen Datenvektor. Hat ein Datenwert beispielsweise eine Bitbreite von 16 Bit (entsprechend 16 Signalleitungen), ist eine ganzzahlige HD zwischen 0 und 16 möglich. Um eine von der Bitbreite unabhängige Betrachtung der HD zu ermöglichen, wird diese häufig in der normalisierten Form $HD_n$ verwendet. Dabei wird die Hamming Distance $HD$ durch die entsprechende Bitbreite des Datenvektors $BW$ geteilt, wodurch sich ein rationaler Wert zwischen 0 und 1 ergibt:

$$HD_n = \frac{HD}{BW} = \{0, \ldots, 1\} \in \mathbb{Q} \quad . \tag{4.31}$$

Nachfolgend werden drei verschiedene Möglichkeiten aufgezeigt, diese Differenzmetrik in das PSM-Modell zu integrieren, wobei Vorteile und Nachteile der einzelnen Konzepte aufgezeigt werden. Bei dem ersten Konzept wird der Durchschnitt der datenabhängigen Leistungsaufnahme berechnet und an die Zustände annotiert. Im zweiten Konzept wird der HD-Raum in mehrere Bereiche unterteilt, dessen Durchschnitt berechnet wird. Bei der dritten Möglichkeit wird versucht, eine möglichst genaue Modellierung der datenabhängigen Leistungsaufnahme vorzunehmen, indem der vollständige HD-Bereich betrachtet wird und nicht nur der letzte Wechsel der Dateneingänge, sondern auch zeitlich davor liegende Wechsel. Diese Erweiterungen des Modells erfüllen damit Anforderung F5.

### 4.8.1. Durchschnittliche datenabhängige Leistungsaufnahme

Das Ziel dieser Modellierung ist, die durchschnittliche datenabhängige Leistungsaufnahme zu berechnen und den entsprechenden Wert der durchschnittlich geschalteten Kapazität auf den bisherigen Ausgabewert der PSM-Zustände zu addieren. Dadurch bleibt die PSM, abgesehen von den Ausgangswerten, unverändert und verliert auch nicht an Simulationsperformanz. Da keine weitere Differenzierung für die Datenabhängigkeiten besteht, führt dies im Vergleich mit den anderen beiden Möglichkeiten zum ungenausten Modell. Diese Art der Modellierung passiert implizit bei der Erstellung der PSM ohne Betrachtung der datenabhängigen Leistungsaufnahme, wenn zur Charakterisierung der Leistungsaufnahme Stimuliwerte eingesetzt werden, bei denen die Nutzdaten gemäß des späteren Anwendungsfalls ihre Werte verändern und dadurch eine datenabhängige Leistungsaufnahme herbeiführen. Formal lässt sich die datenabhängige Ausgabe $\gamma_{data}$ wie folgt ausdrücken:

$$\gamma_{data} = c + \sum_{i=1}^{n} \frac{1}{m} \sum_{j=1}^{m} x_i(d_i(y_{ij}, y_{i(j+1)})), \gamma_{data} \in \Gamma_{data} \quad , \tag{4.32}$$

wobei $n$ die Anzahl der betrachteten Ports, $m$ die Anzahl der aufeinanderfolgend betrachteten Datenwerte $y_{ij}$ und $y_{i(i+1)}$, $x_i$ ein Skalierungsfaktor und $d_i$ die Differenzmetrik für die Daten auf Port $i$ ist. Der Parameter $c$ beschreibt einen Konstantanteil, der die kontrollflussdominierte Leistungsaufnahme darstellt. Diese ist in der Regel auf die Aktivität des Taktsignals und deren Auswirkungen zurückzuführen. Ein Datenwert $y_j$ ist Teil eines Eingabesymbols $v_P$, vgl. (4.7):

$$y_j \subseteq v_P \quad \text{mit} \quad i = (t, v_P) \quad . \tag{4.33}$$

Da neben den datenabhängigen Ausgaben $\Gamma_{data}$ weiterhin zustandsabhängige $\Gamma_{state}$ erlaubt sind, ist die Menge der Ausgabesymbole $\Gamma_{psm}$ wie folgt definiert:

$$\Gamma_{psm} = \Gamma_{state} \cup \Gamma_{data} \quad . \tag{4.34}$$

$\Gamma_{state}$ ist weiterhin entsprechend (4.5) definiert.

In Abbildung 4.13 ist ein Beispiel gezeigt, wie sich die Modellierung auf die Leistungsaufnahme und die Energieaufnahme auswirken kann. Die verwendete Komponente ist der Speicher aus Abbildung 4.14, der über folgende Ein- und Ausgänge verfügt:

- `Clock` (Eingang): Taktsignal für die Komponente,
- `Reset` (Eingang): Rücksetzsignal,
- `Chip Enable` (Eingang): Kontrollsignal, ob eine Operation ausgeführt werden soll oder die Komponente im Ruhezustand sein soll,
- `Write Enable` (Eingang): Kontrollsignal, ob geschrieben oder gelesen werden soll,
- `Data In` (Eingang): Dateneingang für die zu speichernden Datenvektoren,

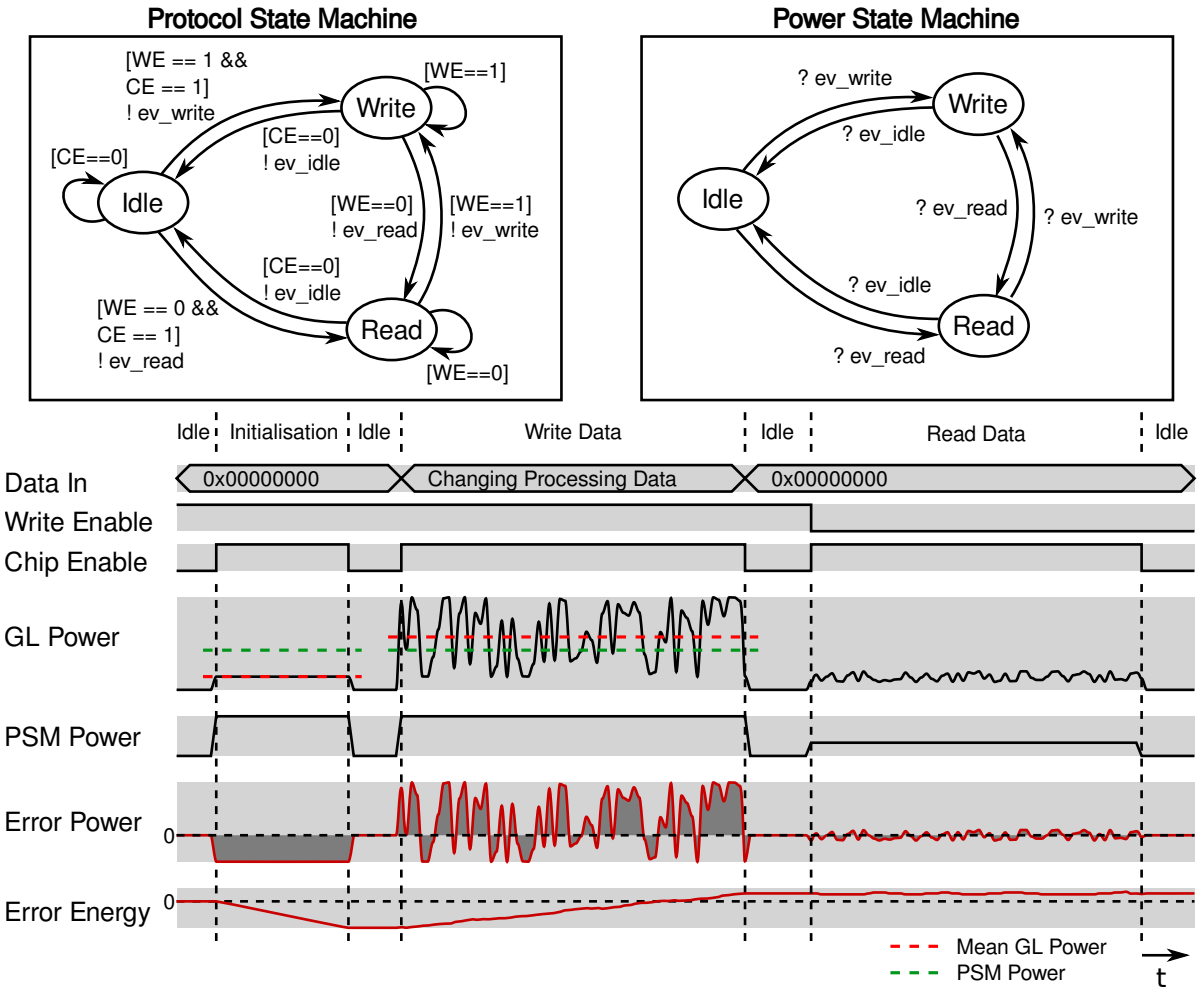



Abbildung 4.13.: Beispiel für die Modellierung der durchschnittlichen datenabhängigen Leistungsaufnahme für eine Speicherkomponente. Die simulierte Leistungs- und Energieaufnahme weicht abhängig von den Daten auf den Ports signifikant von der tatsächlichen Leistungs- und Energieaufnahme ab.

- `Address` (Eingang): Adresse, auf die geschrieben oder von der gelesen wird und
- `Data Out` (Ausgang): Datenausgang für die ausgelesenen Datenvektoren.

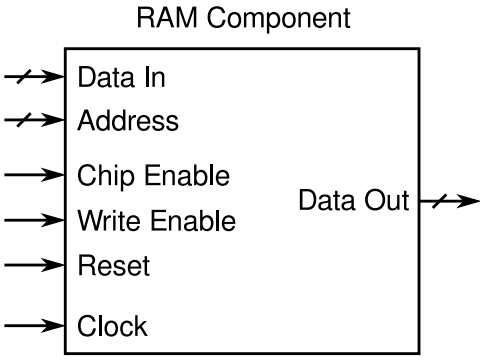



Abbildung 4.14.: Blockschaltbild einer Speicherkomponente.

Da durch die Kontrollsignale `Chip Enable` und `Write Enable` nur unterschieden werden kann, ob der Speicher gerade beschrieben wird, von ihm gelesen wird oder die Komponente im Ruhezustand ist, entsteht bei unterschiedlichen Daten ein signifikanter Fehler durch die datenabhängige Leistungsaufnahme im PSM-Zustand `Write`. Hier im Speziellen durch eine Initialisierungsphase, bei der gar keine Schaltaktivität beim Datenport `Data In` vorhanden ist, und zusätzlich durch eine für andere Anwendungsfälle charakterisierte durchschnittliche Leistungsaufnahme für den PSM-Zustand `Write`. Dieser führt im weiteren Verlauf der Simulation zu einer deutlichen Abweichung der aufgenommenen Energie.

### 4.8.2. Bereichsweise datenabhängige Leistungsaufnahme

Die zweite Möglichkeit ist, den Wertebereich der Datenabhängigkeiten in definierte **Unterbereiche** zu unterteilen und diese separat zu charakterisieren. Für jeden Unterbereich gibt es einen eigenen PSM-Zustand, der diesen Bereich darstellt. In diesem Fall wird für jeden Bereich ein Durchschnittswert ermittelt. Nimmt man beispielsweise drei Bereiche für eine Bitbreite von 8 Bit und der HD als Differenzmetrik, ergibt sich eine Verteilung wie in Tabelle 4.1 gezeigt.

Tabelle 4.1.: Beispiel für Unterteilung des Differenzmetrikwertebereichs in drei Unterbereiche für eine Bitbreite von 8 Bit.

| HD | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Bereich | | 1 | | | 2 | | | 3 | |

Gibt es mehrere relevante Datenports, die eine Änderung der Leistungsaufnahme herbeiführen, müssen diese in Kombination betrachtet werden. Nimmt man beispielsweise zwei Dateneingänge mit einer Bitbreite von 8 Bit, der HD als Differenzmetrik und unterteilt

diese wieder jeweils in drei Bereiche, ergeben sich durch die Kombination dieser Bereiche jeweils neun Bereiche, die betrachtet werden müssen. Dadurch, dass jeder Bereich separat charakterisiert werden muss, erhöht sich die notwendige Zeit für eine vollständige Charakterisierung enorm. Daher eignet sich diese Methode besser für wenige Bereiche und möglichst wenig betrachtete Datenports.

Im Gegensatz zur vorherigen Modellierungsmethode muss hier das PSM-Modell verändert werden. Da auf Änderungen der Datenvektoren zugegriffen wird, müssen diese Änderungen bzw. die dafür verantwortlichen Datenvektoren für die PSM zugreifbar sein. Dies wird dadurch erreicht, dass für jeden betrachteten Datenport zwei Zustandsvariablen angelegt werden. Diese speichern den aktuellen und letzten Wert für den jeweiligen Datenport. Die PrSM ist dafür verantwortlich, diesen Zustandsvariablen die entsprechenden Werte zuzuordnen. Dies wird, wie in Abschnitt 4.6 beschrieben, durch Aktivierungsfunktionen umgesetzt, die an den nötigen Zustandsübergängen der PrSM annotiert sind. Wichtig hierbei ist, dass alle relevanten Änderungen der entsprechenden Datenports durch einen Zustandsübergang in der PrSM zu einer Aktualisierung der Zustandsvariablen führen, damit die Berechnung der Differenzmetrik nicht auf veralteten Daten durchgeführt wird.

In der PSM muss jeder Zustand, der ein datenabhängiges Leistungsaufnahmeverhalten aufweist, ersetzt werden durch so viele Zustände, wie Bereiche für den Wertebereich der Differenzmetrik definiert wurden. Das heißt, nimmt man das Beispiel aus Tabelle 4.1, wird hier jeder Zustand mit datenabhängiger Leistungsaufnahme durch drei Zustände ersetzt. Jeder entspricht dabei einem Unterbereich des Differenzmetrikwertebereichs. Die Zustandsübergange müssen analog zum Zustand ersetzt werden. Die Übergangsbedingungen der Eingangszustandsübergänge werden mit den Bedingungen zur Überprüfung des Differenzmetrikwertebereichs erweitert. Alle anderen Eigenschaften der Zustandsübergänge werden ohne Änderung übernommen (z.B. die Aktualisierungsfunktionen). Ein Beispiel für eine solche Anpassung ist in Abbildung 4.15 gezeigt. Hier wurde wieder das Beispiel aus Tabelle 4.1 verwendet. Es wird deutlich, dass der Zustand S3 durch die drei Zustände S3_1, S3_2 und S3_3 ersetzt wird, die die drei Bereiche des Differenzmetrikwertebereichs darstellen. Analog wurden die Eingangs- und Ausgangszustandsübergänge ersetzt, wobei die Eingangszustandsübergänge (mit den Triggerevents a und b) durch die entsprechenden Prüfbedingungen für die Differenzmetrikwertebereiche erweitert wurden.

Diese Art der Modellierung ermöglicht eine genauere Berechnung der datenabhängigen Leistungsaufnahme auf Kosten der Übersichtlichkeit, da nun für jeden Zustand, der eine datenabhängige Leistungsaufnahme aufweist, mehrere Zustände existieren. Des Weiteren wird die Simulationsperformanz negativ beeinflusst, da zusätzlicher Aufwand nötig ist, um die Berechnung für die Differenzmetrik durchzuführen und abhängig von dieser die Bedingungen an den Zustandsübergängen auszuwerten.

Formal lässt sich die Ausgabe eines Zustands wie folgt darstellen. Es wird die Formel aus (4.32) für die Modellierung der durchschnittlichen Leistungsaufnahme genommen mit der Einschränkung der Datenwerte, sodass nur die ausgewählten Bereiche $Q_i$ der

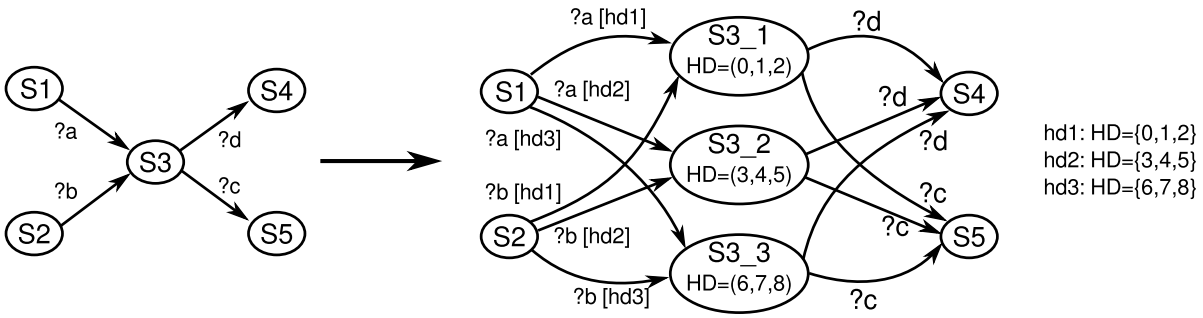



Abbildung 4.15.: Änderung eines PSM-Zustands zur Modellierung von datenabhängiger Leistungsaufnahme mit Unterbereichen im Wertebereich der Differenzmetrik. Entsprechend der Anzahl von Bereichen werden Zustände erstellt und die neuen Zustandsübergänge durch Bedingung zum Prüfen der Differenzmetrik erweitert.

Differenzmetrik $d_i$ für Port $i$ betrachtet werden:

$$\gamma_{data} = c + \sum_{i=1}^{n} \frac{1}{m} \sum_{j=1}^{m} x_i(d_i(y_{ij}, y_{i(j+1)})) \quad \text{mit} \quad d_i(y_{ij}, y_{i(j+1)}) \in Q_i \quad . \tag{4.35}$$

Diese Modellierungsmöglichkeit wird anhand des Beispiels aus Abbildung 4.13 in Abbildung 4.16 gezeigt. Es wurden drei Bereiche gewählt für den PSM-Zustand `Write`, wodurch sich die drei Zustände `Write_1`, `Write_2` und `Write_3` ergeben, die durch die HD charakterisierte Mittelwerte für den HD-Bereich als Ausgabe haben. Zusätzlich müssen die Zustandsvariablen für die Zwischenspeicherung der Eingangsvektoren angelegt werden und die Zustandsübergänge der PrSM mit den nötigen Aktualisierungsfunktionen erweitert werden, damit die Zustandsvariablen immer die aktuellen Werte enthalten (diese Aktualisierungsfunktionen werden aufgrund der Übersichtlichkeit in Abbildung 4.16 nicht gezeigt).

Es wird deutlich, dass der durchschnittliche Fehler im Vergleich zum vorherigen Modell deutlich gesenkt werden kann. Durch spezielles funktionales Verhalten, z.B. die Initialisierungsphase, kann aber auch ein größerer Fehler entstehen, der sich negativ auf die aufgenommene Energieabschätzung auswirkt. Im Mittel gibt es aber deutlich weniger Variation als bei einer Betrachtung eines durchschnittlichen Wertes. Für den Zustand `Read` wurde keine Unterteilung vorgenommen, da die Varianz deutlich kleiner ist und der durchschnittliche Fehler im Bereich das Abstraktionsfehlers liegt. Es ist zu erkennen, dass allein durch drei Zustände für die verschiedene datenabhängige Leistung die Komplexität drastisch ansteigt.

In einigen funktionalen Modellen werden Kommunikationsmechanismen wie der sogenannte *Burst* eingesetzt. Hier wird eine Abfolge von Datenübertragungszyklen über den Kommunikationsbus gesendet, ohne diese durch Busprotokollnachrichten zu unterbrechen. Dies sorgt für einen hohen Durchsatz für sequentielle Daten. Jeder Zyklus wird als *Burst-Tick*

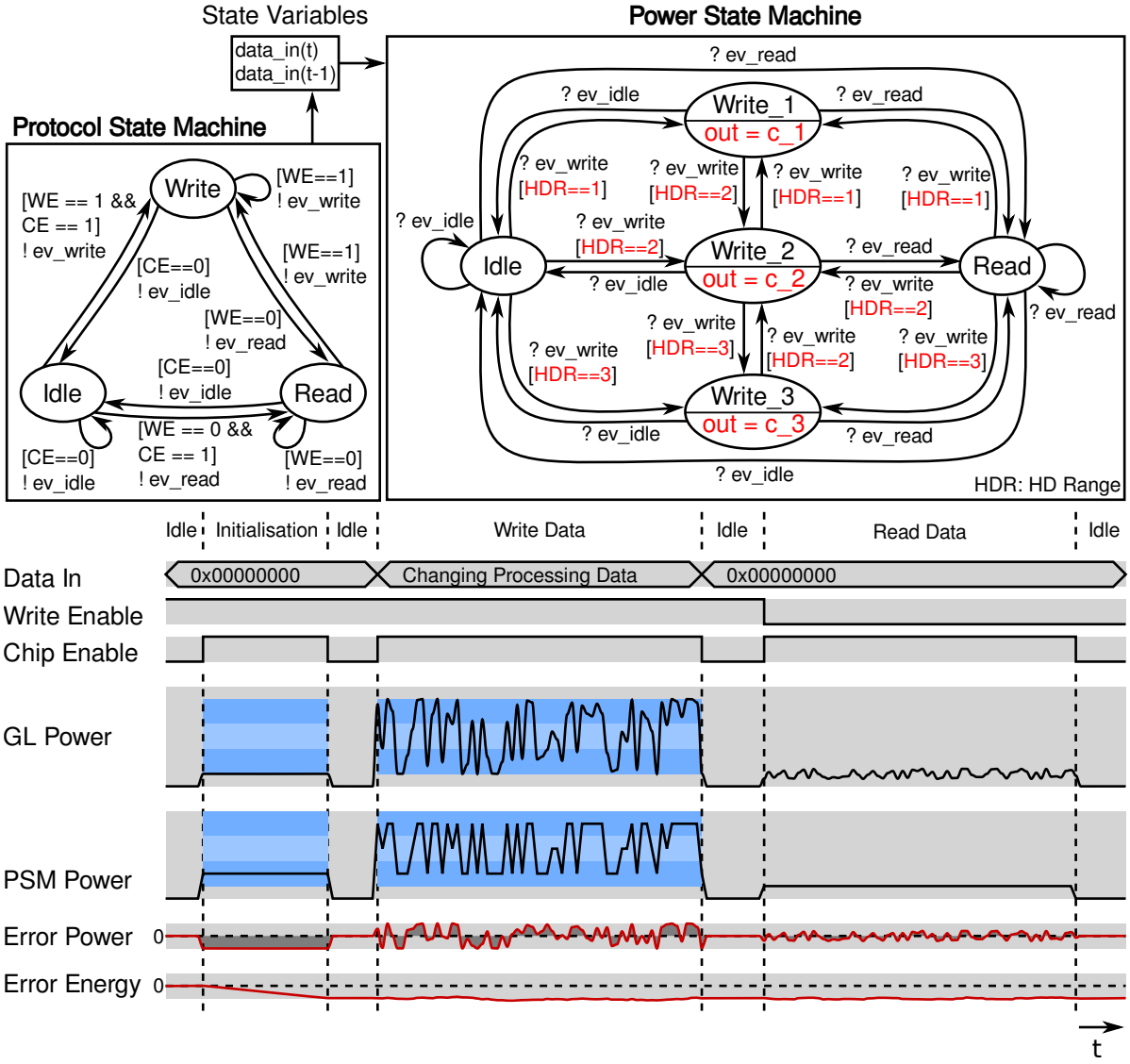



Abbildung 4.16.: Beispiel für die Modellierung der bereichsweisen datenabhängigen Leistungsaufnahme für eine Speicherkomponente. Annäherung der Leistungsaufnahme basierend auf drei Bereichen, die separat charakterisiert wurden. Eine Erweiterung des PSM-Modells mit Zustandsvariablen ist erforderlich, damit die drei Bereiche dargestellt werden können.

bezeichnet. In abstrakten funktionalen Modellierungen wie z.B. TLM werden *Bursts* als eine einzige Übertragungstransaktion dargestellt. Hierbei ergibt sich das Problem, dass in einem *Burst* mehrere Datenänderungen geschehen, welche nicht durch die in diesem Abschnitt vorgestellte Modellierung der datenabhängigen Leistungsaufnahme dargestellt werden kann. Der Grund dafür ist, dass für eine *Burst*-Transaktion lediglich ein Eingabesymbol in der PrSM verarbeitet wird und dementsprechend auch nur ein PSM-Event für die PSM erzeugt wird. Dies führt zu maximal einem Zustandsübergang in der PSM. Da die verschiedenen HD-Bereiche auf mehrere Zustände aufgeteilt sind, wären auch mehrere Zustandsübergange zwischen diesen Zuständen nötig, um die einzelnen Leistungsaufnahmen der *Burst-Ticks* darzustellen. Die im nächsten Abschnitt vorgestellte Erweiterung gibt jedoch die Möglichkeit, diesen Aspekt mit zu betrachten.

### 4.8.3. Vollständige datenabhängige Leistungsaufnahme

Sind die bisherigen Möglichkeiten der Modellierung von datenabhängiger Leistungsaufnahme noch nicht ausreichend genau, kann die im Folgenden beschriebene dritte Methode Abhilfe schaffen. Im Gegensatz zur vorherigen Methode wird hier nicht einem Unterbereich in dem Wertebereich der Differenzmetrik ein spezifischer Wert zugeordnet, sondern jedem einzelnen Wert. Damit wird die Auflösung feingranularer und genauer. Wie bei der vorherigen Möglichkeit müsste nun für jeden Wert der Differenzmetrik ein separater Zustand erstellt werden und bei der kombinierten Betrachtung von mehreren Datenports das Produkt der Werte. Dies kann zu einer Zustandsexplosion führen und ist somit praktisch nicht mehr handhabbar. Um diesem entgegenzuwirken wird stattdessen nur die Ausgabe der Zustände verändert. Anstatt einen konstanten Ausgabewert bekommt der Zustand einen dynamischen Ausgabewert. Dieser Ausgabewert ist eine Funktion, die abhängig ist von den Werten der Zustandsvariablen. Somit entspricht diese Änderung nur einer konzeptionellen Änderung, aber keiner semantischen. Jeder betrachteten Kombination von Zustandsvariablen wird ein eindeutiger Wert zugeordnet, den die Ausgabefunktion berechnet. Anstatt die Ausgabefunktion zu nutzen, könnte für jeden entsprechenden Ausgabewert ein Zustand erstellt werden. Das bedeutet, durch die Anwendung der Ausgabefunktion kann die bisherige Automatengröße beibehalten, aber ein größerer Zustandsraum dargestellt werden. Entsprechend dieser Anpassung ändert sich die Definition der Ausgabe für die datenabhängigen PSM-Zustände aus (4.32) wie folgt:

$$\gamma_{data} : Z \rightarrow \{\gamma \,|\, \gamma \in \mathbb{R}_{\geq 0}\}^n \,|\, n \in \mathbb{N} \quad . \tag{4.36}$$

Ein Beweis für die Äquivalenz zwischen PSMs mit dynamischer und konstanter Ausgabe wird in Abschnitt 4.8.4 geführt.

Wie am Anfang dieses Abschnitts beschrieben wurde, können Änderungen an Datenports für einen längeren Zeitraum die Leistungsaufnahme beeinflussen und auch mehrere hintereinander kommende Änderungen können einen gleichzeitigen Einfluss auf die Leistungsaufnahme haben. Aus diesem Grund betrachtet diese Methode nicht nur aktuelle Änderungen

auf den Datenports, sondern auch vergangene. Wie viele vergangene Daten betrachtet werden müssen, hängt dabei von dem verwendeten Design ab.

Hat man beispielsweise ein Design, bei dem die letzten drei Änderungen auf den Datenports einen Einfluss auf die aktuelle Leistungsaufnahme für einen bestimmten Zustand haben, kann dies bedeuten, dass diese Leistungsaufnahmen auf drei verschiedene Verarbeitungsstufen zurückzuführen sind, die gleichzeitig Berechnungen durchführen. Diese Verarbeitungsstufen können unterschiedliche Berechnungen ausführen und daher auch einen unterschiedlichen Einfluss auf die Leistungsaufnahme haben.

Durch die Unterschiede in den Verarbeitungsstufen kann es sein, dass, um zu einer genauen Modellierung zu gelangen, für jede Verarbeitungsstufe eine andere Differenzmetrik angewendet werden muss. In den meisten Fällen ist aber auch hier die HD die beste Wahl.

Des Weiteren kann sich die Abhängigkeit zwischen Leistungsaufnahme und Differenzmetrik unterscheiden, ist aber in den meisten Fällen linear. Je nach Design und Verarbeitungskette kann aber auch beispielsweise eine quadratische oder exponentielle Abhängigkeit vorhanden sein.

Gibt es mehrere Datenports, kann jeder von diesen einen Einfluss auf die Leistungsaufnahme haben. Das bedeutet auch, dass Änderungen an den entsprechenden Ports einen gleichzeitigen und kombinierten Einfluss haben können. Daher ist es wichtig, dass alle Datenports betrachtet werden, die einen Einfluss auf die Leistungsaufnahme haben.

Tabelle 4.2.: Beispiel für die Variablenbelegung von drei Datenports mit unterschiedlicher langer Betrachtung der Historie. $t$ beschreibt den aktuellen Wert und $t-1$ den letzten (anderen) und nicht den Wert zur letzten Zeitinstanz.

| Zeit | Datenport 1 | Datenport 2 | Datenport 3 |
|---|---|---|---|
| $t$ | $y_{11}$ | $y_{21}$ | $y_{31}$ |
| $t-1$ | $y_{12}$ | $y_{22}$ | $y_{32}$ |
| $t-2$ | $y_{13}$ | - | $y_{33}$ |
| $t-3$ | $y_{14}$ | - | - |
| $t-4$ | $y_{15}$ | - | - |

Damit alle diese Eigenschaften betrachtet werden können, existiert für jeden Zustand $s$ aus $S_{psm}$, der eine datenabhängige Leistungsaufnahme aufweist, folgende Ausgabefunktion:

$$\gamma_{data}(Z) = c + \sum_{i=1}^{n} \sum_{j=1}^{m_i-1} x_{ij}(d_{ij}(z_{ij}, z_{i(j+1)})) \quad , \tag{4.37}$$

wobei $n$ die Anzahl der betrachteten Eingänge, $m_i$ die Anzahl der betrachteten vergangenen Datenvektoren (inklusive dem aktuellen) für Port $i$, $d_{ij}$ die Differenzmetrikfunktion, $x_{ij}$ die Einflussfaktorfunktion und $c$ der Konstantanteil ist. Der Wert $Z$ beschreibt die Belegung der Variablen mit den betrachteten aktuellen und vorherigen Datenvektoren von allen

benötigten Datenports mit $z_{ij} \in Z$. Diese sind in einer Matrix gespeichert, wie beispielhaft in Tabelle 4.2 gezeigt.

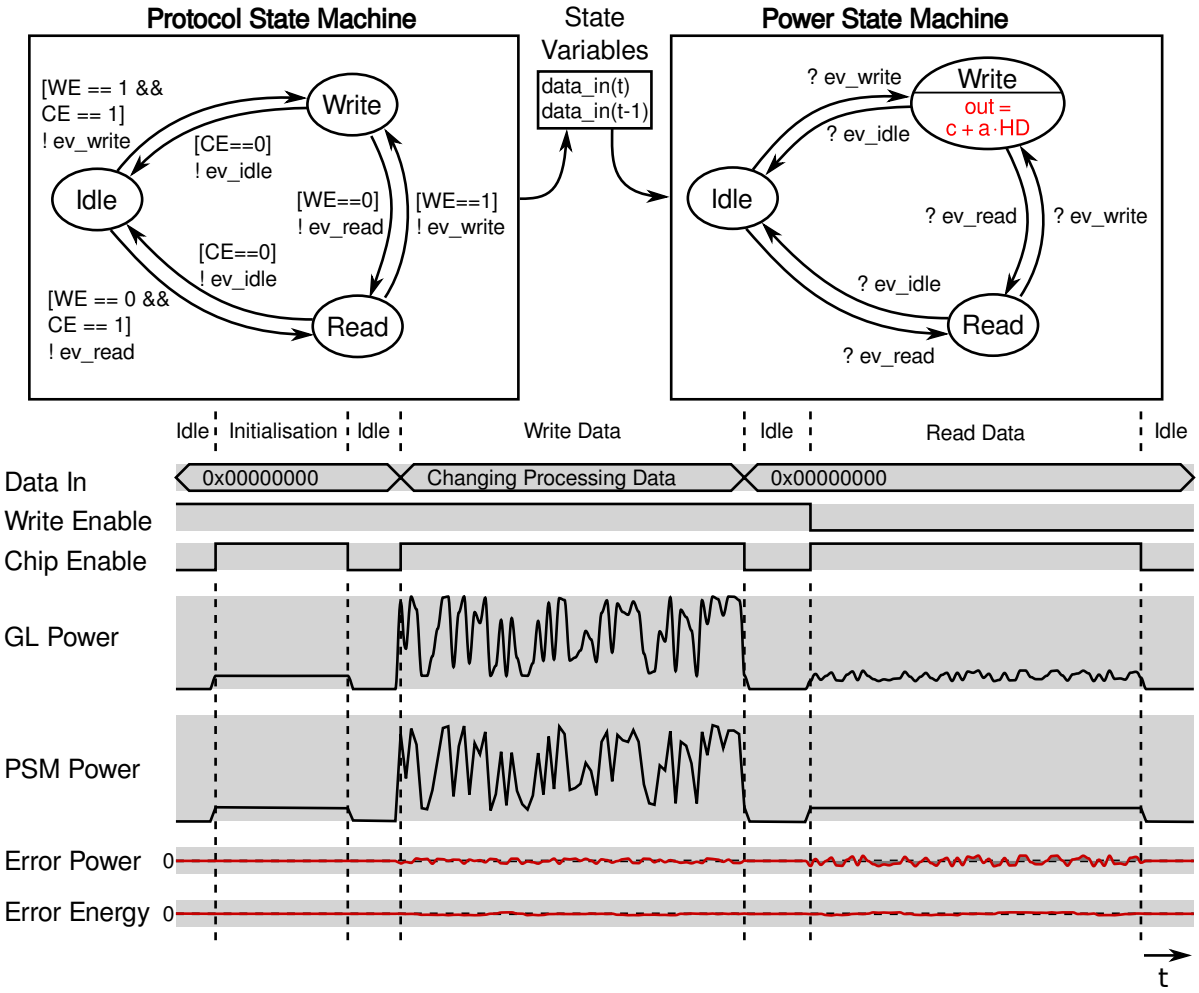



Abbildung 4.17.: Beispiel für die Modellierung der vollständig datenabhängigen Leistungsaufnahme für eine Speicherkomponente. Die datenabhängige Leistungsaufnahme wird durch eine dynamische Ausgabe im Zustand `Write` angenähert, die mit Hilfe der HD auf den Eingangsdaten berechnet wird. Das PSM-Modell muss durch zwei Zustandsvariablen erweitert werden, die die entsprechenden Änderungen darstellen.

Um zu verdeutlichen, wie diese Modellierung in der Praxis aussieht, wird dies an der bereits bekannten Speicherkomponente aus Abbildung 4.14 in Abbildung 4.17 gezeigt. Es wird deutlich, dass nur der Zustand `Write` mit einer dynamischen Ausgabe versehen wird, da die Abweichung im Zustand `Read` im Bereich des Abstraktionsfehlers liegt. Analog zu der vorherigen Modellierung müssen die Zustandsvariablen für die Zwischenspeicherung der Eingangsvektoren angelegt und die Zustandsübergänge der PrSM mit den nötigen Aktualisierungsfunktionen erweitert werden, damit eine Aktualisierung der Zustandsvariablen stattfindet. Es ist zu sehen, dass der durchschnittliche Fehler für den Zustand `Write` auf ein

Minimum reduziert werden kann und auch spezielles Verhalten (wie die Initialisierungsphase) keinen großen Fehler herbeiführt, da der vollständige Wertebereich der Differenzmetrik (in diesem Fall die HD) über eine lineare Funktion charakterisiert wurde. Auch hier wurde aufgrund der geringen Variation im Zustand `Read` keine weitere Modellverfeinerung durchgeführt. Das führt dazu, dass der mittlere Fehler in diesem Zustand nun höher ist als für den Zustand `Write`. Eine Charakterisierung der datenabhängigen Leistungsaufnahme für diesen Zustand ist auch möglich. Der Benutzer muss sich aber fragen, ob die nur wenig bessere Genauigkeit wichtiger ist, als die dadurch sinkende Simulationsperformanz (alle Berechnungen der Ausgabefunktion erhöhen den Mehraufwand und somit die Simulationszeit).

Die hier vorgestellte Möglichkeit zur vollständigen Modellierung der datenabhängigen Leistungsaufnahme bietet den Vorteil, dass sie die genauste Modellierung darstellt, aber auch den größten Mehraufwand bei sowohl der Charakterisierung als auch bei der Simulation besitzt. Der Grund für einen hohen Charakterisierungsaufwand ist, dass bei der Charakterisierung viele Parameter (wie die Anzahl der betrachteten Eingangsvektoren, die verschiedenen Differenzmetriken und die Einflussfaktoren) gefunden werden müssen. Bei der Simulation verursacht die Berechnung der resultierenden Leistungsaufnahme und die Speicherung der relevanten Eingangsvektoren einen höheren Aufwand als die anderen Methoden zur Modellierung der datenabhängigen Leistungsaufnahme.

In vielen Fällen ist vorher bekannt, wie sich die Daten bei bestimmten Anwendungsfällen verhalten. Für diesen Fall kann es sinnvoll sein, für die vorhandenen Daten mit Hilfe der dynamischen Ausgabe die resultierenden geschalteten Kapazitäten zu ermitteln und den Mittelwert zu berechnen, der dann als konstante Ausgabe an den entsprechenden Zustand annotiert wird. Dieses Vorgehen hat den Vorteil, dass die datenabhängige Leistungsaufnahme nur einmal charakterisiert werden muss und danach dynamisch an den entsprechenden Anwendungsfall angepasst werden kann. Des Weiteren entspricht die Simulationsperformanz der des PSM-Modells ohne Modellierung der datenabhängigen Leistungsaufnahme und die Genauigkeit der des in diesem Abschnitt vorgestellten Modells, wenn nur die durchschnittliche Leistungsaufnahme und nicht die Momentleistungsaufnahme betrachtet wird. Denn durch die durchschnittliche Betrachtung geht die Dynamik in der Ausgabe verloren und dementsprechend auch die zeitlich genaue Betrachtung der datenabhängigen Leistungsaufnahme.

Im vorherigen Abschnitt wurde das Problem erwähnt, dass die datenabhängige Leistungsaufnahme von sogenannten *Burst*-Transaktionen nicht dargestellt werden kann. Durch die dynamische Ausgabe ist es nun möglich, auch ohne mehrere Zustandsübergange in der PSM alle Datenänderungen einer *Burst*-Transaktion zu betrachten. Die Annahme ist, dass bei einer *Burst*-Transkation alle Einzeltransaktionen die gleiche Zeit brauchen. Die Grundidee ist nun, die *Burst*-Transaktion in den Zustandsvariablen vorzuhalten und für alle Datenänderungen einen entsprechenden Wert für die Leistungsaufnahme zu berechnen. Aus diesen Werten wird dann der Durchschnitt genommen und für die gesamte aktive Zeit des Zustands als Ausgangswert übernommen. Nur so kann ein erneuter Zustandswechsel in der PSM verhindert werden. Die einzige Voraussetzung dafür ist, dass genügend Zustandsvariablen

vorhanden sind bzw. diese entsprechend ausgelegt sind, um alle benötigten Eingangsdaten der PSM zur Verfügung zu stellen. Möchte man auch hier eine längere Historie betrachten, müssen entsprechend die vorherigen Werte in zusätzlichen Zustandsvariablen vorgehalten werden.

Für einen Burst mit der Länge von *o Burst-Ticks* lässt sich die resultierende Leistungsaufnahme wie folgt darstellen:

$$\gamma_{burst} = \frac{1}{o} \sum_{l=1}^{o} \gamma_{data}(z_l) \quad . \tag{4.38}$$

Diese Erweiterung ermöglicht durch das einfache Rechenmodell eine hohe Performanz in einer Simulation des Modells, eine effiziente Modellierung sowie eine einfache und adaptive Anpassung des Modells.

### 4.8.4. Äquivalenzbeweis der dynamischen Ausgabe

Um die Äquivalenz zwischen PSM-Zuständen mit dynamischer Ausgabe und konstanter Ausgabe zu zeigen, soll im Folgenden beschrieben werden, wie eine PSM mit dynamischer Ausgabe in eine PSM mit konstanter Ausgabe umgeformt werden kann. Dafür wird die Definition der PSM aus Abschnitt 4.3 genommen:

$$psm := (S_{psm}, s_{0,psm}, \Sigma_{psm}, \Gamma_{psm}, E_{psm}, Z, Inv) \quad .$$

Gibt es mehrere Zustände mit einer dynamischen Ausgabe, würde jeder Zustand einzeln umgewandelt werden. Das Vorgehen ist immer analog. Nachfolgend wird die Umformung für einen Zustand gezeigt.

Es wird eine Historie von $x$ Eingangswerten eines Eingangsvektors $y_i \subseteq \nu_P$ betrachtet, wobei $\nu_P$ der Definition aus (4.7) entspricht. Um $x$ Eingangswerte vorhalten zu können, werden entsprechend $x$ Zustandsvariablen $z \in Z$ benötigt, die diese Eingangswerte vorhalten. Damit trifft folgendes zu:

$$(y_i, y_{i-1}, \cdots, y_{i-(x-1)}) = (z_0, z_1, \cdots, z_{x-1})\text{HD} \quad . \tag{4.39}$$

Der Wert $\nu$ stellt den Wertebereich der Zustandsvariablen dar. Wie in (4.36) definiert, kann jeder Zustand eine dynamische Ausgabe haben, die abhängig von den Werten der Zustandsvariablen ist. Die Zuordnung ist folgendermaßen definiert:

$$\gamma_{const} = \gamma_{data}(z_0, \cdots, z_{x-1}) \quad . \tag{4.40}$$

Über die Anzahl $x$ der betrachteten Zustandsvariablen und dem Wertebereich $\nu$ ergibt sich eine Kombination von

$$m_s = x \cdot \nu \tag{4.41}$$

möglichen Wertbelegungen im Zustand $s$. Das bedeutet, ein Zustand mit dieser dynamischen Ausgabe muss in $x \cdot v$ Zustände umgewandelt werden, um eine dynamische Ausgabe in eine konstante zu transformieren. Dementsprechend muss auch der eine Zustandsübergang am Eingang zu $x \cdot v$ Zustandsübergängen verändert werden, die alle Zustände anbinden. Jeder Zustandsübergang bekommt als Aktivierungsbedingung eine Wertbelegung der Zustandsvariablen $z_0, \ldots, z_{x-1}$. Dem Zielzustand wird die konstante Ausgabe $\gamma_{const} = \gamma_{data}(z_0, \cdots, z_{x-1})$ zugewiesen. Die ausgehenden Zustandsübergange werden in gleicher Weise vervielfacht, damit diese auf den selben darauffolgenden Zustand zeigen wie vor der Transformation.

Die hier beschriebenen Eigenschaften sind beispielhaft in Abbildung 4.18 gezeigt. Der Wertebereich des datenabhängigen Eingangsvektors hat die Größe von 2 und es werden nur zwei aufeinanderfolgende Eingangsvektoren betrachtet. Aus (4.41) ergibt sich eine Menge von $4 = 2 \cdot 2$ Zuständen mit konstanter Ausgabe und entsprechend 4 Eingangs- und Ausgangszustandsübergängen.

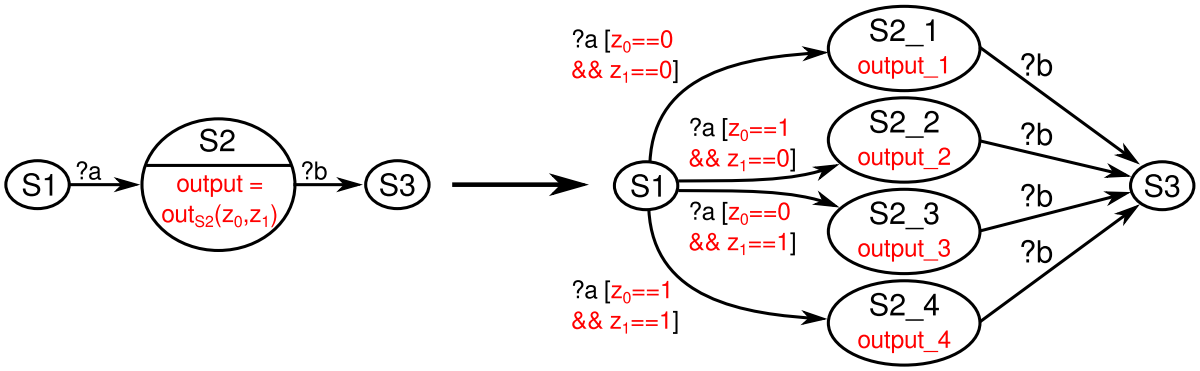



Abbildung 4.18.: Beispielhafte Modellierung des äquivalenten Ausgabeverhaltens mit dynamischer Ausgabe als auch konstanter Ausgabe. Es wird ein Wertebereich von 2 und zwei aufeinanderfolgende Eingabevektoren ($z_0$ und $z_1$) betrachtet.

## 4.9. Zusammenfassung

Dieses Kapitel hat die Herleitung des resultierenden Modells beschrieben, welches eine nichtinvasive Simulation der Leistungsaufnahme von Systemkomponenten ermöglicht. Dabei kann es Zustandsübergange modellieren, die nicht an einer Schnittstellenkommunikation sichtbar sind, als auch Datenabhängigkeiten. Das Konzept ist modular und kann daher unabhängig von oder in Kombination mit weiteren Modellen wie einem *Power Manager* für Versorgungsspannungs- und Taktfrequenzänderungen betrieben werden.

Dieses Modell ist für eine Simulation in Kombination mit einem funktionalen Modell konzipiert. Es wurde im Rahmen dieser Arbeit als *SystemC/TLM*-Modell implementiert. Dieses diente als Test für die Umsetzbarkeit auf ESL und die Integrationsfähigkeit des Modells.

Das PSM -Modell erfüllt alle in Abschnitt 1.2 beschrieben Anforderungen.

1. Anforderung F1 Leistungsaufnahme von Black-Box Komponenten: die entwickelte Methodik erlaubt eine vom funktionalen Modell orthogonale und unabhängige Modellierung der Leistungsaufnahme und ermöglicht damit den Einsatz für funktionale *Black-Box*-Komponenten.

2. Anforderung F2 Nichtinvasive Modellierung der Leistungsaufnahme: die Methodik erlaubt die Modellierung der Leistungsaufnahme über das Beobachten der Interaktion der funktionalen Komponente.

3. Anforderung F3 Leistungsaufnahmemodellierung unabhängig von Taktfrequenz, Versorgungsspannung und Temperatur: die Methodik stellt ein Modell bereit, welches von Taktfrequenz, Versorgungsspannung und Temperatur abstrahiert und erlaubt dadurch die unabhängige Modellierung dieser Parameter, wie z.B. durch einen *Power Manager* oder ein Temperaturmodell.

4. Anforderung F4 Betrachtung von nicht detektierbaren Zustandsänderung des funktionalen Modells: durch die Möglichkeit, zeitlich verzögerte Zustandsübergänge im Leistungsaufnahmemodell darzustellen, können Zustandsübergänge modelliert werden, die nicht durch Änderungen an den Schnittstellen der funktionalen Komponente sichtbar gemacht werden.

5. Anforderung F5 Berücksichtigung von datenabhängiger Leistungsaufnahme: für die Darstellung der datenabhängigen Leistungsaufnahme gibt es die Möglichkeit, einzelne Zustände so zu erweitern, dass diese die Aktivität auf den Ein- und Ausgängen nutzen und daraus die datenabhängige Leistungsaufnahme ableiten.

6. Anforderung F6 Modellierung auf Electronic System Level: durch die Einführung der PrSM als Zustandsautomat zur Darstellung des abstrahierten funktionalen Verhaltens, werden beliebige Schnittstellenarten unterstützt. Dadurch kann das PSM-Modell für verschiedene Abstraktionsebenen eingesetzt werden und unterstützt damit auch ESL.

7. Anforderung F7 Abstraktion der Gate Level-Leistungsaufnahme nach Electronic System Level: in Abschnitt 2.4 wurde beschrieben, wie die Leistungsaufnahme von GL nach ESL abstrahiert werden kann. In diesem Kapitel wurde gezeigt, wie diese in das PSM-Modell überführt werden kann. Das nächste Kapitel beschreibt näher ein Verfahren zur automatisierten Synthese.

8. Anforderung N1 Leistungsaufnahmemodell lässt sich einfach in Simulation integrieren: durch die orthogonale Modellierung und die Abstraktion von Parametern wie Versorgungsspannung, Taktfrequenz und Temperatur lässt sich das resultierende Modell einfach und schnell in bestehende Umgebungen integrieren und mitsimulieren.

Die nicht-funktionalen Anforderungen N2 und N3 werden im nächsten Kapitel adressiert. Des Weiteren wird dort beschrieben, wie das Modell erstellt und mit Leistungsaufnahmewerten befüllt wird. In Kapitel 6 wird das Modell anhand verschiedener Experimente auf seine Genauigkeit und Simulationsperformanz geprüft und bewertet.

# KAPITEL 5

# Automatisierte Modellsynthese

Im letzten Kapitel wurde ausführlich das PSM-Modell vorgestellt und gezeigt, welche Möglichkeiten der Leistungsaufnahmemodellierung es bereitstellt bzw. wo die Grenzen des Modells liegen. Da dieses Modell ohne zu Verfügung stehende Werte für die Leistungsaufnahme nicht eingesetzt werden kann, wird in diesem Kapitel nun ausführlich beschrieben, wie auf Basis des funktionalen Modells auf GL und der Technologiebibliothek die Leistungsaufnahmedaten erhoben und aufgearbeitet werden können. Diese werden dann in eine möglichst genaue Modellbeschreibung überführt. Der gesamte Vorgang wird als Synthese des Modells bezeichnet und beinhaltet mehrere verschiedene Schritte, die jeweils Abbildung 5.1 gezeigt sind und in den folgenden Abschnitten detailliert erläutert werden.

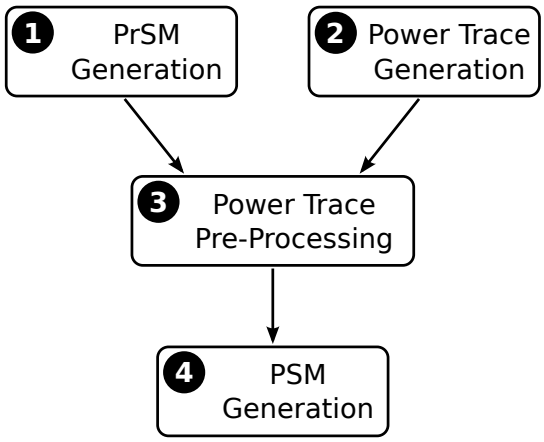



Abbildung 5.1.: Grundlegendes Vorgehen zur Erstellung des PSM-Modells.

