Implementierung eines bayesschen Algorithmus zur Optimierung von Syntheseergebnissen

Masterarbeit
Zur Erlangung des akademischen Grades
Master of Science

Fachhochschule Dortmund

Autor:
Aaron Beer

Prüfer:
Prof. Dr. Michael Karagounis

Wintersemester 2021/2022

Thema der Masterthesis

Implementierung eines bayesschen Algorithmus zur Optimierung von Syntheseergebnissen

Kurzzusammenfassung

Die vorliegende Masterthesis beschreibt die Implementierung eines bayesschen Algorithmus zur Optimierung von Syntheseergebnissen. Zu Beginn wird eine Einleitung in die Synthese digitaler Schaltungen sowie aller für die Optimierung relevanten Parameter gegeben. Das Liberty-Format zur Beschreibung von Zellbibliotheken wird erläutert und die für die Optimierung erstellte Zellbibliothek imes_cc wird vorgestellt. Daraufhin wird die Synthese von Testschaltungen unter Einbezug der Bibliothek mithilfe eines automatisierten Arbeitsablaufs vorgestellt. Hierbei werden Timing-, Area-, und Power-Parameter zur Beurteilung der synthetisierten Netzliste aus den erstellten Reports herausgelesen und vergleichend dargestellt. Die Implementierung des Algorithmus auf Basis des Scikit-Optimize-Moduls wird daraufhin erläutert und die erzielten Optimierungen anhand der Testschaltungen dargestellt.

Title of the Thesis

Implementation of a bayesian Algorithm for the Optimization of Synthesis Results

Abstract

This master thesis describes the implementation of a Bayesian algorithm for the optimization of synthesis results. At the beginning, an introduction to the synthesis of digital circuits, as well as all parameters relevant for the optimization is given. The Liberty format for describing cell libraries is explained and the cell library imes_cc created for the optimization is presented. The synthesis of test circuits using the library is then presented with the aid of an automated workflow. Timing, area and power parameters for the evaluation of the synthesized netlist are read out from the generated reports and presented comparatively. The implementation of the algorithm on the basis of the Scikit Optimize module is then explained and the achieved optimizations are presented on the basis of the test circuits.

Erklärung

Hiermit versichere ich an Eides statt, dass die von mir vorgelegte Prüfungsleistung selbstständig und ohne unzulässige fremde Hilfe erstellt worden ist. Alle verwendeten Quellen sind in der Arbeit so aufgeführt, dass Art und Umfang der Verwendung nachvollziehbar sind.

Ort, Datum	Unterschrift

Inhaltsverzeichnis

Listings

Abbildungsverzeichnis

Teil I

Einleitung

Das Ziel der vorliegenden Thesis ist die Implementierung eines Bayes'schen Optimierungsalgorithmus zur Optimierung von syntheserelevanten Parametern. Gegenstand der Optimierung sind hierbei Attribute der zur Synthese genutzten Zellbibliothek im Liberty-Format, welche durch den Algorithmus angepasst werden. Der Algorithmus wird hierzu in ein Skript eingebettet, welches Möglichkeiten zur kommandozeilengesteuerten Synthese, sowie der Konfiguration der Optimierung bietet. Der Optimierungsvorgang ist schematisch in Abbildung 1 dargestellt.

Liberty-Bibliotheken sind ein Format zur Beschreibung von Zellen. Innerhalb der Bibliothek befinden sich Attribute, die Zellen in Bezug auf ihre Leistungsaufnahme, Platzbedarf und Timing-Charakteristiken definieren. Synthese-Tools nutzen diese Attribute zum Einen für Simulationen in Bezug auf die synthetisierte Schaltung, zum Anderen aber auch für Optimierungen. Wie weitreichend die möglichen Optimierungen hierbei sind, hängt davon ab, ob die Auswechslung von Zellen durch Zellen mit günstigeren Eigenschaften möglich ist. Somit haben die Attribute der Zellbibliothek direkten Einfluss auf die Optimierung des gegebenen RTL-Designs.

Die Bayes'sche Optimierung ist eine globale Optimierungsmethode zur Auffindung bzw. Approximation der Extrema von Zielfunktionen, die zeitintensiv zu evaluieren sind. Sie ist in Situationen anwendbar, in denen man über keinen geschlossenen Ausdruck für die Zielfunktion verfügt, die Funktion aber über Stichproben beobachten kann. Die Bayes'sche Optimierung ist insbesondere dann sinnvoll, wenn die Auswertungen aufwendig sind und man keinen Zugang zu Ableitungen hat oder wenn das vorliegende Problem nicht konvex ist. Bayes'sche Verfahren sind daher effiziente Ansätze in Bezug auf die für die Optimierung benötigte Anzahl an Funktionsauswertungen. Der implementierte Optimierungsalgorithmus basiert auf der skopt-Bibliothek. Scikit-Optimize (kurz: skopt) ist eine einfache und effiziente Bibliothek zur Minimierung von teuer zu evaluierenden und eventuell verrauschten Black-Box-Funktionen. Sie implementiert mehrere Methoden zur sequentiellen modellbasierten Optimierung. Dabei zielt skopt darauf ab, in möglichst vielen Kontexten zugänglich und einfach zu verwenden zu sein. Scikit-Optimize ist auf NumPy, SciPy und Scikit-Learn aufgebaut.

Das geschilderte Verfahren wird auf die Implementierung digitaler Schaltungen in einem FPGA angewendet. Der FPGA wird hierbei als Black-Box betrachtet, während die Ausgaben der zur Verfügung stehenden Implementierungssoftware, welche Netzlisten aus Logikgattern auf die Ressourcen des FPGA abbildet, in Bezug auf verwendete

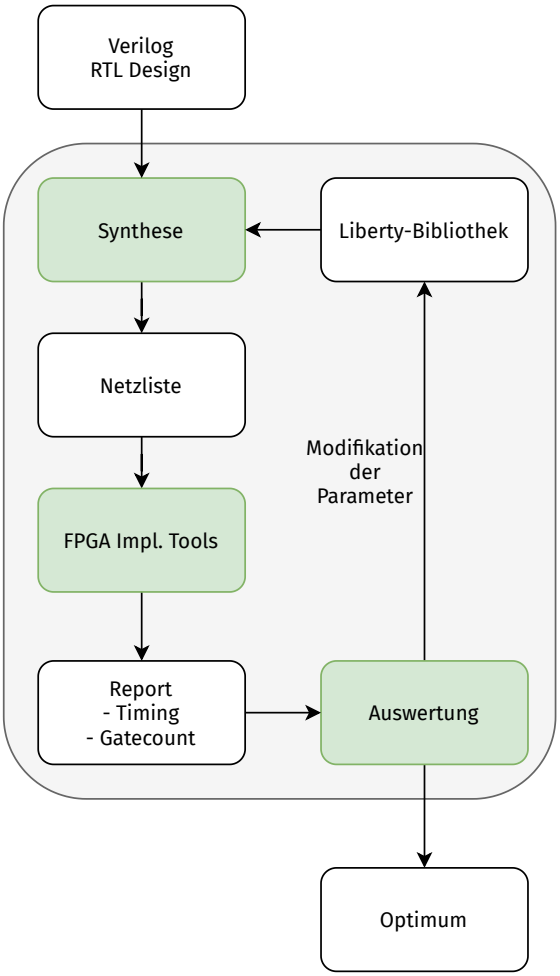

Abbildung 1: Optimierung

Logikressourcen und Timing als Stichprobe genutzt werden. Die Logikgatternetzliste wird aus beispielhaften Schaltungsentwürfen, die in der Hardwarebeschreibung Verilog modelliert wurden, mit Hilfe von proprietären Synthesewerkzeugen erzeugt. Hierfür benötigen die Synthesewerkzeuge eine Beschreibung der in der Zieltechnologie vorhandenen Logikgatter unter Berücksichtigung des Timing- und Power Verhaltens in Form einer Liberty-Bibliothek. Die Parameter der Liberty-Bibliothek werden automatisiert durch den Optimierungsalgorithmus so angepasst, dass am Ende ein optimales Ergebnis in Bezug auf die Stichprobe, d.h. Timing und Anzahl der verwendeter Logikressourcen, der im FPGA implementierten Schaltung, gefunden wird.

Überblick

Die vorliegende Thesis ist thematisch in vier Teile unterteilt.

In Teil I werden grundlegende Begriffe zur Synthese sowie der Simulation relevanter Charakteristiken erläutert. Außerdem werden die verwendeten Sprachen Tool Command Language (TCL), Awk und Python inklusive der genutzten Module vorgestellt.

Teil II beschreibt das Liberty-Format zur Zellbeschreibung und darauf aufbauend die erstellte Liberty-Bibliothek.

Teil III erläutert die Nutzung der Synthese-Tools und stellt die erstellten Synthese-Skripte vor. In diesem Teil werden ebenfalls die zur logischen Äquivalenzprüfung genutzten Tools beschrieben.

Teil IV dient der Erläuterung der erstellten Skripte zur Implementierung des Optimierungsalgorithmus und zur Auswertung der Ergebnisse, die mithilfe des Algorithmus erzielt wurden.

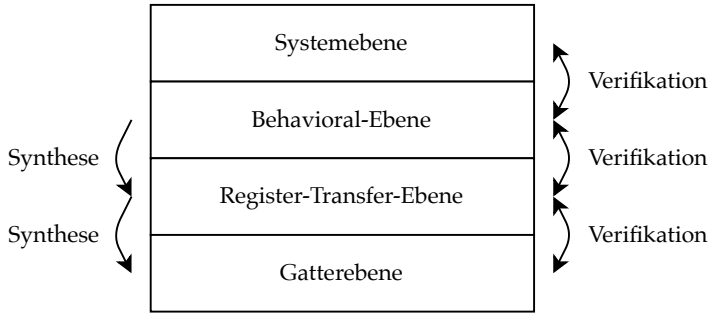

Abbildung 2: Ablauf der Synthese und Verifikation eines Designs

1 Synthese

Digitale Schaltungen können auf verschiedenen Abstraktionsebenen dargestellt werden. Während des Entwurfsprozesses wird eine Schaltung in der Regel auf einer höheren Abstraktionsebene, beispielsweise durch Hardwarebeschreibungssprachen, spezifiziert. Die Implementierung dieser Schaltung geschieht dann durch Konvertierung in eine funktional äquivalente Darstellung auf einer niedrigeren Abstraktionsebene. Wenn diese Konvertierung automatisch durch Software erfolgt, wird der Vorgang als Synthese bezeichnet.
Für die Durchführung der Synthese ist ein Schaltungsentwurf bzw. eine Schaltungsbeschreibung, eine Bibliothek mit Standardzellen und ein Satz von Vorgaben, unter denen das Design optimiert werden soll, notwendig. Bei der genutzten Bibliothek muss darauf geachtet werden, dass sie funktional vollständig ist. Unter funktionaler Vollständigkeit ist zu verstehen, dass die Bibliothek alle benötigten Gatter zur Verfügung stellt. Welche Gatter für die Synthese einer Schaltung notwendig sind, ist hierbei abhängig vom genutzten Synthese-Tool.

Das Endprodukt der Synthese ist eine Gatter-Level Netzliste, welche die in der Bibliothek definierten Zellen nutzt und welche hinsichtlich der Vorgaben optimiert wurde. Ziel der Optimierung ist eine größtmögliche Effizienz in Bezug auf Leistungsaufnahme, Implementationsfläche und Geschwindigkeit. Der Vorteil der Synthese im Gegensatz zu einer manuellen Konvertierung ist die erhöhte Produktivität durch die Nutzung von Automatismen diverser Tools. Des Weiteren können Designs wiederholt auf unterschiedliche Zieltechnologien ausgerichtet synthetisiert werden.

1.0.1 *Ablauf*

Der Ablauf der Synthese und Verifikation zwischen den Abstraktionsebenen ist in Abbildung 2 veranschaulicht. Nachdem die Äquivalenz der Modelle auf Systemebene und Behavioral-Ebene verifiziert worden ist, können die Darstellungen der unteren

Ebenen des Entwurfs mit Hilfe von Synthesewerkzeugen generiert werden. Schließlich werden Register-Transfer-Ebene und Gatterebene verifiziert. Bei der Entwicklung eines Designs wird dieser Designprozess oft in mehreren Iterationen durchlaufen. Dies kann durch spätere Änderung einer Designanforderung verursacht werden oder dadurch begründet sein, dass die Analyse eines Modells mit niedrigerer Abstraktion z.B. die Timing-Analyse auf Gatterebene ergibt, dass eine Design-Änderung erforderlich ist, um die Designanforderungen zu erfüllen. Wann immer eine Änderung auf Behavioral- oder Systemebene stattfindet, muss ihre Äquivalenz erneut überprüft werden, indem Simulationen erneut ausgeführt und die Ergebnisse verglichen werden. Um die Reproduzierbarkeit zu gewährleisten, ist es sinnvoll, dass alle Schritte des Synthese- und Verifikationsvorgangs mit einem festen Satz von Einstellungen erneut ausgeführt werden können.

Aus diesem Grund ist oft gewünscht, dass alle im Synthesefluss verwendeten Tools über Skripte gesteuert werden können. Das bedeutet, dass alle Funktionen über Textbefehle ausführbar sein müssen. Wenn ein Tool eine Graphical User Interface (GUI) zur Verfügung stellt, dann ist diese ergänzend zu der Nutzung des Tools mithilfe von tool command language (tcl)-Skripten und der Befehlszeile zu sehen. Der Synthesefluss in der vorliegenden Arbeit folgt auch einem Skript-basierten Ansatz und wird durch ein Python-Skript gesteuert, welches alle erforderlichen Tools in diesem Fall der Synthese und Verifikation in gewünschter Reihenfolge aufruft. Für die Steuerung der einzelnen Werkzeuge werden dann Skriptdateien im tcl-Format ausgeführt, die spezifische Befehle für das jeweilige Tool enthalten. Alle für die Tools erforderlichen Einstellungen werden durch diese Skripte konfiguriert, sodass keine weitere manuelle Interaktion erforderlich ist.

1.1 Abstraktionsebenen

Im folgenden werden die Abstraktionsebenen, zwischen welchen Synthese- und Verifikationsvorgänge stattfinden, erläutert.

1.1.1 *Systemebene*

Bei Abstraktion eines Entwurfs auf Systemebene werden nur große funktionale Bausteine wie z.B. CPUs und Rechenkerne betrachtet. Auf dieser Ebene wird die Schaltung mit Programmiersprachen wie C/C++ oder Matlab beschrieben. Es können auch spezielle Softwarebibliotheken wie z.B. SystemC verwendet werden, welche auf die Simulation von Schaltungen auf Systemebene ausgerichtet sind. In der Regel werden Synthesetools jedoch nicht verwendet, um eine Schaltung von der Systemebene auf eine niedrigere Abstraktionsebene zu konvertieren.

1.1.2 *Algorithmische Ebene*

Die algorithmische Ebene eines Systems wird mit Programmiersprachen beschrieben, jedoch kommt hier ein reduzierter Funktionsumfang zum Einsatz. Es existieren Werkzeuge zur Synthese von High-Level-Code, welcher normalerweise in C/C++/SystemC-Code mit zusätzlichen Metadaten geschrieben wird, zu verhaltensorientiertem Hardware Description Language (HDL)-Code in Form von Verilog- oder Very High Speed Integrated Circuit Hardware Description Language (VHDL)-Code. Neben kommerziellen Tools für die High-Level-Synthese gibt es auch eine Reihe von Free and Open Source Software (FOSS)-Tools für die High-Level-Synthese.

1.1.3 *Behavioral-Ebene*

Auf der Behavioral-Ebene wird zur Beschreibung der Schaltung eine auf Hardware ausgerichtete Sprache wie Verilog oder VHDL verwendet. Hierbei gibt es Sprachmerkmale, welche die Verwendung von prozeduralen Programmiertechniken zur Beschreibung von Datenpfaden und Registern ermöglicht. In Verilog nennt sich dieses Konstrukt *always*-Block und in VHDL *process*-Block.

1.1.4 *Register-Transfer-Ebene*

Auf der Register-Transfer-Ebene wird der Entwurf durch kombinatorische Datenpfade sowie Register (in der Regel D-Flipflops) beschrieben. Ein Entwurf in Register Transfer Level (RTL)-Darstellung wird unter Verwendung von HDLs wie Verilog und VHDL erstellt. Hierbei wird eine begrenzte Teilmenge des Sprachumfangs verwendet. Neben den bereits genannten always-Blöcken in Verilog oder process-Blöcken in VHDL existieren noch bedingte und unbedingte Zuweisungen, welche zusammen mit Logikoperatoren verwendet werden können. Diese Kontrukte modellieren verwendete Registertypen und Signalzuweisungen für die Datenpfade.

1.1.5 *Generische Logikgatter Ebene*

Auf der Logikgatter-Ebene wird das Schaltungsdesign durch eine Netzliste dargestellt, die nur aus einer kleinen Anzahl von generischen Zellen, wie z.B. den logischen Basisgattern UND, ODER, NICHT, XOR usw. und Registern, in der Regel D-Flipflops, besteht. Es gibt diverse Netzlistenformate, die auf dieser Ebene verwendet werden können, z.B. das Electronic Design Interchange Format (EDIF). Zur Vereinfachung der Simulation wird oft eine HDL-Netzliste verwendet. Bei dieser handelt es sich um eine HDL-Datei, welche nur die grundlegendsten Sprachkonstrukte zur Instantiierung und Verbindung von Zellen nutzt.

1.1.6 *Physikalische Logikgatter Ebene*

Auf der physikalischen Gatterebene werden nur Gatter verwendet, die auf der Zieltechnologie tatsächlich verfügbar sind. Diese können in bestimmten Fällen nur NAND-, NOR- und NOT-Gatter sowie D-Flipflops sein. Es kann sich jedoch auch um Zellen handeln, die komplexer sind als die auf der logischen Gatterebene verwendeten Zellen, wie z.B. Fulladder, Multiplizierer usw. Im Falle eines Field Programmable Gate Array (FPGA)-basierten Entwurfs ist die physikalische Ebene eine Netzliste von Lookup Table (LUT)s mit optionalen Ausgangsregistern, da diese die Grundbausteine der FPGA-Logikzellen darstellen. Für die Synthese-Toolchain ist diese Abstraktion in der Regel die unterste Ebene.

1.1.7 *Transistorebene*

Die Darstellung einer Schaltung auf Transistorebene entspricht einer Netzliste, die einzelne Transistoren als Zellen verwendet. Die Modellierung der Transistorebene ist in Verilog und VHDL möglich, wird aber selten verwendet. Im digitalen Schaltungsentwurf in Zusammenhang mit modernen Application-specific Integrated Circuit (ASIC)- oder FPGA-Technologien werden die physikalischen Gatter als die atomaren Bausteine der Logikschaltung betrachtet.

2 Verzögerungszeiten

2.1 Sequentielle Schaltungen

Der Großteil digitaler Schaltungen ist sequentieller Natur und arbeitet synchron zu einem Takt. Das meistgenutzte Speicherelement in sequentiellen Schaltungen ist das D-Flipflop (DFF). Anhand des DFFs sollen Parameter erklärt werden, die das Bauteil in Bezug auf sein Zeitverhalten beschreiben. Diese Paramater sind im Folgenden:

clock to output - t_{cq}: Die Dauer bis Daten bei eingehender Taktflanke am Ausgang anliegen. Der Timing-Bogen ist in Abbildung 3 dargestellt.

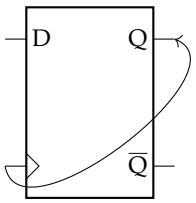

Abbildung 3: Clock to output am D-Flipflop

setup time - t_{setup}: Die Zeit, in der das Datensignal vor dem Taktsignal stabil anliegen muss. In Abbildung 4 ist ein typischer Signalverlauf am D-Flipflop dargestellt. Es ist zu sehen, dass bei den ersten beiden Pegelwechseln am D-Signal die Setup-Time eingehalten wird und somit fehlerfrei am Q-Ausgang übernommen werden können. Im dritten Fall wechselt das D-Signal kurz vor der steigenden Taktflanke, sodass die Setup-Time nicht eingehalten wird.

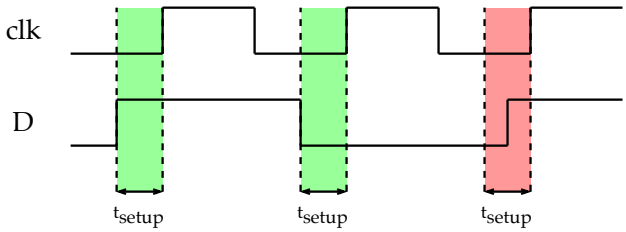

Abbildung 4: Setup Time

hold time - t_{hold}: Die Zeit nach dem Taktsignal, in der das Datensignal weiterhin stabil anliegen muss. In Abbildung 5 sind drei Beispiele für gültige und ungültige Übergänge in Bezug auf die Hold-Zeit t_{hold} dargestellt.

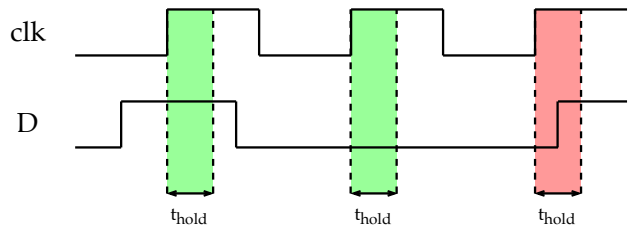

Abbildung 5: Hold Time

2.2 Constraints

Ausgehend von den in Kapitel 2.1 definierten Timing-Parametern eines D-Flipflops lassen sich in diesem Zusammenhang zwei Probleme feststellen, die in synchronen Schaltungen auftreten können. Zum Einen kann es dazu kommen, dass den Daten nicht genug Zeit zur Verfügung steht, um zum nächsten Register zu gelangen, ehe eine Taktflanke bei diesem eingeht. Zum Anderen kann es vorkommen, dass neue Daten zum nächsten Register gelangen, bevor die alten sicher übernommen werden konnten. Den ersten Fall bezeichnet man als »max delay violation«, den zweiten als »min delay violation«. Grund für max delay violations sind ein langsamer Datenpfad. Im Gegensatz dazu entstehen min delay violations durch kurze Datenpfade. Pfade auf denen es zu max delay violations kommen kann, werden als Setup-Pfade bezeichnet und Pfade auf denen min delay violations auftreten können, werden als Hold-Pfad bezeichnet. Zur Vermeidung dieser Fehler ist die Definition von Vorgaben (constraints) notwendig, welche Signallaufzeiten definieren, bei denen die beschriebenen Fehler nicht auftreten. Um Vorgaben festlegen zu können, ist es notwendig sich zwei Signalpfade anzuschauen und zwar den Takt- und den Datenpfad. Eine störungsfreie Funktionsweise der Register wäre dann gegeben, wenn Takt- und Datensignal den exakt gleichen Pfad benutzen, sodass sie bei jedem Flipflop ohne Zeitverzug ankommen. Es ist jedoch nicht möglich, die Schaltung auf diese Weise aufzubauen, da Daten gegebenenfalls durch mehrere Flipflops und Logikgatter geleitet werden (vgl. Abb. 6). Zur Definition der Constraints sind die Signallaufzeiten zu beachten:

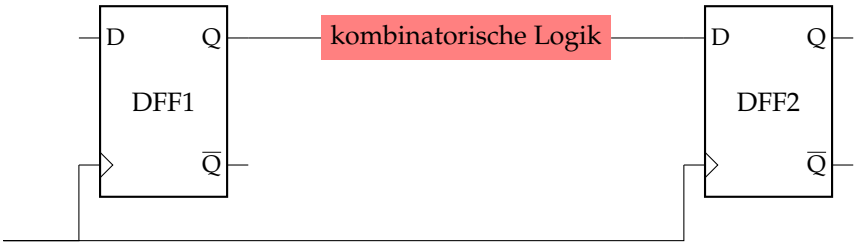

Abbildung 6: Pfade an DFF

1. Signaldurchlaufzeit t_{cq} von Eingang D zu Ausgang Q von DFF1

2. Laufzeit der Daten durch die kombinatorische Logik t_{logic}

3. Setup-Time t_{setup} am Eingang D von Flipflop 2

Daher ist die erste an Flipflop 1 anliegende Taktflanke als startende Taktflanke zu verstehen, welche dazu führt, dass sich das Signal an Ausgang Q von Flipflop 1 nach t_{cq} ändert. Nach diesem Ereignis ist sicherzustellen, dass die Daten in einer Zeit t_{logic} durch die kombinatorische Logik geleitet werden, sodass sie vor der Setup-Time an Eingang D von Flipflop 2 anliegen.

2.2.1 *Setup Time*

Aus diesen Zusammenhängen geht hervor, dass die Periodendauer des Taktsignals (T) größer sein muss als die Summe der Verzögerungszeiten und der Setup Time (vgl. Gleichung 1)

$$T > t_{cq} + t_{logic} + t_{setup} \tag{1}$$

Daher ist es nötig bei Problemen mit der Einhaltung der Setup-Zeit, die Frequenz des Taktsignals zu reduzieren, woraus eine steigende Periodendauer resultiert.

2.2.2 *Hold Constraint*

Hold-Probleme entstehen durch einen logischen Pegelwechsel an Flipflop 2 bevor die Hold-Zeit t_{hold} vergangen ist (vgl. Abb. 7). Dieses Verhalten führt dazu, dass die am Eingang anliegenden Daten nicht sicher übernommen werden können.

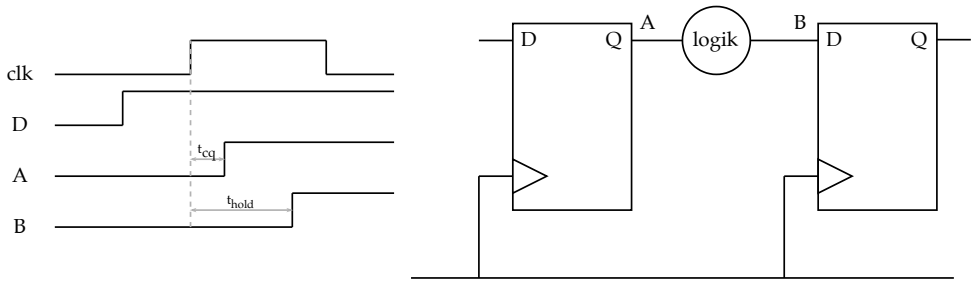

Abbildung 7: Hold Constraint

Abbildung 7 zeigt den Signalverlauf an zwei Flipflops. Nach der Verzögerung t_{cq} (clock to output) wechseln die Daten am Ausgang von Flipflop 1 (A) ihren Wert. Die Verzögerung t_{pd} entsteht durch die Logik zwischen den Flipflops. Nach ihr ändert sich der logische Pegel an Punkt B. Sollte hierbei die Hold-Zeit noch nicht abgelaufen sein, kann der Pegelwechsel bei der gleichen Taktflanke von Flipflop 2 übernommen

werden. Daraus geht hervor, dass die Hold Zeit kleiner sein muss als die Summe aus t_{cq} und t_{logic} (vgl. Gleichung 2).

$$t_{hold} < t_{cq} + t_{logic} \tag{2}$$

2.2.3 *Recovery Time*

Die Recovery Time ist die Zeit, in der ein asynchrones Signal vor der nächsten Taktflanke stabil bleiben muss.

2.2.4 *Removal Time*

Die Removal Time ist die Zeit, die ein asynchrones Signal nach der vorangegangen Taktflanke stabil bleiben muss.

2.2.5 *MPW*

MPW gibt die Zeit an, die ein Taktsignal nach einer steigenden oder fallenden Flanke stabil bleiben muss.

3 Timing Analysis

Unter dem Begriff Timing Analysis versteht man die Static Timing Analysis (STA), sowie die Timing Simulation.

3.1 Static Timing Analysis

Die STA ist eine Methode zur Validierung des Timings einer Schaltung. Die Analyse untersucht alle möglichen Pfade auf Timing-Verletzungen. Sie berücksichtigt die worst-case Verzögerung durch jedes Logikelement, aber nicht die logische Funktion der Schaltung. Anwendung findet die STA bei synchronen Schaltungen.

3.1.1 *Ablauf*

Um eine Schaltung auf Verstöße zu prüfen, gibt es drei Hauptschritte:

1. Die Schaltung wird in Zeitpfade unterteilt.
2. Die Signalausbreitungsverzögerung entlang jedes Pfades wird berechnet.
3. Verstöße gegen Timing Constraints innerhalb des Designs und an der Eingabe-/Ausgabeschnittstelle werden geprüft.

Die STA analysiert die Pfade von jedem Startpunkt zu jedem Endpunkt und vergleicht sie mit den Constraints, die für diesen Pfad bestehen. Alle Pfade sollten dabei über Constraints definierte Timing-Beschränkungen haben.

3.1.2 *Timing Paths*

Ein Timing Path ist eine Route von einem Startpunkt zu einem Endpunkt. Startpunkt in digitalen Schaltungen kann dabei beispielsweise der Eingangsport eines Flipflops sein. Endpunkte sind beispielsweise die Ausgangsports. Hierbei kann es mehrere Pfade geben, die zu einem Endpunkt führen, sowie mehrere Pfade für die selbe Start- und Endpunkt-Kombination. Es gibt vier Kategorien von Timing Paths:

1. Register zu Register (reg2reg)
2. Input zu Register (in2reg)
3. Register zu Output (reg2out)
4. Input zu Output (in2out)

Zum Verständnis der Pfadunterteilungen ist es notwendig, eine Repräsentation der Pfade zu finden, in der die Verzögerungszeiten sowie Anfangs- und Endpunkte dargestellt werden können. Die Darstellung einer logischen Schaltung bestehend

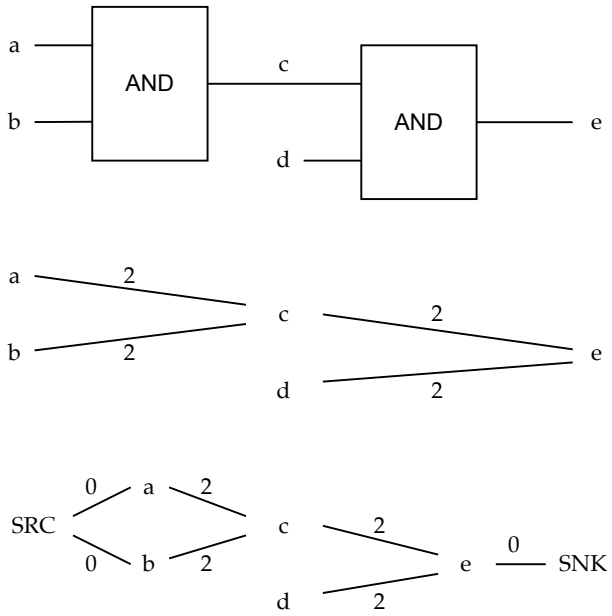

Abbildung 8: Timing Paths

aus zwei UND-Gattern ist in Abbildung 8 dargestellt. Hierbei wird eine interne Delay-Zeit von 2 für das Gatter angenommen. Im ersten Schritt werden die Gatter abstrahiert und als Knoten (engl. Nodes) mit zugehöriger Delay-Zeit dargestellt. Dieser Schritt ist möglich, da die Timing-Analyse nicht die logische Funktion der Schaltung betrachtet. Daraufhin werden Source (SRC)- und Sink (SNK)-Nodes hinzugefügt mit einer Delay-Zeit von 0, sodass alle Pfade an einem Punkt starten und enden. Für die Untersuchung komplexer Schaltungen ist es nicht praktikabel, jeden Pfad einzeln zu betrachten. Stattdessen wird für jede Node nur der worst-delay-Pfad betrachtet. Im Zusammenhang dazu gibt es zwei wichtige Werte anhand derer die Bewertung von Timing-Pfaden erfolgt:

Arrival Time at Node (AT) Der längste Pfad von Source zu Node

Required Arrival Time at node (RAT) Die Zeit, die das Signal zur Verfügung hat, um bei Einhaltung der Constraints von der Node zur Sink zu kommen

Unter *Slack* versteht man die Differenz aus geforderter und tatsächlicher Ankunftszeit eines Signals an einem Zielknoten. Ein positiver Slack-Wert bedeutet, dass das Signal vor seiner geforderten Ankunftszeit am Zielpunkt eintrifft, da die geforderte Ankunfszeit höher ist als die tatsächlich eintretende. Ein negativer Slack signalisiert im Gegensatz dazu, dass das Signal zu spät eintrifft. Der Slack einer Node n berechnet sich aus den aufgeführten Werten zu:

$$Slack(n) = RAT(n) - AT(n)$$

Anhand eines Beispiels soll die Berechnung von ATs und RATs verdeutlicht werden. Hierzu wird eine Beispielschaltung als Directed Acyclic Graph (DAG) dargestellt (vgl. Abb. 9). Die Zahlen zwischen den Nodes stehen für die Verzögerungszeiten. Die Kästchen oberhalb der Nodes dienen der Eintragung von ATs, RATs und Slack. Im

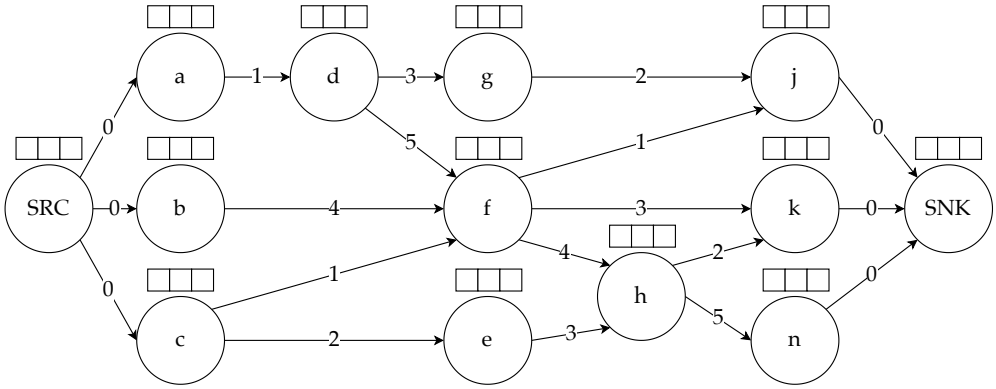

Abbildung 9: Directed Acyclic Graph

ersten Schritt werden die ATs berechnet, woraufhin für jede Node die worst-case Zeiten gefunden werden. Das heißt, es findet eine Betrachtung jedes Eingangs der Node statt und es wird der jeweils höchste Delay-Wert übernommen. Das Ergebnis ist in Abbildung 10 dargestellt.

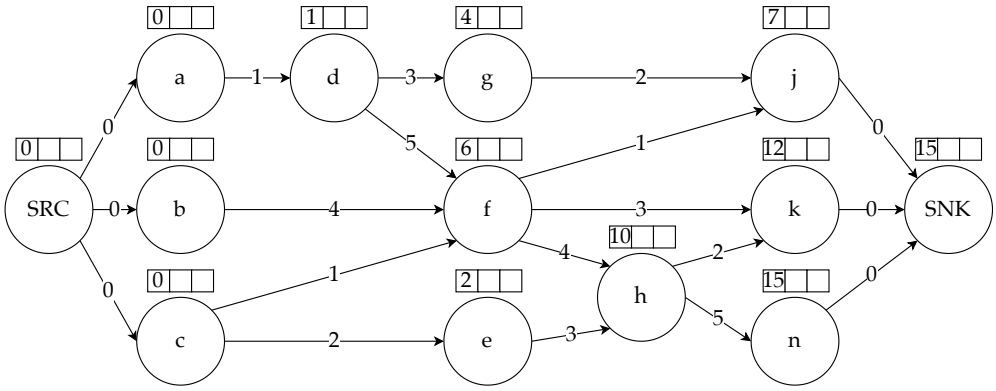

Abbildung 10: Directed Acyclic Graph mit eingetragenen ATs

Im nächsten Schritt müssen die RATs berechnet werden. Für die Berechnung der RATs wird eine benötigte Arrival Time von $T = 12$ vorausgesetzt. RATs werden von der SNK Node ausgehend berechnet. Analog zur Berechnung der ATs wird hier die Delay-Zeit genommen und von der RAT substrahiert. Bei Nodes mit Fanouts größer als 1, wird der niedrigste Wert übernommen. Das Ergebnis ist in Abbildung 11 zu sehen. Auffällig ist hierbei, dass es zu negativen RATs kommen kann.

Nach der Berechnung von ATs und RATs kann im Folgenden der Slack berechnet

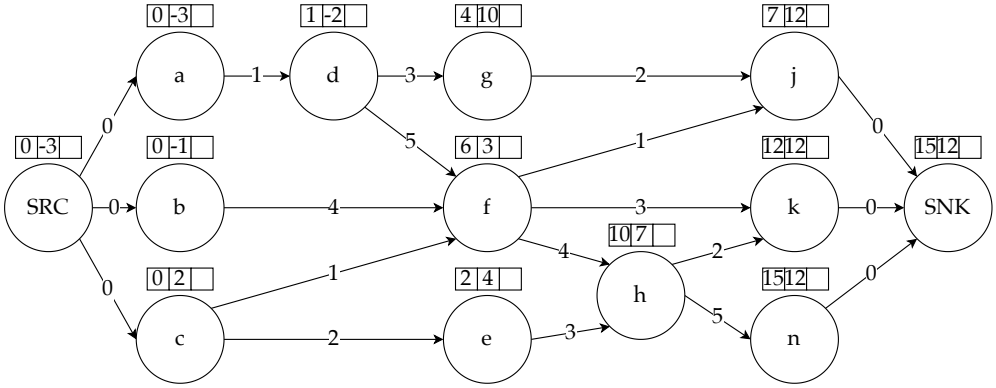

Abbildung 11: Directed Acyclic Graph mit eingetragenen RATs

werden. Der Slack berechnet sich aus $RAT - AT$. Nachdem der Slack ermittelt worden ist, ist es möglich den Critical Path zu definieren. Der Critical Path ist gekennzeichnet durch den niedrigsten Slack-Wert, in diesem Fall handelt es sich um den Wert -3. Da dieser Wert negativ ist, ist der Pfad zu langsam. In Abbildung 12 ist der kritische Pfad im gewählten Beispiel hervorgehoben.

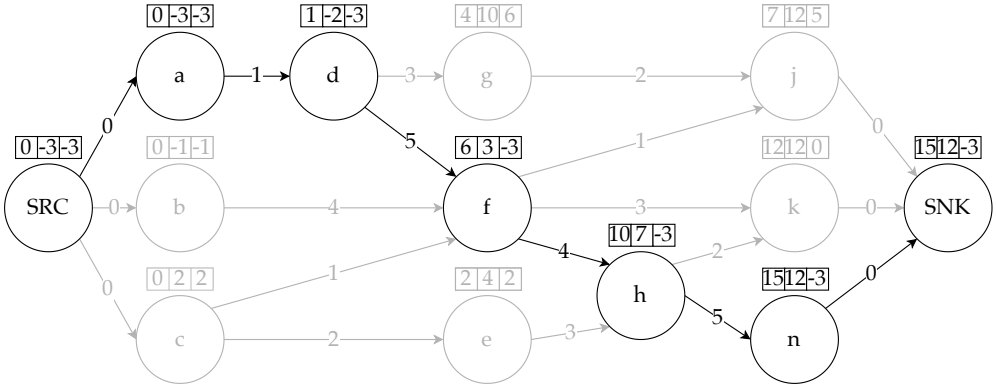

Abbildung 12: Critical Path

3.1.3 *False Paths*

Die STA überprüft nicht die logische Funktion einer Schaltung. Deshalb kann es passieren, dass ein Pfad als Critical Path gekennzeichnet wird, der logisch nicht nutzbar ist. Diese Art von Pfad wird als False Path bezeichnet. Abbildung 13 zeigt ein Beispiel für einen False Path. Auf der linken Seite ist der DAG dargestellt. Über die bereits vorgestellte Analyse der Pfade wird ersichtlich, dass der Pfad $a \rightarrow d \rightarrow f \rightarrow g \rightarrow j$ ein Critical Path ist. Für den Fall, dass die Schaltung wie die rechte Seite aufgebaut ist, ist dieser Pfad allerdings nicht erreichbar, da bei einer logischen 1 an

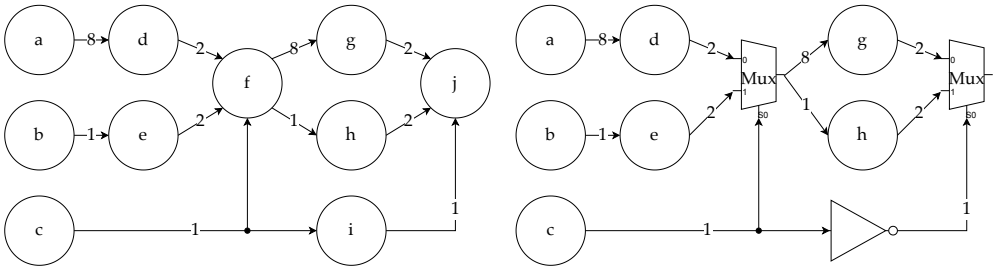

Abbildung 13: False Path

Eingang c der Pfad $b \rightarrow e \rightarrow g$ entsteht und bei einer logischen 0 an c der Pfad $a \rightarrow d \rightarrow h$ entsteht. Derartige Pfade machen es nötig Vorgaben zu definieren, welche diese Pfade von der Analyse ausschließen.

4 TCL

TCL steht für »tool command language« (gesprochen »tickle«) und ist eine Skriptsprache zur Steuerung sowie Erweiterung von Anwendungen. Tcl bietet generische Programmiermöglichkeiten, die für eine Vielzahl von Anwendungen wie z.B. Variablen, Schleifen und Prozeduren nützlich sind. Der TCL-Interpreter ist als Bibliothek von C-Prozeduren implementiert, die leicht in Anwendungen integriert werden können.

4.1 Nutzung von TCL

TCL-Skripte werden oft im Zusammenspiel mit dem Programm tclsh ausgeführt. Tclsh ist ein interaktiver Kommandozeilen-Interpreter, der sich entweder direkt mit einem Skript oder im interaktiven Modus starten lässt. Die Ausführung einer tcl-Datei geschieht über den Befehl `tclsh <skript>.tcl`. Wenn tclsh ohne Argumente gestartet wird, wird der interaktive Modus ausgeführt. Im interaktiven Modus lassen sich Befehle interaktiv eingeben, die Tcl dann ausführt und das Ergebnis oder die daraus resultierenden Fehlermeldungen anzeigt. Der Interpreter wird mit dem exit-Befehl beendet. Eine Liste aller Befehle, die dem Interpreter bekannt sind, erhält man über `info commands`.