Im ersten Schritt wird die PrSM erstellt. Dafür können verschiedene Informationen genutzt werden wie z.B. eine Beschreibung des funktionalen Verhaltens oder des Eingabe-/Ausgabeprotokolls, die dem Datenblatt der Komponente entnommen werden können.

Der früheste Schritt bei dem Design eines Hardware-Systems, an dem Leistungsaufnahmewerte zur Verfügung stehen, ist auf GL. Auf dieser Abstraktionsebene besteht das komplette Design aus einzelnen Logikgattern, die miteinander verschaltet sind. Technologiehersteller stellen für ihre einzelnen Technologien Bibliotheken zur Verfügung, die Aussagen machen

über die Leistungsaufnahme der Logikgatter. Diese sind abhängig von der Beschaltung bzw. dessen Änderung, Versorgungsspannung, Temperatur und der Prozessvariation. Mit Hilfe dieser Bibliothek kann ein Design auf GL simuliert und die resultierende Leistungsaufnahme berechnet werden. Das Problem an dieser Stelle ist, dass Simulationen von gesamten Systemen für alle notwendigen Anwendungsfälle so viel Zeit in Anspruch nehmen, dass diese nicht effizient durchführbar sind.

Eine Lösung ist, das System zu abstrahieren. Da die Leistungsaufnahme sehr stark von den Änderungen der Gatterzustände abhängt und nicht nur von dem Zustand, ist eine statische Betrachtung nicht möglich, sondern nur eine dynamische in Bezug auf die Anwendungsfälle. Um nicht alle Anwendungsfälle für das gesamte System betrachten zu müssen, wird dies Komponentenweise gemacht und dadurch die Anzahl der zu simulierenden Kombinationen drastisch reduziert. Aber auch auf Komponentenebene kann dies noch einen nicht unerheblichen Aufwand ausmachen. Die aus dieser Simulation gewonnenen Leistungsaufnahmewerte werden abstrahiert und auf das PSM-Modell auf RTL oder ESL abgebildet.

Damit eine solche Abbildung möglich ist, wird mit denselben Stimuliwerten, mit denen die GL-Leistungsaufnahme generiert werden, das ESL-Design stimuliert, wobei die erstellte PrSM diese Kommunikation beobachtet und so entsprechend mit simuliert wird. Dadurch generiert sie einen PSM-Event-Ausgabe-Trace, der über die Simulationszeit aufgezeichnet wird. Dieser dient dann als Repräsentant für die funktionale Ausführung auf ESL. Nun können die entsprechenden GL-Leistungsaufnahmewerte auf diesen PSM-Event-Trace abgebildet werden, um auf dieser Basis die PSM zu erstellen.

Die hierdurch generierte PSM ist eine Abstraktion und Zusammenfassung der GL-Leistungsaufnahme-Traces. Dieses Modul ist in der Lage, für die zur Synthese genutzten Stimuliwerte die gleichen Leistungsaufnahme-Traces in abstrahierter Form zu generieren. Die Abstraktion zeigt sich dadurch, dass nicht mehr taktgenaue Änderungen, sondern nur noch Änderungen bei einem funktionalen Zustandsübergang sichtbar sind und durch leichte Abweichungen im Mittelwert.

Um zu verstehen, wie Leistungsaufnahme auf GL und in der PSM zusammenhängen, wurde in Abschnitt 2.4 beschrieben, welche Umformungen nötig sind, um die Abbildung von GL-Werten zu PSM-Zustandswerten zu erhalten.

Abschnitt 5.1 erläutert, welche Informationen genutzt werden können, um die PrSM zu erstellen. Das Ziel dabei ist, dass die PrSM vollständig definiert ist, damit die darauf basierende Synthese der PSM eine höchstmögliche Genauigkeit und Abdeckung erreicht. Je nach Abstraktionsebene findet neben der Abstraktion der Leistungsaufnahmewerte auch eine Abstraktion des Kommunikationsprotokolls statt. Diese als auch die verschiedenen Semantiken der Kommunikationsdaten werden in Abschnitt 5.1.1 näher beschrieben.

In Abschnitt 5.2 wird beschrieben, wie mit Hilfe des RTL-Designs und der Protokollbeschreibung Leistungsaufnahme-Traces und die entsprechenden Eingabe- und Ausgabe-Traces erzeugt werden können, die als Grundlage für die Synthese der PSM dienen. Die in Abschnitt 5.2 erstellten Leistungsaufnahme-Traces müssen für eine genaue Synthese noch

vorverarbeitet werden. Dafür wird das Verhalten des Leistungsaufnahme-Traces analysiert und mit dem Verhalten der PrSM abgeglichen. Dieser Prozess wird in Abschnitt 5.3 beschrieben. Mit den vorverarbeiteten Leistungsaufnahme-Traces kann nun eine PSM erstellt werden, die das Leistungsaufnahmeverhalten auf ESL so genau wie möglich nachbildet. Der iterative Generierungsprozess wird in Abschnitt 5.4 erklärt. Ebenfalls wird dort erläutert, wie PSMs bewertet, miteinander verglichen und optimiert werden können. Der Generierungsprozess der PSM beinhaltet zusätzlich die Charakterisierung der datenabhängigen Leistungsaufnahme.

## 5.1. Generierung der PrSM

Wie im vorherigen Abschnitt erläutert, soll die Charakterisierung der Leistungsaufnahme auf Basis von GL-Simulationen durchgeführt werden, deren funktionale Simulations-Traces und Leistungsaufnahme-Traces als Referenz dienen. Diese werden von der PrSM in einen PSM-Event-Ausgabe-Trace überführt, der dann für Synthese der PSM genutzt wird. Daher muss die PrSM vor der PSM erstellt werden. Abbildung 5.2 zeigt diese Erstellung und wie sie sich in den gesamten Syntheseprozess einfügt.

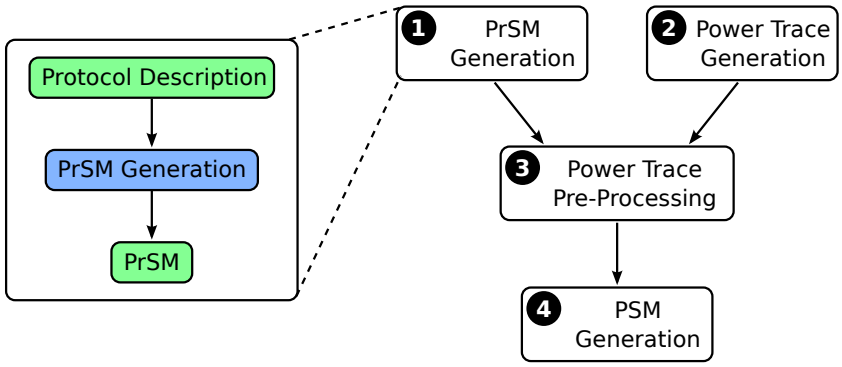



Abbildung 5.2.: Generierung der PrSM im Kontext des gesamten Syntheseprozesses.

Die oben beschriebene Idee ist es, auf Basis eines Anwendungsfalls einen GL-Leistungsaufnahme-Trace und einen funktionalen Ausführungs-Trace auf ESL zu erzeugen und diese aufeinander abzubilden. Auf dieser Grundlage lässt sich die in Abschnitt 5.3 und Abschnitt 5.4 beschriebe Synthese der PSM durchführen.

Da die PSM durch die Ausgaben der PrSM, die PSM-Events, getriggert wird, muss ein entsprechender Trace der PSM-Events für den gegebenen Anwendungsfall erstellt werden. Dafür wird bereits eine vollständig definierte PrSM benötigt. Zur Erzeugung des PSM-Event-Traces werden die Eingabe- und Ausgabewerte der GL-Simulation aufgezeichnet und der PrSM zugeführt. Je nachdem, welche Schnittstellenrepräsentation auf ESL benutzt wird (pinakkurat oder TLM), muss ein sogenannter *Transaktor* verwendet werden.

Eine PrSM, die für eine Komponente mit TLM-Schnittstellen ausgelegt ist, kann keine Werte aus einer pinakkuraten Schnittstelle verarbeiten.

Dieser Transaktor führt eine Abstraktion der pinakkuraten Werte durch. Diese wird in Abschnitt 5.1.1 beschrieben. In Abschnitt 5.1.2 wird erläutert, wie eine PrSM unter verschiedenen Vorbedingungen erstellt werden kann und worauf dabei geachtet werden muss, um eine korrekte und vollständige PrSM zu erhalten. In einigen Fällen ist es nötig, das Auftreten von bestimmte Events in der PrSM zu zählen. Um dieses Verhalten zu modellieren, kann die PrSM lokale Zählvariablen besitzen. Wie diese Events identifiziert und in die PrSM integriert werden, wird in Abschnitt 5.1.3 beschrieben.

### 5.1.1. Protokollabstraktion

Wie in Kapitel 1 beschrieben, ist das Ziel dieser Arbeit, ein Modell und Syntheseverfahren für Komponenten auf ESL zu entwickeln. Wie in Abschnitt 2.1 beschrieben, wird auf ESL heutzutage meist das sogenannte Transaction Level Modelling eingesetzt. Die für die Synthese nötigen Leistungsaufnahmewerte können aber erst auf GL hinreichend genau ermittelt werden, da erst auf dieser Abstraktionsebene die Technologieinformationen mit in das Design einfließen, mit denen die Leistungsaufnahme bestimmt werden kann.

TLM abstrahiert weitestgehend von genauen Busprotokollen auf Signalebene zur Synchronisation und auch von der genauen Repräsentation der Daten. Dadurch sind bei TLM nicht nur deutlich weniger Informationen zum genauen Ablauf der Kommunikation vorhanden, sondern auch die Zeiten können dabei ungenauer werden bzw. sich verändern. Diese Unterschiede werden in diesem Abschnitt erläutert und es wird erklärt, wie die verschiedenen Informationen zueinander zugeordnet werden können.

Es gibt viele verschiedene Implementierungen von Bussen, die sich teilweise stark untereinander unterscheiden, je nachdem für welchen Einsatzzweck sie entwickelt wurden. Beispielsweise sollen diese entweder einen hohen Durchsatz ermöglichen oder sie sollen eine geringe Komplexität und Leistungsaufnahme aufweisen. Letzteres wird z.B. mit dem *AMBA APB Protocol* ermöglicht. Nachfolgend wird dieses Busprotokoll kurz beschrieben, um zu verstehen wie Busprotokolle auf GL dargestellt werden. Danach wird gezeigt, wie dieses Protokoll auf TLM modelliert werden kann, damit der Zusammenhang zwischen den beiden Sichtweisen verdeutlicht wird und man versteht, welche Operationen ein Transaktor durchführt, der die Daten von GL nach TLM übersetzt.

#### Kommunikationsbusse

Die Aufgabe eines Busses ist es, Daten von einem oder mehreren *Master*-Komponenten zu einem oder mehreren *Slave*-Komponenten zu senden oder Daten von diesen zu empfangen. Die Kommunikation läuft dabei über sogenannte Registerschnittstellen. Das heißt, beim Senden von Daten werden diese in bestimmte Register, die von dem jeweiligen *Slave* bereit

gestellt werden, geschrieben und beim Empfangen werden die Daten aus diesen Registern gelesen. Für eine gezielte Ansteuerung der Register haben diese eine lokale Adresse. Da in der Regel mehrere *Slave*-Komponenten an einem Bus angeschlossen sind, hat jedes Register neben der lokalen Adresse auch eine globale Adresse. Diese berechnet sich aus einer für jeden *Slave* spezifischen Basisadresse im globalen Adressraum und der lokalen Adresse. Diese Basisadressen sind so verteilt, dass es zu keiner Überschneidung der Adressbereiche kommt. Diese Basisadressen werden von den *Master*-Komponenten benutzt. Demnach sind in jedem Bus Signalleitungen für die Adresse, den schreibenden und den lesenden Zugriff vorhanden. Da die Anzahl der Signale fest ist, können über den Bus immer nur eine definierte Menge an Daten übertragen werden.

Damit die Daten in die Register vom adressierten *Slave* geschrieben werden, gibt es einen *Decoder*, der zum einen die globale Adresse in eine lokale Adresse übersetzt und ein *Select*-Signal für den ausgewählten *Slave* setzt, damit der entsprechende *Slave* weiß, dass er adressiert wird. Für die Unterscheidung, ob ein lesender oder schreibender Zugriff durchgeführt wird, gibt es ein *Read/Write*-Signal.

Für den Fall, dass mehrere *Master* mit einem Bus verbunden sind, muss geregelt werden, zu welchem Zeitpunkt welcher *Master* auf den Bus zugreifen kann, da nicht mehrere *Master* gleichzeitig den Bus benutzen können. Bei dem *AMBA APB Protocol* ist immer nur ein *Master* erlaubt, daher sind die nachfolgend beschriebenen Signale und Komponenten bei diesem Busprotokoll nicht vorhanden.

Für die Regelung des Buszugriffs gibt es den sogenannten *Arbiter*. Jeder *Master* hat dafür ein *Request*- und ein *Grant*-Signal, das mit dem *Arbiter* verbunden ist. Möchte ein *Master* über den Bus Daten lesen oder schreiben, benutzt er das *Request-Signal*, um eine Anforderung an den *Arbiter* zu senden. Über das *Grant*-Signal kann der *Arbiter* dem *Master* den Zugriff gewähren. Diese Signale werden genau in dieser Form auf GL modelliert und ermöglichen eine exakte Simulation des Busverhaltens.

Bei TLM gibt es keine genaue Betrachtung der Signale. Eine Übertragung von Daten wird als Transaktion bezeichnet. In einer Transaktion werden alle Informationen gekapselt, die für eine korrekte funktionale Ausführung nötig sind. Dies sind die Adresse, die zu übertragenden Daten und ob die Transaktion schreibend oder lesend ist. Des Weiteren ist die Wortbreite festgelegt. Die Wortbreite beschreibt die Größe der Daten die genau zu einem Zeitpunkt über den Bus übertragen werden können.

Hier wird deutlich, dass die einzelnen Signale des konkreten Bus nicht auftauchen und demnach auch nicht modelliert werden. Dennoch sind in der Transaktion alle Informationen vorhanden, um eine korrekte funktionale Ausführung zu garantieren. Dadurch ist eine Abstraktion von konkreten Busprotokollen von GL nach TLM möglich. Muss eine solche Abstraktion zur Simulationszeit durchgeführt werden, kann dies durch einen sogenannten *Transaktor* umgesetzt werden, der eine pinakkurate Modellierung des Busses in eine TLM-Modellierung übersetzt und umgekehrt und bindet damit die PrSM an.

#### Busmaster und Busslave

Systemkomponenten stellen in vielen Fällen keine Busschnittstelle zur Verfügung, sondern eine proprietäre Schnittstelle, die genau die Eingänge und Ausgänge beinhaltet, die für die Datenverarbeitung und -bereitstellung benötigt werden. Der Grund dafür ist, dass der Designer der Komponente zur Designzeit nicht weiß, in welcher Umgebung die Komponente später eingesetzt wird und dementsprechend eine rein funktionale Schnittstelle zur Verfügung stellt. Diese kann mittels eines *Busslaves* an die Zielschnittstelle angepasst werden. Aktive Komponenten bekommen einen *Busmaster*, der abstrakte Buszugriffe in das Protokoll des verwendeten Busses umwandelt.

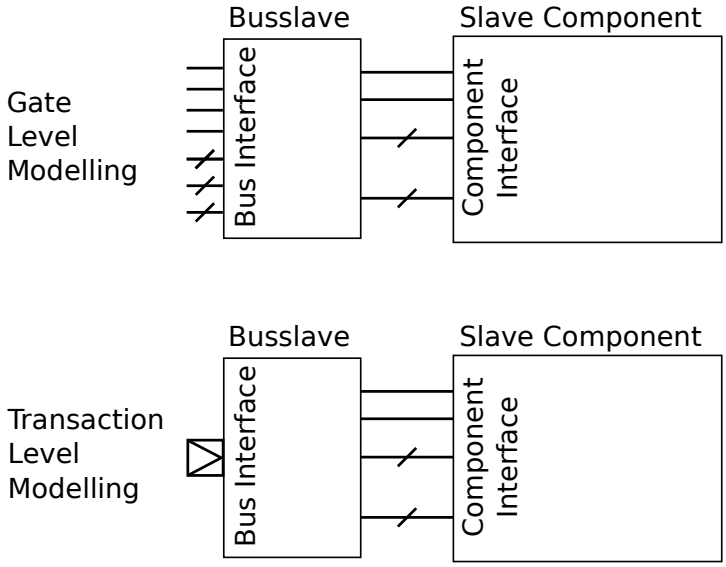



Abbildung 5.3.: Einbindung eines *Busslaves* und Vergleich zwischen GL und TLM.

Der *Busslave* übersetzt die Buszugriffe in Komponentenzugriffe bzw. stellt die Ausgabedaten der Komponente auf dem Bus zur Verfügung. Er implementiert die Registerschnittstelle und instanziiert die Register, wenn diese nicht von der Komponente zur Verfügung gestellt werden. Der *Busmaster* ist hingegen nur ein reiner Übersetzer in das konkrete Protokoll, da eine aktive Komponente in der Regel bereits mit einer Registersemantik arbeitet.

Auf ESL wird ein *Busmaster*/*Busslave* entweder als eigene Komponente dargestellt oder mit in die Beschreibung der Komponente integriert. Wird der *Busmaster*/*Busslave* als eigene Komponente modelliert, ist sowohl die Busschnittstelle als auch die Schnittstelle zwischen *Busmaster*/*Busslave* und Komponente sichtbar. Im anderen Fall ist nur die Busschnittstelle sichtbar und die Kommunikation zwischen *Busmaster*/*Busslave* und Komponente entweder nicht modelliert – hier werden die Daten meist direkt verarbeitet, um den Berechnungsaufwand zu minimieren – oder innerhalb der Komponente umgesetzt. In dieser Arbeit werden nur Komponenten betrachtet, bei denen die Schnittstelle zwischen *Busmaster*/*Busslave* und Komponente auch auf ESL sichtbar ist. Es wird jedoch in den Abschnitten, die sich auf

die Schnittstelle der Komponente beziehen, erläutert, wie das jeweilige Vorgehen und die Konzepte aussehen für Komponenten, bei denen auf ESL nur die Busschnittstelle sichtbar ist.

Für die Synthese wird auf GL und ESL immer die selbe Schnittstelle betrachtet. Das bedeutet, entweder wird nur die Komponente charakterisiert, dann wird in beiden Fällen die proprietäre Signalschnittstelle verwendet, oder der *Busmaster*/*Busslave* wird mit betrachtet und es wird auf ESL die entsprechende Busschnittstelle genommen. Der Grund dafür ist, dass die Verarbeitung in *Busmaster*/*Busslave* etwas Zeit benötigt. Dadurch kann es zu einer Verzögerung von Kommunikationsstrom und Leistungsaufnahme kommen und somit zu einer falschen Zuordnung im Syntheseprozess.

In der Regel wird die Komponentenschnittstelle verwendet, um den direktesten Bezug zwischen Kommunikation und Leistungsaufnahme zu erlangen. Der *Busslave* wird dann als eigene Komponente betrachtet und eine separate Synthese durchgeführt. Ist auf ESL nur die Busschnittstelle sichtbar, werden *Busslave* und Komponente zusammen charakterisiert. Dies führt in den meisten Fällen zu einem etwas ungenaueren Modell durch den oben beschrieben zeitlichen Versatz des *Busslaves*.

#### Protokollsemantik

Im Abschnitt 4.3 wurde beschrieben, dass die Eingaben der PrSM aus den Werten der Ports der funktionalen Komponente bestehen. Dabei haben diese Werte unterschiedliche semantische Eigenschaften, die in diesem Abschnitt näher beschrieben werden. Grob lassen sich die Werte in Konfigurations-, Kontroll- und Datensignale unterteilen.

Wie in Abschnitt 5.1.1 beschrieben, wird das Verhalten der Signale auf RTL auf das Verhalten der Registerschnittstelle abgebildet. Da diese somit äquivalente Wertemengen beschreiben, wird im Folgenden ohne Beschränkung der Allgemeinheit von Werten auf Signalen gesprochen.

Eine Sonderposition nimmt bei dieser Betrachtung das Taktsignal ein, da dieses keine Werte überträgt, sondern nur für die Synchronisation zwischen den Registern zuständig ist. Aus diesem Grund kann ein System auf ESL ohne Taktsignal modelliert und simuliert werden, ohne dass das funktionale Verhalten geändert wird. Da viele Taktzyklen ohne Änderung des funktionalen Verhaltens stattfinden, kann die Modellierung ohne Taktsignal zu einer deutlichen Steigerung der Simulationsperformanz führen.

**Konfigurationssignale** Konfigurationsdaten beeinflussen das Verhalten einer Komponente über einen längeren Zeitraum. Oftmals wird hierüber ein spezifischer Arbeitsmodus eingestellt, der festlegt, wie die Komponente die Eingangsdaten verarbeiten soll. Beispielsweise kann bei einer Verschlüsselungskomponente eingestellt werden, ob diese die Daten ent- oder verschlüsselt. Da ein starker Zusammenhang zwischen Arbeitsmodus und Leistungsaufnahme besteht, sind diese Daten maßgeblich für die Modellierung der Leistungsaufnahme.

**Kontrollsignale** Kontrolldaten beeinflussen den Kontrollpfad der Komponente und somit direkt den internen Zustand. Ein Beispiel für solche Daten ist das `Ready`-Signal, welches einer Komponente mitteilt, dass alle Dateneingaben gültig sind und sie die Ausgaben berechnen kann. Da der Kontrollzustand direkt mit dem Zustand der Leistungsaufnahme zusammenhängt, müssen diese Daten bei der Leistungsaufnahmemodellierung betrachtet werden.

**Datensignale** Über die Datensignale werden die Nutzdaten übertragen. Das sind die zu verarbeitenden Daten. In der Regel zeichnen sie sich dadurch aus, dass sie keinen Einfluss auf den Kontrollpfad und die Konfiguration haben. Diese Daten müssen nur dann für die Leistungsaufnahmemodellierung in Betracht gezogen werden, wenn sich eine direkte Korrelation zwischen der Änderung der Daten an den Ein- und Ausgängen und der Leistungsaufnahme abzeichnet.

Alle vorgestellten Signale können sowohl Eingangs- als auch Ausgangsdaten einer Komponente sein. Bei Konfigurationssignalen werden durch die Eingänge die jeweiligen Einstellungen vorgenommen und über die Ausgänge Auskunft gegeben, welche Einstellungen gegenwärtig konfiguriert sind. Die Eingangskontrollsignale steuern den zeitlichen Ablauf des Kontrollpfads und die Ausgänge der Komponente zeigen an, wann bestimmte Zustände des Kontrollpfades erreicht sind (z.B. Berechnung der Ausgaben wurde erfolgreich beendet). Die Nutzendateneingänge stellen die Nutzdaten bereit, wohingegen auf den Nutzdatenausgängen die berechneten Ergebnisse ausgelesen werden können.

#### Abbildung der Signale

Wie zu Beginn dieses Kapitels beschrieben, wird der GL-Trace genutzt, um die PrSM zu stimulieren, damit diese den für die Synthese nötigen Trace aus PSM-Events erzeugt. Damit eine PrSM für ein ESL-Modell mit einem GL-Trace stimuliert werden kann, müssen die entsprechende Signale von GL nach ESL abgebildet werden. Dabei werden entweder die Signale der Komponentenschnittstelle oder der Busschnittstelle abgebildet, je nachdem welche Schnittstelle für die Synthese genutzt wird. Eine solche Abbildung kann durch einen Transaktor passieren, der die Signale dynamisch zur Laufzeit abbildet.

**Komponentenschnittstelle** Bei der Komponentenschnittstelle sind die Repräsentation auf GL und ESL in der Regel sehr ähnlich. Der größte Unterschied liegt meist darin, dass anstelle konkreter Ports auf GL mit reinen Funktionsaufrufen gearbeitet wird. Des Weiteren fällt eine Modellierung des Taktsignals weg, da das Modell eventbasiert ist. Das Zeitverhalten kann ohne Verzögerung direkt aufeinander abgebildet werden.

**Busschnittstelle** Bei einem Vergleich der Darstellung von einem spezifischen Bus auf GL und ESL fallen deutlich mehr Unterschiede auf. In diesem Abschnitt wird die Abbildung anhand des *AMBA 3 APB Protocols* gezeigt. Dieses Protokoll ist auf einen niedrigen Datendurchsatz ausgelegt und hat entsprechend wenige Signal. Dieses Protokoll besteht aus drei einfachen Zuständen. Der erste Zustand beschreibt den Ruhezustand, in dem der Bus inaktiv ist. Im zweiten Zustand wird eine Transaktion vorbereitet und im dritten Zustand wird der schreibende und lesende Zugriff durchgeführt. Komplexere Protokolle wie das *AMBA 3 AXI3 Protocol* besitzen deutlich mehr Signale, um die Vielfalt der Funktionen zu unterstützen. Beispielsweise gibt es weitere Signale, um sogenannte *Burst*-Transfers zu ermöglichen, die in mehreren Zyklen (*Burst-Ticks*) Daten senden, ohne weitere Vorbereitungszyklen zu benötigen.

Schaut man sich das *APB* Protokoll [4] an, findet man folgende Signale:

- PCLK (Clock Source): Takt. Die positive Taktflanke steuert alle Transfers.
- PRESETn (System bus equivalent): Rücksetzsignal, reagiert auf ein negatives Signal.
- PADDR (APB Bridge): Adresse.
- PSELx (APB Bridge): Signal für die Auswahl des entsprechenden *Slaves*.
- PENABLE (APB Bridge): Zeigt den dritten Zustand an, in dem der Datenzugriff durchgeführt wird.
- PWRITE (APB Bridge): Zeigt an, ob Daten geschrieben oder gelesen werden.
- PWDATA (APB Bridge): Daten, die geschrieben werden sollen.
- PREADY (Slave): Mit diesem Signal kann der *Slave* einen Transfer verzögern.
- PRDATA (Slave): Daten, die gelesen werden sollen.
- PSLVERR (Slave): Zeigt einen Transferfehler an.

Wird ein *APB* eingesetzt, gibt es für die Adresse und die übertragenen Daten eine feste Wortbreite. Diese kann bis zu 32 Bit betragen.

In einer abstrakt modellierten Transaktion auf ESL findet man in der Regel folgende Informationen:

- Daten,
- Datenlänge,
- Wortbreite,
- Adresse,
- Schreibender/Lesender Zugriff und
- Transferfehlerantwort.

Hier wird deutlich, dass diese Art der Modellierung sehr generisch ist und es viele Gemeinsamkeiten zwischen der Busmodellierung auf ESL und der konkreten Umsetzung gibt. Dieses Modell ist bereits in der Lage, viele Funktionalitäten wie z.B. *Burst*-Transfers darzustellen. Werden komplexere Busse mit zusätzlichen Funktionalitäten benutzt, wie z.B. das *AXI4* Protokoll von AMBA, können diese in der Regel nicht mehr durch solch eine generische Modellierung abgebildet werden. Für diesen Fall werden speziellere Modelle auf ESL erstellt und eingesetzt, die diese zusätzlichen Funktionalitäten in einer abstrakten Weise implementieren.

Beim Vergleich der abstrakten Modellierung auf ESL und der konkreten Umsetzung auf GL fällt auf, dass das Taktsignal, das Rücksetzsignal, das Signal für den Datenzugriff und die Verzögerung des Transfers als auch das Auswahlsignal fehlen. Diese werden nicht für eine korrekte funktionale Ausführung auf ESL benötigt, sondern nur für den korrekten zeitlichen Ablauf des Busprotokolls. Das Auswahlsignal wird durch einen direkten Funktionsaufruf des *Masters* ersetzt. Alle anderen Signale lassen sich wie folgt abbilden:

- PADDR → Adresse,
- PWRITE → Schreibender/Lesender Zugriff,
- PWDATA → Daten,
- PRDATA → Daten und
- PSLVERR → Transferfehlerantwort.

Die Informationen für die Wortbreite entspricht der Wortbreite des gewählten *APB* Protokolls. Da dieser keine *Burst*-Transfers unterstützt, ist die Datenlänge immer gleich der Wortbreite. Unterstützt ein Bus *Burst*-Transfers, kann die Datenlänge ein Vielfaches der Wortbreite sein.

Bei der Modellierung von Transaktionen auf ESL werden in der Regel wenige Zeitpunkte genutzt, um das Zeitverhalten zu modellieren. Diese sind jeweils der Beginn und das Ende der Anfrage und Antwort einer Transaktion. Diese Zeitpunkte werden auf das konkrete Protokoll abgebildet. Hierbei werden die Zeitpunkte so gewählt, dass sie mit den Änderungen des funktionalen Zustands zusammenfallen, weil diese ausschlaggebend für eine Veränderung der Leistungsaufnahme sind. Beim *APB* würde man den Zeitpunkt des Datenzugriffs wählen, weil zu diesem Zeitpunkt die Daten übernommen werden und dies die Änderung des funktionalen Zustands einleitet.

### 5.1.2. Erstellung der PrSM

Das Ziel bei der PrSM-Erstellung ist es, eine PrSM zu erhalten, die so genau wie möglich das funktionale Verhalten der untersuchten Komponente nachbildet. Nur so kann gewährleistet werden, dass die davon abhängige PSM die größtmögliche Genauigkeit erlangt.

Für die Erstellung der PrSM gibt es mehrere verschiedene Möglichkeiten, die von verschiedenen Faktoren abhängig sind und unterschiedlich genaue Modellierungen der PrSM ermöglichen:

1. Erstellung durch den IP-Designer. Ist beim Entwickeln der IP-Komponente bereits bekannt, dass diese auch für die Simulation der Leistungsaufnahme genutzt werden soll, kann die PrSM direkt durch den Designer der IP-Komponente erstellt werden.

2. Die Funktion wurde bereits als Zustandsautomat definiert. Häufig ist in der Dokumentation der Komponente das funktionale Verhalten als abstrakter Zustandsautomat definiert. Alle Zustandsübergänge dieses Automaten, die durch ein Eingangs- oder Ausgangssymbol ausgelöst werden, können in der PrSM dargestellt werden. Somit kann durch ein Entfernen aller anderen Zustandsübergänge die PrSM erstellt werden.

3. Es gibt eine textuelle oder grafische Beschreibung des Protokolls, die in eine PrSM überführt werden kann.

4. Keine Protokollbeschreibung ist verfügbar, sondern nur Anwendungsfälle, die die Benutzung der IP-Komponenten verdeutlichen.

Nachdem die PrSM erfolgreich erstellt wurde, werden ihre Zustandsübergänge mit eineindeutigen PSM-Events annotiert. Für diese müssen folgende Kriterien zwingend erfüllt sein:

- Für jede Eingabe muss es genau einen Zustandsübergang geben, der diese Eingabe verarbeitet, damit eine deterministische Ausführung der PrSM möglich ist.

- Zustandsübergänge, die als Start- und Zielzustand den gleichen Zustand haben, zeigen an, dass die PrSM in dem selben Zustand bleibt und demnach eine Änderung der Leistungsaufnahme hier nicht durch eine an den Schnittstellen der Komponente beobachtbare Änderung herbeigeführt wird. Deshalb werden diese Zustandsübergänge nicht mit einem PSM-Event annotiert.

- Alle anderen Zustandsübergänge, also deren Start- und Zielzustand verschieden sind, müssen mit einem eineindeutigen PSM-Event annotiert werden, da sich hier der Kontrollzustand ändert und somit eine Änderung der Leistungsaufnahme herbei führen kann. Die Eineindeutigkeit ist notwendig, da nur so alle Möglichkeiten der Leistungsaufnahmemodellierung identifiziert werden können.

- Es muss einen definierten Startzustand geben, der für alle Szenarien gilt. Andernfalls kann dies zu falschen Simulationsergebnissen und eine falscher Charakterisierung der Leistungsaufnahmewerte führen.

#### Erstellung durch IP-Designer

Die offensichtlich einfachste und genauste Möglichkeit der PrSM-Erstellung erfolgt durch den IP-Designer selbst. Dieser besitzt das genaue Wissen über die Funktionsweise der Komponente, selbst wenn diese eine *Black-Box*- oder *Grey-Box*-Komponente ist. Demnach kann dieser einfach einen (abstrahierten) Zustandsautomaten auf Basis seines Designs erstellen, der dann als PrSM dient. Dieser muss dann lediglich, wie im nächsten Abschnitt beschrieben, so angepasst werden, dass die Zustandsübergänge durch beobachteten Eingabe-/Ausgabewerte getriggert werden und dass an jeden Zustandsübergang, der verschiedene Zustände als Start- und Zielzustand hat, ein eineindeutiges PSM-Event annotiert wird.

Da auch der IP-Designer an die Einschränkung gebunden ist, dass die PrSM nur durch Ein- und Ausgaben der IP-Komponente getriggert werden kann, müssen entsprechende interne Zustandsübergänge und damit verbundene Zustände in der PrSM entfernt werden, weil nur die PSM die Möglichkeit der zeitgesteuerten Zustandsübergänge aufweist (siehe Abschnitt 4.3). Da dem IP-Designer diese Zustandsübergänge aber bekannt sind, kann er diese inklusive ihrer Übergangsbedingung dokumentieren und an den Benutzer weitergeben, damit er diesen Zustandsübergang in der PSM modellieren kann und dadurch eine genaueres PSM-Modell erhält.

Da diese Art der PrSM-Modellierung natürlich den möglicherweise geheimen und schützenswerten inneren Aufbau der IP-Komponente offenlegt, kann der IP-Designer nicht in jedem Fall das gesamte funktionale Verhalten in der PrSM modellieren. Hier kann der Designer das Verhalten abstrahieren. Wichtig dabei ist, dass das vollständige funktionale Verhalten der IP-Komponente in der abstrakten Darstellung enthalten ist, also eine valide Abstraktion durchgeführt wird und dadurch eine korrekte Ausführung der PrSM gewährleistet wird. Dies kann z.B. geprüft werden mit Methoden der sogenannten *Sound Abstraction* [73].

Da viele IP-Komponenten bereits erstellt und eingesetzt werden, ist hier eine PrSM-Erstellung durch den Designer der Komponente nicht mehr möglich.

Liegt eine IP-Komponente als sogenannte *White-Box*- oder *Grey-Box-Komponente* vor, ist der gesamte oder zumindest ein Teil des funktionalen Aufbaus der Komponente auch für den Benutzer sichtbar. Diese Informationen können genutzt werden, um die Funktionsweise der Komponente und die Beziehung mit dem Eingabe- und Ausgabeprotokoll besser zu verstehen. Mit diesem Wissen kann nach dem in diesem Abschnitt beschriebenen Verfahren eine genaue PrSM erstellen werden.

#### Erstellung aus funktionalem Zustandsautomat

Oftmals werden IP-Komponenten mit einer Automatenbeschreibung ausgeliefert, die den Kontrollpfad der Komponente beschreibt. Dieser definiert den Kontrollfluss und somit den Kontrollzustand abhängig von den Eingaben auf den Kontrolleingängen. Des Weiteren zeigt dieser Zustandsautomat an, wann Kontrollausgaben erzeugt werden, die das innere funktionale Verhalten der Komponente anzeigen. Diese Informationen können, wie oben

beschrieben, genutzt werden, um daraus die PrSM zu erstellen, die den funktionalen Kontrollzustand der Komponente von den Ein- und Ausgaben der Komponente ableitet.

#### Erstellung aus Schnittstellenbeschreibung

In manchen Fällen werden IP-Komponenten ohne Beschreibung ihres funktionalen Verhaltens ausgeliefert, sondern nur mit einer Beschreibung ihres Schnittstellenverhaltens. Da das Schnittstellenverhalten das funktionale Verhalten abstrahiert darstellt, kann dieses genutzt werden, um daraus die PrSM zu konstruieren.

Aus der Schnittstellenbeschreibung können die verschiedenen Kommunikationszustände und deren Übergangsbedingungen definiert werden, um daraus die PrSM zu konstruieren.

In einigen Fällen ist es einfacher, einzelne Zustandsautomaten für eine Untermenge der Ports zu erstellen, da diese semantisch miteinander verknüpft sind. Danach können zwei Automaten zu einem Produktautomaten vereint werden. Dies muss so lange wiederholt werden, bis nur noch ein Zustandsautomat übrig ist, da nur eine PrSM existieren kann. Wichtig dabei ist, anschließend alle nicht erreichbaren Zustände und ungültigen Zustandsübergänge zu entfernen. Wird dieser Schritt nicht durchgeführt, kann die automatische Synthese nur eine unvollständigen PSM erzeugen (siehe Abschnitt 5.4.4), da bei der Simulation nicht alle PrSM-Zustände erreicht werden und auch nicht alle Zustandsübergänge ausgeführt werden. Dadurch triggert die PrSM nicht alle existierenden PSM-Events, wodurch diese nicht von der automatischen Synthese verarbeitet werden können. Nicht erreichbare Zustände und ungültige Zustandsübergänge werden erkannt, indem nach der Stimuliauswahl die Abdeckung der Zustände und Übergänge bei einer Simulation überprüft wird (siehe Abschnitt 5.2.3). Für alle nicht erreichten Zustände und Übergänge gilt:

- die Stimuliwerte haben nicht das gesamte funktionale Verhalten abgedeckt oder
- die Zustände sind nicht erreichbar bzw. Zustandsübergänge sind ungültig und müssen entfernt werden.

Um eine automatisierte Erstellung der PrSM aus der Protokollbeschreibung zu erreichen, müsste dieses in einem standardisiertem Format vorliegen. Jedoch gibt es viele verschiedene Möglichkeiten der Protokollbeschreibung und keine Standardisierung oder Festlegung auf ein bestimmtes Format. Bunker beschreibt diese verschiedenen Möglichkeiten der Protokollbeschreibungen und vergleicht sie nach Vor- und Nachteilen [14]. Diese sind grob unterteilt in textuelle und grafische Sprachen. Abdullah beschreibt einen Ansatz, bei dem Protokolle in *Extensible Markup Language* (XML) spezifiziert werden. Das allgemeine Modellierungskonzept erlaubt es, verschiedene Protokolle zu beschreiben. Sie werden durch Zustandsautomaten dargestellt. Dies erlaubt eine konkrete Definition und leichte Generierung von Quellcode für verschiedene Sprachen oder grafische Repräsentationen. Oftmals gibt es lediglich eine textuelle Beschreibung des Verhaltens. Aus diesem Grund hat Kof Methoden entwickelt, diese Beschreibungen automatisiert in Automaten zu übersetzen [36].

Wendet man diese Konzepte an, kann die PrSM somit auch automatisiert aus einer definierten Protokollbeschreibung generiert werden, sobald die Schnittstellenbeschreibung in der standardisierten Form vorliegt. Da dies für heutige IP-Komponenten jedoch selten der Fall ist, muss die PrSM in den meisten Fällen manuell erstellt werden.

Ist die Schnittstelle der Komponente an einen Bus angeschlossen (z.B. den *AMBA Peripheral Bus*), ist das Zugriffsprotokoll bereits klar definiert. Die Daten werden hier auf eine Registerschnittstelle übertragen. Diese ist für jede Komponente unterschiedlich und hängt von der zur Verfügung gestellten Funktionalität ab. Diese Registerschnittstelle wird in der Regel in der dazugehörigen Dokumentation genau beschrieben. Da diese Form der Schnittstellen in heutigen SoC am häufigsten eingesetzt wird, wurde ein standardisiertes Format erstellt, die sogenannte *Register Defenition Language* (RDL) [1], um Registerschnittstellen eindeutig beschreiben zu können. Das Entwicklungsziel bei diesem Format war es, während des gesamten Designverfahrens des Systems eine definierte Spezifikation der Registerschnittstelle zu haben und diese auch zur fehlerfreien Code-Erstellung für die Schnittstellen zu nutzen.

#### Erstellung ohne jegliche Beschreibung

In seltenen Fällen kann es vorkommen, dass eine genaue Beschreibung für die Funktionalität als auch für die Schnittstellen fehlt. Dies passiert, wenn Komponenten veraltet sind und die Unterstützung für diese Komponente eingestellt wurde oder wenn die Dokumentation verlegt wurde und nicht mehr auffindbar ist.

Wenn keine oder nur eine unvollständige Beschreibung vorhanden ist, kann das Schnittstellenverhalten abgeleitet werden, indem man die Schnittstellen der Komponente während einer Simulation beobachtet. Dafür werden die Signalwechsel aufgenommen und die Korrelationen zwischen Signalen extrahiert. Voraussetzung hierfür ist, dass ein oder mehrere Anwendungsfälle genau beschrieben sind bzw. eine Testumgebung mit einem oder mehreren definierten Anwendungsfällen existiert. Dabei müssen diese Anwendungsfälle das gesamte Kontrollflussverhalten an den Ports abbilden. Wie an der nachfolgend beschriebenen Methode deutlich wird, führt jedes nicht definierte Verhalten zu einer fehlerhaften oder unvollständigen PrSM.

Viele der Arbeiten auf diesem Gebiet leiten Automaten aus Sequenzdiagrammen ab, die den Ablauf der Komponente oder deren Kommunikationsverhalten beschreiben [32, 37]. Hierbei werden die einzelnen Zustände über Statusausgaben oder Zustandsvariablen synthetisiert und die Bedingungen identifiziert, die zu einem Zustandsübergang führen.

Ein Vorgehen für eine automatische Automatensynthese auf Basis von Eingabe- und Ausgabe-Traces wurde von Danese gezeigt [15]. Hier werden sogenannte *Temporal Assertions* aus den Eingabe- und Ausgabe-Traces extrahiert, die zeitlich abhängige Bedingungen zwischen verschiedenen Ein- und Ausgangsdaten beschreiben. Aus diesen Bedingungen kann im nächsten Schritt ein ausführbarer Zustandsautomat generiert werden. Des Weiteren wird

der Automat als PSM umgesetzt, um ihn für die Simulation der Leistungsaufnahme einsetzten zu können. Im Gegensatz zu dem in dieser Arbeit beschriebenen Ansatz, wird jedem Zustand des generierten Automaten über eine Mittelwertbildung ein Leistungsaufnahmewert zugewiesen und der Automat wird direkt von den Eingangs- und Ausgangsdaten getriggert. Dadurch gibt es keine Trennung zwischen funktionalem Verhalten und Leistungsaufnahmeverhalten – in dieser Arbeit durch PrSM und PSM umgesetzt (siehe Abschnitt 4.3) – sondern eine direkte Kopplung. Wie in der Evaluation dieser Arbeit deutlich wird (siehe Kapitel 6), kann eine direkte Zuordnung von Funktionalität und Leistungsaufnahme zu Ungenauigkeiten führen.

### 5.1.3. Charakterisierung der Zählvariablen

Die in den letzten Abschnitten beschriebenen Schritte reichen in der Regel aus, um die PrSM vollständig zu beschreiben. In einigen Fällen sind sogenannte Zählvariablen nötig, um die Anzahl der Zustände zu reduzieren und so die Komplexität zu verringern und die Übersichtlichkeit zu gewährleisten. Zählvariablen werden immer dann gebraucht, wenn spezielle Events oder Zustandsübergänge nach einer bestimmten Anzahl von Aktionen auftreten. Dies passiert beispielsweise oft bei *Streaming*-Komponenten, die erst nach einer spezifischen Menge an Eingangsdaten anfangen, diese zu verarbeiten, weil der Algorithmus beispielsweise mehrere Daten braucht, um diese erfolgreich komprimieren zu können oder um einen Puffer zu füllen, damit am Ausgang ein kontinuierlicher Datenstrom erzeugt werden kann. Wenn der Beginn dieser Berechnung nicht durch ein spezifisches Kommunikationsevent signalisiert wird, muss die PrSM die Eingangsdaten zählen, um zu festzustellen, wann die Berechnung beginnt. In diesem Fall baut die PrSM den nötigen Teil des funktionalen Verhaltens nach, um das korrekte funktionale Verhalten abzuleiten.

Zählvariablen werden speziell dann benötigt, wenn der abhängige Wert (z.B. die *Frame*-Größe des Datenstroms) dynamisch konfigurierbar ist und deshalb zur Laufzeit verändert werden kann. Hier wird ersichtlich, dass in diesem Fall zwei Zustandsvariablen gebraucht werden. Der erste Wert beinhaltet den Konfigurationswert und der zweite den aktuellen Zählwert. Der Konfigurationswert bekommt einen Initialwert (dieser wird bei der *Reset*-Phase gesetzt), der von der PrSM verändert werden kann, sobald ein Zugriff erkannt wird, der diesen Konfigurationswert ändert. Die Zählvariable wird analog zur Konfigurationsvariable geändert und immer dekrementiert, sobald die Aktion auftritt, die gezählt wird. Erreicht der Zählvariable den Wert 0, wird der entsprechende Zustandswechsel in der PrSM durchgeführt und die Zählvariable wird wieder auf den Wert der Konfigurationsvariable gesetzt. Es muss darauf geachtet werden, wie sich die gezählten Ereignisse verhalten, wenn sich die Konfigurationsvariable ändert. Eventuell wird diese dabei auch zurückgesetzt oder über die entsprechende Änderungsdifferenz mit geändert.

Durch das Einsetzen von Zählvariablen in der PrSM werden Zustandsübergänge durch das Inkrementieren der Zählvariable abgebildet. Da die Zustandsübergänge der PrSM mit eineindeutigen PSM-Events annotiert werden, muss auch die Veränderung einer Zählvariable mit der Ausgabe eines PSM-Events in Verbindung gebracht werden. Dies führt zu

einer Benachrichtigung der PSM, die basierend auf dem PSM-Event und dem Wert der Zählvariable den Zustand und die Ausgabe anpassen kann.

## 5.2. Generierung des Leistungsaufnahme-Traces

Wie in Abschnitt 2.2.2 beschrieben wurde, ist die dynamische Leistungsaufnahme stark abhängig von der Versorgungsspannung, der Taktfrequenz, der Schaltaktivität sowie der Schaltkapazität. In Abschnitt 2.4 wurde gezeigt, wie in der PSM von der Taktfrequenz sowie Versorgungsspannung abstrahiert wird, um konstante Ausgabewerte in den einzelnen Zuständen der PSM zu ermöglichen. Somit bleiben die beiden zu identifizierenden Parameter die geschaltete Kapazität und die Schaltaktivität.

In der Regel liegen Systemkomponenten auf ESL und auf RTL vor. Auf ESL kann eine vergleichsweise schnelle Simulation durchgeführt werden, um das Design zu prüfen. Ziele dieser Prüfung sind eine korrekte funktionale Ausführung des Systems, zeitlich korrektes Verhalten und auch das in dieser Arbeit adressierte korrekte Leistungsaufnahmeverhalten des Systems bzw. einzelner Systemkomponenten.

Auf RTL wird gewöhnlich keine komplette Systemsimulation durchgeführt, da diese zu viel Zeit beanspruchen würde. Systemkomponenten werden auf dieser Ebene oft nur einzeln simuliert, um die korrekte Funktionalität zu prüfen. Da das Design auf RTL schon taktgenau simuliert wird, kann das zeitliche Verhalten gegen die entsprechenden Vorgaben validiert werden. Auf RTL werden noch keine Technologieparameter betrachtet, wodurch weder der *kritische Pfad* (siehe Abschnitt 2.3.2) bestimmt noch eine Energieaufnahme berechnet werden kann.

Abbildung 5.4 zeigt, wie ein Leistungsaufnahme-Trace erstellt werden kann und sich die einzelnen Schritte in den gesamten Syntheseprozess einfügen.

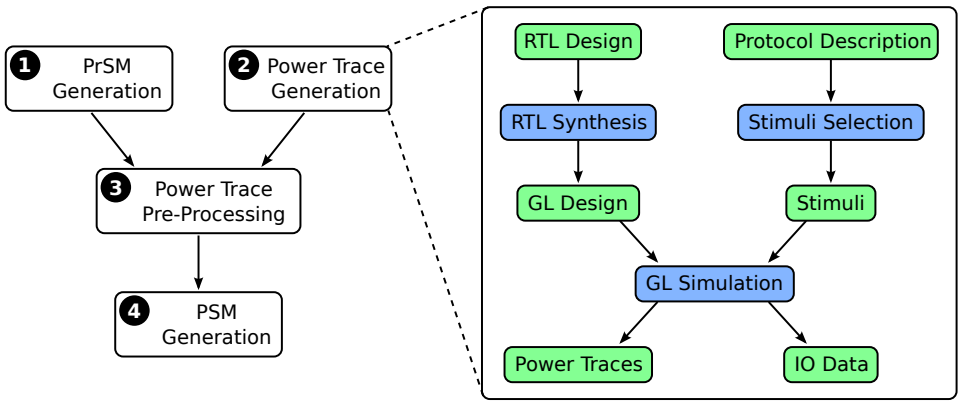



Abbildung 5.4.: Generierung eines Leistungsaufnahme-Traces mit Hilfe des GL-Designs. Als Vorbereitungsschritt wird das vorliegende RTL-Design synthetisiert und eine Auswahl an Stimuliwerten getroffen.

Die Technologieparameter fließen erst auf GL in das Design ein. Aus diesem Grund muss das Design für eine Abschätzung der Leistungsaufnahme erst von RTL nach GL synthetisiert werden. Dieser Vorgang wird in Abschnitt 5.2.1 beschrieben. Durch Hinzunahme der Technologieparameter werden die Kapazitäten für die Schaltelemente konkret festgelegt. Die Kapazitäten für die Signalleitungen ergeben sich erst in dem nachfolgenden *Place-and-Route*-Schritt, da die Kapazität einer Signalleitung stark von der Leitungslänge abhängt.

Die PSM betrachtet den letzten offenen Parameter, die Schaltaktivität. Diese wird beeinflusst durch die Werte bzw. die Wertänderungen an den Eingängen der Komponente. Damit eine möglichst genaue Charakterisierung der Leistungsaufnahme möglich ist, sind also die Stimuliwerte entscheidend. Die Auswahl dieser wird in Abschnitt 5.2.2 diskutiert.

Mit dem synthetisierten GL-Design und den ausgewählten Stimuliwerten kann eine Komponentensimulation auf GL durchgeführt werden. Bei dieser werden alle Signalwechsel aufgenommen und inklusive der entsprechenden Simulationszeiten in einer *Value Change Dump* (VCD)-Datei gespeichert. Mit diesen Informationen kann dann unter Zuhilfenahme der Technologiebibliothek ein Leistungsaufnahme-Trace erzeugt werden, der die Leistungsaufnahme über die Zeit beschreibt. Dieser Vorgang wird im Detail in Abschnitt 5.2.3 beschrieben. Können in einem Simulationslauf nicht alle benötigten Zustände erreicht oder Zustandsübergänge ausgeführt werden, müssen mehrere Stimulivektoren erzeugt werden, um damit verschiedene Simulationen durchzuführen, die alle wichtigen Aspekte der Leistungsaufnahme aufzeigen können.

### 5.2.1. GL-Synthese

Wie oben beschrieben wurde, muss das zur Verfügung stehende RTL-Design nach GL synthetisiert werden, da erst auf dieser Ebene die Technologieparameter mit in das Design einfließen, welche eine Berechnung der Leistungsaufnahme ermöglichen. Das RTL-Design kommt, wie auch das ESL-Design, vom Designer der Komponente. Der Grund ist, dass im RTL-Design viele Designparameter enthalten sind, die auf ESL vernachlässigt werden, um die Simulationsgeschwindigkeit zu erhöhen, aber trotzdem eine korrekte funktionale Ausführung ermöglichen.

Das RTL-Design kann so wie das ESL-Design händisch entwickelt worden sein. In diesem Fall hat der Designer beide Modelle gegeneinander validiert, sodass sie unter Garantie eine äquivalente funktionale Ausführung aufweisen. Ist es möglich, werden auch äquivalente Ausführungszeiten validiert. Durch fehlende Informationen auf ESL bzw. eine abstraktere Darstellung kann es jedoch auch zu Abweichungen im Vergleich zum RTL-Design kommen. Dies kann aber vernachlässigt werden, da auf ESL in der Regel keine vollständig korrekten Ausführungszeiten wichtig sind. Der Grund dafür ist, dass Systemkonfigurationen in der Regel nur relativ miteinander verglichen werden und kleine Abweichungen dabei nicht ausschlaggebend sind.

Des Weiteren kann ein RTL-Design durch eine sogenannte *High-Level-Synthese* erzeugt werden, bei der das ESL-Design mittels Synthese in ein RTL-Design überführt wird. Hierbei

kann durch die Synthese sichergestellt werden, dass beide Designs äquivalent sind. In vielen Fällen wird der gegenteilige Ansatz zur *High-Level-Synthese* durchgeführt, der sogenannte *Bottom-Up-Approach*, bei dem das bestehende RTL-Design abstrahiert und somit in ein ESL-Design überführt wird. Dieser Prozess kann händisch durchgeführt werden, wobei der Abstraktionsgrad deutlich größer ist, aber auch Fehler in das ESL-Design einfließen können. Daher muss dieses gegen das RTL-Design validiert werden. Zusätzlich gibt es automatisierte Abstraktionsprozesse [10]. Die Äquivalenz der beiden Designs ist hier durch den Prozess abgesichert. Der Nachteil dieser Methode ist jedoch, dass der Abstraktionsgrad des ESL-Designs in der Regel sehr gering ist und oft das RTL-Design nur in die entsprechende ESL-Sprache übersetzt wird. Das Problem der Prozesse ist, dass sie nicht wissen, welche Parameter und Informationen sie filtern können bzw. dass sie nicht alle Möglichkeiten ausschöpfen können, da der Lösungsraum zu groß ist. Die so erstellten ESL-Designs haben durch ihre deutlich höhere Komplexität den Nachteil, dass sie die Simulationsgeschwindigkeit der Systemsimulation auf ESL deutlich verringern können. Wie die Synthese im Detail abläuft, hängt von der verwendeten Zielplattform ab und wird daher hier nicht näher erläutert.

Da sich das Design nach der Synthese für die selbe Komponente je nach Zielplattform, System und Platzierung stark unterscheiden kann, führt dies in der Regel zu starken Abweichungen bezüglich der Leistungsaufnahme. Das liegt daran, dass eine unterschiedliche Allokation von Technologieelementen stattfindet sowie die Zuordnung verschieden ist (Berechnungen werden auf anderen Elementen durchgeführt). Durch eine veränderte Platzierung von Elementen ändern sich auch die Leitungswege und dementsprechend die Leitungslängen sowie deren Leistungsaufnahme. Würde man auf dieser Basis eine Abstraktion der Leistungsaufnahmewerte durchführen, könnte nicht sichergestellt werden, dass die resultierenden Ausgabewerte des Systemmodells realistische Werte darstellen.

Um dem entgegenzuwirken kann die sogenannte *Design Preservation* auf die Systemkomponenten angewendet werden. Das bedeutet, dass das Design nach der Synthese nach GL fixiert wird und in den späteren Systemimplementierungen genau in dieser Form eingesetzt und benutzt wird. Dadurch kann sichergestellt werden, dass sich weder die Allokation, die Zuordnung zu den Elementen noch die Leitungslängen verändern. Lediglich die Verbindungen zwischen den Komponenten sind noch variabel, da so ein Designblock beliebig auf dem Chip verschoben werden kann. Der Nachteil bei dieser Variante ist jedoch, dass zum einen der Block von seinen Ausmaßen genau so benutzt werden muss, wie er synthetisiert wurde und zum anderen, dass er in Verbindung mit dem System nicht weiter optimiert werden kann (gemeinsame Nutzung von Ressourcen und Verkürzung der Signalleitungen zwischen den Komponenten). Dadurch kann die Leistungsaufnahme ansteigen und Platz auf dem Chip ungenutzt bleiben, da die bereits synthetisierten Blöcke keine bessere Platzierung erlauben.

Die *Design Preservation* wurde ursprünglich dafür entwickelt, bestimmte Eigenschaften eines synthetisierten Design-Blocks zu sichern. Dies sind zum Beispiel Zeiteigenschaften, die für den GL-Block gegen die Vorgaben validiert wurden. Da die meisten eingesetzten Systemkomponenten IP-Komponenten sind, die wiederholt instantiiert werden, wird für

diese bereits häufig die *Design Preservation* eingesetzt. Dadurch kann dieser Schritt die Synergie effektiv ausnutzen.

Ist eine *Design Preservation* nicht möglich, wird der *Place-and-Route* Schritt nicht durchgeführt. Stattdessen werden für die Leitungskapazitäten abhängig von der Technologie sowie der Anzahl von Elementen und Leitungen Durchschnittswerte abgeschätzt. Dies geschieht in der Regel mit einem sogenannten *Wire Load Modell* (siehe Benutzerhandbuch vom *Synopsys DesignCompiler Version k-2015.06-SP1*). Durch dieses Vorgehen wird die Abweichung zum unbekannten *Place-and-Route* Ergebnis größtmöglich minimiert.

### 5.2.2. Stimuli-Auswahl

Um alle relevanten Leistungsaufnahmezustände zu finden, ist es erforderlich, dass die Stimuliwerte die Systemkomponente so beeinflussen, dass sie alle Leistungsaufnahmezustände erreicht. Da vorher nicht bekannt ist, welche Leistungsaufnahmezustände es gibt und dies auch nicht gezielt ermittelt werden kann, werden in diesem Abschnitt Vorgehensweisen vorgestellt, die die Wahrscheinlichkeit zum Finden aller Leistungsaufnahmezustände erhöhen.

Eine Annahme ist, dass es eine starke Korrelation zwischen funktionalen Zuständen und den Leistungsaufnahmezustände gibt. Demnach kann man davon ausgehen, dass eine Simulation, die alle Pfade des Kontrollpfades abdeckt, die Wahrscheinlichkeit erhöht, alle Leistungsaufnahmezustände zu finden.

Bei IP-Komponenten ist dem Benutzer der Kontrollpfad in der Regel nicht bekannt, da er den inneren Aufbau der Komponente nicht oder nur zum Teil kennt. Aus diesem Grund sollten die Stimulivektoren von dem Designer der IP-Komponente generiert werden. Diese erstellt er in der Regel bereits für den funktionalen Test dieser Komponente. Stehen keine Stimulivektoren des Designers zur Verfügung, kann der Kontrollpfad genommen werden, der durch die PrSM beschrieben wird. Da in den meisten Fällen die PrSM nur eine Abstraktion des funktionalen Verhaltens darstellt, wie in Abschnitt 5.1 beschrieben, werden eventuell nicht alle existierenden Kontrollpfade simuliert, was zu einem ungenaueren Modell führen kann. Daher ist diese Möglichkeit nur dann zu wählen, falls keine Stimulivektoren vom Designer der IP-Komponente vorliegen und die innere Struktur der IP-Komponente ganz oder zum Teil unbekannt ist.

Ein wichtiger Punkt bei der Stimuliauswahl sind die internen funktionalen Zustandsübergänge, die nicht zu Änderungen an den Ein- und Ausgängen führen. Die Stimuliwerte sollten so gewählt werden, dass auch diese Zustandsübergänge ausgeführt und bei der Synthese mit betrachtet werden. Sind diese nicht bekannt, ist es Zufall, ob diese mit den gewählten Stimuliwerten erreicht und somit in der Synthese mit betrachtet werden. Trifft letzteres zu, kann dies zu einer nicht unerheblichen Abweichung im resultierenden Modell führen.

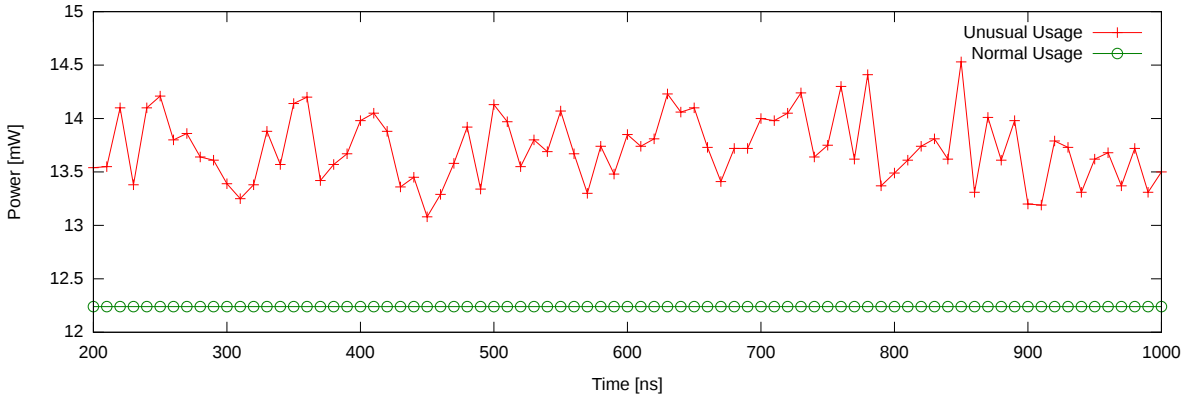



Abbildung 5.5.: Beispielhafte Verdeutlichung für eine Speicherkomponente, dass eine nicht spezifizierte Benutzung zu einer großen Abweichung in der Leistungsaufnahme führen kann. In diesem Beispiel werden in dem `Reset`-Zustand einmal alle anderen Eingangssignale konstant gehalten (normale, spezifizierte Benutzung) und einmal geändert (ungewöhnliche, unspezifizierte Benutzung).

In jedem Fall sollten Einschränkungen zur Benutzung der Komponente gemacht werden. Versuche haben gezeigt, dass eine ungewöhnliche Benutzung einer Komponente zu einer unvorhergesehenen Leistungsaufnahme führt. Ein Beispiel für eine Speicherkomponente ist in Abbildung 5.5 gezeigt. Hier wird deutlich, dass bei einer normalen Benutzung des `Reset`-Signals ohne Änderung der anderen Signale die Leistungsaufnahme konstant auf einem Wert bleibt. Ändert sich jedoch zum Beispiel das `Data_In`-Signal, zeigt die Leistungsaufnahme einen starken Einfluss auf diese Änderungen. Dies ist eine ungewöhnliche Benutzung der Komponente und kann, wenn sie nicht mit charakterisiert wurde, zu einer falschen Leistungsaufnahme führen. Demnach sollte für dieses Beispiel beschränkt werden, dass solange die Komponente im `Reset`-Zustand ist, sich kein anderes Kontroll- oder Datensignal ändern darf. Ein solche Einschränkung lässt sich beispielsweise mit Kontrakten festlegen [50]. Diese legen fest, für genau welches Verhalten die Leistungsaufnahme spezifiziert ist. Damit lässt sich bei der späteren Simulation des Leistungsaufnahmemodells feststellen, ob sich die entsprechende Systemkomponente außerhalb dieser Spezifikationen bewegt und eine Warnung oder ein Fehler angezeigt werden soll. Dies kann zum einen dazu benutzt werden, um entweder die Simulation entsprechend anzupassen (die Komponente wurde falsch benutzt) oder die Synthese der Komponente um diesen Fall zu erweitern (die Synthese war unvollständig). In vielen Fällen wird genau so eine Spezifikation mit Hilfe der Kontrakte ohnehin schon vorgenommen, um eine falsche Benutzung der Komponente zu verhindern und ein definiertes Verhalten für die Komponente zu garantieren.

Eine wichtige Komponente bei den Stimuliwerten ist die datenabhängige Leistungsaufnahme. Da Änderungen auf den Datenschnittstellen der Komponente einen sehr großen Einfluss auf die Leistungsaufnahme haben können, muss diese Abhängigkeit identifiziert und model-

liert werden. Wie Datenabhängigkeiten bei der Leistungsaufnahme in das Modell integriert werden können, wurde bereits in Abschnitt 4.8 erläutert. Wie oben beschrieben, sollte das Eingabeverhalten der Komponente eingeschränkt werden. Das bezieht sich insbesondere auf die Dateneingänge. Demnach müssen auch nur Zustände auf eine datenabhängige Leistungsaufnahme untersucht werden, in denen Änderungen auf den Datenschnittstellen erwartet werden. Beispielsweise wird beim lesenden Zugriff auf einem Speicher keine Datenänderung auf dem Dateneingang erwartet, sondern nur auf dem Adresseingang und dem Datenausgang. Diese müssen dann bei der Charakterisierung der datenabhängigen Leistungsaufnahme mit betrachtet werden.

Je nachdem, wie die datenabhängige Leistungsaufnahme charakterisiert werden soll, muss dies bei der Generierung der Stimuliwerte beachtet werden. Dabei gibt es zwei Möglichkeiten:

1. gleichzeitige Charakterisierung von kontrollpfadabhängiger und datenabhängiger Leistungsaufnahme sowie
2. separate Charakterisierung von kontrollpfadabhängiger und datenabhängiger Leistungsaufnahme.

#### Gleichzeitige Charakterisierung

Hierbei werden die Stimuliwerte so gewählt, dass zum einen alle funktionalen Zustände und Zustandsübergänge erreicht werden und zum anderen in datenabhängigen Zuständen die erwarteten Änderungen auf Dateneingängen dargestellt werden. Hierbei ist es wichtig, dass die Datenwerte den wirklichen Werten im Zielsystem entsprechen und keine Zufallswerte sind. Andernfalls kann es bei der Charakterisierung passieren, dass ungleiche Zustände in einem Zustands zusammengefasst werden. Diese Methode eignet sich für den Fall, dass die zustands- und datenabhängige Leistungsaufnahme gleichzeitig charakterisiert werden soll, so wie es bei der Modellierung der durchschnittlichen Leistungsaufnahme benötigt wird (siehe Abschnitt 4.8.1).

#### Separate Charakterisierung

Bei der separaten Charakterisierung werden mehrere Stimulisätze erstellt. Der erste ist für die zustandsabhängige Leistungsaufnahme. Es werden also Stimuliwerte gewählt, die alle funktionalen Zustände und Zustandsübergänge erreichen, aber für die Dateneingänge konstante Daten verwenden.

In den weiteren Stimulisätzen werden gezielt die datenabhängigen Zustände eingestellt und dann mit den erwarteten Datenänderungen auf den Dateneingängen simuliert. Dies ermöglicht eine separate Charakterisierung von zustands- und datenabhängiger Leistungsaufnahme. Diese Methode eignet sich für die Modellierung von bereichsweiser oder vollständiger datenabhängiger Leistungsaufnahme (siehe Abschnitt 4.8.2 und Abschnitt 4.8.3).

### 5.2.3. GL-Simulation

Stehen die Stimuliwerte fest, kann das GL-Design mit Hilfe dieser und einer passenden Testumgebung simuliert werden. In der Testumgebung muss darauf geachtet werden, dass ein entsprechendes Taktsignal angelegt wird und die Wertänderungen der Signale zum richtigen Zeitpunkt erfolgen. In der GL-Synthese wird für das Taktsignal eine Frequenz festgelegt, auf die das Design optimiert wird. Das heißt, dass der Ressourcenbedarf so niedrig wie möglich gehalten wird, um die Leistungsaufnahme gering zu halten, aber der kritische Pfad des Designs nicht zu lang wird (siehe Abschnitt 2.3.2). Aus diesem Grund sollte diese Frequenz auch in der Testumgebung eingestellt werden. Die Signalwechsel der Eingangssignale sollten möglichst mit der fallenden Taktflanke des Taktsignals zusammenfallen. Dadurch wird vermieden, dass Signalwechsel in der *Setup-* oder *Hold-Zeit* der Eingangsregister stattfinden. Finden in diesen Zeiten Signalwechsel statt, können die Register unerwartete Werte annehmen und zu einer falschen funktionalen Simulation führen.

Während der Simulation werden alle Signalwechsel im Inneren der Systemkomponente inklusive der aktuellen Simulationszeit aufgezeichnet und in einer sogenannten *Value Change Dump* (VCD)-Datei abgespeichert. Mit Hilfe dieser Aktivitätswerte, dem GL-Design und der Technologiebibliothek kann die Leistung berechnet werden, die über die Zeit von der Systemkomponente aufgenommen wird. Da die Signalverzögerung zwischen zwei Registern durch das GL-Design und die Technologiebibliothek bekannt sind, kann der genaue Zeitpunkt der Leistungsaufnahme berechnet werden. In dieser Arbeit wird die Leistungsaufnahme abhängig von den Ein- und Ausgaben betrachtet. Daher reicht hier ein taktgenauer Leistungsaufnahme-Trace, was die Berechnung des Traces beschleunigt.

Einige Werkzeuge wie der *Xilinx XPower Analyzer* bieten nur die Berechnung von durchschnittlichen Werten für die Leistungsaufnahme. Damit hier ein taktgenauer Leistungsaufnahme-Trace erzeugt werden kann, muss der Aktivitäts-Trace in die einzelnen Takte aufgeteilt und in eigene VCD-Dateien abgespeichert werden. Danach wird das Werkzeug einzeln mit der jeweiligen Datei aufgerufen. Dies führt zu unverhältnismäßig langen Berechnungszeiten für den Leistungsaufnahme-Trace auf GL. Eine Betrachtung der durchschnittlichen Leistungsaufnahmewerte für ganze Aktivitäts-Traces erlaubt aber keine Rückschlüsse auf die Abhängigkeit zwischen Eingabe-/Ausgabeverhalten und der Leistungsaufnahme. Der Vorteil der hier vorgestellten Charakterisierung ist, dass die Berechnung dieser Leistungsaufnahme-Traces nur einmal durchgeführt werden muss, da die Werte danach in abstrahierter Form im ESL-Modell integriert werden. Im anderen Fall wäre für jede unterschiedliche Systemkonfiguration eine erneute Berechnung der Leistungsaufnahme auf GL nötig und dieses nicht nur komponentenweise, sondern für das komplette System. Aus diesem Grund relativiert sich der Aufwand für die lange einmalige Berechnung der Leistungsaufnahme-Traces.

#### Auswahl der Leistungsaufnahmewerte

Die einzelnen Werkzeuge der Anbieter liefern nicht nur einen Leistungsaufnahmewert, sondern verschiedene Werte abhängig von der Art der Leistungsaufnahme und von der Art der Elemente, die diese Leistung aufnehmen. Dabei gibt es zusätzlich noch große Unterschiede zwischen ASICs und FPGAs.

Bei ASICs wird in der Regel zwischen den folgenden vier Werten der Leistungsaufnahme unterschieden, wobei hier die originalen Begriffe aus dem Werkzeug von *Synopsys PowerCompiler* benutzt werden:

- *Leakage Power* beschreibt die statische Leistungsaufnahme, die durch Leckströme verursacht wird, wie es in Abschnitt 2.2.1 beschrieben ist.
- *Cell Internal Power* ist der Teil der dynamischen Leistungsaufnahme, der beim Umschalten der Logikgatter aufgenommen wird.
- *Net Switching Power* ist der Teil der dynamischen Leistungsaufnahme, der beim Umschalten der Signalleitungen aufgenommen wird.
- *Dynamic Power* bezeichnet die gesamte dynamische Leistungsaufnahme, die durch Signaländerungen verursacht wird, wie in Abschnitt 2.2.2 beschrieben. Sie ergibt sich aus der *Cell Internal Power* und der *Net Switching Power*.

Bei der GL-Synthese auf einen ASIC wird das Design auf die Gatterelemente abgebildet und diese durch Signale miteinander verknüpft. Die Platzierung der Elemente und die Festlegung der Signalwege entsteht erst bei dem sogenannten *Place-and-Route* Schritt. Wird eine Abschätzung der Leistungsaufnahme vor diesem Schritt durchgeführt, sind die Längen der Signalleitungen unbekannt und es wird abhängig von der Anzahl der verwendeten Technologieelemente eine Standardlänge benutzt. Dies führt dazu, dass die Werte für die Leistungsaufnahme ungenauer werden. Bei der Betrachtung der durchschnittlichen Leistungsaufnahme fällt dies nicht ins Gewicht, da sich positive und negative Ungenauigkeiten gegenseitig ausgleichen und damit zu einer sehr kleinen, vernachlässigbaren Abweichung führen. Da auf ESL in den meisten Fällen ein relativer Vergleich zwischen Systemkonfigurationen erfolgt, werden keine sehr genauen Werte benötigt.

Im Unterschied zum ASIC sind auf einem FPGA alle Elemente schon fest platziert, womit ein *Place-and-Route* nicht nötig ist. Hier werden die Elemente nur aktiviert, konfiguriert und korrekt miteinander verknüpft, um das gewünschte Schaltungsdesign zu erzeugen. Somit kann bei der Berechnung der Leistungsaufnahme das entsprechende Signal bzw. die dazugehörigen Ressourcen und dementsprechend auch die Leistungsaufnahme einer spezifischen Semantik zugeordnet werden. Bei dem *Xilinx XPower Analyzer* wird beispielsweise unterschieden in:

- *Logic* ist analog zu oben beschriebener *Cell Internal Power* und
- *Signals* zu oben beschriebener *Net Switching Power*.

- *IOs* beschreibt die dynamische Leistungsaufnahme, die an den Eingabe- und Ausgabepins des FPGAs umgesetzt wird. Durch ihre große Kapazität ist dies ein entscheidender Anteil.
- *BRAMs* bezeichnet die dynamische Leistungsaufnahme von festen Speicherblöcken und
- *DSPs* die dynamische Leistungsaufnahme von Kombinatorik, die häufig in einem FPGA benötigte Berechnungen durchführen. Diese sind auf die jeweiligen Operationen optimiert und daher sehr effizient und performant.
- *Static Power* beschreibt die statische Leistungsaufnahme analog zu oben beschriebener *Leakage Power*.
- *Total* ist die gesamte Leistungsaufnahme inklusive dynamischer und statischer Leistungsaufnahme.

Wird eine Komponente einzeln auf einen FPGA synthetisiert, werden die Eingangs- und Ausgangsports auf die Eingangs- und Ausgangspins des Chips gelegt, da das Synthesewerkzeug annimmt, dass dies ein komplettes FPGA-Design darstellt und nur auf diesem Weg die Eingangs- und Ausgangspins der Komponente mit der Umgebung verbunden werden können. Dadurch werden die Eingabe- und Ausgabepins des FPGA bei der GL-Simulation mit betrachtet. Da beim späteren Systementwurf die Ports der Komponente mit anderen Komponenten verbunden sind und nur wenige Ports an die Pins des FPGA angebunden werden, darf diese Leistungsaufnahme bei der Synthese der PSM nicht mit betrachtet werden. Damit die entsprechende Leistungsaufnahme im späteren Systemdesign für die verbundenen Pins nicht fehlt, müssen diese gesondert charakterisiert und einberechnet werden.

## 5.3. Vorverarbeitung der Leistungsaufnahme-Traces

In Abschnitt 5.1 und Abschnitt 5.2 wurde beschrieben, wie eine vollständige PrSM erstellt und die für die Synthese nötigen Leistungsaufnahme-Traces auf GL erstellt werden können. In Abschnitt Abschnitt 5.4.1 werden eine manuelle und eine automatisierte Variante zur PSM-Synthese vorgestellt. Die manuelle Synthese ist für einfache Modelle geeignet, für die direkt die generierten GL-Leistungsaufnahme-Traces genutzt werden können. Für komplexere Modelle ist die automatisierte Variante vorzuziehen. Da die in Abschnitt 5.4 vorgestellten Algorithmen zur automatischen Synthese der PSM einen GL-Leistungsaufnahme-Trace nicht sinnvoll nutzen können, müssen vorher einige Schritte unternommen werden, um die Daten entsprechend aufzubereiten. Führt eine manuelle Synthese nicht zu einem Modell, welches die Genauigkeitsanforderungen an die Leistungsaufnahme erfüllt, können die Maßnahmen in diesem Abschnitt helfen, Fehler in dem erstellten Modell zu finden.

Ein PSM-Zustand repräsentiert eine Reihe von Intervallen auf einem Leistungsaufnahme-Trace auf GL. Im ersten Schritt müssen diese Intervalle bzw. die Zeitpunkte an denen ein

neues Intervall beginnt identifiziert werden. Ein solcher Zeitpunkt zeichnet sich dadurch aus, dass sich die Charakteristik des Leistungsaufnahmeverhaltens hier verändert. In Abschnitt 5.3.1 wird ein Verfahren vorgestellt, diese Zeitpunkte zu finden. Zusätzlich können die von der PrSM ausgegebenen PSM-Events hierfür genutzt werden. Der Grund dafür ist, dass PSM-Events immer dann ausgegeben werden, wenn sich der Protokollzustand ändert, was auch zu einer Änderung des funktionalen Verhaltens führt und demnach auch das Leistungsaufnahmeverhalten ändern kann. In Abschnitt 5.3.2 wird der Vorgang erläutert, wie die Zeitpunkte für das Auftreten der PSM-Events identifiziert werden können. Im nächsten Schritt müssen die verschiedenen Zeitpunkte aufeinander abgebildet werden, um eine Korrelation zwischen funktionalem Verhalten und Leistungsaufnahme zu erhalten. Die Zeitpunkte, an denen sich das Verhalten der Leistungsaufnahme ändert oder ändern kann (wenn zu diesem Zeitpunkt ein PSM-Event anliegt), wird nachfolgend als *Wechselpunkt* bzw. *Change Point* (CP) bezeichnet. Abschnitt 5.3.2 zeigt, wie die Zuordnung durchgeführt wird und worauf dabei geachtet werden muss. In Abschnitt 5.3.4 wird erläutert, wie die Charakteristik der einzelnen Intervalle zwischen den gefundenen Zeitpunkte erfasst wird und diese zusammen mit diesen Zeitpunkten und den dazugehörenden PSM-Events in einen sogenannten Segment-Trace überführt werden. Dieser ist die Grundlage für die Synthese der PSM. Das gesamte Vorgehen zur Vorverarbeitung des Leistungsaufnahme-Traces wird in Abbildung 5.6 gezeigt und an welcher Stelle des Syntheseprozess dieses Vorgehen angewendet wird.

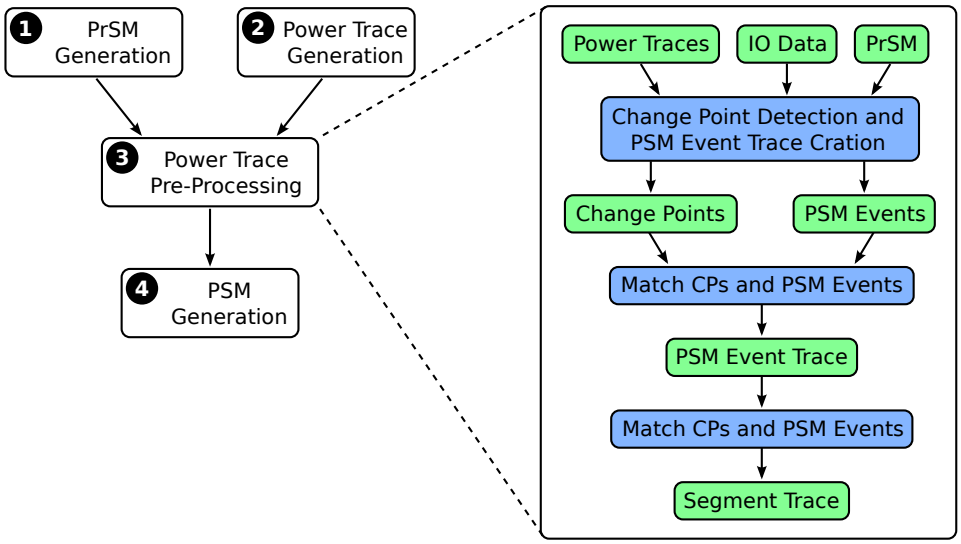


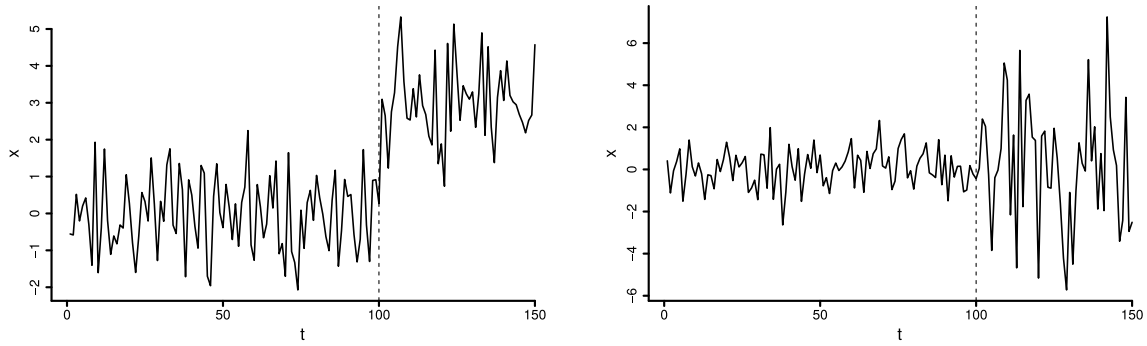

Abbildung 5.6.: Erzeugung des Segment-Traces mit Hilfe der CP-Detektion und des PSM-Event-Traces.

### 5.3.1. Detektion der Change Points

Damit die *Change Points* (CPs) in einem Leistungsaufnahme-Trace zuverlässig gefunden werden können, muss man verstehen, wodurch sich die Intervalle unterscheiden, die vor

und hinter dem CP liegen. Ein wichtiger Parameter ist der Mittelwert eines Intervalls, da die PSM-Zustände (ohne Betrachtung der datenabhängigen Ausgabe) später genau diesen Wert darstellen. Ein weiterer Parameter ist die Varianz in einem Intervall. Gibt es eine deutliche Änderung in der Varianz, weißt dies in der Regel auch auf ein anderes funktionales Verhalten hin, selbst wenn der Mittelwert gleich bleibt. Das liegt z.B. an Datenabhängigkeiten in der Leistungsaufnahme aber auch an verschiedenen Berechnungen innerhalb der funktionalen Komponente. Beispiele für die Änderung des Mittelwerts und der Varianz sind in Abbildung 5.7a und Abbildung 5.7b zu sehen.

Weitere Parameter wie z.B. die maximale Abweichung von dem Mittelwert und der maximale oder minimale Wert geben Auskunft über das Verhalten, helfen aber nicht bei dem Finden von Wechselpunkten, da sie sich nur auf einen bestimmten Zeitpunkt und nicht auf das gesamte Intervall beziehen. Des Weiteren treten diese Parameter auch eher sporadisch auf – der maximale Wert der Leistungsaufnahme kann sich z.B. abhängig von den verwendeten Daten für den gleichen Kontrollzustand in verschiedenen Intervallen deutlich unterscheiden.




(a) Änderung beim Durchschnittswert (b) Änderung bei der Varianz

Abbildung 5.7.: Einfache Beispiele für Änderungen auf einer univarianten Wertabfolge mit eingezeichneten CPs [60].

Das in diesem Abschnitt vorgestellte Verfahren basiert auf der von Ortland vorgestellten Methode zum Finden von CPs in Leistungsaufnahme-Traces [55]. Ortland hat mehrere Möglichkeiten untersucht, CPs zu identifizieren. Für die Sprache *R*, die für die Lösung von statistischen Problemen entwickelt wurden [56], existieren eine Reihe von Paketen die unter der Betrachtung folgender Parameter von Ortland [55] untersucht wurden:

- parameterlose Detektion und
- kurze Ausführungszeiten für lange Wertabfolgen.

Es wurden mehrere Pakete betrachtet, wobei das *cpm*-Paket von Ross [61] ausgewählt wurde, da es durch seine Implementierung in *C++* kurze Ausführungszeiten garantiert (alle anderen sind in *R* implementiert) und eine parameterlose Detektion auf Wertabfolgen mit unbekannter Verteilung der Werte anbietet [60].

Nachfolgend soll kurz erläutert werden, wie CPs detektiert werden wie von Ross beschrieben [60]. Es gibt zwei Möglichkeiten zur Identifikation: die sogenannte *Batch Detection*, bei der eine Sequenz mit fester Länge betrachtet wird und die sogenannte *Sequential Detection*, bei der die Sequenz sequentiell verarbeitet wird. Die *Batch Detection* arbeitet sehr gut, wenn nur wenige CPs in der Sequenz existieren. Die *Sequential Detection* basiert auf der *Batch Detection* und eignet sich hingegen gut, wenn die betrachtete Sequenz viele CPs enthält, da bei jedem neu gefundenen CP die Suche neu startet für die nachfolgende Sequenz.

#### Batch Detection

Bei der *Batch Detection* gibt es eine Sequenz mit einer festen Länge, die $n$ Werte $x_1, \ldots, x_n$ enthält. Diese Sequenz kann (aber muss nicht) einen CP enthalten. Zur einfacheren Darstellung wird angenommen, dass die Sequenz maximal einen CP enthält. Gibt es keinen CP, sind die Werte unabhängig und gleichverteilt gemäß einer Verteilung $F_0$. Ist ein CP vorhanden zur Zeit $\tau$, haben die Werte vor diesem Punkt die Verteilung $F_0$ und die $F_1$ nach dem Punkt, wobei $F_0 \neq F_1$.

Der erste Test, ob ein CP an einer bestimmten Position $k$ in der Wertabfolge existiert, wird durch zwei Hypothesen getestet:

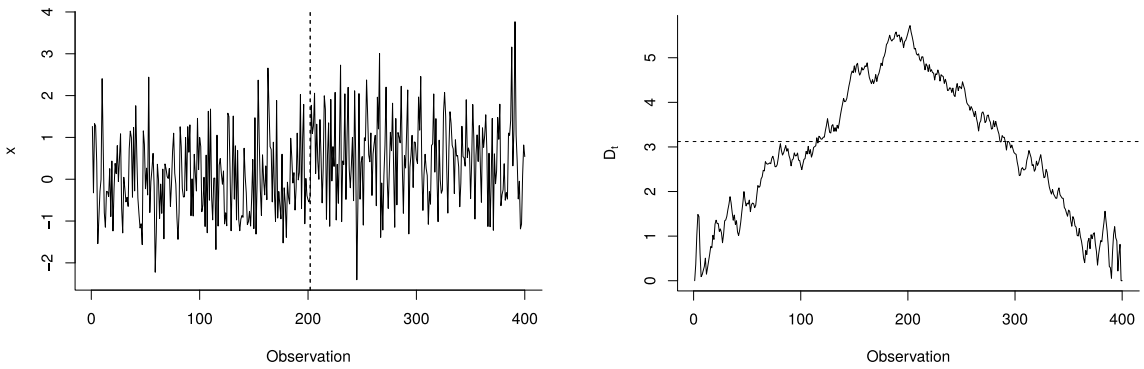
$$H_0 : X_i \sim F_0(x; \theta_0), \quad i = 1, \ldots, n \quad \text{und} \tag{5.1}$$

$$H_1 : X_i \sim \begin{cases} F_0(x; \theta_0), & i = 1, \ldots, k \\ F_1(x; \theta_1), & i = k+1, \ldots, n \end{cases} . \tag{5.2}$$

Dieses Problem ist ein Standardproblem, welches mit einem Zweistichproben-Hypothesentest gelöst werden kann, wobei die Wahl der Teststatistik von der angenommenen Verteilung (z.B. Gauß-Verteilung) der Wertabfolge abhängt und welche Art von Änderung detektiert werden soll. Wurde die Teststatistik gewählt, berechnet diese einen Wert $D_{k,n}$ und wenn dieser einen zuvor gewählten Schwellwert $h_{k,n}$ überschreitet, dann wird die Nullhypothese, dass beide Sequenzen eine identische Verteilung haben, verworfen und geschlussfolgert, dass ein CP direkt nach dem Wert $x_k$ existiert. Da es mehrere Möglichkeiten für einen CP gibt, wird $D_{k,n}$ für jeden Wert $1 < k < n$ ausgewertet und der maximale Wert genommen [60]. Ein Beispiel für den Verlauf von $D_{k,n}$ und $h_n$ ist in Abbildung 5.8b gezeigt, wobei die zugehörige Sequenz von Werten mit gefundenem CP in Abbildung 5.8a zu sehen ist. Es wird deutlich, dass der maximale Wert von $D_{k,n}$ als CP gewählt wurde.

#### Sequential Detection

Der Ansatz des Hypothesentests für zwei Sequenzen aus der *Batch Detection* kann für die *Sequential Detection* erweitert werden, bei der neue Werte nach und nach hinzugenommen werden und mehrfache CPs in der Sequenz existieren können. Im Folgenden bezeichnet $x_t$ den $t$-ten Wert, der betrachtet wird, mit $t \in \{1, 2, \ldots\}$.




(a) Wertabfolge mit detektiertem CP (b) $D_{k,n}$-Statistik

Abbildung 5.8.: Die linke Grafik zeigt eine Sequenz von Werten mit einem abgeschätzten CP bei der gestrichelten Linie, der mit dem Student-t und Mann-Whitney Test gefunden wurde. Die rechte Grafik zeigt den entsprechenden Wert $D_{k,n}$ und dem Schwellwert $h_n$ als gestrichelte Linie [60].

Immer, wenn ein neuer Wert hinzugenommen wird, wird die Wertabfolge $x_1, \ldots, x_t$ als feste Wertabfolge behandelt, wodurch sich die oben beschriebene *Batch Detection* darauf anwenden lässt. Mit Hilfe der Teststatistik wird der Wert $D_t$ berechnet und ein CP festgestellt, wenn $D_t \geq h_t$ für einen angemessenen Schwellwert $h_t$ ist. Wird kein CP festgestellt, wird der nächste Wert $x_{t+1}$ hinzugenommen, $D_{t+1}$ berechnet und mit dem Schwellwert $h_{t+1}$ verglichen und so weiter. Der Vorgang besteht demnach aus einer wiederholten Sequenz von Hypothesentests [60].

#### Anwendung auf Leistungsaufnahme-Trace

Für die Detektion der CPs bei einem Leistungsaufnahme-Trace wird die *Sequential Detection* genutzt, da Leistungsaufnahme-Traces in den meisten Fällen eine hohe Anzahl an Werten beinhalten. Der Grund dafür ist, dass für jedes Taktintervall ein Leistungsaufnahmewert generiert wird und die korrespondierenden Simulationen das komplette Leistungsaufnahmeverhalten der Komponente darstellen sollen, wie in Abschnitt 5.2.2 beschrieben. Zusätzlich ist bei einer hohen Anzahl von Werten entsprechend eine hohe Anzahl von CPs zu erwarten. Abhängig vom gewählten Design kann die Anzahl der CPs aber auch ungewöhnlich niedrig ausfallen, was zu sehr langen Ausführungszeiten bei der CP-Detektion führt. Das Problem wird am Ende dieses Abschnitts beschrieben und gezeigt, wie dies gelöst werden kann.

Die Wahl der zuvor beschriebenen Teststatistik hängt sowohl von der erwarteten Werteverteilung ab als auch davon, welche Änderung detektiert werden soll. Leistungsaufnahme-Traces verhalten sich meistens nicht nach einer spezifischen Verteilung, sondern eher willkürlich. Wie am Anfang dieses Kapitels beschrieben, soll sowohl eine Änderung des Mittelwerts als auch der Varianz ermittelt werden. Aus diesem Grund wird als Teststatistik der *Mann-Whitney*-Test [45] gewählt, da dieser eine unbekannte Verteilung annimmt, parameterlos ist und Änderungen beim Mittelwert sowie in der Varianz detektiert. Es wurden andere in

Frage kommende Teststatistiken getestet, wobei der *Mann-Whitney* die wenigsten falschen Treffer zeigt und eine hohe Zuverlässigkeit bei der Entdeckung vorhandener CPs aufweist.

Alle Teststatistiken haben Probleme bei der Detektion, wenn die Intervalle zwischen den CPs sehr klein sind (weniger als zehn Werte), wie in Abbildung 5.9 gezeigt. Dort wird deutlich, dass das Ende des Zustands `Key` nicht erkannt wird und dies zu einer Verschiebung von CP 2 führt. Der Grund, dass der erste CP nicht erkannt wird, liegt daran, dass bei jedem gefundenen CP eine neue Suche beginnt. Die erste Observation der Werte erfolgt aber erst nach minimal 20 Werten. Da immer der maximale Wert der Teststatistik genommen wird, kann es passieren, dass bei zwei oder mehreren CPs in diesem Intervall nicht zwingend der erste CP detektiert und somit vom Suchalgorithmus nicht gefunden wird. Die Verschiebung des zweiten Wertes erfolgt durch die Einberechnung der Werte vom Zustand `Key`, wodurch dies zu einer anderen Verteilung für das linke Intervall führt und sich ein anderer CP ergibt. Eine Lösung für dieses Problem kann in vielen Fällen die Verlängerung der jeweiligen Phase (hier der Zustand `Idle`) sein, die durch die Simulationsumgebung vom Benutzer gesteuert werden kann. Dadurch wird die Wahrscheinlichkeit minimiert, dass in einem Suchintervall mehrere CPs vorhanden sind. Eine Erklärung zu dem ersten ungültigen Wert erfolgt im nachfolgenden Abschnitt Besonderheiten.

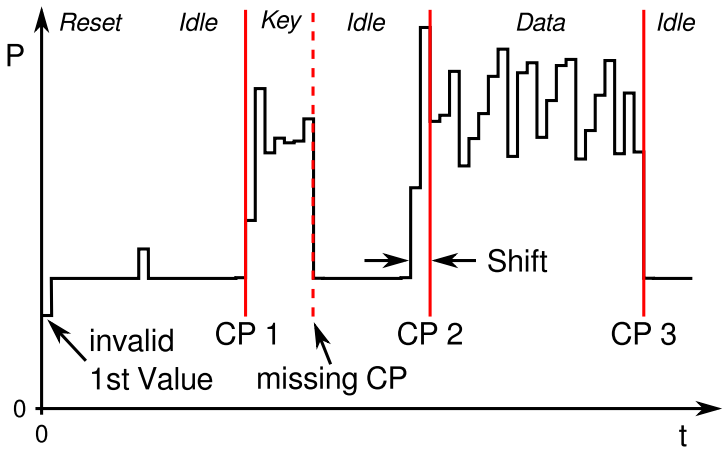



Abbildung 5.9.: Leistungsaufnahme-Trace von einer *Camellia*-Verschlüsselungskomponente und die gefundenen CPs. Durch zu kleine Intervalle zwischen CPs werden nicht alle CPs gefunden, was zu einer Verschiebung der anderen CPs führt. Der erste Wert des Traces ist ungültig (siehe Abschnitt Besonderheiten).

In Abbildung 5.10 wird ein Leistungsaufnahme-Trace gezeigt, bei dem CPs mit der *Sequential Detection* und dem *Mann-Whitney*-Test zuverlässig detektiert wurden. Es wird deutlich, wie sich die Änderungen von Mittelwert und Varianz auf die Detektion auswirken und wie sich der Vorgang in das gesamte Synthesevorgehen einbettet.

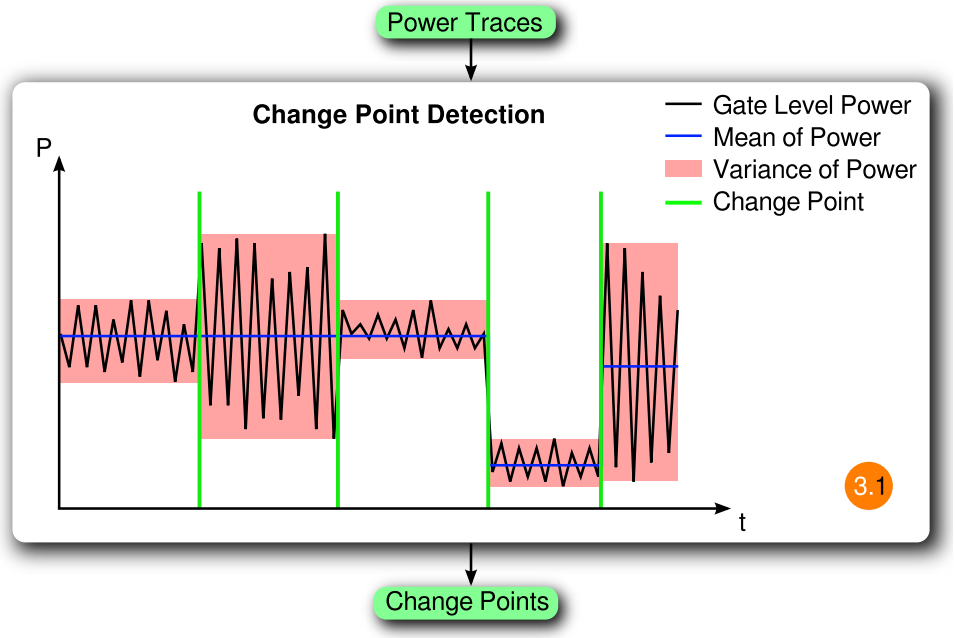



Abbildung 5.10.: Sequentielle Change Point-Detektion auf einem Leistungsaufnahme-Trace mit *Mann-Whitney*-Teststatistik in Abhängigkeit von Mittelwert und Varianz mit gefundenen CPs.

#### Besonderheiten

Beim Finden der *Change Points* (CPs) gibt es einige Besonderheiten zu beachten, damit genaue Ergebnisse in kurzer Zeit berechnet werden können. Die erste Besonderheit betrifft den ersten und letzten Wert eines Leistungsaufnahme-Traces. Diese können unter Umständen eine deutlich geringere Leistungsaufnahme als die anderen Werte haben und den Trace dadurch verfälschen. Dadurch werden zum einen eventuell CPs gefunden, die nicht existieren und zum anderen gibt es eine große Abweichung des minimalen Wertes – dies kann einen Einfluss auf nachfolgende Statistikberechnungen haben, die den minimalen Wert nutzen.

Der Grund für eine deutlich zu niedrige Leistung ist, dass in den entsprechenden Intervallen keine positive Taktflanke vorhanden ist, wie in Abbildung 5.11 dargestellt. Wie in Abschnitt 5.2.3 beschrieben, wird für die Erstellung des Leistungsaufnahme-Traces immer ein Intervall mit der Länge einer Taktperiode betrachtet. Beginnt in der GL-Simulation das Taktsignal mit dem Wert 1, wäre die erste positive Taktflanke theoretisch zur Zeit 0 s. Da hier aber in der Regel keine Signalwechsel stattfindet, sondern nur die Initialisierung der Signale und Variablen, wird die erste positive Taktflanke erst nach einer Taktperiode ausgeführt. Diese wird aber bereits von dem zweiten Intervall für den Leistungsaufnahme-Trace betrachtet. Am Ende der Simulation fehlt eine positive Taktflanke genau dann, wenn das betrachtete Intervall auf der fallenden Taktflanke beginnt. Hört die Simulation

zwischen fallender und steigender Taktflanke oder genau zu dem Zeitpunkt der steigenden Taktflanke auf, wird die positive Taktflanke nicht mehr ausgeführt. Da die betrachteten Systeme synchrone digitale Systeme sind, die nur bei einer positiven Taktflanke Werte in Register übernehmen, findet aus dem funktionalen Gesichtspunkt in den oben beschrieben Szenarien keine Aktivität statt. Dadurch fehlt in diesem Intervall auch die Signalaktivität und die resultierende Leistung ist damit ungültig. Um diesem falschen ersten und letzten Wert entgegen zu wirken, können diesen Zeitpunkten des Leistungsaufnahme-Traces der nachfolgende bzw. vorhergehende Wert zugewiesen werden. Da diese Änderung aber eventuell eine Verfälschung des Leistungsaufnahme-Traces herbeiführt, wird dieses Vorgehen nicht empfohlen.

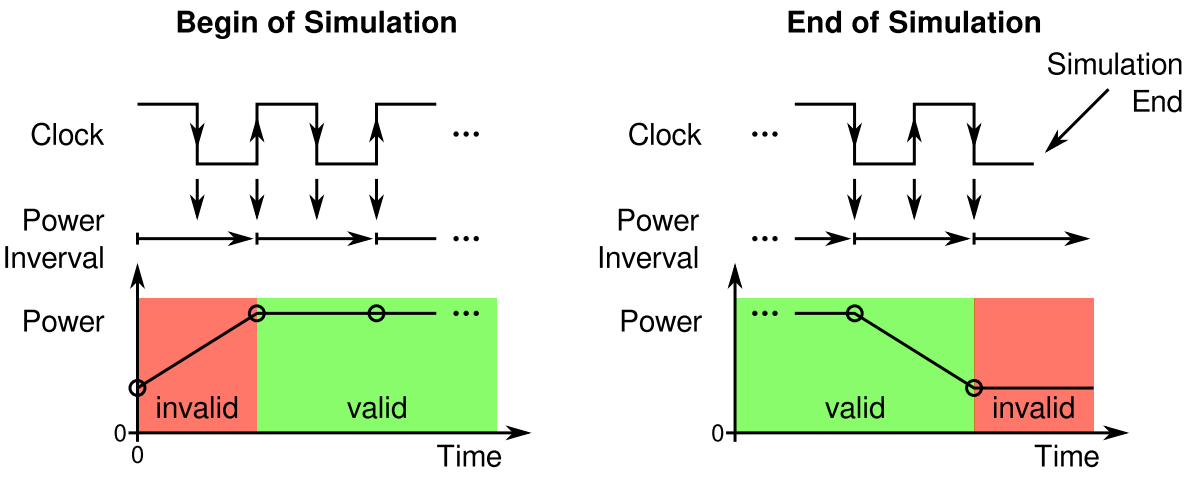



Abbildung 5.11.: Ungültige Werte am Anfang oder Ende eines Leistungsaufnahme-Traces durch das Fehlen einer fehlenden positiven Taktflanke im betrachteten Intervall.