4.2 Wichtige Funktionen

4.2.1 *Textausgabe*

Die Ausgabe von Strings erfolgt in tcl über den puts-Befehl. Wenn die Zeichenfolge aus mehr als einem Wort besteht, muss die Zeichenfolge in doppelte Anführungszeichen oder geschweifte Klammern gesetzt werden. Ein Satz von Wörtern, der in Anführungszeichen oder geschweiften Klammern eingeschlossen ist, wird als ein String behandelt.

```
puts "string"
```

Listing 1: puts-Befehl

4.2.2 *Variablen*

Variablen werden in tcl mit einem $ gekennzeichnet.

4.2.3 *Zuweisung von Variablen*

Tcl folgt dem »everything is a string« -Paradigma, das heisst Daten werden ausschließlich als String dargestellt. Intern können Daten jedoch als Liste, Ganzzahl, Double oder anderer Typ gespeichert werden. Der Befehl zur Zuweisung von Werten auf Variablen in Tcl ist `set`. Beim Aufruf von `set` mit zwei Argumenten wird der Wert des zweiten Arguments in den durch das erste Argument referenzierten Speicher geschrieben.

```
set varName ?value?
```

Listing 2: set-Befehl

4.2.4 *Starten von Programmen*

In TCL gibt es zwei Möglichkeiten Programme zu öffnen. Die Syntax ist in Listing 3 dargestellt.

```
exec ?switches? arg ?arg ...? ?&?
open fileName access permissions
```

Listing 3: exec- und open-Befehl

5 awk

Im Rahmen des Projekts wird awk zur Analyse der Reports genutzt. Die Skriptsprache awk dient der Analyse und Auswertung von Daten. Der Name leitet sich von den Namen der Entwickler ab: Aho, Weinberger, Kernighan. Im Folgenden sollen grundlegende Funktionen erläutert werden.

5.1 Aufbau

Der allgemeine Aufbau von awk-Befehlen ist in Listing 4 dargestellt.

```
        Bedingung {Ausdruck}
```

Listing 4: Aufbau von awk-Befehlen

Zur Verarbeitung der Daten liest awk jede Zeile der zu analysierenden Daten ein und es wird ermittelt, ob die Bedingung zutrifft. Ist dies der Fall, wird der Ausdruck, welcher innerhalb der geschweiften Klammern angegeben wird, auf die aktuelle Zeile ausgeführt. Falls keine Bedingung angegeben ist, wird der Ausdruck auf jede Zeile angewandt.

5.2 Variablen

Zur Wahl eines Feldes innerhalb einer Zeile wird das $-Symbol gefolgt von der Nummer des Feldes genutzt. Soll beispielsweise das zweite Feld einer Zeile ausgewählt werden, geschieht das über die Variable $2. Die Variable $NF gibt die Gesamtzahl der Felder einer Zeile an, wodurch die Variable genutzt werden kann, um das letzte Feld der Zeile zu selektieren. Felder werden in awk standardmäßig durch Leerzeichen getrennt. Für den Fall, dass ein alternatives Trennungszeichen verwendet wird, kann dieses über die Variable $FS (Field Separator) definiert werden.

5.3 Bedingungen

Bedingungen erlauben die Kontrolle darüber, welche Daten modifiziert oder anderweitig weiter verarbeitet werden. Bedingungen werden über einen Vergleich mit einem *regulären Ausdruck* realisiert. Der Term *regulärer Ausdruck* (engl. regular expression) bezeichnet die Beschreibung eines Textmusters. Diese Beschreibungen können aus mehreren Untereinheiten aufgebaut sein und über Operatoren miteinander verbunden werden. Listing 5.3 zeigt Beispiele für reguläre Ausdrücke.

```
/regexp/       # Vergleicht aktuelle Zeile mit regexp
$0 ~ /regexp/  # Vergleicht aktuelle Zeile mit regexp
/^set /        # Vergleicht alle Zeilen, die mit "set" beginnen
```

6 Python

Python hat sich zu einer der meistgenutzten Sprachen in den Bereichen Data Science und Machine Learning entwickelt. Es kombiniert die Leistungsfähigkeit allgemeiner Programmiersprachen mit der Benutzerfreundlichkeit von domänenspezifischen Skriptsprachen wie MATLAB oder die Programmiersprache R. Python verfügt über Bibliotheken für Daten, Visualisierung, Statistik, Verarbeitung natürlicher Sprache, Bildverarbeitung und mehr. Dieser umfangreiche Werkzeugkasten bietet dem Nutzer eine große Auswahl an allgemeinen und spezifischen Funktionen.
Maschinelles Lernen und Datenanalyse sind grundlegend iterative Prozesse, bei denen Vorgänge getrieben durch Daten verfolgt werden. Es ist für diese Prozesse wichtig, Werkzeuge zu haben, die eine schnelle Iteration und einfache Interaktion ermöglichen. Als universelle Programmiersprache erlaubt Python auch die Erstellung von komplexen grafischen Benutzeroberflächen (GUIs) und Web-Services sowie die Integration in bestehende Systeme.

6.1 Module

Während Funktionen es ermöglichen, Code-Segmente so zu unterteilen, dass sie wiederverwendet werden können, sind Module eine Sammlung von Funktionen und benutzerdefinierten Datentypen. Die gebündelten Funktionalitäten können so von einer beliebigen Anzahl von Programmen verwendet werden. Weiterhin bietet Python Möglichkeiten zur Erstellung von Paketen. Pakete sind Gruppen von Modulen, die zusammengefasst werden. In der Regel bieten diese Module verwandte Funktionalitäten oder sind voneinander abhängig.

Im Folgenden werden die Module vorgestellt, mit denen der in dieser Arbeit erstellte Optimierungsalgorithmus arbeitet.

6.1.1 *Matplotlib*

Matplotlib ist ein Python-Paket für 2D-Diagramme, das produktionsreife Diagramme erzeugt. Es unterstützt interaktives und nicht-interaktives Plotten und kann Bilder in verschiedenen Ausgabeformaten speichern. Es kann mehrere Fenster-Toolkits verwenden, wie GTK+, wxWidgets und Qt und bietet eine große Auswahl an Diagrammtypen, wie beispielsweise Linien, Balken, Kreisdiagramme und Histogramme. Darüber hinaus ist es in hohem Maße anpassbar, flexibel und einfach zu bedienen. Die duale Natur von Matplotlib erlaubt die Anwendung sowohl in interaktiven als auch in nicht-interaktiven Skripten. Es kann in Skripten ohne grafische Darstellung verwendet werden, eingebettet in grafische Anwendungen, oder auf Webseiten. Mit dem Python-Interpreter oder IPython kann es auch interaktiv verwendet werden.

In den Listings 5 und 6 ist ein einfacher Anwendungsfall zur Generierung eines Plots aufgeführt. Die Listings zeigen zwei Möglichkeiten der Plot-Erstellung mittels Matplotlib. In Listing 5 ist die Generierung des Plots durch die Nutzung der plot-Funktion gezeigt. Listing 6 zeigt einen objektorientierten Ansatz. Beide Ansätze führen zu dem in Abbildung 14 gezeigten Plot.

Zur Nutzung der Module müssen zunächst die entsprechenden Importe vorgenommen werden, sodass daraufhin über die Funktion plot die x-Werte des Graphen übergeben werden. Über die Funktionen xlabel/ylabel bzw. set_xlabel/set_ylabel werden die Achsen beschriftet. Durch title bzw. set_title wird dem Plot ein Name gegeben. Legenden lassen sich über die Funktion legend erzeugen. Durch die show-Funktion werden die erzeugten Plots angezeigt (siehe Abb. 14).

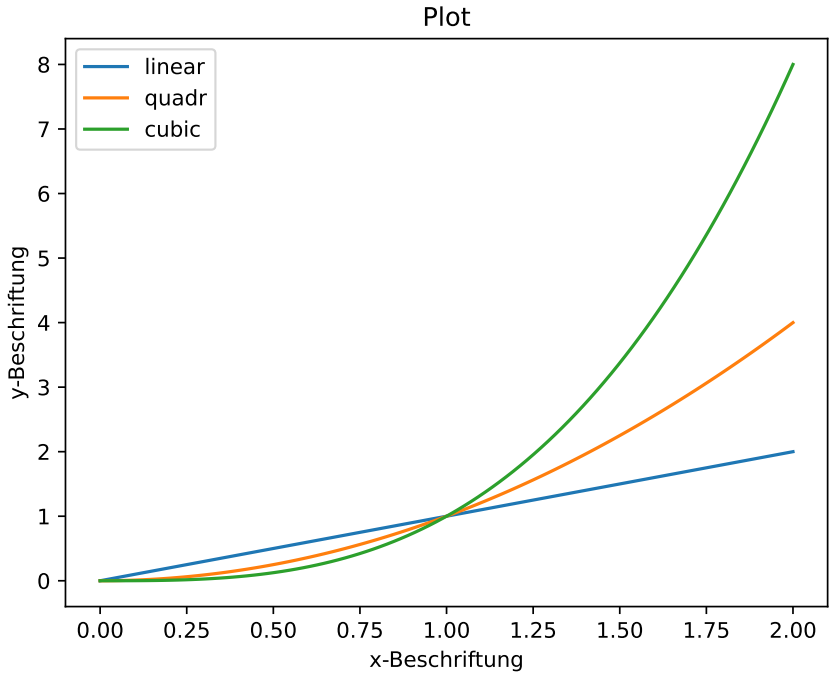

Abbildung 14: pyplot-Graph

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)


plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadr')
plt.plot(x, x**3, label='cubic')
plt.xlabel('x-Beschriftung')
plt.ylabel('y-Beschriftung')
plt.title("Plot")
plt.legend()
plt.show()
```

Listing 5: plt

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)

fig, ax = plt.subplots()
ax.plot(x, x, label='linear')
ax.plot(x, x**2, label='quadr')
ax.plot(x, x**3, label='cubic')
ax.set_xlabel('x-Beschriftung')
ax.set_ylabel('y-Beschriftung')
ax.set_title("Plot")
ax.legend()
plt.show()
```

Listing 6: OOP

6.1.2 *SciPy - Scientific Python*

SciPy ist eine Sammlung von Funktionen für wissenschaftliche Kalkulationen in Python. Sie bietet erweiterte Routinen der linearen Algebra, mathematische Funktionsoptimierung, Signalverarbeitung, spezielle mathematische Funktionen und statistische Verteilungen. Das Modul scikit-learn greift für die Implementierung seiner Algorithmen auf die Funktionssammlung von SciPy zurück.

6.1.3 *Scikit-Learn*

Das scikit-learn-Projekt wird von einer aktiven Userbase konstant weiterentwickelt. Es enthält eine Reihe von Algorithmen für maschinelles Lernen sowie eine umfassende Dokumentation zu den implementierten Algorithmen. Das Projekt ist die populärste Python-Bibliothek für maschinelles Lernen und ist in der Industrie sowie im akademischen Bereich weit verbreitet. Eine Fülle von Tutorien und Code-Beispielen sind online verfügbar, welche den Einstieg in das Modul erleichter.

6.1.4 *NumPy - Numerical Python*

NumPy ist eines der grundlegenden Pakete für wissenschaftliches Rechnen in Python. Es Es enthält Funktionen für mehrdimensionale Arrays, mathematische Funktionen, wie lineare Algebra-Operationen und die Fourier-Transformation, sowie Pseudozufallszahlengeneratoren. In scikit-learn ist das NumPy-Array die grundlegende Datenstruktur. Scikit-learn nimmt Daten in Form von NumPy-Arrays auf. Alle Daten, die das Modul verwendet, müssen in ein NumPy-Array umgewandelt werden. Die Kernfunktionalität von NumPy ist die Klasse ndarray, welche ein mehrdimensionales (n-dimensionales) Array darstellt. Alle Elemente des Arrays müssen hierbei vom gleichen Typs sein. Ein NumPy-Array sieht wie folgt aus:

```
import numpy as np

x = np.array([[1, 2, 3], [4, 5, 6]])
print(f'x:\n{x}')
>>x:
>>[[1 2 3]
  [4 5 6]]
```

6.1.5 *re - Regular expression operations*

Ein regulärer Ausdruck (*englisch: regular expression*) ist eine kompakte Notation zur Darstellung einer Sammlung von Zeichenketten. Reguläre Ausdrücke sind vielseitig einsetzbar, weil bereits ein einzelner regulärer Ausdruck eine unbegrenzte Anzahl von Zeichenketten darstellen kann. Voraussetzung dafür ist, dass sie die Anforderungen des regulären Ausdrucks erfüllen. Im einfachsten Fall ist ein Regex ein Ausdruck (z. B. ein Literalzeichen), optional gefolgt von einem Quantifizierer. Komplexere Regex bestehen aus einer beliebigen Anzahl von Quantifizier-Ausdrücken und können Bedingungen enthalten sowie durch Flags beeinflusst werden. Reguläre Ausdrücke werden in einer Sprache definiert, die sich von Python unterscheidet. Python enthält jedoch das Modul re, mit dem Regex erstellt und verwendet werden können. Reguläre Ausdrücke werden für folgende Zwecke verwendet:

Parsing: Identifikation und Extraktion von Textstücken, die mit bestimmten Kriterien übereinstimmen. Dabei werden Regex zur Erstellung von Ad-hoc-Parsern und auch von herkömmlichen Parsing-Tools verwendet.

Suchen: Auffindung von Teilstrings, die mehr als eine Form haben können, zum Beispiel das Auffinden von »bild.png« , »bild.jpg« , »bild.jpeg« oder »bild.svg« , während es vermieden wird, andere Dateinamen zu finden.

Suchen und Ersetzen: Überall dort, wo reguläre Ausdrücke auf eine Zeichenkette passen, wird diese durch eine neue Zeichenkette ersetzt. Diese Funktionalität wird insbesondere bei der Ersetzung von Signalnamen innerhalb der Verilog-Netzlisten genutzt.

Aufteilen von Zeichenketten: Aufteilen einer Zeichenkette an jeder Stelle, auf die der reguläre Ausdruck zutrifft. Zum Beispiel kann überall dort aufgeteilt werden, wo ein Semikolon oder ein Zeilenumbruch vorkommt.

Validierung: Prüfen, ob ein Textteil bestimmte Kriterien erfüllt.

6.1.6 *shutil - Shell Utilities*

Das shutil-Modul bietet eine Reihe von High-Level-Operationen auf Dateien und Sammlungen von Dateien. Insbesondere werden Funktionen bereitgestellt, welche die Erstellung von Kopien und die Entfernung von Dateien unterstützen.

6.1.7 *argparse - Argument Parser*

Mit dem argparse-Modul ist es möglich, benutzerfreundliche Kommandozeilenschnittstellen zu schreiben. Innerhalb des Programms wird definiert, welche Argumente benötigt werden und über argparse werden diese Argumente analysiert. Zudem generiert das argparse-Modul automatisch Hilfs- und Benutzungsmeldungen und gibt Fehler aus, wenn Benutzer dem Programm ungültige Argumente übergeben.

6.1.8 *configparser - Configuration Parser*

Dieses Modul stellt die Klasse ConfigParser zur Verfügung, die es ermöglicht, Initialisierungsdateien auszulesen. Initialisierungsdateien wiederum erlauben es Python-Programme zu schreiben, die durch den Anwender leicht angepasst werden können.

6.1.9 *subprocess*

Mit dem subprocess-Modul können neue Prozesse erzeugt werden. Gleichzeitig ist es möglich, Verbindungen zu der Ein-/Ausgabe sowie Fehlerleitung dieser Prozesse

herzustellen.

Für den Start von Subprozessen gibt es zwei Ansätze. Zum Einen kann ein objektorientierter Ansatz gewählt werden, bei dem ein Objekt der Klasse Popen erstellt wird. Zum Anderen ist es möglich, die run-Methode direkt aufzurufen. Vorteil beim Aufruf der run-Methode ist ein simplere Syntax, wie sie in Listing 7 zu sehen ist.

```
subprocess.run(['ls'])
```

Listing 7: subprocess.run

Hierbei wird das Programm ls aufgerufen. Für komplexere Aufgaben wie beispielsweise die Interaktion mit einem aufgerufenen Programm ist es nötig ein Objekt der Klasse Popen zu erzeugen.

```
proc = Popen(['ls'], stdout=PIPE, stdin=PIPE, stderr=PIPE )
```

6.1.10 *prettytable*

Das Modul prettytable dient dazu tabellarische Daten in einem übersichtlichen ASCII-Tabellenformat anzuzeigen.

6.1.11 *Scikit-Optimize*

Scikit-Optimize (kurz: skopt) ist eine einfache und effiziente Bibliothek zur Minimierung von teuren und unter Umständen verrauschten Black-Box-Funktionen. Sie implementiert mehrere Methoden zur sequentiellen, modellbasierten Optimierung. Scikit-Optimize zielt darauf ab, universell und einfach einsetzbar zu sein. Die Bibliothek ist auf NumPy, SciPy und Scikit-learn aufgebaut.

7 Bayes'sche Optimierung

Im folgenden Kapitel soll auf die grundlegenden theoretischen Hintergründe der Bayes'schen Optimierung eingegangen werden, um ein Verständnis für die Funktionsweise des Optimierungsalgorithmus zu schaffen. Die Bayes'sche Optimierung stellt ein im maschinellen Lernen häufig genutztes Verfahren zur Auffindung von Optima dar. Diese Optimierungsmethode zeichnet aus, dass die Methode Annahmen über das Problem miteinbeziehen kann, welche das Sampling sowie das Verhältnis zwischen Exploration und Ausnutzung (*engl. exploitation*) des Suchraums lenken. Dies geschieht dadurch, dass das Bayes'sche Optimierungsverfahren ein probabilistisches Modell der Zielfunktion mithilfe von Gauß-Prozessen erzeugt. Dieses Modell wird dann iterativ durch die Hinzunahme von weiteren Datenpunkten präzisiert. Sie wird *bayesianisch* genannt, weil sie auf dem Bayes'schen Theorem aufbaut.

7.1 Bayes'sches Theorem

Das Bayes'sche Theorem ist ein Ergebnis aus der Wahrscheinlichkeitstheorie und liefert einen Zusammenhang zwischen bedingten Wahrscheinlichkeiten.

7.1.1 *Bedingte Wahrscheinlichkeit*

Bedingte Wahrscheinlichkeiten tauchen beispielsweise in mehrstufigen Zufallsexperimenten auf. Anhand des Baumdiagramms in Abbildung 15 lässt sich erkennen, wie bedingte Wahrscheinlichkeiten zu verstehen sind. Das Baumdiagramm kann als

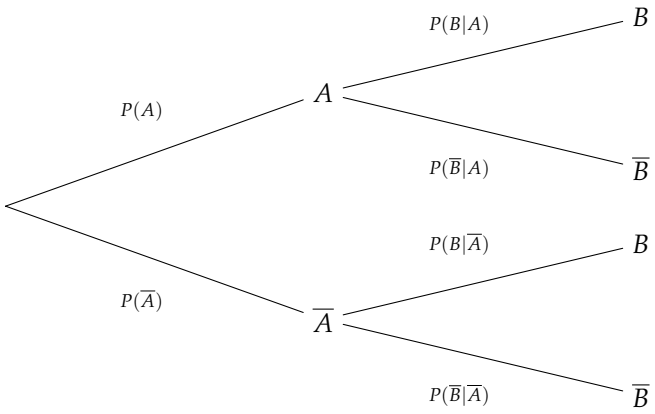

Abbildung 15: Bedingte Wahrscheinlichkeit

zweistufiges Experiment verstanden werden, bei dem auf der ersten Stufe das Ereignis A eintreten (A) oder ausbleiben ($\overline{A}$) kann. Auf der zweiten Stufe kann das Ereignis B entweder eintreten (B) oder ausbleiben ($\overline{B}$). Die Wahrscheinlichkeit mit der A eintritt

oder ausbleibt wird mit $P(A)$ bzw. $P(\overline{A})$ bezeichnet. Die Wahrscheinlichkeit mit der B, unter der Vorraussetzung, dass Ereignis A eingetreten ist, eintritt oder ausbleibt wird mit $P(B|A)$ bzw. $P(\overline{B}|A)$ bezeichnet und bedeutet somit die Wahrscheinlichkeit für B unter der Bedingung A. Hierbei wird $P(A|B)$ als Ereignis und $P(\overline{A}|B)$ als Gegenereignis bezeichnet. Für Ereignis und Gegenereignis ergibt sich der Zusammenhang:

$$P(A|B) + P(\overline{A}|B) = 1 \tag{3}$$

Die allgemeine Definition für die bedingte Wahrscheinlichkeit lautet:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{4}$$

Hierbei ist $P(A \cap B)$ die Wahrscheinlichkeit, dass A und B gemeinsam auftreten.

7.1.2 *Bayes'sches Theorem für zwei Ereignisse*

Wenn man die Ereignisse A und B vertauscht, wird nach der bedingten Wahrscheinlichkeit B in Abhängigkeit von A, d.h. $P(B|A)$ gefragt. Durch Auflösen der Gleichung nach $P(A \cap B)$ erhält man:

$$P(A|B) \cdot P(B) = P(A \cap B) = P(B|A) \cdot P(A) \tag{5}$$

$$\tag{6}$$

Eingesetzt in Formel 4 ergibt sich die Bayes'sche Formel bzw. das Bayes'sche Theorem:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{7}$$

Die Bayes'sche Formel stellt somit den Zusammenhang zwischen den bedingten Wahrscheinlichkeiten $P(A|B)$ und $P(B|A)$ her. Um einen intuitiven Zugang zu dieser Formel zu erlangen, soll anhand eines Beispiels verdeutlicht werden, wie sich die Wahrscheinlichkeit anhand dieser Formel berechnet.

Beispiel: Als Beispiel sei das Würfeln einer 6 als Ereignis A unter der Bedingung angenommen, dass eine gerade Zahl gewürfelt wird. Das Würfeln einer 6 hat die Wahrscheinlichkeit $P(A) = \frac{1}{6}$. Das Würfeln einer geraden Zahl hat die Wahrscheinlichkeit $P(B) = \frac{1}{2}$. Es ist intuitiv ersichtlich, dass die Wahrscheinlichkeit einer 6 unter den Zahlen 2, 4 und 6 gleich $P(A|B) = \frac{1}{3}$ ist.
Die Formel bestätigt diese Annahme, denn:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} = \frac{1 \cdot \frac{1}{6}}{\frac{1}{2}} = \frac{1}{3}$$

7.2 Stochastische Prozesse

In der Mathematik werden zeitlich geordnete, zufällige Vorgänge als stochastische Prozesse bezeichnet. Das heisst, dass ein stochastischer Prozess eine Serie von Zufallsvariablen ist. Er beschreibt die zeitliche oder räumliche Entwicklung zufälliger Systeme. Zur Veranschaulichung ist in Abbildung 16 ein spezifischer stochastischer

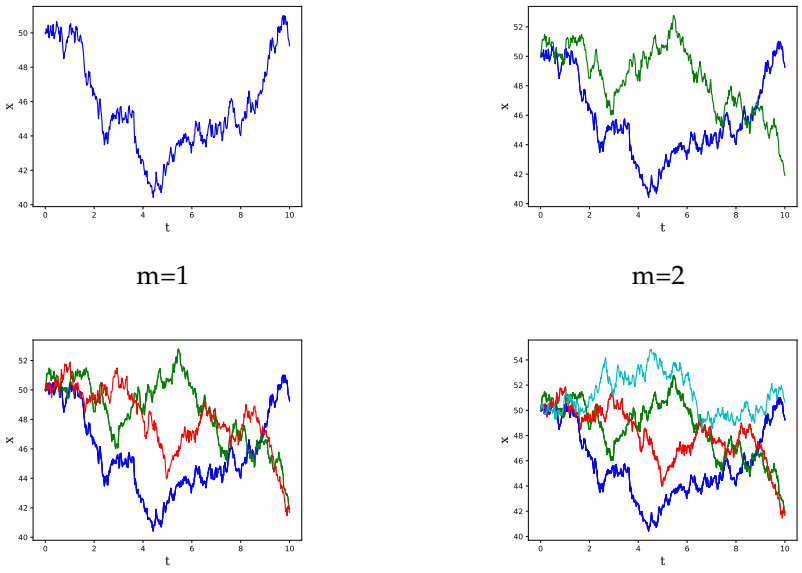

Abbildung 16: Brown'sche Brücke

Prozess dargestellt und zwar die Brownsche Brücke. Abbildung 16 zeigt vier Verläufe dieses stochastischen Prozesses. Die Verläufe werden als Pfade, Realisierungen oder Trajektorien des Prozesses bezeichnet. Mathematisch wird der stochastische Prozess X wie folgt beschrieben:

$$X : (\Omega, \mathcal{F}, \mathbb{P}) \longrightarrow (E, \epsilon)$$

Hierbei ist $(\Omega, \mathcal{F}, \mathbb{P})$ ein Wahrscheinlichkeitsraum. Der Ausdruck (E, ϵ) bezeichnet einen Messraum.

7.2.1 *Wahrscheinlichkeitsraum*

Der Wahrscheinlichkeitsraum ist durch das Tripel $(\Omega, \mathcal{F}, \mathbb{P})$ definiert. Ω ist eine nichtleere Menge, welche als Ergebnismenge bezeichnet wird. Die in ihr enthaltenen Elemente heißen Ergebnisse.

$\mathcal{F}$ ist eine σ-Algebra über der Menge Ω. Die σ-Algebra ist ein Mengensystem, welches die Teilmenge einer Potenzmenge beschreibt. Elemente von $\mathcal{F}$ heißen Ereignisse. Die σ-Algebra $\mathcal{F}$ wird als Ereignisalgebra bezeichnet. Eine Potenzmenge ist die Menge aller Teilmengen einer gegeben Grundmenge. Das heißt für eine Menge $\Omega = \{x_1, x_2\}$ ergibt sich die Potenzmenge $P(\Omega)$ zu $P(\Omega) = \{\varnothing, \Omega, \{x_1\}, \{x_2\}\}$. Hierbei ist hervorzuheben, dass die Potenzmenge die leere Menge $\varnothing$ und die Grundmenge $\Omega = \{x_1, x_2\}$ beinhaltet.

Ω	Menge
$P(\Omega)$	Potenzmenge
$\mathcal{A}$	σ-Algebra

Die Menge $\mathcal{A} \in P(\Omega)$ wird als σ-Algebra bezeichnet, wenn folgende Bedingungen gelten:

1. $\varnothing, \Omega \in \mathcal{A}$
   Die leere Menge und die Grundmenge muss im Mengensystem $\mathcal{A}$ enthalten sein. Die leere Menge muss prinzipiell nicht angegeben werden, da sie aus Bedingung 2 folgt.

2. $A \in \mathcal{A} \Rightarrow A^C := \Omega \backslash A$
   Für das Element $A \in \mathcal{A}$ muss das Komplement von A ebenfalls in der Menge enthalten sein.

3. $A_i \in \mathcal{A}$ für $i \in \mathbb{N} \Rightarrow \cup_{i=1}^{\infty} A_i \in \mathcal{A}$
   Aus den abzählbaren Elementen $A_i \in \mathcal{A}$ muss die Vereinigung gebildet werden können. Diese Vereinigung muss in der σ-Algebra liegen.

Die Elemente $A \in \mathcal{A}$ werden als $\mathcal{A}$-messbare Teilmengen der Grundmenge Ω bezeichnet.

$\mathbb{P}$ ist ein Wahrscheinlichkeitsmaß, welches den Elementen der Ereignisalgebra $\mathcal{F}$ Zahlen zuordnet.

Beispiel: Glücksrad Anhand eines Glücksrads soll der Wahrscheinlichkeitsraum veranschaulicht werden. Die Ergebnismenge, das heißt die möglichen Ergebnisse des Glücksrads sind $\Omega = \{1, 2, 3\}$. Die Ereignisse sind rechts davon abgebildet, sie umfassen alle möglichen Ereignisse des Glücksrads. Die leere Menge, auch wenn sie unmöglich erreicht werden kann, ist ebenfalls angegeben. Das Wahrscheinlichkeitsmaß $\mathbb{P}$ weist diesen Ereignissen ihre Wahrscheinlichkeiten zu.

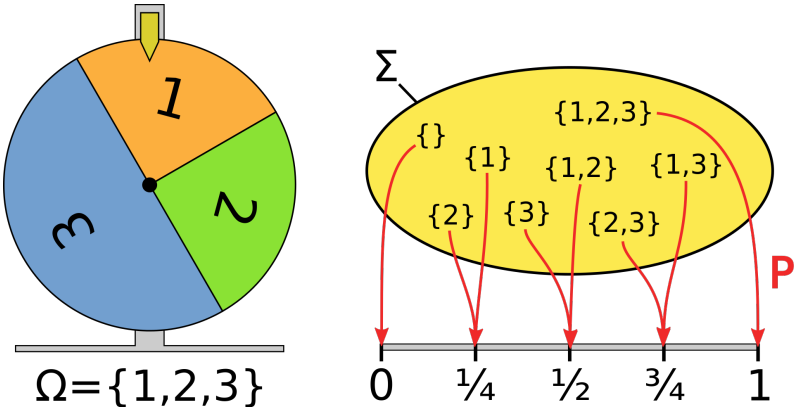

Abbildung 17: Glücksrad [1]

7.2.2 *Erwartungswert*

Der Erwartungswert $E(X)$ beschreibt den Wert, den die Zufallsvariable X im Mittel annimmt. Der Erwartungswertvektor $E(\mathbf{X})$ ist definiert als Erwartungswert des Zufallsvektors $\mathbf{X}$. Jedes Element des Erwartungswertvektors beschreibt somit den Erwartungswert der zugehörigen Dimension.

$$\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{pmatrix}$$

$$\mathrm{E}(\mathbf{X}) = \begin{pmatrix} \mathrm{E}(X_1) \\ \mathrm{E}(X_2) \\ \vdots \\ \mathrm{E}(X_k) \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_k \end{pmatrix}$$

7.2.3 *Varianz-Kovarianzmatrix*

Die Kovarianz ist ein Maß mit dem die wechselseitige Beziehung zwischen Zufallsvariablen angegeben werden kann. Eine Kovarianzmatrix zwischen zwei Variablen A und B ist in Gleichung 8 dargestellt.

$$\begin{pmatrix} \mathrm{Cov}(A,A) & \mathrm{Cov}(A,B) \\ \mathrm{Cov}(B,A) & \mathrm{Cov}(B,B) \end{pmatrix} = \begin{pmatrix} \sigma_A^2 & \sigma_{AB} \\ \sigma_{BA} & \sigma_B^2 \end{pmatrix} = \mathbf{\Sigma} \tag{8}$$

Die Kovarianzen zwischen einer Variable und sich selbst wird als Varianz bezeichnet, das heisst $Cov(A,A) = Var(A)$. Zwischen den Variablen können die Kovarianzen

vertauscht werden, d.h. $Cov(A, B) = Cov(B, A)$. Daraus folgt, dass es sich bei der Kovarianzmatrix um eine symmetrische Matrix handelt. Die Kovarianzfunktion zwischen zwei Variablen A und B, $Cov(A, B)$ ist wie folgt definiert:

$$\mathrm{Cov}(X, Y) := \mathrm{E}\big[(X - \mathrm{E}(X)) \cdot (Y - \mathrm{E}(Y))\big]$$

Eine einfachere Darstellung folgt aus dem Steinerschen Verschiebungssatz:

$$\begin{aligned}
\mathrm{Cov}(X, Y) &= \mathrm{E}\big[(X - \mathrm{E}(X)) \cdot (Y - \mathrm{E}(Y))\big] \\
&= \mathrm{E}\big[(XY - X\,\mathrm{E}(Y) - Y\,\mathrm{E}(X) + \mathrm{E}(X)\,\mathrm{E}(Y))\big] \\
&= \mathrm{E}(XY) - \mathrm{E}(X)\,\mathrm{E}(Y) - \mathrm{E}(Y)\,\mathrm{E}(X) + \mathrm{E}(X)\,\mathrm{E}(Y) \\
&= \mathrm{E}(XY) - \mathrm{E}(X)\,\mathrm{E}(Y)
\end{aligned}$$

Beispiel: Zum Verständnis der Kovarianzmatrix soll ausgehend von Datensatz 1 die Kovarianz berechnet werden. Für die Kovarianzmatrix 8 ergibt sich folgende

A	B
3	1
-1	-1
2	4

Tabelle 1: Beispieldatensatz für Kovarianzberechnung

Form:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \mathrm{Var}(A) & \mathrm{Cov}(A, B) \\ \mathrm{Cov}(B, A) & \mathrm{Var}(B) \end{pmatrix} \tag{9}$$

Mit

$$\begin{aligned}
\mathrm{Cov}(A,B) &= \mathrm{Cov}(B,A) \\
&= E(AB) - E(A)E(B) \\
&= \underbrace{\left(\frac{3\cdot 1 + (-1)\cdot(-1) + 2\cdot 4}{3}\right)}_{E(AB)} - \underbrace{\frac{3+(-1)+2}{3}}_{E(A)} \cdot \underbrace{\frac{1+(-1)+4}{3}}_{E(B)} \\
&= \frac{20}{9}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Var}(A) &= \mathrm{Cov}(A,A) \\
&= E(A^2) - E(A)^2 \\
&= \underbrace{\frac{3^2 + (-1)^2 + 2^2}{3}}_{E(A^2)} - \underbrace{\left(\frac{3+(-1)+2}{3}\right)^2}_{E(A)^2} \\
&= \frac{26}{9}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Var}(B) &= E(B^2) - E(B)^2 \\
&= \frac{38}{9}
\end{aligned}$$

Daraus folgt die Kovarianzmatrix:

$$\Sigma = \begin{pmatrix} \frac{26}{9} & \frac{20}{9} \\ \frac{20}{9} & \frac{38}{9} \end{pmatrix}$$

7.3 Multivariate Gauß-Verteilung

Die Gauß-Verteilung bildet die Grundlage Gaußscher Prozesse. Hierbei ist besonders der multivariate Fall von Bedeutung, bei dem jede Variable normalverteilt ist und ihre gesamte Verteilung der Gaußschen Verteilung entspricht. Bei multivariaten Gaußschen Verteilungen wird der Erwartungswert, im Gegensatz zu univariaten Verteilungen, mit dem Erwartungswertvektor μ angegeben. Die Kovarianz der Verteilung wird durch die Kovarianzmatrix Σ definiert. Die Definition der beiden Begriffe finden sich in Kapitel 7.2.2 und 7.2.3.

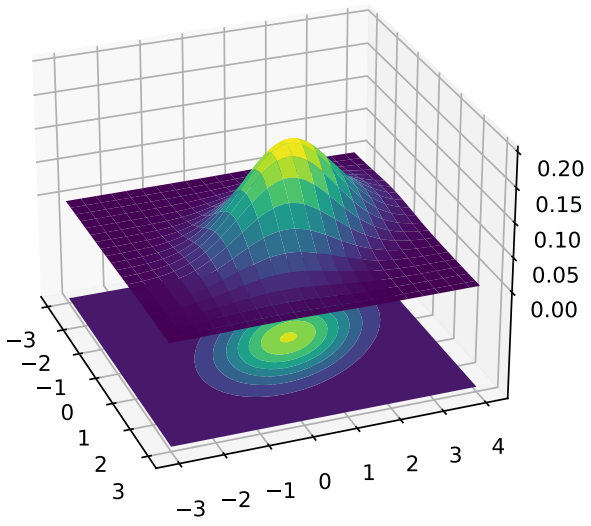

Abbildung 18: Bivariate Gaussverteilung

Unter der Bedingung

$$X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma)$$

spricht man davon, dass X einer Normalverteilung folgt. Mathematisch wird die Verteilung von X mit dem Symbol »∼« kenntlich gemacht. Zur Veranschaulichung ist in Abbildung 18 eine bivariate Gaussverteilung dargestellt. Anhand der Darstellung können die Parameter der Verteilung veranschaulicht werden. Die Kovarianzmatrix Σ beeinflusst die Form der Wölbung. Der Erwartungswertvektor μ definiert den Mittelpunkt.

7.4 Gauß-Prozess

Gauß-Prozesse sind stochastische Prozess, bei denen jede endliche Teilmenge von Zufallsvariablen mehrdimensional normalverteilt ist. Somit sind sie eine Generalisierung der mehrdimensionalen Normalverteilung. Gauß-Prozesse können durch Erwartungs-

wertfunktion $m(x)$ und Kovarianzmatrix $k(x, x')$ vollständig und eindeutig bestimmt werden. Somit wird der Gauß-Prozess $\mathcal{GP}$ folgendermaßen notiert:

$$f(x) \sim \mathcal{GP}(m(x),\ k(x, x')) \tag{10}$$

Die Ähnlichkeit zur Gauß-Verteilung ist ersichtlich (vgl. Definition 11).

$$X \sim \mathcal{N}(\mu,\ \sigma^2) \tag{11}$$

Während die Gauß-Verteilung einer Verteilung von Zufallsvariablen mit entsprechendem Erwartungswert und Varianz entspricht, sind Gauß-Prozesse analog dazu eine Verteilung von Funktionen, die einem gegebenen Satz von Punkten liefern.

Gauß-Prozesse ermöglichen es, Vorhersagen über Daten zu treffen, indem Vorwissen miteinbezogen wird. Der Anwendungsbereich ist unter anderem die Anpassung einer Funktion an gegebene Daten. Dies wird als Regression bezeichnet und kommt beispielsweise in der Robotik oder bei der Vorhersage von Zeitreihen zum Einsatz. Gaußsche Prozesse sind jedoch nicht auf die Regression beschränkt, sondern können auch für Klassifizierungs- und Clustering-Aufgaben verwendet werden.
Für einen gegebenen Satz von Punkten gibt es potenziell unendlich viele Funktionen, die zu den Daten passen. Dieses Problem lösen Gaußsche Prozesse, indem sie jeder Funktion, die durch die gegebenen Punkte verläuft, eine Wahrscheinlichkeit zuweisen. Der Mittelwert dieser Wahrscheinlichkeitsverteilung stellt dann die wahrscheinlichste Charakterisierung der Daten dar. Darüber hinaus ist es möglich, durch die Verwendung einer probabilistischen Methode die Zuverlässigkeit der Vorhersage in das Regressionsergebnis einzubeziehen.

Bei Optimierungsanwendungen wird für einen Satz von Trainingsdaten die zugrunde liegende Verteilung durch Gauß-Prozesse erzielt. Im Folgenden werden die Testdaten mit X und die Trainingsdaten mit Y notiert. Somit ist das Ziel, die zugrunde liegende Verteilung von X zusammen mit Y als multivariate Gauß-Verteilung zu modellieren. Das bedeutet, dass die gemeinsame Wahrscheinlichkeitsverteilung $P_{X,Y}$ den Raum der möglichen Funktionswerte für die Funktion, die approximiert werden soll, aufspannt. Die gemeinsame Verteilung von Test- und Trainingsdaten hat hierbei $|X| + |Y|$ Dimensionen. Um eine Regressionsanalyse der Trainingsdaten vorzunehmen, wird das Problem als Bayesianische Inferenz behandelt. Die Bayesianische Inferenz ist eine Methode zur Aktualisierung der Wahrscheinlichkeit einer Annahme, wenn neue Informationen verfügbar sind. Im Falle von Gaußschen Prozessen bezieht sich dies auf die bedingte Wahrscheinlichkeit $P_{X,Y}$. Aus den Eigenschaften der Gauß-Verteilung geht hervor, dass diese Wahrscheinlichkeit ebenfalls normalverteilt ist.

7.4.1 *Kernel*

Um die Gauß-Verteilung zu bestimmen, müssen der Erwartungswert $m(x)$ und die Kovarianzfunktion $k(x, x')$ definiert werden (vgl. Gleichung 10). Bei der Bestimmung dieser Parameter liegt das Augenmerk auf der Kovarianzfunktion. Für den Erwartungswert wird $\mu = 0$ angenommen, auch für Fälle in denen $\mu \neq 0$ ist. Dies hat den Grund, dass Daten zentriert werden können, was die Konfiguration von μ vereinfacht.

Somit verbleibt nur die Kovarianzmatrix Σ für die Definition des Prozesses. Die Kovarianzmatrix wird durch die Evaluierung des Kernels k ermittelt, welcher auch als Kovarianzfunktion bezeichnet wird. Der Kernel (vgl. Gleichung 12) bestimmt aus zwei Punkten $x, x' \in \mathbb{R}^n$ ein Gleichheitsmaß dieser Punkte in Form eines Skalars.

$$k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}, \quad \Sigma = \mathrm{Cov}(X, X') = k(x, x') \tag{12}$$

Wird diese Gleichung für alle paarweisen Kombinationen von Punkten der Testdaten evaluiert, gelangt man zur Kovarianzmatrix. Bei der Optimierung der Syntheseergebnisse wird der Matérn Kernel genutzt. Der Matérn-Kernel ist eine Generalisierung des Radial-Basis-Function-Kernels[2]. Der RBF-Kernel hat die Form:

$$k_{RBF}(x, x') = \sigma^2 e^{\left(-\frac{(x-x')^2}{2l^2}\right)} \tag{13}$$

Die Variable l stellt hierbei die Längenskala dar. Sie ist ein Maß für die Länge der Wellen der Funktion.

7.4.2 *A-Priori-Verteilung*

Die a-priori-Verteilung (engl. *Prior Distribution*) P_X ist die Verteilung, von der ausgegangen wird, wenn keine Trainingsdaten evaluiert wurden. Abbildung 19 veranschaulicht diese Verteilung. Die markierte Fläche stellt hierbei die Varianz dar, innerhalb derer sich die Funktionsverteilung ergibt. Da noch keine Punkte existieren, ergeben sich die Funktionen als zufällige Trajektorien innerhalb dieser.

7.4.3 *A-posteriori-Verteilung*

Ausgehend von der a-priori-Verteilung liegt die Frage nahe, inwiefern sich die Verteilung verändert, wenn Trainingsdaten evaluiert werden. Die Methode der Bayesschen Inferenz besagt, dass neue Informationen miteinbezogen werden können, so dass diese zur a-posteriori-Verteilung (engl. *Posterior Distribution*) $P_{X|Y}$ führen. Zunächst wird dafür die gemeinsame Verteilung $P_{X,Y}$ zwischen den Testpunkten X

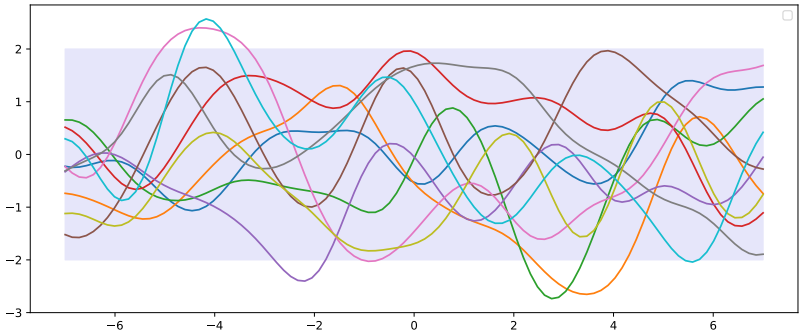

Abbildung 19: A-priori-Verteilung, quadratisch-exponentieller Kernel

und den Trainingspunkten Y gebildet. Das Ergebnis ist eine multivariate Gaußsche Verteilung mit den Dimensionen $|Y| + |X|$.