Eine bessere Alternative ist, die entsprechenden Werte bei den nachfolgenden Berechnungen zu ignorieren bzw. zu entfernen. Dabei ist zu beachten, dass bei allen korrespondierenden Traces (z.B. Eingabe-/Ausgabe-Trace) diese Werte auch ignoriert oder entfernt werden, damit es nicht zu einer falschen Verknüpfung oder Verschiebung zwischen den betrachteten Traces kommt.

Eine weitere Besonderheit tritt bei besonders langen Leistungsaufnahme-Traces auf, die über lange Intervalle (mehr als 100 000 Werte) die gleiche Verteilung der Werte aufweisen. Dies führt dazu, dass das zu prüfende Intervall besonders groß wird, da dieses immer bei dem letzten gefundenen CP beginnt und inkrementell vergrößert wird, bis der nächste CP gefunden wird. Bei großen Intervallen führt dies dazu, dass alle Zwischenwerte als möglicher CP getestet werden und für alle jeweils die Verteilung für das Intervall vor und nach dem CP berechnet werden, wodurch eine große Menge an Berechnungen mit vielen betrachteten Werten durchgeführt wird. Für jeden Vergleich müssen zwei Verteilungen berechnet werden, einmal für das Intervall vor und einmal für das Intervall nach dem möglichen CP. Für jedes Inkrementieren des untersuchten Intervalls erhöht sich die Be-

rechnungsanzahl um zweimal die Größe des letzten Intervalls. Damit ergibt sich für eine Intervallgröße von $n$ Elementen ohne CP folgende Anzahl von Berechnungen $b_n$:

$$b_n = \sum_{i=1}^{n-1} i \cdot 2 = n \cdot (n+1) \quad . \tag{5.3}$$

Für ein Suchintervall von z.B. 500 000 Werten führt dies zu etwa 250 Milliarden Berechnungen, was zu nicht praktisch nutzbaren Berechnungszeiten führt, da auch die Berechnungszeiten mit größer werdenden Intervallen mit ansteigen.

Um dem entgegen zu wirken, wird nachfolgend beschriebenes Vorgehen benutzt. Ist der gesamte Leistungsaufnahme-Trace größer als 1 000 Werte, wird immer nur in einem Intervall von 200 Werten gesucht. Dabei ist es wichtig, dass sich die Intervalle überschneiden (hier wurden 100 Werte gewählt), da eine gewisse Menge an Werten nötig ist, um eine zuverlässige Verteilung festzustellen. Daher beginnt der Algorithmus zum Finden der CPs auch erst nach 20 Werten mit der Suche nach CPs. Werden ein oder mehrere CPs in dem entsprechenden Intervall gefunden, dient der letzte gefundene CP als Beginn des neuen Suchintervalls. Damit wird das Verhalten des Suchalgorithmus aufgegriffen und führt hier zu keinen Verfälschungen. Diese Optimierung führt für das PPM-Beispiel aus Abschnitt 6.2.5 mit 2,5 Millionen Werten und 16 CPs dazu, dass die Berechnungszeit von ursprünglich ca. 3 Stunden auf 30 Sekunden gesenkt werden kann, ohne das Ergebnis zu verändern. Das oben genannte Intervall mit 500 000 Werten stammt aus diesem Beispiel. Im Vergleich dazu benötigt das Einlesen der 2,5 Millionen Werte eine Zeit von 1:40 Minuten.

Schaut man sich nun die Anzahl der Berechnungen mit der optimierten Methode für das Beispielintervall mit den 500 000 Werten an, ergeben sich nun 2 500 Intervalle mit einer Größe von 300 Werten (200 plus 100 Überschneidung), was insgesamt zu folgender Berechnungsanzahl führt:

$$2\,500 \cdot 300 \cdot 301 = 225\,750\,000 \quad .$$

Dadurch ist die Berechnungsanzahl etwa 1 100× kleiner und zusätzlich die Berechnungszeit kürzer, da weniger Werte in den zu berechnenden Verteilungen betrachtet werden.

Die Ergebnisse mit verschiedenen Beispielen zeigen, dass die gleichen CPs mit und ohne Optimierung gefunden werden, es bei wenigen Ausnahmen aber zu einer Verschiebung von einem Takt kommt. Dies ist darauf zurückzuführen, dass durch das Reduzieren der Intervalle die Verteilung sich minimal unterscheiden kann. Diese Verschiebung wirkt sich aber nicht negativ aus, da durch die Natur der Leistungsaufnahme-Traces die CP schwer einem direkten Zeitpunkt zugeordnet werden können. Für CPs, die mit einem PSM-Event zusammentreffen, wird diese Verschiebung implizit durch das im nächsten Abschnitt vorgestellte Verfahren kompensiert. Im anderen Fall führt die Verschiebung um einen Takt zu einer vernachlässigbar kleinen Abweichung.

### 5.3.2. Zuordnen von CPs und PSM-Events

Wie oben beschrieben, werden neben den Leistungsaufnahme-Traces auch die von der PrSM ausgegebenen PSM-Events genutzt, um die Zeitpunkte zu finden, an denen sich das Leistungsaufnahmeverhalten verändert. Dafür wird die PrSM, wie in Abschnitt 5.1 beschrieben, simuliert. Ihr werden dafür die Eingabe- und Ausgabewerten der funktionalen GL-Simulation zugeführt, die bei Bedarf durch einen Transaktor nach ESL umgeformt werden (siehe Abschnitt 5.1.1). Daraus ergibt sich ein sogenannter PSM-Event-Trace.

Zusätzlich existiert zu diesem Zeitpunkt der Trace aus CPs, die in dem in Abschnitt 5.3.1 beschriebenen Prozess detektiert wurden. Diese CPs werden in der Regel durch eine Änderung des funktionalen Verhaltens hervorgerufen. Dieses Verhalten ist zum Teil auch im Eingabe- und Ausgabeverhalten zu sehen, welches durch die PSM-Events dargestellt wird. Daher ist es wichtig, die Verknüpfungen zwischen den PSM-Events und den entsprechenden CPs zu finden. Da die Analyse der CPs nur auf der Mittelwert- und Varianzänderung beruht, kann es hier in Bezug auf das funktionale Verhalten einen zeitlichen Versatz geben. Daher wird folgendes Vorgehen angewendet, um die PSM-Events und die CPs zueinander zuzuordnen.

Es wird über alle vorhandenen PSM-Events in zeitlich aufsteigender Reihenfolge iteriert und in einem vorher definierten Intervall $\triangle_s$ um den Zeitpunkt des PSM-Events nach CPs gesucht. Für ein PSM-Event $e$, das zur Zeit $t_e$ auftritt, und die Taktperiode $t_{\text{clk}}$ ergeben sich der Beginn $\triangle_{\text{start},e}$ und das Ende $\triangle_{\text{end},e}$ für das Intervall $\triangle_s$:

$$\triangle_{\text{start},e} = t_e - \frac{\triangle_s}{2}, \quad \triangle_{\text{end},e} = t_e + \frac{\triangle_s}{2}, \quad \text{mit} \quad \triangle_s = 2 \cdot i \cdot t_{\text{clk}} \quad \text{und} \quad i \in \mathbb{N}. \tag{5.4}$$

Die Variable $i$ wird anfangs auf den Wert 0 gesetzt, wodurch nur zu dem Zeitpunkt des Auftretens des PSM-Events $e$ gesucht wird. Nach jedem Durchlauf wird $i$ um 1 erhöht. Dies passiert so oft, bis sie entweder bei $i_{\text{max}}$ angekommen ist oder bereits alle CPs zugeordnet wurden. Durch dieses Verfahren wird sichergestellt, dass immer der nächstliegendste CP gefunden wird. Haben zwei PSM-Events den gleichen zeitlichen Abstand zu einem CP, wird das frühere PSM-Event zugeordnet. Der Grund ist, dass eine Änderung des funktionalen Verhaltens eine Änderung der Leistungsaufnahme nach sich führt und nicht umgekehrt. Der Wert $i_{\text{max}}$ wird standardmäßig auf den Wert 2 gesetzt, da sich die meisten Verschiebungen in diesem Intervall befinden.

Bei diesem Vorgang kann es nun passieren, dass CPs oder PSM-Events nicht zugeordnet wurden. Nicht zugeordnete PSM-Events können bedeuten, dass

1. dieser Zustandswechsel des funktionalen Verhaltens keinen Einfluss auf das Leistungsaufnahmeverhalten hat oder dass
2. die Detektion der CPs auf dem Leistungsaufnahme-Trace einen CP nicht erkannt hat.

Beide Fälle können ignoriert werden. Bei 1. ist keine Änderung des Leistungsaufnahmeverhaltens vorhanden und kann somit auch nicht falsch charakterisiert werden. Zusätzlich

sorgt dieses PSM-Event aber dafür, dass die generierte PSM möglichst alle PSM-Events verarbeitet und vollständig ist (siehe Abschnitt 5.4.4). Bei 2. sorgt dieses PSM-Event dafür, dass bei der Synthese (beschrieben in Abschnitt 5.4.1) dieser Zeitpunkt als möglicher CP erkannt wird.

Gibt es einen oder mehrere nicht zugeordnete CPs kann es für diese bedeuten, dass:

1. ein CP detektiert wurde, der keiner ist,
2. Vektoren von Datenports sich ändern, von denen die Leistungsaufnahme abhängig ist (diese führen nicht zu einem PSM-Event),
3. der CP außerhalb des durch $i_{\max}$ definierten Suchintervalls lag,
4. das entsprechende funktionale Verhalten nicht in der PrSM modelliert ist und deshalb kein PSM-Event zu diesem Zeitpunkt ausgegeben wurde oder
5. eine interne funktionale Zustandsänderung, die nicht an den Ein- und Ausgängen sichtbar ist, für diese Änderung der Leistungsaufnahme verantwortlich ist.

Wird ein CP detektiert, der keiner ist, muss der CP ignoriert werden. Eine Statistiktest der davorliegenden und nachfolgenden Werteverteilung zeigt hier eine sehr geringe Abweichung. Diese hat aber schon dazu geführt, dass die Teststatistik (siehe Abschnitt 5.3.1) diesen als CP erkannt hat.

Wenn Vektoren auf Datenports sich ändern, kann das zwei Gründe haben. Entweder dass sie für eine korrekte funktionale Ausführung nicht konstant gehalten werden können (wie in Abschnitt 5.2.2 beschrieben) oder dass diese versehentlich geändert wurden. Ist die Änderung notwendig, kann dieser CP auch ignoriert werden. Wurden die Daten versehentlich geändert, muss das Verhalten in den Stimuli-Werten korrigiert und wieder bei Schritt (1.3) (siehe Abschnitt 5.2.3 und Abbildung 5.1) fortgefahren werden.

Einen CP, der durch ein zu kleines Suchintervall nicht zugeordnet wurde, kann man dadurch identifizieren, dass dieser einen nur wenig größeren zeitlichen Abstand zum nächstgelegenen PSM-Event hat als durch das durch $i_{\max}$ definierte Intervall. Oft tritt dieses Verhalten systematisch auf oder in Einzelfällen durch eine seltene spezifische Konstellation im funktionalen Verhalten. Ist der Grund ein systematisches Verhalten, muss $i_{\max}$ so weit erhöht werden, bis die entsprechenden CPs zugeordnet werden. Danach muss das oben beschriebene Verfahren für die Zuordnung erneut durchgeführt werden. Ist der Grund eine seltene Konstellation im funktionalen Verhalten, kann eine Zuordnung manuell erfolgen oder auch, wie vorher beschrieben, indem das Suchintervall vergrößert wird. In diesem Fall muss darauf geachtet werden, dass dadurch keine falsche Zuordnung von CPs zu PSM-Events durchgeführt wird. Um dieses Fehlverhalten zu umgehen, kann nur das Suchintervall für ein spezifisches PSM-Event vergrößert werden.

Wurde ein Fehler bei der Erstellung der PrSM gemacht, gibt es in den meisten Fällen signifikante Änderungen im Eingabe- oder Ausgabeverhalten, die keine Zustandsänderung in der PrSM auslösen. Ist dies der Fall, muss die PrSM entsprechend angepasst und, wie am Anfang dieses Abschnitts beschrieben, ein neuer PSM-Event-Trace erzeugt werden.

Ist eine interne funktionale Zustandsänderung, die nicht an den Ein- und Ausgängen sichtbar ist, für den CP verantwortlich, gibt es häufig mehrere nicht zugeordnete CPs, deren Auftreten immer in einem bestimmten Intervall nach dem identischen PSM-Event erfolgt. Diese CPs müssen berücksichtigt werden, um das entsprechende Leistungsaufnahmeverhalten später in der PSM modellieren zu können. Die CPs werden mit dem auslösenden PSM-Event und dem Intervall zwischen PSM-Event und CP abgespeichert und der nachfolgenden Synthese, die in Abschnitt 5.4.1 beschrieben ist, zur Verfügung gestellt. Die Analyse wird im nachfolgenden Abschnitt detailliert beschrieben.

### 5.3.3. Abhängigkeitsanalyse

Wie im vorherigen Abschnitt beschrieben, können nicht zugeordnete *Change Points* (CPs) dann auftreten, wenn es interne funktionale Zustandsänderungen gibt, die nicht an den Ein- oder Ausgängen sichtbar sind. Damit diese angemessen in der PSM-Synthese betrachtet werden können, müssen zuvor die vorhandenen Abhängigkeiten gefunden werden. Als Grundlage für diesen Abschnitt dient die These, dass der CP im Verhalten der Leistungsaufnahme durch eine Änderung des funktionalen Zustands herbeigerufen wird. Da die PrSM eine abstrahierte Version des funktionalen Verhaltens darstellt, wird diese repräsentativ für das funktionale Verhalten eingesetzt. Eine an den Komponentenschnittstellen nicht sichtbare Zustandsänderung kann nur dann in der PSM modelliert werden, solange sie von den Ein- oder Ausgaben abhängt. Diese Abhängigkeit ist im Gegensatz zu den bisher betrachteten eine zeitlich verzögerte Reaktion der funktionalen Komponente auf eine Ein- oder Ausgabe. Das heißt, es muss herausgefunden werden, ob eine solche Abhängigkeit existiert, welches die relevante Ein- oder Ausgabe ist und wie groß die zeitliche Verzögerung ist. Für den nachfolgenden Vorgang wird von den konkreten Ein- und Ausgaben abstrahiert und stattdessen der PSM-Event-Trace aus Abschnitt 5.3.2 als Repräsentant für die funktionale Ausführung genutzt. Das Vorgehen sieht folgendermaßen aus:

Da zeitlich verzögerte Reaktionen von dem funktionalen Zustand abhängen (verschiedene Zustände können unterschiedliche zeitlich verzögerte Reaktionen hervorrufen), werden im ersten Schritt allen nicht zugeordneten CPs die PrSM-Zustände annotiert, die zu dem gegenwärtigen Zeitpunkt des CP aktiv waren.

Im zweiten Schritt wird bei jedem dieser CPs geschaut, wie groß das Intervall zwischen Auftreten des CP und den letzten Auftreten aller PSM-Events war. Dadurch ergibt sich für jedes bereits aufgetretene PSM-Event ein spezifisches Intervall, welches die potentielle zeitliche Verzögerung darstellt. Diese Werte werden auch dem CP zugeordnet. Alle noch nicht aufgetretenen PSM-Events werden nicht betrachtet, da, wie bereits beschrieben, eine Änderung des funktionalen Verhaltens eine Änderung der Leistungsaufnahme nach sich führt.

Intervalle werden im Folgenden als gleich bezeichnet, wenn sie den gleichen Wert besitzen bzw. der Unterschied sich unter einem definierten Schwellwert befindet. Dieser ist in der Regel der gleiche Wert, wie das im vorherigen Abschnitt definierte Suchintervall $\triangle_s$,

das Ungenauigkeiten bei der CP-Suche ausgleicht. Im nächsten Schritt werden für jeden PrSM-Zustand, dem CPs zugeordnet wurden, alle dem Zustand zugeordneten CPs betrachtet und jeweils die Intervalle für ein PSM-Event miteinander verglichen.

Gibt es für genau ein PSM-Event für alle Intervalle den gleichen Wert, ist ein von den Ein- und Ausgängen abhängiges Verhalten sehr wahrscheinlich. Damit dieser CP bei der nachfolgenden automatischen PSM-Synthese mit betrachtet werden kann, wird für diesen ein eindeutiges CP-PSM-Event erstellt und an den Zeitpunkten des Auftretens der PSM-Events im PSM-Event-Trace eingefügt. Dieses Event wird zusätzlich mit dem Verzögerungsintervall annotiert und mit dem PSM-Event, von dem es abhängig ist. Abbildung 5.12 zeigt grafisch die zeitliche Abhängigkeit zwischen PSM-Event und CP.

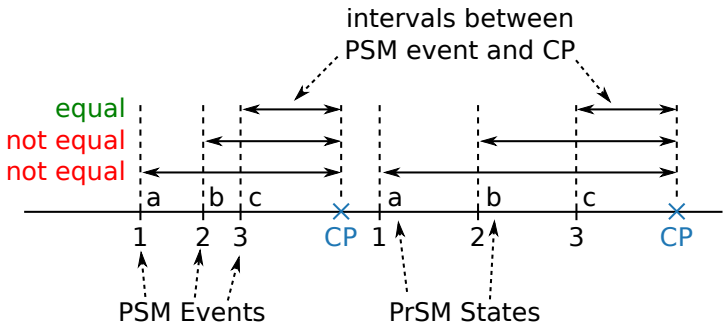



Abbildung 5.12.: Eindeutige Abhängigkeit zwischen PSM-Event und zeitlich verzögertem CP, der einen Wechsel der Leistungsaufnahmecharakteristik anzeigt.

Gibt es für mehrere PSM-Events für alle Intervalle den gleichen Wert, kann diese Änderung von all diesen PSM-Events abhängen. In den meisten Fällen kommt dies zustande, da die Simulationsumgebung für ähnliche Abläufe gleiche Intervalle benutzt und daher nur zufällig für mehrere PSM-Events alle Intervalle den gleichen Wert haben. Ist ein solches Verhalten durch das Kommunikationsprotokoll vorgegeben, können alle PSM-Events als Grundlagen genommen werden. Dann kann, wie oben beschrieben, für eines der PSM-Events ein CP-PSM-Event erstellt und in den PSM-Event-Trace eingefügt werden. Im anderen Fall sollten die Intervalle der Komponentensimulation verändert werden, sodass sie sich nun unterscheiden und wieder bei Schritt 1.3 (Abschnitt 5.2.3) begonnen werden. Damit werden zufällige Korrelationen vermieden.

In einigen Fällen ist nicht das letzte Auftreten eines PSM-Events der Auslöser für das zeitlich verzögerte Verhalten, sondern das erste Auftreten in einer Reihe aufeinanderfolgender Auftreten des selben PSM-Events. Dies kommt genau dann vor, wenn ein Zustandsübergang den selben Zustand als Start und Ziel hat und damit ein PSM-Event mehrfach direkt hintereinander von der PrSM ausgegeben werden kann. In diesem Fall ist das Intervall zu dem ersten Auftreten des PSM-Events in der Reihe immer identisch. Aus diesem Grund wird hier das CP-PSM-Event mit dem Verzögerungsintervall zum ersten Auftreten des auslösenden PSM-Events in den PSM-Event-Trace eingefügt. Diese Abhängigkeit wird in Abbildung 5.13 gezeigt.

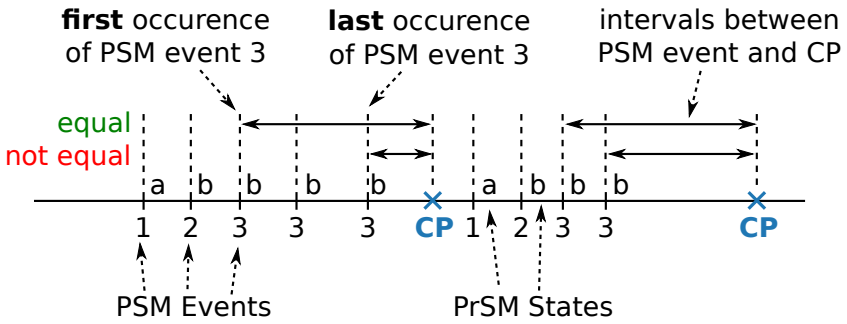



Abbildung 5.13.: Eindeutige Abhängigkeit zwischen erstem PSM-Event einer Reihe gleicher PSM-Events und zeitlich verzögertem CP, der einen Wechsel der Leistungsaufnahmecharakteristik anzeigt.

In den bisherigen Überlegungen wurde stets davon ausgegangen, dass es zu jedem PrSM-Zustand maximal einen zeitlich verzögerten Zustandsübergang geben kann. Gibt es mehrere, zeigt es sich in der Regel dadurch, dass nicht genau ein PSM-Event gefunden werden kann, welches das gleiche Intervall für alle Auftreten aufweist. Bei $n$ verschiedenen zeitlich verzögerten Reaktionen für einen PrSM Zustand kann die Menge der CPs in $n$ Teile geteilt werden, wobei es für jeden Teil ein eindeutiges PSM-Event gibt, für das alle Intervalle gleich sind. Wichtig dabei ist, dass es nicht das gleiche PSM-Event für verschiedene zeitlich verzögerte Reaktionen geben kann, da hiermit kein deterministisches Verhalten abgeleitet werden kann.

Treffen vorherig beschriebene Bedingungen auf kein PSM-Event zu, kann kein deterministisches zeitlich verzögertes Verhalten gefunden und somit auch nicht modelliert werden. Aus diesem Grund sollte der CP nicht weiter betrachtet und alle Auftreten des CPs entfernt werden.

Am Ende dieses Schrittes besteht der PSM-Event-Trace also zusätzlich zu den PSM-Events noch aus den oben beschriebenen CP-PSM-Events, die das deterministische zeitlich verzögerte Verhalten der funktionalen Komponente abbilden.

### 5.3.4. Generierung des Segment-Traces

In den letzten Abschnitten wurde gezeigt, wie relevante CPs gefunden werden können und diese mit den entsprechenden Ereignissen im Eingabe- und Ausgabe-Trace verknüpft werden können. Weiterhin wurde beschrieben, wie ein von den PSM-Events abhängiges und zeitlich verzögertes Verhalten der Leistungsaufnahme identifiziert und als spezielles Event in den PSM-Event-Trace eingefügt werden kann. In diesem Abschnitt wird beschrieben, wie die Events aus dem PSM-Event-Trace genutzt werden, um einen sogenannten Segment-Trace zu erzeugen.

Die Events aus dem PSM-Event-Trace unterteilen den Leistungsaufnahme-Trace in einzelne Intervalle, für die jeweils ein eigenes Segment erstellt wird. Ein Segment beschreibt die Leistungsaufnahmecharakteristik des zugehörigen Intervalls. Wie in Abschnitt 4.8 beschrieben, gibt es in vielen Modellen eine kontrollpfadabhängige und datenabhängige Leistungsaufnahme. Da in einem Segment sich der Kontrollzustand nicht ändert, kann die davon abhängige Leistungsaufnahme durch einen konstanten Wert dargestellt werden. Wie bereits gezeigt, ist die datenabhängige Leistungsaufnahme in den meisten Fällen linear abhängig von aktuellen oder vergangen Werten der Dateneingänge bzw. -ausgänge. Demnach wird jedes Segment in Bezug auf die kontrollpfadabhängige und datenabhängige Leistungsaufnahme charakterisiert und das Ergebnis an des jeweilige Segment annotiert.

Zusätzlich wird jedes Segment mit dem PSM-Event annotiert, welches am Beginn des jeweiligen Intervalls zu finden ist. Das bedeutet, dass dem ersten Segment kein PSM-Event zugeordnet wird, da dieses Intervall nie durch ein PSM-Event eingeleitet wird. Zusätzlich zu diesen Werten können noch weitere Werte wie der maximale und minimale Wert oder andere benötigte statistische Werte an die einzelnen Segmente annotiert werden. Diese werden nicht für die Synthese benötigt, können aber hierdurch an die erstellte PSM annotiert und anschließend in formalen Analysen oder für die Absicherung der Werte genutzt werden.

Ein Segment ist also definiert durch:

$$seg := (e, P_c, p_d(D)), \tag{5.5}$$

wobei $e \in E$ das einleitende PSM-Event ist, $P_c$ der konstante Wert für die kontrollpfadabhängige Leistungsaufnahme und $p_d(D)$ die Funktion, die die datenabhängige Leistungsaufnahme von den betrachteten Daten $D$ beschreibt.

Daraus ergibt sich der Segment-Trace $T_{seg}$ als zeitliche sortierte Liste aus Segmenten:

$$T_{seg} := (seg_1, \ldots, seg_n) \tag{5.6}$$

wobei $n$ die Anzahl der Segmente angibt. Dieser Segment-Trace dient im nachfolgenden Abschnitt als Ausgangsbasis für die Erstellung der PSM.

Nachfolgend wird beschrieben, wie $p_d$ berechnet wird.

#### Berechnung der Funktion für die datenabhängigen Leistungsaufnahme

Wie in Abschnitt 4.8 beschrieben, kann die datenabhängige Leistungsaufnahme für viele Komponenten durch die schaltenden Bits auf den Dateneingängen und -ausgängen, also der HD, ausgedrückt werden. Dabei können sowohl die letzten Wertänderungen als auch die davor aufgetretenen Wertänderungen relevant sein. Zwischen der datenabhängigen Leistungsaufnahme und der HD besteht in der Regel ein linearer Zusammenhang. Damit die datenabhängige Leistungsaufnahme genau beschrieben werden kann, muss identifiziert werden, welche Dateneingänge und -ausgänge die Leistungsaufnahme beeinflussen, wie

lang die betrachtete Historie der Wertänderungen sein muss und die Linearparameter müssen berechnet werden, die eine Gewichtung der einzelnen Werte festlegen.

Als Grundannahme gilt, dass alle Dateneingänge und -ausgänge einen Einfluss auf die Leistungsaufnahme haben können und eine Historie der Länge drei ausreichend ist, da eine längere Historie ìn allen Untersuchungen dieser Arbeit keine Verbesserungen herbeigeführt hat. Hat der Benutzer zusätzliches Wissen über das funktionale Verhalten, kann er die Historienlänge für einzelne oder alle Dateneingänge- und ausgänge entsprechend anpassen.

Für das dem Segment zugeordnete Intervall werden nun anhand der linearen Regression die entsprechenden Linearparameter berechnet. Ist die HD für einen Dateneingang oder -ausgang im ganzen Intervall bei null bzw. ändert sich nicht über das ganze Intervall, kann die lineare Regression für diese Daten keinen passenden Linearparameter finden, wodurch der resultierende Linearparameter mit hoher Wahrscheinlichkeit falsch ist und für nachfolgende Schritte entsprechend gekennzeichnet wird. Dateneingänge und -ausgänge, die verschiedene HDs in dem Intervall aufweisen und dennoch einen sehr geringen Linearparameter zugeordnet bekommen (kleiner als 1% des maximalen Linearparameters), haben keinen Einfluss auf die datenabhängige Leistungsaufnahme in diesem Segment und können ignoriert werden.

## 5.4. Generierung der PSM

Basierend auf der in Abschnitt 5.1 erstellten PrSM und dem in Abschnitt 5.3 erstellten Segment-Trace wird in diesem Abschnitt erklärt, wie die PSM in einem iterativen Prozess konstruiert, bewertet und optimiert wird. Bei der Erstellung der PSM wird grundsätzlich unterschieden zwischen der Charakterisierung des kontrollflussdominierten und des datenabhängigen Leistungsaufnahmeverhaltens. Im ersten Schritt wird eine PSM erstellt, die kein datenabhängiges Leistungsaufnahmeverhalten modelliert, was in Abschnitt 5.4.1 beschrieben wird. Dabei können ein oder mehrere mögliche Umsetzungen entstehen, die dann simuliert werden (Abschnitt 5.4.3) und anschließend wie in Abschnitt 5.4.4 beschrieben bewertet werden. Dabei kann festgestellt werden, dass das Design ein datenabhängiges Verhalten der Leistungsaufnahme aufweist, dessen Charakterisierungsvorgehen in Abschnitt 5.4.2 beschrieben wird oder dass das Modell an einigen Stellen noch Fehler oder Ungenauigkeiten aufweist. Nach jeder Änderung des PSM-Modells erfolgt eine weitere Simulation und Bewertung des Modells, bis die Benutzeranforderungen in Bezug auf Genauigkeit und Simulationsperformanz erfüllt sind. Abschnitt 5.4.5 beschreibt, wie das weitere Vorgehen aussieht, wenn die Genauigkeitsanforderung des PSM-Modells noch nicht erfüllt sind.

Die einzelnen Schritte der PSM-Generierung und deren Bewertung werden in Abbildung 5.14 gezeigt undwie sie sich in dem gesamten Syntheseprozess einfügen.

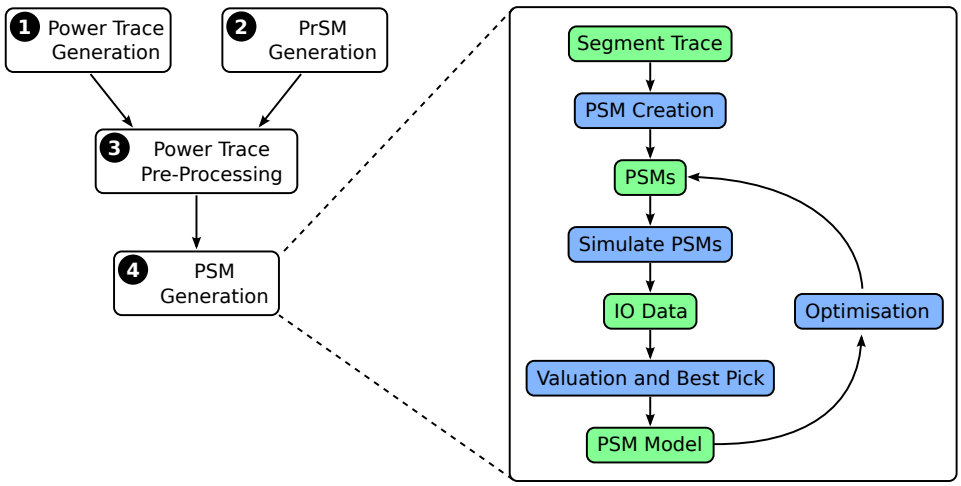



Abbildung 5.14.: Erstellung und Bewertung der PSM.

### 5.4.1. Synthese der PSM

Nachdem alle Vorbereitungsmaßen durchgeführt wurden, kann die PSM erstellt werden. Hier liegt der Fokus auf einer möglichst automatisierten Erstellung. Am Ende dieses Abschnitts wird näher auf das Vorgehen für eine manuelle Erstellung der PSM eingegangen.

Wie bereits in Abschnitt 5.1 beschrieben, dient die PrSM bzw. ihre Ausgaben, die durch die ausgewählten Stimuli aus Abschnitt 5.2.2 erzeugt wurden, als Grundlage für die Synthese der PSM.

Damit eine möglichst genaue Modellierung der Leistungsaufnahme erzielt werden kann, ist es wichtig, dass folgende Bedingungen für die PrSM erfüllt sind:

- durch das in Abschnitt 5.1 vorgestellte Verfahren wurde eine vollständige und korrekte PrSM erstellt,
- jeder Zustandsübergang ist mit einem eineindeutigen PSM-Event annotiert, da sonst verschiedene Zustände fälschlicherweise in der PSM zusammengelegt werden könnten sowie
- bei den Stimuliwerten muss darauf geachtet werden, dass die Annahmen bezüglich des Startzustands für alle Stimulivektoren erfüllt sind und diese mit Startzustand der PrSM übereinstimmen, wie es in Abschnitt 5.2.2 beschrieben wurde.

Sind diese Bedingungen erfüllt, kann sicher gestellt werden, dass das Verhalten der Leistungsaufnahme korrekt modelliert wird. Aus diesem Grund werden diese Bedingungen in den nachfolgenden Abschnitten als erfüllt angenommen.

An dieser Stelle sei noch einmal erwähnt, dass eine vollständige Modellierung in Bezug auf die datenabhängige Leistungsaufnahme aufgrund der hohen Komplexität in der Regel nicht möglich ist, da nicht alle Kombinationen von Wertänderungen an den Dateneingängen simuliert werden können (siehe Abschnitt 5.2.2). Wie gezeigt, reicht ein Untermenge der

Vektoren aus, um die Abweichung so weit zu reduzieren, dass die resultierenden Modelle den Zielen dieser Arbeit entsprechen.

Eine weitere Grundlage für die Synthese ist, dass mindestens ein oder besser mehrere Anwendungsfälle zur Verfügung stehen, die, wie in Abschnitt 5.2.2 beschrieben, möglichst das gesamte funktionale Verhalten der Komponente abdecken. Eine Bewertung erfolgt durch die in Abschnitt 5.2.2 beschriebenen Möglichkeiten. Für alle zur Verfügung stehenden Anwendungsfälle müssen

- GL-Simulationen durchgeführt und Leistungsaufnahme-Traces berechnet werden (Abschnitt 5.2.3),
- die Leistungsaufnahme-Traces vorverarbeitet werden (Abschnitt 5.3.1),
- mit der PrSM die PSM-Event-Traces erzeugt werden (Abschnitt 5.3.2) und
- aus vorverarbeiteten Leistungsaufnahme-Traces und PSM-Event-Traces die daraus resultierenden Segment-Traces erzeugt werden (Abschnitt 5.3.4), die als Grundlage für die Synthese der PSM dienen.

#### Automatisierte PSM-Synthese

Wie in der Einleitung von Abschnitt 5.4 beschrieben wurde, wird hier das Vorgehen vorgestellt, wie eine PSM erstellt wird, die das kontrollflussdominierte Verhalten der Leistungsaufnahme modelliert. Dafür wird als Grundlage das von Ortland [55] vorgestellte Verfahren zur automatischen Synthese genutzt und erweitert. Dieses baut auf dem von Koskimies vorgestellten Verfahren der automatischen Synthese von Zustandsautomaten aus Sequenzdiagrammen [37] auf.

Die Grundidee des vorgestellten Ansatzes ist, jedes Segment als einzelnen Zustand zu betrachten und dass alle aufeinanderfolgenden Segmente zu einer Verknüpfung der entsprechenden Zustände durch einen Zustandsübergang führen. Dabei wird der Zustandsübergang mit dem PSM-Event des zweiten Zustands annotiert, da dieses Event diesen Zustandsübergang auslöst.

Die Aufgabe der automatischen Synthese ist nun, die Segmente zu identifizieren, die das gleiche Leistungsaufnahmeverhalten abbilden und diese in den selben Zustand abzubilden. Um bei der Synthese eine saubere Differenzierung der Zustände zu erreichen, werden die Zustände sowohl nach ihrer kontrollflussdominierten als auch nach ihrer datenflussabhängige Leistungsaufnahme (siehe Abschnitt 5.4.2) charakterisiert und miteinander verglichen. Dafür ist es notwenig, dass der zugrundeliegende Leistungsaufnahme-Trace mit Stimuliwerten erzeugt wurden, die entsprechende Datenabhängigkeiten aufweisen. Datenabhängigkeiten in der Leistungsaufnahme werden nur für die Schnittstellen betrachtet, bei denen es signifikante Änderungen gibt. Andernfalls führt die Regression der Parameter zu falschen Werten.

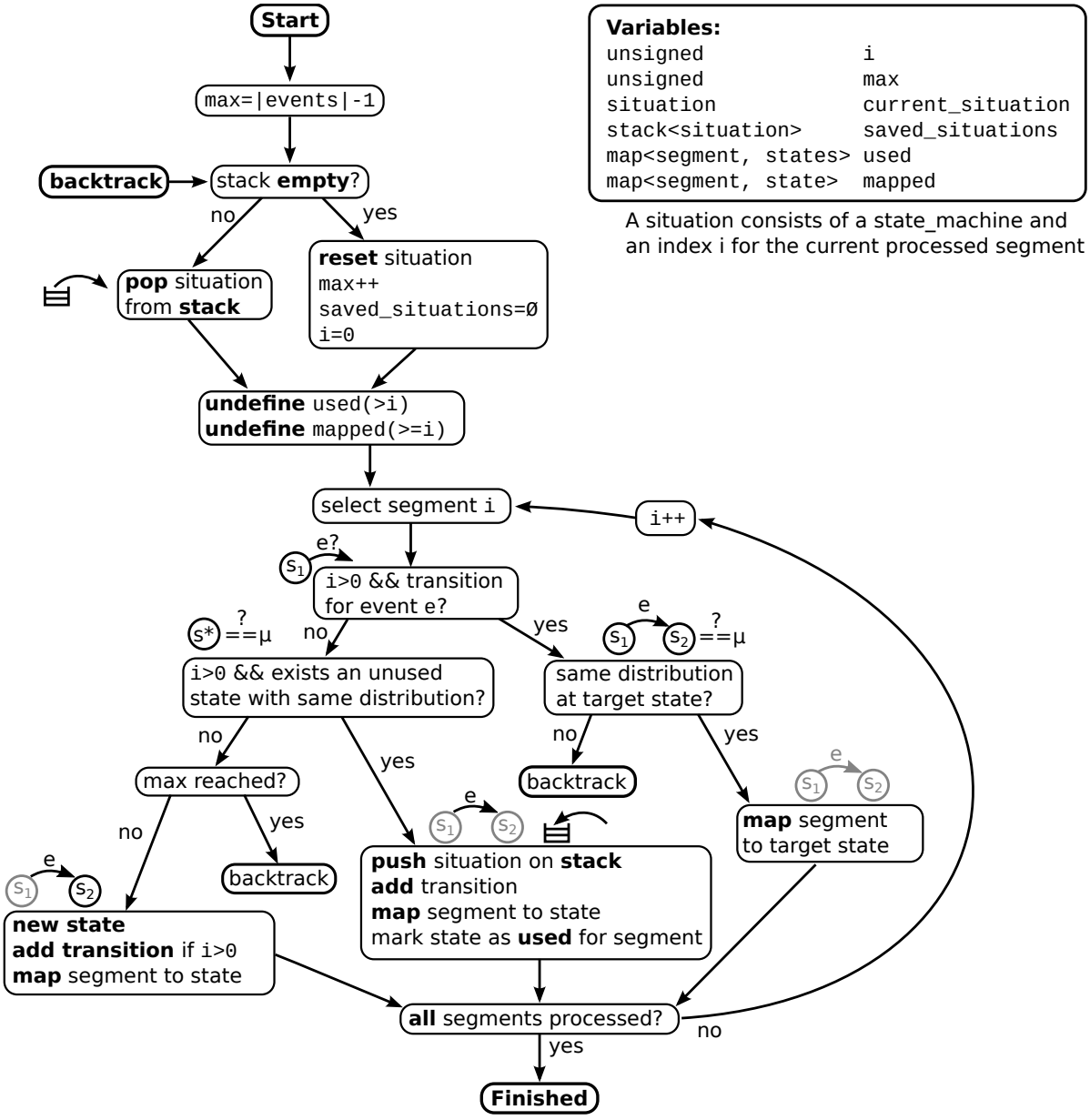

Start
max=|events|-1
backtrack
stack empty?
no
yes
pop situation from stack
reset situation
max++
saved_situations=∅
i=0
undefine used(>i)
undefine mapped(>=i)
select segment i
i++
e?
i>0 && transition for event e?
s* ?==μ
no
yes
e
s1 s2 ?==μ
i>0 && exists an unused state with same distribution?
same distribution at target state?
no
yes
no
yes
max reached?
backtrack
e
s1 s2
map segment to target state
no
yes
e
s1 s2
backtrack
push situation on stack
add transition
map segment to state
mark state as used for segment
e
s1 s2
new state
add transition if i>0
map segment to state
all segments processed?
no
yes
Finished
Variables:
unsigned i
unsigned max
situation current_situation
stack<situation> saved_situations
map<segment, states> used
map<segment, state> mapped
A situation consists of a state_machine and an index i for the current processed segment

Abbildung 5.15.: Synthesealgorithmus zum Erstellen der PSM auf Basis der generierten PSM-Event-Traces und GL-Leistungsaufnahme-Traces.

Das genau Vorgehen der automatisierten Synthese ist in Abbildung 5.15 gezeigt. An dieser Stelle wird anstelle von Pseudo-Code eine Grafik gezeigt, um das Verständnis zu verbessern.

Für den Synthesealgorithmus werden folgende Variablen benutzt, um die aktuelle Situation und die offenen Möglichkeiten zu speichern:

- `i`: beschreibt das gerade betrachtete Segment, welches in die PSM überführt werden soll.
- `max`: beschreibt die maximale Anzahl an möglichen Zuständen.
- `current_situation`: stellt die gegenwärtige PSM dar.
- `saved_situations`: speichert alle vorherigen PSMs, die für das *Backtracking* gebraucht werden.
- `used`: speichert, für welches Segment welche Zustände bereits betrachtet wurden.
- `mapped`: ordnet jedem bereits betrachteten Segment den entsprechenden PSM-Zustand zu.

Die Synthese beginnt damit, alle benötigten Variablen zu initialisieren. Die Variable `max` wird auf die Anzahl der Events gesetzt, da anfangs davon ausgegangen wird, dass die Anzahl der PSM Zustände der Anzahl der PSM-Events entsprechen wird.

Da noch keine Zustände definiert sind, werden die Zuordnungen in `used` und `mapped` zurückgesetzt und im ersten Durchlauf (`i==0`) nur der erste Zustand erstellt. Danach beginnt eine Schleife, die solange durchlaufen wird, bis alle Segmente einem Zustand zugeordnet werden konnte.

Im ersten Schritt der Schleife wird geschaut, ob es für den aktuellen Zustand bereits einen Zustandsübergang gibt, an der das gleiche Event annotiert ist, wie an das Segment. Ist dies der Fall, muss dieser Zustandsübergang ausgewählt werden, da ein deterministischer Automat generiert werden soll. Da nur Segmente in dem gleichen Zustand zusammengeführt werden sollen, die die gleiche Charakteristik aufweisen, wird nun geprüft, ob der Zielzustand die gleiche Charakteristik besitzt. Wie dies geprüft wird, ist weiter unten beschrieben. Sind die Charakteristika gleich, wird das Segment dem Zustand mit Hilfe von `mapped` zugewiesen und das nächste Segment ausgewählt. Ist die Charakteristik verschieden, kann das Segment nicht dem Zustand zugeordnet werden. Da der bestehende Zustandsübergang dies erfordert, um einen deterministischen Automaten zu erstellen, wird hier das später beschriebene *Backtracking* ausgeführt, um in einem vorherigen Schritt eine andere Verzweigung auszuwählen.

Gibt es im aktuellen Zustand keinen Zustandsübergang mit dem Event aus dem aktuell betrachteten Segment, wird geschaut, ob es bereits einen Zustand gibt, dessen zugeordnete Segmente die gleiche Charakteristik aufweisen. Bei mehreren möglichen Zuständen wird ein zufälliger Zustand ausgewählt. Nun wird die aktuelle PSM auf den Stack

`saved_situations` gepackt. Dadurch kann die aktuelle Situation wieder hergestellt werden, wenn mit dem ausgewählten Zustand in der weiteren Folge keine PSM gebildet werden kann. Dies wird weiter unten beim *Backtracking* beschrieben. Als nächstes wird ein Zustandsübergang mit dem PSM-Event des Segments vom aktuellen zum ausgewählten Zustand erzeugt und das Segment dem ausgewählten Zustand zugeordnet. Dieser Zustand wird mit Hilfe von `used` als *genutzt* für dieses Segment markiert, damit dieser Zustand nach dem *Backtracking* nicht wieder ausgewählt wird. Danach wird das nächste Segment ausgewählt und verarbeitet.

Sind die beiden vorherigen Bedingungen (existierender Zustandsübergang für PSM-Event oder noch nicht genutzter Zustand mit gleicher Leistungsaufnahmecharakteristik) nicht erfüllt, muss ein neuer Zustand erzeugt werden, solange die Anzahl der Zustände das in `max` definierte Maximum nicht übersteigt. Ist das Maximum erreicht, wird auch hier das *Backtracking* ausgeführt. Im anderen Fall wird ein neuer Zustand angelegt und ein entsprechender Zustandsübergang vom aktuell ausgewählten Zustand zum neuen Zustand hinzugefügt. Das Segment wird dem neuen Zustand zugeordnet und das nächste Segment verarbeitet.

In den drei oben beschriebenen Möglichkeiten wird immer der zuletzt zugeordnete Zustand zum neuen aktuellen Zustand der für das nächste Segment betrachtet wird.

Wird das sogenannte *Backtracking* ausgeführt, konnte entweder für das gesetzte Maximum keine deterministische PSM gefunden werden oder für einen in der Folge ausgewählten bestehenden Zustand. Ist der Stack `saved_situations` leer, wurden alle mit dieser Anzahl möglichen Kombinationen getestet. Daher muss in diesem Fall das Maximum `max` erhöht werden. Ist der Stack nicht leer, bedeutet das, dass noch nicht alle möglichen Kombinationen getestet wurden. Demnach wird die letzte Situation, in der ein Zustandsübergang zu einem bestehenden Zustand erstellt wurde, hergestellt und fortgefahren. Da mit Hilfe von `used` festgelegt ist, welche Zustände bereits getestet wurden, können nun Zustände mit gleicher Charakteristik ausgewählt werden, die noch nicht betrachtet wurden. So werden alle Möglichkeiten getestet, bis ein neuer Zustand erstellt werden muss. Da durch das Zurücksetzen der PSM die Zuordnungen, die in `used` und `mapped` aufgebaut wurden, ihre Gültigkeit verlieren, werden die entsprechenden Werte gelöscht.

Wie oben beschrieben, überprüft der Synthesealgorithmus bei dem Zusammenlegen von mehreren Segmenten in einen Zustand, ob diese die gleiche Verteilung aufweisen. Hier werden nun zwei Dinge unterschieden: Zustände, die eine konstante Ausgabe haben und Zustände, die eine von den Daten abhängige dynamische Ausgabe haben. Nur Segmente, die hier die gleiche Eigenschaft aufweisen, können zusammen in den gleichen Zustand überführt werden.

Für Segmente mit einer konstanten Leistungsaufnahme wird die Gleichheit der Verteilung folgendermaßen überprüft. Vom Benutzer wird vorher eine Schwelle festgelegt, die angibt, wie groß die Differenz zwischen den Werten der konstanten Leistungsaufnahme der verglichenen Segmente sein darf, damit diese zusammen gelegt werden. Der Grund dafür ist, dass die Leistungsaufnahme immer kleine Unterschiede aufweist und daher ein direkter

Vergleich dazu führen würde, dass jedes Segment in einen eigenen Zustand überführt wird. Ein solcher Automat wäre mit hoher Wahrscheinlichkeit nicht vollständig (siehe Abschnitt 5.4.4) und daher nicht einsetzbar. Als weiterer Vergleichsparameter kann die Varianz in den Segmenten genutzt werden. Auch hier wird eine vom Benutzer festgelegte Schwelle für die Differenz verwendet. Dieses Vorgehen ist dann sinnvoll, wenn der Synthesealgorithmus fälschlicherweise verschiedene Zustände in den selben Zustand überführt, weil die Mittelwerte der verglichenen Segmente zu nah aneinander liegen. Gibt es aber einen deutlichen Unterschied in der Varianz, kann hier die korrekte Differenzierung herbeigeführt werden.

Bei dem Zusammenlegen von Segmenten mit dynamischer Ausgabe ist der Vergleich etwas schwieriger, da er von den konkreten Ausgabefunktion abhängt. Im Folgenden wird der Vergleich an der hier benutzten linearen Funktion auf Basis der HD gezeigt.

Für die verglichenen Segmente wird analysiert, ob die Leistungsaufnahme von genau den selben Daten abhängt. Ist dies der Fall, werden die Parameter einzeln miteinander verglichen. Zum einen wird hier der Konstantwert von den einzelnen Segmenten verglichen und zum anderen die Linearparameter. Liegen alle Differenzen unter den vom Benutzer festgelegten Schwellen, werden die Zustände zusammengelegt.

Im Laufe des Algorithmus werden mehrere Segmente einem Zustand zugeordnet. Das bedeutet, dass ein neues Segment mit mehreren anderen Segmenten verglich werden muss. Aus diesem Grund werden die bereits zugeordneten Segmente zusammen betrachtet und ein resultierender Wert bzw. eine Ausgabefunktionen berechnet. Die Ausgaben der Zustände wird somit immer dann aktualisiert, sobald ein neues Segment zugeordnet wird. Dies ist erforderlich, da, wie bereits oben beschrieben, die Leistungsaufnahme immer leicht variiert und durch den Vergleich von einzelnen Segmenten eventuell gleiche Segmente als unterschiedlich klassifiziert werden könnten.

Da der Synthesealgorithmus alle Möglichkeiten versucht, für eine vorgegebene maximale Anzahl von Zuständen eine Automatenumsetzung zu finden, garantiert dieser Algorithmus, dass der resultierende Automat minimal ist. Dies sorgt zum einen dafür, dass die PSM übersichtlicher ist und dadurch einfacher vom Benutzer optimiert werden kann und zum anderen, dass die Wahrscheinlichkeit auf eine vollständige PSM steigt (siehe Abschnitt 5.4.4). Der Grund ist, dass für Segmente, die fälschlicherweise nicht zusammen gelegt werden, die Wahrscheinlichkeit sinkt, dass auf mehreren Zuständen alle Pfade durchlaufen werden und damit Zustandsübergänge in der resultierenden PSM fehlen.

Beispielsweise gibt es einen Zustand, der Zustandsübergänge hat für die PSM-Events `a` und `b`. Erzeugt die Synthese hier nun zwei Zustände von denen der eine einen Zustandsübergang für PSM-Event `a` hat und der andere Zustand einen Zustandsübergang für PSM-Event `b`, kann im ersten Zustand trotzdem PSM-Event `b` auftreten, welches dann nicht verarbeitet werden kann.

Da das zeitlich verzögerte Verhalten durch hinzufügte CP-PSM-Events im Segment-Trace mit dargestellt wird, werden diese implizit bei automatisierten Synthese mit betrachtet und in die resultierende Automatenbeschreibung integriert. Nach der Synthese müssen die

CP-PSM-Events durch Bedingungen für einen zeitgesteuerten Zustandsübergang ersetzt werden. Dabei werden die ermittelten Zeiten der Abhängigkeitsanalyse aus Abschnitt 5.3.3 verwendet.

In diesem Abschnitt wurde gezeigt, dass diese Arbeit ein automatisiertes Vorgehen zur PSM-Synthese zur Verfügung stellt und damit eine einfache Anwendung und Herleitung des Modells ermöglicht. Dadurch erfüllt die Arbeit die Anforderung N2 zur einfachen Benutzung.

#### Manuelle PSM-Synthese

Wenn die automatische Synthese nicht zum gewünschten Ziel führt oder der Benutzer sie nicht durchführen möchte, kann die manuelle Synthese der PSM gewählt werden. Dafür gibt es mehrere Möglichkeiten. In einigen Fällen ist die Komplexität der Komponente sehr gering und der Benutzer kann bereits anhand des Leistungsaufnahme-Traces erkennen, welche Zustände existieren und die PSM intuitiv erstellen. Ist dieser Fall nicht gegeben, wird folgendes Vorgehen angewandt: Als Basis wird eine PSM erstellt, die genau der PrSM entspricht. Das heißt, sie haben die gleichen Zustände und Zustandsübergänge. Demnach verhält sich die PSM in diesem Zustand äquivalent zur PrSM. Für jeden PSM-Zustand wird geschaut, wann dieser aktiv ist und von allen GL-Leistungsaufnahmewerten aus diesen Zeitintervallen der Durchschnittswert berechnet. Dieser wird dann als Ausgabewert an den entsprechenden PSM-Zustand annotiert.

Im nächsten Schritt wird geschaut, wo Diskrepanzen auftreten. Dafür wird das erstellte PSM-Modell (PrSM und PSM) mit den generierten Stimuli aus Abschnitt 5.2.2 wie in Abschnitt 5.4.3 beschrieben simuliert und die Ausgabe-Traces der PSM aufgezeichnet. Diese werden dann nach der in Abschnitt 5.4.4 beschrieben Methode analysiert. Dort zeigt sich, welche Zustände zusammengeführt werden können und welche Zustände große Abweichungen aufweisen. Diese Abweichungen können vom Benutzer bewertet werden, damit er entsprechende Änderungen an der PSM durchführen kann. Diese sind:

- Datenabhängige Ausgabe einfügen, wenn die Ausgabe abhängig von den Dateneingängen und -ausgängen stark schwankt.
- Zählvariablen einführen, wenn erkennbar ist, dass nach einer bestimmten Anzahl von Ereignissen eine Änderung auftritt.
- Zustände trennen, wenn sich die durchschnittliche Leistungsaufnahme in einem Zustand bei verschiedenen Durchläufen unterscheidet. Hier wird dann die Vorgehensweise angewendet, die die automatische Synthese bei dem *Backtracking* anwendet.
- Verzögerte Zustandsübergänge einführen, wenn in einem Zustand nach einem spezifischen Ereignis nach genau definierter Zeit eine Änderung der durchschnittlichen Leistungsaufnahme stattfindet.

Das Trennen, Zusammenführen und Einfügen von Zuständen sollte nach den gültigen Gesetzen der Graphentheorie [11] erfolgen, damit der dadurch entstehende Graph weiterhin ein gültige Darstellung des vorherigen Graphen ist.

Wurden Änderungen durchgeführt, kann die geänderte PSM wie oben beschrieben erneut getestet werden und danach weiter optimiert werden. Dadurch ist eine inkrementelle Verbesserung des Modells bis zur gewünschten Genauigkeit möglich.

### 5.4.2. Datenabhängige Leistungsaufnahme

Die bisher in diesem Kapitel vorgestellte Methode zur Charakterisierung der Leistungsaufnahme betrachtet bisher noch keine Datenabhängigkeiten in der Leistungsaufnahme. Diese können aber einen signifikanten Teil der Leistungsaufnahme ausmachen. Ist die Abweichung für das PSM-Modell ohne Betrachtung der Datenabhängigkeiten ausreichend klein – er entspricht also den Benutzeranforderungen – kann dieser Schritt ausgelassen werden.

In diesem Abschnitt wird nun das Vorgehen beschrieben, um diese Datenabhängigkeiten zu identifizieren, zu charakterisieren und in das bestehende Modell zu integrieren. Wie in Abschnitt 4.8 beschrieben, gibt es verschiedene Möglichkeiten, die datenabhängige Leistungsaufnahme zu modellieren. Diese sind die **durchschnittliche** datenabhängige Leistungsaufnahme (Abschnitt 4.8.1), die **bereichsweise** datenabhängige Leistungsaufnahme (Abschnitt 4.8.2) und die **vollständige** datenabhängige Leistungsaufnahme (Abschnitt 4.8.3).

Im nächsten Abschnitt wird kurz darauf eingegangen, wie die durchschnittliche datenabhängige Leistungsaufnahme charakterisiert werden kann. Im nachfolgenden Abschnitt wird dann der Prozess erläutert, wie sowohl die bereichsweise als auch die vollständige datenabhängige Leistungsaufnahme charakterisiert werden.

#### Durchschnittliche datenabhängige Leistungsaufnahme

Wie bereits in Abschnitt 4.8.1 beschrieben, wird bei dieser Art der Modellierung die datenabhängige Leistungsaufnahme nicht explizit modelliert, sondern implizit bei der Charakterisierung der kontrollflussabhängigen Leistungsaufnahme mit betrachtet. Dies geschieht dadurch, dass bei der Stimuli-Auswahl (siehe Abschnitt 5.2.2) die Werte gezielt so ausgewählt werden, dass diese den späteren Anwendungsszenarien entsprechen und gerade die Datenwerte nicht konstant sind. Dies beinhaltet sowohl die Kontrollflusswerte als auch die Datenwerte. Dadurch wird erreicht, dass die resultierenden Leistungsaufnahmewerte die datenabhängigen Leistungsaufnahmewerte mit beinhalten.

Durch die datenabhängige Leistungsaufnahme kann es passieren, dass es bei der automatischen Synthese aus Abschnitt 5.4.1 dazu kommt, dass unterschiedliche Datenänderungen zu unterschiedlichen Verhaltensweisen der Leistungsaufnahme für den gleichen Zustand

führen und dieser dadurch möglicherweise in mehrere Zustände synthetisiert wird. Dies kann bei der späteren Simulation zu großen Abweichungen oder auch in eine nicht vollständig simulierbare PSM (siehe Abschnitt 5.4.4) führen. Daher sollten die Stimuli-Werte so ausgewählt werden, dass die Datenänderungen möglichst immer eine gleichbleibende Änderungscharakteristik aufweisen. Ist dies nicht möglich, sollte eine der anderen Modellierungsmöglichkeiten wie die bereichsweise oder vollständige Modellierung für die datenabhängige Leistungsaufnahme genutzt werden.

Der Vorteil dieser Modellierung ist, dass diese Vorgehensweise Zeit spart, da eine zusätzliche Charakterisierung der Datenabhängigkeiten nicht nötig und die Simulationsperformanz höher ist, da die Datenabhängigkeiten zur Simulationszeit nicht berechnet werden müssen.

Des Weiteren müssen die Schwellwerte für die automatische Synthese erhöht werden, da die Leistungsaufnahme durch die Datenabhängigkeiten mehr fluktuiert. Dies kann dazu führen, dass Zustände zusammengefasst werden, die eine unterschiedliche Leistungsaufnahme aufweisen. Dadurch kann sich die Genauigkeit des resultierenden Modells vermindern.

#### Bereichsweise und vollständige datenabhängige Leistungsaufnahme

Wird ein genaueres Modell benötigt, können die Datenabhängigkeiten separat modelliert werden. Die Charakterisierung der bereichsweisen und vollständigen datenabhängigen Leistungsaufnahme greift die Eigenschaften der datenabhängigen Leistungsaufnahmemodellierung aus Abschnitt 4.8 auf. Die bereichsweise dynamische Leistungsaufnahme aus (4.35) von Seite 66 ist wie folgt definiert:

$$\gamma_{data} = c + \sum_{i=1}^{n} \frac{1}{m} \sum_{j=1}^{m} x_i(d_i(y_{ij}, y_{i(j+1)})) \quad \text{mit} \quad d_i(y_{ij}, y_{i(j+1)}) \in Q_i$$

und die vollständige dynamische Leistungsaufnahme aus (4.37) von Seite 69:

$$\gamma_{data}(Z) = c + \sum_{i=1}^{n} \sum_{j=1}^{m_i - 1} x_{ij}(d_{ij}(z_{ij}, z_{i(j+1)})) \quad .$$

Im ersten Schritt müssen alle Zustände identifiziert werden, die eine datenabhängige Leistungsaufnahme aufweisen. Für jeden Zustand müssen dann folgende Parameter gefunden werden:

1. relevante Ein- und Ausgangsports,
2. Auswahl der Differenzmetrik,
3. Abhängigkeit zwischen Differenzmetrik und Leistungsaufnahme und
4. Historienlänge.

**Zustände mit datenabhängiger Leistungsaufnahme identifizieren** Damit Datenabhängigkeiten in der Leistungsaufnahme charakterisiert werden können, ist es zu Beginn nötig, alle Zustände zu identifizieren, die eine datenabhängige Leistungsaufnahme aufweisen. Dabei ist zu beachten, dass unter Umständen auch Zustände Datenabhängigkeiten aufweisen, die keine Verarbeitung der Daten durchführen. Der Grund dafür ist in den meisten Fällen, dass Änderungen auf den Datenports auch Schaltaktivität in Logikblöcken der Komponente auslösen, die aber nicht gespeichert oder weitergeleitet werden. Daher müssen alle Zustände, bei denen Änderungen auf den entsprechenden Datenports stattfinden können, auf eine entsprechende datenabhängige Leistungsaufnahme getestet werden. Kann durch die Komponentenumgebung garantiert werden, dass in bestimmten Zuständen keine Änderungen auf den Datenports stattfinden, müssen diese nicht überprüft werden. Diese Garantie kann z.B. durch einen Bustransaktor hergestellt werden, der Signaländerungen in bestimmten Systemphasen (z.B. dem `Reset`-Zustand) nicht an die Komponentenschnittstelle weiterleitet, oder durch Kontrakte [50]. Die Auswahl der zu betrachtenden Zustände erfolgt durch den Benutzer.

Für alle Zustände, die eine potenzielle datenabhängige Leistungsaufnahme haben können, werden entsprechend der in Abschnitt 5.2.2 beschriebenen Methode Stimuliwerte generiert. Dabei ist es wichtig, dass möglichst verschiedene Arten von Schaltaktivitäten in Bezug auf die betrachteten Metriken betrachtet werden. Das heißt z.B. für die normalisierte HD, dass die Werte das gesamte Spektrum von 0 bis 1 abdecken. Werden neben den aktuellen Werten auch vergangene Werte betrachtet, sollten möglichst viele unterschiedliche Kombinationen erzeugt werden. Werden mehrere Datenports betrachtet, muss auch hier versucht werden, möglichst viele unterschiedliche Kombinationen zu betrachten. Nur so kann eine gute Abdeckung aller Möglichkeiten erreicht und ein genaues Modell generiert werden. Die Anzahl an möglichen Kombination steigt extrem mit größeren Bitbreiten, längerer Historie und mehreren Eingängen. Eine Simulation aller Möglichkeiten ist dann nicht mehr effizient möglich. Hier wird die Anzahl der betrachteten Kombinationen so weit wie nötig reduziert. In diesem Fall ist es wichtig, besonders die Randfälle zu analysieren und einige dazwischenliegende Kombinationen, um eine gute Interpolation der fehlenden Kombinationen zu erreichen. Randfälle für die normalisierte HD sind z.B. 0 und 1.

Hat man beispielsweise eine Komponente mit zwei 4 Bit breiten Dateneingängen, ergeben sich pro Eingang fünf mögliche Werte $(0,\ldots,4)$ für die HD, also $5 \cdot 5 = 25$ Kombination. Wird eine Historie von zwei Wertänderungen betrachtet ergeben sich damit $25^2 = 625$ Kombination, die betrachtet werden müssen. Für das gleiche Beispiel mit 32 Bit breiten Dateneingängen wächst die Anzahl an Kombination bereits auf $(33 \cdot 33)^2 = 1\,048\,576$.

**Detektion der relevanten Ein- und Ausgangsports** Nachdem die Zustände ausgewählt wurden, müssen auch die entsprechenden Ports ausgewählt werden. Für diesen Fall müssen die Datenports, wie sie in Abschnitt 5.1.1 definiert wurden, ausgewählt werden. Die Auswahl sollte, wie oben definiert, so getroffen werden, dass nur die Ports betrachtet werden, auf denen Änderungen in den entsprechenden Zuständen erwartet werden. Diese Auswahl

muss für jeden datenabhängigen Zustand durchgeführt werden, denn in unterschiedlichen Zuständen können andere Ports relevant sein.

Ob ein Port betrachtet werden muss, kann durch eine Betrachtung der Leistungsaufnahme bei konstanten als auch bei ändernden Werten herausgefunden werden. Unterscheidet sich die Leistungsaufnahme in beiden Fällen signifikant, ist der Port relevant für eine Betrachtung, im anderen Fall nicht. Wichtig dabei ist, dass alle anderen Ports konstante Werte bekommen. Haben diese beispielsweise auch einen Einfluss auf die datenabhängige Leistungsaufnahme, wird die Analyse verfälscht.

Die Auswahl muss sich nicht auf einen Port beschränken. Beispielsweise können sich bei einigen Komponenten auf mehreren Ports gleichzeitig die Datenwerte ändern. Wenn beide Ports einen Einfluss auf die Leistungsaufnahme besitzen, müssen beide mit in die Betrachtung aufgenommen werden.

**Auswahl der Differenzmetrik** Wie in Abschnitt 4.8 vorgestellt, ist die am häufigsten benutzte Differenzmetrik die sogenannte HD. Diese zeigt häufig die besten Ergebnisse, weil sie die Anzahl der schaltenden Symbole angibt und dadurch die Anzahl der Signale die eine dynamische Leistungsaufnahme hervorrufen. Eine weitere Differenzmetrik, die ausgewählt oder zur HD dazu genommen werden kann, ist die Signaldistanz. Diese gibt an, wie viele Symbole konstant bleiben. Welche Differenzmetrik sinnvoll ist, hängt stark von der funktionalen Komponente ab. Hat der Benutzer Wissen über die interne Verarbeitung, kann er dieses in Form der Differenzmetrik in das Modell übertragen und erhält damit eine genauere Modellierung. Diese Metrik kann demnach auch eine selbst definierte Funktion sein.

**Abhängigkeit zwischen Differenzmetrik und Leistungsaufnahme** Nachdem die Differenzmetrik feststeht, kann die Abhängigkeit zwischen den Ergebnissen dieser und der dynamischen Leistungsaufnahme festgelegt werden. Häufig ist eine lineare Abhängigkeit festzustellen. Aber auch quadratische oder exponentielle Abhängigkeiten sind möglich. Wird bei der Differenzmetrik eine eigene Funktion erstellt, wird die Abhängigkeit in der Regel direkt mit abgebildet. Auch hier hängt die Auswahl wieder stark von der funktionalen Komponente ab. Wird eine lineare Abhängigkeit gewählt, müssen die Linearparameter identifiziert werden. Diese lassen sich über eine lineare Regression berechnen. Wichtig dabei ist, den Konstantwert für die kontrollflussabhängige Leistungsaufnahme abzuziehen oder direkt mit zu berechnen, indem ein Konstantwert mit in der linearen Regression eingeführt wird.

**Auswahl der Historienlänge** Wird für die datenabhängige Leistungsaufnahme die vollständige Modellierung gewählt, kann entschieden werden, ob nur die letzte Änderung auf den Datenports betrachtet wird oder eine längere Historie. Die Historienlänge richtet sich danach, wie genau das resultierende Modell sein soll. Dabei ist festzuhalten, dass

eine längere Historie nicht zwingend zu einem besseren Modell führt. In der Regel richtet sich die Historienlänge danach, wie lange die Eingangsdaten Einfluss auf die Leistungsaufnahme haben. Dies hängt davon ab, in wie vielen Schritten die Eingangsdaten verwendet werden.

Werden Daten in unveränderter Form in immer der gleichen Anzahl von Schritten verwendet, macht eine längere Betrachtung Sinn. Ist die Anzahl von Schritten dynamisch oder werden die Daten innerhalb der funktionalen Komponente verändert, ist häufig nur die Betrachtung der letzten Änderung sinnvoll. Gibt es starke Veränderungen der Daten innerhalb der funktionalen Komponente wie beispielsweise in Verschlüsselungskomponenten, gehen die Datenabhängigkeiten in der Leistungsaufnahme komplett verloren, wodurch die durchschnittliche Betrachtung der dynamischen Leistungsaufnahme die besten Ergebnisse liefert.

#### Automatisierte Variante

In dem vorherigen Abschnitt wurde erläutert, wie ein manuelles Vorgehen aussieht, um eine größtmögliche Genauigkeit für die datenabhängige Leistungsaufnahme zu erzielen. Da, wie in Abschnitt 2.2.2 beschrieben, die dynamische Leistungsaufnahme durch schaltende Signalleitungen und CMOS-Schaltungen herbeigeführt wird, wird durch die Auswahl der HD als Differenzmetrik in einem Großteil der Schaltungen das bestmögliche Ergebnis erzielt, da dies genau die Anzahl der schaltenden Signalleitungen wiedergibt. Außerdem ist die Abhängigkeit zwischen HD und Leistungsaufnahme oft linear. Aus diesem Grund werden in dem automatisierten Ansatz voreingestellt lediglich die HD als Differenzmetrik und eine lineare Abhängigkeit betrachtet. Weitere Differenzmetriken können nach Bedarf hinzugefügt werden (z.B. Signaldistanz).

Wie im vorher beschriebenen manuellen Ansatz müssen sowohl die Zustände als auch die Datenports ausgewählt werden, die von dem automatisierten Ansatz betrachtet werden. Standardmäßig wird eine Historienlänge von drei Änderungen (also der letzten vier Werte auf den Ports) betrachtet. Hat der Benutzer weitere Informationen über die Verarbeitungslänge in der Komponente, kann er die betrachtete Historienlänge entsprechend anpassen.

Damit ergibt sich für jeden Leistungsannahmewert eine mögliche Kombination aus Modellvariablen, die sich aus den HDs für die verschiedenen Datenports und Historienwerte ergeben. Die automatisierte Variante versucht nun, die relevanten Modellvariablen zu identifizieren und die fehlenden Linearparameter zu finden. Dies macht sie über den sogenannten *lasso* (least absolute shrinkage and selection operator) [71]. Dabei wird folgendes Vorgehen benutzt: Es wird begonnen mit einer Formel, die nur einen Konstantwert benutzt. Dies entspricht dem Mittelwert aller Leistungsaufnahmewerte und dementsprechend dem Modell ohne Modellierung der dynamischen Leistungsaufnahme mit

$$P_{dyn} = c \quad . \tag{5.7}$$

Nun beginnt die Vorwärtssuche, bei der alle Modellvariablen nach dem Zufallsprinzip einzeln der Formel hinzugefügt und dann eine lineare Regression durchgeführt wird. Danach werden die Ursprungsformel und die neue Formel nach der Standardabweichung miteinander verglichen. Ist die Abweichung der neuen Formel niedriger und liegt die Verbesserung über einem vorher definierten Schwellwert, wird die neue Formel als Referenz übernommen und die Vorwärtssuche für die übrigen Modellvariablen ausgeführt. Dieser Schritt wird so lange wiederholt, bis keine Modellvariable mehr hinzugefügt wird, da der Schwellwert nicht überschritten wird.

Ist die Vorwärtssuche beendet, beginnt die Rückwärtssuche. Hier werden jeweils alle Modellvariablen der Formel einzeln entfernt, solange sich die Standardabweichung verbessert. Das Vorgehen ist ansonsten äquivalent zur Vorwärtssuche, nur dass hier kontinuierlich Modellvariablen entfernt anstatt hinzugefügt werden. Dieser Schritt wird so lange wiederholt, bis keine Modellvariable mehr entfernt wurde.

Vorwärtssuche und Rückwärtssuche werden so lange wiederholt, bis keine Verbesserung mehr möglich ist. Ein kontinuierliches Erhöhen des Schwellwerts verhindert eine Endlosschleife zwischen beiden Suchen. Mit diesem Vorgehen werden alle relevanten Modellvariablen identifiziert und gleichzeitig die Linearparameter bestimmt.

### 5.4.3. Simulation der PSMs

Nachdem eine oder mehrere PSMs nach dem im vorherigen Abschnitt beschriebenen Vorgehen erstellt wurden, können diese bewertet werden. Als Referenz dienen die auf GL erstellten Leistungsaufnahme-Traces. Daher müssen die erstellten PSMs mit den identischen Stimuli der GL-Simulation simuliert werden, um Leistungsaufnahme-Traces zu erzeugen, die mit den GL-Leistungsaufnahme-Traces verglichen werden können. Je nach Modellierungsansatz auf ESL wird ein wie in Abschnitt 5.1 beschriebener Transaktor verwendet, um die GL-Stimuliwerte in der PrSM nutzen zu können. Dadurch ergibt sich an der PSM ein Ausgabestrom von mittleren geschalteten Kapazitäten, die mit den Spannungs- und Frequenzwerten der GL-Simulation verrechnet werden und dadurch ein Leistungsaufnahme-Trace auf ESL entsteht. Ein Vergleich mit dem entsprechenden GL-Leistungsaufnahme-Trace gibt Rückschlüsse über die Genauigkeit des Modells. Zusätzlich lässt sich die Ausführungszeit der Simulationen messen und damit der zeitliche Simulationsmehraufwand der PSM-Modelle feststellen. Dadurch können die PSM-Modelle bewertet und mögliches Fehlverhalten identifiziert werden. Dies wird im nachfolgenden Abschnitt beschrieben.

### 5.4.4. Bewertung der PSMs

In Abschnitt 5.4.1 wurde vorgestellt, wie eine PSM manuell bzw. automatisiert erstellt werden kann. In den jeweiligen Abschnitten wurde bereits vorgestellt, welche Probleme dabei auftreten können und wie diese erkannt bzw. gelöst werden können. Da aber nicht

sichergestellt werden kann, dass alle Probleme ausreichend gelöst wurden, werden in diesem Abschnitt Methoden vorgestellt werden, die diese Probleme erkennen. Wie im nächsten Abschnitt erklärt wird, sollte eine PSM erst dann eingesetzt werden, wenn alle genannten Probleme gelöst wurden.

Simulierbare PSMs können nach ihrer Erstellung in Bezug auf ihre Genauigkeit in mehreren Aspekten bewertet werden oder mehrere PSMs können miteinander verglichen werden. Letzteres trifft dann zu, wenn durch die automatisierte Synthese mehrere PSMs erzeugt wurden bzw. mehrere von Hand generierte PSMs zur Verfügung stehen. Da eine nachträgliche Optimierung von automatisch generierten PSMs möglich ist, bietet die Genauigkeitsanalyse ein Maß, mit dem sich die Optimierung bewerten lässt oder ob diese sogar einen negativen Einfluss auf die Genauigkeit hat.

#### Prüfung der Vollständigkeit

In Abschnitt 5.4.1 wurde beschrieben, welche Möglichkeiten es zur automatischen Synthese von PSMs gibt und welche Vor- und Nachteile diese haben. Einer dieser Nachteile ist nicht charakterisiertes Verhalten, wenn die zur Charakterisierung genutzten Traces dieses Verhalten nicht abgedeckt haben. Aus diesem Grund soll in diesem Abschnitt eine Metrik vorgestellt werden, die automatisch synthetisierte PSMs bewertet und nicht einsetzbare PSMs markiert.

Wird eine PSM generiert, kann es vorkommen, dass nicht an jedem PSM-Zustand für jedes PSM-Event des Eingangsalphabets ein ausgehender Zustandsübergang existiert. In diesem Fall ist die PSM nicht vollständig. Vollständigkeit ist eine notwendige Voraussetzung, damit ein Zustandsautomat für alle verfügbaren Eingaben simuliert werden kann.

**Vollständige PSM** Eine PSM wird genau dann als vollständig bezeichnet, wenn es für jeden Zustand zu jedem beliebigen PSM-Event aus dem Eingangsalphabet einen entsprechenden Zustandsübergang gibt. Ist die Prüfbedingung des Zustandsübergangs erweitert auf Variablenbelegungen, muss für jede mögliche Kombination ein entsprechender Zustandsübergang vorhanden sein. Bezeichnet $e \in E$ ein PSM-Event, $v \in V$ eine Variablenbelegung für alle verfügbaren Zustandsvariablen und $s_{ps} \in S_{ps}$ einen PSM-Zustand, so gilt für alle Kombination aus $e, v$ und $s$, dass ein Zustandsübergang $t_{ps}$ zu einem Folgezustand $s'_{ps} \in S_{ps}$ existiert:

$$\forall e \in E, v \in V, s_{ps} \in S_{ps} : \exists t_{ps} = (s_{ps}, e, v, s'_{ps}) \quad \text{mit} \quad s'_{ps} \in S_{ps} \quad .$$

Für alle fehlenden Zustandsübergänge muss geprüft werden, ob es eine Zustandsabfolge in der PrSM gibt, die den fehlenden PSM-Zustandsübergang auslösen kann. Ist dies der Fall, ist die PSM unvollständig und eine valide Ausführung der PSM nicht möglich. Daher muss der Umfang der Stimuli bei der Synthese so erweitert werden, dass dieser Zustandsübergang abgedeckt ist und mit synthetisiert werden kann. Gibt es keine entsprechende

Zustandsabfolge in der PrSM, kann dieser Zustandsübergang ignoriert werden. In diesem Fall ist die PSM nicht vollständig, aber vollständig simulierbar.

**Vollständig simulierbare PSM** Eine PSM wird genau dann als vollständig simulierbar bezeichnet, wenn es für alle Ausführungsmöglichkeiten der PrSM zu jedem ausgegebenen PSM-Event genau einen PSM-Zustandsübergang gibt, der dieses Event verarbeitet. Diese Bedingung ist dann erfüllt, sobald die Ausgabesprache der PrSM eine Teilmenge der PSM-Eingabesprache ist:

$$L_{out}(PrSM) \subseteq L_{in}(PSM) \quad .$$

Als Ausgabesprache der PrSM wird die Sprache bezeichnet, die genau die Wörter enthält, die bei allen mögliche Ausführungen der PrSM erzeugt werden können.

Da sich die PrSM ändern kann, wenn z.B. andere Abstraktionsebenen verwendet werden, sollten fehlende PSM-Zustandsübergänge trotzdem erstellt werden und zu einem Fehlerzustand führen. Damit wird versehentliches Fehlverhalten vermieden, da die PSM somit bei nicht charakterisiertem Verhalten in den Fehlerzustand wechselt und bis zum Ende der Simulation bleibt. Dadurch kann der Benutzer des Modells den Fehler erkennen oder eine Simulation abbrechen, sobald der Fehlerzustand betreten wird.

#### Bewertung der Genauigkeit

Je nach Parametrisierung der automatischen bzw. manuellen PSM-Synthese können unterschiedliche PSMs generiert werden. Diese können in Bezug auf ihre Genauigkeit miteinander verglichen werden, wobei nur die nach der obigen Definition vollständig simulierbaren PSMs betrachtet werden. Da für eine Genauigkeitsbewertung nur die Leistungsaufnahme-Traces von GL verwendet werden können, muss ein Vergleich auf Trace-Ebene geschehen. Aus diesem Grund wird ein Vergleich auf Basis von mehreren Anwendungsfällen durchgeführt, für die auf GL und mit der PSM auf ESL die entsprechenden Leistungsaufnahme-Traces generiert werden. Hier kann nun ein Vergleich in Bezug auf die Leistungsaufnahme und die aufgenommene Energie erfolgen.

Für die Bewertung der Genauigkeit gibt es verschieden Auswahlmöglichkeiten. Diese unterscheiden sich durch den Bereich, der für den Vergleich herangezogen wird. Dabei können verschiedene Problemquellen identifiziert und optimiert werden.

1. **Vergleiche alle zur Verfügung stehenden Traces:** hier kann eine Aussage über die durchschnittliche Abweichung im allgemeinen gemacht werden. Dies ist sinnvoll, um mehrere PSMs miteinander zu vergleichen.

2. **Vergleiche einzelne Traces miteinander:** ermöglicht eine Bewertung der PSM in Bezug auf einzelne Traces und somit Anwendungsfälle. Hier können mögliche Szenarien identifiziert werden, für die die PSM kein gutes Ergebnis liefert.

3. **Vergleiche alle Segmente, die zu einem PSM-Zustand gehören:** identifiziert Zustände mit einer hohen Abweichung. Dies kann Aufschluss über eine mögliche Optimierung geben, indem der Zustand z.B. in zwei verschiedene Zustände aufgeteilt wird oder Datenabhängigkeiten modelliert werden.
4. **Vergleiche alle einzelnen Segmente:** identifiziert einzelne Segmente mit hoher Abweichung. Eine hohe Abweichung in nur einem Segment eines Zustands kann Indiz für eine fehlende Datenabhängigkeiten in der Leistungsaufnahmemodellierung sein oder für einen nicht modellierten separaten funktionalen Zustand, der nur unter bestimmten Bedingungen eintritt.

Für eine Bewertung der Abweichung zwischen zwei Traces bzw. Trace-Ausschnitten bieten sich verschiedene Maße an. Diese zeigen unterschiedliche Eigenschaften in Bezug auf die Abweichung. In den nächsten Abschnitten wird folgende Funktion zur Mittelwertberechnung auf einem Vektor $X$ mit den Werten $x_0, \ldots, x_n$ eingesetzt:

$$\text{mean}(X) = \frac{1}{n} \sum_{i=0}^{n} x_i \quad . \tag{5.8}$$

**Durchschnittliche Abweichung** Die durchschnittliche Abweichung $E_d$ gibt den Mittelwert für die relative Abweichung zwischen GL-Trace und ESL-Trace an, bei der sich positive und negative gegenseitig aufheben:

$$E_d = \text{mean}(X - Y) \quad , \tag{5.9}$$

wobei $X$ die GL-Tracewerte und $Y$ die ESL-Tracewerte angibt. Die durchschnittliche Abweichung ist der ausschlaggebendste Wert für einen Vergleich, denn trotz hoher Fluktuation und verschiedenster Verhaltensmuster des Leistungsaufnahme-Traces auf GL kann der Mittelwert sehr konstant sein, wodurch auch die durchschnittliche Abweichung sehr gering ausfällt. Dadurch ergibt sich eine geringere Abweichung in der modellierten aufgenommenen Energie. Die durchschnittliche Leistungsaufnahme und Energieaufnahme sind die wichtigsten Werte auf ESL. Die Gründe sind, dass hierdurch eine grobe Abschätzung der Leistungsaufnahme des Gesamtsystems ermöglicht wird, um ein Leistungsaufnahmebudget und Energieaufnahmebudget abzuschätzen und der Vergleich in Bezug auf Leistungsaufnahme und Energieaufnahme von verschiedenen Systemkonfigurationen möglich ist.

**Durchschnittliche absolute Abweichung** Dieser Wert gibt die durchschnittliche absolute Abweichung an, das heißt negative und positive Werte heben sich nicht gegenseitig auf, sondern werden aufaddiert:

$$E_a = \text{mean}(|X - Y|) \quad , \tag{5.10}$$

wobei die Bezeichnungen analog zu (5.9) sind. Hier werden die absoluten Abweichungen addiert und normiert. Damit ergibt sich im Unterschied zur durchschnittlichen Abweichung ein Wert, der angibt, um wie viel der GL-Trace vom ESL-Trace abweicht. Gibt es sehr große

Abweichungen, muss geschaut werden, ob hier eine Optimierung möglich ist. In der Regel ist die durchschnittliche absolute Abweichung hoch, wenn der GL-Trace eine hohe Varianz aufweist. Diese könnte durch Datenabhängigkeiten hervorgerufen werden. In diesem Fall kann die Datenabhängigkeit modelliert und in das Modell integriert werden. Es kann aber auch sein, dass diese hohe Abweichung bedingt ist durch die Verarbeitung in der Komponente und keine Datenabhängigkeiten aufweist. In diesem Fall sollte die (normale) durchschnittliche Abweichung einen geringen Wert aufweisen und eine Optimierung ist nicht nötig.

**Standardabweichungen** Mit diesem Wert werden größere Abweichungen höher gewichtet als bei der durchschnittlicher Abweichung und ist im Englischen als sogenannter *Root Mean Square Error* (RMSE) bekannt. Die Standardabweichung kann für die beiden Vergleichsvektoren $X = x_0, \ldots, x_n$ und $Y = y_0, \ldots, y_n$ wie folgt berechnet werden:

$$E_r = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2} \quad . \tag{5.11}$$

Das bedeutet, gibt es mehr kleine Abweichungen, ist die Standardabweichung niedriger, im anderen Fall höher. Mit Hilfe dieser Größe lässt sich feststellen, ob verschiedene Modelle eher höhere oder niedrigere Abweichungen aufweisen, wenn der Durchschnittswert gleich ist. Die Wurzel sorgt dafür, dass das Ergebnis wieder in Relation mit den Abweichungen gesetzt werden kann. Gibt es eine hohe Abweichung bei der Standardabweichung, kann dies darauf hindeuten, dass große Abweichungen durch separate Zustände oder eine Modellierung der Datenabhängigkeit nachgebildet werden können, um diese Abweichung zu senken.

**Maximale Abweichung** Dieser Wert gibt die maximale Abweichung zwischen dem GL- und ESL-Trace an:

$$E_m = \max(|X - Y|) \quad . \tag{5.12}$$

Die Bezeichnungen sind wieder analog zu (5.9). Ist die maximale Abweichung sehr hoch, kann an dieser Stelle eventuell eine Optimierung erfolgen. Dies kann auch wieder auf eine Datenabhängigkeit in der Leistungsaufnahme hinweisen oder auf die Verarbeitung der Daten innerhalb der Komponente. Ein Beispiel für einen sehr hohen Ausschlag ist die *Reset*-Phase. Hier werden alle Signale zurückgesetzt was für einen kurzen Moment zu einer sehr hohen Schaltaktivität führt und dadurch eine hohe Leistungsaufnahme herbeiführt. Da diese Spitze nur für einen sehr kurzen Augenblick stattfindet – in den meisten Fällen nur ein oder zwei Takte – die *Reset*-Phase aber in der Regel über einen längeren Zeitraum aktiv ist, bleibt die durchschnittlichen Abweichung niedrig. Möchte man dieses Verhalten mit modellieren, bietet sich hier der zeitgesteuerte Zustandsübergang an. Mit diesem ist es möglich, für den festgelegten Zeitraum die Spitzenleistung zu modellieren und anschließend in einen PSM-Zustand zu wechseln, der die normale Leistungsaufnahme modelliert.

**Relative Abweichungen** Da es häufig nötig ist, die Abweichung von einem Modell auch für verschiedene Komponenten zu vergleichen, sind die oben vorgestellten Werte nicht ausreichend. Unterschiedliche Komponenten haben in der Regel auch deutlich unterschiedliche Leistungsaufnahmen in Höhe von einigen Größenordnungen. In ähnlichem Maß können sich dabei auch die Energieaufnahme und die daraus resultierenden Abweichungen verändern. Um dem entgegenzuwirken, kann die Abweichung in Relation zu der durchschnittlichen Leistungsaufnahme des GL-Traces gesetzt werden. Dies ist im Folgenden am Beispiel der durchschnittlichen Abweichung aus (5.9) mit äquivalenten Bezeichnern gezeigt, $rv$ zeigt hier den Bezug als Relativwert (*Relative Value*):

$$E_d^{rv} = \frac{\text{mean}(X - Y)}{\text{mean}(X)} \quad . \tag{5.13}$$

Häufig ist auch der Wertebereich ein besseres relatives Maß als der durchschnittliche Wert des GL-Traces, da durch eine größere Varianz auch eine größere Abweichung verursacht wird. Die Berechnung ist im folgenden wieder am Beispiel der durchschnittlichen Abweichung aus (5.9) mit äquivalenten Bezeichnern gezeigt, $rm$ als Bezug zum Varianzwert (*Relative Min-Max*):

$$E_d^{rm} = \frac{\text{mean}(X - Y)}{\max(X) - \min(X)} \quad . \tag{5.14}$$

**Verteilung der Abweichungen** Wenn man sich die Verteilung der Abweichungen anschaut, können weitere Rückschlüsse getroffen werden. Die Verteilung wird berechnet, indem das Intervall zwischen maximaler und minimaler Abweichung in eine definierte Anzahl gleichgroßer Unterintervalle unterteilt wird. Es wird die Anzahl der Abweichungen in den einzelnen Intervallen gezählt und als Histogramm dargestellt. Die Abweichungen können absolut oder vorzeichenbehaftet betrachtet werden. Es sollte eine Anhäufung hin zur Abweichung 0 geben, wobei bei 0 die maximale Anzahl vorhanden sein sollte, also in etwa eine Normalverteilung. Entspricht die Verteilung nicht diesem Bild, gibt es in der Regel eine systematische Abweichung im Modell und spiegelt damit einen Fehler im Modell wieder.

Bei der automatisierten Bewertung wird der Bereich in 100 Unterintervalle geteilt, wobei das 90%-Perzentil der Werte im untersten Unterintervall zu finden sein muss.

### 5.4.5. Optimierung der PSM

Nachdem das Modell erstellt und bewertet wurde, kann entschieden werden, ob die verbleibenden Ungenauigkeiten des Modells in einem akzeptablen Bereich liegen oder das PSM-Modell weiter verfeinert werden soll. Dies kann eine Änderung der PrSM sein, wenn das funktionale Verhalten nicht genau genug abgebildet wurde oder eine Änderung in der PSM. Dies kann durch manuelle Anpassungen passieren, indem Zustände oder Zustandsübergänge verändert werden oder durch Adaptierungen bei der datenabhängigen

Leistungsaufnahme und den zeitgesteuerten Zustandsübergängen. Bei der automatischen Synthese können Parameter und Schwellwerte verändert werden, um ein anderes Syntheseergebnis zu erzeugen. Nach diesem Schritt ist die Erstellung des Modell vollständig und hinreichend genau und kann für die Simulation eingesetzt werden. Damit bietet die Arbeit ein vollständiges Vorgehen zur Synthese des PSM-Modells.

## 5.5. Zusammenfassung

In diesem Kapitel wurde das gesamte Vorgehen beschrieben, wie man von einem RTL- bzw. einem ESL-Design zu einem vollständigen PSM-Modell gelangt. Dabei wurde zum einen erklärt, wie sich die PrSM generieren lässt und wie man gezielt Leistungsaufnahme-Traces erzeugt, die als Grundlage für die Modellsynthese dienen und als Referenz für die Bewertung der Genauigkeit in den erstellten Modellen. Des Weiteren wurde erklärt, wie die PSM manuell oder automatisiert erstellt werden kann mit Betrachtung der datenabhängigen Leistungsaufnahme und den zeitgesteuerten Zustandsübergängen. Damit wurde ein automatisiertes Synthesevorgehen gezeigt, welches Anforderung N3 erfüllt. Das Vorgehen erlaubt eine schnelle und einfache Erstellung des Modells und erfüllt damit Anforderung N2.

# KAPITEL 6

# Evaluation

In den Kapiteln 4 und 5 wurde das Konzept und das Vorgehen zur Abstraktion der Leistungsaufnahme vorgestellt, welches die Leistungsaufnahmemodellierung und Simulation auf ESL ermöglicht. Diese erfüllen die in Abschnitt 1.2 aufgestellten Anforderung in Bezug auf die Modellierung und Abstraktion der Leistungsaufnahme.

Wie in der Einleitung dieser Arbeit beschrieben, soll das Modell neben den aufgestellten Anforderungen noch möglichst genau und performant sein, um mit weniger Zeitaufwand bessere Entscheidungen in Bezug auf die Leistungsaufnahme des resultierenden Systems auf ESL treffen zu können.

In diesem Kapitel soll nun evaluiert werden, wie hoch die Genauigkeit des Modells im Vergleich zur Leistungsaufnahmewerten auf GL ist und wie performant das Modell simuliert werden kann. Dafür wird in Abschnitt 6.1 die Testumgebung und das Vorgehen für die Evaluation beschrieben. In Abschnitt 6.2 werden die einzelnen Evaluationskomponenten und deren Ergebnisse in Bezug auf die Genauigkeit vorgestellt. In Abschnitt 6.4 wird die Performanz des PSM-Modells bewertet. Am Ende dieses Kapitels gibt es eine qualitative Bewertung des Syntheseprozesses (Abschnitt 6.5).

## 6.1. Beschreibung der Testumgebung

Damit eine Evaluation der einzelnen Beispiele möglich ist, muss eine Testumgebung geschaffen werden, die Umgebungsbedingungen festlegt und somit eine einheitliche Ausführung und Vergleichbarkeit garantiert. Diese ist in Abbildung 6.1 gezeigt. In der Evaluation soll die Frage beantwortet werden, ob das PSM-Modell genau genug ist, um korrekte Designentscheidungen bezüglich der Leistungsaufnahme zu treffen und ob eine signifikante Performanzsteigerung im Vergleich zu Simulationen auf niedrigeren Abstraktionsebenen erreicht werden kann.

Da die GL-Leistungsaufnahme-Traces verwendet werden, um das PSM-Modell zu erstellen, werden diese Werte auch für die Evaluation als Referenz verwendet. Dafür wird das PSM-Modell aus einem speziellen Trace für die Synthese erstellt. Danach wird das Modell mit diesem und weiteren anderen Anwendungsfällen simuliert. Damit kann sichergestellt werden, dass das Ergebnis nicht auf den Trace für die Synthese optimiert ist und in allen

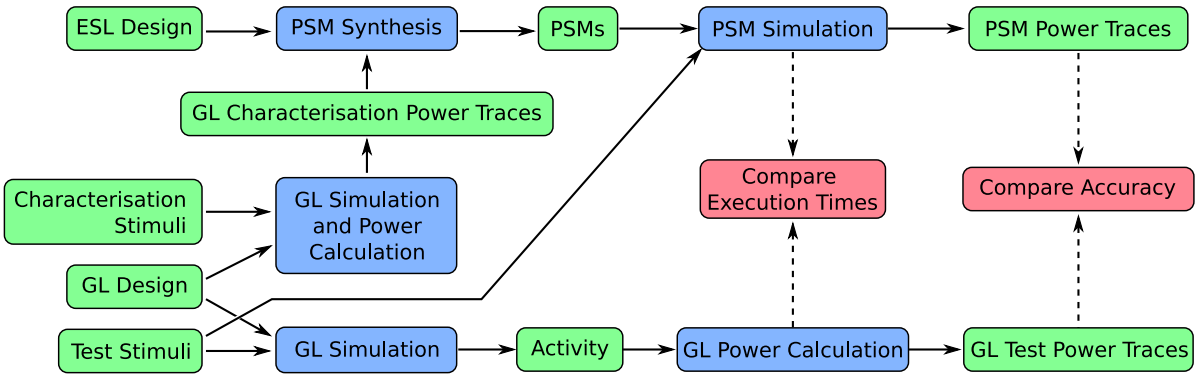



Abbildung 6.1.: Aufbau der Testumgebung, um Genauigkeit und Ausführungszeit des PSM-Modells zu evaluieren.

Anwendungsfällen ähnliche Ergebnisse zeigt. Die resultierenden Leistungsaufnahme-Traces werden mit der auf GL erzeugten Leistungsaufnahme verglichen. Daraus lässt sich die Genauigkeit des Modells ableiten.

Für die Bewertung der Performanz werden die Simulationszeiten auf ESL mit den Zeiten verglichen, die benötigt werden, um auf GL den entsprechenden Leistungsaufnahme-Trace zu erstellen.

Die Evaluation soll die Performanzsteigerung durch des Modells dieser Arbeit zeigen. Dabei müssen die Effekte durch ein abstrakteres funktionales Modell ausgeblendet werden. Um dies zu ermöglichen, werden sowohl auf GL als auch auf ESL Signal-Schnittstellen eingesetzt. Damit wird eine Verbesserung der Simulationsgeschwindigkeit durch abstraktere TLM-Schnittstellen vermieden. Zusätzlich wird bei der Simulation der PSM das funktionale Modell nicht mitsimuliert, sondern die GL-Signale verwendet. Auf GL wird ausschließlich die Berechnung der Leistungsaufnahmewerte betrachtet. Dadurch beeinflusst die Ausführung des funktionalen Modell nicht die untersuchte Performanz.

Zur Simulation der PSM wurde ein Bibliothek entwickelt, die die Erstellung von beliebigen PSM-Modellen erlaubt. Diese wurde in der Programmiersprache C++ geschrieben, um zusammen mit der Bibliothek *SystemC* [9] eingesetzt zu werden. Diese Bibliothek dient zur Simulation der entsprechenden PSM-Modelle und zur Evaluierung der Simulationsperformanz. Die Bibliothek unterstützt alle in Kapitel 4 beschriebenen Fähigkeiten.

Für die in Abschnitt 5.4.1 beschriebene automatische PSM-Synthese wurde ein C++ Programm entwickelt, welches den Algorithmus wie beschrieben umsetzt. Dieser wird genutzt, um für einige Evaluationskomponenten automatisch PSMs zu generieren und diese gegen die händisch generierten zu vergleichen. Generiert der Algorithmus die gleichen Automaten wie die manuell erstellten, werden diese nicht explizit erwähnt. Da die automatische Synthese nicht für die datenabhängige Leistungsaufnahme funktioniert, wird für die entsprechenden Komponenten auch keine automatische Synthese durchgeführt.

Für die Bewertung werden die folgenden drei Fehler aus Abschnitt 5.4.4 betrachtet:

- durchschnittliche Abweichung,
- durchschnittliche absolute Abweichung und
- Standardabweichung.

Die Zustände der PSMs werden, wie in Abschnitt 4.5 beschrieben, mit Leistungsaufnahmewerten $P_{dyn}$ annotiert und nicht mit mittleren geschalteten Kapazitäten $\overline{C}$. Da die Spannung und Frequenz in GL-Simulationen konstant bleibt und es einen linearen Einfluss zwischen $P_{dyn}$ und $\overline{C}$ gibt, hat dies somit keinen Einfluss auf die Ergebnisse.

Alle Simulationen werden auf dem gleichen Rechner ausgeführt, um eine Vergleichbarkeit der Ergebnisse zu garantieren. Des Weiteren wird sichergestellt, dass keine zusätzlichen nicht benötigten Anwendungen ausgeführt werden, die die Ausführungszeit der Simulationen beeinflussen.

### 6.1.1. Erzeugung der GL-Leistungsaufnahme-Traces

Für die Erzeugung der GL-Leistungsaufnahme-Traces wird ein Design auf RTL benötigt, Stimuliwerte für die Simulation aller benötigten Operationszustände und eine Bibliothek der Zieltechnologie. In dieser Arbeit wird als Technologiebibliothek *tcbn65gpluslt* von *TSMC* in der Revision 200 eingesetzt. Dies ist eine 65 nm Technologie. Mit dieser werden alle Designs mit dem *Synopsys Design Compilers Version k-2015.06-SP1* nach Gate Level synthetisiert. Für das Syntheseskript siehe Anhang Abschnitt A.2.

Bei der Synthese wird, wie in Abschnitt 5.2.1 beschrieben, das *Standard Wire Delay Model* eingesetzt und kein *Place-and-Route* durchgeführt, um keine Abhängigkeit zwischen den Referenzwerten und der spezifischen Platzierung zu haben.

Die Stimuliauswahl erfolgt manuell anhand der erstellten PrSM, der bekannten Informationen des funktionalen Verhaltens und dem in Abschnitt 5.2.2 vorgestellten Vorgehen. Anschließend wird jedes Design mit den ausgewählten Stimuli und einer geeigneten Testbench in *Questa Sim -64 Version 10.5b* von *Mentor Graphics* simuliert und die gesamte Aktivität in eine sogenannte *Value Change Dump* (VCD)-Datei geschrieben. Die Aktivität beschreibt jeden Signalwechsel innerhalb der untersuchten Komponente. Die VCD-Datei beinhaltet, welche Signale in welcher Modulhierarchie betrachtet werden und zu welchen Zeitpunkten welcher Signalwechsel stattfindet. Zusätzlich wird eine weitere VCD-Datei erzeugt, die nur die Signaländerungen an den Ein- und Ausgängen enthält. Diese wird für die Erzeugung der PSM-Event-Traces genutzt, wie in Abschnitt 5.3.2 beschrieben, und für die PSM-Modellsimulation, damit identische Eingangsdaten genutzt werden. Die VCD-Datei mit allen Aktivitätswerten wird mit Hilfe der oben genannten Technologiebibliothek in einen Leistungsaufnahme-Trace überführt. Dieser Prozess wird mit dem Werkzeug *PrimeTime PX Version K-2015.06* von *Synopsys* durchgeführt. Dieses erlaubt verschiedene Ausgaben von Leistungsaufnahme-Traces. In dieser Arbeit werden die Traces taktgenau berechnet, da die Leistungsaufnahme mit den Eingaben und Ausgaben korreliert werden. Diese werden bei

einer positiven Taktflanke übernommen. Aus diesem Grund ist eine genauere Betrachtung der Leistungsaufnahme nicht nötig. Diese Traces dienen sowohl als Referenz-Trace für die Synthese als auch als Test-Traces für die Überprüfung der Genauigkeit. Für die Bewertung des Modells werden Traces verwendet, die nicht zur Synthese genutzt wurden. Es werden die Zeiten der GL-Simulation als auch der Berechnung der Leistungsaufnahme-Traces gemessen, um einen Vergleich zur Ausführungszeit zum PSM-Modell zu ermöglichen.

### 6.1.2. Modellerstellung

Aus den Referenz-Traces der Leistungsaufnahme werden mehrere Modelle erstellt. Sowohl manuell erstellte Modelle als auch teilweise automatisch synthetisierte Modelle. Die Erstellung und Optimierung der Modelle wird nach den in Kapitel 5 beschriebenen Methoden durchgeführt und für jede Evaluationskomponente im einzelnen beschrieben. Das heißt, es wird zuerst eine PrSM erstellt und die Stimuliwerte entsprechend ausgewählt. Danach gibt es eine erste Betrachtung der Traces für die Synthese mit einer Ableitung von PSMs unter Betrachtung der funktionalen Eigenschaften der untersuchten Komponente. Dabei wird speziell untersucht, ob eine datenabhängige Leistungsaufnahme oder zeitgesteuerte Zustandsübergänge vorhanden sind.

Die Modelle werden in einer XML-Datei gespeichert, die dem Schema aus Abschnitt A.1 entsprechen. In dieser XML-Datei kann das Modell vollständig beschrieben und parametrisiert werden. Dies ermöglicht eine implementierungsunabhängige Speicherung und Änderung des Modells sowie eine Simulation des Modells in einer beliebigen Simulationsumgebung. Diese muss folgende Voraussetzungen erfüllen:

1. eine Implementierung des PSM-Modells in der Simulationsumgebung,
2. ein XML-Einlesemodul, welches das PSM-Modell dynamisch instanziieren kann und
3. Adapter an den Schnittstellen der Komponente, die ein nichtinvasives Auslesen der Kommunikationsdaten inklusive Zeitstempel der Simulationszeit erlaubt.