7.5 Acquisition function

Die Acquisition function ist ein Mechanismus, der zur Umsetzung des Verhältnisses zwischen Exploration und Exploitation bei einer Bayes-Optimierung genutzt wird. Die Acquisition function zielt darauf ab, die Suche nach dem Optimum zu Punkten mit potenziell niedrigen Werten der Zielfunktion zu leiten, entweder weil die Vorhersage von $f(x)$, basierend auf dem probabilistischen Surrogate-Modell niedrig ist, die Unsicherheit basierend auf demselben Modell hoch ist oder weil beides zutrifft.

Exploitation bedeutet, sich auf den Bereich zu fokussieren, der mehr Chancen zur Verbesserung der aktuellen Lösung (in Bezug auf das aktuelle Surrogate-Modell) hat. Exploration bedeutet, sich in weniger erforschte Bereiche des Suchraums zu bewegen, in denen die Vorhersagen auf Basis des Surrogate-Modell unsicherer sind.

7.6 Surrogate-Modell

Ein Surrogate-Modell ist eine technische Methode, die genutzt wird, wenn ein gesuchtes Ergebnis nicht direkt ermittelt werden kann. In diesem Fall wird ein Modell des Ergebnisses verwendet. Eine Vielzahl technischer Probleme erfordert Experimente und Simulationen, um Ziel- und Einschränkungsfunktionen als Funktion von Eingangsvariablen zu bewerten. Bei vielen Problemen kann eine Simulation jedoch einen erheblichen Zeitaufwand mit sich bringen.

Eine Möglichkeit, diese Belastung zu verringern, besteht in der Ermittlung von Näherungsmodellen, die als Surrogate-Modelle bekannt sind und das Verhalten des

Simulationsmodells so genau wie möglich nachahmen, während sie mit geringerem Aufwand zu bewerten sind. Bei der Erstellung des Modells wird die genaue, innere Funktionsweise des Simulationscodes nicht als bekannt vorausgesetzt, lediglich das Input-Output-Verhalten ist hierbei von Interesse. Ein Modell wird auf der Grundlage der Modellierung der Reaktion des Simulators auf eine begrenzte Anzahl von Datenpunkten erstellt. Dieser Ansatz ist auch als Verhaltensmodellierung oder Black-Box-Modellierung bekannt. Wenn nur eine einzige Designvariable betroffen ist, wird der Prozess als Kurvenanpassung bezeichnet.

Obwohl die Verwendung von Surrogate-Modellen anstelle von Experimenten und Simulationen in der Konstruktion üblicher ist, kann die Surrogate-Modellierung auch in vielen anderen Bereichen der Wissenschaft angewendet werden, in denen teure Experimente und/oder Funktionsauswertungen vorhanden sind.

Teil II
Zellcharakterisierung

8 Liberty-Format

Das Ziel der Charakterisierung ist es eine Bibliothek mit Modellen zu erstellen, die das Verhalten von Zellen abbildet. Die Beschreibung dieser Bibliotheken erfolgt im Liberty-Format. Die Parameter, welche zur Charakterisierung der Zellen verwendet werden, sind Timing, Leistungsaufnahme, Input-Output-Charakterisierung und Noise bei gegebenen Bedingungen in Bezug auf Prozessvariation, Versorgungsspannung und Temperatur.
Das folgende Kapitel soll eine Übersicht über die Art und den Aufbau einer Bibliothek im Liberty-Format geben.

8.1 Aufbau

Der allgemeine Aufbau einer Liberty-Bibliothek ist in Abbildung 20 dargestellt.

8.1.1 *Statements*

Statements sind die Bausteine einer Bibliothek. Alle Bibliotheksinformationen lassen sich einer der folgenden Kategorien zuordnen:

Gruppen Gruppen sind eine Ansammlung von Statements, die verschiedene Objekte in der Bibliothek beschreiben. Zu diesen Objekten gehören beispielsweise Zellen, Pins und Timing-Arcs. Der Inhalt einer Gruppe wird durch geschweifte Klammern gekennzeichnet.

Attribute Ein Attributs-Statement dient der Beschreibung einer Charakteristik eines Objekts in der Bibliothek. Die Zugehörigkeit zu Objekten in der Bibliothek erhalten Attribute durch die Zuordnung zu einer Gruppe. Attribute lassen sich in zwei Kategorien unterteilen: einfache und komplexe Attribute. Einfache Attribute haben genau einen Wert, der ihnen zugewiesen wird. Komplexe Attribute dagegen haben mehrere Parameter.

Definitionen Mit define lassen sich *einfache* Attribute definieren.

8.2 Technology Library

Eine Liberty-Bibliothek hat genau eine Technology Library Gruppe. Sie beinhaltet Attribute, die für die gesamte Bibliothek gültig sind. Der Anfang der Technology Library wird mit der Angabe `library (library_name) {...}` gekennzeichnet.

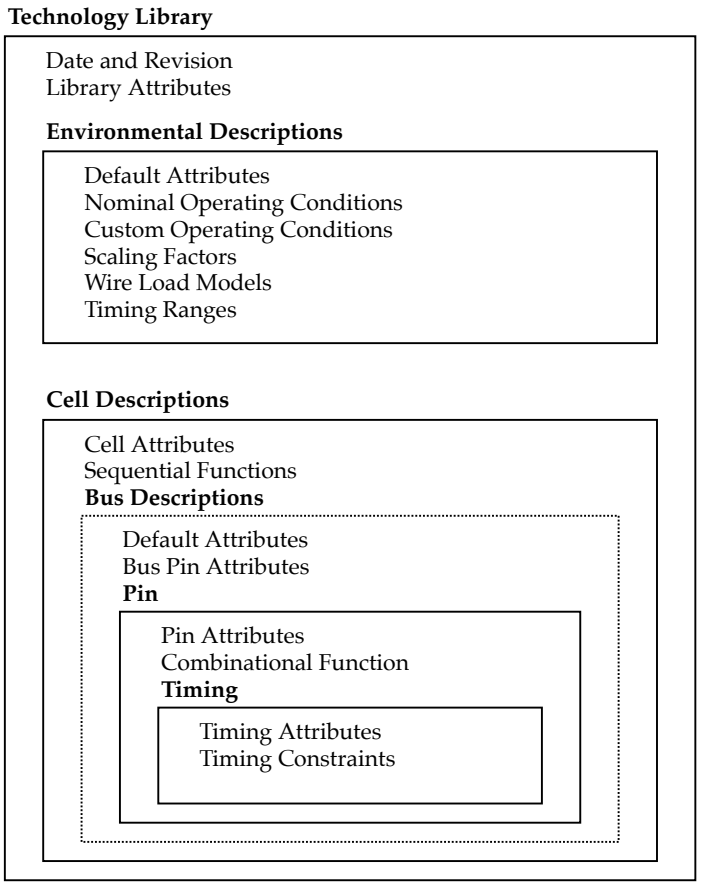

Abbildung 20: Aufbau von Liberty Files

1. Date and Revision
   Die Attribute Date und Revision dienen der Organisation der Bibliothek. Auf die Funktionsweise der Zellen haben sie keinen Einfluss.

   Date

   Revision

2. Library Attributes

   technology Dieses Attribut identifiziert die in der Bibliothek verwendete Technologie. Der einzig gültige Wert ist cmos. Das technology-Attribut muss das erste definierte Attribut sein und an den Anfang der Library gestellt werden.

```
technology (name);
→ cmos
```

delay_model Das Delay Model definiert, nach welchem Ansatz die Verzögerungszeiten innerhalb der Schaltung simuliert werden sollen. Eine Erklärung zu den Modellen ist in Kapitel 9 zu finden. Mögliche Werte sind die folgenden:

```
delay_model : value;
→  generic_cmos | table_lookup |
    piecewise_cmos | polynomial | dcm
```

bus_naming_style Dieses Attribut definiert die Namenskonvention für Busse in der Bibliothek.

```
bus_naming_style : "string";
```

Einheitendefinition Die hier definierten Werte dienen als Einheiten für die gesamte Bibliothek.

```
time_unit : "unit";
→  1ps | 10ps | 100ps | 1ns (default)

voltage_unit : "unit";
→  1mV | 10mV | 100mV | 1V (default)

current_unit : "unit";
→  1uA | 10uA | 100uA | 1mA | 10mA | 100mA | 1A

pulling_resistance_unit : "unit";
→  1ohm | 10ohm | 100ohm | 1kohm

capacitive_load_unit (value(float), unit)
→  value : floating-point number
→  unit : ff | pf

leakage_power_unit : value;
 →  1mW | 100mW | 10mW | 1mW | 100nW |
     10nW | 1nW | 100pW | 10pW | 1pW
```

8.3 Environmental Descriptions

Syntax

1. Default Attributes
   Unter Default Attributes werden Werte gesetzt, die für alle Zellen beziehungsweise für alle Pins der Bibliothek gelten. Hierbei werden die unter Einheitendefiniton gewählten Werte genutzt.

   default_inout_pin_cap: Legt eine Vorgabe für die Kapazität aller bidirektionalen Pins der Bibliothek fest.

   default_input_pin_cap: Legt eine Vorgabe für die Kapazität aller Eingangspins in der Bibliothek fest.

   default_output_pin_cap: Legt eine Vorgabe für die Kapazität aller Ausgangspins in der Bibliothek fest.

   default_fanout_load: Legt eine Vorgabe für die kapazitive Last per Fanout fest.

   default_max_fanout: Setzt eine Voreinstellung für max_fanout für alle Ausgangspins in der Bibliothek

   default_max_transition: Legt eine Voreinstellung für max_transition für alle Ausgangspins in der Bibliothek fest.

```
default_inout_pin_cap : value(float) ;
default_input_pin_cap : value(float) ;
default_output_pin_cap : value(float) ;
default_fanout_load : value(float) ;
default_max_fanout : value(float) ;
default_max_transition : value(float) ;
```

2. Nominal Operating Conditions

3. Custom Operating Conditions

4. Scaling Factors
   Scaling Factors sind Multiplikatoren, die Bibliothekswerte unter Berücksichtigung von Prozess-, Temperatur- und Spannungsänderungen skalieren. Sollte das Non Linear Delay Model (NLDM) gewählt werden, sind hier spezifische Attribute für das Delay-Model zu wählen.

5. Wire Load Models
   Die folgenden Attribute werden auf Bibliotheksebene genutzt, um die Standardeinstellung für den wire_load-Parameter festzulegen.

   default_wire_load: Bestimmt die genutzte wire_load-Gruppe.

```
default_wire_load : wire_load_name;
→  wire_load_name : WL1
```

default_wire_load_capacitance: Gibt einen Wert für die Standard Leitungskapazität an.

default_wire_load_resistance: Gibt einen Wert für den Standard Leitungswiderstand an.

default_wire_load_area

6. Timing Ranges

7. Weitere environment-Attribute
   Zur Zuweisung von Attributen auf library-Ebene können die Attribute default_operating_conditions und default_connection_class genutzt werden.

   default_operating_conditions Weist eine zuvor definierte Konfiguration von Attributen den Voreinstellungen zu. Der Gebrauch dieser Funktion ist in Listing 10 dargestellt.

```
operating_conditions(OP){
. . .
}
default_operating_conditions : OP1;
```

Listing 8: Zuweisung von default_operating_conditions

default_connection_class: Weist wie default_operating_conditions Werte der connection_class zu.

8.4 Cell Descriptions

Syntax Die Zellbeschreibung stellt den Großteil einer Liberty-Bibliothek dar. In ihr finden sich Informationen zu Fläche, Funktion und Timing der Komponenten. Über `cell (name)` wird die cell-Gruppe für die jeweilige Komponente innerhalb der Bibliothek definiert.

1. Cell Attributes

   area Das area-Attribut legt die Fläche der Zelle fest.

   cell_footprint Über cell_footprint lassen sich Zellen gruppieren, die das gleiche Layout haben und austauschbar im Zuge der Schaltungsoptimierung sind.

2. Sequential Functions
   Sequentielle Elemente müssen in der cell-Gruppe definiert werden. Hierfür stehen die Attribute ff und latch zur Verfügung.

3. Bus Descriptions
   Ein Bus kann über die type- oder bus-Gruppe definiert werden. Soll die type-Gruppe genutzt werden, muss diese auf library-Ebene definiert werden.

   (a) Default Attributes

   (b) Bus Pin Attributes

   (c) Pin
   Die Pin-Gruppe dient der Definition von Timings und Funktionsattributen für jeden einzelnen Pin der Zelle. Sie wird initiiert durch `pin (pin_name)`. Es ist möglich Pins mit gleichen Eigenschaften zusammenzufassen. Hierfür können Pins durch Kommata getrennt angeben werden.

```
pin(pin1, pin2){
.
.
.
}
```

   i. Pin Attributes

   capacitance Über capacitance wird die kapazitive Last eines Pins definiert. Die Einheit dieses Werts muss innerhalb der library-Gruppe definiert werden.

   direction Direction dient dazu , die Richtung des Pins zu kennzeichnen.

   ii. Combinational Function
   Kombinatorische Funktionen werden über das function-Attribut definiert (vgl. Tabelle 2). Listing 9 zeigt ein Beispiel für die Modellierung eines AND-Gatters mit zwei Eingängen.

```
cell (AND02){
    pin (IN1){
        direction : input;
    }
    pin (IN2){
        direction : input;
    }
    pin (OUT){
    function : "IN1 * IN2";
```

```
        }
    }
```

Listing 9: Beispiel: AND02-Gatter

Operator	Beschreibung
’	Invertiere vorangegangen Ausdruck
!	Invertiere folgenden Ausdruck
∧	Exklusiv-Oder XOR
*	UND
&	UND
space	UND
+	ODER
|	ODER
1	logische 1
0	logische 0

Tabelle 2: Boole’sche Operatoren im function-Attribut

iii. Timing Siehe 8.5

A. Timing Attributes

B. Timing Constraints

8.5 Timing-Gruppe

Die Timing-Gruppe dient dazu, die Timing-Informationen der Pins zu definieren. Die Informationen, die in der timing-Gruppe stehen, sind abhängig davon, welches delay_model in der library-Gruppe ausgewählt wurde (siehe 2). In Kapitel 9 werden Delay Models und ihre Parameter beschrieben. Unabhängig von dem gewählten Delay Model sieht die Syntax der timing-Gruppe allgemein so aus:

```
timing(){
related_pin : ;
.
.
.
}
```

Listing 10: timing-Gruppe

8.6 Aufbau sequentieller Logik

Bei der Beschreibung der sequentiellen Logikelemente, das heisst Flipflops und Latches, bietet das Liberty-Format zwei Herangehensweisen. Zum Einen ist es möglich

die Beschreibung mittels Gruppen vorzunehmen. Bei der Nutzung von Gruppen stehen Attribute zur Verfügung mit denen das Verhalten von Flipflops und Latches abgebildet werden kann. Zum Anderen kann die Beschreibung sequentieller Logik mithilfe von Statetables erfolgen. Über statetables lässt sich das Verhalten von sequentieller Logik flexibler beschreiben, da nicht auf vordefinierte Attribute zurückgegriffen werden muss.

8.6.1 *Gruppen*

Zur Beschreibung mittels Gruppen stehen die `ff`- und `latch`-Gruppe zur Verfügung.

ff-Gruppe Listing 11 zeigt die Syntax zur Definition der ff-Gruppe, sowie deren Attribute.

```
ff ( variable1, variable2 ) {
    clocked_on : "pin" ;
    next_state : "pin" ;
    clear : "pin" ;
    preset : "pin" ;
    clear_preset_var1 : L/H/N/T/X ;
    clear_preset_var2 : L/H/N/T/X ;
}
```

Listing 11: ff-Gruppe

Die ff-Gruppe wird durch `ff (variable1, variable2)` initiiert. Die Variablen 1 und 2 stehen für die Outputs des Flipflops. Variable 1 gibt den Status des nichtinvertierten und Variable 2 den des invertierten Ausgangs des Flipflops an. Die Variablen können umbenannt werden, solange der Name nicht identisch mit einer Pinbezeichnung ist.

Attribute

clocked_on Über clocked_on wird angegeben, welches Signal dem Flipflop als Taktsignal dient.

```
    clocked_on : clk ; /* rising edge */
    clocked_on : clk'; /* falling edge */
```

Listing 12: clocked_on-Attribut

Listing 12 zeigt die Syntax für rising_edge- beziehungsweise falling_edge-Konfiguration.

next_state Das next_state-Attribut gibt den Wert an, den der Ausgang des Flipflops bei der nächsten steigenden beziehungsweise fallenden Flanke annimmt.

clear Das clear-Attribut definiert das Verhalten des clear-Eingangs. Für einen active-low clear muss der Pin negiert angegeben werden.

preset Das Preset-Attribut definiert den Wert des Preset-Eingangs. Der Preset-Eingang wird auch als Set bezeichnet. Wenn er aktiv ist, wird der Ausgang des Flipflops »high« (Q=1) gesetzt.

clear_preset_var1 Bestimmt den Wert, den Variable 1 (siehe 8.6.1) annimmt, wenn clear und preset aktiv sind.

clear_preset_var2 Bestimmt den Wert, den Variable 2 (siehe 8.6.1) annimmt, wenn clear und preset aktiv sind.

Die Attribute clear_preset_var1 und clear_preset_var1 können die Werte L, H, N, T oder X annehmen. Die Bedeutung der Werte ist in Tabelle 3 aufgeführt.

Wert	Bedeutung
L	Low '0'
H	High '1'
N	No Change: der bestehende Wert bleibt erhalten
T	Toggle: Ändert den Wert von 0 auf 1 und 1 auf 0. Sollte X anliegen erfolgt keine Änderung.
X	Unknown

Tabelle 3: Werte für clear_preset_var

latch-Gruppe Die latch-Gruppe wird über `latch (variable1, variable2)` initiiert. Variable 1 steht für den nichtinvertierten und Variable 2 für den invertierten Ausgang der Zelle. Die Variablennamen können geändert werden, solange sie nicht dem Namen eines Pins entsprechen.

```
latch (variable1, variable2) {
    enable : "pin" ;
    data_in : "pin" ;
    clear : "pin" ;
    preset : "pin" ;
    clear_preset_var1 : L/H/N/T/X ;
    clear_preset_var2 : L/H/N/T/X ;
}
```

Listing 13: latch-Gruppe

Attribute

enable Muss zusammen mit data_in verwendet werden. Gibt den Pin für den enable-Eingang an.

data_in Muss zusammen mit enable verwendet werden. Gibt den Pin für den data_in-Eingang an.

clear Das clear-Attribut definiert das Verhalten des clear-Eingangs. Für einen active-low clear muss der Pin negiert angegeben werden.

preset Definiert den Wert des Preset-Eingangs. Der Preset-Eingang wird auch als Set bezeichnet. Wenn er aktiv ist, wird der Ausgang des Latch »high« (Q=1) gesetzt.

clear_preset_var1 Bestimmt den Wert, den Variable 1 annimmt, wenn clear und preset aktiv sind. Mögliche Werte sind in Tabelle 3 aufgeführt.

clear_preset_var2 Bestimmt den Wert, den Variable 2 annimmt, wenn clear und preset aktiv sind. Mögliche Werte sind in Tabelle 3 aufgeführt.

8.6.2 *Statetables*

Durch die Nutzung von Statetables zur Beschreibung sequentieller Elemente ist es möglich, den Wert der Ausgänge abhängig der Eingangsbelegung anzugeben. Die Beschreibung erfolgt über eine Wahrheitstabelle. Ein Ausschnitt einer Statetable ist in Listing 14 zu sehen.

```
0  statetable(" R    S    D    EN    C  ", " IQ "){
     table :"   H    -    -    -     -  :    -       : L ,\
                L    -    H   H      R  :    -       : H ";
   }
```

Listing 14: Statetable

Zeile 0 der Statetable führt zunächst die Namen der Eingangsspins der Zelle auf (R, S, D, EN, C). Nach dem Komma stehen die Ausgangsspins. In diesem Fall handelt es sich nur um den Ausgangspin IQ. Unter den Eingangspins (R, D, S, D, EN, C) werden Eingangskombinationen eingetragen. In Zeile 1 findet sich unter dem Reset-Eingangspin der Wert H. Die anderen Eingangspins in dieser Zeile stehen auf »don't care«. Das bedeutet, dass dieser Zustand allein durch ein High-Signal am Reset-Pin ausgelöst wird und die anderen Eingänge in diesem Zusammenhang irrelevant sind. Tritt diese Kombination von Eingangssignalen ein, wird über die Spalte IQ der Ausgangswert der Zelle definiert. Die erste Angabe in der Spalte IQ steht dabei für

den aktuellen Wert des Ausgangsspins. Die Angabe nach dem Doppelpunkt definiert den Wert, den der Pin bei gegebener Eingangskombination annimmt. In Listing 14 ist zu erkennen, dass der erste Wert hier in Zeile 1 don't care ist und der zweite Wert Low. Das bedeutet für die Zelle, dass, ungeachtet in welchem Zustand sich die Zelle befindet, bei einem High-Pegel am Reset-Eingang der Ausgang der Zelle auf Low gesetzt wird. Die möglichen Werte für die Ausgangsspins sind in Tabelle 5 dargestellt. Die möglichen Werte für die Eingangspins unter »table« sind in Tabelle 4 dargestellt.

Wert	Bedeutung
L	Low
H	High
-	Don't care
L/H	Kurzform Low/High
H/L	Kurzform High/Low
R	Rising edge
F	Falling edge
˜R	Nicht rising edge
˜F	Nicht falling edge

Tabelle 4: Mögliche Werte von Eingangspins in statetables

Werte	Bedeutung
L	Low
H	High
-	Output is not specified
L/H	Expands to both L and H
H/L	Expands to both H and L
X	Unknown
N	No event from current value

Tabelle 5: Mögliche Werte für Ausgangspins in statetables

In Zeile 1 von Listing 14 ist zu sehen, dass für den Pin R die Belegung H angegeben ist. Die weiteren Eingangsspins sind auf don't care gesetzt, wie die aktuelle Belegung des Ausgangsspins. Die Belegung, die danach erreicht werden soll, ist L ('0'). Diese Definition realisiert ein asynchrones, active-high Reset. Asynchron, da der Clock-Eingang auf don't care liegt und active-high, weil der Reset Pin high ('1') sein muss. Der aktuelle Wert des Ausgangsspins ist auf don't care gesetzt, weil der Ausgang unabhängig von der aktuellen Belegung auf low wechseln soll, sobald der Reset aktiv ist.
Zeile 2 definiert das Verhalten der Zelle, wenn der Reset logisch 0 ist. Hierbei wird das Verhalten der Zelle bei dem Zustand, wenn der Daten- und Enable-Eingang auf High gesetzt ist, beschrieben, was durch ein 'H' gekennzeichnet ist. Unter Clock wird die Angabe 'R' gemacht, damit der Wert am Dateneingang bei einer steigenden Taktflanke

auf den Ausgang gelegt wird.

Nutzung von Kurzformen Zur Verkürzung der Zuweisungen ist es möglich, die Low- und High-Werte mit L/H beziehungsweise H/L zu verkürzen (vgl. Tabelle 4). Die Verwendung der Kurzformen zeigt an, dass die Werte vor beziehungsweise nach dem Schrägstrich einander zugewiesen werden.

9 Delay Models

Die Konfiguration des Delay Models erlaubt es Verzögerungszeiten in die Synthese miteinzubeziehen. Zu diesem Zweck stehen die in Tabelle 6 aufgeführten Modelle zur Verfügung.

Name	delay_model
Linear	generic_cmos
Nonlinear	table_lookup
Piecewise Linear	piecewise_cmos
Scalable Polynomial	polynomial
Delay Calculation Module	dcm

Tabelle 6: Delay Models

9.1 Linear Delay Model

Die Gesamtverzögerungszeit berechnet sich im linearen Delay Models aus:

$$D_{ges} = D_I + D_S + D_C + D_T \tag{14}$$

Die Summanden des Terms 14 haben folgende Bedeutung:

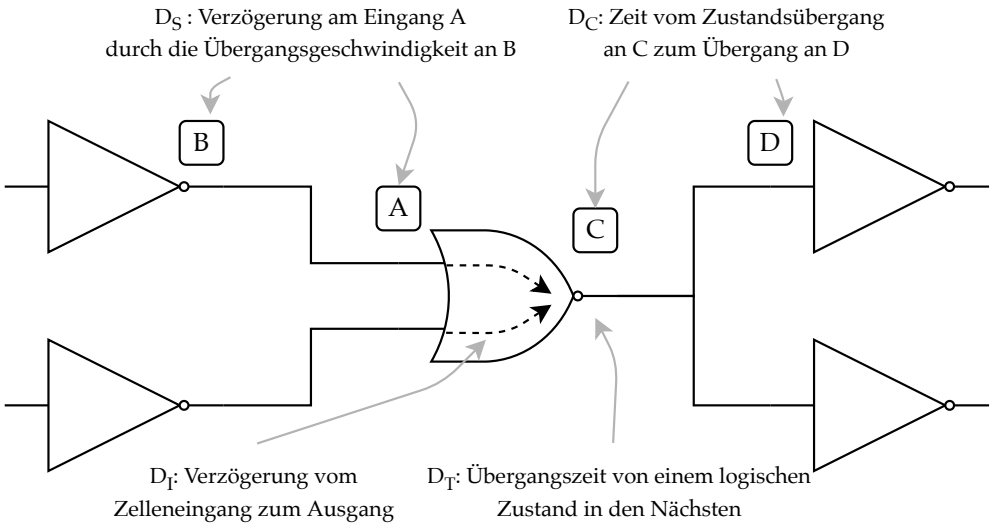

Abbildung 21: Linear Delay Model

Intrinsisches Delay D_I Das intrinsische Delay bezeichnet die Verzögerungszeit des Elements, wenn es keine Last treibt.

Slope Delay D_S Das Slope Delay bezeichnet die Verzögerungszeit, die durch die Anstiegszeit des Eingangssignals entsteht. Hierbei führt ein langsamerer Übergang am Eingang der Zelle zu einer höheren Verzögerungszeit. Die Verzögerungszeit berechnet sich als Produkt aus der eingehenden Anstiegs- bzw. Abfallzeit und dem Flankensensitivitätsfaktor S (Gleichung 15). Durch den Faktor S wird die Zeit mit einkalkuliert, während der die Eingangsspannung zu steigen beginnt, aber noch nicht den Schwellenwert erreicht hat, bei dem die Kanalleitung beginnt.

$$D_S = D_{T_{previous}} \cdot S \tag{15}$$

Connect Delay D_C Das Connect Delay gibt die Zeit an, die ein Eingangspin benötigt, um seinen Pegel zu ändern, wenn der ihn treibende Ausgangspin seinen Pegel ändert. Es wird auch als »time-of-flight delay« bezeichnet. Die Timing-Analyse unterstützt drei Fälle in Bezug auf die Topologie der Schaltung.

1. Best Case (best_case_tree)
   Best Case bezeichnet eine Topologie, bei der Last und treibender Pin physisch aneinander liegen. Daraus folgt, dass der Leitungswiderstand $R_{wire} = 0$ ist, sodass der Term

   $$D_{C_{best}} = R_{wire} \cdot \left(C_{wire} + C_{pin}\right) \tag{16}$$

   0 ergibt.

2. Balanced (balanced_tree)
   Die Balanced-Topologie gibt an, dass alle Lastelemente auf separaten, gleichartigen Zweigen des Netzes liegen. Leitungskapazität und Widerstand verteilen sich zu gleichen Teilen auf N Lastpins.

   $$D_{C_{balanced}} = \frac{R_{wire}}{N} \left(\frac{C_{wire}}{N} + C_{pin}\right) \tag{17}$$

3. Worst Case (worst_case_tree)
   Worst-case beschreibt den Fall, bei dem sich die Last am äußersten Ende der Leitung befindet. Daher fällt auf jeden Pin die gesamte Leitungskapazität, sowie der Leitungswiderstand an.

   $$D_{C_{worst}} = R_{wire} \cdot \left(C_{wire} + \sum_{pins} C_{pin}\right) \tag{18}$$

Transition Delay D_T Das Transition Delay ist die Verzögerungszeit, welche durch die kapazitive Last am Ausgang eines Gatters definiert wird.

$$D_T = R_{drive} \cdot \left(\sum_{pins} C_{pin} + C_{wire} \right)$$

9.1.1 *Berechnung des linearen Delay Model*

Abbildung 22 zeigt einen Inverter, welcher drei NAND-Gatter treibt und eine fallende Flanke am Eingang registriert. Zur Berechnung der Verzögerung des Inverter-

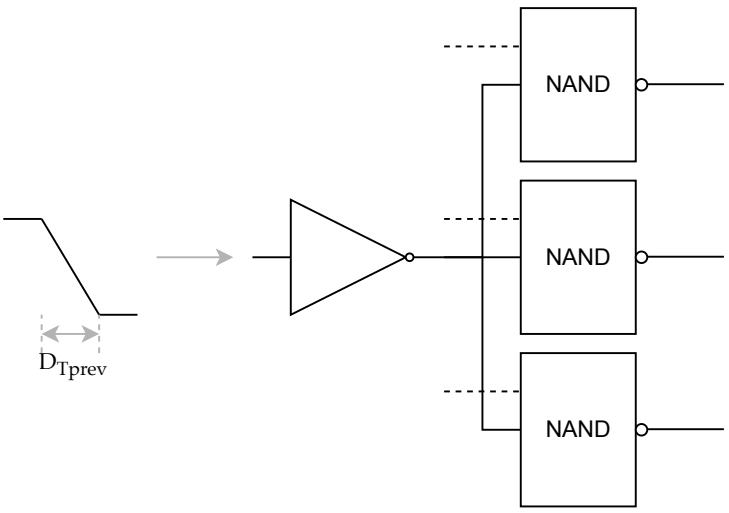

Abbildung 22: Linear Delay anhand eines Inverters

signals werden folgende Werte benötigt. Die Abfallzeit der Flanke am Eingang (D_T), die intrinsische Verzögerungszeit des Inverters (D_I), der Flankensensitivitätsfaktor S_{rise}, der Ausgangswiderstand des Inverters R_{rise}, die Leitungskapazität (C_{wire}) sowie die Eingangskapazität der NAND-Gatter (C_{pin}). Anhand des folgenden Beispiels soll die Berechnung des LDM durchgeführt werden. Hierbei werden folgende Werte

benutzt:

Abfallzeit der eingehenden Flanke	$D_{T_{prev}} = 1,2ns$
Intrinsische Verzögerungszeit des Inverters	$D_I = 1,4$
Flankensensitivitätsfaktor	$S_{rise} = 0,02$
Ausgangswiderstand des Inverters	$R_{rise} = 0,14$
Leitungskapazität	$C_{wire} = 0,14$
Eingangskapazität NAND-Gatter	$C_{pin} = 1,1$

Nach Gleichung 14 berechnet sich die Verzögerungszeit zu:

$D_{ges} = D_I + D_S + D_C + D_T$ mit
$D_I = 1,4\text{ns}$
$D_S = D_{T_{prev}} \cdot S_{rise} = 1,2\text{ns} \cdot 0,02 = 0,024$
$D_C = 0$ (best-case model)
$D_T = R_{rise} \cdot \left(\sum_{pins} \cdot C_{pin} + C_{wire} \right) = 0,14\text{k}\Omega \cdot (3 \cdot 1,1\text{pF} + 2,6\text{pF}) = 0,826\text{ns}$
$\Rightarrow D_{ges} = 1,4\text{ns} + 0,024\text{ns} + 0\text{ns} + 0,826\text{ns} = 2,25\text{ns}$

Die gesamte Verzögerungszeit beträgt 2,25ns.

9.1.2 *Einbindung in Liberty-Bibliothek*

Intrinsisches Delay D_I Die intrinisischen Anstiegs- und Abfallzeiten der Zellen lassen sich über `intrinsic_rise` und `intrinsic_fall` konfigurieren.

Ausgangswiderstand R_{drive} Der Ausgangswiderstand lässt sich über rise_resistance und fall_resistance konfigurieren.

Lastkapazität C_{pin} Die Kapazität des Pins wird über `capacitance` konfiguriert.

Leitungskapazität C_{wire} Die Leitungskapazität wird in der wire_load-Gruppe definiert.

Leitungswiderstand R_{wire} Der Leitungswiderstand wird in der wire_load-Gruppe definiert.

Flankensensitivitätsfaktor S Das Attribut wird in der timing-Gruppe über slope_rise und slope_fall definiert.

9.2 Nonlinear Delay Model

Das Nonlinear Delay Model betrachtet die Verzögerungszeiten anhand von Tabellen. Bei der Verzögerungsanalyse wird die Gesamtverzögerung berechnet, welche die Zellverzögerung (Cell Delay D_{cell}) und Verbindungsverzögerung (Connect Delay D_C) umfasst (Gleichung 19).

$$D_{total} = D_{cell} + D_C \tag{19}$$

9.2.1 *Parameter*

Cell Delay D_{cell}: Die Verzögerung, die vom Gatter selbst eingebracht wird, wird typischerweise von 50% der Eingangsspannung bis 50% der Ausgangsspannung gemessen. Das Cell Delay D_{cell} kann über zwei Methoden berechnet werden. Zum Einen kann das Delay über cell_rise bzw. cell_fall-Tabellen durch Interpolation der darin befindlichen Werte berechnet werden. Zum Anderen lässt es sich über propagation und transition-Tabellen nach Gleichung 20 berechnen.

$$D_{cell} = D_{propagation} + D_{transition} \tag{20}$$

Beide Methoden zur Berechnung lassen sich in einer Liberty-Bibliothek kombinieren.

Connect Delay D_C: Die Verzögerungszeit wird entweder mit Hilfe des tree_type-Attributs in der environment-Gruppe über das wire-load-Modell eingestellt oder aus einer Standard Delay Format (SDF)-Datei eingelesen. Das Connect Delay wird nach der gleichen Methode berechnet, mit der es auch im Linear Delay Model berechnet wird.

Propagation Delay $D_{propagation}$: Das Propagation Delay ist die Zeitdauer vom Erreichen eines stabilen Eingangssignals bis zu einer Änderung am Ausgang des Logikgatters.

Transition Delay $D_{transition}$: Das Transition Delay ist die Zeit zwischen zwei Referenzspannungspegeln am Ausgangspin. Diese Pegel können zum Beispiel 20 bis 80 Prozent oder 10 bis 50 Prozent betragen. Das Delay wird innerhalb der Liberty-Bibliothek durch zugewiesene Werte in LUTs oder Interpolation derer berechnet.

9.2.2 *Einbindung in die Liberty-Bibliothek*

Zur Nutzung des NLDM muss in der library-Gruppe das delay_model table_lookup eingestellt werden. Des Weiteren müssen in der library-Gruppe NLDM Templates definiert werden. Templates sind Tabellen, welche Informationen speichern, die von mehreren lookup tables genutzt werden können. Der Inhalt der Templates dient der Definition der variable-Werte, welche dann die Achsenbeschriftung für das Diagramm zur Interpolation bilden (vgl. Abbildung 23).

```
lu_table_template(name) {
variable_1 : value;
variable_2 : value;
variable_3 : value;
index_1 ("float, ..., float");
index_2 ("float, ..., float");
index_3 ("float, ..., float");
}
```

Listing 15: Template Syntax

Listing 15 zeigt die Syntax für die Eingabe des Templates.

9.2.3 *Variable / Index*

Die möglichen Werte für variable_1 bis variable_3 sind in sets unterteilt. Die

Set	Variable
1	input_net_transition
2	total_output_net_capacitance
	output_net_length
	output_net_wire_cap
	output_net_pin_cap
3	related_out_total_output_net_capacitance
	related_out_output_net_length
	related_out_output_net_wire_cap
	related_out_output_net_pin_cap

Tabelle 7: Einteilung der Variable-Werte in Sets

Variablen, welche im Template genutzt werden können, hängen von der Dimension des Templates ab. Die möglichen Kombinationen sind in Tabelle 8 aufgeführt.

- input_net_transition
- total_output_net_capacitance
- output_net_length
- related_out_total_output_net_capacitance

Template Dimension	Variable		
	1	2	3
1	set1		
	set2		
2	set1	set2	
	set2	set1	
3	set1	set2	set3
	set1	set3	set2
	set2	set3	set1
	set2	set1	set3
	set3	set1	set2
	set3	set2	set1

Tabelle 8: Variablen für Template Dimension

- output_net_pin_cap
- output_net_wire_cap

Innerhalb der Bibliothek sind folgende Angaben in der Timing-Gruppe möglich:

- Rise propagation
- Fall propagation
- Cell rise
- Cell fall
- Rise transition
- Fall transition

Die Timing-Gruppe eines Pins kann *entweder* Cell-Angaben (cell_rise / cell_fall) *oder* Propagation-Angaben (rise_propagation / fall_propagation) beinhalten. Beide Werte innerhalb der timing-Gruppe eines Pins sind nicht zulässig. Die Transition-Angaben muss jede timing-Gruppe beinhalten.

9.2.4 *Beispiel*

```
lu_table_template(basic_template_4x3) {
  variable_1 : total_output_net_capacitance;
  variable_2 : input_net_transition;
  index_1 ("0, 0.025, 0.05, 0.075");
  index_2 ("0.1, 0.5, 1");
}
...
cell("and02") {
 timing() {
    related_pin : "A0";
    fall_transition(basic_template_4x3) {
       index_1 ("0, 0.025, 0.05, 0.075"); //Output Capacitance
```

```
        index_2 ("0.1, 0.5, 1"); //Input Transition time
        values ("0.039903, 0.052789, 0.068544", \
                "0.18664,  0.19353,  0.20444", \
                "0.33635,  0.34074,  0.34821", \
                "0.48791,  0.49095,  0.49658");
    }}
```

Listing 16: Lookup Table

Listing 16 zeigt den Quellcode einer Tabelle, welche der Zuordnung von Abfallzeiten dient. Der Name des Templates (4x3) gibt die Größe der Matrix an. Die Werte, welche unter values angegeben sind, entsprechen dem Produkt der Matrix (4x3 = 12), d.h. 12 Werte unter values. In Abbildung 23 sind die Werte aus Listing 16 graphisch dargestellt. Nicht vorhandene Werte werden interpoliert.

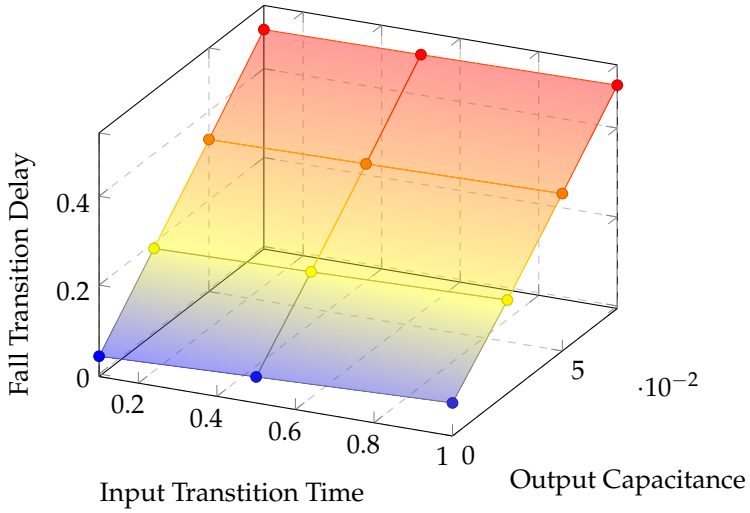

Abbildung 23: Fall Transition Delay eines and02-Gatter im NLDM

Anhand der Abbildung ist zu erkennen, dass die Verzögerungszeit bei zunehmender Ausgangskapazität und Eingangsübergangszeit zunimmt.

10 Modellierung der Leistungsaufnahme

Die Verlustleistung von Zellen lässt sich in zwei Kategorien unterteilen:

1. Statische Verlustleistung
2. Dynamische Verlustleistung

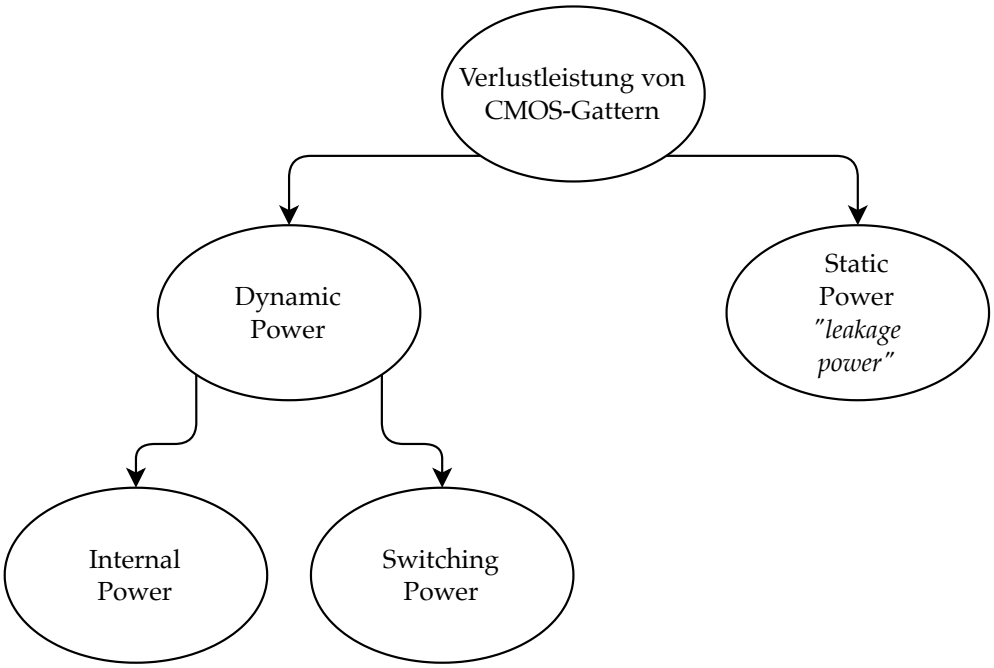

Abbildung 24: Verlustleistung von CMOS-Gattern

10.1 Static Power

Unter der statischen Verlustleistung versteht man die Leistung, die von einem Gatter abgeführt wird, wenn es inaktiv ist. Der Hauptanteil der statischen Leistung resultiert aus einem Leckstrom zwischen Source und Drain des CMOS-Transistors. Darüber hinaus entsteht eine statische Verlustleistung, wenn Strom zwischen den Diffusionsschichten des Source- und Drain-Anschlusses und dem Substrat leckt. Aus diesem Grund wird die statische Leistung auch als »leakage power« bezeichnet.