### 6.1.3. Modellausführung

Nachdem die verschiedenen Modelle manuell und automatisch erstellt wurden, können diese getestet werden. Dafür werden die funktionalen Modelle auf ESL inklusive dem erstellten PSM-Modell mit den Stimuli aus der Synthese und anderen Stimuli simuliert. Dabei wird sowohl die Leistungsaufnahme über die Zeit aufgezeichnet als auch die Ausführungszeit gemessen, die für die entsprechende Simulation benötigt wird. Die Werte der Leistungsaufnahme und die Ausführungszeiten können dann gegen die Referenz auf GL verglichen werden. Die Stimuliwerte werden der PrSM als mit Zeit annotierter Datenstrom übergeben. Damit verhält sie sich und das restliche PSM-Modell, wie wenn es an ein funktionales Modell annotiert ist, welches diese Daten empfängt. Die resultierenden Leistungsaufnahmewerte der PSM werden als mit Zeit annotierter Datenstrom aufgezeichnet.

## 6.2. Evaluation anhand von Komponentenklassen

Auf einem *System on a Chip* (SoC) kann es viele verschiedene Klassen von Komponenten geben. Komponenten aus einer Klasse weisen häufig Ähnlichkeiten in ihrer Funktionalität und dementsprechend auch in ihrer Leistungsaufnahme auf. Beispielsweise gibt es bei Speicherkomponenten häufig eine hohe Korrelation zwischen schaltenden Bits an den Ein- und Ausgängen sowie der Leistungsaufnahme, aber nur eine geringe oder sogar keine Korrelation zwischen den verschiedenen Operationszuständen. Bei einer Verschlüsselungskomponente hingegen verhält es sich genau umgekehrt, da hier die Daten stark verändert werden und somit die Korrelation zu den Daten durch die Verarbeitung verloren geht. Im Folgenden werden verschiedene Klassen vorgestellt und Beispiele für Komponenten genannt, die nachfolgend repräsentativ für die jeweilige Komponentenklasse evaluiert werden. Dadurch kann garantiert werden, dass das in dieser Arbeit entwickelte Modell und der Syntheseprozess auf eine Vielzahl von Komponenten angewendet werden kann:

- Speicherkomponenten,
- Berechnungskomponenten,
- Verschlüsselungskomponenten und
- Eingabe-/Ausgabekomponenten.

Wie bereits in Kapitel 4 erläutert, eignet sich das Modell nicht für aktive Komponenten, da hier der innere funktionale Zustand nur schwer über das Eingabe-/Ausgabeverhalten prognostiziert werden kann. Aus diesem Grund werden an dieser Stelle keine aktiven Komponenten wie etwa Prozessoren oder DMA Controller betrachtet.

### 6.2.1. Gesamtsystem

Das PSM-Modell wurde im Verbund mit weiteren Leistungsaufnahmemodellen für ein Gesamtsystem eingesetzt und evaluiert [26]. Dies zeigt, dass sich dieses Modell nahtlos integrieren lässt und die Anbindung an weitere Modelle erlaubt, die die Leistungsaufnahme von einzelnen Komponenten entgegennimmt und weiter verarbeiten. Des Weiteren ermöglicht es einen guten Kompromiss zwischen Genauigkeit und Simulationsgeschwindigkeit bei der Simulation eines Gesamtsystems.

### 6.2.2. Speicherkomponenten

Eine Speicherkomponente bietet in der Regel einen recht überschaubaren funktionalen Zustandsraum. Die wichtigsten Operation sind das Schreiben und Lesen von Daten in bestimmten Adressbereichen. Wird nicht gelesen oder geschrieben, ist die Komponente im Ruhemodus. Die naive Annahme ist, dass die Komponente beim Schreiben und Lesen eine

unterschiedliche Leistungsaufnahme hat, die im Durchschnitt aber gleich ist, da sich Datenabhängigkeiten über verschiedene Daten herausmitteln. Um diese Annahme zu überprüfen, wird ein Speicher, der als IP-Komponente vorliegt, diesbezüglich untersucht. Ein Kompo-

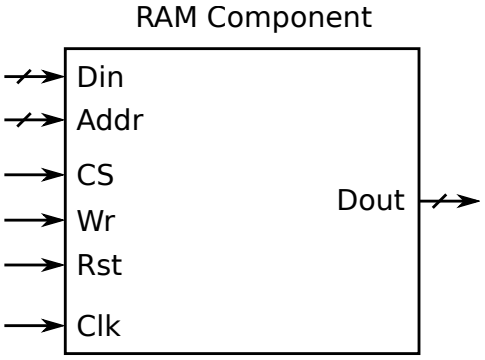



Abbildung 6.2.: Komponentendiagramm des Speichers.

nentenschaltbild ist in Abbildung 6.2 zu sehen. Über den `CS` Eingang wird die Komponente aktiv (Wert = 1) geschaltet. Der Eingang `Wr` gibt an, ob Daten geschrieben (Wert = 1) oder gelesen (Wert = 0) werden. Über den `Reset`-Eingang werden alle Speicherwerte auf den Wert 0 zurückgesetzt. Dadurch ergibt sich die in Abbildung 6.3 gezeigte PrSM.

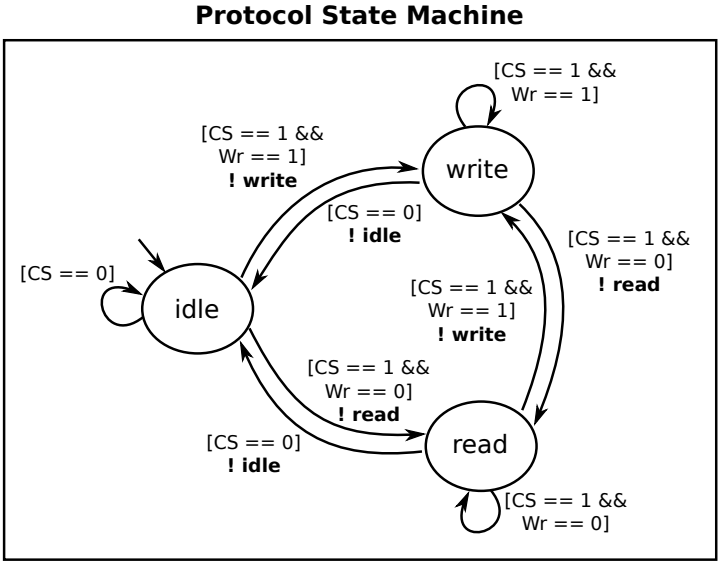



Abbildung 6.3.: PrSM der Speicherkomponente.

Dieser Speicher ist ein sogenannter *Asynchronous single port RAM (Flip-Flop Based)* aus der *Synopsys DesignWare* Bibliothek in Version 38f7ad6e. Der Speicher besitzt eine Datenbreite von 32 Bit und einer Datentiefe von 256. Die Komponente wird, wie in Abschnitt 6.1.1 beschrieben, synthetisiert und simuliert.

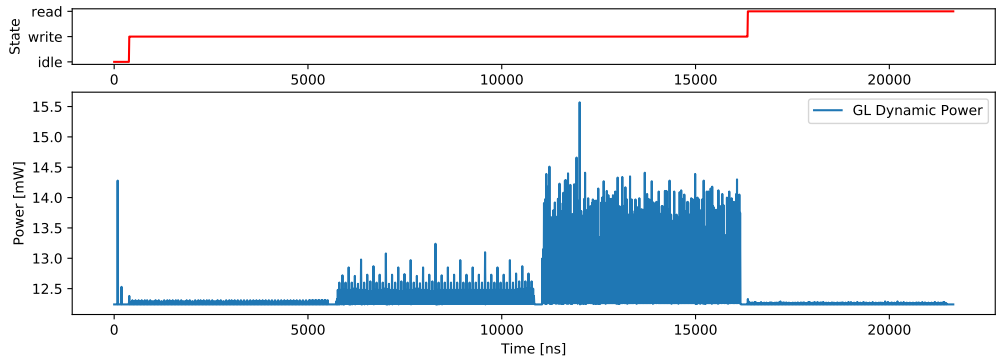



Abbildung 6.4.: Anwendungsfall 1: Leistungsaufnahme-Trace der Speicherkomponente mit allen Modi und verschiedenen Datencharakteristika.

Für die Synthese und die Evaluation werden folgende Anwendungsfälle genutzt.

- Anwendungsfall 1: dieser zeigt gewöhnliche Anwendungsmuster eines Speichers, um die Charakteristik der Leistungsaufnahme zu identifizieren.
- Anwendungsfall 2: dieser wird für die Charakterisierung der Leistungseigenschaften eingesetzt. Er führt alle verfügbaren Zustandsübergänge aus und stellt explizit alle HDs für die Zustände mit datenabhängige Leistungsaufnahme ein.
- Anwendungsfall 3: dieser dient als unabhängige Kontrolle des erstellen Modells, indem er alle Zustandsübergänge ausführt und zufällige Daten nutzt.

In Anwendungsfall 1 werden verschiedene Daten auf den Speicher geschrieben und von ihm gelesen, um zu untersuchen, wie stark sich unterschiedliche Daten auf die Leistungsaufnahme auswirken. Die Adresse wird dabei stetig um 1 inkrementiert, da ein Speicher in der Regel sequentiell beschrieben und gelesen wird. Bei jedem Schreib- oder Lesezugriff werden alle Adressen von Adresse 0x00 bis Adresse 0xFF geschrieben bzw. gelesen. Es werden folgende Datenmuster verwendet, die gewöhnliche Benutzungsmuster von Speichern nachbilden sollen:

- es wird nur der Wert 0 geschrieben: dies entspricht dem Initialisieren einer Datenstruktur,
- es werden inkrementierende Werte geschrieben beginnend bei 0: dies entspricht dem Ablegen einer sortierten Liste,
- es werden zufällige Werte geschrieben: dies entspricht dem gewöhnlichen Schreibverhalten bei Datenstrukturen und
- es werden Daten gelesen.

Der Leistungsaufnahme-Trace für diesen Anwendungsfall wird in Abbildung 6.4 gezeigt. Bei diesem Trace wird deutlich, dass die unterschiedlichen Operationszustände nur einen sehr geringen Einfluss auf die Leistungsaufnahme haben. Die Daten bzw. die Datenänderungen tragen deutlich zu höheren Leistungsaufnahmen bei. Dieses Verhalten ist besonders beim schreibenden Zugriff zu beobachten. Wertänderungen bei der Adresse zeigen eine Erhöhung der Leistungsaufnahme, aber diese ist vergleichsweise gering und unkorreliert zu den Werten oder Wertänderungen auf dem Adresseingang. Analog verhält es sich mit den gelesenen Daten.

Demnach wird für dieses Beispiel die in Abschnitt 4.8 vorgestellte Möglichkeit der datenabhängigen Leistungsaufnahme benutzt, um die Leistungsaufnahme so genau wie möglich modellieren zu können.

Wie in Abschnitt 5.4.2 erläutert, ist die Leistungsaufnahme in vielen Fällen linear abhängig von der HD auf den entsprechenden Ein- und Ausgängen. Aus diesem Grund wird für die Zustände `read` und `write` folgende Formel zur Berechnung der Leistungsaufnahme eingesetzt:

$$P_{dyn} = c + \sum_{i=1}^{n} m_i \cdot HD_i \quad , \tag{6.1}$$

wobei $c$ den Konstantwert beschreibt, $m_i$ den Linearfaktor für die $HD_i$ und $n$ die Anzahl der Ein- und Ausgänge, die als datenabhängige Eingangsgröße charakterisiert werden. Wie oben beschrieben, wird hier anstatt der durchschnittlich geschalteten Kapazität direkt die Leistungsaufnahme berechnet. Dadurch ergeben sich bei der Synthese andere Werte für $c$ und $m$. Das Ergebnis ist jedoch äquivalent, solange sich Versorgungsspannung und Taktfrequenz nicht ändern. Dies wird nachfolgend mit $n = 1$ verdeutlicht:

$$\begin{aligned} P_{dyn} = V_{dd}^2 \cdot f &= c_1 + m_1 \cdot HD \cdot \overline{C} \\ \overline{C} &= \frac{c_1}{V_{dd}^2 \cdot f} + \frac{m_1 \cdot HD}{V_{dd}^2 \cdot f} \\ \overline{C} &= c_2 + m_2 \cdot HD \quad \text{mit} \quad c_2 = \frac{c_1}{V_{dd}^2 \cdot f}, m_2 = \frac{m_1}{V_{dd}^2 \cdot f} \quad . \end{aligned} \tag{6.2}$$

Für die Charakterisierung wird der Anwendungsfall 2 genutzt, der für die Datenabhängigkeiten gezielt Nutzdaten mit unterschiedlichen HDs verwendet, um den linearen Zusammenhang zwischen HD und Leistungsaufnahme so genau wie möglich abzubilden. Für diese wird eine lineare Regression durchgeführt, um genaue Werte für $c$ und $m$ zu identifizieren. Es werden in mehreren Durchgängen die verschiedenen Operationsmodi durchlaufen, um eine große Abdeckung der möglichen Kombinationen zu erreichen. Der entsprechende Leistungsaufnahme-Trace ist in Abbildung 6.5 gezeigt. Auch hier wird wieder deutlich, dass bei einem Zustandswechsel (bei Zeit 5500 ns von `write` auf `idle`) durch viele Schaltvorgänge innerhalb der Komponente große Leistungsspitzen entstehen können. Wie bereits in Abbildung 6.4 zu sehen, ist beim Schreiben eine hohe Variation zu sehen

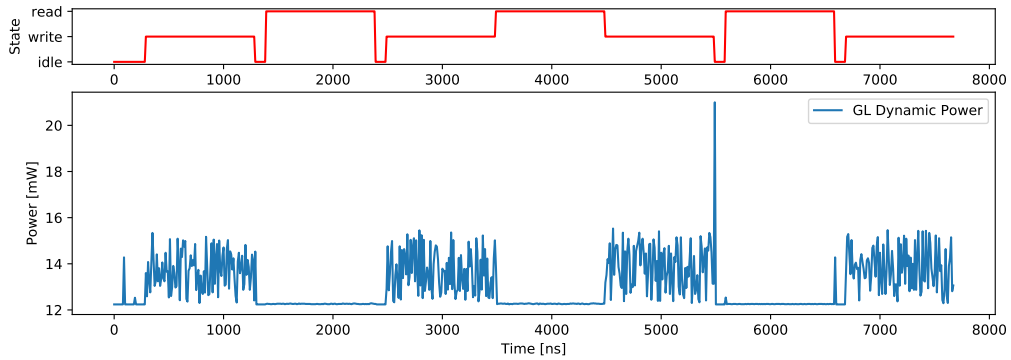



Abbildung 6.5.: Anwendungsfall 2: Leistungsaufnahme-Trace der Speicherkomponente mit allen Modi und explizit unterschiedlichen HDs auf den Dateneingängen.

und damit die Wahrscheinlichkeit zu einer Abhängigkeit zwischen Datenänderung und Leistungsaufnahme am größten. Beim lesenden Zugriff ist eine Variation zu sehen, diese ist aber deutlich geringer als im schreibenden Zugriff. Der Lesevorgang wird daher auch auf eine datenabhängige Leistungsaufnahme untersucht.

Auf Basis dieses Anwendungsfalls wird eine lineare Regression für die Zustände `write` und `read` durchgeführt. Dabei geschieht die Auswahl über alle Kombinationen der Dateneingänge und -ausgänge (`data_in`, `addr_in` und `data_out`). Die entsprechenden Modelle werden sowohl gegen den Synthese-Trace (Anwendungsfall 2) als auch gegen einen neuen Trace mit zufälligen Daten (Anwendungsfall 3) getestet.

Für den Zustand `write` sind die Ergebnisse in Abbildung 6.6 gezeigt. Für alle Modelle ist der durchschnittliche Fehler sehr niedrig. Beim absoluten durchschnittlichen Fehler und der Standardabweichung zeigt sich, dass nur bei Hinzunahme der HD von `Din` die Abweichung deutlich gesenkt werden kann. Die absolute Abweichung sinkt für Anwendungsfall 2 und 3 um 88,31% und 62,73% und die Standardabweichung um 86,56% und 60,57%. Das bedeutet, dass es eine große Abhängigkeit zwischen der Datenänderung an `Din` und der Leistungsaufnahme gibt. Alle anderen Dateneingänge und -ausgänge zeigen eine sehr geringe Verbesserung, wenn sie mit betrachtet werden. Demnach reicht es für diesen Speicher aus, `Din` als datenabhängigen Eingang mit in die Berechnung der Leistungsaufnahme einzubeziehen. Die Fehler für die Modelle ohne `Din` sind in Anwendungsfall 2 deutlich höher als in Anwendungsfall 3. Der Grund dafür ist, dass in Anwendungsfall 3 Zufallsdaten genutzt werden und in Anwendungsfall 2 die Daten so gewählt sind, dass möglichst alle HDs vorkommen. Dadurch ist die durchschnittliche Anzahl der schaltenden Bits in Anwendungsfall 2 wesentlich höher und somit auch die datenabhängige Leistungsaufnahme. Ohne Betrachtung von `Din` als Einflussgröße ist die Abweichung hier bedeutend höher.

Die Ergebnisse für den Zustand `read` sind in Abbildung 6.7 zu sehen. Dieses Ergebnis zeigt eine Ausnahme, denn trotzdem die Daten auf `Din` nicht geändert werden, sinkt die Abweichung, wenn dieser Eingang mit betrachtet wird. Der Grund dafür ist beim

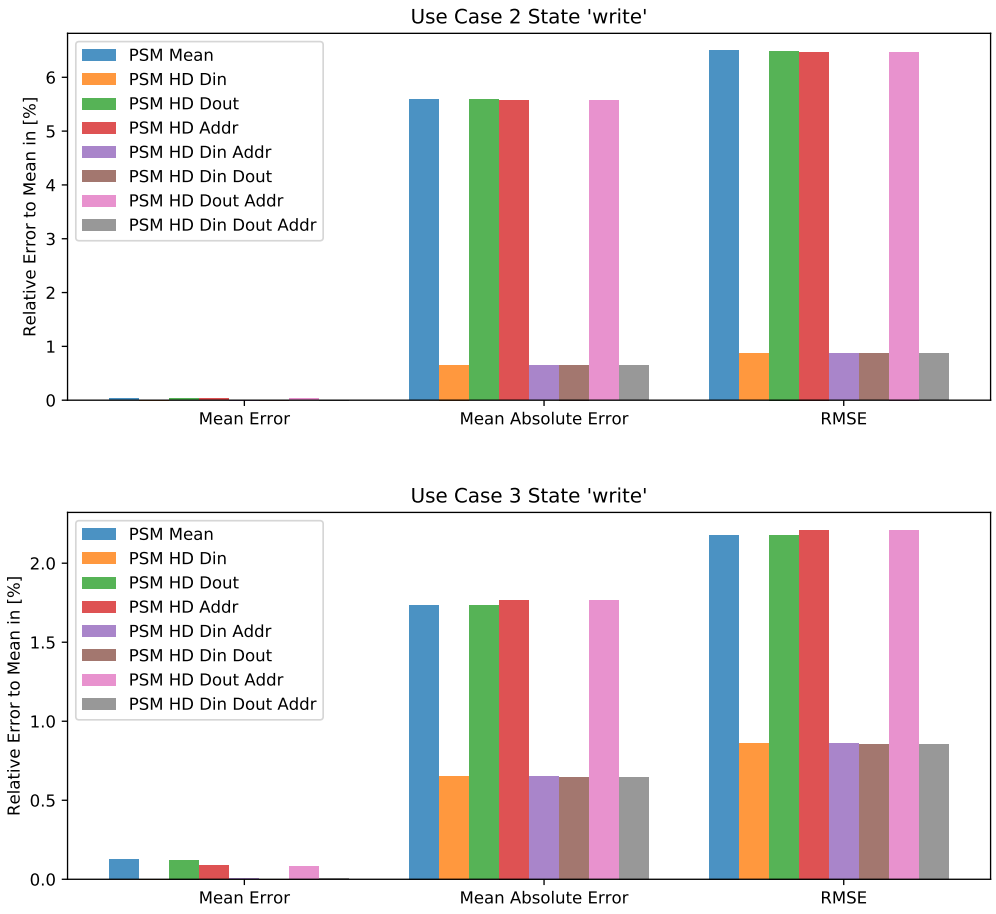



Abbildung 6.6.: Fehlergrößen von verschiedenen PSMs für den Zustand `write`.

Umschalten vom schreibenden auf den lesenden Zugriff zu finden. Hier wird `Din` vom letzten geschriebenen Wert auf den Wert 0 gestellt. Dieser Umschaltvorgang ruft auch bei einem lesenden Vorgang einen so hohen Unterschied in der Leistungsaufnahme hervor, dass hier eine höhere Korrelation herrscht als zu `Dout`. Lässt man im lesenden Betrieb den `Din` unverändert, findet die lineare Regression keinen Parameter, da die HD immer den Wert 0 hat und die resultierende Leistungsaufnahme nicht existiert. Der Ausgang `Dout` hat wie im schreibenden Betrieb einen zu vernachlässigenden Einfluss. Des Weiteren fällt auf, dass die Abweichung für `read` deutlich niedriger ist. Das liegt an der geringen Varianz. Ein ähnliches Bild zeigt sich für den Zustand `idle` und wird daher an dieser Stelle nicht gezeigt.

Die resultierende PSM hat einen äquivalenten Aufbau zur PrSM, wie in Abbildung 6.9 zu sehen. Im Unterschied zur PrSM aus Abbildung 6.3 hat der Zustandsübergang mit `write` als Start- und Zielzustand das PSM-Event `write` annotiert. Nur so bekommt die PSM bei jeder neuen Eingabe ein Signal zur Neuberechnung der Leistungsaufnahme auf Basis der

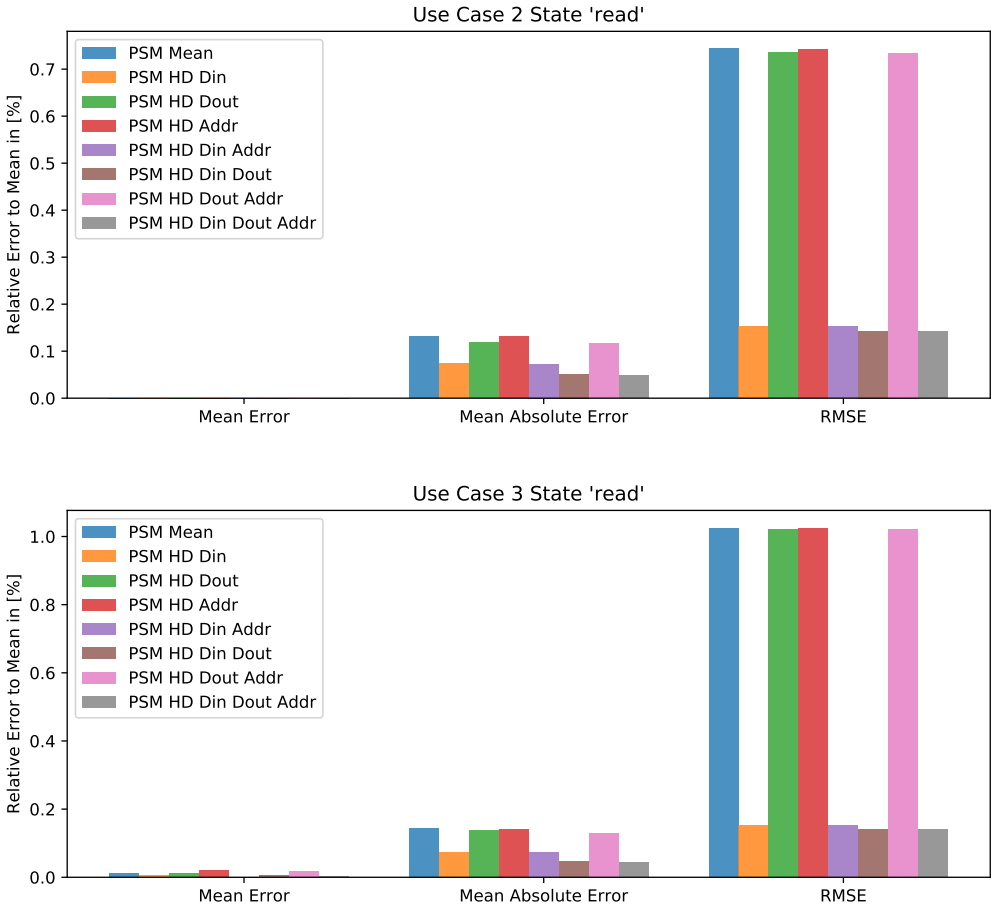



Abbildung 6.7.: Fehlergrößen von verschiedenen PSMs für den Zustand `read`.

geänderten Daten.

Für die Zustände `idle` und `read` werden die Durchschnittswerte, die aus allen Leistungsaufnahmewerten der kompletten Zeit, zu der Zustand aktiv ist, berechnet. Für den Zustand `write` ergibt sich die Leistungsaufnahme aus nachfolgender Formel:

$$\begin{aligned} P &= c + HD(Din_t, Din_{t-1}) \cdot m \\ &= 12,302\,\text{mW} + HD(Din_t, Din_{t-1}) \cdot 93,268 \cdot 10^{-03}\,\text{mW} \end{aligned} \tag{6.3}$$

mit der Konstanten $c$ und dem Linearparameter $m$ als Ergebnis der linearen Regression.

Damit ergibt sich für Anwendungsfall 3 der in Abbildung 6.9 gezeigte Leistungsaufnahme-Trace. Zum Vergleich ist der entsprechende GL-Trace mit dargestellt. Es wird deutlich, dass eine sehr gute Annäherung an diesen erreicht wird. Wie in Anwendungsfall 2 gibt es auch hier eine Leistungsspitze bei 5500 ns durch den Zustandswechsel von `write` zu `idle`, die durch viele interne Schaltvorgänge entsteht. Dadurch, dass die Leistungsspitze nur mit

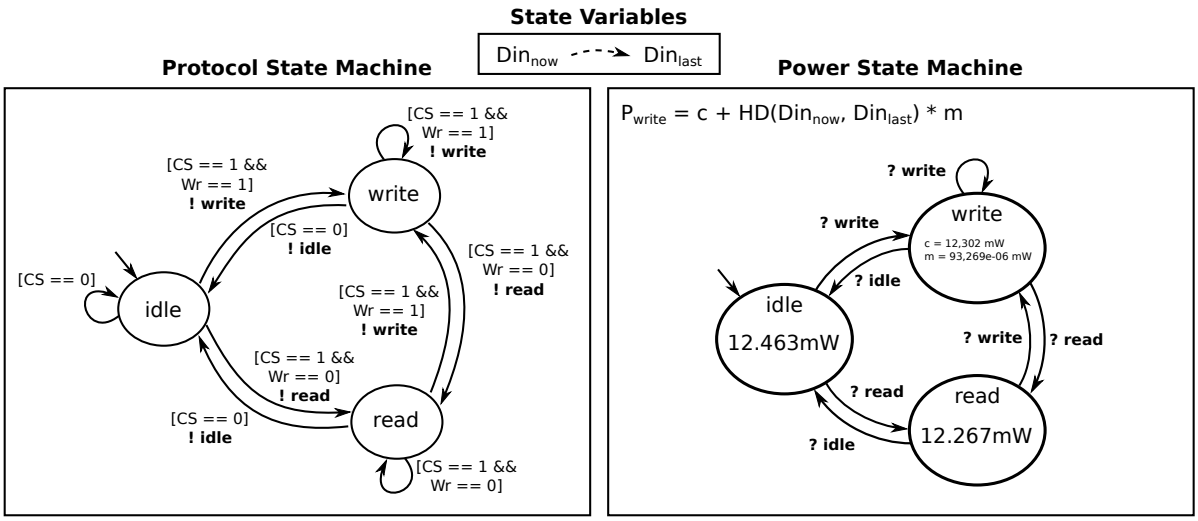



Abbildung 6.8.: PSM mit Modellierung der datenabhängigen Leistungsaufnahme des Dateneingangs `Din` für die Speicherkomponente. Zustandsvariablen speichern die notwendigen Daten als Grundlage für die Berechnung im Zustand `write`.

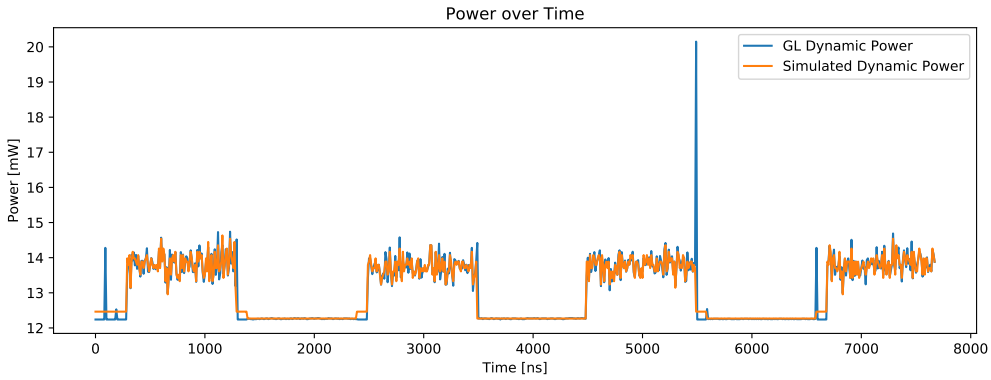



Abbildung 6.9.: Anwendungsfall 3: Leistungsaufnahme-Trace für eine PSM mit Modellierung der datenabhängigen Leistungsaufnahme im Zustand `write` im Vergleich zum GL-Trace für eine Speicherkomponente.

einem Zustandswechsel und nicht mit dem Zustand oder den Datenänderungen korreliert, wird diese als einer von vielen Werten beim Berechnen des Durchschnittswertes für den Zustand `idle` mit einberechnet und führt dazu, dass die Leistungsaufnahme im Vergleich zum GL-Trace (12,24 mW) etwas erhöht ist. Dadurch taucht diese Spitze auch nicht explizit in der Simulation auf. Durch Verlängern der Aktivitätszeiten im Zustand `idle` nehmen die Umschaltphasen einen geringen Zeitraum ein und die Abweichung kann über den sinkenden Einfluss minimiert werden. Durch das explizite Herauslassen dieser Umschaltphasen bei der Durchschnittswertberechnung wird das genauste Ergebnis erzielt. Hier tauchen

dann im Vergleich zum GL-Trace nur noch die Umschaltspitzen als Abweichungen auf.

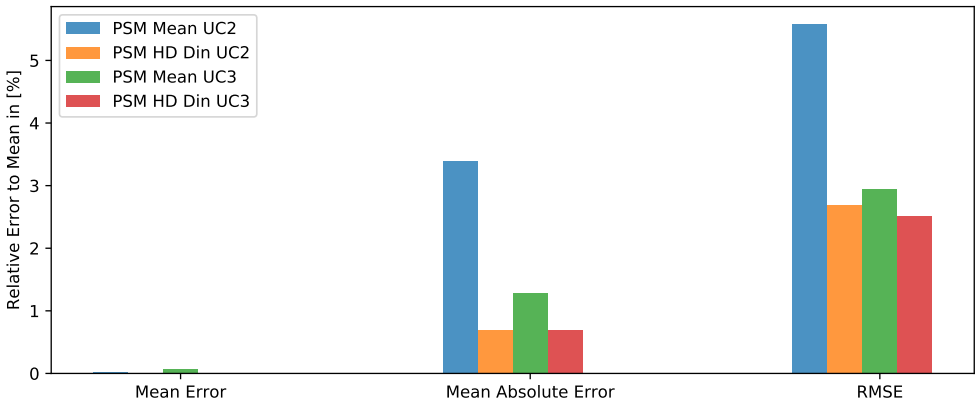



Abbildung 6.10.: Fehlergrößen von PSM Mean und PSM HD Din für die Anwendungsfälle 2 und 3.

Die Abweichung für des gesamte Modell (PSM HD Din) im Vergleich zum Modell mit Durchschnittswerten (PSM Mean) ist in Abbildung 6.10 für Anwendungsfall 2 (UC2) und Anwendungsfall 3 (UC3) gezeigt. Es wird deutlich, dass die durchschnittliche Abweichung bereits beim Modell PSM Mean sehr niedrig ist. Diese kann durch die Modellierung der Datenabhängigkeit nahezu auf 0 gesenkt werden. Bei den absoluten Abweichungen und der Standardabweichung ist der Einfluss deutlich größer. Diese zeigen, dass Anwendungsfall 3 eine geringe Varianz hat und damit auch das Modell PSM Mean deutlich bessere Ergebnisse zeigt. Diese Werte belegen, dass bei einem Anwendungsfall, bei dem der mittlere Fehler groß abweicht, weil beispielsweise Daten mit einer großen HD vorhanden sind, das Modell PSM HD Din einen deutlich geringeren mittleren Fehler hat, weil die datenabhängige Leistungsaufnahme mit betrachtet wird.

Wie Abschnitt 4.8.3 vorgestellt, gibt es sogenannte Burst-Transaktionen, bei denen auf ESL mehrere Übertragungszyklen in einem Kommunikationsevent dargestellt werden. Wie beschrieben, ist die entwickelte PSM in der Lage, diese Transaktionen korrekt zu modellieren, indem über alle Zyklen der Durchschnittswert berechnet und über die gesamte Zeit dargestellt wird. Dafür wird eine Zustandsvariable eingesetzt, die die komplette Transaktion inklusive aller Datenänderungen sichert. Dadurch kann die PSM alle Änderungen betrachten und die korrekte durchschnittliche Energie berechnen.

### 6.2.3. Berechnungskomponenten

Berechnungskomponenten stellen Komponenten dar, die in der Regel eine oder mehrere Eingangsgrößen besitzen und diese zu einer oder mehreren Ausgangsgrößen berechnen.

Einige dieser Komponenten können in verschiedenen Modi betrieben werden. Zum Beispiel kann eine Komponente entweder zwei Zahlen addieren oder subtrahieren.

Da die einzelnen Berechnungen in den meisten Fällen unabhängig voneinander sind, müssen keine Werte zwischengespeichert werden, die aufeinanderfolgende Berechnungen benötigen. Eine Komponente speichert lediglich dann Werte zwischen, wenn die Berechnung der Ausgangsgrößen nicht in einem Taktzyklus durchgeführt werden kann. Hier wird das Zwischenergebnis temporär gesichert, um im nächsten Taktzyklus die Berechnung fortführen zu können. Je nach Komplexität der Berechnung können diese auch mehrere Taktzyklen beanspruchen.

Dies bedeutet, dass eine Berechnungskomponente in der Regel auch keinen Zustand in Form eines Ausführungszustandes besitzt, sondern nur einen Zustand in Form eines Operationsmodus. Die Berechnung in mehreren Taktzyklen kann als ein Zustand betrachtet werden, da die Berechnung ein streng inkrementeller Prozess ist. Dieser kann aber je nach Berechnung und Eingangsgrößen unterschiedlich lange Ausführungszeiten besitzen.

Für die meisten Berechnungskomponenten ist es charakteristisch, dass diese eine datenabhängige Leistungsaufnahme besitzen. Das liegt daran, dass Daten – anders als bei Verschlüsselungskomponenten – meist ohne interne Rückkopplungen und in wenigen Berechnungszyklen direkt algorithmisch verrechnet werden. Dadurch führt die Anzahl an veränderten Bits der Eingangssignale oft zu einer ähnlich korrelierten Anzahl an veränderten Bits innerhalb der Komponente. Da diese schaltenden Bits ausschlaggebend sind für die dynamische Leistungsaufnahme, ergibt sich hier eine Korrelation zwischen Änderungen der schaltenden Bits und der dynamischen Leistungsaufnahme (siehe Abschnitt 4.8). Diese Abhängigkeit wird in den untersuchten Komponenten analysiert und bewertet.

#### MAC

Die *Multiply-Accumulate* (MAC) Komponente ist auch bekannt als *Multiply-Add* (MAD) Komponente. Sie führt eine Multiplikation von zwei Werten `A` und `B` durch und addiert diese zu dem letzten Ergebnis `C`. Diese Operation wird häufig für die Signalverarbeitung benötigt und daher oftmals als spezieller Hardwareblock zur Verfügung gestellt, damit diese Operation in wenigen Zyklen und ressourceneffizient berechnet werden kann.

Die hier vorgestellte Komponente hat drei verschiedene Berechnungsmodi:

1. multipliziere die beiden Eingangswerte und addiere das letzte Ergebnis,
2. multipliziere die beiden letzten Eingangswerte und addiere das vorletzte Ergebnis sowie
3. konkateniere die beiden letzten Eingangswerte und addiere diesen Wert auf das vorletzte Ergebnis.

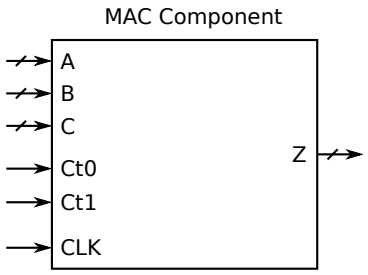



Abbildung 6.11.: MAC Komponente für die kombinierte Multiplikation und Addition des letzten Ergebnisses.

Um die verschiedenen Modi einzustellen, hat die Komponente zwei Kontrolleingänge `Ct0` und `Ct1`. Das Blockschaltbild ist in Abbildung 6.11 gezeigt.

Dadurch, dass eine Berechnungskomponente in der Regel nur einen oder wenige Betriebsmodi hat, werden nur diese in der PrSM dargestellt. Aus diesem Grund hat die PrSM für die MAC Komponente die drei vorgestellten Modi. Die Belegung ist wie folgt:

- Modus 0: `Ct1 == 0 && Ct2 == 0` ,
- Modus 1: `Ct1 == 1` und
- Modus 2: `Ct1 == 0 && Ct2 == 1`.

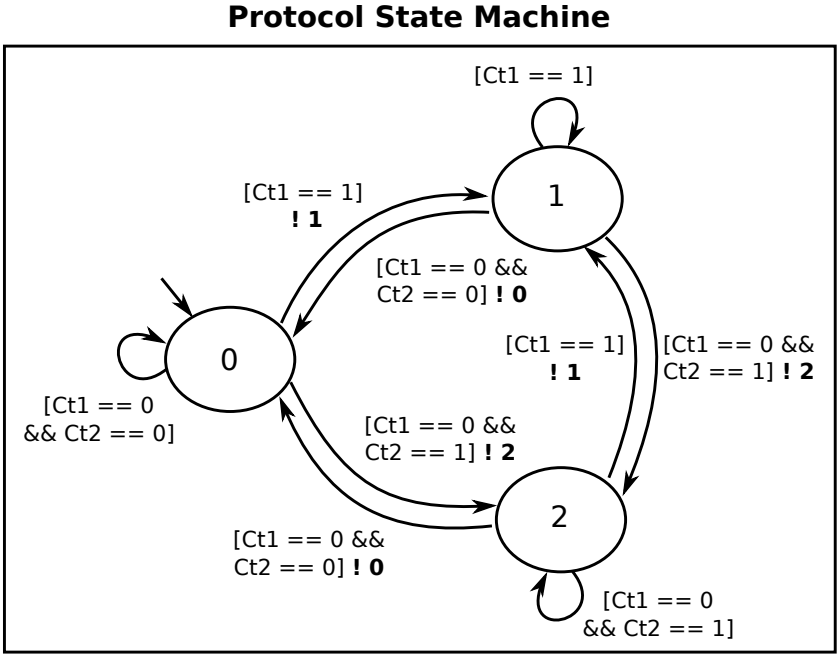



Abbildung 6.12.: PrSM der MAC-Komponente.

Damit ist die PrSM für die MAC Komponente wie in Abbildung 6.12 aufgebaut. Auch hier besitzen alle Zustandsübergänge mit gleichem Eingangs- und Ausgangszustand kein PSM-Event, da kein Zustandswechsel stattfindet, sondern nur eine neue Eingabe vorliegt.

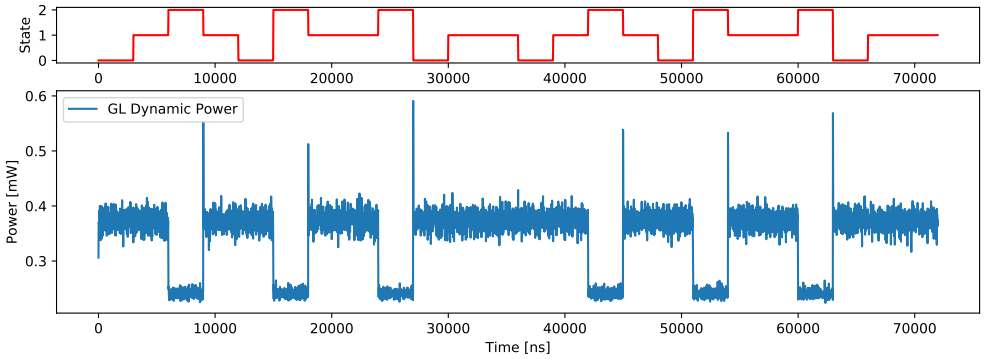



Abbildung 6.13.: Leistungsaufnahme-Trace der MAC Komponente mit allen Modi (gezeigt über den Grafiken) und zufälligen Werten.

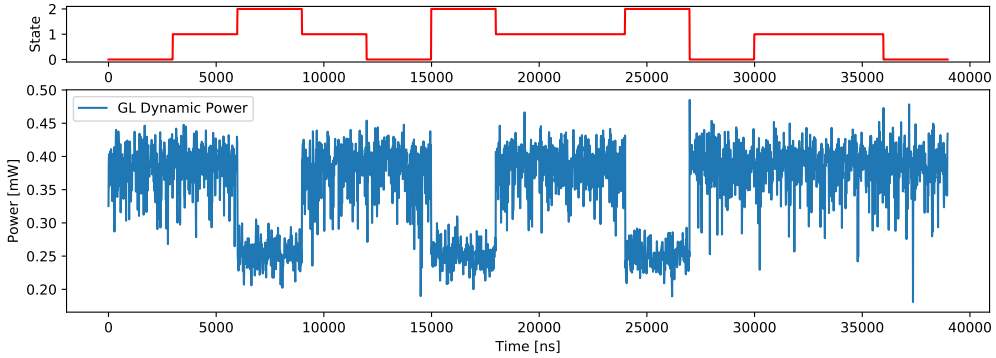



Abbildung 6.14.: Leistungsaufnahme-Trace der MAC Komponente mit allen Modi (gezeigt über den Grafiken) und zufälligen HD-Werten.

Die Komponente wird in zwei Anwendungsfällen mit unterschiedlichen Stimuli untersucht. Es werden dabei nacheinander alle Betriebsmodi in verschiedenen Reihenfolgen und Längen benutzt. Bei dem ersten Anwendungsfall (Abbildung 6.13) werden zufällige Daten zur Berechnung gewählt. Bei dem zweiten Anwendungsfall (Abbildung 6.14) werden explizit unterschiedliche HDs in zufälliger Auswahl eingestellt.

In Anwendungsfall 1 wird deutlich, dass bei zufälligen Daten der Mittelwert sehr stabil ist und die Varianz mit 0,0598 mW in einem einheitlichen Bereich stattfindet. Modus 0 und Modus 1 sind bezüglich ihrer Charakteristik identisch. Dies liegt daran, dass hier die gleichen Operationen durchgeführt werden. Modus 2 hat einen deutlich niedrigeren Mittelwert als

auch eine geringere Varianz. Der Grund dafür ist, dass hier keine Multiplikation sondern nur eine Konkatenation der beiden Eingangswerte `A` und `B` und eine Addition durchgeführt wird. Bei dem Wechsel von Modus 2 auf Modus 0 oder 1 ist eine große Spitze zu erkennen. Dieses Verhalten ist häufig bei arithmetischen Komponenten zu beobachten, wenn von einem Modus mit wenig Aktivität in einen Modus mit hoher Aktivität gewechselt wird. Die Ursache ist auch hier wieder eine hohe Anzahl von schaltenden Logikgattern und Signalleitungen.

In Anwendungsfall 2 gibt es ebenso erkennbare Mittelwerte für die einzelnen Modi. Der durchschnittliche Abweichung als auch die Varianz sind aber höher als in Anwendungsfall 1, 3,898% und 2,33%. Dies zeigt, dass durch explizite Auswahl der HD die Randbereiche der Energieaufnahme herbeigeführt werden können.

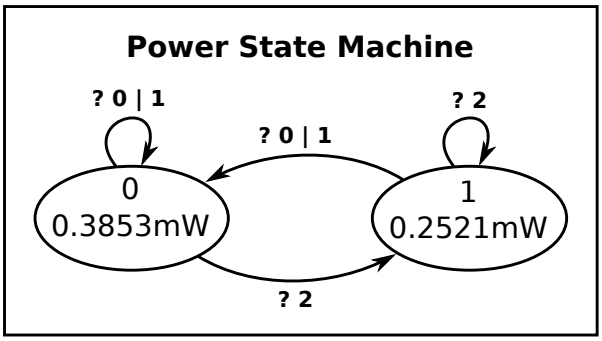



Abbildung 6.15.: Automatisch synthetisierte PSM der MAC-Komponente auf Basis von Anwendungsfall 2.

Wird der Anwendungsfall 2 als Grundlage für die automatische Synthese genutzt, wird die PSM aus Abbildung 6.15 generiert. Dies zeigt, dass die PSM-Zustände 0 und 1 so ähnlich sind, dass sie zusammen in einen Zustand synthetisiert werden.

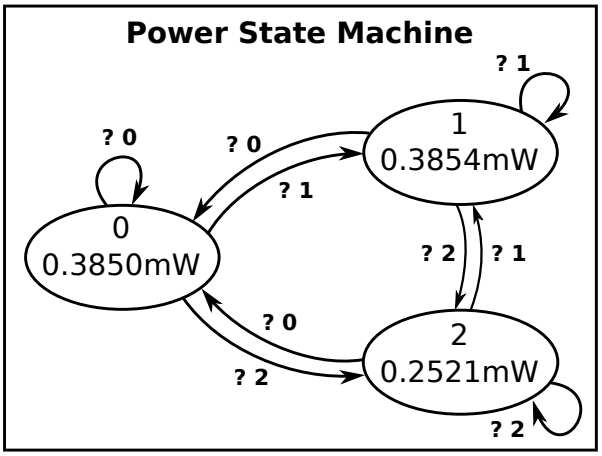



Abbildung 6.16.: PSM der MAC-Komponente mit durchschnittlichen Werten.

Als Vergleich werden basierend auf beiden Anwendungsfällen manuell Automaten erstellt, die für jeden PrSM-Zustand einen einzelnen PSM-Zustand besitzen. Es wird ein Automat erstellt, der einen Durchschnittswert modelliert und einer, der die datenabhängige Leistungsaufnahme abbildet. Basierend auf Anwendungsfall 2 ergibt sich für eine PSM mit durchschnittlichen Werten der in Abbildung 6.16 gezeigt Aufbau. Es wird deutlich, dass sich die Werte für Zustand 0 und 1 leicht zu dem einen Zustand 0 des automatisch synthetisierten Automaten aus Abbildung 6.15 unterscheiden.

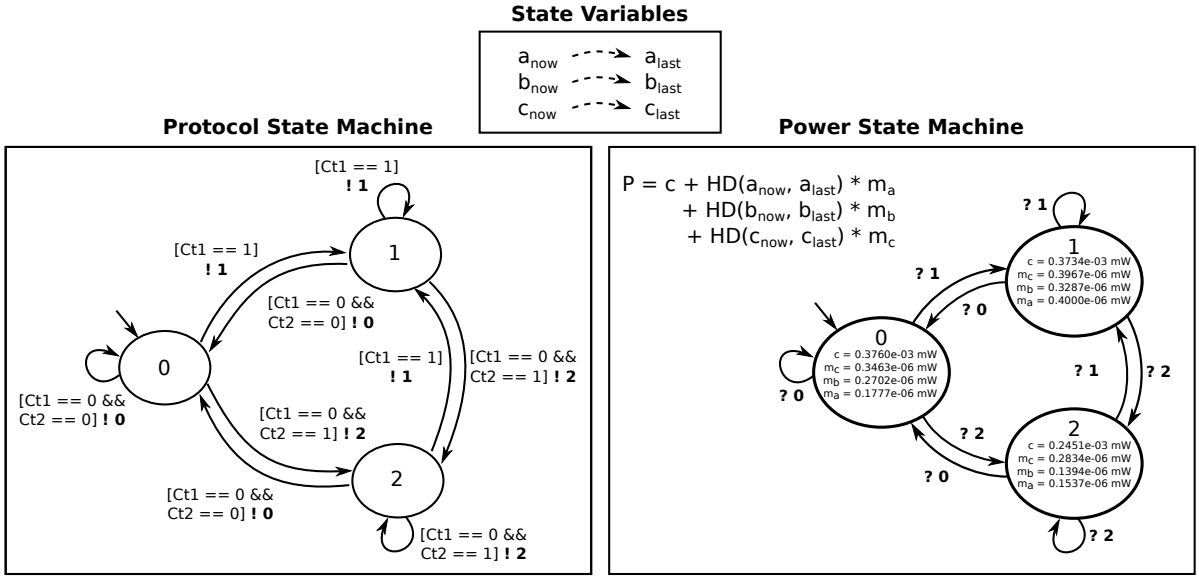



Abbildung 6.17.: PSM-Modell der MAC-Komponente mit Modellierung der datenabhängigen Leistungsaufnahme. Die PrSM speichert die aktuellen und letzten Werte in den Variable `now` und `last`. Die PSM berechnet daraus die HD und mit Hilfe der Linearparameter die resultierende Leistungsaufnahme.

Für eine PSM mit Modellierung der datenabhängigen Leistungsaufnahme ergibt sich mit Anwendungsfall 2 als Grundlage der in Abbildung 6.17 gezeigte Automat. Es werden die HDs der Eingänge `a`, `b` und `c` verwendet. Dafür werden für jeden Eingang zwei Zustandsvariablen `now` und `last` angelegt, die den aktuellen und den letzten Eingangswert speichern. Diese werden dann von der PSM genutzt, um die resultierende HD zu berechnen.

Damit bei jeder Eingabe eine neue Leistungsaufnahme in der PSM berechnet wird, müssen die Zustandsübergange der PrSM mit selbem Start- und Zielzustand auch mit einem PSM-Event versehen werden. Nur so gibt die PrSM ein Signal an die PSM zur Neuberechnung der Leistungsaufnahme. Diese wird über eine lineare Funktion mit dem Konstantanteil $c$ und den Linearparametern $p_a, p_b, p_c$ für die einzelnen Eingänge berechnet. Die Parameter für jeden Zustand werden über eine lineare Regression ermittelt.