10.1.1 *Modellierung innerhalb der Liberty-Bibliothek*

Die statische Verlustleistung kann abhängig und unabhängig vom Status des Gatters angegeben werden. Für die unabhängige Beschreibung steht das Attribut

cell_leakage_power zur Verfügung. Für die statusabhängige Beschreibung wird eine Gruppe genutzt, welche leakage_power_group genannt wird. Falls keine Angabe innerhalb der Zellbeschreibung gemacht wird, wird bei der Synthese der unter default_cell_leakage_power genannte Wert benutzt.

10.2 Dynamische Verlustleistung

Die dynamische Verlustleistung wird umgesetzt, wenn eine Schaltung aktiv ist und sich das Ausgangssignal der Schaltung aufgrund eines an die Schaltung angelegten Stimulus ändert. Die dynamische Verlustleistung setzt sich aus folgenden Termen zusammen:

- Internal Power
- Switching Power

10.2.1 *Internal Power*

Die Internal Power wird während Schaltungsvorgängen durch die Umladung interner Kapazitäten der Zelle umgesetzt. Darüber hinaus beinhaltet die Definition der Internal Power noch die »short-circuit power« , welche sich auf Grund kurzzeitiger Kurzschlüsse zwischen der Versorgungsspannung und Masse ergeben, wenn alle Transistoren d.h. sowohl PMOS als auch NMOS kurzzeitig geschlossen sind. Die Short-circuit Power macht den Großteil der Internal Power aus. Im Vergleich zur Switching-Power ist sie aber meist wesentlich geringer. Abbildung 25 verdeutlicht die short-circuit Power anhand eines CMOS-Inverters.

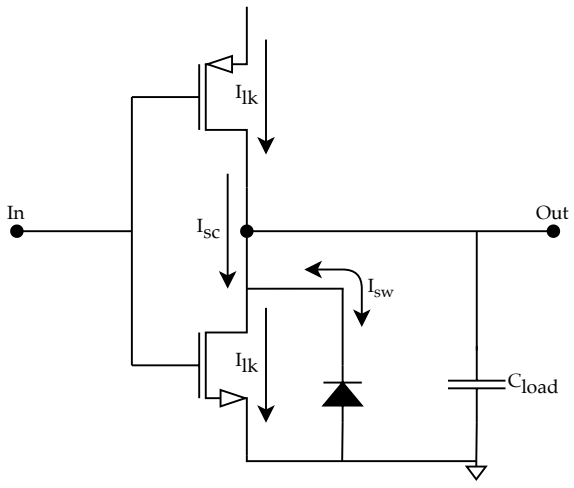

Abbildung 25: Dynamische Verlustleistung an CMOS-Inverter

Bei einer langsam ansteigenden Eingangsspannung, gibt es einen Zeitpunkt zu dem NMOS, sowie PMOS-Transistor zeitweise leiten. In diesem Moment fließt der Kurzschlussstrom I_{sc} von VDD zu GND. Die Höhe der short-circuit Power variiert von Schaltung zu Schaltung. Einfluss auf den Strom hat die Anstiegszeit der Eingangsspannung: bei hohen Anstiegszeiten ist I_{sc} kleiner, da sich die Zeit verringert in der beide Transistoren leiten. Demnach ist I_{sc} für geringe Anstiegszeiten der Eingangsspannung höher. Ein weiterer Einfluss ist die kapazitive Last am Ausgang des Gatters. Je größer die kapazitive Last desto länger dauert es, bis sich eine Spannung zwischen Drain und Source des jeweiligen Transistors aufbauen kann, die zu einem signifikanten Stromfluss führen kann. Wenn während dieser Zeit der Schaltvorgang am Eingang bereits abgeschlossen wurde, kann sich kein Kurzschlussstrom einstellen.

10.2.2 *Power-Modellierung im Liberty-Format*

Für die Modellierung der Internal Power stehen in Liberty Bibliotheken zwei Optionen zur Auswahl:

1. Nutzung des internal_power-Attributs. Hierbei wird die Ausgangskapazität der Pins auf 0 gesetzt.
2. Den Ausgangspins eine Kapazität zuweisen, dadurch werden sie in die Switching Power miteinbezogen, sodass die Internal Power nur den Wert der Short-circuit Power widerspiegelt.

Power Lookup Tables Innerhalb der Library-Gruppe werden ein-, zwei- und dreidimensionale Lookup Tables unterstützt. Abbildung 26 zeigt ein Beispiel für einen dreidimensionalen Graphen, welcher von den Variablen Input Transition Time, Output capacitance und Energy/Transition abhängt.

Die Beschreibung der Internal Power muss über das power_model-Attribut definiert werden. Der einzig gültiger Wert für das Attribut ist table_lookup.

```
power_model : table_lookup
```

Listing 17: Power Model Attribut

Die Variablen zur Definition des Power LUT werden über power_lut_template definiert.

```
power_lut_template(name) {
variable_1 : string ;
variable_2 : string ;
variable_3 : string ;
index_1("float, ... , float") ;
index_2("float, ... , float") ;
index_3("float, ... , float") ;}
```

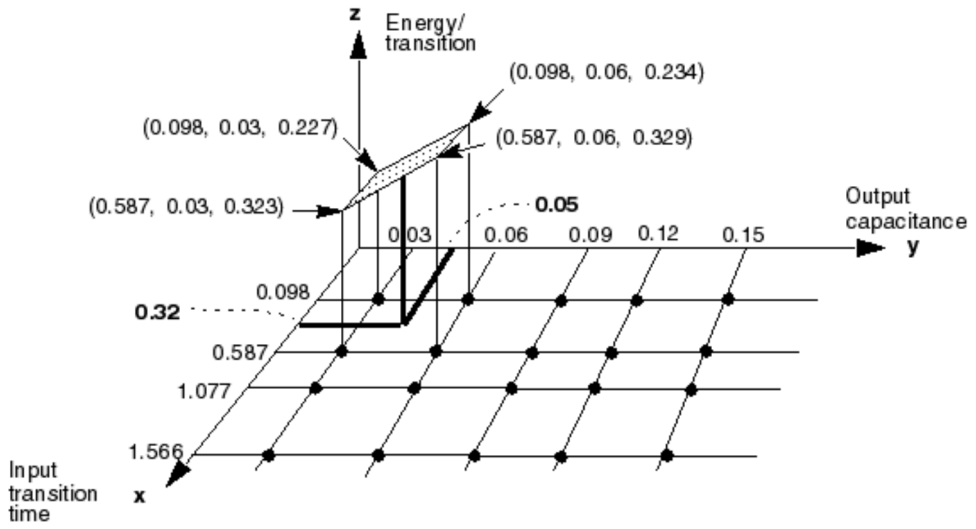

Abbildung 26: Internal Power Lookup Table[20]

Listing 18: Power Lookup Table

Der Name des power_lut_template wird daraufhin als Name für die fall_power-, power- oder rise_power-Gruppe genutzt. Variable 1-3 kann folgende Werte annehmen:

total_output_net_capacitance Wert für die Ausganskapazität, welche in der internal_power-Gruppe innerhalb der pin-Gruppe definiert wird.

equal_or_opposite_net_capacitance Wert für die Ausganskapazität, welche unter equal_or_opposite_output definiert wird.

input_transition_time Die Übergangszeit des Pins, welcher über related_pin in der internal_power-Grupper definiert wird.

10.2.3 *Switching Power*

Die Switching Power einer Zelle ist die Leistung, welche durch das Auf- und Entladen der Lastkapazitäten bei Schaltaktivität abgeführt wird. Diese Lastkapazitäten sind größtenteils Netz und Gatterkapazitäten. Da das Auf- und Entladen der Lastkapazitäten nur durch logische Übergänge am Ausgang der Zelle entstehen kann, nimmt die Switching Power mit der Schaltaktivität der Zelle zu. Dieser Zusammenhang ist in Gleichung 21 dargestellt.

$$P_{sw} = \alpha \cdot C_{load} \cdot V^2 \cdot f \tag{21}$$

Hierbei ist α ein Faktor, der die Schaltungsaktivität der Zelle beschreibt. Die Schaltungsaktivität ist ein Maß für die Anzahl der logischen Zustandsänderungen im Netz der zu betrachtenden Zelle. Aus der Gleichung geht hervor, dass es möglich ist, die Verlustleistung durch eine Absenkung der Taktfrequenz f oder der Lastkapazität C_{load} und der Versorgungsspannung V zu verringern.

10.2.4 *Modellierung innerhalb der Liberty-Bibliothek*

Die Beschreibung von Internal und Switching Power erfolgt über eine lookup-Table. Hierfür steht das Attribut power_model : table_lookup zur Verfügung. Die Nutzung der Lookup Table wird in Kapitel 10.2.2 erklärt.

11 Bibliothek

11.1 Vorgaben

Zur Definition benötigter Zellen und Timings wurden folgende Vorgaben festgelegt.

11.1.1 *Zellen*

CC_IBUF Input-Buffer-Zelle

CC_OBUF Output-Buffer-Zelle

CC_BUFG Clock-Buffer-Zelle

CC_INV Inverter

CC_AND4 4-Input UND-Gatter

CC_AND08 8-Input UND-Gatter

CC_OR4 4-Input ODER-Gatter

CC_OR8 8-Input ODER-Gatter

CC_XOR4 4-Input EXLUSIV-ODER-Gatter

CC_XOR8 8-Input EXLUSIV-ODER-Gatter

CC_NAND2 2-Input NICHT-UND-Gatter

CC_NOR2 2-Input NICHT-ODER-Gatter

CC_DFF D-Flipflop

CC_DLT Latch

CC_ADDF Volladierer

CC_MX2 2-Input Multiplexer

CC_MX4 4-Input Multiplexer

CC_PLL Phasenregelschleife

CC_MULT Multiplizierer

CC_SERDES Serializer/De-serializer

CC_DPSRAM_20K RAM-Block

CC_DPSRAM_40K RAM-Block

11.1.2 *Timing*

Buffer und Inverter sollen ein Timing von 0 haben. Der Fulladder soll im Carry-Pfad ein Timing von 0.1 haben und alle restlichen Elemente ein Timing von 1. Die *Slope Rise* und *Slope Fall*-Werte werden auf 0 gesetzt.

11.1.3 *Area*

Die Area der Gatter ist in Tabelle 9 aufgeführt.

Gatter	Area
8-Bit Gatter	1
4-Bit Gatter	0,5
2-Bit Gatter	0,5
Latch und FF	0,5
Inverter und Buffer	0

Tabelle 9: Area der Gatter

11.2 Header

Listing 19 zeigt den Header der Bibliothek.

```
library(imes_cc) {

   delay_model : generic_cmos;
   in_place_swap_mode : match_footprint;

 /* unit attributes */
   time_unit : "1ns";
   voltage_unit : "1V";
   current_unit : "1uA";
   pulling_resistance_unit : "1kohm";
   leakage_power_unit : "1nW";
   capacitive_load_unit (10, ff);
   default_input_pin_cap : 1.0 ;
   default_output_pin_cap : 0.0 ;
   slew_upper_threshold_pct_rise : 80;
   slew_lower_threshold_pct_rise : 20;
   slew_upper_threshold_pct_fall : 80;
   slew_lower_threshold_pct_fall : 20;
   input_threshold_pct_rise : 50;
   input_threshold_pct_fall : 50;
   output_threshold_pct_rise : 50;
   output_threshold_pct_fall : 50;
   nom_process : 1;
   nom_voltage : 5;
```

```
    nom_temperature : 25;
    operating_conditions ( OC1 ) {
    process : 1;
    voltage : 5;
    temperature : 25;
  }
  wire_load(WL1){
    resistance : 0 ;
    capacitance : 1.0 ;
    area : 0 ;
    slope : 0 ;
    fanout_length(1, 1) ;
    fanout_length(2, 2) ;
    fanout_length(3, 3) ;
    fanout_length(4, 4) ;
    fanout_length(5, 5) ;
    fanout_length(6, 6) ;
    fanout_length(7, 7) ;
    fanout_length(8, 8) ;
    fanout_length(9, 9) ;
    fanout_length(10, 10) ;
  }
    default_wire_load : WL1;
  default_operating_conditions : OC1;
}
```

Listing 19: Library Header

Über den Header wird das Delay-Model auf generic_cmos gestellt. Dies entspricht dem Linear Delay Model. Das voltage_map-Attribut wird genutzt, um die Spannungspegel für VDD und VSS zu definieren. Im Falle der Bibliothek entsprechen diese 1.2V und 0V. Die innerhalb der Bibliothek genutzte Einheit wird in voltage_unit definiert und entspricht hier 1V. Analog dazu müssen die leakage_power_unit und capacitive_load_unit definiert werden. Die Bibliothek verwendet 1pW für die leakage_power und 10fF für die Kapazität. Die default-Werte gelten für alle Zellen der Bibliothek und werden immer dann angewendet, wenn keine eigenen Werte festgelegt worden sind.

11.3 Zellen

Die folgende Auflistung der Zellen soll einen Überblick über ihre Funktion und ihr Timing geben.

11.3.1 *CC_IBUF*

Area	0
Inputs	I
Outputs	O
Function	I

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O $\rightarrow$ I	0	0	0	0	0	0

Tabelle 10: IBUF Timing

11.3.2 *CC_OBUF*

Area	0
Inputs	I
Outputs	O
Function	I

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O $\rightarrow$ I	0	0	0	0	0	0

Tabelle 11: OBUF Timing

11.3.3 *CC_BUFG*

Area	0
Inputs	I
Outputs	O
Function	I

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O → I	0	0	0	0	0	0

Tabelle 12: BUFG Timing

11.3.4 *CC_INV*

Area	0
Inputs	I
Outputs	O
Function	!I

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O → I	0	0	0	0	0	0

Tabelle 13: CC_INV Timing

11.3.5 *CC_AND4*

Area	1
Inputs	I0, I1, I2, I3
Outputs	O
Function	I0 * I1 * I2 * I3

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O → I0	0.00135	0.00142	0.00421	0.00428	0	0
O → I1	0.00135	0.00142	0.00421	0.00428	0	0
O → I2	0.00135	0.00142	0.00421	0.00428	0	0
O → I3	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 14: CC_AND4 Timing

11.3.6 *CC_AND8*

Area	1
Inputs	I0, I1, I2, I3, I4, I5, I6, I7
Outputs	O
Function	I0 * I1 * I2 * I3 * I4 * I5 * I6 * I7

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O → I0	0.00135	0.00142	0.00421	0.00428	0	0
O → I1	0.00135	0.00142	0.00421	0.00428	0	0
O → I2	0.00135	0.00142	0.00421	0.00428	0	0
O → I3	0.00135	0.00142	0.00421	0.00428	0	0
O → I4	0.00135	0.00142	0.00421	0.00428	0	0
O → I5	0.00135	0.00142	0.00421	0.00428	0	0
O → I6	0.00135	0.00142	0.00421	0.00428	0	0
O → I7	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 15: CC_AND8 Timing

11.3.7 *CC_OR4*

Area	1
Inputs	I0, I1, I2, I3
Outputs	O
Function	I0 ∧ I1 ∧ I2 ∧ I3

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O → I0	0.00135	0.00142	0.00421	0.00428	0	0
O → I1	0.00135	0.00142	0.00421	0.00428	0	0
O → I2	0.00135	0.00142	0.00421	0.00428	0	0
O → I3	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 16: CC_OR4 Timing

11.3.8 *CC_OR8*

Area	1
Inputs	I0, I1, I2, I3, I4, I5, I6, I7
Outputs	O
Function	I0 ∧ I1 ∧ I2 ∧ I3 ∧ I4 ∧ I5 ∧ I6 ∧ I7

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O → I0	0.00135	0.00142	0.00421	0.00428	0	0
O → I1	0.00135	0.00142	0.00421	0.00428	0	0
O → I2	0.00135	0.00142	0.00421	0.00428	0	0
O → I3	0.00135	0.00142	0.00421	0.00428	0	0
O → I4	0.00135	0.00142	0.00421	0.00428	0	0
O → I5	0.00135	0.00142	0.00421	0.00428	0	0
O → I6	0.00135	0.00142	0.00421	0.00428	0	0
O → I7	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 17: CC_OR8 Timing

11.3.9 *CC_XOR4*

Area	1
Inputs	I0, I1, I2, I3
Outputs	O
Function	I0 ∧ I1 ∧ I2 ∧ I3

Tabelle 18: XOR08

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O → I0	0.00135	0.00142	0.00421	0.00428	0	0
O → I1	0.00135	0.00142	0.00421	0.00428	0	0
O → I2	0.00135	0.00142	0.00421	0.00428	0	0
O → I3	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 19: CC_XOR4 Timing

11.3.10 *CC_XOR8*

Area	1
Inputs	I0, I1, I2, I3, I4, I5, I6, I7
Outputs	O
Function	I0 ∧ I1 ∧ I2 ∧ I3 ∧ I4 ∧ I5 ∧ I6 ∧ I7

Tabelle 20: XOR08

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slop_rise	slope_fall
O → I0	0.00135	0.00142	0.00421	0.00428	0	0
O → I1	0.00135	0.00142	0.00421	0.00428	0	0
O → I2	0.00135	0.00142	0.00421	0.00428	0	0
O → I3	0.00135	0.00142	0.00421	0.00428	0	0
O → I4	0.00135	0.00142	0.00421	0.00428	0	0
O → I5	0.00135	0.00142	0.00421	0.00428	0	0
O → I6	0.00135	0.00142	0.00421	0.00428	0	0
O → I7	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 21: CC_XOR8 Timing

11.3.11 *CC_NAND2*

Area	0.5
Inputs	I0, I1
Outputs	O
Function	!(I0*I1)

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slope_rise	slope_fall
O → I0	0.00135	0.00142	0.00421	0.00428	0	0
O → I1	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 22: CC_NAND2 Timing

11.3.12 *CC_NOR2*

Area	0.5
Inputs	I0, I1
Outputs	O
Function	(!(I0+I1))

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slope_rise	slope_fall
O → I0	0.00135	0.00142	0.00421	0.00428	0	0
O → I1	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 23: CC_NOR2 Timing

11.3.13 *CC_DFF*

Area	0.5
Inputs	CLK, SR, D
Outputs	Q
next_state function	D

	timing_type	timing_sense	int_rise	int_fall	rise_res	fall_res	slope_rise	slope_fall
D → CLK	setup_rising		0.00135	0.00142	0.00421	0.00428	0	0
D → CLK	hold_rising		0.00135	0.00142	0.00421	0.00428	0	0
SR → CLK	recovery_rising		0.00135	0.00142	0.00421	0.00428	0	0
SR → CLK	removal_rising		0.00135	0.00142	0.00421	0.00428	0	0
Q → CLK	rising_edge		0.00135	0.00142	0.00421	0.00428	0	0
Q → SR	clear	positive_unate	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 24: CC_DFF Timing

11.3.14 *CC_DLT*

Area	0.5
Inputs	G, D, R, S
Outputs	Q, QN
Function	

	timing_type	timing_sense	int_rise	int_fall	rise_res	fall_res	slope_rise	slope_fall
Q → G	rising_edge	non_unate	0.00135	0.00142	0.00421	0.00428	0	0
Q → D	combinational	positive_unate	0.00135	0.00142	0.00421	0.00428	0	0
Q → R	clear	positive_unate	0.00135	0.00142	0.00421	0.00428	0	0
Q → S	clear	negative_unate	0.00135	0.00142	0.00421	0.00428	0	0
QN → G	rising_edge	non_unate	0.00135	0.00142	0.00421	0.00428	0	0
QN → D	combinational	negative_unate	0.00135	0.00142	0.00421	0.00428	0	0
QN → R	preset	negative_unate	0.00135	0.00142	0.00421	0.00428	0	0
QN → S	clear	positive_unate	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 25: CC_DLT Timing

11.3.15 *CC_ADDF*

Area	0.5
Inputs	A, B, CI
Outputs	CO
Function	$(((AB) + (BCI)) + (CIA))$

	timing_type	timing_sense	int_rise	int_fall	rise_res	fall_res	slope_rise	slope_fall
CO → A		positive_unate	0.00135	0.00142	0.00421	0.00428	0	0
CO → B		positive_unate	0.00135	0.00142	0.00421	0.00428	0	0
CO → CI		positive_unate	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 26: CC_DLT Timing

11.3.16 *CC_MX2*

Area	2
Inputs	D0, D1, S0
Outputs	Y
Function	(!((S0 D0) + (!S0 D1)))

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slope_rise	slope_fall
Y → D0	0.00135	0.00142	0.00421	0.00428	0	0
Y → D1	0.00135	0.00142	0.00421	0.00428	0	0
Y → S0	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 27: CC_MX2 Timing

11.3.17 *CC_MX4*

11.3.18 *CC_ADDF*

Area	0.5
Inputs	A, B, CI
Outputs	CO, S
Function	(((A B)+(B CI))+(CI A))

11.3.19 *CC_PLL*

Area	1
Inputs	REFCLK, FBKCLK
Outputs	OUTCLK_1x, OUTCLK_2x

Area	2
Inputs	D0, D1, D2, D3, S0, S1
Outputs	Y
Function	(!S0*!S1*D0)+(S0*!S1*D1)+(!S0*S1*D2)+(S0*S1*D3)

	intrinsic_rise	instrinsic_fall	rise_resistance	fall_resistance	slope_rise	slope_fall
Y → D0	0.00135	0.00142	0.00421	0.00428	0	0
Y → D1	0.00135	0.00142	0.00421	0.00428	0	0
Y → D2	0.00135	0.00142	0.00421	0.00428	0	0
Y → D3	0.00135	0.00142	0.00421	0.00428	0	0
Y → S0	0.00135	0.00142	0.00421	0.00428	0	0
Y → S1	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 28: CC_MX4 Timing

	timing_type	timing_sense	int_rise	int_fall	rise_res	fall_res	slope_rise	slope_fall
CO → A	combinational	positive_unate	0.00135	0.00142	0.00421	0.00428	0	0
CO → B	combinational	positive_unate	0.00135	0.00142	0.00421	0.00428	0	0
CO → CI	combinational	positive_unate	0.00135	0.00142	0.00421	0.00428	0	0
S → A	combinational	non_unate	0.00135	0.00142	0.00421	0.00428	0	0
S → B	combinational	non_unate	0.00135	0.00142	0.00421	0.00428	0	0
S → CI	combinational	non_unate	0.00135	0.00142	0.00421	0.00428	0	0

Tabelle 29: CC_ADDF Timing

	timing_type	timing_sense	int_rise	int_fall	rise_res	fall_res	slope_rise	slope_f
OUTCLK_1x → REFCLK	combinational_rise	positive_unate	0.00135	0.00142	0.00421	0.00428	0	
OUTCLK_1x → REFCLK	combinational_fall	positive_unate	0.00135	0.00142	0.00421	0.00428	0	
OUTCLK_2x → REFCLK	combinational_rise	positive_unate	0.00135	0.00142	0.00421	0.00428	0	
OUTCLK_2x → REFCLK	combinational_fall	positive_unate	0.00135	0.00142	0.00421	0.00428	0	

Tabelle 30: CC_PLL Timing

11.3.20 *CC_MULT*

Area	0.001
Inputs	A0, A1, B0, B1
Outputs	X0, X1, X2, X3
Function	X0: A0 * B0
	X1: (A0 * B1) ∧ (A1 * B0)
	X2: A1 * B1 ∧ ((A0 * B1) * (A1 * B0))
	X3: A1 * B1 * (A0 * B1) * (A1 * B0)

	timing_type	timing_sense	int_rise	int_fall	rise_res	fall_res	slope_rise	slope_fall
X0 → A0			0.0135	0.0142	0.0421	0.0428	0	0
X0 → B0			0.0135	0.0142	0.0421	0.0428	0	0
X1 → A0			0.0135	0.0142	0.0421	0.0428	0	0
X1 → A1			0.0135	0.0142	0.0421	0.0428	0	0
X1 → B0			0.0135	0.0142	0.0421	0.0428	0	0
X1 → B1			0.0135	0.0142	0.0421	0.0428	0	0
X2 → A0			0.0135	0.0142	0.0421	0.0428	0	0
X2 → A1			0.0135	0.0142	0.0421	0.0428	0	0
X2 → B0			0.0135	0.0142	0.0421	0.0428	0	0
X2 → B1			0.0135	0.0142	0.0421	0.0428	0	0
X3 → A0			0.0135	0.0142	0.0421	0.0428	0	0
X3 → A1			0.0135	0.0142	0.0421	0.0428	0	0
X3 → B0			0.0135	0.0142	0.0421	0.0428	0	0
X3 → B1			0.0135	0.0142	0.0421	0.0428	0	0

Tabelle 31: CC_MULT Timing

11.3.21 *CC_SERDES*

Der SerDes (Serializer / Deserializer) ist als Makro-Zelle in der Bibliothek implementiert. Das heisst, dass er kein function-Attribut besitzt.

Area	0
Inputs	A
Outputs	Y

11.3.22 *CC_DPSRAM_20K*

Makro-Zelle, d.h. kein function-Attribut.

Area	0
Inputs	DIA[19:0], DIB[19:0], WEA[19:0], WEB[19:0], ADDRA[15:0], ADDRB[15:0], CLKA, CLKB, ENA, ENB, GLWEA, GLWEB
Outputs	DOA[19:0], DOB[19:0], ECCA_1B, ECCB_1B, ECCA_2B, ECCB_2B

11.3.23 *CC_DPSRAM_40K*

Makro-Zelle, d.h. kein function-Attribut.

Area	0
Inputs	DIA[39:0], DIB[39:0], WEA[39:0], WEB[39:0], ADDRA[15:0], ADDRB[15:0], CLKA, CLKB, ENA, ENB, GLWEA, GLWEB
Outputs	DOA[39:0], DOB[39:0], ECCA_1B, ECCB_1B, ECCA_2B, ECCB_2B

11.4 Leistungsaufnahme

Die Internal Power ist für jede Zelle der Bibliothek identisch beschrieben worden. Diese Beschreibung wurde über eine Lookup-Table getätigt (vgl. Listing 20). Die zugewiesenen Parameter index 1 und 2 werden in der library-Gruppe definiert. Das Template, in dem diese Werte definiert sind, ist in Listing 21 zu sehen. Index 1 wird *input_transition_time* zugewiesen und Index 2 wird *total_output_net_capacitance* zugewiesen.

```
internal_power() {
  related_pin : "I0 I1";
  rise_power(energy_template_5x5) {
    index_1 ("0.025, 0.05, 0.1, 0.3, 0.6");
    index_2 ("0.06, 0.18, 0.42, 0.6, 1.2");
    values ( \
      "0.175608, 0.272801, 0.56388, 0.796823, 1.60912", \
      "0.175608, 0.272801, 0.56388, 0.796823, 1.60912", \
      "0.175608, 0.272801, 0.56388, 0.796823, 1.60912", \
      "0.175608, 0.272801, 0.56388, 0.796823, 1.60912", \
      "0.175608, 0.272801, 0.56388, 0.796823, 1.60912");
  }
  fall_power(energy_template_5x5) {
    index_1 ("0.025, 0.05, 0.1, 0.3, 0.6");
    index_2 ("0.06, 0.18, 0.42, 0.6, 1.2");
    values ( \
      "0.682097, 0.778941, 1.07036, 1.30235, 2.11129", \
      "0.682097, 0.778941, 1.07036, 1.30235, 2.11129", \
      "0.682097, 0.778941, 1.07036, 1.30235, 2.11129", \
      "0.682097, 0.778941, 1.07036, 1.30235, 2.11129", \
      "0.682097, 0.778941, 1.07036, 1.30235, 2.11129");
  }
}
```

Listing 20: Internal Power Lookup-Table

Die oberste Zeile der Werte, welche unter values eingetragen sind, ergibt sich aus der Zuweisung des ersten Wertes von Index 1 mit den Werten von Index 2. Die Spalten der Tabelle wurden auf einen festen Wert gesetzt, da so die Internal Power unabhängig von der Input Transition Time definiert ist.

```
power_lut_template (energy_template_7x7) {
  variable_1 : input_transition_time;
  variable_2 : total_output_net_capacitance;
  index_1 ("0.008, 0.04, 0.08, 0.12, 0.16, 0.224, 0.28");
  index_2 ("0.01, 0.06, 0.1, 0.15, 0.2, 0.25, 0.3");
}
```

Listing 21: Internal Power Template

Teil III
Einbindung in Synthese-Tools

12 Synthese-Tools

Im Folgenden werden die genutzten Synthese-Tools und die Einbindung von Liberty-Bibliotheken in diese beschrieben.

12.1 Design-Compiler

Die Synthese mit Design-Compiler wird über ein Skript gesteuert. Der Aufbau und die Funktionsweise des Skripts sollen im folgenden Abschnitt erläutert werden.

12.1.1 *Einlesung der Bibliothek und des Designs*

Zunächst liest der Design-Compiler die im übergeordneten Skript definierten Variablen ein (vgl. Listing 22). Dazu zählt auch der search_path. Mit dieser Variable wird der Pfad gesetzt, in dem Design-Compiler nach Bibliotheken sucht. Über read_lib wird die Liberty-Bibliothek mit Timing-Informationen eingelesen und in das .db-Format konvertiert. Die Konvertierung der Bibliothek übernimmt hierbei der Library-Compiler. Das Argument no_warnings unterdrückt die Ausgabe von Warnungen und wird genutzt, um die Lesbarkeit der Kommandozeile zu verbessern.

```
set search_path $env(LIB_DIR)
read_lib -no_warnings $env(LIB_NAME_TIMING).lib
list_libs
set target_library $env(LIB_NAME_TIMING).db
set link_library $env(LIB_NAME_TIMING).db
set HDL_FILES [glob $env(RTL_DIR)/*.v]
set check_design_allow_multiply_driven_nets_by_inputs_and_outputs true
read_file $HDL_FILES -autoread -top $env(TOP_LEVEL)
```

Listing 22: Einlesen von Variablen in Design-Compiler

Das Verilog-Design wird über den Befehl read_file eingelesen. Um mehrere Dateien einzulesen, muss dem Befehl eine Liste übergeben werden. Diese Liste wird zuvor der HDL_FILES-Variable zugeordnet. Dies geschieht über den glob-Befehl. Der glob-Befehl gibt eine Liste von Dateinamen zurück, die dem Argument entsprechen. In diesem Falle schließt das alle Dateien mit der Endung *.v* ein. Dies wird über die *Regular Expression* »*.v« erreicht.

12.1.2 *Logische Minimierung, Einlesen von Constraints und Kompilierung*

```
set_flatten -effort high -minimize multiple_output
read_sdc /user/abeer/SyntheseProjekt-Git/SyntheseProjekt/Scripts/genus_sdc.sdc
compile_ultra
```

Listing 23: logische Minimierung Einlesen von Constraints und Kompilierung

Der set_flatten-Befehl dient dazu, eine flache Hierarchie zu erzeugen (vgl. Listing 23). Über das Attribut *effort* wird der CPU-Aufwand definiert, mit dem dieser Optimierungsschritt durchgeführt werden soll. Das Attribut *minimize* gibt an, welche Minimierungsstrategie nach der Abflachung der Hierarchie genutzt werden soll. Der Befehl read_sdc dient der Einlesung einer Constraint-Datei. Daraufhin wird über compile_ultra das Design kompiliert. Hierbei ist compile_ultra mit einem höheren Rechenaufwand verbunden als compile und führt zu einem Ergebnis mit höherem Optimierungsgrad.

```
write_file -hierarchy -format verilog -output $env(OUTPUT_DIR)/$env(DESIGN_NAME)
    _designcomp.v
report_area > $env(REPORTS_DIR)/$env(DESIGN_NAME)_designcomp.rpt
report_timing -path_type end >> $env(REPORTS_DIR)/$env(DESIGN_NAME)_designcomp.rpt
report_power -analysis_effort low -include_input_nets >> $env(REPORTS_DIR)/$env(
    DESIGN_NAME)_designcomp.rpt
exit
```

Listing 24: Schreiben der Netzliste und Erzeugung der Reports

In Listing 24 sind die Funktionen zum Schreiben der Netzliste und zur Generierung der Reports dargestellt. Die Reports werden in eine Datei geschrieben, die den Namen des Designs sowie des Tools trägt, sodass diese später selektiert und relevante Informationen herausgefiltert werden können.

12.1.3 *Switching Activity*

Die Schaltaktivität (engl. Switching Activity) lässt sich für die Berechnung der Leistungsaufnahme in Design-Compiler auf folgende Arten definieren:

Einfache Schaltaktivität Bei der einfachen Schaltaktivität wird die statische Wahrscheinlichkeit (engl. Static Probability) und Umschaltrate (engl. Toggle Rate) definiert.

Zustandsabhängige Umschaltraten an Eingangspins Die interne Leistungscharakterisierung des Eingangspins einer Bibliothekszelle kann zustandsabhängig sein. Die Eingangspins von Instanzen solcher Zellen können mit zustandsabhängigen Umschaltraten annotiert werden.

Zustands- und pfadabhängige Umschaltraten an Ausgangspins Wenn die Beschreibung der Internal Power von Ausgangspins zustands- und pfadabhängig erfolgt ist, kann die Umschaltrate dieser ebenfalls zustands- und pfadabhängig beschrieben werden.

Zustandsabhängige statische Wahrscheinlichkeit Die Leakage Power von Zellen kann über zustandsabhängige LUTs charakterisiert werden. Hierbei können Zellen mit zustandsabhängiger statischer Wahrscheinlichkeit annotiert werden.

Die Beschreibung der Schaltaktivität in Design-Compiler erfolgt über die Parameter der einfachen Schaltaktivität. Dementsprechend wird die Umschaltrate und statische Wahrscheinlichkeit annotiert. Listing 25 zeigt den Befehlt set_switching_activity mit Attributen.

```
set_switching_activity -static_probability 0.5 -toggle_rate 1 -period 20 -type {
inputs}
```

Listing 25: Switching Activity in Design-Compiler

Die Attribute des Befehls haben folgende Bedeutung:

-static_probability 0.5 Das Attribut gibt die statische Wahrscheinlichkeit an. Die statische Wahrscheinlichkeit ist der Bruchteil der Zeit, in der sich das Signal auf logisch 1 befindet. Mit einem Wert von 0.5 wird das Signal so konfiguriert, dass es 50% der Zeit auf logisch 1 ist.

-toggle_rate 1 Die Umschaltrate ist die Rate, mit der das Signal zwischen logisch 0 und logisch 1 wechselt. Die Angabe der Umschaltrate macht es nötig eine Periode anzugeben, auf die sie sich bezieht. Aus der Umschaltrate und der Periode folgt die Frequenz, mit der sich das Signal ändert (vgl. Gleichung 22).

$$f = \frac{\text{toggle\_rate}}{\text{period}} = \frac{1}{20\text{ns}} = 50\text{MHz} \tag{22}$$

-period 20 Die hier angegebene Periode bezieht sich auf die Umschaltrate. Die Einheit der Periode ist die unter time_unit angegebene Einheit in der Liberty-Bibliothek.

-type inputs Mit dem type-Attribut kann angegeben werden auf welche Ein- bzw. Ausgänge die Schaltaktivität angewendet werden soll. Über den hier angegebenen Wert *inputs* werden die Eingangspins des Designs, sowie der hierachisch untergeordneten Zellen, einbezogen. Weitere mögliche Werte für das type-Attribut sind unter anderem ports, nets und registers.

12.1.4 *Switching Power*

Die Berechnung der Switching Power erfolgt nach Formel 23.

$$P_c = \frac{V_{dd}^2}{2} \sum_{\forall nets(i)} \left(C_{load_i} \cdot T_{toggle_i} \right) \tag{23}$$

Die Variablen haben dabei folgende Bedeutung:

P_c : Die gesamte Switching Power

T_{toggle_i} : Umschaltrate des Netzes i, in s^{-1}

V_{dd} : Versorgungsspannung

C_{Load_i} : Die gesamte kapazitive Last des Netzes i, einschließlich der parasitären Kapazität, der Gate-Kapazität und der Drain-Kapazität aller mit dem Netz i verbundenen Pins.

12.2 Genus

Die Synthese mit Genus wird über ein Skript konfiguriert. Hierbei werden zuerst die Variablen, welche aus dem Skript übernommen werden, gesetzt, um sie im Skript zu nutzen.

12.2.1 *Synthese-Skript*

```
    set LOCAL_DIR "[exec pwd]"
    set SYNTH_DIR $env(SYNTH_DIR)
    set TCL_PATH $env(TCL_DIR)
    set REPORTS_PATH $env(REPORTS_DIR)
    set LIB_PATH $env(LIB_DIR)
    set RTL_PATH $env(RTL_DIR)
    set DESIGN $env(DESIGN_NAME)
    set OUTPUTS_PATH $env(OUTPUT_DIR)
    set LEAKAGE_PWR_EFFORT high
    set GEN_EFF high
    set MAP_OPT_EFF high
    set HDL_FILES [glob $RTL_PATH/*.v]

    set lp_power_analysis_effort low
```

Listing 26: Zuweisung von Variablen in Genus

Über den set-Befehl werden die aus dem Skript übergebenen Variablen gesetzt. Wenn es sich bei den Variablen um eine Liste handelt, wird der glob-Befehl genutzt. Dieser Befehl dient dazu, den Inhalt des angegebenen Ordners aufzulisten (vgl. Listing 26). Durch die eckigen Klammern wird dieser Befehl ausgewertet und in der zugehörigen Variable gespeichert. Die Konfiguration der Power-Analyse erfolgt hier durch lp_power_analysis_effort low. Durch den Befehl wird nur die vom Nutzer angegebene Schaltaktivität genutzt.

```
# Elaboration
elaborate

# Read constaint File
read_sdc genus_sdc.sdc -stop_on_errors

syn_generic
syn_map
syn_opt
```

Listing 27: Elaborate und Synthese in Genus

Elaborate Der elaborate-Befehl erstellt eine Hierarchie, die aus einem Top-Level-Entwurf und referenzierten untergeordneten Modulen besteht. Während der Ausar-

beitung führt Genus die folgenden Aufgaben aus:

- erzeugt Datenstrukturen
- inferiert Register
- führt HDL-Optimierung auf höherer Ebene durch, wie die Entfernung von überflüssigem Code
- prüft die Semantik

read_sdc Durch den read_sdc-Befehl kann eine .sdc (Synopsys Design Constraint)-Datei eingelesen werden. Diese Datei enthält unter anderem Angaben zum Taktsignal.

syn_generic Nimmt ein Design als Input und synthetisiert es mithilfe von RTL- und Datenpfad-Optimierungen zu einer Netzliste von generischen Gates.

Die wichtigsten Schritte umfassen:

1. Optimierung von Datenpfadkomponenten wie Addierern, Multiplikatoren und Schieberegistern, um arithmetische Module zu implementieren, die gegebene Einschränkungen bei minimaler Fläche und Leistungsaufnahme einhalten. Mit dem Attribut dp_analytical_opt kann global der Kompromiss zwischen Timing und Area gesteuert werden.
2. Implementierung von Multiplexer-Strukturen und Auswahl der besten Architektur (binär oder one-hot) für diese Strukturen.
3. Verwendung von enable-Kondition im Fanin von Registern zur Implementierung von clock-gating-Logik. Dies dient der Leistungsreduzierung. Das Attribut zur Aktivierung dieser Funktion ist lp_insert_clock_gating. Das Attribut steht standardmäßig auf *off*.
4. Anwenden von Retiming, um die Leistungsfähigkeit des Designs zu verbessern. Diese Technik wird auf alle Module angewendet, die mit dem Attribut retime gekennzeichnet sind. Eine weitere Steuerung ist über Attribute der Kategorie retime möglich.

syn_map Durch den syn_map-Befehl erfolgt das Mapping eines Designs von generischen Gates auf eine Technologiebibliothek bei gleichzeitiger Optimierung der Leistung und Fläche. Nach einer anfänglichen Mapping-Phase erfolgt eine weitere Bearbeitung durch inkrementelle Optimierung der Netzliste, zur Reduzierung des Flächenbedarfs und der Leistungsaufnahme unter Beibehaltung des Timings.

Die wichtigsten Schritte umfassen:

1. Strukturierung und Mapping der generischen Logik unter Einhaltung gegebener Einschränkungen.

2. Selektive Aufhebung der Gruppierung von Modulen zur Optimierung des Timings oder des Flächenbedarfs.

3. Optimierung für Leakage- und Dynamic Power. Der Aufwand (engl. effort) für die Reduzierung der Leakage Power wird über das Attribut leakage_power_effort angegeben. Standardmäßig wird bei Genus keine Leistungsoptimierung durchgeführt.

4. Mapping von Einzelbit- auf Multibit-Bibliothekszellen. Standardmäßig ist diese Funktion ausgeschaltet. Sie kann über das use_multibit_cells-Attribut aktiviert werden.

Der dem Mapper zur Verfügung stehende Bibliothekszellensatz kann durch Anwendung des dont_use-Attributs auf Bibliothekszellenobjekte eingeschränkt werden.

Der während der Optimierung zu verwendende Aufwand wird durch das Attribut syn_map_effort gesteuert.

syn_opt Der syn_opt-Befehl nimmt ein gemapptes Design als Input und optimiert Timing, Flächenbedarf und Leistungsaufnahme. Wenn die Optionen -spatial oder -physical nicht angegeben sind, erfolgen die Optimierungen rein logisch.

siehe 12.2.2.

```
report_area -summary > $REPORTS_PATH/${DESIGN}_genus.rpt
report_timing -unconstrained >> $REPORTS_PATH/${DESIGN}_genus.rpt
report_power >> $REPORTS_PATH/${DESIGN}_genus.rpt
report_activity >> $REPORTS_PATH/${DESIGN}_genus.rpt
```

Listing 28: Reports in Genus

In dem in Listing 28 dargestellten Abschnitt des Synthese-Skripts werden die Reports erzeugt und in .rpt-Dateien geschrieben. Der >-Operator dient der Erzeugung und Schreiben des Reports. Die nachfolgenden Befehle nutzen den >>-Operator, um ihren Inhalt an die Datei anzuhängen. Der Name der Datei beinhaltet auf Grund der Verwendung der ${DESIGN}-Variable den Namen des Designs. Die Bedeutung der einzelnen Befehle wird im Folgenden erläutert:

report_area Erzeugt den Area-Report. Der Area Report enthält die folgenden Informationen:

- die Gesamtzahl der Zellen
- die Summe des Zellbereichs in jedem der Blöcke und dem Top-Level
- die Zellbereichsnummern basieren auf den Zellimplementierungen, die nach dem Mapping aus der Technologiebibliothek entnommen wurden
- die Net Area bezieht sich auf die geschätzte Post-Route-Netzfläche

- das für jeden der Blöcke genutzte wire load-Modell

report_timing Erstellt einen Timing-Report für den aktuellen Entwurf. Standardmäßig bietet der Bericht eine detaillierte Ansicht des kritischen Pfades des aktuellen Entwurfs. Es kann ebenfalls ein Bericht über mögliche Probleme mit Zeiteinschränkungen (Timing-Lint) erstellt werden oder Slack an Endpunkten angezeigt werden.

report_power Erzeugt den Report der Leistungsaufnahme. Der Bericht enthält die drei Komponenten der Leistungsaufnahme (Internal, Leakage und Switching Power) und gibt diese für die Kategorien memory, register, latch, logic, bbox, clock, pad und pm prozentual und absolut an.

report_activity Listet den Duty-Cycle und die Frequenz für die Kategorien memory, register, latch, logic, bbox, clock, pad und pm auf.

12.2.2 *Switching Activity*

Die Schaltungsaktivität wird in Genus über den Befehl set_pin_activity konfiguriert. Die Konfiguration ist in Listing 29 dargestellt.

```
set_pin_activity -activity_type user -duty 0.5 -freq 50e+6 -pin {a}
```

Listing 29: Switching Activity in Genus

-activity_type user

-duty 0.5 Legt den Duty-Cycle des Signals fest.

-freq 50e+6 Bestimmt die Frequenz des Signals.

-pin {a} Das Pin-Attribut legt fest, auf welche Pins des Designs der Stimulus angewandt wird. Es besteht die Möglichkeit Pins mithilfe des get_db-Befehls zu spezifizieren. So können beispielsweise alle Eingangsports mit dem Befehl »get_db ports -if {.direction == in}« ausgewählt werden.

12.2.3 *Switching Power*

Die Switching Power wird unter Genus nach Gleichung 24 berechnet.

$$P_{net} = \frac{V_{dd}^2}{2} \cdot C_L \cdot T_{toggle} \tag{24}$$

Die Variablen haben folgende Bedeutung:

P_{net} gesamte Switching Power

V_{dd} die Versorgungsspannung

C_L kapazitive Last des Netzes

T_{toggle} die Umschaltrate des Netzes

12.3 LeonardoSpectrum

Zur Nutzung benutzerdefinierter Bibliotheken in LeonardoSpectrum muss die Bibliothek in die Ordnerstruktur des Tools eingebunden werden. Abbildung 27 zeigt die Ordnerstruktur von LeonardoSpectrum. Die einzig nötige Datei ist <lib_name.syn>.

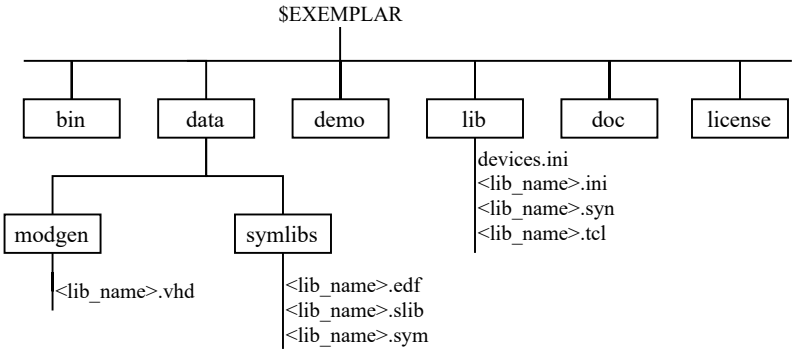

Abbildung 27: Ordnerstruktur LeonardoSpectrum

12.3.1 *Konvertierung in .syn-Format*

Verglichen mit anderen Synthese-Tools nutzt LeonardoSpectrum die Liberty-Files nicht direkt. Es ist nötig Liberty-Dateien zuerst in .syn-Dateien zu konvertieren. Diese Konvertierung geschieht über die Tools *syngen* und *lgen*. Die Befehle zur Konvertierung sind im Folgenden aufgeführt.

```
syngen <input_file> <output_file>
lgen <input_file> <output_file>
```

12.3.2 *Einbindung in die Nutzeroberfläche*

Syntax Um benutzerdefinierte Bibliotheken in der Nutzeroberfläche auswählen zu können, muss die `devices.ini` (vgl. Abb. 27 modifiziert werden. Dazu muss die neue Bibliothek unter `[DEVICES]` hinzugefügt werden. Dies geschieht durch Ergänzung eines Eintrags (siehe Listing 30).

```
DEVICE_245=xilinx_artix7
DEVICE_246=xilinx_kintex7
DEVICE_247=flexlogix_virtex4
DEVICE_248=fh_dortmund_device
```

Listing 30: Auszug aus modifizierter devices.ini

Des Weiteren muss das Modul in der Datei ergänzt werden.

```
[fh_dortmund_device]
CONTACT=
DIRECTION=BOTH
FAMILY=FHDortmund
FORM=FHDortmund
HTML_PAGE=www.fh-dortmund.de
INI_FILE=
LIBRARY_NAME=fh_do_dev_lib
LICENSE_PACKAGE_NAME=act
MANUFACTURER=FHDortmund
NUMBEROFPASSES=4
PROPTIONS=TRUE
IPOFLOW=TRUE
SYMBOL_LIBRARY=
TECHNOLOGY_TYPE=FPGA
VENDOR_NAME=FH Dortmund
BMPFILE=logo_fh_bmp
```

Listing 31: Benutzerdefinierte Library in devices.ini

Bei dem Modul in Listing 31 ist zu beachten, dass die Angabe `LIBRARY_NAME` dem Namen der .syn-Datei entsprechen muss. Die Bibliothek in der Nutzeroberfläche ist in Abbildung 28 dargestellt.

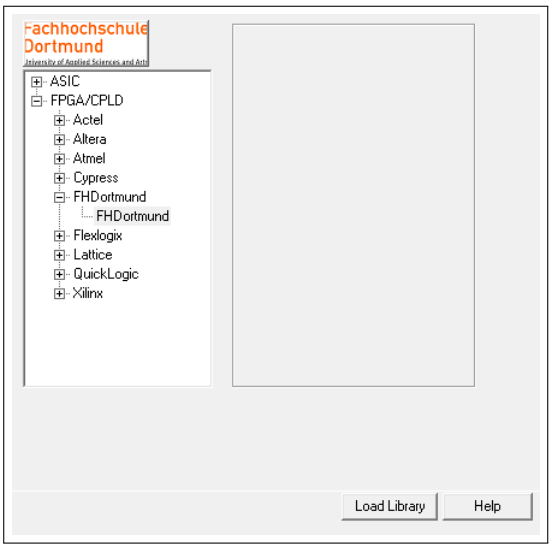

Abbildung 28: LeonardoSpectrum GUI

12.3.3 *Start der Synthese*

Der Synthesevorgang muss bei erfolgreicher Einbindung der .syn-Datei über die Reiter des Tools konfiguriert werden.

Technology Unter *Technology* muss die Bibliothek geladen werden.

Input Als *Input* muss das einzulesende Verilog-Design gewählt werden.

Optimize Unter *Optimize* werden die Optimierungen konfiguriert.

Output Unter *Output* muss das Format der Ausgabe konfiguriert werden. In Abbildung 29 ist eine Konfiguration auf das Output-Format Verilog dargestellt.

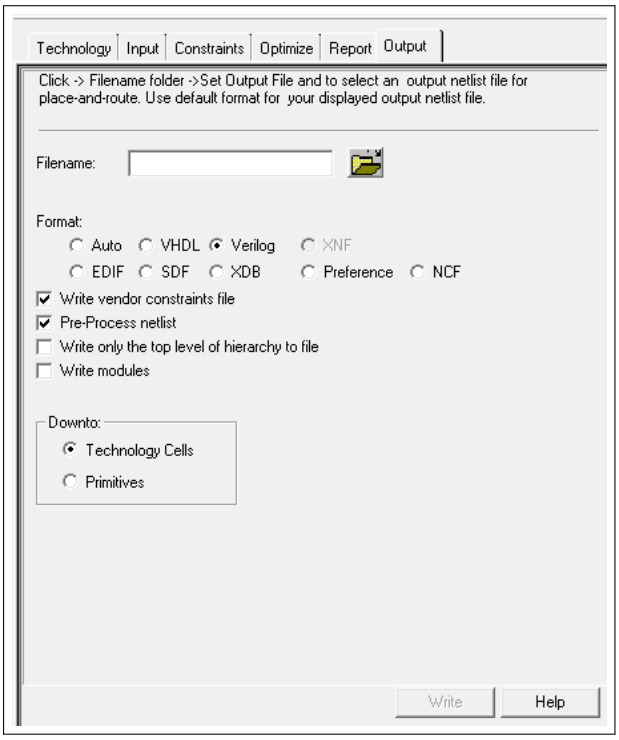

Abbildung 29: LeonardoSpectrum Output Format

Alternativ dazu lässt sich die Synthese per tcl-Skript konfigurieren und starten.

12.3.4 *Synthese über TCL-Skript*

LeonardoSpectrum akzeptiert alle Befehle der Tcl-Sprache. Zur Nutzung eines TCL-Skripts kann die GUI von LeonardoSpectrum umgangen werden. Hierfür muss das Tool über den Befehl `spectrum -file <skript>` gestartet werden. Das genutzt Skript zeigt Listing 32.

```
load_library fh_do_dev_lib
set_working_dir /user/abeer/LeonardoSpectrum
read  { /user/abeer/cadence_genus/rtl/counter.v }
pre_optimize -common_logic -unused_logic -boundary -xor_comparator_optimize
pre_optimize -extract
optimize .work.counter.INTERFACE -chip -auto -effort standard -hierarchy auto
optimize_timing .work.counter.INTERFACE
set novendor_constraint_file FALSE
auto_write -format Verilog counter.v
```

Listing 32: TCL-Skript zur Synthese mit LeonardoSpectrum

Zum Verständnis des Synthesevorgangs mittels TCL-Skript werden im Folgenden die genutzten Befehle und Argumente erläutert.

Befehle

load_library Lädt die gewählte Library

set_working_directory Legt das Arbeitsverzeichnis fest

read Lädt die VHDL/Verilog-Module

pre_optimize Der Befehl pre_optimize führt eine technologieunabhängige Logikoptimierung durch, z. B. gemeinsame Nutzung von Ressourcen, Entfernung ungenutzter Logik und Extraktion von Datenpfadelementen, wie z.B. Zähler, Decoder und RAMs. LeonardoSpectrum führt immer den Befehl pre_optimize als Teil des Optimierungsprogramms aus. Die Funktionsweise des Befehls lässt sich über Argumente genauer steuern. Im genutzten TCL-Skript (Listing 32) werden die Argumente common_logic, unused_logic, boundary, xor_comparator_optimize und extract genutzt.

Attribute

-common_logic Nutzung gemeinsamer Ressourcen von Operatoren und Primitives, die gleiche Eingänge haben.

-unused_logic Entfernt Logikelemente, die auf den Output der Schaltung keinen Einfluss haben.

-boundary Propagiert Konstanten, d.h. auf *high* oder *low* gebundene Eingänge.

-xor_comparator_optimize Optimiere weite XORs und Komparatoren.

-extract Detektiere counter, decoder und RAM-Zellen in generischer Logik.

optimize Der optimize-Befehl führt eine technologiespezifische Logikoptimierung und das Technologie-Mapping durch. Der Optimierungs- und Mappingaufwand wird gesteuert, indem Variablen und Einschränkungen (engl. *constraints*) für

die Entwurfsgrenzen und die Angabe von Flächen- und Verzögerungsoptionen gesetzt werden. Der reguläre Ablauf der Optimierung besteht darin, dass jede Hierarchieebene, beginnend von der obersten, optimiert wird.

optimize_timing Der Befehl optimize_timing untersucht zeitkritische Pfade des Designs und versucht diese im Hinblick auf die Ankunftszeit zu optimieren. Angaben des Nutzers zum Taktsignal, Ankunfszeit der Eingangssignale und timing-Informationen zu Ausgangssignalen verbessern die Effektivität dieses Optimierungsschritts, da aus diesen Informationen timing-constraints gewonnen werden. Angaben zu Taktsignalen (siehe Abbildung 30) können in Form von Attributen gemacht werden.

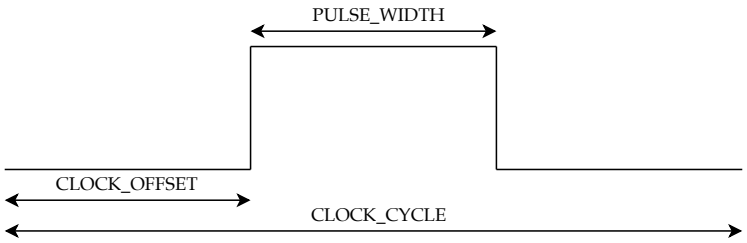

Abbildung 30: optimize_timing-Parameter

```
CLOCK_CYCLE 20 clock
CLOCK_OFFSET 5 clock
PULSE_WIDTH 10 clock
ARRIVAL_TIME 3 inputa
```

Die numerischen Angaben beziehen sich hierbei auf Nanosekunden. Sollen die Angaben über eine VHDL-Datei erfolgen, wird die Syntax folgendermaßen eingegeben.

```
ATTRIBUTE CLOCK_CYCLE OF clock:SIGNAL is 20ns;
ATTRIBUTE CLOCK_OFFSET OF clock:SIGNAL is 5ns;
ATTRIBUTE PULSE_WIDTH clock:SIGNAL is 10ns;
ATTRIBUTE ARRIVAL_TIME OF inputa:SIGNAL is 3ns;
```

set siehe 4.2.3

auto_write Dieser Befehl führt den Schreibbefehl aus. Zusätzlich dazu führt er die erforderliche Verarbeitung für die gewählte Zieltechnologie einschließlich dem Setzen von Variablenwerten und Aufruf von generate_timespec, decompose_luts und apply_rename_rules aus.

12.4 Yosys

Yosys ist ein Open-Source Synthese-Tool, das 2012 von Clifford Wolf[24] entwickelt wurde. Die Befehle zur Konfiguration der Synthese können entweder manuell eingegeben oder beim Aufruf des Programms über ein Skript spezifiziert werden. Yosys unterscheidet sich von anderen Synthese-Tools dadurch, dass keine Timing- oder Power-Angaben unterstützt werden. Daher muss die benutzerdefinierte Bibliothek so modifiziert werden, dass sie keine *timing*- beziehungsweise *power*-Gruppen enthält. Des Weiteren unterstützt Yosys keine Multi-Output Zellen, wie den in der Bibliothek enthaltenen Full-Adder.

12.4.1 *Synthese-Skript*

Das vollständige Skript zur Synthese eines Designs ist in Listing 33 aufgeführt. Das tcl-Skript wird Yosys innerhalb des Skripts übergeben. Mit dem Befehl »help <command>« lassen sich weitere Informationen zu den Befehlen aufrufen.

```
set HDL_FILES [glob $env(RTL_DIR)/*.v]
split $HDL_FILES " "
for {set i 0} {$i < [llength $HDL_FILES]} {incr i} {
    yosys read_verilog [lindex $HDL_FILES ${i}]
}
yosys hierarchy -check -top $env(TOP_LEVEL)
yosys proc;
yosys flatten;
yosys opt;
yosys memory;
yosys opt;
yosys techmap;
yosys opt;
yosys dfflibmap -liberty $env(LIB_DIR)/$env(LIB_NAME).lib
yosys abc -liberty $env(LIB_DIR)/$env(LIB_NAME).lib
yosys clean
yosys tee -o $env(REPORTS_DIR)/$env(DESIGN_NAME)_yosys.rpt stat -liberty $env(LIB_DIR
    )/$env(LIB_NAME).lib
yosys write_verilog -noattr $env(OUTPUT_DIR)/$env(DESIGN_NAME)_yosys.v
```

Listing 33: Skript zur Synthese eines Designs in Yosys

Im Folgenden sollen die im Synthese-Skript (vgl. Listing 33) verwendeten Befehle erläutert werden.

hierarchy In parametrischen Designs kann ein Modul in mehreren Varianten mit unterschiedlichen Parameterwerten existieren. Dieser Durchlauf betrachtet alle Module im aktuellen Entwurf und führt die Sprach-Frontends für die parametrischen Module bei Bedarf erneut aus.

proc Dieser Befehl ruft alle weiteren proc-Befehle in der dargestellten Reihenfolge auf.

1. **proc_clean** Entfernt leere Teile von Prozessen und ebenso Prozesse, die ausschließlich leere Strukturen enthalten.
2. **proc_rmdead** Identifiziert nicht erreichbare Zweige in Entscheidungsbäumen und entfernt diese.
3. **proc_init** Extrahiert die *init*-Aktionen von Prozessen (generiert aus Verilog *initial*-Blöcken) und setzt, darauf basierend, Anfangswerte.
4. **proc_arst** Identifiziert asynchrone Resets in den Prozessen und wandelt sie in eine andere interne Darstellung um, die für die Generierung von Flip-Flop-Zellen mit asynchronen Resets geeignet ist.
5. **proc_mux** Wandelt die Entscheidungsbäume in Prozessen (die aus if-else- und case-Anweisungen stammen) in Bäume von Multiplexerzellen um.
6. **proc_dlatch** Identifiziert Latches in den Prozessen und wandelt sie in D-Latches um.
7. **proc_dff** Identifiziert Flip-Flops in den Prozessen und wandelt sie in DFF-Zellen um.
8. **proc_clean** siehe Punkt 1

flatten Flacht den Entwurf ab, indem er Zellen durch ihre Implementierung ersetzt. Dieser Durchlauf ist dem *techmap*-Durchlauf sehr ähnlich. Der einzige Unterschied besteht darin, dass dieser Durchlauf den aktuellen Entwurf als Mapping-Bibliothek verwendet. Zellen beziehungsweise Module, bei denen das Attribut *keep_hierarchy* gesetzt ist, werden durch diesen Befehl nicht abgeflacht.

opt Dieser Befehl ruft opt_*-Befehle in einer vorgegebenen Reihenfolge auf. Er führt Optimierungen und Bereinigungen durch.

memory Konvertiert Speicher in D-Flipflops und Adressdecoder oder Multiport-Speicherblöcke, wenn sie mit der Option -nomap aufgerufen werden.

techmap Dieser Befehl implementiert einen Technologie-Mapper, der Zellen im Entwurf durch Implementierungen ersetzt, die in Form einer Verilog- oder ilang-Quelldatei angegeben sind.

dfflibmap Abbildung der internen Flip-Flop-Zellen auf die Flip-Flop-Zellen in der Liberty-Bibliothek. Dieser Durchlauf kann bei Bedarf Inverter hinzufügen. Daher wird empfohlen, zuerst diesen Durchgang auszuführen und dann die Logikpfade auf die Zieltechnologie abzubilden. Wenn dieser Befehl mit -prepare aufgerufen wird, konvertiert er die internen Flipflop-Zellen in die internen Zelltypen, die am besten zu den in der angegebenen Liberty-Bibliothek gefundenen Zellen passen.

abc Verwendet das ABC-Tool für das Technologie-Mapping der yosys-internen Gatterbibliothek auf eine Zielarchitektur.

clean Dieser Befehl ist identisch mit 'opt_clean'.

stat Druckt einige Statistiken (Anzahl der Zellen) über den ausgewählten Teil des Designs.

write_verilog Schreibt den aktuellen Entwurf in eine Verilog-Datei.

12.5 Yosys_Gatemate

Yosys_Gatemate stellt eine Erweiterung des Yosys-Tools dar, welche auf die Synthese von Netzlisten zur Implementierung auf dem Gatemate FPGA abzielt. In Listing 34 ist das TCL-Skript zur Nutzung des Tools dargestellt.

```
set HDL_FILES [glob $env(RTL_DIR)/*.v]
split $HDL_FILES " "
for {set i 0} {$i < [llength $HDL_FILES]} {incr i} {
    yosys read_verilog [lindex $HDL_FILES ${i}]
}
yosys synth_gatemate
yosys tee -o $env(REPORTS_DIR)/$env(DESIGN_NAME)_yosys_gatemate.rpt stat
yosys write_verilog -noattr $env(OUTPUT_DIR)/$env(DESIGN_NAME)_yosys_gatemate.v
```

Listing 34: yosys_gatemate.tcl

Zur Sammlung der HDL-Dateien in einer Liste wird der glob-Befehl auf den RTL-Ordner aufgeführt, sodass alle Verilog Dateien erfasst werden. Daraufhin werden die Dateien über read_verilog eingelesen. Der Befehl synth_gatemate führt die Snythese für die Gatemate-Erweiterung aus. Mithilfe des Befehls stat werden die relevanten Parameter zur Generierung des Reports ausgegeben. Diese Parameter werden in eine .rpt-Datei geschrieben, die mit dem Namen des aktuellen Designs, sowie des Tools gekennzeichnet ist. Zuletzt wird die Netzliste mit write_verilog im Verilog-Format in den Output-Ordner geschrieben.

12.6 Xilinx Vivado

Vivado ist Teil des Toolflows, obwohl es die Einbindung benutzerdefinierter Liberty-Bibliotheken nicht erlaubt. Grund für die Aufnahme in den Toolflow ist die Möglichkeit, die erstellte Bibliothek mit kommerziellen Bibliotheken hinsichtlich der Gatterzahl, Leistungsaufnahme und Timing zu vergleichen. Im Folgenden wird das Skript vorgestellt, mit dem das Design synthetisiert wird.

12.6.1 *Synthese-Skript*

Das Synthese-Skript ist in Listing 35 dargestellt. Zu Beginn wir die Variable outputDir auf den, über $env(OUTPUT_DIR) übermittelten String, gesetzt. Daraufhin werden die Verilog Dateien über read_verilog eingelesen.

```
# Set Output Directory
set outputDir $env(OUTPUT_DIR)

# Design Sources
read_verilog [ glob $env(RTL_DIR)/*.v ]

# Run Synthesis
synth_design -top $env(TOP_LEVEL) -part xc7k70tfbg676-2 $outputDir/post_synth.dcp
report_timing_summary -file $env(REPORTS_DIR)/post_synth_timing_summary.rpt
report_utilization -file $env(REPORTS_DIR)/post_synth_util.rpt

# Logic Optimization
opt_design
place_design
route_design
report_route_status -file $env(REPORTS_DIR)/post_route_status.rpt
report_timing_summary -file $env(REPORTS_DIR)/post_route_timing_summary.rpt
report_power -file $env(REPORTS_DIR)/post_route_power.rpt
report_drc -file $env(REPORTS_DIR)/post_imp_drc.rpt
write_verilog -force $outputDir/$env(DESIGN_NAME)_vivado.v
```

Listing 35: Xilinx Vivado Synthese-Skript

synth_design Synthetisiert ein über read_verilog eingelesenes Design. Über das top-Attribut muss, im Falle eines hierarchisch aufgebauten Designs, das Top-Level angegeben werden. Das part-Attribut gibt an, welche Zielarchitektur genutzt werden soll.

report_timing_summary Generiert eine Zusammenfassung der Timing-Informationen. Durch das Attribut *-file* wird der Bericht in die angegebene Datei geschrieben.

report_utilization Erzeugt einen Bericht über die Ressourcennutzung durch den aktuell synthetisierten oder implementierten Entwurf. Der Bericht wird an die

Standardausgabe zurückgegeben, sofern nicht die Optionen -file, -return_string oder -namearguments angegeben sind.

opt_design Optimiert die aktuelle Netzliste. Durch den Befehl werden standardmäßig die Optimierungsschritte *retarget*, *propconst*, *sweep* und *bram_power_opt* durchgeführt.

place_design Platziert die angegebenen Ports und Logikzellen des aktuellen Designs auf den Ressourcen der Zieltechnologie. Das Werkzeug optimiert die Platzierung, um negativen Slack zu minimieren und die wire-Länge zu reduzieren. Gleichzeitig wird versucht die Platzierung zu verteilen, um Problemen beim Routing vorzubeugen.

route_design Führt das Routing für die Netze im aktuellen Entwurf aus, um logische Verbindungen auf der Zieltechnologie zu vervollständigen.

report_route_status Meldet den Status des Routings im aktuellen Entwurf.

report_power Führt eine Power-Analyse für den aktuellen Entwurf durch und meldet Details zur Leistungsaufnahme auf Grundlage der aktuellen Betriebsbedingungen und Schaltraten des Designs. Die Betriebsbedingungen können mit dem Befehl set_operating_conditions festgelegt werden. Die Schaltaktivität kann mit dem Befehl set_switching_activity festgelegt werden.

report_drc Prüft den aktuellen Entwurf anhand eines festgelegten Satzes von Design Rules und meldet gefundene Fehler.

write_verilog Schreibt die aktuelle Netzliste im Verilog-Format in die angegebene Datei.

13 Verifikation

13.1 Cadence Conformal

Der Cadence Logic Equivalence Check dient der Prüfung zweier Entwürfe auf funktionale Äquivalenz. Für die Erstellung eines Skriptes zur automatisierten Äquivalenzprüfung gibt es zwei Möglichkeiten. Zum Einen kann das Skript per Hand geschrieben und an Conformal übergeben werden. Zum Anderen ist es möglich, ein Skript von Genus, über den write_do_lec-Befehl, erzeugen zu lassen. Im folgenden Kapitel wird zunächst das manuell erstellte Skript vorgestellt.

13.1.1 *Skript*

```
if {$env(TOOL)=="yosys"} {
    read_library -Both -Replace  -sensitive  -Statetable  -Liberty $env(LIB_DIR)/$env
    (LIB_NAME).lib -nooptimize
} else {
    read_library -Both -Replace  -sensitive  -Statetable  -Liberty $env(LIB_DIR)/$env
    (LIB_NAME_TIMING).lib -nooptimize
}
set HDL_GOLDEN [glob $env(RTL_DIR)/*.v]
set HDL_REVISED [glob $env(OUTPUT_DIR)/*.v]
read_design $HDL_GOLDEN -Verilog -Golden -sensitive -continuousassignment
    Bidirectional -nokeep_unreach  -nosupply
read_design $env(OUTPUT_DIR)/$env(DESIGN_NAME)_$env(TOOL).v -Verilog -Revised
    -sensitive -continuousassignment Bidirectional -nokeep_unreach -nosupply
set_system_mode lec
add_compared_points -all
compare
report_compared_points -summary > $env(REPORTS_DIR)/$env(DESIGN_NAME)_$env(TOOL)
    _lec.rpt
report_verification -summary -verbose > $env(REPORTS_DIR)/$env(DESIGN_NAME)_$env(TOOL
    )_lec.rpt
exit
```

Listing 36: Cadence LEC Skript

Zu Beginn des Skripts erfolgt eine if-Abfrage, die dazu dient, die passende Bibliothek auszuwählen. Falls die TOOL-Variable den String yosys enthält, wird über read_library die Bibliothek ohne Timing-Informationen eingelesen. Sollte der String ungleich »yosys« sein, wird die Bibliothek mit Timing-Informationen gewählt. In den Zeilen 5 und 6 werden die Variablen HDL_GOLDEN und HDL_REVISED gesetzt. Diesen Variablen wird mithilfe des glob-Befehls eine Liste aller Verilog-Dateien zugewiesen, die sich im RTL- beziehungsweise OUTPUT-Ordner befinden. Die zugewiesenen Listen werden daraufhin von dem read_design-Befehl genutzt, welcher die beiden Netzlisten einliest. Die beim read_design-Befehl genutzten Attribute haben folgende Funktion:

-Verilog Gibt den Dateityp an.

-Golden -Revised Kennzeichnet die zu vergleichenden Netzlisten.

-sensititive Das Attribute gibt an, ob die Groß- und Kleinschreibung des Designs zu beachten ist.

-continuousassignment Bidirectional Legt fest, dass die kontinuierliche Zuordnung im Design als bidirektionale Zuordnung interpretiert werden soll.

-nokeep_unreach Entfernt alle nicht erreichbare D-Flipflops oder Latches in einem Modul während der RTL-Synthese.

-nosupply Konvertiert die supply0- und supply1-Netze in Netze des Typs wire.

Durch den Befehl set_system_mode lec wechselt Conformal in den Logic Equivalence Check Modus. Der Befehl add_compared_points -all dient dazu alle Vergleichspunkte zur Äquivalenzprüfung zu nutzen. Daraufhin wird über compare der Vergleich gestartet. Die Befehle report_compared_points und report_verification schreiben das Ergebnis der Äquivalenzprüfung in die angegebenen Dateien.

13.1.2 *Generiertes Conformal-Skript*

Neben der Möglichkeit die tcl-Skripte für Conformal manuell zu schreiben, gibt es die Möglichkeit eine do-Datei von Genus generieren zu lassen. Zur Generierung dieses Skripts wird der write_do_lec-Befehl genutzt (vgl. Listing 37). Dazu ist es erforderlich, dass Genus die Liberty-Bibliothek sowie die HDL-Dateien einliest und den elaborate-Vorgang durchführt.

```
0 write_do_lec -flat -no_exit -revised_design ${OUTPUTS_PATH}/${DESIGN}_${TOOL}.v
      -logfile $REPORTS_PATH/rtl2final.lec.log > $REPORTS_PATH/${DESIGN}_${TOOL}.lec.do
```

Listing 37: Befehl zur Generierung eines Conformal Skripts

Die Attribute des write_do_lec-Befehls:

-flat Führt den Vergleich der Dateien in flacher Hierarchie aus.

-no_exit Der exit-Befehl wird nicht an das Ende des erzeugten Skripts gesetzt.

-revised_design Gibt an welches Design mit dem unter *golden* eingelesenen Design verglichen werden soll.

-logfile Der Name des Logfiles.

Innerhalb des generierten Skripts werden Attribute durch den set_flatten_model-Befehl gesetzt (vgl. Listing 38).

```
set_flatten_model -seq_constant
set_flatten_model -seq_constant_x_to 0
set_flatten_model -nodff_to_dlat_zero
set_flatten_model -nodff_to_dlat_feedback
set_flatten_model -hier_seq_merge
```

Listing 38: set_flatten_model-Befehl

Die Funktion dieser Attribute soll im Folgenden erläutert werden.

-seq_constant Synthese-Tools sind oft in der Lage unnötige Register zu entfernen, wenn ihr Ausgangssignal einer Konstante entspricht. Dieses Attribut wird verwendet, um die Optimierung von Registern mit konstanten Ausgangswerten in Conformal zu berücksichtigen (siehe Abb. 31).

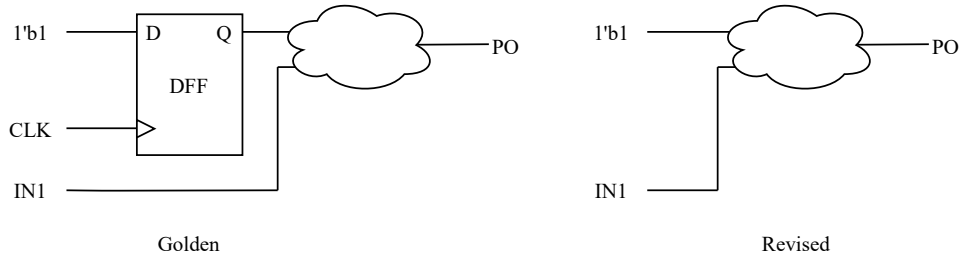

Abbildung 31: Register mit konstanten Eingängen werden optimiert

-seq_constant_x_to 0 Die Synthese kann ein D-Flipflop auf eine konstante 0 oder 1 optimieren, wenn der Eingang eine direkte Rückkopplung ist und das Setzen sowie Rücksetzen deaktiviert ist. Um eine ähnliche Modellierung in Conformal durchzuführen, ist das Attribut -seq_constant_x_to erforderlich. Dies ist ein globales Attribut und optimiert alle Register dieser Art auf die angegebene Konstante.

-nodff_to_dlat_zero Das dff_to_dlat_zero-Attribut konvertiert ein D-Flipflop zu einem Latch, wenn der Clock-Port konstant Null ist. Diese Umwandlung (vgl. Abbildung 32) wird in der Regel durch die Synthese durchgeführt, da die Funktionalität erhalten bleibt und die Gatteranzahl geringer ist. Somit werden sowohl Fläche als auch Leistung erhalten.

-nodff_to_dlat_feedback Ein rückgekoppeltes D-Flipflop wird als Latch mit Rückkopplung betrachtet. Daher optimieren Synthesetools solche DFFs gegebenenfalls zu Latches. Um dieses Szenario zu behandeln, wird -dff_to_dlat_feedback verwendet. Es wandelt diese DFFs in Latches um (vgl. Abb. 33).

-hier_seq_merge Aktiviert die benutzerdefinierte sequentielle Zusammenführung (engl. merging) bei der hierarchischen Dofile-Generierung. Die sequentielle

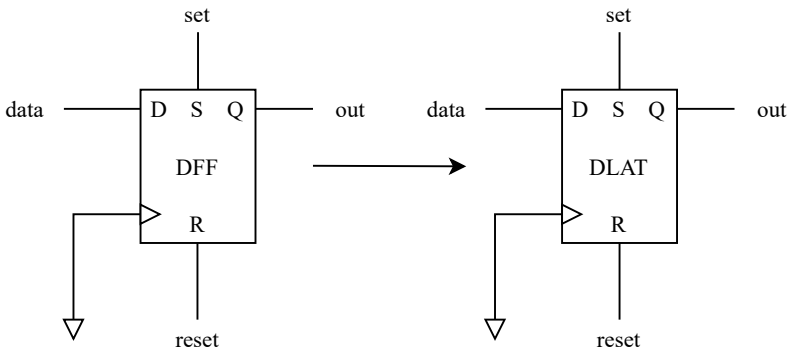

Abbildung 32: dff_to_dlat_zero

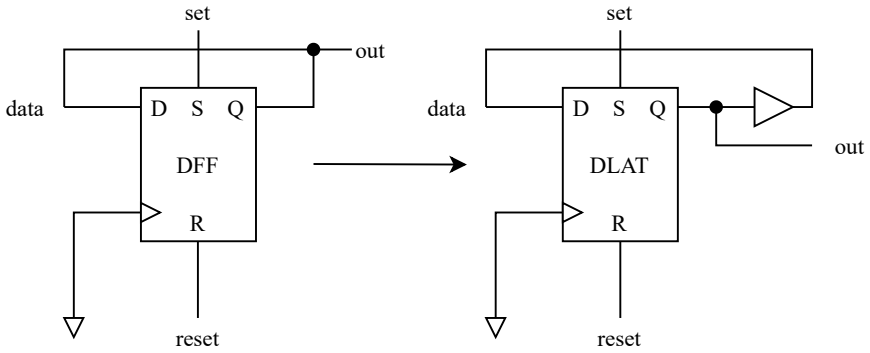

Abbildung 33: nodff_to_dlat_feedback

Zusammenführung wird nach dem Flattening und vor der constraint-Extraktion geprüft. Wenn Antivalenzen gefunden werden, flacht Conformal die Designs ab und überspringt diese sequentiellen Zusammenführungen.

13.2 Formality

Zur logischen Äquivalenzprüfung der mit Design-Compiler synthetisierten Netzlisten wird das Tool Formality genutzt. Das Skript zur Äquivalenzprüfung ist in Listing 39 dargestellt.

```
set_svf verif.svf
read_db -r $env(LIB_DIR)/$env(LIB_NAME_TIMING).db
set HDL_GOLDEN [glob $env(RTL_DIR)/*.v]
read_verilog -r $HDL_GOLDEN
set_top r:/WORK/$env(TOP_LEVEL)
set_reference r:/WORK/$env(TOP_LEVEL)

read_db -i $env(LIB_DIR)/$env(LIB_NAME_TIMING).db
read_verilog -i -netlist $env(OUTPUT_DIR)/$env(DESIGN_NAME)_$env(TOOL).v
set_top i:/WORK/$env(TOP_LEVEL)
set_impl i:/WORK/$env(TOP_LEVEL)
match
verify > $env(REPORTS_DIR)/$env(DESIGN_NAME)_$env(TOOL)_lec.rpt
exit
```

Listing 39: Formality TCL-Skript

Da das Formality-Skript ausschließlich für die Prüfung der durch Design-Compiler synthetisierten Netzlisten genutzt wird, ist es möglich auf eine svf-Datei zurückzugreifen. Die svf-Datei enthält die durch Formality ausgeführten Optimierungsschritte und dient somit der Konfiguration der Äquivalenzprüfung.

Zu Beginn des Skripts wird die svf-Datei über den set_svf-Befehl eingelesen. Daraufhin wird die db-Bibliothek eingelesen. Die Bibliothek befindet sich bereits im db-Format, da die Konvertierung durch Design-Compiler vorgenommen wird. Das Einlesen der Designs geschieht über read_verilog. Ob das Design als Referenz oder als Implementierung eingelesen wird, lässt sich über das Argument -i beziehungsweise -r konfigurieren. Über verify wird die Äquivalenzprüfung durchgeführt und das Ergebnis wird anschließend in eine rpt-Datei geschrieben.

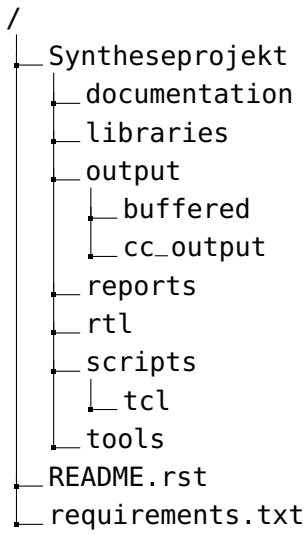

Abbildung 34: Aufbau Repository

Teil IV

Optimierung

14 Repository

Zur Versionsverwaltung der Liberty-Bibliothek, sowie der Skripte wurde ein GitHub-Repository genutzt. In Abbildung 34 ist der Aufbau des Github-Repositories gezeigt. In dem Hauptverzeichnis befinden sich eine readme-Datei und die requirements-Datei. Die readme-Datei wird innerhalb des Github Repository angezeigt. Das Format der Datei ist die Markup-Sprache reStructuredText (rst). reStructuredText hat den Vorteil in Textform leicht lesbar zu sein und gleichzeitig erweiterte Befehle zur Dokumentation von Repositories zur Verfügung zu stellen. So können beispielsweise Tabellen und Listen leicht dargestellt werden.

Die requirements-Datei dient dazu, die Installation aller benötigten Module zu vereinfachen. Mithilfe des in Listing 40 gezeigten Befehls können die Module zur Nutzung der Skripte direkt installiert werden.

```
pip install -r requirements.txt
```

Listing 40: Installation der benötigten Module

Die Unterordner im Verzeichnis Syntheseprojekt enthalten folgende Dateien:

documentation: Enthält die Dokumentation der Zellen im Verilog Format sowie im

Markdown-Format.

libraries: Enthält die Zellbibliotheken.

output: Enthält die synthetisierten Netzlisten

buffered: Die Netzlisten, welche durch das create_buffer-Skript bearbeitet wurden, werden hier gespeichert.

cc_output: Der Output der CC-Tools wird hier abgelegt.

reports: In diesem Ordner werden die von den Tools erzeugten Reports sowie die Ergebnisse der Optimierung gespeichert.

rtl: Enthält die Testdesigns.

scripts: Enthält die Python- und Shell-Skripte.

scripts/tcl: Enthält die TCL-Skripte zur Konfiguration und Steuerung der Tools sowie die constraint-Dateien.

tools: Zur Nutzung der Cologne Chip-Tools XConvert und pnr müssen die Tools in diesem Ordner vorhanden sein.

15 Buffer

Zur Konvertierung der synthetisierten Netzlisten durch das Tool XConvert ist es notwendig, Eingangs- und Ausgangsbuffer in die Netzlisten einzufügen. Da diese Aufgabe nicht von den Synthese-Tools übernommen wird, dient hierzu das Skript create_buffer.

15.1 Übersicht

Das create_buffer-Skript ist in Python geschrieben und sucht innerhalb der Netzliste nach input- und output-Ausdrücken, um über diese Ausdrücke Eingangs- und Ausgangsports zu identifizieren. Sind die Ports identifiziert, werden sie zusammen mit ihrer Bitweite in eine Liste von Objekten der Klasse Port gespeichert. Hierbei dient die Klasse Port als eine Hilfsklasse, welche mit den Attributen name, direction, width und clock deklariert ist. Das Attribut direction dient der Unterscheidung zwischen Eingangs- und Ausgangsports. Mit width wird die Weite des Busses gespeichert, sodass diese Weite auch im Buffer erzeugt wird. Das Attribut clock dient der Markierung des Taktsignals und nimmt eine Sonderstellung in der Buffer-Generierung ein, da für das Taktsignal die BUFG-Zelle erzeugt wird.

Anhand der gespeicherten Ports werden daraufhin die Buffer, das heisst IBUF für Eingangsports und OBUF für Ausgangsports in die Netzliste eingefügt. Die Buffer werden daraufhin über wire mit den Ports verbunden. Die Verbindung unterscheidet sich hierbei zwischen Eingangs- und Ausgangsports dahingehend, dass die Eingangsports mit den Eingängen der Eingangsbuffer und die Ausgangsports mit den Ausgängen der Ausgangsbuffer verbunden werden. Hierfür ist die Generierung spezieller wire-Signale notwendig. Diese wire-Signale werden anhand der Eingangs- und Ausgangsports generiert. Hierbei wurde eine Namenskonvention gewählt, bei der die generierten wire-Signale für die Buffer mit dem Affix _buf gekennzeichnet werden. Sind die Buffer mit den wire-Signalen verbunden, wird innerhalb der Verschaltung der Zellen nach den Signalnamen der wire gesucht, um diese ebenfalls um den Affix _buf zu ergänzen. Im Folgenden soll der Prozess zur Buffer-Generierung anhand des Skripts genauer erläutert werden.

15.1.1 *Module*

```
0 import re
  import datetime
```

Listing 41: Module

Mithilfe des Imports Befehls werden die Module re, datetime und pdb importiert. Das Modul re dient der Einfügung und der Suche nach regulären Ausdrücken. Das Modul datetime bietet die Funktionalität, die konvertierte Netzliste mit dem aktuellen Datum und der aktuellen Zeit zu kennzeichnen. Dies soll dem Nutzer des Skripts helfen, bei Änderungen innerhalb des Synthese-Ablaufs die Aktualität der konvertierten Netzliste zu erkennen.

15.1.2 *Klasse Port*

```
5  class Port:
       def __init__(self, name, direction, width=0, clock=False):
           self.name = ''.join(name)
           self.direction = direction
           self.width = int(width)+1 if width is not None else 1
10         self.clock = clock
```

Listing 42: Klasse Port

In Listing 42 ist die Klasse Port aufgeführt. Die Klasse dient als Hilfsklasse zur Kategorisierung gefundener Ports innerhalb der Netzliste. Mithilfe des name-Attributes wird der Name des Ports festgehalten. Da dieser Name als Listenelement vorliegt, wird er dem Attribut über die Methode join als String zugewiesen. Das Attribut direction wird bei der Generierung von Objekten der Klasse direkt angegeben und dient der Unterscheidung zwischen Eingangs- und Ausgangsports. Die Weite der Ports wird über das Attribut width festgelegt und hat den default-Wert 0. Dieser default-Wert ist in Zeile 6 mit width=0 angegeben. Die Weite wird bei Zuweisung immer um 1 inkrementiert, da Busweiten in Netzlisten in der Regel von 0 beginnend angegeben werden. Wenn also ein Bus beispielsweise mit $[31:0]$ deklariert wird, muss die korrespondierende Weite 32 sein. Durch das Attribut clock kann angegeben werden, ob es sich bei dem Port um ein Taktsignal handelt. Sollte dies der Fall sein, wird ein BUFG-Buffer für das Signal zusätzlich zu dem Input-Buffer IBUF eingefügt.