Die Abweichungen für Anwendungsfall 2 sind in Abbildung 6.18 gezeigt. Es wird ersichtlich, dass alle PSMs mit Anwendungsfall 2 als Erstellungsgrundlage (UC2) eine sehr geringe

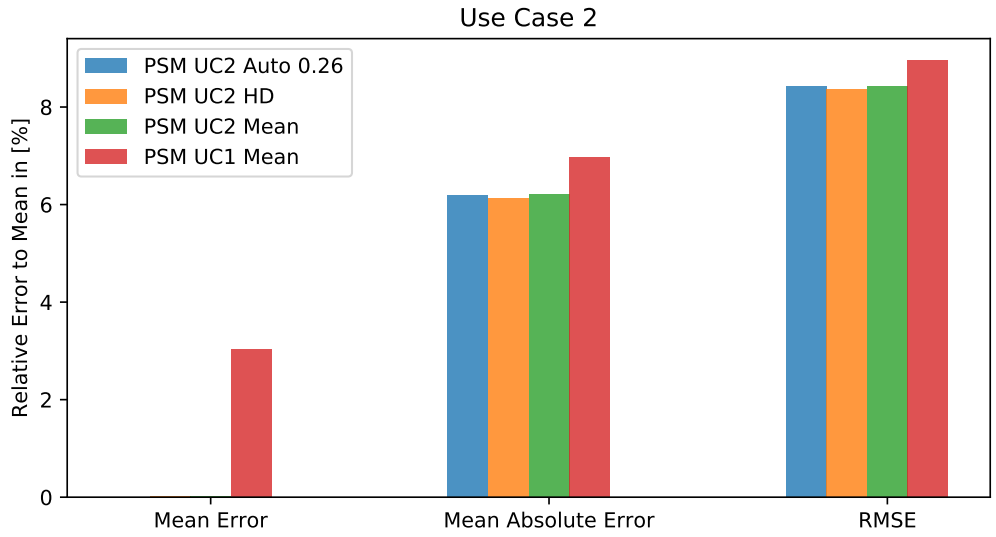



Abbildung 6.18.: Fehlergrößen von verschiedenen PSMs für Anwendungsfall 2

.

durchschnittliche Abweichung haben. Die automatisch synthetisierte PSM (PSM UC2 Auto 0.26) als auch die manuelle mit durchschnittlichen Werten (PSM UC2 Mean) haben etwa vergleichbare Ergebnisse. Dies entspricht der Erwartung, da sich die Werte der Zustände zwischen den beiden Automaten nicht groß unterscheiden. Die Modellierung der datenab-

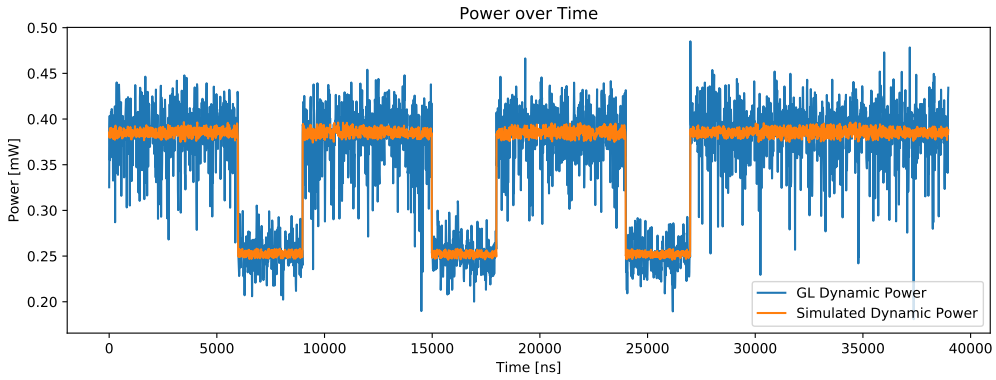



Abbildung 6.19.: Vergleich der Leistungsaufnahme-Traces von GL-Design und PSM UC2 HD für die MAC Komponente.

hängigen Leistungsaufnahme (PSM UC2 HD) bringt eine kleine Verbesserung. Das bedeutet, dass Datenabhängigkeiten vorhanden sind, aber in einem so kleinen Umfang, dass diese nur zu minimalen Verbesserungen beitragen. Dies wird besonders beim Blick auf die Leistungsaufnahme-Traces in Abbildung 6.19 ersichtlich. Hier wird deutlich, dass nur eine

geringe Annäherung der modellierten Leistungsaufnahme an die wirkliche stattfindet.

Die Automaten auf Basis von Anwendungsfall 1 haben hier größere Abweichungen. Die PSM mit durchschnittlichen Werten (PSM UC1 Mean) hat bei dem Durchschnittswert eine Abweichung von 2,99%, ist aber bei dem absoluten Durchschnittswert und der Standardabweichung lediglich 0,77% und 0,51% schlechter. Dies zeigt, dass der Mittelwert stark von den verwendeten Daten abhängt, sich aber bei unterschiedlichen Werten durch eine hohe Streuung trotzdem nur geringe absolute und quadratische Abweichungen ergeben.

Mit einer Modellierung der Datenabhängigkeiten auf Basis von Anwendungsfall 1 ergibt sich ein Modell, welches für Anwendungsfall 2 unverhältnismäßig große Abweichungen erzeugt. Diese sind daher nicht in Abbildung 6.18 gezeigt. Das macht deutlich, dass für eine Bestimmung der datenabhängigen Leistungsaufnahme die richtigen Stimuli von entscheidender Bedeutung sind, da ansonsten ein nicht verwendbares Modell entsteht.

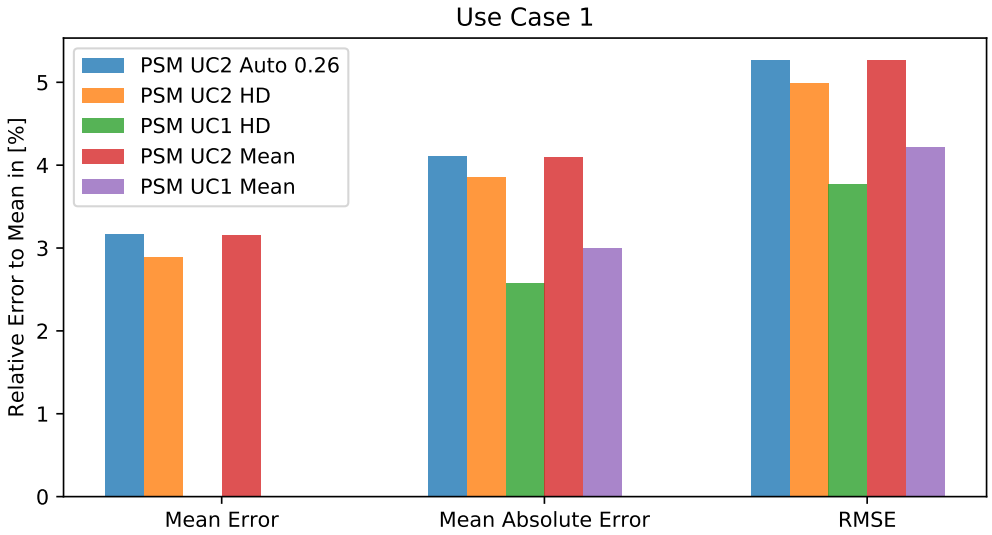



Abbildung 6.20.: Fehlergrößen von verschiedenen PSMs für Anwendungsfall 1 bei der MAC Komponente.

Die Ergebnisse für Anwendungsfall 1 sind in Abbildung 6.20 gezeigt. Hier zeigen, wie erwartet, die beiden Automaten mit Anwendungsfall 1 als Basis eine niedrige Abweichung beim Durchschnittswert. Bei den absoluten und quadratischen Abweichungen zeigt sich die mögliche Verbesserung durch die Modellierung der datenabhängigen Leistungsaufnahme (PSM UC1 HD). Der Trace für diese PSM ist in Abbildung 6.21 gezeigt. Hier wird deutlich, dass das Modell eine sehr gute Annäherung an den GL-Trace findet. Wie oben beschrieben, sind die Fehler für Anwendungsfall 2 so groß, dass das Modell praktisch nicht angewendet werden kann.

Für die Modelle, die auf Basis von Anwendungsfall 2 synthetisiert wurden, zeigt sich, dass hier eine deutliche Verbesserung durch die Modellierung der Datenabhängigkeiten

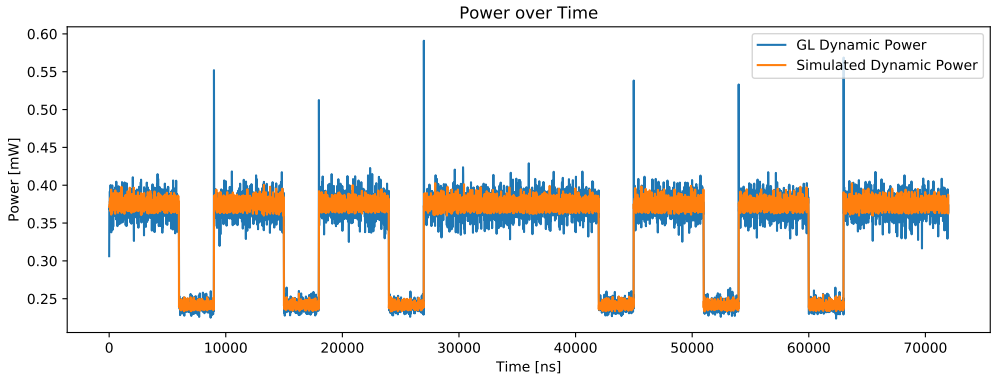



Abbildung 6.21.: Vergleich der Leistungsaufnahme-Traces von GL-Design und PSM UC1 Mean für die MAC-Komponente.

(PSM UC2 HD) erzielt werden kann. Die Ergebnisse sind aber schlechter als für die PSM mit Durchschnittswerten auf Basis von Anwendungsfall 1 (PSM UC1 Mean). Dies zeigt, dass nur eine geringe lineare Abhängigkeit zwischen den HDs der Eingangsdaten und der Leistungsaufnahme besteht.

Als Fazit für die MAC-Komponente bleibt festzuhalten, dass

- die automatische Synthese einen Automaten gefunden hat, der vergleichbar ist mit händisch generierten PSMs mit durchschnittlichen Werten,
- die Wahl der Stimuli-Daten sehr entscheidend für den Mittelwert ist und diese somit möglichst den Daten im späteren Anwendungsfeld entsprechen sollten und
- die Modellierung der Datenabhängigkeiten
  1. Verbesserungen herbeiführen können,
  2. zu großen Fehlern führen, wenn die Stimuli für die Charakterisierung falsch gewählt werden und
  3. stark von dem eingesetzten Anwendungsfall abhängen können.

### 6.2.4. Verschlüsselungskomponenten

Die Sicherheit gegen Angriffe hat in der Vergangenheit stark an Bedeutung gewonnen und ist heute fester Bestandteil jedes eingebetteten Systems. Für die Verschlüsselung der Kommunikation zwischen Systemen und Komponenten wurden mehrere Algorithmen entwickelt mit verschiedenen Vor- und Nachteilen als auch unterschiedlich hohen Sicherheitsklassen. Einer dieser Algorithmen ist der sogenannte *Advanced Encryption Standard* (AES), auch bekannt als *Rijndeal* [20]. Eine Komponente, die diesen Algorithmus implementiert, soll in diesem Abschnitt als Referenz charakterisiert und bewertet werden.

Verschlüsselungsalgorithmen zeichnen sich dadurch aus, dass sie eine hohe Permutation der Eingangsdaten durchführen. In der Regel passiert dies mit Hilfe eines vom Benutzer definierten Schlüssels. Diese Permutation wird meist in mehreren Schritten durchgeführt, um eine Rückführung der Originalwerte zu erschweren. Diese Eigenschaften wirken sich direkt auf die Leistungsaufnahme auf. Durch die hohe Permutation werden alle Datenabhängigkeiten in der Leistungsaufnahme eliminiert und sorgen für einen sehr stabilen Mittelwert. Dieser unterscheidet sich in der Regel nicht entscheidend zwischen den einzelnen Permutationsschritten. Einige Verschlüsselungen haben eine dynamische Ausführungszeit abhängig vom Schlüssel und den Daten. Hier ist die Betrachtung dieser Zeit im PSM-Modell entscheidend. In manchen Fällen unterscheidet sich der Mittelwert der Leistungsaufnahme zwischen Ver- und Entschlüsselung. Diese Eigenschaften sollen beispielhaft an einer AES-Komponente gezeigt werden.

### AES

Die hier eingesetzte AES-Komponente hat folgende Ein- und Ausgänge:

- Dateneingang mit 128 Bit Datenbreite,
- Datenausgang mit 128 Bit Datenbreite,
- Schlüsseleingang mit 128 Bit Datenbreite,
- Mode-Eingang zum Verschlüsseln und Entschlüsseln,
- Takt- und Rücksetz-Eingang,
- Eingangssignal für die Übernahme der Eingangsdaten und
- Ausgangssignal zur Anzeige der Ausgangsdatenverfügbarkeit.

In Abbildung 6.22 ist ein Blockschaltbild der Komponente gezeigt.

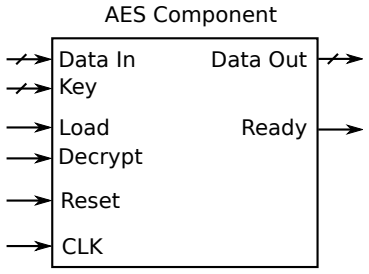



Abbildung 6.22.: AES-Komponente zur Ver- und Entschlüsselung von Datenströmen.

Aus den verschiedenen Ein- und Ausgaben ergibt sich die in Abbildung 6.23 gezeigt PrSM. Diese zeigt die beiden Betriebszustände `encrypt` und `decrypt` als auch den Ruhezustand `idle`. Es gibt keinen weiteren Zustand für das Zurücksetzen, da dies kein Betriebsmodus

darstellt, sondern nur in Ausnahmefällen kurzfristig eingesetzt wird, um einen definierten Startzustand herzustellen. Aufgrund der geringen Auftretenswahrscheinlichkeit und aktiven Zeit ist er für eine Betrachtung der Leistungsaufnahme nicht relevant.

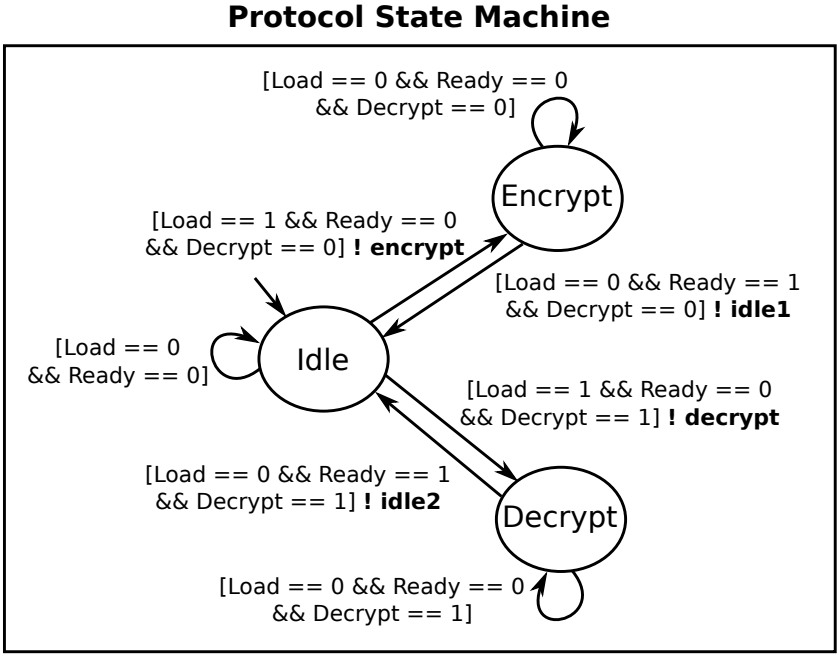



Abbildung 6.23.: PrSM der AES-Komponente.

Wird das Zurücksetzen bei einer Komponente häufiger durchgeführt oder dauert dieser Zustand länger an, kann ein zusätzlicher Zustand in der PrSM modelliert werden, damit der nun größere Einfluss auf die Leistungsaufnahme korrekt in der PSM dargestellt werden kann.

Es wird deutlich, dass die Zustandsübergänge auf den eigenen Zustand kein PSM-Event auslösen, da nur ein Zustandswechsel eine Änderung in der Leistungsaufnahme herbeiführt. Damit die beiden Zustandsübergänge, die zum Zustand `idle` gehen, unterschieden werden können, sind diese mit den beiden unterschiedlichen PSM-Events `idle1` und `idle2` versehen.

In Abbildung 6.24 und Abbildung 6.25 sind zwei Ausschnitte von GL-Traces von zwei hintereinander geschalteten AES-Komponenten gezeigt. Die erste verschlüsselt einen Datenstrom, der danach von der zweiten wieder entschlüsselt wird.

Die zugeführten Daten weisen folgende Eigenschaften auf:

1. die ersten sieben Datensätzen wurden mit dem Schlüsselwert 0 und zufälligen Daten geschrieben,
2. die nächsten 21 Datensätze mit einem zufälligen Schlüssel und Daten mit dem Wert 0 und

3. die restlichen Daten mit zufälligen Daten und einem Schlüssel beginnend bei 0, bei dem mit jedem weiteren Durchgang das höchstwertigste Bit auf eins gesetzt wird.

Hier wird deutlich, dass die Leistungsaufnahme weder beim Ver- noch beim Entschlüsseln von den Eingangsdaten abhängt und einen sehr stabilen Mittelwert aufweist. Einzige Ausnahme hier sind Daten mit dem Wert 0. Hier gibt es eine deutliche Abweichung des Minimalwertes. Dieser Anwendungsfall wird ignoriert, da in der Regel nur schützenswerte Daten verschlüsselt werden, die selten den Wert 0 aufweisen. Des Weiteren ist die Abweichung vom Mittel-, Maximal- und Minimalwert mit Ausnahme des eben benannten Sonderfalls sehr gering für die einzelnen Betriebszustände `idle`, `encrypt` und `decrypt`. Die beiden Betriebsmodi `Encrypt` und `Decrypt` unterscheiden sich deutlich in ihrer Charakteristik bei Maximal-, Minimal- und Durchschnittswert.

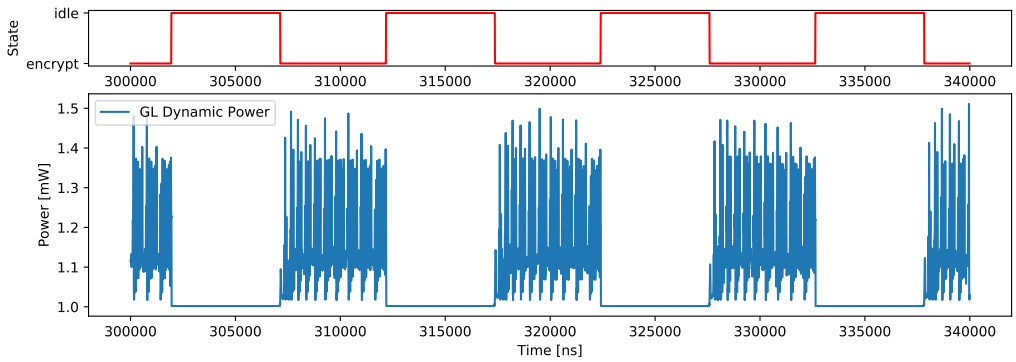



Abbildung 6.24.: Dynamische Leistungsaufnahme der AES-Komponente im Modus `Encrypt`.

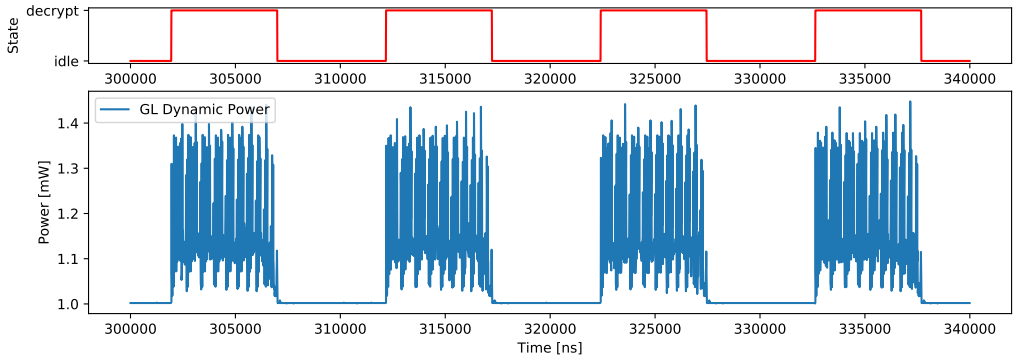


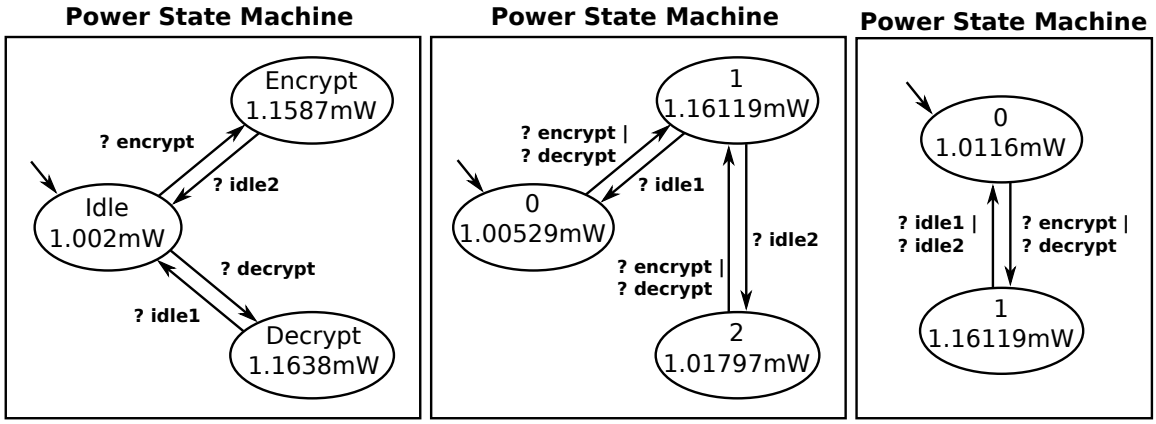

Abbildung 6.25.: Dynamische Leistungsaufnahme der AES-Komponente im Modus `Decrypt`.

Daraus ergeben sich die manuell und automatisch erstellten PSM-Modelle aus Abbildung 6.26. Es zeigt sich, dass alle drei PSMs unterschiedlich aussehen.

Der manuelle Entwurf aus Abbildung 6.26a bildet exakt die PrSM ab und verfolgt damit den intuitiven Ansatz, dass das funktionale Verhalten direkt auf die Leistungsaufnahme abgebildet werden kann. Die Berechnung der Zustandswerte erfolgt durch eine Mittelwertberechnung aller Leistungsaufnahmewerte zur aktiven Zeit.

Wird bei der automatischen Synthese ein Schwellwert von 0,16 eingestellt, wird ein Automat mit drei Zuständen generiert, wie in Abbildung 6.26b zu sehen. Im Unterschied zur manuellen Variante gibt es hier aber zwei Zustände für `Idle` (Zustand 0 und 2) und nur einen für `Encrypt` und `Decrypt`. Weiteres Dezimieren des Schwellwerts führt hier zu einer Zustandsexplosion und kann dadurch nicht in eine PSM synthetisiert werden.

Wird der Schwellwert etwas angehoben auf 0,17, hat die generierte PSM nur noch zwei Zustände, einen für `Idle` und einen für `Encrypt` und `Decrypt`. Durch den größeren Schwellwert werden alle Leistungsaufnahmewerte im Zustand `Idle` zusammengelegt.




(a) Manuell erstellte PSM

(b) Automatisch erstellte PSM mit dem Schwellwert 0,16

(c) Automatisch erstellte PSM mit dem Schwellwert 0,17

Abbildung 6.26.: Verschiedene PSMs für die AES-Komponente.

Diese Automaten werden mit drei verschiedenen Anwendungsszenarien simuliert und die Ergebnisse miteinander verglichen. Diese sind in Abbildung 6.27 zu sehen.

In Anwendungsfall 1 werden lediglich Daten verschlüsselt, in Anwendungsfall 2 Daten entschlüsselt und in Anwendungsfall 3 werden Daten sowohl ver- als auch entschlüsselt. Da in diesem Anwendungsfall alle Betriebsmodi benutzt werden, dient dies als Basis für die automatische Synthese für PSM 0,16 und PSM 0,17 als auch für die manuell erstellte PSM.

Es wird deutlich, dass die manuell erstellte im Durchschnitt den niedrigsten Fehler hat, aber für Anwendungsfall 3 den größten. Dies zeigt, dass die automatische PSM Synthese für einen spezifischen Anwendungsfall den Fehler größtmöglich senken kann, aber nicht zwingend für alle Anwendungsfälle das bestmögliche Ergebnis liefert. Die Konsequenz

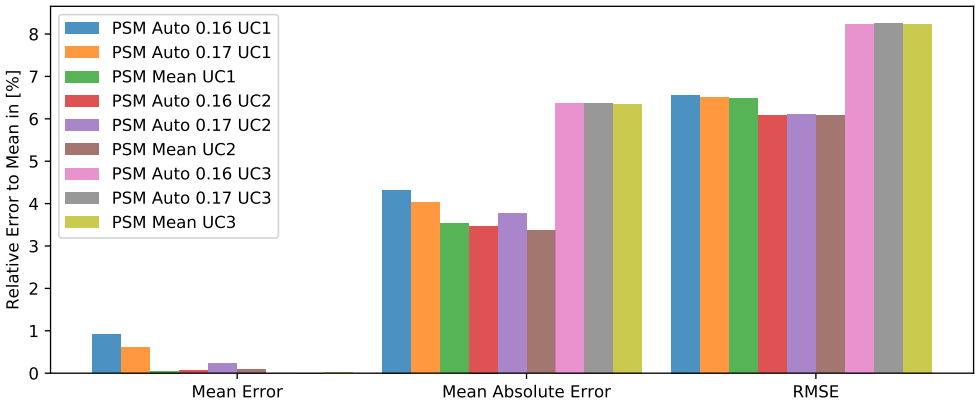



Abbildung 6.27.: Fehlergrößen von verschiedenen PSMs für mehrere Anwendungsfälle bei der AES Komponente.

ist, dass für eine automatische Synthese ein möglichst generischer und umfangreicher Anwendungsfall gewählt werden sollte.

Als weiteres Ergebnis ist festzuhalten, dass eine manuell erstellte PSM, die weitestgehend der PrSM entspricht, sehr gute Ergebnisse liefern kann. Dies trifft besonders für Verschlüsselungskomponenten zu, die so gut wie keine Datenabhängigkeiten aufzeigen.

Auch hier zeigen die Ergebnisse beim durchschnittlichen absoluten Fehler und der Standardabweichung, dass trotz gutem Mittelwert die Einzelwerte im Mittel große Abweichungen haben können.

### 6.2.5. Eingabe-/Ausgabekomponenten

Eingabe- und Ausgabekomponenten sind sogenannte Transformationskomponenten. Das bedeutet, ihre Aufgabe ist es, Daten von einem Protokoll in ein anderes Protokoll zu übersetzen. Dies kann eine Übersetzung auf physikalischer Ebene, logischer Ebene oder auf beiden Ebenen sein. Je nach Protokoll kann dies die Komponenten verschieden komplex machen. In der Regel sind diese aber eher simpel, wie z.B. UART- oder SPI-Komponenten, die die Daten in serielle Datenströme umwandeln und die Ausgangsleitungen entsprechend der physikalischen Spezifikation treiben.

In diesem Abschnitt soll ein spezielles Protokoll behandelt werden, welches Daten in ein zeitbehaftetes Protokoll übersetzt. Dieses zeigt dadurch, wie dieses Verhalten mit Hilfe des PSM Modells dargestellt werden kann.

### PPM

Im Vergleich zu den bisher vorgestellten Protokollen existieren auch welche, die über Zeitlängen verschiedene Daten darstellen bzw. zeitgesteuert die Protokollzustände ändern. Ein Beispiel dafür ist die sogenannte *Puls-Pausen-Modulation* (PPM), das primär in Modellbaufunkfernsteuerungen zur Übertragung der Steuersignale von der Funkfernsteuerung zum Modell eingesetzt wird. Die PPM ist ähnlich zur weit verbreiteten *Puls-Weiten-Modulation* (PWM). Im Gegensatz zur PWM – bei der die Frequenz konstant gehalten wird – ist bei der PPM die Pulsbreite konstant. Das Protokoll ist in Abbildung 6.28 abgebildet. Über das Protokoll werden $n$ Kanäle übertragen, deren Daten über Pausenlängen dargestellt werden. Diese $n$ Kanäle werden durch $n+1$ Pulse mit gleicher Länge voneinander getrennt. In der Regel sind die Pulse 0,5 ms lang und die Pausen 0,5 ms bis 1,5 ms. Eine Pause von 0,5 ms gibt den niedrigsten Wert an, also die Nullstellung, und 1,5 ms den höchsten Wert, den Vollausschlag. Zur Synchronisation wird zwischen letztem und erstem Puls eine längere Pause – diese ist länger als die Pause für einen Kanal, in diesem Fall also länger als 1,5 ms – eingefügt. Häufig wird PPM auch invertiert verwendet oder mit kürzeren Pulsen (z.B. mit 0,3 ms und Pausen von 0,7 ms bis 1,7 ms). Die Gesamtlänge von Puls und Pause beträgt aber häufig 2 ms.

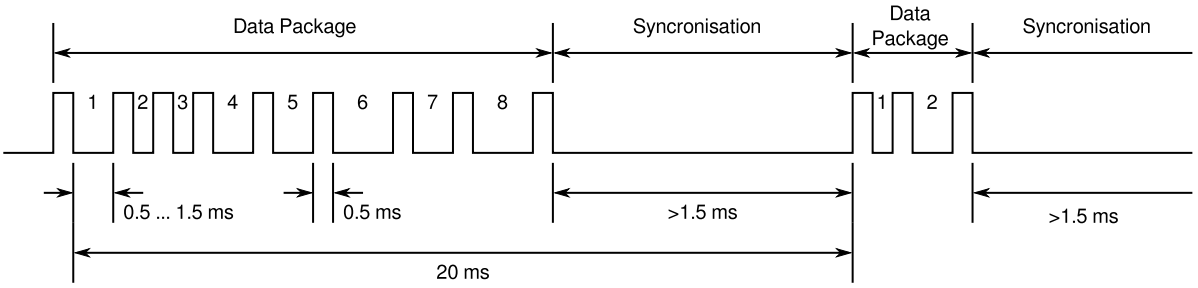



Abbildung 6.28.: Ablauf des PPM-Protokolls. Ein Datenpaket besteht aus acht Kanälen, dessen Wert über die Pausenzeit angegeben wird. Die Pausen werden durch Pulse getrennt und eine lange Pause dient zur Synchronisation.

In dem Beispiel aus Abbildung 6.28 werden acht Kanäle genutzt (dies ist die gewöhnliche Anzahl für eine Frequenz). Hier übertragen die Kanäle 1, 4, 5 und 7 eine mittlere Stellung, Kanal 2 und 3 eine Nullstellung sowie Kanal 6 und 8 einen kompletten Ausschlag.

Bei dem PPM-Protokoll müssen nicht in jedem Fall alle Kanäle übertragen werden. Wird z.B. bereits nach dem dritten Puls eine lange Synchronisationspause übertragen – wie in Abbildung 6.28 für das zweite Datenpaket gezeigt – wurden nur Daten für Kanal 1 und 2 übermittelt.

Die Synchronisationszeit kann dynamisch so angepasst werden, dass zwischen zwei Datenpaketen immer ein zeitlichen Abstand von 20 ms besteht und somit eine Frequenz von 50 Hz erreicht wird. Dies entspricht also einer Übertragung von 50 Datenpaketen pro Sekunde.

Demnach ist die kürzeste Synchronisationszeit:

$$S_{\text{min}} = (20 - 8 \cdot 1.5 + (8+1) \cdot 0,5)\,\text{ms} = 3,5\,\text{ms}$$

und die längste Synchronisationszeit:

$$S_{\text{max}} = (20 - 1 \cdot 1.5 + (1+1) \cdot 0,5)\,\text{ms} = 17,5\,\text{ms} \quad .$$

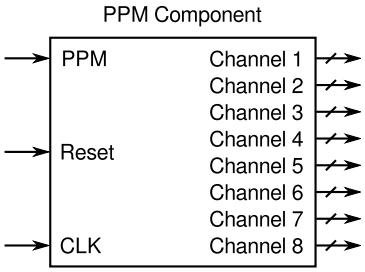



Abbildung 6.29.: PPM-Komponente zur Dekodierung eines PPM-Signals in acht Kanäle.

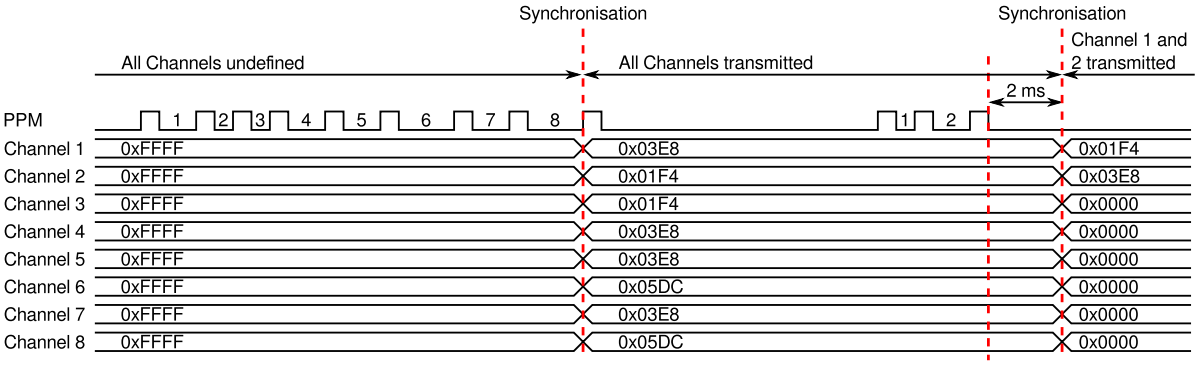



Abbildung 6.30.: Beispiel für das Ein- und Ausgabeverhalten einer PPM-Komponente. Am Anfang sind alle Kanäle undefiniert (Wert 0xFFFF). Im ersten Datenpaket werden alle Kanäle übertragen und die Ausgänge am Ende von Paket für Kanal 8 gesetzt. Beim zweiten Datenpaket werden nur zwei Kanäle übertragen und die Ausgänge nach einer zeitlichen Verzögerung von 2 ms gesetzt. Alle nicht übertragenen Kanäle bekommen den Wert 0x0000.

In Abbildung 6.29 ist eine Komponente gezeigt, die das PPM-Protokoll auf acht Kanäle dekodiert. Abbildung 6.30 zeigt das Ein- und Ausgabeverhalten der Komponente. Hierbei fällt auf, dass die Ausgaben bereitgestellt werden, sobald alle acht Kanäle übertragen wurden oder eine Synchronisationspause die Übertragung vorzeitig beendet. Die Ausgabe bei einer Synchronisationspause erfolgt hierbei aber erst nach einer bestimmten Zeit, die größer ist als die maximale Pause für einen Kanal. In diesem Fall also größer als 1,5 ms. Ein möglicher Wert ist z.B. 2 ms.

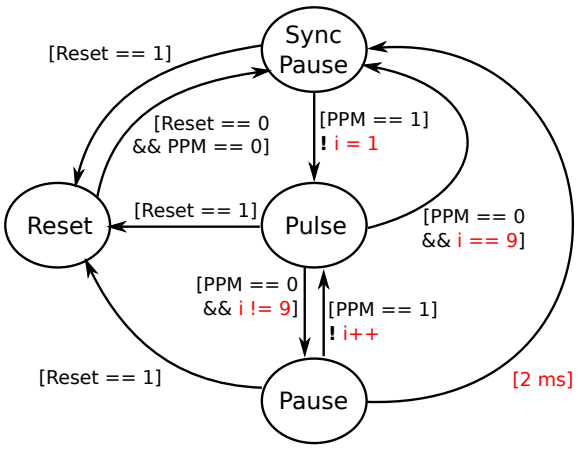



Abbildung 6.31.: Dieser Zustandsautomat beschreibt den Ablauf des PPM-Protokolls, welches für die Komponente aus Abbildung 6.29 eingesetzt wird. Die Zeiten für die PPM sind der Abbildung 6.30 entnommen. Es existiert eine Zählvariable `i`, die in den Zustandsübergängen geprüft oder gesetzt wird. Ein Zustandswechsel im Zustand `Pause` passiert automatisch 2 ms nach dem letzten Zustandswechsel.

Diese Eigenschaften des Protokolls führen zu zwei Schlussfolgerungen für das PSM Modell. Zum einen muss es erkennen, wenn alle Kanäle übertragen wurden und zum anderen, wenn durch eine längere Pause wieder eine Synchronisation durchgeführt wird. Das PPM-Protokoll kann wie in Abbildung 6.31 gezeigt als Automat dargestellt werden. Hier wird ersichtlich, dass einige Zustandsübergänge von der Zählvariablen `i` abhängig sind und andere nach einer definierten Zeit (2 ms) ausgeführt werden müssen.

Für eine Bewertung des PSM-Modells soll hier eine PPM-Komponente untersucht werden, die 16 Ausgangskanäle und eine Synchronisationszeit von 2,1 ms besitzt.

Die PrSM stellt in diesem Fall nur die grundlegenden Signalwechsel dar, da sie keine Mechanismen zur Zeitdarstellung besitzt (siehe Abschnitt 4.3). Sie erkennt demnach nur, ob ein Puls oder eine Pause anliegt oder die Komponente im `Reset`-Zustand ist, wie in Abbildung 6.32 zu sehen.

Das übrige Protokollverhalten wird in der PSM dargestellt, wie in Abbildung 6.32 gezeigt. Hier ist der Zustandsübergang von `Pause` zu `Sync Pause` zu sehen, der 2,1 ms nach dem PSM-Event `pause` ausgeführt wird, solange vorher kein PSM-Event `pulse` kam. Des Weiteren existiert die Zustandsvariable `i`, die zum Zählen der Pulse benutzt wird. Da das Zurücksetzen der Zustandsvariable neben der Anzahl von Pulsen auch abhängig von dem zeitgesteuerten Zustandsübergang ist, wird auch dieses in der PSM modelliert.

Nachfolgend wird anhand einiger Anwendungsfälle die Genauigkeit dieses Modells bewertet. Der Leistungsaufnahme-Trace für einen Anwendungsfall ist in Abbildung 6.33 gezeigt. Hier wird ersichtlich, dass es einen Unterschiede in der Leistungsaufnahme zwischen den

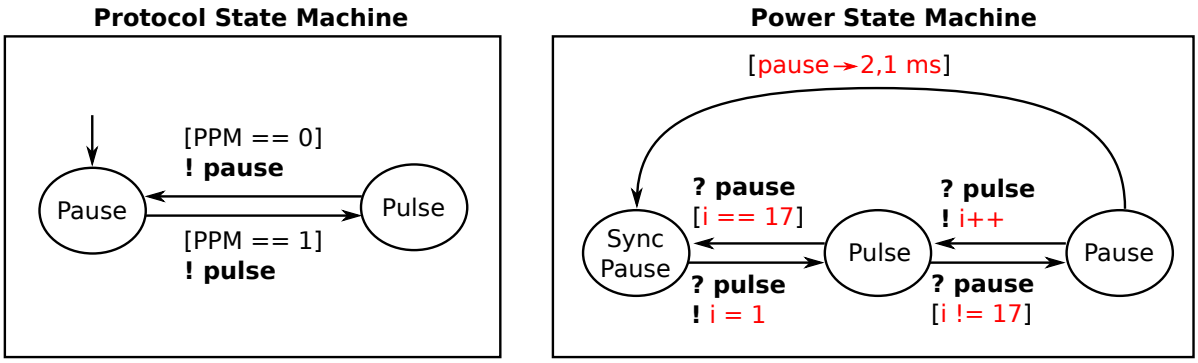



Abbildung 6.32.: Grundlegendes PSM-Modell für die Leistungsaufnahmemodellierung einer PPM-Komponente. Zeitabhängiges Verhalten im PPM-Protokoll wird durch zeitabhängige Zustandsübergänge in der PSM realisiert. Zum Zählen der Kanäle wird die lokale Zählvariable `i` genutzt. Je nach tatsächlichem Leistungsaufnahmeverhalten muss die PSM verändert werden.

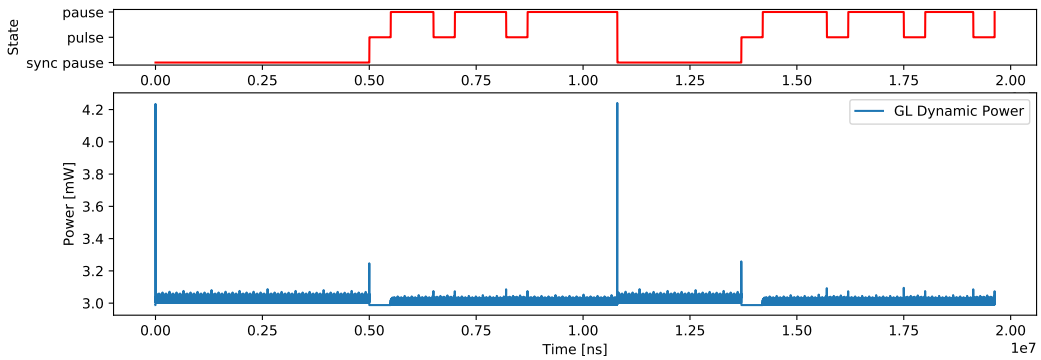



Abbildung 6.33.: Leistungsaufnahme-Trace einer PPM-Komponente mit zwei Übertragungen. Die erste Übertragung beinhaltet zwei Kanäle. Der Trace endet innerhalb der zweiten Übertragung.

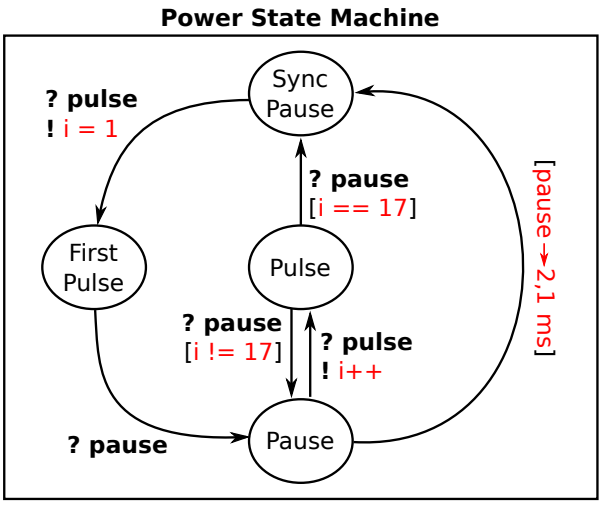



Abbildung 6.34.: PSM-Modell für PPM-Komponente mit zusätzlichem Zustand für ersten Puls, da dieser eine Charakteristik in der Leistungsaufnahme hat.

einzelnen Zuständen gibt. Für die Zustände `Pulse` und `Pause` gibt es optisch keinen Unterschied mit Ausnahme des ersten Pulses nach der Synchronisationspause (Zustand `Sync Pause`). Aus diesem Grund wird ein weiteres PSM-Modell erstellt, das diesen Operationsmodus als zusätzlichen separaten Zustand in der PSM darstellt, wie in Abbildung 6.34 dargestellt. Wie auch bei anderen Modellen ist eine Spitzenleistung beim Umschalten auf den eben beschriebenen ersten Puls als auch beim Umschalten auf den Zustand `Sync Pause` zu sehen. Der Grund liegt hier in dem Zurücksetzen der Zählregister, die die Zeiten der Pausen zählen, um diese Werte in Ausgabewerte umzuwandeln bzw. um nach 2,1 ms auf den Synchronisierungszustand `Sync Pause` zu wechseln.

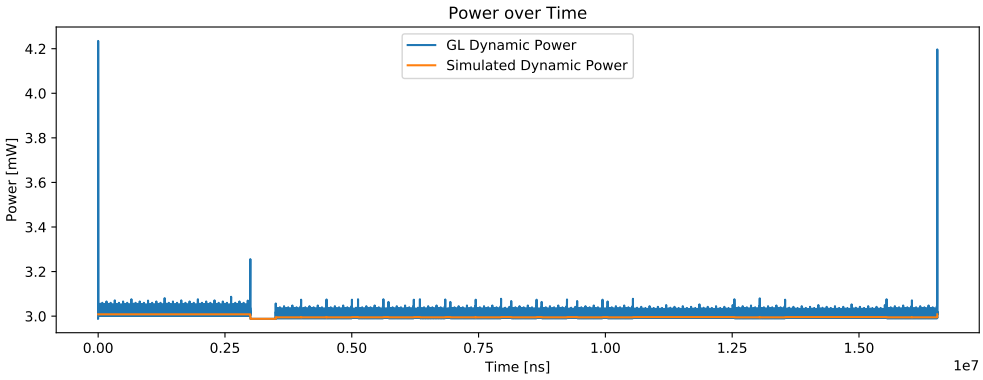



Abbildung 6.35.: Vergleich der Leistungsaufnahme-Traces von GL-Design und PSM Mean für die PPM-Komponente in Anwendungsfall 2.

Für einen weiteren Anwendungsfall ist der Vergleich in Abbildung 6.35 gezeigt. Hier zeigt sich, dass sich die Leistungsaufnahme des PSM-Modells sehr gut dem GL-Trace annähert.

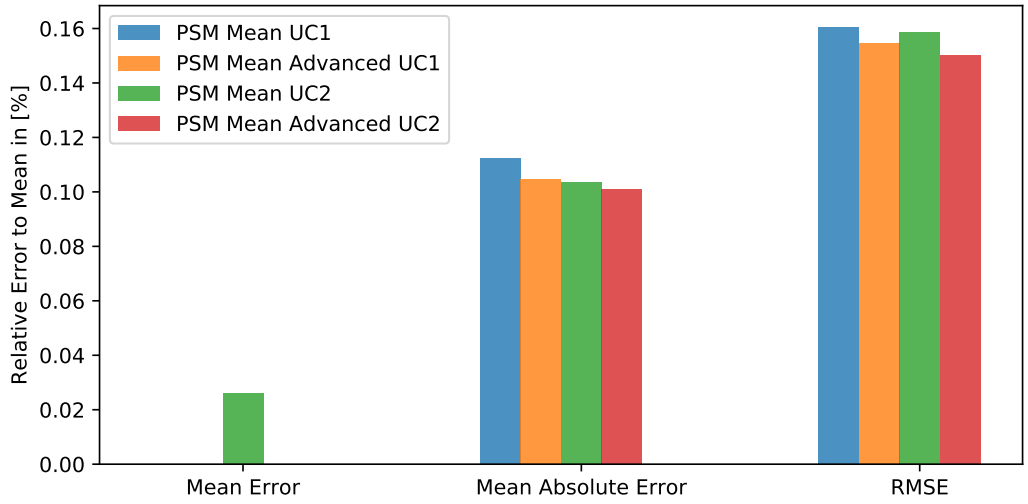



Abbildung 6.36.: Fehlergrößen von verschiedenen PSMs für mehrere Anwendungsfälle bei der PPM Komponente.

Dies unterstreichen auch die Abweichungen, wie in Abbildung 6.36 zu sehen. Der Anwendungsfall 1 ist hier der Referenz-Trace, mit dem die PSM-Modelle synthetisiert wurden. Ein Unterschied ist für den zweiten Anwendungsfall zu erkennen. Hier zeigt die PSM ohne zusätzlichen Zustand für den ersten Puls eine kleine Abweichung im Durchschnitt. Auch bei den absoluten Abweichungen und der Standardabweichung ist zu erkennen, dass der weitere Zustand den Fehler wenig verbessern kann. Dieses Beispiel zeigt, dass nicht immer die PrSM direkt als PSM übernommen werden kann und in manchen Fällen Änderungen bzw. zusätzliche Zustände nötig sind, um den Fehler größtmöglich zu senken.

Dadurch, dass sich die Leistungsaufnahme durchschnittlich in einem sehr kleinen Bereich bewegt, ist die Abweichung für alle Modelle sehr gering.

## 6.3. Betrachtung der statischen Leistungsaufnahme

In Abschnitt 2.4 wurde beschrieben, dass die statische Leistungsaufnahme im Vergleich zur dynamischen Leistungsaufnahme einen deutlich geringeren Anteil aufweist. Dadurch ist der Dynamikumfang auch entsprechend geringer und liegt mit unter 5% im Bereich des Abstraktionsfehlers des in Kapitel 5 beschriebenen Syntheseverfahrens für die dynamische Leistungsaufnahme. Dementsprechend kann die Modellierung der Dynamik in den meisten Fällen vernachlässigt werden.

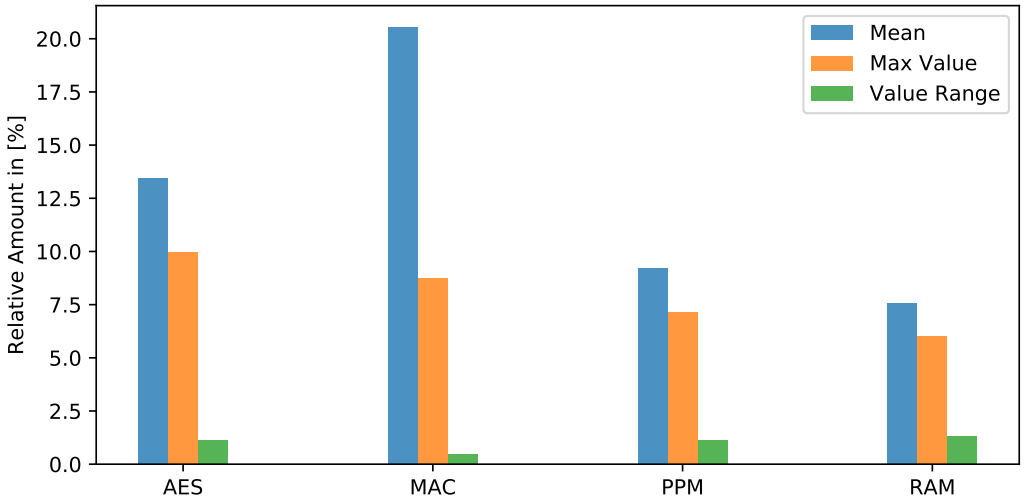



Abbildung 6.37.: Diese Grafik zeigt den relativen Vergleich in Prozent von statischer zu dynamischer Leistungsaufnahme für den Durchschnittswert und den Maximalwert eines Werteverlaufs sowie die Differenz zwischen Maximal- und Minimalwert. Letzterer zeigt den Arbeitsbereich der Werte an.