15.1.3 *Einlesen der Netzliste*

```
def create_buffer(input_file, clk: str):
    input_list = []
15  output_list = []
    with open('/user/abeer/SyntheseProjekt-Git/SyntheseProjekt/output/'+input_file+'.
    v','r') as netlist:
        fin = netlist.read()
```

Listing 43: Einlesen der Netzliste

Bei Aufruf des Skripts müssen der Funtkion create_buffer zwei Attribute übergeben werden. Der Name der Verilog-Netzliste muss über input_file und der Name des Taktsignals über den String clk (vgl. Zeile 13) übergeben werden. Zu Beginn werden zwei leere Listen erzeugt. Die Liste input_list dient der späteren Speicherung der Eingangsports und die Liste output_list dient der Speicherung der Ausgangsports (Zeile 14-15). Über die with open Anweisung wird daraufhin die Netzliste geöffnet und in die Variable netlist eingelesen. Über die Zuweisung auf die Variable fin in Zeile 18 wird die Netzliste als formatierter String in fin gespeichert. Die geöffnete Netzliste wird bei diesem Vorgang nur im Lese-Modus geöffnet, da eine Modifikation der Netzliste an dieser Stelle noch nicht erfolgt. Der Lesemodus wird über das Attribut `r` im open-Befehl gekennzeichnet.

15.1.4 *Generierung von Port-Objekten aus gefundenen Ports*

Abschnitt 44 dient dem Finden und Kategorisieren von Ports. Das korrekte Auffinden der Ports ist eine zentrale Funktionalität, da an der Korrektheit die weitere Generierung und Ersetzung von Signalen abhängt.

```
    # iterate through ports in file
20  # create object of class 'Port' from name and width
    for m in re.finditer('(?<=input )(?:\[(?P<width>\d*):\d*\])*(?P<port_name>[^;]*)
    ?(?=,|;)', fin, flags=re.MULTILINE):
        port_name_list = m.group('port_name').split(',')
        for k in port_name_list:
            input_list.append(Port(k.split(), 'input', m.group('width')))
25  for m in re.finditer('(?<=output )(?:\[(?P<width>\d*):\d*\])*(?P<port_name>[^;]*)
    ?(?=,|;)', fin, flags=re.MULTILINE):
        port_name_list = m.group('port_name').split(',')
        for k in port_name_list:
            output_list.append(Port(k.split(), 'output', m.group('width')))
```

Listing 44: Generierung von Port-Objekten aus gefundenen Ports

Über eine for-Schleife (Zeile 21) werden alle, durch die Methode finditer gefundenen Eingangsportnamen, der Liste port_name_list zugewiesen. Da die Netzliste mithilfe der regulären Ausdrücke zeilenweise verarbeitet wird, ist es nötig, die port-Namen mit dem Splitbefehl zu trennen. Andernfalls würde eine Zeile wie

```
input clk, rst, enable;
```

dazu führen, dass dem Port-Namen der Ausdruck »`clk, rst, enable`« zugewiesen werden würde. Die so erstellte Liste port_name_list wird daraufhin mithilfe einer for-Schleife genutzt, um die Liste input_list mit Port-Objekten zu füllen. Hierbei wird das Attribut direction auf input gesetzt. Auf die Weite der Ports kann über die Methode group zugegriffen werden. Diese Methode dient dazu, die gefundenen Ausdrücke

anhand von definierten Namen zuweisbar und aufrufbar zu machen.

Die Zeilen 25-29 erfüllen die gleiche Funktionalität analog dazu für Ausgangsports.

Zum besseren Verständnis der Zuweisungen mithilfe der group-Methode, soll im Folgenden auf den Regulären Ausdruck zur Auffindung der Eingangsports eingegangen werden.

15.1.5 *Reguläre Ausdrücke*

Der reguläre Ausdruck zum Finden der Eingangsports lautet:

```
(?<=input )(?:\[(?P<width>\d*):\d*\])*(?P<port_name>[^;]*)?(?=,|;)
```

Zum Verständnis kann er so unterteilt werden, dass der Sinn einzelner Abschnitte leichter ersichtlich wird.

```
(?<=input )
```

Über (?<=...) wird eine positive lookbehind-Gruppe definiert. Eine positive lookbehind-Gruppe legt fest, dass vor dem gesuchten Ausdruck der hier definierte Ausdruck erscheinen muss. In diesem Falle ist der Einsatz der Gruppe sinnvoll, da in Verilog Eingänge syntaktisch mit dem Ausdruck input gekennzeichnet werden.

```
(?:\[(?P<width>\d*):\d*\])*
```

Der Ausdruck (?:...) definiert eine match-Gruppe. Eine match-Gruppe kennzeichnet die Ausdrücke, nach denen innerhalb des gesamten regulären Ausdrucks gesucht wird. Ziel dieser Gruppe ist es, die Weite von Ports einzulesen. Da bei Ports mit der Weite 1 keine Weitenangabe gemacht wird, ist das Ende der Gruppe mit $*$ versehen. Der Ausdruck $*$ ist ein Quantifier und dient dazu, eine Aussage über die Häufigkeit des zu suchenden Ausdrucks zu machen. Im Speziellen dient der $*$ dazu, Ausdrücke zu kennzeichnen, die zwischen 0 und ∞ auftreten. Durch die Gruppe (?P<width>...) wird der gefundene Ausdruck mit dem Namen width versehen, was dazu führt, dass dieser im Skript durch Angabe der Gruppe ausgewertet werden kann.

```
(?P<port_name>[^;]*)?
```

Wie bereits im Falle der width-Gruppe, wird mithilfe der Gruppe port_name alles – ausgenommen das Semikolon – eingelesen. Der ?-Quantifier dient der Kennzeichnung, dass der Ausdruck ein- oder keinmal auftritt.

```
(?=,|;)
```

Mit (?=...) wird eine positive-lookahead-Gruppe definiert, das heisst der gesucht Ausdruck muss mit der hier aufgeführten Syntax enden. Innerhalb von Verilog-Netzlisten wird die Zeile mit einem Semikolon beendet und eine Unterteilung zwischen verschiedenen Ports, die in einer Zeile deklariert werden, wird mit einem Komma vorgenommen. Aus diesem Grund muss der gesuchte Ausdruck mit einem der beiden Ausdrücke enden. Der Mittelstrich | hat die Funktion eines logischen ODER.

15.1.6 *Kennzeichnung des Taktsignals*

```
    # find clock
    if clk:
        for input_port in input_list:
            if input_port.name == clk:
                input_port.clock = True
```

Listing 45: Kennzeichnung des Taktsignals

Zur Auffindung des Taktsignals muss dieses über das Attribut clock gekennzeichnet sein. Um dieses Attribut auf True zu setzen, wird mithilfe einer for-Schleife durch alle Eingangsports iteriert und der Name des Ports mit dem Namen des Taktsignals verglichen. Sollte dieser Vergleich positiv sein, wird das clock-Attribut für den betreffenden Port auf True gesetzt.

15.1.7 *Formatierung der eingelesenen Netzliste*

```
    # remove comments
    fin = re.sub('\/\/.*','',fin)
    fin = re.sub('\/\*.*','',fin)
    # remove line breaks
    fin = fin.replace('\n','')
    # split elements at semi colon
    fin = fin.split(';')
    for idx,i in enumerate(fin):
        if not i.startswith('endmodule'):
            i = re.sub('^\s{2,}','',i)+';'
            i = re.sub('\s{2,}',' ',i)
            i = re.sub('(?<=\S)\(',' (',i)
            fin[idx] = i
```

Listing 46: Formatierung der eingelesenen Netzliste

Zur weiteren Verarbeitung der Netzliste ist es nötig die Kommentare zu entfernen, da innerhalb der Kommentare Ausdrücke stehen können, die andernfalls über reguläre

Ausdrücke gefunden oder modifiziert werden können. Um dies zu realisieren, wird in Zeile 37-38 die sub (*substitute*)-Anweisung genutzt. Mithilfe derer nach Zeilen gesucht wird, die mit den kennzeichnenden Ausdrücken für Kommentare, das heisst // oder /∗, beginnen. Diese Zeilen werden daraufhin gelöscht. Das Löschen des Zeileninhalts nach Kommentarsyntax geschieht über die .-Anweisung, welche alle Zeichen, bis auf Zeilenumbrüche, beinhaltet. Die aktualisierte Netzliste wird in fin gespeichert.

Innerhalb der Netzliste können Zeilenumbrüche an verschiedenen Stellen stattfinden. Um eine fehlerhafte Verarbeitung solcher Zeilen auszuschließen, werden die Zeilenumbrüche durch die replace-Anweisung (Zeile 40) gelöscht.

Für eine Verarbeitung der Netzliste ist es hilfreich, diese mit for-Schleifen auf spezifische Ausdrücke durchsuchen zu können. Um das zu ermöglichen, wird die Netzliste, welche bisher als String in der Variable fin gespeichert war, in eine Liste aus Zeilen konvertiert. Die Konvertierung in eine Liste erfolgt hierbei an den Semikola (vgl. Zeile 42), da diese syntaktisch die Zuweisungen voneinander trennen. Innerhalb der for-Schleife in den Codezeilen 43-48 werden die Zeilen der Liste von überflüssigen Leerzeichen bereinigt und über die Anweisung fin[idx] = i gespeichert. Die for-Schleife nutzt hierbei die zwei Indizes idx und i, um über idx auf die Zeilennummer und über i auf den Inhalt der Zeile zuzugreifen.

15.1.8 *Ersetzung der Signalnamen mit Buffer-Signalen*

```
    # replace signal names in cells with _buf signals
    for idx, i in enumerate(fin):
        if not (i.startswith('input') and i.startswith('output')
                and i.startswith('module') and i.startswith('wire')):
            for element in input_list:
                if element.width > 1:
                    for k in range(element.width):
                        i = re.sub(f'(?<=\W){element.name}\[{k}\](?=>\W)?', f'{
element.name}_buf[{k}]', i)
                elif element.width <= 1:
                    i = re.sub(f'(?<=\W){element.name}(?=>\W)?', f'{element.name}_buf
', i)
            fin[idx] = i
            for element in output_list:
                if element.width > 1:
                    for k in range(element.width):
                        i = re.sub(f'(?<=\W){element.name}\[{k}\](?=>\W)?', f'{
element.name}_buf[{k}]', i)
                elif element.width <= 1:
                    i = re.sub(f'(?<=\W){element.name}(?=>\W)?', f'{element.name}_buf
', i)
```

```
            fin[idx] = i
```

Listing 47: Ersetzung der Signalnamen mit Buffer-Signalen

Damit die zu generierenden Buffer-Zellen korrekt genutzt werden, müssen sie mit der Schaltung verbunden werden. Die Verbindung der Schaltung wird über eine Umbenennung der bereits bestehenden Signale gelöst. Hierbei iteriert das Skript über eine for-Schleife durch die Zeilen der Netzliste, um im ersten Schritt Zeilen auszuschließen, die mit den Schlüsselwörtern input, output, module oder wire beginnen (vgl. Zeile 53). Für Zeilen die nicht mit diesen Schlüsselwörtern beginnen wird eine for-Schleife, welche durch die Elemente der input_list iteriert, durchlaufen. Ziel dieser for-Schleife ist es, die Elemente der input_list mit der aktuellen Zeile der Netzliste zu vergleichen. Sollte der Vergleich eine Übereinstimmung erzielen, wird der Name des Signals innerhalb der Netzliste durch das Affix _buf ergänzt, sodass dieses Signal mit dem korrespondierenden Buffer verbunden ist (vgl. Zeile 58). Zur Ersetzung der Signalnamen wird die sub-Methode des re-Moduls (6.1.5) genutzt. Zur Auffindung der relevanten Ausdrücke kommen reguläre Ausdrücke zum Einsatz, auf die im Folgenden genauer eingegangen wird.

Das Anhängen der Affixe an die vorhandenen Signalnamen muss über eine spezifische Syntax erfolgen, da die Namensgebung von Ports in Verilog beispielsweise schon Teilelemente von Signalnamen enthalten kann. So ist es nicht unüblich, dass der Enable-Eingang eines Flipflops mit »en« benannt ist, was sich ebenfalls in einem anderen Port mit dem Namen »enable« wiederfinden würde. Zentrale Funktionalität der Suche muss es daher sein, keine Teilstücke von Namen in den Signalen auszutauschen. Um dieses Ziel zu erreichen wird der reguläre Ausdruck zur Suche so aufgebaut, dass nur Namen erkannt werden, die mit nicht-Wort Symbolen enden und beginnen. Der Ausdruck dazu ist in Listing 48 aufgeführt.

```
(?<=\W){element.name}\[{k}\](?=>\W)?
```

Listing 48: Regulärer Ausdruck zum Ersetzen von Signalnamen

Das zu suchende Element ist element.name. Dabei handelt es sich um ein Objekt der Klasse Port, welches über das Attribut name verfügt. Über den Ausdruck k wird ein spezifischer Port eines Busses ausgewählt. Das zu suchende Element ist von zwei Gruppen umgeben. Die erste Gruppe (?<=\W) ist eine positive lookbehind-Gruppe. Das heisst ein Ausdruck wird nur ausgewertet, wenn er mit \W beginnt. Der Ausdruck \W dient hierbei der Auswahl aller nicht-Wort-Symbole. Die Gruppe (?=>\W) ist eine positive lookahead-Gruppe und führt dazu, dass Ausdrücke nur ausgewertet werden, wenn sie mit einem nicht-Wort-Symbol enden.

15.1.9 *Zeilennummer für Einfügen der wire finden*

```
70    # find line after wire
      for line_number, line in enumerate(fin):
          if not line.startswith('wire') and fin[line_number-1].startswith('wire'):
              line_after_wire = line_number
```

Listing 49: wire

Listing 49 dient der Auffindung der Zeile, in der die Wire platziert werden sollen. In der Regel folgen Netzlisten der festen Anordnung: Inputs, Outputs, wire. Das muss allerdings nicht der Fall sein, denn Yosys beispielsweise hat keine feste Struktur für die Anordnung der Elemente.

15.1.10 *Einfügen der Ausgangsbuffer*

```
75    # insert output buffer between wires and cell description
      for port in output_list:
          for j in range(port.width):
              if port.width > 1:
                  fin.insert(line_after_wire, f'CC_OBUF {port.name}_buf_inst{j} (.I ({
      port.name}_buf[{j}]), .O ({port.name}[{j}]));')
80            else:
                  fin.insert(line_after_wire, f'CC_OBUF {port.name}_buf_inst (.I ({port
      .name}_buf), .O ({port.name}));')
```

Listing 50: Einfügen der Ausgangsbuffer

Das Einfügen der Ausgangsbuffer geschieht über eine for-Schleife, welche durch alle Port-Objekte in output_list iteriert. Sollte der Port eine Bitweite größer als eins haben, werden dementsprechend viele Buffer erzeugt. Bei der Erzeugung der Buffer wird der erzeugten Instanz ein Name gegeben, der sich aus dem Portnamen, dem Affix _buf_inst und dem aktuellen Bit zusammensetzt. Jeder Ausgangsbuffer wird an seinem Ausgang mit dem Ausgangsport und an seinem Eingang mit dem Buffer-Signal verbunden (vgl. Zeile 78-79). Für Ports mit einer Bitweite von 1 wird analog dazu ein Port ohne numerische Kennzeichnung erzeugt und in gleicher Weise mit dem Buffer-Signal verbunden.

15.1.11 *Einfügen der Eingangs- und Clockbuffer*

```
    # insert input buffer
    for port in input_list:
        for j in range(port.width):
            if port.width > 1:
                fin.insert(line_after_wire, f'CC_IBUF {port.name}_buf_inst{j} (.I ({
port.name}[{j}]), .O ({port.name}_buf[{j}]));')
            else:
                if port.clock == True:
                    fin.insert(line_after_wire, f'CC_IBUF {port.name}_buf_inst{j} (.I
 ({port.name}), .O ({port.name}_bufg));')
                    fin.insert(line_after_wire, f'CC_BUFG {port.name}_bufg_inst{j} (.
I ({port.name}_bufg), .O ({port.name}_buf));')
                else:
                    fin.insert(line_after_wire, f'CC_IBUF {port.name}_buf_inst{j} (.I
 ({port.name}), .O ({port.name}_buf));')
```

Listing 51: Einfügen der Eingangs- und Clockbuffer

Die Generierung der Eingangsbuffer geschieht über eine for-Schleife, welche durch alle Ports der input_list iteriert. Für jedes Element wird durch eine if-Abfrage kontrolliert, ob es sich um einen Bus oder einen Port handelt. Für Ports wird der Buffer, wie in Kapitel 15.1.10 erläutert, eingefügt. Ein Unterschied zu der Generierung der Ausgangsbuffer besteht hierbei in der Möglichkeit, dass es sich bei dem Port um ein Taktsignal handeln kann. Für das Taktsignal ist es notwendig eine weitere Buffer-Zelle, die BUFG-Zelle einzufügen. Diese Zelle wird erzeugt, wenn der Name des Eingangsports mit dem Namen des Taktsignals übereinstimmt.

15.1.12 *Einfügen der wire-Signale*

```
    # print buffer wire
    for i in reversed(output_list):
        if i.width > 1:
            fin.insert(line_after_wire, f'wire [{i.width-1}:0] {i.name}_buf;')
        else:
            fin.insert(line_after_wire, f'wire {i.name}_buf;')
    for i in reversed(input_list):
        if i.width > 1:
            fin.insert(line_after_wire, f'wire [{i.width-1}:0] {i.name}_buf;')
        else:
            fin.insert(line_after_wire, f'wire {i.name}_buf;')
    fin.insert(line_after_wire, f'wire {clk}_bufg;')
```

Listing 52: Einfügen der wire-Signale

Zur Einfügung der wire-Signale werden die Listen, welche die Ports beinhalten über for-Schleifen iteriert. Die for-Schleifen werden dazu durch die reversed-Anweisung rückwärts durchlaufen, um die Anordnung der Buffer innerhalb der Netzliste auch in den wire-Deklarationen beizubehalten. Für jeden Port wird dabei eine Unterscheidung nach der Weite vorgenommen, sodass die wire-Signale die entsprechende Bitweite haben. Auf die Bitweite wird über das width-Attribut zugegriffen. Eine Nutzung des Attributs innerhalb der for-Schleife wird über einen f-String erreicht, der es erlaubt auf Variablen zuzugreifen. Nachdem die for-Schleifen für Ausgangs- (Zeile 96) und Eingangsbuffer (Zeile 101) durchlaufen wurden, wird noch ein wire für die Clockbuffer-Zelle erzeugt.

15.1.13 *Entfernung und Einfügung bestehender Eingangsports*

```
110     # find and remove input lines
        input_lines = [idx for idx,i in enumerate(fin) if fin[idx].startswith('input')]
        for index in input_lines:
            fin[index] = ''

115     # arrange inputs
        for i in reversed(input_list):
            if i.width > 1:
                fin.insert(input_lines[0],f'input [{i.width-1}:0] {i.name};')
            else:
120             fin.insert(input_lines[0],f'input {i.name};')
```

Listing 53: Entfernung und Einfügen bestehender Eingangsports

Um eine übersichtliche Netzliste zu generieren, werden die bestehenden Eingangsports zuerst entfernt und danach sortiert wieder in die Netzliste geschrieben. Im ersten Schritt müssen dafür alle Zeilen gefunden werden, in denen ein input vorkommt (Zeile 111). Sind diese Zeilen über die startswith-Methode gefunden, werden sie gelöscht. Daraufhin werden die Eingangsports mithilfe der zuvor geschaffenen Liste input_list in die Netzliste geschrieben.

15.1.14 *Entfernung und Einfügung bestehender Ausgangsports*

```
        # find and remove output lines
        output_lines = [idx for idx,i in enumerate(fin) if fin[idx].startswith('output')]
        for index in output_lines:
125         fin[index] = ''

        for i in reversed(output_list):
            if i.width > 1:
                fin.insert(output_lines[0],f'output [{i.width-1}:0] {i.name};')
```

```
130         else:
                fin.insert(output_lines[0],f'output {i.name};')
    #print('\n'.join(fin))
    fin[:] = [element for element in fin if element != '']
```

Listing 54: Entfernen der Ausgangsports

Analog zu dem zuvor genannten Prozess der Einfügung der Eingangsport, wird mithilfe der in Zeile 121-133 gezeigten Funktionalität das Gleiche für Ausgangsports vorgenommen.

15.1.15 *Schreiben der modifizierten Netzliste*

```
    now = datetime.datetime.now()
    fin.insert(0, f'// create_buffer: {now}')
    #print('\n'.join(fin))
    with open('/user/abeer/SyntheseProjekt-Git/SyntheseProjekt/output/buffered/'
145             +input_file+'_buffered.v','w') as w_file:
        w_file.write('\n'.join(fin))

    return input_file+'_buffered.v'
```

Listing 55: Schreiben der modifizierten Netzliste

Beim Schreiben der gebufferten Netzliste wird zur Kennzeichnung der Netzliste das aktuelle Datum und die aktuelle Zeit in die Netzliste geschrieben. Dafür werden diese Daten zunächst über datetime.now() erzeugt und in der Variable now gespeichert (vgl. Zeile 140). Daraufhin wird eine Kommentarzeile, in der die Variable now steht, über fin.insert in die Netzliste an oberster Stelle eingefügt. Über with open wird eine Datei im Schreibmodus erzeugt und der Inhalt von fin wird in diese Datei geschrieben. Die Netzliste mit Buffer wird in den Ordner ./output/buffered geschrieben. Die Kennzeichnung der Netzliste erfolgt über den Dateinamen, welcher durch das Affix _buffered ergänzt wird.

16 Klasse Reports

Das Reports-Modul dient als Hilfsmodul zur Auffindung und Kategorisierung relevanter Parameter aus den, durch die Tools erzeugten, Reports.

16.1 Übersicht

Zur Erstellung der Klasse Report sind die Parameter design und tool nötig. Anhand dieser Parameter kann die Methode read die korrespondierenden Reports auslesen und die so ausgelesenen Werte den Attributen der Klasse zuweisen. Die Attribute der Klasse sind:

- design
- tool
- logic_equivalence
- max_clk_freq
- gatecount
- cell_count
- total_area
- slack
- crit_path_start
- crit_path_end
- internal_power
- switching_power

16.1.1 *Module*

```
import subprocess
import re
import pdb
```

Listing 56: Module der Klasse Reports

Das Modul subprocess dient der Ausführung von awk. Mit re werden reguläre Ausdrücke zum Auffinden gesuchter Parameter innerhalb der Report-Dateien genutzt.

16.1.2 *Initialisierung der Klasse*

```
class Report:
    def __init__(self, design, tool):
        self.design = design
        self.tool = tool
        self.logic_equivalence = 'n/a'
        self.max_clk_freq = 'n/a'
        self.gatecount = 'n/a'
        self.cell_count = 'n/a'
        self.total_area = 'n/a'
        self.slack = 'n/a'
        self.crit_path_start = 'n/a'
        self.crit_path_end = 'n/a'
        self.internal_power = 'n/a'
        self.switching_power = 'n/a'
```

Listing 57: Initialisierung der Klasse

Bei Initialisierung einer Klasse wird die dunder (double underscore)-Methode init aufgerufen. Mithilfe dieser Methode werden die Attribute der Klasse zunächst auf einen festen default-Wert gesetzt. Grund dafür ist, dass nicht jedes Tool die Möglichkeit hat, alle Attribute zu simulieren. So hat beispielsweise LeonardoSpectrum keine Möglichkeiten zur Power-Simulation. Die einzigen Attribute, die der Klasse übergeben werden müssen, sind design und tool. Anhand dieser Attribute wird nach dem korrespondierenden Report gesucht.

16.1.3 *Lesen der Reports*

```
    def read(self, cc_pr_report=False):
        report_file = [(f'./SyntheseProjekt/reports/'
                        f'{self.design}_{self.tool}.rpt')]
```

Listing 58: Initialisierung der Klasse

Die read-Methode dient dem Lesen der Reports. Das Attribut cc_pr_report der Funktion kontrolliert, ob ein Report des pnr-Tools ausgelesen werden soll. Zu Beginn wird die Report-Datei anhand von Tool- und Designnamen definiert und der Variable report_file zugewiesen.

```
        if self.tool == 'genus':
            pattern = ["awk",
                "$3 ~ /[0-9]/ && $2 ~ /[0-9]/ && $NF ~ /(D)|(S)/ {print $2}"]
            self.cell_count = (subprocess.Popen(pattern+report_file,
                               stdout=subprocess.PIPE).stdout
```

```
                                        .read().decode('ascii')).strip()
```

Listing 59: Abfrage des Tools

Anhand einer if-Abfrage wird daraufhin eine Auswahl bezüglich der zu suchenden Parameter getroffen. Die if-Abfrage erfüllt hier die Funktionalität einer switch-case-Anweisung anhand derer man in anderen Sprachen die Auswahl nach dem Parameter tool treffen würde. Python verfügt in der genutzten Version (3.9.2.) nicht über eine solche Anweisung, sodass stattdessen die if-Abfrage genutzt wird. In Zeile 26 wird der Befehl definiert, anhand dessen die Suche nach dem relevanten Attribut implementiert ist. Die Variable pattern ist eine Liste aus Strings. Der erste Eintrag ist hierbei awk und der zweite Eintrag besteht aus einem Awk-Skript zur Auffindung und Ausgabe des gesuchten Parameters. In Zeile 27 wird das Attribut cell_count dem Wiedergabewert eines Subprozess-Aufrufs zugeordnet. Der Subprozess, der hierbei aufgerufen wird, setzt sich aus dem pattern und dem report_file zusammen. Auf diese Weise durchsucht das Awk-Skript die Datei nach dem genannten Ausdruck. Der Rückgabewert des Subprozesses ist über die print-Funktion gegeben. Damit die Print-Funktion den Wert an das Attribut self_count übergeben kann, wird der Subprozess Aufruf folgendermaßen umgesetzt: Über Subprocess.Popen wird ein Objekt der Klasse Popen erzeugt. Durch die Option stdout=subprocess.PIPE wird eine Pipe zur Auslesung des Subprozess-Outputs erzeugt, welche mit stdout.read() ausgelesen wird. Durch .decode('ascii') wird der ausgelesene Stream in das ASCII-Format dekodiert und mit .strip() werden überflüssige Leerzeichen von diesem String entfernt. Dieser Vorgang, das heisst die Zuweisung eines Patterns und der Aufruf eines Awk-Skripts, wiederholt sich für jedes Attribut, welches von dem Tool erzeugt wird. Gleichermaßen wiederholt sich der Vorgang auch für jedes Tool. Der Unterschied liegt diesbezüglich in der Änderung der verwendeten Report-Datei über die Variable report_file und die Nutzung anderer Ausdrücke zur Suche der Parameter.

16.1.4 *Auslesung des Reports der logischen Äquivalenzprüfung*

Eine Sonderstellung bei den Attributen nimmt das Ergebnis der logischen Äquivalenzprüfung ein, da hierfür neben dem eigentlichen Synthesetool noch das Tool für die Prüfung zum Einsatz kommt.

```
            report_file = [(f'./SyntheseProjekt/reports/'
65                          f'{self.design}_{self.tool}_lec.rpt')]
            pattern = ["awk", "/Compare Results: / {print $NF}"]
            self.logic_equivalence = (subprocess.Popen(pattern+report_file,
                                      stdout=subprocess.PIPE).stdout
                                      .read().decode('ascii'))
70          try:
                self.logic_equivalence = self.logic_equivalence.split()[0]
            except IndexError:
```

```
                print(f'IndexError in lec for tool {self.tool}')
```

Listing 60: Lesen des LEC-Reports

Für die Einlesung des LEC-Reports wird die Variable report_file auf den Pfad des erzeugten LEC-Reports gesetzt. Die Variable pattern dient der Zuweisung des awk-Aufrufs und des zu suchenden Ausdrucks. Daraufhin wird das Attribut logic_equivalence dem Rückgabewert des subprocess.Popen-Objekts zugewiesen. Anders als bei der Zuweisung des Attributs wird in Zeile 70-73 ein exception handling eingefügt. Grund dafür ist, dass das Skript keinen Error erzeugen und anhalten darf, wenn zwei Ergebnisse für den gesuchten Ausdruck gefunden werden. Dies passiert beispielsweise bei der Nutzung einer do-Datei für die Überprüfung einer mit Genus synthetisierten Netzliste. Über den Ausdruck try: wird versucht mehrfach gefundene Ergebnisse zu teilen. Sollte es nicht mehrere Ergebnisse geben wird ein IndexError ausgelöst, da versucht wird, eine Liste mit einem einzigen Element zu teilen. Für diesen Fall wird lediglich eine Nachricht ausgegeben (Zeile 73).

16.1.5 *Auslesung der Reports des pnr-Tools*

```
        if cc_pr_report == True:
            report_file = [f'./SyntheseProjekt/reports/pnr.rpt']
225         pattern = ["awk",
                       "/Maximum Clock Frequency on/ {print $(NF-1)}"]
            try:
                self.max_clk_freq = (subprocess.Popen(pattern+report_file,
                                     stdout=subprocess.PIPE).stdout
230                                  .read().decode('ascii'))
                try:
                    self.max_clk_freq = self.max_clk_freq.split()[0]
                except IndexError:
                    print('IndexError')
235         except FileNotFoundError:
                self.max_clk_freq = 'unknown'
            pattern = ["awk",
                       "/Gatecount:/ {print $2}"]
            try:
240             self.gatecount = (subprocess.Popen(pattern+report_file,
                                  stdout=subprocess.PIPE).stdout
                                  .read().decode('ascii')).strip()
            except FileNotFoundError:
                self.gatecount = 'unknown'
```

Listing 61: Auslesen der Reports des pnr-Tools

Falls das Attribut cc_pr_report den Wert True besitzt, wird ein Vorgang durchlaufen, der die beiden relevanten Parameter, d.h. Maximum Clock Frequency und Gatecount aus dem Report extrahieren soll. Hierfür wird die Variable report_file auf die pnr-

Report-Datei gesetzt und über pattern wird der Befehl awk sowie der Ausdruck zur Suche der Maximum Clock Frequency gespeichert. Das Attribut max_clk_frequency ist daraufhin der Rückgabewert des Subprozesses mit genannter pattern-Variable. Der Ausdruck Maximum Clock Frequency kann mehrfach in der Report-Datei vorkommen, daher wird durch den Error handling Mechanismus in Form des try und except-Ausdrucks versucht max_clk_freq zu teilen. Dieser Error handling Mechanismus wird auch auf die Auslesung der Maximum Clock Frequency und des Gatecounts angewendet. Grund dafür ist, dass das Nichtvorhandensein eines Reports einen Error ausgelösen kann und somit auch bereits aufgenommene Werte der Synthesetools verloren gehen würden.

16.1.6 *Ausgabe der eingelesenen Parameter*

```
    def show(self):
        table = PrettyTable(['Key', 'Value'])
        for key, value in self.__dict__.items():
            table.add_row([f'{key}', f'{value}'])
        table.header = False
        table.align = 'l'
        print(table)
```

Listing 62: Ausgabe der eingelesenen Parameter

Die Methode show dient der Ausgabe der eingelesenen Variablen. Um die Variablen in übersichtlicher Form auszugeben, wird das Modul prettytable genutzt. Die Zuweisung table = PrettyTable dient der Erzeugung eines Objekts der Klasse Prettytable. Daraufhin iteriert eine for-Schleife durch die Dictionary-Struktur self.__dict__.items(). Diese Struktur beinhaltet alle Attribute des Objekts Report. Die Attribute sowie die zugehörigen Werte werden mit den Variablen key und value in die Tabelle gesetzt. Die Zuweisungen table.header = False und table.align = l dienen der Formatierung der Tabelle und führen dazu, dass die Tabelle keinen header hat und alle Elemente der Tabelle linksbündig angeordnet sind. Der Befehl print(table) gibt die so erzeugte Tabelle in der Konsole aus.

16.1.7 *Standalone Aufruf des Moduls*

```
if __name__ == '__main__':
255     genus = Report('IMS_ALU', 'designcomp')
    genus.read()
    genus.show()
```

Listing 63: Standalone Aufruf des Moduls

Das Modul Reports kann auch ohne die Einbindung in das Optimisierungsskript aufgerufen werden. Dadurch ergibt sich der Vorteil, dass es einfacher ist, neue Funktionen zu debuggen, wenn nur ein kleiner Teilablauf aufgerufen wird. Die if-Abfrage if __name__ == '__main__' prüft, ob das Skript als Hauptmodul aufgerufen wird, oder ob es als Teil eines Skripts genutzt wird. Wenn es als Teil eines Skripts genutzt wird, ist der Name des Report-Moduls nicht '__main__', da in Python nur das aufgerufene Skript intern diesen Namen trägt.

17 Optimization Flow

Das Optimization Flow Skript dient der Konfiguration und Steuerung des Synthese- und Optimierungsablaufs.

17.1 Übersicht

Innerhalb des Skripts werden dazu zwei Klassen deklariert. Die Configuration-Klasse dient der Übergabe von relevanten Parametern an die Funktionen des Skripts, d.h. Pfade, welche über die Datei settings.ini eingelesen werden.
Die Library-Klasse dient der Auslesung und Modifikation der designierten Liberty-Bibliothek. Diese Funktionalitäten werden für den Optimierungsprozess genutzt. Die Konfiguration des Ablaufs sowie die Unterscheidung, ob optimiert oder nur synthetisiert werden soll, geschieht über Kommandozeilenattribute. Ein Überblick über den Ablauf des Skripts findet sich im Flussdiagramm 35.

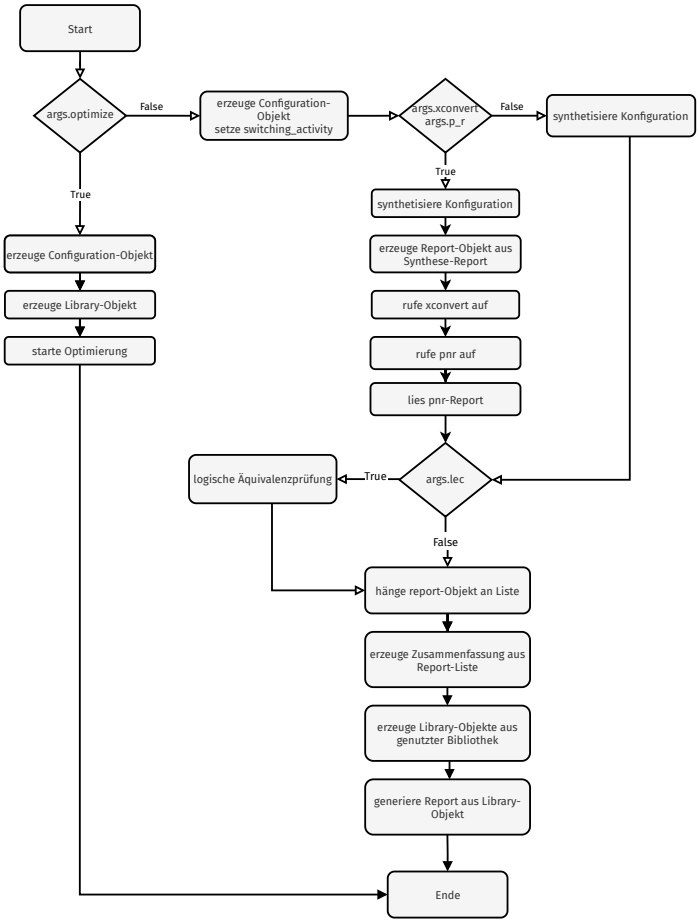

Abbildung 35: Ablauf des Skripts

17.1.1 *Kommdandozeilenargumente*

Um eine grobe Übersicht auf die Kommdandozeilenargumente des Skripts und deren Nutzung zu gewinnen, lässt sich die Hilfe über -h aufrufen. Die Hilfe ist in Listing 64 aufgeführt.

```
usage: optimization_flow.py [-h]
                            [-t {genus,designcomp,leonardo,vivado,yosys,
    yosys_gatemate}]
                            [-d DESIGN] [-lec] [-xc] [-pr] [-opt]
                            [-tar {total_area,cell_count,max_clk_freq,gatecount}]

* * * Toolflow for Optimization and Synthesis * * *

optional arguments:
  -h, --help            show this help message and exit
  -t {genus,designcomp,leonardo,vivado,yosys,yosys_gatemate}, --tool {genus,
    designcomp,leonardo,vivado,yosys,yosys_gatemate}
                        The Synthesis Tool to use
  -d DESIGN, --design DESIGN
                        See settings.ini for possible designs
  -lec, --lec           lec tool: conformal, formality or False
  -xc, --xconvert       XConvert
  -pr, --p_r            Place and Route
  -opt, --optimize
  -tar {total_area,cell_count,max_clk_freq,gatecount}, --target {total_area,
    cell_count,max_clk_freq,gatecount}
```

Listing 64: Optimization Flow Hilfe

Die Parameter können in einer Kurzform mit einem Bindestrich oder ausgeschrieben mit zwei Bindestrichen angegeben werden.

Tool: Der Parameter -t/–tool gibt das zu nutzende Tool an. Mögliche Werte sind genus, designcomp, leonardo, vivado, yosys und yosys_gatemate. Für den Syntheseablauf wird, im Gegensatz zur Optimierung, das Einbinden mehrerer Tools zugelassen. Für die Nutzung mehrerer Tools müssen diese erneut mit dem Parameter aufgeführt werden, das heißt für die Nutzung von Genus und Design-Compiler muss -t genus -t designcompiler eingegeben werden.

Design: Der Parameter -d/–design gibt das zu synthetisierende Desing an. Die möglichen Designs müssen in der setting.ini-Datei aufgeführt werden, um über den Parameter aufrufbar zu sein.

Logic Equivalence Check: Die logische Äquivalenzprüfung zwischen RTL-Code und synthetisierter Netzliste wird durch die Parameter -lec/–lec aktiviert. Für Genus wird fest das LEC-Tool Conformal verwendet, da die von Genus generierte do-Datei, welche Optimierungsschritte während des Synthesevorgangs abbildet,

direkt an das Tool weitergegeben werden kann. Für Design-Compiler erfolgt das in gleicher Weise mittels .svf-Datei für das LEC-Tool Formality. Die restlichen Synthese-Tools werden mittels Conformal geprüft.

XConvert: Der Parameter -xc/–xconvert aktiviert die Konvertierung der synthetisierten Netzliste mittels XConvert.

Place & Route Der Parameter -pr/–p_r aktiviert die Nutzung des Place & Route Tools, welches über wine gestartet wird. Für die Nutzung des Tools muss zuvor eine Konvertierung der Netzliste mittels XConvert erfolgt sein.

Optimization Mittels des Parameters -opt/–optimize wird der Optimierungsablauf aktiviert. Bei Nutzung des Optimierungsablaufs ist es nicht möglich mehrere Tools zu nutzen. Gleichzeitig muss bei Nutzung von -opt auch ein Optimierungsziel mit dem Parameter -tar/–target angegeben werden.

Target Das Ziel der Optimierung wird durch den Parameter -tar/–target bestimmt. Mögliche Werte sind total_area, cell_count, max_clk_freq und gatecount. Für die beiden ersten Werte wird der erweiterte Syntheseablauf mit XConvert und Place And Route nicht durchlaufen. Für Maximum Clock Frequency und Gatecount werden die Reports dieser Tools hingegen benötigt.

17.1.2 *Module*

Im folgenden Abschnitt sollen die für die Funktionalität des Skripts notwendigen Module kurz dargestellt werden.

```
import re
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import os
import os.path
import subprocess
from subprocess import Popen, PIPE
import shutil
import argparse
import configparser
import pdb
```

Listing 65: Module Optimization Flow

Für die Darstellung des Optimierungsergebnisses wird das Modul matplotlib.plyplot importiert. Das Kennwort »as« macht kenntlich, dass das Modul innerhalb des Skripts unter dem Namen plt verwendet wird. Mit den Modulen os und os.path kann auf Funktionen zugegriffen werden, die in der Regel vom Betriebssystem zur Verfügung gestellt werden. Hierzu gehören Funktionen zum Kopieren, Verschieben oder Löschen

von Dateien. Das Modul subprocess wird zusammen mit Popen und PIPE importiert, um die Synthesetools aus dem Skript heraus zu starten. Besonders die Nutzung der Klasse Popen sowie der Pipes ist erforderlich, da diese Module einen Datenstromzugang ermöglichen, um den Terminal-Output der Tools auszulesen und Eingaben zu tätigen. Die Möglichkeit Eingaben an die Tools zu senden, ist bei der Verwendung der XConvert und Place & Route Tools notwendig, da diese nach ihrer Programmausführung mit einer Eingabe quittiert werden müssen. Das Modul shutil dient der Nutzung von Shell-Befehlen. Argparse und Configparser dienen der Verarbeitung von Kommandozeilenargumenten und der Nutzung einer Konfigurationsdatei im ini-Format. Das Modul pdb stellt debugging-Funktionen zur Verfügung.

```
from skopt import gp_minimize, dump, load
from skopt.space import Real
from skopt.utils import use_named_args
from skopt.plots import plot_objective, plot_evaluations
```

Listing 66: Module Optimization Flow

Für die Bayes'sche Optimierung wird das Modul skopt genutzt. Für die Nutzung des Moduls wird zuerst gp_minimize importiert. Die weiteren Importe sind Hilfsfunktionen des Moduls, welche im Skript genutzt werden. Die Funktionen dump und load dienen der Speicherung und dem Laden von Optimierungszwischenständen. Die Klasse Real wird genutzt, um die Parameter der Optimierung zu initialisieren. Objekte der Klasse Real verfügen über einen Namen sowie Grenzen innerhalb der die Optimierung stattfindet. Die Methoden plot_objective und plot_evaluations dienen der Visualisierung des Optimierungsergebnisses.

```
from prettytable import PrettyTable
from reports import Report
import create_buffer as cb
```

Listing 67: Module Optimization Flow

PrettyTable dient der Ausgabe der Syntheseergebnisse in Tabellenform. Das Modul Reports ist eigens für das Skript erstellt und in Kapitel 16 beschrieben. Das Modul create_buffer ist ebenfalls im Rahmen des Projekts entstanden und wird in Kapitel 15 erläutert.