Um diese Tatsache zu belegen, wurde bei allen Versuchen neben der dynamischen Leistungsaufnahme auch die statische aufgezeichnet und verschiedene Parameter wie der maximale Anteil, der Dynamikbereich als auch der Durchschnittswert analysiert.

Die Ergebnisse sind in Abbildung 6.37 gezeigt. Für eine bessere Übersicht werden für alle Beispiele die Durchschnittswerte über alle Anwendungsbeispiele gezeigt. Die detaillierten Ergebnisse sind in Anhang A.3 nachzusehen.

Hierbei zeigen alle Versuche, dass sich der Wertebereich der statischen Leistungsaufnahme im Vergleich zum Wertebereich der dynamischen Leistungsaufnahme im Bereich von deutlich unter 2% bewegt. Dennoch liegt der Mittelwert höher, zwischen 5% und 20%. Das bedeutet, dass die statischen Leistungsaufnahme eine deutlich geringe Dynamik als die dynamische Leistungsaufnahme besitzt.

Des Weiteren liegt der Anteil (gemessen an dem Maximalwert) bei nur etwa 5 bis 10% der dynamischen Leistungsaufnahme und zeigt damit, dass auch die maximalen Ausschläge bei der statischen Leistungsaufnahme deutlich geringer sind. Die Spitzen bei der dynamischen Leistungsaufnahme entstehen, wenn viele Gatter gleichzeitig ihren Zustand ändern. Die statische Leistungsaufnahme ergibt sich im Unterschied dazu nur durch die Zustände der Logikgatter und nicht durch deren Änderung. Dadurch entstehen in der Regel keine verhältnismäßig hohe Spitzen im Verlauf der statischen Leistungsaufnahme.

Damit ist dieser Wert auf einem Niveau mit der Ungenauigkeit der Abstraktion der dynami-

schen Leistungsaufnahme und kann dementsprechend mit einem konstanten Wert belegt werden.

Der Speicher ist eine Komponente, die mehr Elemente beinhaltet, die Inhalte abspeichern, als welche die Operationen durchführen. Diese Elemente zeigen eine höhere Varianz der statischen Leistungsaufnahme. Die statische Leistungsaufnahme unterscheidet sich dabei stark je nach Speicherzustand der einzelnen Elemente. Der Durchschnittswert beträgt für die meisten Fälle unter 10% der dynamischen Leistungsaufnahme, welches in etwa dem Abstraktionsfehler der PSM entspricht und daher auch vernachlässigt werden kann.

Bei dem PPM lässt sich die erhöhte Varianz dadurch erklären, dass die Varianz der dynamischen Leistungsaufnahme sehr gering ist. Durch den inneren Aufbau, der lediglich aus einzelnen Zählern besteht, ist die HD sehr niedrig (im Durchschnitt bei 2). Da auch hier wieder datenspeichernde Elemente zum Einsatz kommen, variiert die statische Leistungsaufnahme jedoch stärker.

Als Konsequenz lässt sich ableiten, dass eine erhöhte relative Varianz der statischen Leistungsaufnahme bei Komponenten mit vielen speichernden Elementen bzw. Komponenten mit einer geringen Varianz der dynamischen Leistungsaufnahme zu sehen ist, diese jedoch auch hier vernachlässigt werden kann, da zum einen die statischen Leistungsaufnahme nur einen geringen Anteil an der gesamten Leistungsaufnahme hat und zum anderen der Bereich, in dem die statische Leistungsaufnahme variiert, deutlich kleiner ist als bei der dynamischen Leistungsaufnahme.

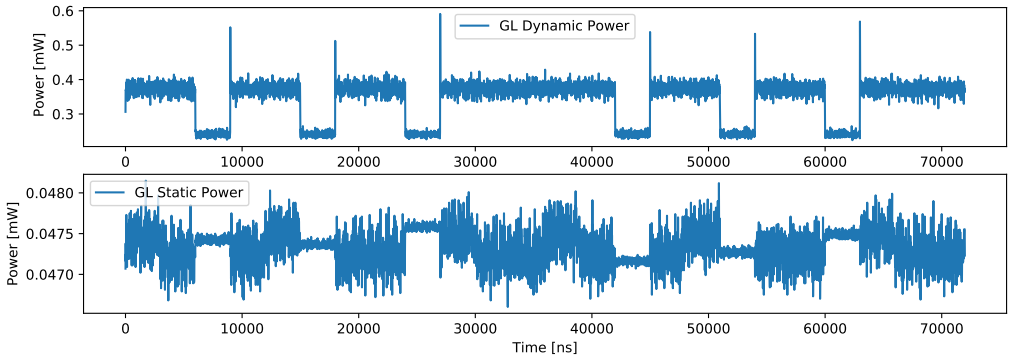



Abbildung 6.38.: Trace der statischen GL Leistungsaufnahme im Vergleich zur dynamischen GL Leistungsaufnahme.

Diese Ergebnisse wurden mit einer 65 nm Bibliothek generiert. Der Stand der Technik sind zum Zeitpunkt dieser Arbeit Technologiegrößen von 5 nm, die in Zukunft weiter sinken werden. Hierdurch werden die Isolationsschichten in den Transistoren immer dünner und dadurch die Leckströme größer. Entsprechend steigt auch die statische Leistungsaufnahme in Bezug auf die dynamische Leistungsaufnahme und gewinnt dadurch immer mehr an Bedeutung. Der Einfluss könnte dabei so groß werden, dass eine Modellierung der statischen

Leistungsaufnahme als konstante Größe einen zu großen Fehler verursachen würde. Die statische Leistungsaufnahme muss hier auch als dynamische Größe modelliert werden. Dass dies unweit schwerer ist, zeigt ein Blick auf Abbildung 6.38. Die statische Leistungsaufnahme folgt nicht dem Verhalten der dynamischen Leistungsaufnahme wie Abschnitt 2.2.1 erläutert. Der Wert der statischen Leistungsaufnahme ergibt sich hier durch die einzelnen Zustände in den Logikgattern. Diese lässt sich nicht trivial aus den Ein- und Ausgaben ableiten oder in geeigneter Weise abstrahieren.

## 6.4. Performanzzuwachs des Modells

Eines der Ziele dieser Arbeit ist die Beschleunigung der Leistungsaufnahmesimulationen von Hardwarekomponenten in SoCs. Dies wird mit dem PSM-Modell durch die Abstraktion der GL-Leistungsaufnahme-Daten umgesetzt.

Tabelle 6.1.: Simulations- und Berechnungszeiten der Leistungsaufnahme auf GL und für die PSM-Modelle.

| Design | Anwendungs-fall | Simulations-zeit GL [s] | Berechnungszeit Leistungsaufnahme [s] | Simulierte Zeit [ms] | Simulations-zeit PSM [s] | Performanzgewinn PSM zu GL | Simulationszeit PSM ohne Dateieingabe und -ausgabe [s] |
|---|---|---|---|---|---|---|---|
| AES | 0 | 1260,59 | 1189,06 | 2,905650 | 4,06304 | 293x | 0,20418 |
| | 1 | 234,56 | 223,11 | 0,511830 | 0,72623 | 307x | 0,03125 |
| | 2 | 885,52 | 1696,68 | 2,942570 | 4,46561 | 380x | 0,20883 |
| MAC | 0 | 19,59 | 30,46 | 0,071988 | 0,19896 | 153x | 0,09598 |
| | 1 | 12,92 | 18,44 | 0,038988 | 0,10897 | 196x | 0,04736 |
| PPM | 0 | 4661,62 | 16135,31 | 19,625190 | 61,53349 | 262x | 1,08622 |
| | 1 | 3782,08 | 13415,08 | 16,543670 | 51,69428 | 260x | 0,92512 |
| RAM | 0 | 62,46 | 111,76 | 0,021670 | 0,05188 | 2154x | 0,02119 |
| | 1 | 28,74 | 63,91 | 0,007690 | 0,02094 | 3052x | 0,00759 |
| | 2 | 26,62 | 58,90 | 0,007690 | 0,02193 | 2686x | 0,00722 |

In den vorherigen Kapiteln wurden einige Komponenten vorgestellt und gezeigt, wie man ein PSM-Modell erstellt, welches das erforderliche Fehlermaß einhält. In Tabelle 6.1 wird gezeigt, welche Zeiten zum Erfassen der GL-Daten benötigt werden sowie die der entsprechenden PSM-Simulationen. Auf GL wird dabei unterschieden zwischen der Simulation zur

Aufnahme der Aktivitätsdaten und der eigentlichen Berechnung der Leistungsaufnahme aus diesen Daten. Wie am Anfang dieses Kapitels beschrieben, wird die PSM simuliert, indem dieser die GL-Eingangsdaten zugespielt werden. Dafür werden sie am Anfang der Simulation aus einer Datei eingelesen. Zusätzlich werden am Ende der Simulation die Leistungsaufnahme-Traces in Textdateien gespeichert. Diese Vorgänge passieren in ähnlicher Weise auch auf GL. Daher werde die entsprechenden Zeiten dieser Vorgänge mit in den Vergleich einbezogen. In der letzten Tabellenspalte werden die reinen Simulationszeiten aufgezeigt.

Hier wird ersichtlich, dass teilweise sehr hohe Zeitaufwände nötig sind, um die entsprechenden Daten auf GL zu erfassen. Speziell für die PPM-Komponente, die ein zeitabhängiges Protokoll umsetzt, sind sehr lange Simulationszeiten notwendig, um alle relevanten Zustände und Zustandsübergänge der Komponente zu erfassen. Dies zeigt, dass auch bei relativ einfachen Komponenten hohe Simulationszeiten für die Synthese nötig werden können. Des Weiteren wird anhand der simulierten Zeit deutlich, dass diese in den meisten Fällen lediglich wenigen Millisekunden beträgt. Für Systemsimulationen werden in der Regel Anwendungsfälle simuliert, die über mehrere Sekunden oder Minuten Systemlaufzeit das Verhalten nachbilden. Anhand dieser Zeiten wird ersichtlich, dass dies mit GL-Modellen nicht effizient umsetzbar ist.

Der Geschwindigkeitsgewinn vom PSM-Modell ist gering. Dabei fällt aber auf, dass die Dateioperationen einen Hauptteil der Simulationszeit ausmachen. Dies wird dadurch hervorgerufen, dass GL-Eingangs- und Ausgangsdaten für die Stimulation der PrSM benutzt werden. Da auf GL die Zeiten für das Einlesen der Daten nicht herausgerechnet werden können, müssen diese bei der PSM mit einbezogen werden. Zieht man die bei der PSM gemessenen Zeiten ab, wird offensichtlich, dass die reine Simulationszeit nur einen Bruchteil ausmacht und der Geschwindigkeitsgewinn erheblich ist. Bei einer entsprechenden ESL-Simulationen sind die Eingabe- und Ausgabedaten in abstrahierter Form vorhanden, was zu einer deutlich geringeren Datenmenge führt. Des Weiteren werden diese nicht über eine Datei zwischengespeichert, sondern von dem funktionalen Modell direkt an die PrSM weitergegeben. Damit entfallen die Zeiten für das Speichern bzw. Lesen der Daten. Dieses zieht zusätzlich eine deutlich beschleunigte Simulation nach sich.

## 6.5. Qualität des Charakterisierungsprozesses

In den in diesem Kapitel gezeigten Evaluationskomponenten wurde gezeigt, wie eine beispielhafte Synthese aussieht. Dabei wurden die Vorgehensweisen und Erkenntnisse aus Kapitel 5 angewendet und anhand der Komponenten erläutert. Es hat sich gezeigt, dass die beschriebenen Vorgehensweisen ein strukturierter und zielorientierter Prozess sind, der auf einfache und schnelle Weise Modelle mit einem niedrigen Fehler erzeugt. Diese können in einer beliebigen Sprache implementiert und mit einer erheblichen Performanzsteigerung ausgeführt werden.

Der Syntheseprozess betrachtet alle Fähigkeiten des PSM-Modells wie z.B. die datenabhängige Leistungsaufnahme oder zeitgesteuerte Zustandsübergänge. Des Weiteren beschreibt der Prozess verschiedene Lösungsmöglichkeiten, um die bestmöglichsten Modelle mit der erforderlichen Genauigkeit zu erzeugen. Zusätzlich werden verschiedene Methoden erklärt, die eine genauere Analyse der Leistungsaufnahme-Traces erlauben, wie z.B. das Finden von CPs.

Als Hilfemaßnahme wird ein Algorithmus vorgestellt, der eine automatische Synthese von einfachen PSMs – jene ohne datenabhängige Leistungsaufnahme oder zeitgesteuerte Zustandsübergänge – erlaubt, um so schnell für komplexe funktionale Modelle ein PSM-Modell zu generieren.

## 6.6. Zusammenfassung

In diesem Kapitel wurden das in Kapitel 4 beschriebene Modell und der in Kapitel 5 beschriebene Syntheseprozess an verschiedenen Evaluationskomponenten evaluiert. Dabei konnte deutlich gemacht werden, dass durch die beschriebene Synthese Modelle erzeugt werden können, die einen guten Kompromiss zwischen Genauigkeit und Simulationsperformanz eingehen. Es wurde gezeigt, dass eine dynamische Betrachtung der statischen Leistungsaufnahme nicht nötig ist und diese als konstant angenommen werden kann. Lediglich die Beeinflussung des Leckwiderstands durch die Temperatur muss mit einbezogen werden, sobald ein Temperaturmodell integriert wird.

Des Weiteren wurden die wissenschaftlichen Fragestellungen aus Abschnitt 1.1 beantwortet und bewertet.

1. Abdeckung der Leistungsaufnahmezustände: Ein Vergleich der Werte aus dem Modell mit den Werten von GL zeigt nur eine kleine Abweichung. Da die Leistungsaufnahme lediglich von den Schnittstellen abgeleitet wurde, belegt dies eine starke Kopplung zwischen funktionalem Verhalten bzw. Verhalten auf den Kommunikationsschnittstellen und der Leistungsaufnahme.

2. Untersuchung von verschiedenen Komponentenklassen: In der Evaluation wurden mehrere Komponenten aus relevanten Komponentenklassen untersucht. Für alle Komponenten konnte ein Modell erstellt werden, welches eine hinreichend genaue Abbildung der Leistungsaufnahme ermöglicht. Damit ist belegt, dass ein generisches Modell für verschiedene Klassen von passiven, aber nicht aktiven Komponenten eingesetzt werden kann.

3. Abhängigkeit zwischen Leistungsaufnahme und Komponenteninteraktion: Es konnte gezeigt werden, dass eine Leistungsaufnahmeänderung in den meisten Fällen mit einer Komponenteninteraktion korreliert. Am Beispiel der PPM Komponente wurde deutlich, dass es funktionale Zustandsübergänge gibt, die nicht durch eine Kommunikation auf den äußeren Schnittstellen sichtbar sind. Es konnte gezeigt werden, dass

diese häufig einem deterministischen Verhalten folgen und durch zeitlich verzögerte Zustandsübergänge im Leistungsaufnahmemodell dargestellt werden können.

4. Einfluss der Datenabhängigkeit auf die Leistungsaufnahme: Anhand der Speicherkomponente konnte gezeigt werden, dass die datenabhängige Leistungsaufnahme einen großen Anteil einnehmen kann. Des Weiteren wurde gezeigt, dass die entwickelte Methodik eine zuverlässige Möglichkeit bietet, diese datenabhängige Leistungsaufnahme hinreichend genau zu modellieren.

# KAPITEL 7

# Fazit

In den vorherigen Kapiteln wurde in das Thema dieser Arbeit eingeführt, die Grundlagen und angrenzenden Arbeiten beschrieben, die Methodik und Modellsynthese erläutert als auch die Ergebnisse von gezielten Untersuchungen gezeigt. Dieses Kapitel fasst die Arbeit zusammen und gibt einen Ausblick auf zukünftige Arbeiten.

## 7.1. Zusammenfassung

Die Leistungsaufnahme in eingebetteten System wird heute ein immer wichtigerer Faktor, da eingebettete Systeme oft nur eine begrenzte Menge an Energie zur Verfügung stellen können, was zu geringen Laufzeiten führt. Des Weiteren heizt sich ein System durch eine höhere Leistungsaufnahme stärker auf, was zu kürzen Lebenszeiten oder zur Zerstörung des System führen kann. Dieser Aspekt kann am besten adressiert werden, in dem er so früh wie möglich im Designprozess betrachtet und optimiert wird. Stand der Technik ist es, das System erstmalig auf der Abstraktionsebene ESL aufzubauen und zu testen. Daher hat sich diese Arbeit das Ziel gesetzt, auf dieser Ebene die Möglichkeit zu erforschen, bestehende funktionale, ausführbare Modelle mit der Modellierung von Leistungsaufnahme zu erweitern. Die Lösung sollte eine möglichst genaue Abbildung der tatsächliche Leistungsaufnahme ermöglichen und dabei die Zusatzaufwand so gering wie möglich halten.

In dieser Arbeit wurde eine Methodik zur Erstellung von ausführbaren Modellen zur Simulation der Leistungsaufnahme auf ESL vorgestellt, welches an virtuelle Hardware-Plattformen gekoppelt werden kann. Dies ermöglicht die Aufzeichnung der Leistungsaufnahme von Systemkomponenten über die Zeit in Abhängigkeit von einem konkreten Anwendungsfall. Da mehrheitlich sogenannte IP-Komponenten eingesetzt werden, bei denen der innere Aufbau unbekannt ist, konzentriert sich die in dieser Arbeit entwickelte Methodik auf die Herausforderungen, die diese Komponenten mit sich bringen wie der verborgene innere Zustand und an den Schnittstellen nicht sichtbare Zustandsänderungen. Des Weiteren wird das zustandsbasierte Modell mit der Fähigkeit erweitert, Datenabhängigkeiten in der Leistungsaufnahme abzubilden, die einen signifikanten Einfluss haben können. Da diese Modelle in der Regel nicht eigenständig eingesetzt werden, sind sie unabhängig von Taktfrequenz, Versorgungsspannung und Temperatur und ermöglichen dadurch eine nahtlose Integration.

Für das entwickelte Modell beschreibt diese Arbeit ein Verfahren zur manuellen und teilautomatisierten Synthese des Modells. Dabei werden alle Fähigkeiten wie die Modellierung von Datenabhängigkeiten in der Leistungsaufnahme oder zeitlich verzögerte Zustandsübergänge betrachtet, damit das resultierende Modell den besten Kompromiss aus Simulationsperformanz und Genauigkeit erzielt.

Die Ergebnisse in dieser Arbeit zeigen, dass die Methodik in der Lage ist Modelle zu erstellen, die

1. alle in der Simulation aufgetretenen Leistungsaufnahmezustände abdecken,
2. verschiedene Klassen von passiven Komponenten unterstützen,
3. aus den Komponenteninteraktionen hinreichend genau die Leistungsaufnahme abbilden und
4. mögliche Datenabhängigkeiten in der Leistungsaufnahme darstellen.

Damit gibt diese Arbeit Antworten auf alle wissenschaftlichen Fragestellungen aus Abschnitt 1.1. Dabei erzeugt das Modell bei einem minimalen rechnerischen Mehraufwand ein sehr genaues Ergebnis in Bezug auf die auf GL erfassten Leistungsaufnahmedaten.

## 7.2. Ausblick

Beim Vergleich mit anderen Arbeiten ist besonders aufgefallen, dass die in dieser Arbeit entwickelte Methodik nicht gut auf aktive Komponenten angewendet werden kann (Abschnitt 3.4) und auch die Fähigkeit fehlt, die Leistungsaufnahme von Signalleitungen zu modellieren(Abschnitt 3.5).

In Zukunft könnte untersucht werden, wie die hier vorgestellte Methodik so erweitert wird, dass sie den Einsatz für aktive Komponenten ermöglicht. Dies kann ähnlich wie bei Onnebrink et al. umgesetzt werden, indem die Schnittstellen von relevanten Peripheriekomponenten beobachtet werden, um den inneren Zustand abzuleiten [54] oder durch Trigger direkt aus dem Modell, sobald *White-Box*- Komponenten verwendet werden.

Der PSM-Ansatz dieser Arbeit ist bereits in der Lage, vielfältig datenabhängige Leistungsaufnahme darzustellen. Es ist denkbar, dass die Leistungsaufnahme von Signal- und Busleitungen auch über ein entsprechendes PSM-Modell dargestellt werden kann. Der entscheidende Unterschied hierbei ist die Synthese des Modells, bei der die zugrundelegenden Werte in der Regel nicht aus einer GL-Simulation entnommen werden können, da hierfür das gesamte System auf GL synthetisiert und simuliert werden müsste, was unverhältnismäßig viel Zeit in Anspruch nehmen würde. Aus diesem Grund würde eine statische Abschätzung der Leitungslängen auf Basis der Chipgröße stattfinden und mit den Signalstatistiken aus der Buskommunikation die Berechnung der Leistungsaufnahme ermöglichen, wie dies in Abschnitt 3.5 beschrieben wurde.

# ANHANG A

# Anhang

## A.1. PSM XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema elementFormDefault="qualified"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="psm">
    <xsd:sequence>
      <xsd:element name="states">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="state" type="psm_state"
                  maxOccurs="unbounded" minOccurs="1"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="transitions">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="transition" type="psm_transition"
                  maxOccurs="unbounded" minOccurs="1"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="timed_transitions" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="timed_transition" type="timed_transition"
                  minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="init_state">
        <xsd:complexType>
          <xsd:attribute name="ref" type="xsd:string" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="prsm">
    <xsd:sequence>
      <xsd:element name="states">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="state" type="prsm_state" maxOccurs="unbounded"
              minOccurs="1"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
```

```
      <xsd:element name="transitions">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="transition" type="prsm_transition"
                  maxOccurs="unbounded" minOccurs="1"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="init_state">
        <xsd:complexType>
          <xsd:attribute name="ref" type="xsd:string" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="prsm_state">
    <xsd:attribute name="id" type="xsd:ID" use="prohibited"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="prsm_transition">
    <xsd:sequence>
      <xsd:element name="updates" maxOccurs="1" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="update" type="update" maxOccurs="unbounded"
                  minOccurs="0" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="guards" minOccurs="1" maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="atomic_guard" type="atomic_guard"
                  minOccurs="1" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="from_state_ref" type="xsd:string" use="required"/>
    <xsd:attribute name="to_state_ref" type="xsd:string" use="required"/>
    <xsd:attribute name="event_ref" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="update">
    <xsd:sequence>
      <xsd:element name="param" type="parameter" maxOccurs="unbounded"
          minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="type" type="xsd:string" use="required"/>
    <xsd:attribute name="state_var_ref" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="psm_state">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="output" type="output"/>
        <xsd:element name="capacitance" type="phys_value"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID" use="prohibited"/>
    <xsd:attribute name="name" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="timed_transition">
    <xsd:attribute name="delay_unit" type="xsd:string" use="required"/>
    <xsd:attribute name="delay" type="xsd:string" use="required"/>
    <xsd:attribute name="from_state_ref" type="xsd:string" use="required"/>
    <xsd:attribute name="to_state_ref" type="xsd:string" use="required"/>
    <xsd:attribute name="event_ref" type="xsd:string" use="required"/>
```

```
</xsd:complexType>
<xsd:complexType name="psm_transition">
  <xsd:sequence>
    <xsd:element name="updates" maxOccurs="1" minOccurs="0">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="update" type="update" maxOccurs="unbounded"
                minOccurs="0" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="guards" minOccurs="0" maxOccurs="1">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="atomic_guard" type="atomic_guard"
                minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="from_state_ref" type="xsd:string" use="required"/>
  <xsd:attribute name="to_state_ref" type="xsd:string" use="required"/>
  <xsd:attribute name="event_ref" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="atomic_guard">
  <xsd:sequence>
    <xsd:element name="param" type="parameter" maxOccurs="unbounded"
          minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="psm_event">
  <xsd:attribute name="id" type="xsd:ID" use="prohibited"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="state_var">
  <xsd:attribute name="id" type="xsd:ID" use="prohibited"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="value_type" type="xsd:string" use="required"/>
  <xsd:attribute name="init_value" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="parameter">
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="value" type="xsd:string" use="required"/>
  <xsd:attribute name="value_type" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="output">
  <xsd:sequence>
    <xsd:element name="param" type="parameter" maxOccurs="unbounded"
          minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="psm_block">
  <xsd:sequence>
    <xsd:element name="psm_events">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="psm_event" type="psm_event"
                maxOccurs="unbounded" minOccurs="1"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="state_vars" maxOccurs="1" minOccurs="0">
      <xsd:complexType>
        <xsd:sequence>
```

```
            <xsd:element name="state_variable" type="state_var"
                  maxOccurs="unbounded" minOccurs="0"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="psm" type="psm"/>
      <xsd:element name="prsm" type="prsm"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="system" type="system">
  </xsd:element>
  <xsd:complexType name="module">
    <xsd:sequence>
      <xsd:element name="psm_block" type="psm_block" maxOccurs="1" minOccurs="0"/>
      <xsd:element name="module" type="module" maxOccurs="unbounded" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID" use="prohibited"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="system">
    <xsd:sequence>
      <xsd:element name="module" type="module" maxOccurs="unbounded" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID" use="prohibited"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="phys_value">
    <xsd:attribute name="value" type="xsd:string" use="required"/>
    <xsd:attribute name="unit" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="timeout">
    <xsd:sequence>
      <xsd:element name="time" type="phys_value"/>
    </xsd:sequence>
    <xsd:attribute name="to_state_ref" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:schema>
```

## A.2. GL-Syntheseskript

GL-Synthesescript am Beispiel des Camellia-Designs.

```
#########################################################################
#       libraries
#########################################################################
set SYNOPSYS /opt/sw/eda/synopsys/syn_vK-2015.06-SP1
set search_path [list . [format "%s%s" $SYNOPSYS {/libraries/syn }] \
                        [format "%s%s" $SYNOPSYS {/dw/sim_ver/ }] \
                        ../src/rtl ../../../tech ./WORK/]
set target_library [list "../../../tech/tcbn65gpluslt.db"]
set symbol_library [list ""]
set synthetic_library [list standard.sldb dw_foundation.sldb]
set link_library [concat $target_library $synthetic_library]
#########################################################################
#       link design
#########################################################################
analyze -format verilog camellia.v
elaborate camellia
set_driving_cell -lib_cell INVD0 -library tcbn65gpluslt [all_inputs]
link
create_clock [get_ports CLK]  -period 10  -waveform     {0 5}
check_design
```

```
########################################################################
#       compile
########################################################################
compile -map_effort high
########################################################################
#       save resulting design
########################################################################
write_sdc camellia.sdc
write_file -format ddc -hierarchy -output camellia.ddc
write -format verilog -hierarchy -output camellia.vg
write_parasitics -output camellia.spf
exit
```

## A.3. Vergleich statische zu dynamische Leistungsaufnahme

Tabelle A.1.: In dieser Tabelle werden der Durchschnittswert, der Maximalwert sowie der Wertebereich der statischen Leistungsaufnahme absolut und prozentual im Vergleich zur dynamischen Leistungsaufnahme gezeigt. Diese sind sortiert nach Komponente und Anwendungsfall.

| Komponente | Durchschnittswert [W] | % | Maximalwert [W] | % | Wertebereich [W] | % |
|---|---|---|---|---|---|---|
| AES | 1,48e-4 | 13,75 | 1,51e-4 | 9,77 | 1,11e-5 | 1,05 |
| | 1,49e-4 | 13,72 | 1,51e-4 | 10,28 | 1,06e-5 | 1,08 |
| | 1,48e-4 | 12,81 | 1,51e-4 | 9,79 | 1,26e-5 | 1,20 |
| MAC | 4,73e-5 | 13,87 | 4,83e-5 | 8,17 | 2,28e-6 | 0,42 |
| | 4,72e-5 | 13,32 | 4,82e-5 | 9,92 | 2,08e-6 | 0,49 |
| PPM | 2,84e-4 | 9,45 | 3,05e-4 | 7,19 | 4,37e-5 | 1,42 |
| | 2,67e-4 | 8,92 | 3,00e-4 | 7,07 | 3,79e-5 | 1,24 |
| RAM | 9,89e-4 | 7,94 | 1,03e-3 | 6,60 | 1,38e-4 | 1,45 |
| | 9,84e-4 | 7,54 | 1,02e-3 | 4,84 | 1,36e-4 | 0,91 |
| | 9,85e-4 | 7,54 | 1,02e-3 | 5,06 | 1,36e-4 | 0,97 |

# ANHANG B

# Verzeichnisse

## Abkürzungsverzeichnis

**AES** Advanced Encryption Standard: ist ein verbreitet eingesetzter standardisierter Verschlüsselungsalgorythmus.

**ASIC** Application-Specific Integrated Circuit: ist eine integrierte Schaltung, die für einen bestimmten Anwedungszeck entwickelt wurde. Im Gegensatz dazu stehen integrierte Schaltungen für einen allgemeinen Anwendungszweck.

**CMOS** Complementary Metal-Oxide Semiconductor: ist eine Technologie für integrierte Schaltungen. Diese benutzt zwei in Reihe geschaltete MOSFETs (Metall Oxid Semiconductor Field Effect Transistor). Diese schalten jeweils im wechselseitigen Ausschluss (ist einer leitend, sperrt der andere)

**CP** Change Point: beschreibt einen Zeitpunkt in eine Leistungsaufnahme-Trace bei dem sich die Charakteristik (Durchschnittswert und/oder Varianz) ändert.

**DMA** Direct Memory Access: beschreibt den direkten Datenzugriff anstatt Daten über eine Kommunikationsschnittstelle zu übertragen. Dies wird häufig in Simulationen auf hoher Abstraktionsebene eingesetzt, um die Simulationsgeschwindigkeit zu erhöhen.

**DVFS** Dynamic Voltage and Frequency Scaling: beschreibt die gleichzeitige Skalierung von Spannung und Frequenz zur Senkung der Energieaufnahme.

**ECA** Event-Clock Automaton: Zustandsautomat mit jeweils einer Uhr für jedes Eingabesymbol zur Darstellung von zeitlichen Übergängen.

**EPA** Event-Predicting Automaton: Ein Zustandsautomat, der zukünftige Events vorhersagt.

**ERA** Event-Recording Automaton: Ein Zustandsautomat, der vergangene Events speichert und auf diese reagiert.

**EFSM** Extended Finite State Machine: Ein mit Zustandsvariablen erweiterter Zustandsautomat.

**ESL** Electronic System Level: bezeichnet eine höhere Abstraktionsebene, die für das Design und die Verifaktion von gesamten Systemen eingsetzt wird. Dabei wird das Verhalten der Systeme mit Hochsprachen wie *C++* oder grafischen Werkzeugen wie Simulink beschrieben

**FD-SOI** Fully Depleted Silicon On Insulator

**FPGA** Field Programmable Gate Array: ist ein integrierter Schaltkreis (IC), dessen logische Schlatung programmiert werden kann.

**GL** Gate Level: bezeichnet die Abstraktionsebene für integrierte Schaltungen, bei der das Design nur noch aus Gattern – z.B. Und-Gatter oder Register – aufgebaut ist.

**HD** Hamming Distance: beschreibt die Anzahl der unterschiedlichen Zeichen von zwei gleichgroßen Worten

**IC** Integrated Circuit: beschreibt eine elektrische Schaltung, die auf einer Halbleiterplatine aufgebraucht wurde und gewöhnlich in ein Gehäuse eingebracht wird, welches die Kontaktierung vereinfacht und die Platine schützt. Die Schaltung besteht in den meisten Fällen aus Halbleiterbauelementen wie Transistoren und weiteren elektronischen Komponenten.

**IP** Intellectual Property: bezeichnet geistiges Eigentum. Hier im Speziellen geschütztes geistiges Eigentum in Bezug auf Schlatungsentwurf mit Sicht auf Copyright Rechten und Patent Rechten.

**ISS** Instruction Set Simulator: ist ein Simulator, der auf Instruktionsebene das Verhalten eines Prozessors simuliert.

**MOSFET** Metal Oxid Semiconductor Field Effect Transistor: ist ein Transistor, dessen Durchlass über ein elektisches Feld gesteuert wird.

**PPM** Puls-Pausen-Modulation: ist ein Protokoll, welches die Signalwerte über verschieden lange Pausen darstellt. Dies wird häufig für die Übertragung von Fernsteuerungssignalen verwendet.

**PWM** Puls-Weiten-Modulation: ist ein Protokoll, welches die Signalwerte über die Pulsweite darstellt. Dies wird häufig für die Aktorikansteuerung verwendet.

**PrSM** Protocol State Machine: ist ein Zustandsautomat, der das Protokollverhalten einer Komponente modelliert. Als Eingabe dienen die Kommunkationsdaten der entsprechenden Komponente.

**PSM** Power State Machine: ein Zustandsautomat, bei dem jeder Zustand einer Leistungsaufnahme entspricht. Die Transitionen können optional einer Energieaufnahme annotiert sein.

**RDL** Register Defenition Language: Sprache zur Beschreibung von Registerschnittstellen.

**RMSE** Root Mean Square Error: ist die sogenannte Standardabweichung, die sich aus der Wurzel der aufsummierten quadratischen Abweichungen berechnet.

**RTL** Register Transfer Level: beschreibt eine Abstraktionsebene beim digitalen Schaltungsentwurf, bei dem eine synchrone Digitalschaltung aus Registern und den logischen Operationen, die diese Register verbinden, modelliert wird.

**SoC** System on a Chip: beschreibt ein komplettes System inkluse Prozessor, Speicher, Busse, etc., das auf nur einem Chip untergebracht ist.

**TLM** Transaction Level Modelling: auf dieser Ebene findet die Kommunikation allein über abstrakte Transaktionen statt.

**VCD** Value Change Dump: Dateiformat für die Darstellung von Wertverläufen über die Zeit.

**XML** Extensible Markup Language: Standardisiertes Format zur Beschreibung von Informationsstrukturen.

# ANHANG B

# Abbildungsverzeichnis

# ANHANG B

# Tabellenverzeichnis

# ANHANG B

# Literaturverzeichnis


[1] Accellera. SystemRDL 2.0 Register Description Language, January 2018.

[2] Sumit Ahuja, Deepak A. Mathaikutty, Gaurav Singh, Joe Stetzer, Sandepp K. Shukla, and Ajit Dingankar. Power estimation methodology for a high-level synthesis framework. In *Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design*, pages 541–546, 2009.

[3] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211(1):253–273, 1999.

[4] ARM. AMBA 3 APB Protocol v1.0, August 2004.

[5] Nikhil Bansal, Kanishka Lahiri, Anand Raghunathan, and Srimat T. Chakradhar. Power monitors: a framework for system-level power estimation using heterogeneous power models. In *18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*, pages 579–585, January 2005.

[6] Luca Benini, Robin Hodgson, and Polly Siegel. System-level power estimation and optimization. In *International Symposium on Low Power Electronics and Design*, ISLPED'98, pages 173–178, Monterey, CA, USA, August 1998. ACM.

[7] Reinaldo A. Bergamaschi and Yunjian W. Jiang. State-based power analysis for systems-on-chip. In *Proceedings of the 40th Annual Design Automation Conference*, DAC '03, pages 638–641, New York, NY, USA, 2003. ACM.

[8] Labros Bisdounis, Spyridon Nikolaidis, and Odysseas Koufopavlou. CMOS short-circuit power dissipation including velocity saturation and gate-to-drain capacitive coupling. In *Power And Timing Modeling, Optimization and Simulation - 6th International Workshop (PATMOS'1996)*, January 1996.

[9] David C. Black and Jack Donovan. *SystemC: from the ground up*. Springer-Verlag, Berlin, December 2005.

[10] Nicola Bombieri, Franco Fummi, and Graziano Pravadelli. Automatic abstraction of RTL IPs into equivalent TLM descriptions. *IEEE Transactions on Computers*, 60(12):1730–1743, December 2011.

[11] John Adrian Bondy. *Graph theory with applications*. Elsevier Science Ltd., GBR, 1976.

[12] Tayeb Bouhadiba, Matthieu Moy, and Florence Maraninchi. System-level modeling of energy in TLM for early validation of power and thermal management. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1609–1614, San Jose, CA, USA, 2013. EDA Consortium.

[13] Laura Brandan Briones and Mathias Roehl. *Test derivation from timed automata*, pages 201–231. Number 3472 in Lecture Notes in Computer Science. Springer, 2005.

[14] Annette Bunker, Ganesh Gopalakrishnan, and Sally A. Mckee. Formal hardware specification languages for protocol compliance verification. *ACM Trans. Des. Autom. Electron. Syst.*, 9(1):1–32, January 2004.

[15] Alessandro Danese, Ivan Zandonà, and Graziano Pravadelli. Automatic generation of power state machines through dynamic mining of temporal assertions. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2016.

[16] Nagu Dhanwada, David Hathaway, Victor Zyuban, Peng Peng, Karl Moody, William Dungan, Arun Joseph, Rahul Rao, and Christopher Gonzalez. Efficient PVT independent abstraction of large IP blocks for hierarchical power analysis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 458–465. IEEE Press, 2013.

[17] Nagu Dhanwada, Ing-Chao Lin, and Vijay Narayanan. A power estimation methodology for SystemC transaction level models. In *3rd IEEE/ACM/IFIP Intl. Conf. on Hardware/Software Codesign and System Synthesis*, CODES+ISSS'05, pages 142–147, October 2005.

[18] Wilm E. Donath. Placement and average interconnection lengths of computer logic. *Circuits and Systems, IEEE Transactions on*, 26(4):272–277, 1979.

[19] Thomas Ducroux, Germain Haugou, Vincent Risson, and Pascal Vivet. Fast and accurate power annotated simulation: application to a many-core architecture. In *23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 191–198, September 2013.

[20] Niels Ferguson and Bruce Schneier. *Practical cryptography*. John Wiley & Sons, Inc., USA, 2003.

[21] OFFIS Institute for Information Technology. Timed value streams. `https://github.com/offis/libtvs`. Accessed: 2018-02-21.

[22] Martin Gag, Tim Wegner, and Dirk Timmermann. System level power estimation of system-on-chip interconnects in consideration of transition activity and crosstalk. In René Leuken and Gilles Sicard, editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation*, volume 6448 of *Lecture Notes in Computer Science*, pages 21–30. Springer Berlin Heidelberg, 2011.

[23] Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. Event-clock automata: from theory to practice. *CoRR*, abs/1107.4138, 2011.

[24] Bhavishya Goel and Sally A. McKee. A methodology for modeling dynamic and static power consumption for multicore processors. *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 273–282, 2016.

[25] David J. Greaves and Mehboob Yasin. TLM POWER3: power estimation methodology for SystemC TLM 2.0. In *FDL*, pages 106–111. IEEE, 2012.

[26] Kim Grüttner, Philipp Hartmann, Tiemo Fandrey, Kai Hylla, Daniel Lorenz, Stefan Hauck-Stattelmann, Björn Sander, Oliver Bringmann, Wolfgang Nebel, and Wolfgang Rosenstiel. A timed-value stream based ESL timing and power estimation and simulation framework for heterogeneous MPSoCs. *International Journal of Parallel Programming*, 48, 12 2020.

[27] Philipp A. Hartmann, Kim Grüttner, and Wolfgang Nebel. Advanced SystemC tracing and analysis framework for extra-functional properties. In *The 11th International Symposium on Applied Reconfigurable Computing (ARC'15)*, April 2015.

[28] Domenik Helms. *Leakage models for high level power estimation*. Phd thesis, Universitaet Oldenburg, 2009.

[29] Digh Hisamoto, Wen-Chin Lee, Jakub Kedzierski, Hideki Takeuchi, Kazuya Asano, Charles Kuo, Erik Anderson, Tsu-Jae King, Jeffrey Bokor, and Chenming Hu. FinFET—a self-aligned double-gate MOSFET scalable to 20 nm. *Electron Devices, IEEE Transactions on*, 47(12):2320–2325, December 2000.

[30] Gerard J. Holzmann. *Design and validation of computer protocols*, pages 176–178. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[31] Kai Hylla, Philipp A. Hartmann, Domenik Helms, and Wolfgang Nebel. Early power & timing estimation of custom hardware blocks based on automatically generated combinatorial macros. In *16. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, March 2013.

[32] Swaminathan Jayaraman, Kishor Kamath D., and Bharat Jayaraman. Towards program execution summarization: Deriving state diagrams from sequence diagrams. In *Conference on Contemporary Computing (IC3), 2014 Seventh International*, pages 299–305, August 2014.

[33] Pawan Kapur, Gaurav Chandra, and Krishna C. Saraswat. Power estimation in global interconnects and its reduction using a novel repeater optimization methodology. In *Proceedings of the 39th annual Design Automation Conference*, pages 461–466. ACM, 2002.

[34] Prince Waqas Khan, Yongjun Kim, Yung-Cheol Byun, and Sang-Joon Lee. Influencing factors evaluation of machine learning-based energy consumption prediction. *Energies*, 14(21), 2021.

[35] Felipe Klein, Rodolfo Azevedo, Luiz Santos, and Guido Araujo. SystemC-based power evaluation with PowerSC. In Sandro Rigo, Rodolfo Azevedo, and Luiz Santos, editors, *Electronic System Level Design*, pages 129–144. Springer Netherlands, 2011.

[36] Leonid Kof. Translation of textual specifications to automata by means of discourse context modeling. In Martin Glinz and Patrick Heymans, editors, *Requirements Engineering: Foundation for Software Quality*, volume 5512 of *Lecture Notes in Computer Science*, pages 197–211. Springer Berlin Heidelberg, 2009.

[37] Kai Koskimies and Erkki Mäkinen. Automatic synthesis of state machines from trace diagrams. *Software: Practice and Experience*, 24(7):643–658, 1994.

[38] Matthias Kuehnle, Alisson Brito, Christoph Roth, Matthias Kruesselin, and Juergen Becker. An approach for power and performance evaluation of reconfigurable SoC at mixed abstraction levels. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011*, 2011.

[39] Matthias Kuehnle, Andre Wagner, and Juergen Becker. A statistical power estimation methodology embedded in a SystemC code translator. In *Proceedings of the 24th Symposium on Integrated Circuits and Systems Design*, SBCCI '11, pages 79–84, New York, NY, USA, 2011. ACM.

[40] Paul E. Landman and Jan M. Rabaey. Architectural power analysis: the dual bit type method. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 3(2):173–187, 1995.

[41] Hugo Lebreton and Pascal Vivet. Power Modeling in SystemC at Transaction Level, Application to a DVFS Architecture. In *IEEE Annual Symposium on VLSI*, ISVLSI'08, pages 463–466, April 2008.

[42] Dongwook Lee, Taemin Kim, Kyungtae Han, Yatin Hoskote, Lizy K. John, and Andreas Gerstlauer. Learning-based power modeling of system-level black-box IPs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '15, pages 847–853, Piscataway, NJ, USA, 2015. IEEE Press.

[43] Ikhwan Lee, Hyunsuk Kim, Peng Yang, Sungjoo Yoo, Eui-Young Chung, Kyu-Myung Choi, Jeong-Taek Kong, and Soo-Kwan Eo. PowerViP: SoC power estimation framework at transaction level. In *Asia and South Pacific Conference on Design Automation*, volume 2006, pages 8+, February 2006.

[44] Dominik Macko, Katarína Jelemenská, and Pavel Cicák. Power-management specification in SystemC. In *Proceedings of the 2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, DDECS '15, pages 259–262, USA, 2015. IEEE Computer Society.

[45] Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18(1):50–60, March 1947.

[46] Grant Martin, Brian Bailey, and Andrew Piziali. *ESL design and verification: a prescription for electronic system level methodology*. Elsevier Inc., 2007.

[47] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. Black-box and white-box early power intent simulation and verification: Two novel approaches. In *Conference on Design and Architectures for Signal and Image Processing (DASIP), 2012*, pages 1–8, 2012.

[48] Ons Mbarek, Alain Pegatoquet, Michel Auguin, and Houssem Eddine Fathallah. Power-aware wrappers for transaction-level virtual prototypes: a black box based approach. In *Proceedings of the 26th International Conference on VLSI Design*, volume 0, pages 239–244, Los Alamitos, CA, USA, 2013. IEEE Computer Society.

[49] Renu Mehra and Jan Rabaey. Behavioral level power estimation and exploration. In *Proc. First International Workshop on Low Power Design*, pages 197–202, April 1994.

[50] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992.

[51] Fabian Mischkalla and Wolfgang Müller. Architectural low-power design using transaction-based system modeling and simulation. In *Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International*, pages 258–265, July 2014.

[52] K. Mistry, C. Allen, C. Auth, B. Beattie, D. Bergstrom, M. Bost, M. Brazier, M. Buehler, A. Cappellani, R. Chau, C.-H. Choi, G. Ding, K. Fischer, T. Ghani, R. Grover, W. Han, D. Hanken, M. Hattendorf, J. He, J. Hicks, R. Huessner, D. Ingerly, P. Jain, R. James, L. Jong, S. Joshi, C. Kenyon, K. Kuhn, K. Lee, H. Liu, J. Maiz, B. Mcintyre, P. Moon, J. Neirynck, S. Pae, C. Parker, D. Parsons, C. Prasad, L. Pipes, M. Prince, P. Ranade, T. Reynolds, J. Sandford, L. Shifren, J. Sebastian, J. Seiple, D. Simon, S. Sivakumar, P. Smith, C. Thomas, T. Troeger, P. Vandervoorn, S. Williams, and K. Zawadzki. A 45nm logic technology with high-k+metal gate transistors, strained silicon, 9 cu interconnect layers, 193nm dry patterning, and 100packaging. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, pages 247–250, December 2007.

[53] Gereon Onnebrink, Rainer Leupers, and Gerd Ascheid. ESL black box power estimation: automatic calibration for IEEE UPF 3.0 power models. In *Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '18, pages 6–11, New York, NY, USA, 2018. ACM.

[54] Gereon Onnebrink, Rainer Leupers, Gerd Ascheid, and Stefan Schürmans. Black box ESL power estimation for loosely-timed TLM models. In *ViPES Workshop 2016*, July 2016.

[55] Vincent Ortland. Automatic generation of power state machines based on power traces. Bachelor thesis, Universitaet Oldenburg, 2015.

[56] The R project for statistical computing. `https://www.r-project.org/`. Accessed: 2015-11-22.

[57] Jan M. Rabaey. *Digital integrated circuits: a design perspective*, chapter 3.3.4, page 144f. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[58] Sherief Reda and Abdullah N. Nowroz. Power modeling and characterization of computing devices. *Foundations and Trends in Electronic Design Automation*, 6(2):121–216, 2012.

[59] Sven Rosinger. *RT-level power-gating models optimizing dynamic leakage-management*. Phd thesis, Universitaet Oldenburg, 2012.

[60] Gordon J. Ross. Parametric and nonparametric sequential change detection in R: the cpm package. *Journal of Statistical Software*, 66(1):1–20, 2015.

[61] Gordon J. Ross. *Sequential and batch change detection using parametric and nonparametric methods*, July 2015. Package 'cpm'.

[62] Yasaman Samei and Rainer Dömer. PowerMonitor: a versatile API for automated power-aware ESL design. In *Forum on specification & Design Languages (FDL)*, 2014.

[63] Stefan Schürmans, Gereon Onnebrink, Rainer Leupers, Gerd Ascheid, and Xiaotao Chen. ESL power estimation using virtual platforms with black box processor models. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 354–359, July 2015.

[64] Stefan Schürmans, Diandian Zhang, Dominik Auras, Rainer Leupers, Gerd Ascheid, Xiaotao Chen, and Lun Wang. Creation of ESL power models for communication architectures using automatic calibration. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 1–6, New York, NY, USA, 2013. ACM.

[65] Mel Keytingan M. Shapi, Nor Azuana Ramli, and Lilik J. Awalin. Energy consumption prediction by using machine learning for smart building: case study in Malaysia. *Developments in the Built Environment*, 5:100037, 2021.

[66] George Sobral Silveira, Alisson V. Brito, Helder F. de A. Oliveira, and Elmar U. K. Melcher. Open Systemc simulator with support for power gating design. *Int. J. Reconfig. Comput.*, 2012:1–8, January 2012.

[67] Efstathios Sotiriou-Xanthopoulos, G. Shalina Percy Delicia, Peter Figuli, Kostas Siozios, George Economakos, and Jürgen Becker. A power estimation technique for cycle-accurate higher-abstraction SystemC-based CPU models. In *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, 2015.

[68] STMicroelectronics. FD-SOI technology. `https://www.st.com/content/st_com/en/about/innovation---technology/FD-SOI/learn-more-about-fd-soi.html`. Accessed: 2021-09-11.

[69] Dirk Stroobandt, Herwig Van Marck, and Jan Van Campenhout. An accurate interconnection length estimation for computer logic. In *VLSI, 1996. Proceedings., Sixth Great Lakes Symposium on*, pages 50–55. IEEE, 1996.

[70] Donald E. Thomas, Elizabeth D. Lagnese, Robert A. Walker, Jayanth V. Rajan, Robert L. Blackburn, and John A. Nestor. *Algorithmic and register-transfer level synthesis: the system architect's workbench*. Springer US, 1990.

[71] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, pages 267–288, 1996.

[72] Chiraz Trabelsi, Rabie Ben Atitallah, Samy Meftali, Jean-Luc Dekeyser, and Abderrazek Jemai. A model-driven approach for hybrid power estimation in embedded systems design. *EURASIP Journal on Embedded Systems*, 2011(1):569031, 2011.

[73] Joakim Urdahl, Dominik Stoffel, and Wolfgang Kunz. Path predicate abstraction for sound system-level models of RT-level circuit designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(2):291–304, 2014.

[74] Giovanni B. Vece, Massimo Conti, and Simone Orcioni. PK tool 2.0: a SystemC environment for high level power estimation. In *Conference on Electronics, Circuits and Systems, 2005. ICECS 2005. 12th IEEE International*, December 2005.

[75] Cedriv Walravens, Yves Vanderperren, and Wim Dehaene. ActivaSC: a highly efficient and non-intrusive extension for activity-based analysis of SystemC models. In *46th ACM/IEEE Design Automation Conference (DAC'2009)*, pages 172–177, July 2009.

[76] Lin Zhong and Niraj K. Jha. Interconnect-aware low-power high-level synthesis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(3):336–351, 2005.