17.1.3 *Optimierungsdimensionen*

```
dim1 = Real(name='intrinsic_rise', low=0.0001, high=0.0015)
dim2 = Real(name='intrinsic_fall', low=0.0001, high=0.0015)
dim3 = Real(name='rise_resistance', low=0.0005, high=0.005)
dim4 = Real(name='fall_resistance', low=0.0005, high=0.005)
dim5 = Real(name='default_input_pin_cap', low=0.1, high=1.0)
dim6 = Real(name='wire_load_capacitance', low=0.01, high=1.0)
```

```
dimensions = [dim1, dim2, dim3, dim4, dim5, dim6]
```

Listing 68: Optimierungsdimension

In Zeile 23-28 werden die Parameter der Optimierung, auch Dimensionen genannt, deklariert und belegt. Bei den Dimensionen handelt es sich um Objekte der Klasse Real, welche Teil des skopt-Moduls ist. Den Dimensionen werden Namen zugeordnet, welche den Liberty-Attributen entsprechen, die sie darstellen. Die Attribute low und high dienen dazu eine Unter- und Obergrenze festzulegen, welche bei der Optimierung nicht über- bzw. unterschritten wird. Die Objekte werden dann dem Listenobjekt dimensions zugewiesen.

17.1.4 *Klasse Configuration*

Die Klasse Configuration dient der Übergabe von Pfaden und Bezeichnungen an Funktionen. Ihre Attribute und Methoden sind in Diagramm 36 dargestellt. Im

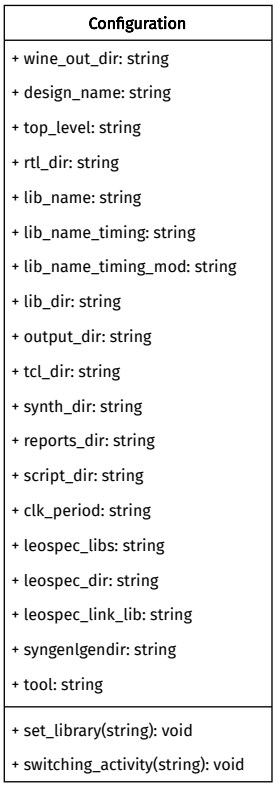

Abbildung 36: Klassendiagramm Configuration

folgenden Abschnitt sollen die Attribute und Methoden der Klasse erläutert werden.

wine_out_dir Der Pfad wine_out_dir weist auf den output-Ordner. In diesen Ordner

werden alle erzeugten Dateien mit Ausnahme der Reports geschrieben.

design_name Diesem Attribut wird der Name des Designs zugewiesen.

top_level Das Attribut top_level weist auf das Top Level des zu synthetisierenden Designs.

rtl_dir Der Pfad rtl_dir weist auf den Ordner, in dem sich das zu synthetisierende Design befindet.

lib_name Der Name der Liberty-Bibliothek ohne Timing- und Powerangaben.

lib_name_timing Der Name der Liberty-Bibliothek mit Timing- und Powerangaben.

lib_name_timing_mod Der Name der modifizierten Liberty-Bibliothek. Die modifizierte Bibliothek wird im Rahmen der Optimierungsdurchläufe erstellt und beinhaltet die, gemäß dem Optimierungsalgorithmus gewählten, veränderten Parameter (siehe 17.1.3).

lib_dir Der Pfad weist auf den Ordner, in dem sich die Liberty-Bibliothek befindet.

output_dir Der Pfad weist auf den Output-Ordner.

tcl_dir Der Pfad weist auf den Ordner, welcher die TCL-Skripte für die Tools enthält.

synth_dir Der Pfad weist auf den Output-Ordner.

reports_dir Der Pfad weist auf den Ordner, in den die Reports der Synthesetools geschrieben werden sollen.

script_dir Der Order, in dem die Skripte enthalten sind.

clk_period Die Periodendauer des Taktsignals.

leospec_libs Der Ordner, in dem die Bibliotheken für das Tool LeonardoSpectrum enthalten sind.

leospec_dir Der Ordner in dem das Tool LeonardoSpectrum ist.

leospec_link_lib Der Ordner, in dem der Link auf die Liberty-Bibliothek enthalten ist.

syngenlgendir Der Ordner, in dem die Tools Syngen und Lgen sind.

tool Der Name des Tools, mit welchem der aktuelle Synthese-/Optimierungsdurchlauf stattfindet.

set_library(string): void Die Methode set_library dient dazu, die aktuell zu nutzende Bibliothek zu ändern (vgl. Listing 69). Der Nutzen dieser Methode liegt darin, dass bei der Optimierung ein erster Durchlauf mit der angegebenen Liberty-Bibliothek stattfindet, ehe diese modifiziert und, unter anderem Namen, kopiert wird.

```
    def set_library(self, library_name):
        os.environ['LIB_NAME_TIMING'] = library_name
        os.environ['LIB_NAME'] = library_name
```

Listing 69: set_library-Methode

switching_activity(duty: float, clk_freq: float): void Die Methode switching_activity dient dazu, die switching_activity im TCL-Skript der Synthese-Tools zu modifizieren (Listing 70). Zum aktuellen Zeitpunkt wird diese Funktionalität nur für das Tool Genus unterstützt. Um Taktfrequenz und Periodendauer zu modifizieren, werden die dafür vorgesehenen Werte als Parameter an die Methode übergeben. Daraufhin wird das TCL-Skript genus.tcl im Lesemodus geöffnet und der Variable fin zugewiesen. Über die sub-Methode des re-Moduls wird dann nach den Ausdrücken für die Definition des Duty-Cycle sowie der Taktfrequenz gesucht. Gefundene Werte werden mit den übergebenen Parametern ersetzt. Zuletzt wird das Skript genus.tcl im Schreibmodus geöffnet und der Inhalt der Variable fin wird in das Skript geschrieben.

```
80      def switching_activity(self, duty, clk_freq):
            with open(f'{cfg.tcl_dir}/genus.tcl', 'r') as tool_script:
                fin = tool_script.read()
            fin = re.sub('(?<=-duty )(?P<duty>\d*\.*\d*)(?= -freq)', duty,
                                                fin, flags=re.MULTILINE)
85          fin = re.sub('(?<=-freq )(?P<freq>\d*e?\+?\d*)(?= -pin)', clk_freq,
                                                fin, flags=re.MULTILINE)
            with open(f'{cfg.tcl_dir}/genus.tcl', 'w+') as tool_script:
                tool_script.write(fin)
```

Listing 70: switching_activity-Methode

constraint(clock_period: float): void Die constraint-Methode dient der Konfiguration der Constraint-Datei. Die unterstützte Funktionalität ist nur die clock_period. Für jedes Tool muss die Constraint-Datei in einem anderen Format vorliegen (siehe Listing 71).

```
90      def constraint(self, clock_period):
            if self.tool == 'vivado':
                format = 'xdc'
            elif self.tool == 'leonardo':
                format = 'ctr'
95          else:
```

```
            format = 'sdc'
        with open(f'{self.script_dir}/constraint.{format}', 'w') as constraint:
            fin = []
            fin.insert(
                0, f'create_clock [get_ports {self.clock}] -name {self.clock}'
                   f' -period {clock_period}'
                .format(name=self.clock, period=self.clk_period))
            constraint.write(''.join(fin))
```

Listing 71: constraint-Methode

Zur Wahl des korrekten Formats wird das tool-Attribut der Instanz über eine if-Abfrage ausgelesen. Für vivado wird die Variable auf xdc gesetzt, für LeonardoSpectrum auf ctr und für die weiteren Tools auf sdc. Die Möglichkeit allein das Format zu ändern, ist gegeben, da sich die Formate in Bezug auf die genutzt Syntax nicht voneinander unterscheiden. Ist das Format gewählt, wird eine constraint-Datei im Schreibmodus erstellt und ein String mit dem genutzten Taktsignalnamen und der über clock_period übergebenen Periodendauer in die Datei geschrieben.

Die Attribute der Klasse werden, sofern es benötigt wird, als Umgebungsvariable gespeichert (siehe Listing 72).

```
        os.environ['WINE_OUT_DIR'] = self.wine_out_dir
        os.environ['DESIGN_NAME'] = self.design_name
        os.environ['TOP_LEVEL'] = self.top_level
        os.environ['RTL_DIR'] = self.rtl_dir
        #os.environ['CLOCK'] = self.clock
        os.environ['LIB_NAME'] = self.lib_name
        os.environ['LIB_NAME_TIMING'] = self.lib_name_timing
        os.environ['LIB_DIR'] = self.lib_dir
        os.environ['OUTPUT_DIR'] = self.output_dir
        os.environ['TCL_DIR'] = self.tcl_dir
        os.environ['SYNTH_DIR'] = self.synth_dir
        os.environ['REPORTS_DIR'] = self.reports_dir
        os.environ['SCRIPT_DIR'] = self.script_dir
        os.environ['LEOSPEC_LIBS'] = self.leospec_libs
        os.environ['LEOSPEC_DIR'] = self.leospec_dir
        os.environ['LEOSPEC_LINK_LIB'] = self.leospec_link_lib
        os.environ['SYNGENLGEN_DIR'] = self.syngenlgendir
        os.environ['TOOL'] = tool
```

Listing 72: Zuweisung der Klassenattribute als Umgebungsvariable

Diese Übergabe muss erfolgen, damit die Variablen auch innerhalb der aufgerufenen TCL-Skripte verfügbar sind.

17.1.5 *Klasse Library*

Die Klasse Library dient der Modifikation der Liberty-Bibliothek durch den Optimisierungsablauf. Ein Klassendiagramm ist in Abbildung 37 dargestellt.

```
class Library:
    def __init__(self, cfg: Configuration):
        self.library_file = f'{cfg.lib_dir}/{cfg.lib_name_timing}.lib'
        with open(self.library_file, 'r+') as ext_file:
            self.file_source = ext_file.read()
        self.read()
```

Listing 73: Klasse Library, Initialisierung

Bei Initialisierung eines Library-Objekts muss ein Configuration-Objekt als Parameter übergeben werden. Anhand dieses Objekts werden Pfad und Name der Bibliothek übergeben. Daraufhin wird das Attribut library_file als Pfad zur Liberty-Bibliothek initialisiert. In das Attribut file_source wird der Inhalt der Liberty-Bibliothek geschrieben.

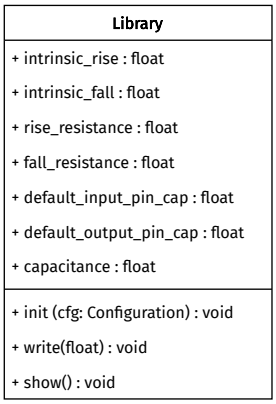

Abbildung 37: Klassendiagramm Library

```
    def read(self):
        pattern = re.compile(r'(?<=intrinsic_rise : )(\d*\.\d*)')
        self.intrinsic_rise = re.search(pattern, self.file_source).group(0)

        pattern = re.compile(r'(?<=intrinsic_fall : )(\d*\.\d*)')
        self.intrinsic_fall = re.search(pattern, self.file_source).group(0)

        pattern = re.compile(r'(?<=rise_resistance : )(\d*\.\d*)')
        self.rise_resistance = re.search(pattern, self.file_source).group(0)

        pattern = re.compile(r'(?<=fall_resistance : )(\d*\.\d*)')
        self.fall_resistance = re.search(pattern, self.file_source).group(0)
```

```
125
        pattern = re.compile(r'(?<=default_input_pin_cap : )(\d*\.\d*)')
        self.default_input_pin_cap = re.search(pattern,
                                    self.file_source).group(0)

130     pattern = re.compile(r'(?<=default_output_pin_cap : )(\d*\.\d*)')
        self.default_output_pin_cap = re.search(pattern,
                                    self.file_source).group(0)

        pattern = re.compile(r'(?<=capacitance : )(\d*\.\d*)')
135     self.wire_load_capacitance = re.search(pattern,
                                    self.file_source).group(0)
```

Listing 74: Klasse Library, read-Methode

Listing 74 zeigt die read-Methode. Die read-Methode dient dazu, die für die Optimierung relevanten Parameter aus der Liberty-Bibliothek auszulesen, sodass sie als Attribut der Klasse zu Verfügung stehen. Innerhalb der Bibliothek sind die Timing-Parameter – bis auf Werte, die das Timing 0 haben sollen – im float-Format gesetzt. Der Grund dafür ist, dass bei der Optimierung keine Timing-Parameter ausgelesen werden sollen, die fest auf 0 gesetzt sind. Um diese Parameter von der Optimierung auszuschließen, wird innerhalb der Liberty-Bibliothek nur nach Parametern im float-Format gesucht. Die Suche nach Parametern im float-Format geschieht über reguläre Ausdrücke, die speziell nach dem Format `\d*\.\d*` suchen. Beim Auffinden der genannten Parameter wird der Zahlenwert dem korrespondierenden Attribut der Klasse zugewiesen.

17.1.6 *write-Methode*

```
    def write(self, intrinsic_rise='0.00135',
              intrinsic_fall='0.00142',
140           rise_resistance='0.00421',
              fall_resistance='0.00428',
              default_input_pin_cap='1.0',
              wire_load_capacitance='1.0'):
        self.file_source = re.sub(fr'(?<=intrinsic_rise : )(\d*\.\d*)',
145                                   str(intrinsic_rise),
                                      self.file_source)
        self.file_source = re.sub(fr'(?<=intrinsic_fall : )(\d*\.\d*)',
                                      str(intrinsic_fall),
                                      self.file_source)
150     self.file_source = re.sub(fr'(?<=rise_resistance : )(\d*\.\d*)',
                                      str(rise_resistance),
                                      self.file_source)
        self.file_source = re.sub(fr'(?<=fall_resistance : )(\d*\.\d*)',
                                      str(fall_resistance),
155                                   self.file_source)
        self.file_source = re.sub(fr'(?<=default_input_pin_cap : )(\d*\.\d*)',
                                      str(default_input_pin_cap),
                                      self.file_source)
        self.file_source = re.sub(fr'(?<=capacitance : )(\d*\.\d*)',
160                                   str(wire_load_capacitance),
                                      self.file_source, count=1)

        with open(f'{cfg.lib_dir}/{cfg.lib_name_timing_mod}.lib', 'w') as lib:
            lib.write(self.file_source)
```

Listing 75: Klasse Library, write-Methode

Die write-Methode bildet funktional das Gegenstück zur read-Methode. Die write-Methode akzeptiert als Parameter Zahlenwerte, die den Attributen innerhalb der Liberty-Bibliothek entsprechen, die während der Optimierung geändert werden sollen. Um die Möglichkeit zu haben, nur einzelne Werte in der Bibliothek zu ändern, ohne die Funktionalität zu gefährden, werden zu Beginn der Funktion default-Werte gesetzt (vgl. Zeile 138-144). Bei der Ersetzung der Werte, werden die Parameter zuvor in das string-Format umgewandelt (vgl. Zeile 145, Listing 75). Der Parameter capacitance wird mit der Option count=1 angegeben, da hier nur der capacitance-Wert in der Wire Load-Gruppe geschrieben werden soll, welcher in der library-Gruppe der Bibliothek und somit am Anfang der Liberty-Datei steht. Alle darauf folgenden capacitance-Attribute werden somit nicht verändert. Am Ende der Funktion werden die modifizierten Werte in eine Liberty-Bibliothek geschrieben, dessen Name innerhalb der Konfiguration unter der Variable `lib_name_timing_mod` gespeichert ist. Dazu wird die Datei im Schreibmodus geöffnet und der Inhalt von file_source wird in diese Datei geschrieben.

17.1.7 *show-Methode*

```
    def show(self):
        table = PrettyTable(['Key', 'Value'])
        for key, value in self.__dict__.items():
            if key != 'file_source':
                table.add_row([f'{key}', f'{value}'])
        table.header = False
        table.align = 'l'
        print(table)
```

Listing 76: Klasse Library, show-Methode

Die show-Methode dient dem Aufführen aller Attribute der Klasse Library. Die Attribute werden in übersichtlicher Form mithilfe der PrettyTable-Klasse ausgegeben. Die Zuweisung table = PrettyTable dient der Erzeugung eines Objekts der Klasse Prettytable. Daraufhin iteriert eine for-Schleife durch die Dictionary-Struktur `self.__dict__.items()`. Diese Struktur beinhaltet alle Attribute des Objekts Report. Die Attribute sowie die zugehörigen Werte werden mit den Variablen key und value in die Tabelle gesetzt. Die Zuweisungen table.header = False und table.align = l dienen der Formatierung der Tabelle und führen dazu, dass die Tabelle keinen header hat und alle Elemente der Tabelle linksbündig angeordnet sind. Der Befehl print(table) gibt die so erzeugt Tabelle in der Konsole aus.

17.1.8 *Synthese*

Die Synthese des RTL-Codes wird über die synth-Funktion gesteuert. Die Funktion nimmt als Parameter ein Configuration-Objekt und gibt ein Report-Objekt zurück.

```
def synth(cfg: Configuration) -> Report:
    try:
        os.remove(f'{cfg.reports_dir}/{cfg.design_name}_{cfg.tool}.rpt')
        os.remove(f'{cfg.reports_dir}/{cfg.design_name}_{cfg.tool}_lec.rpt')
    except FileNotFoundError:
        print('No previous report file found')

    try:
        cfg.constraint(clock_period='20')
    except AttributeError:
        print('no clock in design')
```

Listing 77: synth-Funktion

Zunächst wird versucht bereits bestehende Tool-Reports zu löschen (vgl. Zeile 178-179). Grund dafür ist, dass bei einem Fehler während der Synthese kein neuer Report

erzeugt wird und somit der alte ausgewertet wird. Dieses Verhalten kann dazu führen, dass Probleme während des Synthesevorgangs unbemerkt bleiben. Die remove-Anweisungen stehen innerhalb eines Error-handlers, da die Möglichkeit besteht, dass kein Report vorhanden ist und somit ein FileNotFound-Error ausgelöst wird, welcher das Skript nicht beenden darf. Daraufhin wird versucht, die Periodendauer des Taktsignals auf 20ns zu setzen, falls innerhalb des Designs ein Taktsignal vorhanden ist.

```
    if cfg.tool == 'genus':
        subprocess.run(['module load Cadence/2019_2020;'
                        f"genus -lic_startup_options Joules_RTL_Power\
                        -log {cfg.reports_dir}/GenusLog\
                        -overwrite\
                        -f {cfg.tcl_dir}/genus.tcl"], shell=True)
    elif cfg.tool == 'yosys_gatemate':
        subprocess.run(['module load Yosys/0.9_gatemate;'
                        f'yosys -c {cfg.tcl_dir}/yosys_gatemate.tcl'], shell=True)
    elif cfg.tool == 'designcomp':
        subprocess.run(['module load Synopsys/2019_2020;'
                        f'dc_shell -f {cfg.tcl_dir}/designcomp.tcl'], shell=True)
    elif cfg.tool == 'yosys':
        conv_bib_yosys(cfg)
        subprocess.run(['module load Yosys/0.9;'
                        f'yosys -c {cfg.tcl_dir}/yosys.tcl'], shell=True)
    elif cfg.tool == 'vivado':
        subprocess.run(['module load Xilinx/Vivado/2020.1;'
                        'vivado -mode batch -source $SCRIPT_DIR/vivado.tcl'], shell=
    True)
    elif cfg.tool == 'leonardo':
        create_syn(cfg)
        subprocess.run(['module load Mentor/LEONARDO/2017a;'
                        'module load Mentor/PRECISION/2019.1.1;'
                        f'spectrum -file {cfg.tcl_dir}/leonardo.tcl'], shell=True)
    else:
        print('no tool selected')
    report = Report(f'{cfg.design_name}', f'{cfg.tool}')
    #report.read()
    return report
```

Listing 78: Wahl des Synthese-Tools

Das zu nutzende Synthese-Tool wird innerhalb des Configuration-Objekts an die synth-Funktion übergeben. Mittels einer if-Abfrage wird daraufhin der korrespondierende Subprozess gestartet. Die hierfür zu nutenden TCL-Skripte sind in dem Ordner `Syntheseprojekt/scripts/tcl` hinterlegt. Für die Tools Yosys und LeonardoSpectrum müssen noch Umwandlungen der Netzliste vorgenommen werden, bevor die Tools gestartet werden können, da Yosys eine Bibliothek ohne Timing-/Power-Angaben nutzt und LeonardoSpectrum die Bibliothek ausschließlich im .syn-Format nutzen kann.

Die nötigen Konvertierungen der Liberty-Bibliothek sind in Abbildung 38 dargestellt. Nachdem die Synthese beendet ist, wird ein Report-Objekt anhand des Designs

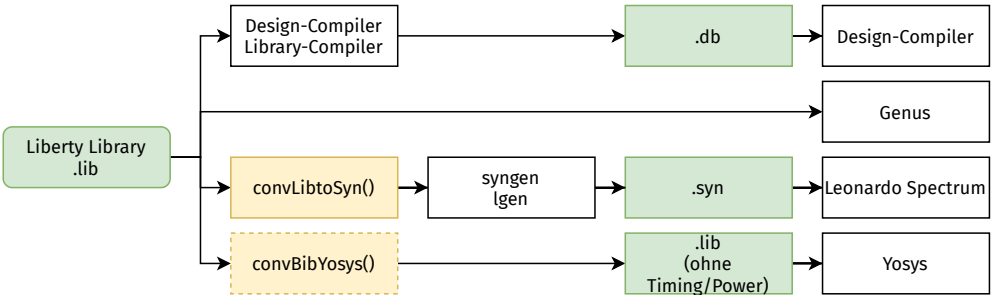

Abbildung 38: Konvertierung der Liberty-Bibliothek

und des Tools erzeugt. Dieses Objekt wird am Ende der Funktion als Wiedergabewert zurückgegeben.

17.1.9 *Aufruf der CC-Tools*

Der Aufruf der CC-Tools, das heisst XConvert und pnr muss nach der Synthese erfolgen, da die Tools eine Konvertierung der synthetisierten Netzliste vornehmen. Des Weiteren müssen alle Netzlisten, bis auf die mit Yosys GateMate synthetisierten, mit Buffern versehen werden, da es sonst zu einem Error beim pnr-Tool kommt.

```
def cc_tools(cfg: Configuration, xconvert=False, p_r=False):
    if xconvert == True:
        if cfg.tool != 'yosys_gatemate':
            buffered_design = cb.create_buffer(f'{cfg.design_name}_{cfg.tool}',
                                               clk=cfg.clock)
            #pdb.set_trace()
            path = f'buffered/{buffered_design}'
        else:
            pdb.set_trace()
            path = f'{cfg.design_name}_{cfg.tool}.v'
        p = subprocess.Popen([f'wine {cfg.cc_tools}/x_convert.exe \
                        -i:SyntheseProjekt/output/{path} \
                        -o:{cfg.cc_out_dir}/{cfg.design_name}_{cfg.tool} \
                        +net --lib:ccag;'], stdin=PIPE, shell=True, stdout=PIPE)
        p.communicate(input=b'\n')

    if p_r == True:
        with open(f'{cfg.reports_dir}/pnr.rpt', 'w') as fileout:
            p = subprocess.Popen([f'wine {cfg.cc_tools}/p_r.exe \
                            -i:{cfg.cc_out_dir}/'
                            f'{cfg.design_name}_{cfg.tool} +crf +uCIO; exit'],
                             stdout=fileout, stdin=PIPE, shell=True)
```

```
            p.communicate(input=b'\n')
```

Listing 79: Aufruf der Cologne Chip Tools

Nach der Erstellung des Buffers mittels create_buffer-Skript (Kapitel 15) wird die Variable Pfad auf die so modifizierte Netzliste gesetzt. Für den Start des Subprozesses XConvert wird ein Popen-Objekt, mit den Optionen stdin=PIPE, stdout=PIPE und shell=True initialisiert. Die Optionen stdin und stdout definieren, wo Prozesseinund -ausgänge hingeleitet werden. Mit dem Kennwort PIPE wird definiert, dass eine Pipe zum Popen-Objekt erstellt wird. Über die communicate-Methode (vgl. Zeile 233) wird ein Input an den Prozess gesendet. Durch den b-String wird ein binär kodierter String gesendet, welcher äquivalent zu »\n« ist. Dies ist nötig, damit das Tool nach Ausführung der Konvertierung beendet wird. Gleichermaßen wird der Prozess und die Prozesskommunikation bei dem pnr-Tool gelöst (Zeile 235-241).

17.1.10 *Konvertierung der Bibliothek*

```
def conv_bib_yosys(cfg: Configuration):
245     subprocess.run([f'{cfg.script_dir}/filterTimingPower.sh \
                        {cfg.lib_dir}/{cfg.lib_name_timing}.lib \
                        > {cfg.lib_dir}/{cfg.lib_name}.lib'], shell=True)
```

Listing 80: Konvertierung der Bibliothek

Die Funktion conv_bib_yosys (Listing 80) ist ein Wrapper für das filterTimingPower-Shellskript. Die Funktion nimmt als Parameter ein Configuration-Objekt und konfiguriert anhand dieses Objekts das filterTimingPower-Skript. Ziel des Skripts ist die Entfernung von Timing- und Power-Angaben innerhalb der Liberty-Bibliothek. Das so modifizierte Skript wird im library-Ordner unter dem für lib_name definierten Namen gespeichert.

17.1.11 *Konvertierung der Bibliothek in das .syn-Format*

```
250 def create_syn(cfg: Configuration):
    subprocess.run(['module load Mentor/LEONARDO/2017a;'
                    'module load Mentor/PRECISION/2019.1.1;'
                    f'tclsh {cfg.tcl_dir}/mentor_lib_conv.tcl'], shell=True)
    shutil.copy(f'{cfg.lib_dir}/{cfg.lib_name_timing}.syn', cfg.leospec_libs)
255 #if [ -f $LEOSPEC_LINK_LIB/fh_do_dev_lib.syn ]
    #then
    #rm $LEOSPEC_LINK_LIB/fh_do_dev_lib.syn
    #fi
    subprocess.run([f'ln -fs $LEOSPEC_LIBS/{cfg.lib_name_timing}.syn \
260                 $LEOSPEC_LINK_LIB/{cfg.lib_name_timing}.syn'], shell=True)
```

Listing 81: Konvertierung der Bibliothek in das .syn-Format

Die Funktion create_syn (81) dient dazu, die Liberty-Bibliothek in das .syn-Format zu konvertieren. Damit die Konvertierung fehlerfrei funktioniert, müssen die Module LEONARDO und PRECISION zuvor geladen werden. Daraufhin wird über tclsh das mentor_lib_conv-Skript aufgerufen. Die dadurch erstellte .syn-Datei wird dann in den library-Ordner kopiert und ein symbolischer Link wird durch den Befehl `ln -fs` im LeonardoSpectrum Link Library-Ordner erstellt.

17.1.12 *Logische Äquivalenzprüfung*

```
def logic_equivalence_check(cfg: Configuration, lec_tool: str):
    if cfg.tool == 'designcomp':
        subprocess.run(['module load Synopsys/2019_2020;'
                        f'fm_shell -f {cfg.tcl_dir}/formality.tcl'], shell=True)
    else:
        if os.path.isfile(f'{cfg.reports_dir}/{cfg.design_name}_{cfg.tool}.lec.do'):
            subprocess.run(['module load Cadence/2019_2020;'
                            'lec -lpgxl -nogui -dofile $REPORTS_DIR/${DESIGN_NAME}_${
    TOOL}.lec.do \
                            > $REPORTS_DIR/${DESIGN_NAME}_${TOOL}_lec.rpt'],
                            shell=True)
        else:
            subprocess.run(['module load Cadence/2019_2020;'
                            f'lec -lpgxl -TclMode {cfg.tcl_dir}/conformal.tcl'],
    shell=True)
```

Listing 82: Logische Äquivalenzprüfung

Die Funktion logic_equivalence_check dient der Ausführung der logischen Äquivalenzprüfung. Um auszuwählen, welches LEC-Tool verwendet werden soll, wird das zu nutzende Tool über das Attribut lec_tool an die Funktion übergeben. Die durch Genus und Design-Compiler synthetisierten Netzlisten nehmen eine Sonderstellung bei der logischen Äquivalenzprüfung ein, da die Tools eine Datei generieren, welche die Optimierungen aufführt, die durch das Tool an der synthetisierten Netzliste vorgenommen wurden. Bei Genus ist das die .do-Datei und bei Design-Compiler ist das die .svf-Datei. Sollte die zu überprüfende Netzliste durch ein anderes Tool als den angeführten synthetisiert worden sein, wird das Tool Conformal aufgerufen und eine Prüfung unter Einbezug der Liberty-Bibliothek vollzogen. Für Design-Compiler Netzlisten wird das Tool Formality genutzt. Conformal und Formality werden mittels TCL-Skripten konfiguriert.

17.1.13 *Lesen des pnr-Reports*

```
def read_p_r(cfg: Configuration) -> Report:
    #with open(f'{cfg.reports_dir}/pnr.rpt', 'r') as pnr_report:
```

```
    #    fin = pnr_report.read()
    report = Report(f'{cfg.design_name}', f'{cfg.tool}')
    report.read(cc_pr_report=True)
    return report
```

Listing 83: Lesen des pnr-Reports

Die Funktion read_p_r dient dazu, ein report-Objekt zu erzeugen, welches die, durch die CC-Tools erzeugten, Attribute enthält. Dies ist sinnvoll, da die CC-Tools nach der Synthese aufgerufen werden, sodass der Report der Synthese-Tools schon vorliegt und die Attribute gegebenenfalls schon einem report-Objekt zugewiesen wurden. Der Rückgabewert der Funktion ist das report-Objekt.

17.1.14 *Objective-Funktion*

```
@use_named_args(dimensions=dimensions)
def objective(intrinsic_rise, intrinsic_fall,
              rise_resistance, fall_resistance,
              default_input_pin_cap, wire_load_capacitance):
    liberty.write(intrinsic_rise, intrinsic_fall,
                  rise_resistance, fall_resistance,
                  default_input_pin_cap, wire_load_capacitance)
    # pdb.set_trace()
    cfg.lib_name_timing = f'{cfg.lib_name_timing_mod}'
    cfg.set_library(f'{cfg.lib_name_timing_mod}')
    res = synth(cfg)
    if args.target == 'max_clk_freq' or 'gatecount':
        print(f'target: {args.target}, executing cc_tools')
        cc_tools(cfg, True, True)
        res =  read_p_r(cfg)
    return np.double(getattr(res, args.target))
```

Listing 84: Objective-Funktion

Die objective-Funktion ist die Funktion, welche durch den Algorithmus optimiert wird. Der use_named_args-Wrapper umschließt die Funktion. Seine Nutzung ermöglicht es eine Liste von benannten Parametern an die Funktion zu übergeben. Die objective-Funktion akzeptiert als Parameter die Attribute der Liberty-Bibliothek, die zuvor als Dimensionen der Optimierung definiert wurden. Über die write-Methode des liberty-Objekts werden die übergebenen Parameter in die Liberty-Bibliothek geschrieben und die zu nutzende Bibliothek wird durch set_library auf die so erzeugte Bibliothek gesetzt. In Zeile 304 wird der Synthesevorgang ausgeführt und das erzeugte Objekt der Report-Klasse wird der Variable res zugewiesen. Daraufhin wird eine Entscheidung anhand des Optimierungsziels getroffen. Soll Maximum Clock Frequency oder Gatecount optimiert werden, müssen die CC-Tools aufgerufen werden und die durch sie erzeugten Werte müssen dem Report-Objekt res zugewiesen werden. Dies

geschieht mithilfe einer if-Abfrage, welche diese Entscheidung anhand des target-Arguments trifft. Der Wiedergabewert der objective-Funktion muss ein double-Objekt des Numpy-Moduls sein. Welches Attribut des res-Objekts als return-Wert dient, wird anhand der getattr-Methode definiert. Durch diese Methode wird nur das Attribut von res als Rückgabewert wiedergeben, welches durch das target-Argument definiert wurde.

17.1.15 *Optimierungsalgorithmus*

Die Funktion optimize dient dem Aufruf der gp_minimize-Funktion, welche die Auswertung der Ergebnisse und die Festlegung der Samplepunkte kontrolliert.

```
def optimize():
    min_result = gp_minimize(objective,
                             dimensions=dimensions,
315                          acq_func="gp_hedge",
                             n_calls=n_calls,
                             n_random_starts=n_random_starts,
                             noise=1e-10)
    return min_result
```

Listing 85: Optimierungsalgorithmus

Der gp_minimize-Aufruf erfolgt mit folgenden Parametern:

Parameter:

func Ist die zu minimierende Funktion. Die Funktion muss eine Liste von Parametern nehmen und einen np.double-Wert zurückgeben. Falls der Wrapper use_named_args genutzt werden, kann die Funktion mit den benannten Parametern aufgerufen werden.

dimensions Die Dimensionen, anhand derer der Suchraum der Optimierung definiert ist.

acq_func Die Acquisition-Funktion, bei der über den Parameter gp_hedge zwischen den Optionen Lower Confidence Bound, Negative Expected Improvement und Probability of Improvement bei jeder Iteration gewählt werden kann.

n_calls Die Anzahl der Aufrufe von func.

n_random_starts Die Anzahl der zufälligen Samplepunkte, die vor der Schätzung durch base_estimator evaluiert werden.

noise Wird genutzt, um rauschbehaftete Observationen zu simulieren. Für den Fall, dass kein Rauschen vorliegt wird empfohlen, einen sehr kleinen Wert zu nehmen, da ein noise-Wert von 0 zu Stabilitätsproblemen führen kann.

Bei abgeschlossener Optimierung wird ein Rückgabewert wiedergegeben mit folgenden Attributen:

Rückgabebewert:

min_result Das Ergebnis der Optimierung ist ein OptimizeResult-Objekt. Die Attribute des Objekts sind:

x: das Minimum.

fun: der Funktionswert am Minimum.

models: Surrogate-Modell, das für die Iterationen genutzt wurde.

x_iters: Werte der Dimensionen für die evaluierten Punkte.

func_vals: Funktionswert für jede Iteration.

space: der Optimierungsraum.

specs: die Spezifikationen der Aufrufe.

17.1.16 *Zusammenfassung der Reports*

```
def summarize(report_list):
    categories = ['design', 'cell_count', 'total_area',
                  'internal_power', 'switching_power', 'crit_path_start',
325               'crit_path_end', 'slack', 'logic_equivalence',
                  'Max Clock Frequency (pnr)', 'Gatecount (pnr)']
    summary = PrettyTable()
    column_names = ['']
    for i in args.tool:
330     column_names.append(i)
    summary.add_column(column_names[0],categories)
    for idx, obj in enumerate(report_list):
        column_list = [obj.design, obj.cell_count, obj.total_area,
                    obj.internal_power, obj.switching_power,
335                 obj.crit_path_start, obj.crit_path_end,
                    obj.slack, obj.logic_equivalence, obj.max_clk_freq,
                    obj.gatecount]
        summary.add_column(column_names[1+idx],column=column_list)
    print(summary)
```

Listing 86: Zusammenfassung der Reports

Die summarize-Funktion wird am Ende des Synthesedurchlaufs aufgerufen und dient dazu, die Attribute der Report-Objekte in visuell übersichtlicher Form zu präsentieren. Als Parameter wird eine Liste von Report-Objekten akzeptiert. Die Liste categories beinhaltet alle Kategorien, die in der Zusammenfassung gezeigt werden sollen. Die Zusammenfassung geschieht über ein PrettyTable-Objekt, welches in Zeile 327 initialisiert wird. Über eine for-Schleife werden die Attribute jedes Objektes der Liste

in der Liste column_list gespeichert. Diese Listen-Objekte werden dann über die add_column-Methode zu der Tabelle summary hinzugefügt. Die vollständige Tabelle wird daraufhin über `print(summary)` in der Konsole ausgegeben.

17.2 Ablaufsteuerung

Im Folgenden wird, auf den vorangegangenen Erläuterungen zur Funktionsweise der Funktionen und Klassen aufbauend, die Steuerung des Ablaufs beschrieben. Der bei Start des Skripts aufgerufene Programmteil ist durch die if-Abfrage nach dem Skriptnamen `if __name__ == '__main__'` gekennzeichnet.

17.2.1 *Definition und Einlesen der Argumente*

```
    description = '* * * Toolflow for Optimization and Synthesis * * *'
    parser = argparse.ArgumentParser(description=description)
345 parser.add_argument('-t', '--tool', action='append',
                        choices=['genus', 'designcomp', 'leonardo', 'vivado',
                                 'yosys', 'yosys_gatemate'],
                        help='The Synthesis Tool to use')
    parser.add_argument('-d', '--design',
350                     help='See settings.ini for possible designs')
    parser.add_argument('-lec', '--lec', action='store_true',
                        help='lec tool: conformal, formality or False')
    parser.add_argument('-xc', '--xconvert', action='store_true',
                        help='XConvert')
355 parser.add_argument('-pr', '--p_r', action='store_true',
                        help='Place and Route')
    parser.add_argument('-opt', '--optimize', action='store_true')
    parser.add_argument('-tar', '--target', default=False,
                        choices=['total_area', 'cell_count', 'max_clk_freq',
360                              'gatecount'])
    args = parser.parse_args()
```

Listing 87: Definition und Einlesen der Argumente

Listing 88 zeigt das Hinzufügen und Parsen der Argumente, welche in der Kommandozeile eingegeben werden. Zunächst wird der Variable description ein String zugewiesen, welcher bei der Ausgabe der Hilfe als Header dient. Daraufhin wird ein ArgumentParser-Objekt erstellt (Zeile 344). Das ArgumentParser-Objekt verfügt über die Methode add_argument, mit der sich die Kommandozeilenargumente hinzufügen lassen. Bei der Hinzufügung der Argumente stehen Optionen zur Verfügung, die das Verhalten der Parameter beeinflussen. Die genutzten Optionen werden im Folgenden beschrieben:

action Das action-Attribut spezifiziert, wie das mit ihm assoziierte Kommandozeilenargument zu behandeln ist. Die genutzten Argumente lauten wie folgt:

append Mit append wird eine Liste erzeugt und jedes Argument mit diesem Parameter wird in dieser Liste gespeichert. Innerhalb des Skripts wird dieser Parameter für die Speicherung einer Liste genutzt, in der die zu

nutzenden Tools gespeichert sind. In einem Synthesedurchlauf können somit mehrere Tools für ein Design genutzt werden.

store_true Der Parameter store_true wird genutzt, um einen boolschen Wert zu speichern. Wenn Kommandozeilenargumente mit diesem Attribut initialisiert werden, können sie alleinstehend genutzt werden.

choices Mit choices kann eine Auswahl von Parametern definiert werden, die vom Kommandozeilenargument akzeptiert wird.

help Mit help wird definiert, welcher Text beim Aufruf der Hilfe für das korrespondierende Kommandozeilenargument angezeigt wird.

Das Kommdandozeilenargument wird in einer Kurzfassung und einer langen Fassung angegeben. Die Kurzfassung wird mit einem Bindestrich gekennzeichnet, die lange Fassung mit zwei. Die Funktionen der Kommandozeilenargumente wird in Kapitel 17.1.1 erläutert.

17.2.2 *Definition und Einlesen der Konfigurationsdatei*

Um Konfigurationen festzuhalten, wird die Datei settings.ini eingelesen.

```
config = configparser.ConfigParser()
config.read('SyntheseProjekt/scripts/settings.ini')
```

Listing 88: Definition und Einlesen der Argumente

Die Einlesung der Datei geschieht über Funktionalitäten, die das Modul configparser zur Verfügung stellt. Zur Einlesung einer Datei wird zunächst ein ConfigParser-Objekt erzeugt. Mit der read-Methode dieses Objekts wird die settings.ini-Datei eingelesen.

settings.ini Eine INI-Datei ist eine Konfigurationsdatei, die aus einem textbasierten Inhalt mit einer Struktur und Syntax besteht. Die Datei beinhaltet Schlüssel-Wert-Paare für Eigenschaften und Abschnitte zur Organisation dieser Eigenschaften. Der Name dieser Konfigurationsdateien stammt von der Dateinamenerweiterung INI für Initialisierung, die im MS-DOS-Betriebssystem verwendet wurde, welches diese Methode der Softwarekonfiguration populär gemacht hat.

Innerhalb der settings.ini finden sich neben Angaben zu den genutzten Pfaden der Configuration-Klasse auch Angaben zu den Testdesigns.

17.2.3 *Steuerung der Synthese*

```
    start = datetime.datetime.now()
    ''' Loop Synthesis for every tool '''
    if args.optimize == False:
        report_list = []
        for idx, tool in enumerate(args.tool):
            #print(idx, args.tool[idx])
            cfg = Configuration(args.tool[idx], args.design)
            cfg.switching_activity('0.5', '50e+6')
            if args.p_r == False and args.xconvert == False:
                report = synth(cfg)
            else:
                report = synth(cfg)
                cc_tools(cfg, args.xconvert, args.p_r)
                report = read_p_r(cfg)
            logic_equivalence_check(cfg, args.lec)
            report.read()
            report_list.append(report)
        summarize(report_list)
        liberty = Library(cfg)
        liberty.show()
```

Listing 89: Steuerung der Synthesedurchgänge

Der Synthesedurchlauf wird gestartet, wenn das optimize-Argument False ist. Zunächst wird eine leere Liste für die Report-Objekte erstellt. Daraufhin iteriert eine for-Schleife durch alle Einträge in der Liste args.tool. Die Variable args.tool beinhaltet die in der Kommandozeile angegebenen Tools. Ein Configuration-Objekt mit Namen und Design des aktuellen Synthesedurchlaufs wird erzeugt und das Attribut switching_activity wird mit den Parametern duty=0.5 und frequency=50e+6 gesetzt. Über eine if-Abfrage wird ermittelt, ob die CC-Tools genutzt werden sollen. Sind beide Argumente p_r und xconvert False, werden die CC-Tools nicht genutzt und es wird ein Report-Objekt aus einem Synthesedurchlauf erzeugt. Andernfalls werden nach dem Synthesedurchlauf die CC-Tools genutzt und das Report-Objekt wird um die Attribute dieser Tools ergänzt.

Die Durchführung der logischen Äquivalenzprüfung wird mit dem Argument -lec bestimmt. Sollte das Argument True sein, wird auf logische Äquivalenz geprüft. Andernfalls hat das Argument den default-Wert false und es wird kein LEC durchgeführt. Zuletzt wird das Report-Objekt durch die Methode read aktualisiert. Dieser Schritt ist nötig, um das Ergebnis der logischen Äquivalenzprüfung miteinzubeziehen.

Das erzeugte Report-Objekt wird durch die append-Methode der report_list hinzugefügt. Ist die for-Schleife, welche durch alle gegebenen Tools iteriert, durchgelaufen,

wird eine Zusammenfassung der gesammelten Report-Objekte erstellt. Die Zusammenfassung wird über summarize erstellt und gibt die wichtigsten Attribute des Syntheseresultats aus. Durch die Erzeugung eines Library-Objekts und Aufruf der show-Methode wird zudem noch die genutzte Liberty-Bibliothek kurz dargestellt.

17.2.4 *Optimierung*

```
    elif args.optimize == True:
        call_count=0
        n_calls=5
        n_random_starts=2
        ''' Optimization Setup '''
        cfg = Configuration(args.tool[0], args.design)
        liberty = Library(cfg)
        opt_result = optimize()
        dump(opt_result, f'{cfg.output_dir}/opt_result_noobj_dump.pk1',
                         store_objective=False)
        with open(f'{cfg.reports_dir}/optimization_result.txt', 'w') as fopt:
            fopt.write(str(opt_result))

        _ = plot_evaluations(opt_result)
        plt.savefig('opt_result_visual_eval.pdf')
        _ = plot_objective(opt_result)
        plt.savefig('opt_result_visual_obj.pdf')
        print(f'start: {start} \nend: {datetime.datetime.now()}\nduration: {(datetime
    .datetime.now()-start)}')
```

Listing 90: Optimierung

Sollte das Argument Optimize den Wert True haben, wird der Optimierungsalgorithmus durchlaufen. Zunächst wird dafür die Aufrufanzahl der Objective-Funktion mit n_calls und die zufälligen Aufrufe mit n_random_starts definiert.

Zur Konfiguration der Optimierung wird ein Configuration-Objekt erzeugt, welches mit dem Namen des Tools und dem Design initialisiert wird. Zur Modifikation der Werte innerhalb der Liberty-Bibliothek wird ein Library-Objekt erstellt und mit dem Configuration-Objekt initialisiert. Über die Zuweisung `opt_result = optimize()` wird der Optimierungslauf gestartet und gleichzeitig wird festgelegt, dass das Ergebnis der Optimierung, d.h. das OptimizationResult-Objekt in opt_result gespeichert wird. Der dump-Befehl dient der Speicherung des Zwischenstands der Optimierung und kann dafür genutzt werden, eine Optimierung weiterzuführen. Ist der Optimierungslauf beendet, wird das Ergebnis in eine Textdatei namens opt_result.txt geschrieben. Die Visualisierung des Optimierungsergebnisses geschieht über die Funktionen plot_evaluations und plot_objective. Diese Plots werden über die Methode savefig im pdf-Format gespeichert. Zuletzt wird eine Nachricht ausgegeben, anhand der die Dauer der Optimierung abzulesen ist.

18 Auswertung

Im folgenden Kapitel sollen zunächst die Ergebnisse des Synthesedurchlaufs vorgestellt werden und im Anschluss daran die Ergebnisse der Optimierung. Die Ergebnisse der Synthese liegen in Form einer Tabelle vor, welche am Ende des Skripts ausgegeben wird. Die Optimierung erzeugt eine Visualisierung der Optimierungsergebnisse und eine Textdatei, in welcher die Parameter jeder einzelnen Iteration vermerkt sind.

18.1 Synthese

18.1.1 *Genus*

```
+--------------------------+----------------------+
|                          |        genus         |
+--------------------------+----------------------+
|           design         |       IMS_ALU        |
|         cell_count       |         1301         |
|         total_area       |       668.000        |
|       internal_power     |     5.81471e+02      |
|      switching_power     |     3.73441e+04      |
|      crit_path_start     | opcode_ff_reg[0]/CLK |
|       crit_path_end      |   res_ff_reg[31]/D   |
|           slack          |        19.245        |
|     logic_equivalence    |         PASS         |
| Max Clock Frequency (pnr) |        44.25         |
|      Gatecount (pnr)     |         779          |
+--------------------------+----------------------+
```

Listing 91: Synthese ALU Genus

In Tabelle 91 ist das Ergebnis der Synthese des ALU-Designs dargestellt. Es ist zu erkennen, dass die Synthese einschließlich der logischen Äquivalenzprüfung mittels Conformal durchgeführt wurde. Die Gatterzahl beträgt 1301 und die genutzte Fläche wird mit 668 Flächeneinheiten angegeben. Die Internal Power hat einen Wert von 581uW und die Switching Power beläuft sich auf 37,3mW. Die Timing-Analyse ergibt, dass ein kritischer Pfad gefunden wurde, welcher am Taktausgang des opcode_ff_reg-Registers startet und auf dem Datenausgang des res_ff_reg-Registers endet. Es ist an diesem Pfad keine Timing Violation aufgetreten, da der Slack-Wert 19,245ns beträgt. Die logische Äquivalenzprüfung, welche mittels do-File konfiguriert wurde, wird bestanden. Das CC-Tool pnr gibt eine maximale Taktfrequenz von 44,25MHz an und eine Gatterzahl von 779. Hierbei fällt auf, dass die Gatterzahl um 522 Gatter reduziert wird zwischen der ursprünglichen Genus Netzliste und der, durch das pnr-Tool optimierten, Netzliste.

```
+--------------------------+-----------------------------------------------------+
|                          |                        genus                        |
+--------------------------+-----------------------------------------------------+
|           design         |                    Canakari_Tripl                   |
|         cell_count       |                         6662                        |
|         total_area       |                       3085.000                      |
|       internal_power     |                     9.67395e+02                     |
|      switching_power     |                     4.85215e+04                     |
|      crit_path_start     | LogicalLinkControl_llc_fsm_2_CURRENT_STATE_reg[1]/CLK |
|       crit_path_end      |       MediumAccessControl_receivecrc_edged_reg/D    |
|           slack          |                         9.449                       |
|     logic_equivalence    |                          PASS                       |
| Max Clock Frequency (pnr) |                        30.45                       |
|      Gatecount (pnr)     |                         4044                        |
+--------------------------+-----------------------------------------------------+
```

Listing 92: Synthese Canakari Genus

Tabelle 92 zeigt die Parameter des mit Genus synthetisierten CAN-Controllers. Die Gatterzahl beläuft sich auf 6662 Gatter, welche eine Fläche von 3085 in Anspruch nehmen. Die Leistungsaufnahme beträgt 967uW für die Internal Power und 48,5mW für die Switching Power.
Das Timing der synthetisierten Schaltung weist keine Violation auf, da der kritische Pfad mit Startpunkt am Taktausgang des LogicalLinkControl_llc_fsm_2_CURRENT_STATE-Registers und Endpunkt am Dateneingang des MediumAccessControl_receivecrc_edged-Registers einen Slack von 9,449ns aufweist.
Die logische Äquivalenzprüfung wird bestanden. Das CC-Tool pnr gibt eine maximale Taktfrequenz von 30,45MHz an und eine Gatterzahl von 4044. Wie schon bei der ALU findet eine Reduzierung der Gatterzahl durch das pnr-Tool statt. In diesem Fall beträgt die Reduktion 2618 Gatter.

18.1.2 *Design-Compiler*

```
+--------------------------+------------------+
|                          |    designcomp    |
+--------------------------+------------------+
|           design         |     IMS_ALU      |
|         cell_count       |       2024       |
|         total_area       |    824.500000    |
|       internal_power     |   282.5837 uW    |
|      switching_power     |    13.4000 mW    |
|      crit_path_start     | opcode_ff_reg[0] |
|       crit_path_end      |  res_ff_reg[31]  |
|           slack          |      19.09       |
|     logic_equivalence    |    SUCCEEDED     |
| Max Clock Frequency (pnr) |      14.91       |
|      Gatecount (pnr)     |       1185       |
+--------------------------+------------------+
```

Listing 93: Synthese ALU Design-Compiler

In Tabelle 93 sind die Ergebnisse der Synthese der ALU mittels Design-Compiler aufgeführt. Die Gatterzahl beträgt 2024, was eine Fläche von 824 darstellt.
Die Internal Power beträgt 282,58uW und die Switching Power beträgt 13,4mW.
Das Timing der Synthese weist keine Violation auf, da der Slack zwischen opcode_ff-Register und res_ff_-Register 19,09ns beträgt. Die logische Äquivalenzprüfung wird durch Formality ausgeführt. Die Konfiguration von Formality wird hierbei mithilfe einer svf-Datei durchgeführt, welche durch Design-Compiler bei der Synthese erzeugt wurde. Das pnr-Tool gibt die maximale Taktfrequenz mit 14,91MHz und die Gatterzahl mit 1185 an. Wie schon bei der Synthese mit Genus findet hier bei eine Reduktion der Gatterzahl, durch Optimierungen des Tools statt.

```
+--------------------------+-------------------------------------------------+
|                          |                    designcomp                   |
+--------------------------+-------------------------------------------------+
|           design         |                  Canakari_Tripl                 |
|         cell_count       |                       6882                      |
|         total_area       |                    2947.500000                  |
|       internal_power     |                    933.3367 uW                  |
|      switching_power     |                     48.0354 mW                  |
|      crit_path_start     | LogicalLinkControl/llc_fsm_2/CURRENT_STATE_reg[0] |
|       crit_path_end      |     MediumAccessControl/transhift/enable_i_reg  |
|           slack          |                        9.48                     |
|     logic_equivalence    |                    SUCCEEDED                    |
| Max Clock Frequency (pnr) |                                                 |
|      Gatecount (pnr)     |                                                 |
+--------------------------+-------------------------------------------------+
```

Listing 94: Synthese Canakari Design-Compiler

Tabelle 94 zeigt die Ergebnisse der Synthese des CAN-Controllers mit Design-Compiler. Die Gatterzahl beträgt 6882 und die Gatterfläche beträgt 2947. Die Leistungsaufnahme liegt für die Internal Power bei 933,33uW und für die Switching Power bei 48,03mW. Bei der Timing-Analyse tritt keine Violation auf, da der Slack zwischen LogicalLinkControl/llc_fsm_2/CURRENT_STATE_reg[0] und MediumAccessControl/transhift/enable_i_reg bei 9,48ns liegt. Das pnr-Tool konnte die Netzliste nicht verarbeiten, daher sind keine Werte für maximale Taktfrequenz und Gatterzahl vorhanden. Die Fehlermeldung ist in Listing 95 zu sehen. Der Grund für den Error konnte nicht gefunden werden.

```
    FATAL-ERROR 19 Write protected
```

Listing 95: Fehlermeldung des pnr-Tools bei Verarbeitung der LeonardoSpectrum Canakari Netzliste

```
********************************** Matching Results ***********************
 705 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 27 Matched primary inputs, black-box outputs
 24(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
 5(0) Unmatched reference(implementation) unread points
-------------------------------------------------------------------------------
Unmatched Objects                                                        REF
    IMPL
-------------------------------------------------------------------------------
 Registers                                                                24
      0
```

Listing 96: Logische Äquivalenzprüfung des Canakari durch Formality

Die Ausgabe der logischen Äquivalenzprüfung durch Formality ist in Listing 96 dargestellt. Der Bericht zeigt, dass die Äquivalenzprüfung fehlschlägt, da 24 Punkte des Referenz-Designs nicht dem Implementation-Design zugeordnet werden können.

18.1.3 *LeonardoSpectrum*

LeonardoSpectrum verfügt nicht über die Möglichkeit der Power-Simulation, sodass diese Werte nicht vom Tool ausgegeben werden können.

```
+--------------------------+--------------------+
|                          |      leonardo      |
+--------------------------+--------------------+
|           design         |      IMS_ALU       |
|         cell_count       |        4743        |
|         total_area       |        793         |
|       internal_power     |        n/a         |
|      switching_power     |        n/a         |
|      crit_path_start     | reg_opcode_ff(0)/Q |
|       crit_path_end      |  reg_res_ff(31)/D  |
|           slack          |        0.62        |
|     logic_equivalence    |        PASS        |
| Max Clock Frequency (pnr) |       13.41        |
|      Gatecount (pnr)     |        939         |
+--------------------------+--------------------+
```

Listing 97: Synthese ALU LeonardoSpectrum

Die Ergebnisse der Synthese sind in Tabelle 97 für die ALU dargestellt. Die Gatterzahl beläuft sich auf 4743, was einer Fläche von 793 entspricht. Die hohe Gatterzahl in Verbindung mit der relativ geringen Fläche kommt dadurch zustande, dass LeonardoSpectrum 3242 Inverter nutzt, welche eine Fläche von 0 haben. Bei der Timing-Analyse ist keine Violation aufgetreten, da der Slack 0,62ns beträgt. Der kritische Pfad wurde zwischen dem Ausgang des reg_opcode_ff(0)-Registers und dem Dateneingang des reg_res_ff(31)-Registers berechnet.
Die logische Äquivalenzprüfung erfolgte mit Conformal und wurde bestanden. Das pnr-Tool gibt die maximale Taktfrequenz mit 13,41MHz und die Gatterzahl mit 939 an.

```
+--------------------------+-----------------------------------------------+
|                          |                    leonardo                   |
+--------------------------+-----------------------------------------------+
|           design         |                 Canakari_Tripl                |
|         cell_count       |                      21286                    |
|         total_area       |                      3529                     |
|       internal_power     |                      n/a                      |
|      switching_power     |                      n/a                      |
|      crit_path_start     |       reg_MediumAccessControl_count(6)/Q      |
|       crit_path_end      | MediumAccessControl_fsm_reg_current_state(2)/D |
|           slack          |                      0.25                     |
|     logic_equivalence    |                   FAIL:NONEQ                  |
| Max Clock Frequency (pnr) |                                               |
|      Gatecount (pnr)     |                                               |
+--------------------------+-----------------------------------------------+
```

Listing 98: Synthese Canakari LeonardoSpectrum

Die in Tabelle 98 dargestellten Ergebnisse der Synthese des CAN-Controllers zeigen eine Gatterzahl von 21286 mit einer Fläche von 3529. Wie schon bei der Synthese der ALU fällt die hohe Gatterzahl auf. Grund dafür ist, dass LeonardoSpectrum bei der Synthese des CAN-Controllers 14720 Inverter nutzt. Das Timing der Synthese weist keine Violation auf. Der Slack beträgt 0,25ns auf dem kritischen Pfad zwischen Ausgang des reg_MediumAccessControl_count(6)-Registers und Dateneingang des MediumAccessControl_fsm_reg_current_state(2)-Registers. Die logische Äquivalenzprüfung mittels Conformal schlägt bei dieser Schaltung fehl, da durch Conformal hier 681 nicht äquivalente Punkte gefunden werden (vgl. Listing 99).

```
================================================================================
Compared points      PO     DFF       Total
--------------------------------------------------------------------------------
Equivalent           30     0         30
--------------------------------------------------------------------------------
Non-equivalent       7      674       681
================================================================================
```

Listing 99: LEC für Canakari bei Synthese mit LeonardoSpectrum

18.1.4 *Yosys*

Yosys nutzt keine Timing- und Leistungsangaben der Liberty-Bibliothek zur Synthese, sodass sich die Reports auf Gatterzahl und Fläche beschränken.

```
+--------------------------+---------+
|                          |  yosys  |
+--------------------------+---------+
|           design         | IMS_ALU |
|         cell_count       |   1857  |
|         total_area       |   883   |
|       internal_power     |   n/a   |
|      switching_power     |   n/a   |
|      crit_path_start     |   n/a   |
|       crit_path_end      |   n/a   |
|           slack          |   n/a   |
|     logic_equivalence    |   PASS  |
| Max Clock Frequency (pnr) |  18.67  |
|      Gatecount (pnr)     |   979   |
+--------------------------+---------+
```

Listing 100: Synthese ALU Yosys

Bei der Synthese der ALU mit Yosys ist zu sehen, dass die Gatterzahl 1857 Gatter beträgt und sich die Fläche daraus zu 883 berechnet. Die logische Äquivalenzprüfung wird mit Conformal durchgeführt und es wird logische Äquivalenz festgestellt.
Das pnr-Tool gibt die maximale Taktfrequenz mit 18,67MHz und die Gatterzahl mit 979 an.

```
+--------------------------+----------------+
|                          |     yosys      |
+--------------------------+----------------+
|           design         | Canakari_Tripl |
|         cell_count       |      7291      |
|         total_area       |     3422.5     |
|       internal_power     |      n/a       |
|      switching_power     |      n/a       |
|      crit_path_start     |      n/a       |
|       crit_path_end      |      n/a       |
|           slack          |      n/a       |
|     logic_equivalence    |   FAIL:NONEQ   |
| Max Clock Frequency (pnr) |     24.79      |
|      Gatecount (pnr)     |      4923      |
+--------------------------+----------------+
```

Listing 101: Synthese Canakari Yosys

Das Ergebnis der Synthese des CAN-Controllers mit Yosys ist in Tabelle 101 dargestellt. Die Gatterzahl beträgt 7291 und die korrespondierende Fläche 3422,5. Der Canakari fällt bei der Synthese mit Yosys durch die logische Äquivalenzprüfung, da 268 Punkte

als nicht äquivalent ausgewertet werden. Die Zusammenfassung ist in Listing 102 dargestellt. Das pnr-Tool beziffert die maximale Taktfrequenz mit 24,79Mhz und die Anzahl der Gatter mit 4923.

```
============================================================================
Compared points      PO     DFF       Total
----------------------------------------------------------------------------
Equivalent           15     0         15
----------------------------------------------------------------------------
Non-equivalent       22     246       268
============================================================================
```

Listing 102: LEC für Canakari bei Synthese mit Yosys

18.1.5 *Yosys_Gatemate*

Die Yosys Gatemate-Erweiterung gibt von den relevanten Parametern nur die Gatterzahl aus.

```
+--------------------------+----------------+
|                          | yosys_gatemate |
+--------------------------+----------------+
|           design         |    IMS_ALU     |
|         cell_count       |      603       |
|         total_area       |      n/a       |
|       internal_power     |      n/a       |
|      switching_power     |      n/a       |
|      crit_path_start     |      n/a       |
|       crit_path_end      |      n/a       |
|           slack          |      n/a       |
|     logic_equivalence    |      PASS      |
| Max Clock Frequency (pnr) |                |
|      Gatecount (pnr)     |                |
+--------------------------+----------------+
```

Listing 103: Synthese ALU Yosys Gatemate

Aus Listing 103 geht hervor, dass die Gatterzahl bei Synthese der ALU 603 beträgt. Die logische Äquivalenzprüfung wird bestanden. Die Verarbeitung der Netzliste mit pnr-Tool schlägt Fehl. Die Fehlermeldung ist in Listing 104 zu sehen.

```
    FATAL ERROR: Output Buffer drive strength _STRENGTH 2'h0 for signal result[0]
    component 570 is not valid
    halt_procedure called with exit code: 10
```

Listing 104: Fehlermeldung des pnr-Tools bei Verarbeitung der Yosys_Gatemate ALU Netzliste

```
+--------------------------+----------------+
|                          | yosys_gatemate |
+--------------------------+----------------+
|           design         | Canakari_Tripl |
|         cell_count       |      2881      |
|         total_area       |      n/a       |
|       internal_power     |      n/a       |
|      switching_power     |      n/a       |
|      crit_path_start     |      n/a       |
|       crit_path_end      |      n/a       |
|           slack          |      n/a       |
|     logic_equivalence    |   FAIL:NONEQ   |
| Max Clock Frequency (pnr) |                |
|      Gatecount (pnr)     |                |
+--------------------------+----------------+
```

Listing 105: Synthese Canakari Yosys Gatemate

Bei Synthese des CAN-Controllers nutzt Yosys_Gatemate 2881 Gatter. Die logische Äquivalenzprüfung wird nicht bestanden, da Conformal 595 Punkte als nicht äquivalent auswertet (vgl. Listing 107). Das pnr-Tool kann die Netzliste nicht verarbeiten und bricht den Vorgang mit der in Listing 106 aufgeführten Fehlermeldung ab.

```
    FATAL ERROR! Outpin 2 at Component 2886, netline[6] not found!
    halt_procedure called with exit code: 0
```

Listing 106: Fehlermeldung des pnr-Tools bei Verarbeitung der Yosys_Gatemate Canakari Netzliste

```
================================================================================
Compared points      PO     DFF       Total
--------------------------------------------------------------------------------
Equivalent           17     10        27
--------------------------------------------------------------------------------
Non-equivalent       20     575       595
================================================================================
```

Listing 107: LEC für Canakari bei Synthese mit Yosys Gatemate

18.2 Vergleich der Ergebnisse

Im Folgenden soll auf die Unterschiede der Syntheseergebnisse eingegangen werden.

18.2.1 *ALU*

Tool	Genus	DC	LeonardoSpectrum	Yosys	Yosys_GM
cell_count	1301	2024	4743	1857	603
total_area	668	824,5	793	883	
Gatecount (pnr)	779	1185	939	979	

Tabelle 32: Anzahl der Gatter und Fläche für ALU

Tabelle 32 zeigt die Anzahl der Gatter sowie die Fläche für die Synthese der ALU bei den genutzten Tools. Es ist zu erkennen, dass sich die Anzahl der Gatter der synthetisierten Schaltung zwischen 4743 für LeonardoSpectrum und 603 für Yosys_Gatemate bewegt. Auffallend ist, dass LeonardoSpectrum die höchste Anzahl von Gattern zur Synthese genutzt hat, aber die Fläche im Vergleich dazu relativ gering ist. Dies ist dadurch zu erklären, dass 3242 von den insgesamt 4743 Gattern Inverter sind.

Die Leistungsaufnahme der ALU ist für die Tools Genus und Design-Compiler in Ta-

Tool	Genus	DC
Internal Power	581,47 uW	282,58 uW
Switching Power	37,34 mW	13,4 mW

Tabelle 33: Leistungsaufnahme ALU

belle 33 dargestellt. Ein Teilziel der Arbeit war die Konfiguration der Tools, sodass ein Vergleich der Leistungsaufnahme möglich ist. Auf dem gezeigten Stand weichen die Werte der Internal Power um 298,89uW beziehungsweise $\approx$ 51% voneinander ab. Die Werte der Switching Power unterscheiden sich um 26,65mW beziehungsweise $\approx$ 71%. Unter der Beobachtung, dass sich die Anzahl der Gatter zwischen Design-Compiler und Genus ungefähr um 36% unterscheiden, kann die erhöhte Leistungsaufnahme nicht durch die Anzahl der Gatter erklärt werden.

18.2.2 *Canakari*

Der Vergleich der Gatterzahl und Fläche für den CAN-Controller, zeigt ähnliche Ergebnisse für die Tools Genus und Design-Compiler. Die durch LeonardoSpectrum synthetisierte Netzliste weist abermals die höchste Gatterzahl auf, hat jedoch eine relativ geringe Fläche von 3529, durch die Nutzung von 14720 Invertern. Die Unter-

Tool	Genus	DC	LeonardoSpectrum	Yosys	Yosys_GM
cell_count	6662	6882	21286	7291	2881
total_area	3085	2947,5	3529	3422,5	
Gatecount (pnr)	4044			4923	

Tabelle 34: Anzahl der Gatter und Fläche für Canakari

Tool	Genus	DC
Internal Power	967,39 uW	933,34 uW
Switching Power	48,52 mW	48,03 mW

Tabelle 35: Leistungsaufnahme Canakari

schiede in der Leistungsaufnahme betragen 34,05uW beziehungsweise $\approx 3,5\%$ für die Internal Power und 0,49mW beziehungsweise $\approx 1\%$. Die Ähnlichkeit der Werte weist auf eine korrekte Konfiguration der Tools hin.

18.3 Optimierung

Die Graphen 39 und 40 zeigen in welchen Wertebereichen der Algorithmus die meisten Evaluierungen für die Optimierungsparameter intrinsic_rise, intrinsic_fall, rise_resistance, fall_resistance, default_input_pin_cap und wire_load_capacitance durchgeführt hat und inwiefern sich das Optimum bei zunehmenden Iterationen verändert hat.

18.3.1 *Gatecount*

In Abbildung 39 ist das Optimierungsergebnis für 1125 Iterationen dargestellt. Das genutzt Tool ist Genus und das Design ist IMS_ALU.

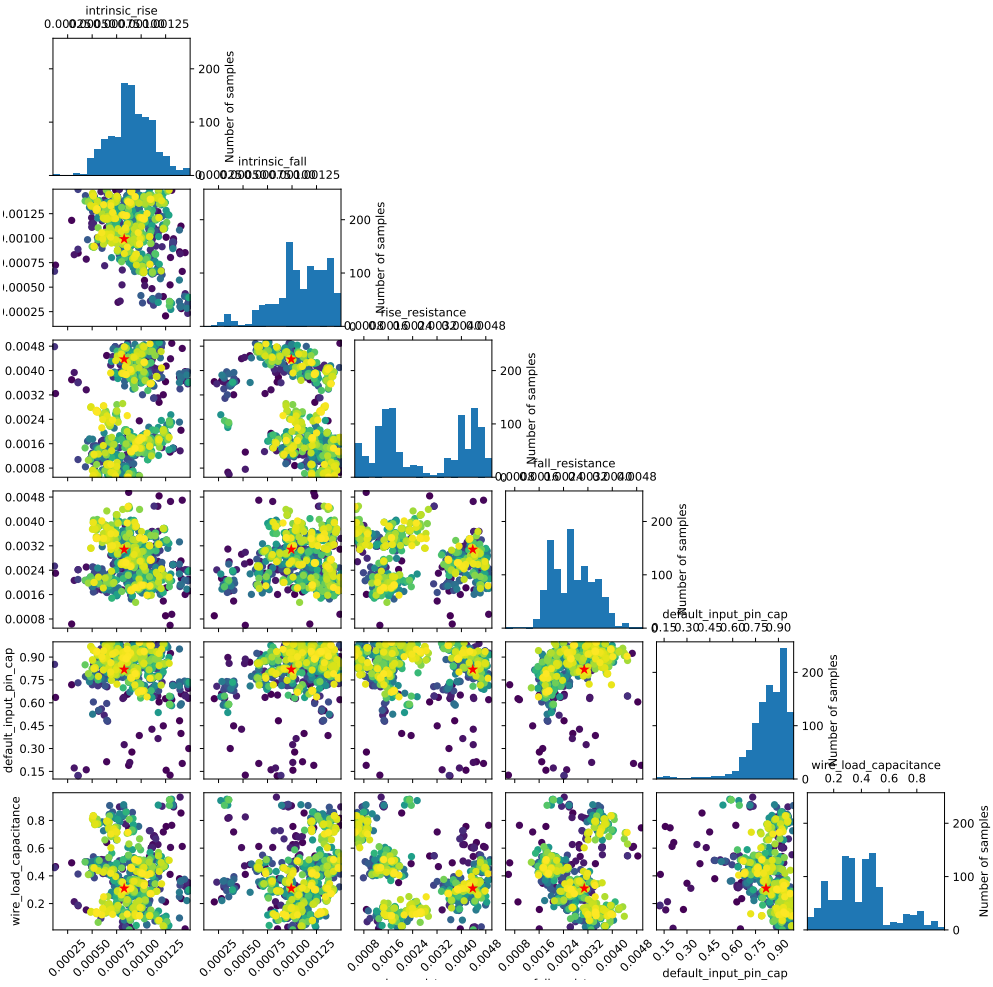

Abbildung 39: Gatecount ALU Genus, n=1125

Aus den Abbildungen ist zu erkennen, in welchen Bereichen der Optimierungsal-

Dimension	Wert
intrinsic_rise	0.000826759785425684
intrinsic_fall	0.0009921578554420707
rise_resistance	0.004372459811584241
fall_resistance	0.0030801156249628896
default_input_pin_cap	0.8188751291959171
wire_load_capacitance	0.31012168793282874

Tabelle 36: Parameter im Optimum für Gatterzahl

gorithmus Stichproben entnommen hat. In der Diagonalen werden Histogramme für jeden der Optimierungsparameter dargestellt. Die Diagramme unterhalb der Diagonale sind Streudiagramme der Stichproben für alle Kombinationen von Optimierungsparametern. Die Reihenfolge, in der die Punkte ausgewertet wurden, ist durch die Farbe der einzelnen Punkte markiert. Dunklere Farben entsprechen früheren Proben und hellere Farben entsprechen späteren Proben. Ein roter Stern zeigt die Lage des Optimums an, das durch den Algorithmus gefunden wurde. Es ist zu erkennen, dass sich die Punkte um die Position des Optimums herum sammeln. Punkte, die abseits des Optimums liegen, sind dunkel gefärbt. Dies hat den Grund, dass der Algorithmus zu Beginn versucht ein größeres Gebiet zu samplen. Ist ein Gebiet gefunden, in dem sich starke Verbesserungen ergeben, werden in diesem Bereich mehr Stichproben entnommen.

Durch die Optimierung konnte die Gatterzahl von anfänglich 779 Gattern auf 736 reduziert werden. Die Parameter bei diesem Optimum sind:

```
x: [0.000826759785425684, 0.0009921578554420707, 0.004372459811584241,
    0.0030801156249628896, 0.8188751291959171, 0.31012168793282874]
```

Die Werte für die einzelnen Parameter sind in Tabelle 36 aufgeführt.

18.3.2 *Interpretation der Ergebnisse*

Für die Optimierung der Gatter ist keine Funktion bekannt, sodass es nicht möglich ist eine genaue Vorhersage über die zu erwartenden Ergebnisse zu treffen. Es ist jedoch anzunehmen, dass Genus die Schaltungen hinsichtlich des Timings und der Leistungsaufnahme optimiert, was dazu führt, dass der Synthesealgorithmus bei hohem Leistungsverbrauch und hohen Verzögerungszeiten eine möglichst geringe Anzahl von Gattern nutzt. Die Ergebnisse unterstützen diese These nur teilweise. Es ist zu erkennen, dass die Werte für die intrinsischen Anstiegs- und Abfallzeiten im mittleren Bereich liegen. Das Optimum für den Wert rise_resistance findet sich bei $\approx 0,043$. Was darauf hindeutet, dass ein höherer Wert zu einer geringeren Gatterzahl durch die Synthese führt. Der default_input_pin_cap-Wert bestätigt die Annahme ebenfalls, da hier das Optimum bei $\approx 0,82$ liegt.

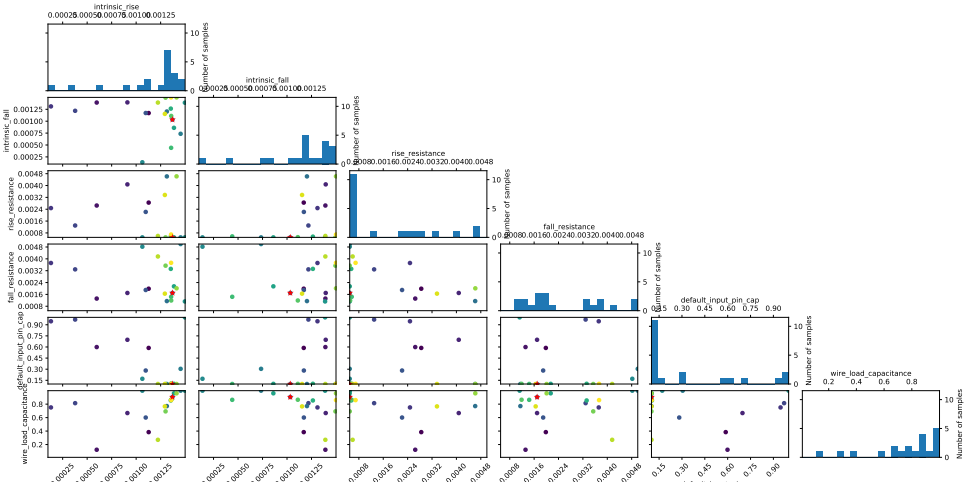

Abbildung 40: Maximum Clock Frequency ALU Genus, n=410

18.3.3 *Maximum Clock Frequency*

Abbildung 40 zeigt ähnlich zu Abbildung 39 Histogramme für die Anzahl der Evaluierungen an den jeweiligen Punkten und Streudiagramme, welche für die Optimierungsparameter paarweise die gesampelten Werte darstellen. Durch die geringere Anzahl von Optimierungsdurchläufen im Vergleich zur Optimierung des Gatecount, zeigen sich geringere Häufungen. Im Bereich dieser Häufungen befindet sich hierbei jedoch auch wieder das Optimum.

Abbildung 41 zeigt die gefundene maximale Taktfrequenz in Abhängigkeit von den Iterationen des Algorithmus. Es ist zu erkennen, dass bei weniger als 50 Iterationen ein mögliches Optimum mit einer maximalen Taktfrequenz von $\approx$ 50MHz gefunden wurde. Der Suchalgorithmus nähert sich daraufhin weiter $\approx$ 40MHz an.

Das gefundene Optimum steht in der Textdatei, welche am Ende der Optimierung geschrieben wird. Die maximale Taktfrequenz konnte von 38MHz auf 50,76MHz erhöht werden. Die Parameter im Optimum sind angegeben mit:

```
x: [0.0001, 0.0001, 0.0005, 0.0034630298056336946, 0.9415031049302455,
0.2437739318047803]
```

Die Parameter zusammen mit den Werten im Optimum sind in Tabelle 37 aufgeführt.

18.3.4 *Interpretation der Ergebnisse*

Für eine hohe maximale Taktfrequnz ist zu erwarten, dass sich sehr geringe Werte für die intrinsischen Anstiegs- und Abafallzeiten, die Anstiegs- und Abfallwiderstände sowie die Eingangs- und Leitungskapazitäten ergeben. Grund dafür ist, dass diese

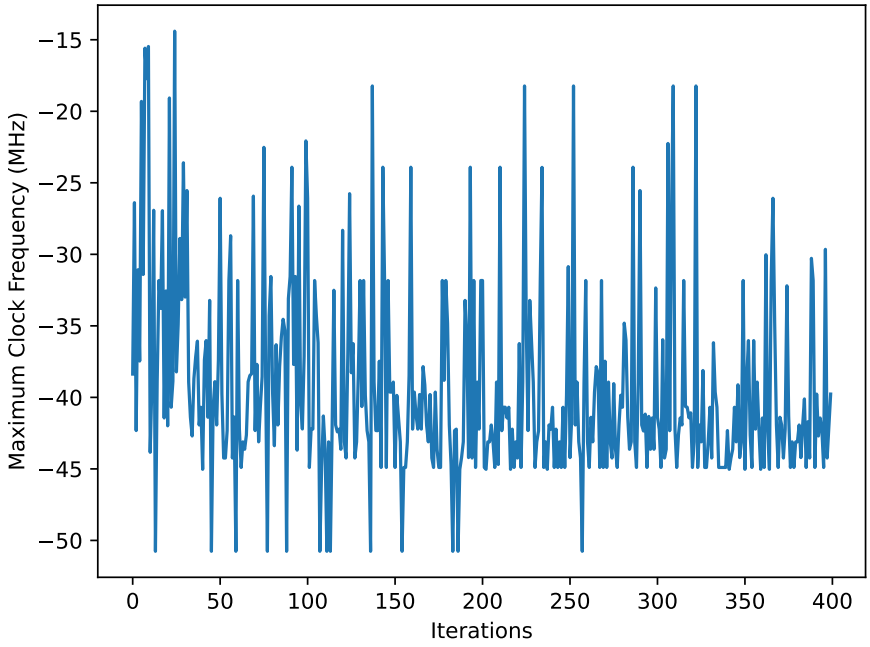

Abbildung 41: Maximum Clock Frequency ALU Genus, n=410

Werte die Verzögerungszeiten der Schaltung erhöhren.

Die intrinsischen Anstiegs- und Abfallzeiten sind hierbei für das intrinsische Delay verantwortlich. Die Eingangspin- und Leitungskapazitäten sowie die Widerstände erhöhen das Transition Delay. Die Betrachtung der Ergebnisse zeigt, dass sich die Ergebnisse mit der Theorie decken.
Die Parameter intrinsic_rise, intrinsic_fall und rise_resistance haben den kleinstmöglichen Wert angenommen, welcher über die low-Grenze innerhalb des Optimierungsskripts angegeben ist. Der Parameter fall_resistance nimmt innerhalb seiner Grenzen von 0,0005 und 0,005 einen relativ hohen Wert an. Die Eingangspin- und Leitungskapazität nehmen beide relativ hohe Werte an, welche ausgehend von der Berechnung der Verzögrungszeiten nicht zu erwarten waren.

Dimension	Wert
intrinsic_rise	0.0001
intrinsic_fall	0.0001
rise_resistance	0.0005
fall_resistance	0.0034630298056336946
default_input_pin_cap	0.9415031049302455
wire_load_capacitance	0.2437739318047803

Tabelle 37: Parameter im Optimum für maximale Taktfrequenz

19 Resümee

Anhand der in Kapitel 18 dargestellten Ergebnisse ist zu sehen, dass die Implementierung des Synthese- sowie Optimierungsablaufs mithilfe des Skripts durchgeführt werden kann.

Die Synthesefunktionalität ermöglicht es alle in Kapitel 12 vorgestellten Synthese-Tools zu nutzen und die Ergebnisse der Synthese vergleichend in einer Tabelle darzustellen. Innerhalb des Syntheselaufs ist es möglich Funktionalitäten mithilfe von Kommandozeilenargumenten zu konfigurieren. Durch den Aufruf dieser Kommandozeilenargumente kann die logische Äquivalenzprüfung sowie die Nutzung der CC-Tools XConvert und pnr hinzugefügt werden. Die Überprüfung der Funktionalitäten hat gezeigt, dass bei Nutzung der Arithmetischen Logikeinheit die Synthese und Optimierung fehlerfrei funktionieren sowie die relevanten Parameter korrekt dargestellt werden. Der Vergleich der simulierten Leistungsaufnahme der Arithmetischen Logikeinheit zwischen den Tools Genus und Design-Compiler hat gezeigt, dass die erstellte Bibliothek imes_cc zu einer Abweichung von 71% bei der Switching Power und 51% bei der Internal Power führt. Für den CAN-Controller beträgt die Abweichung $3,5\%$ für die Internal Power und 1% für die Switching Power.

Weitere Bemühungen sind bei der Synthese des CAN-Controllers *Canakari*, mittels LeonardoSpectrum, Yosys und Yosys_Gatemate nötig, da die Netzlisten dieser Tools durch die logische Äquivalenzprüfung durchfallen. Hierbei sollte das Augenmerk darauf liegen, die Optimierungsschritte der Tools auszuwerten und diese nach Möglichkeit an das genutzte Tool zur logischen Äquivalenzprüfung zu übermitteln.

Die Optimierung der Synthese-Ergebnisse hat gezeigt, dass Verbesserungen durch die Anpassung von Zellbibliotheksattributen erzielt werden können. Diese Verbesserungen beziehen sich auf die Gatterzahl (Gatecount) sowie die maximale Taktfrequenz (Maximum Clock Frequency). Die maximale Takfrequenz konnte durch Optimierung von anfänglichen 38.37MHz auf 50.76MHz erhöht werden. Bei der Optimierung der Gatterzahl ist eine Verringerung von 779 auf 736 Gatter erreicht worden.

19.1 Ausblick

Ausgehend von den hier vorgestellten Ergebnissen ist eine Vielzahl von Weiterentwicklungen denkbar.

Der genutzt Optimierungsalgorithmus gp_minimize bietet eine Vielzahl an Konfigurationsmöglichkeiten, welche bisher kaum ausgeschöpft wurden. Hierbei ist eine Untersuchung denkbar, inwiefern die Konfiguration verschiedener Optimierungspa-

rameter zu einer Verbesserung der Optimierungsergebnisse führt.

Innerhalb des Skripts sind Entwicklungen möglich, die das Skript um Optimierungsmöglichkeiten bei der Nutzung mit Yosys erweitern. Bisher war eine Optimierung mit Yosys nicht möglich, da Yosys die Parameter, welche zur Optimierung modifiziert werden, nicht unterstützt. Die Konfiguration des Skripts kann ebenfalls dahingehend angepasst werden, dass die constraints, welche für die Synthese genutzt werden, wie z.B. die Taktfrequenz ebenfalls zu einer Dimension der Optimierung werden.

In Bezug auf den Syntheseablauf hat sich gezeigt, dass die durch Yosys und LeonardoSpectrum synthetisierten Netzlisten nicht die logische Äquivalenzprüfung bestehen. Hierbei ist es sicherlich sinnvoll die Optimierungen, welche durch die Tools angewendet werden, zu untersuchen und diese in den LEC-Tools einzustellen.

20 Anhang - USB-Stick

Auf dem dem beigefügten USB-Stick sind folgende Inhalte zu finden:

- masterthesis_beer.pdf
- Ordner »Projekt«

Der Ordner enthält das Repository mit dem erstellten Quellcode.

Glossar

ASIC Application-specific Integrated Circuit. 7

DAG Directed Acyclic Graph. 14, 15

EDIF Electronic Design Interchange Format. 6

FOSS Free and Open Source Software. 6

FPGA Field Programmable Gate Array. 7

GUI Graphical User Interface. 5, 89

HDL Hardware Description Language. 6

LUT Lookup Table. 7, 54, 60

MPW Minimum Clock Pulse Width. iv, 11

NLDM Non Linear Delay Model. 41, 55

RTL Register Transfer Level. 6

SDF Standard Delay Format. 54

STA Static Timing Analysis. 12, 15

TCL Tool Command Language. 3

tcl tool command language. 5, 17, 89

VHDL Very High Speed Integrated Circuit Hardware Description Language. 6, 91

Literatur

[1] URL: https://de.wikipedia.org/wiki/Wahrscheinlichkeitsraum.

[2] URL: https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html#sklearn.gaussian_process.kernels.RBF.

[3] Kenneth C. Smith Adel S. Sedra. *Microelectronic Circuits. Sixth Edition*. Oxford University Press, 2009. ISBN: 9780195323030.

[4] Aaron Beer. *Charakterisierung von Zellen im Liberty-Format zur Einbindung in die Synthese digitaler Schaltungen*. 2020.

[5] Inc. Cadence Design Systems. *Conformal Equivalence Checking Command Reference*. 2020.

[6] Inc. Cadence Design Systems. *Conformal Equivalence Checking User Guide*. 2020.

[7] Inc. Cadence Design Systems. *Genus Command Reference*. 2019.

[8] Inc. Cadence Design Systems. *Genus User Guide*. 2020.

[9] Sarah Harris David Harris. *Digital Design and Computer Architecture*. 2007.

[10] Prof. Dr.-Ing. Thomas Giebel. *Grundlagen der CMOS-Technologie*. 1. Aufl. Vieweg+Teubner Verlag, 2002.

[11] Jochen Görtler, Rebecca Kehlbeck und Oliver Deussen. „A Visual Exploration of Gaussian Processes“. In: *Distill* (2019). https://distill.pub/2019/visual-exploration-gaussian-processes. DOI: 10.23915/distill.00017.

[12] Mentor Graphics. *LeonardoSpectrum™User's Manual*.

[13] Neil H.E. Weste David Money Harris. *Integrated Circuit Design. Fourth Edition*. Pearson, 2010.

[14] Michael A. Karagounis. „Design eines CAN Controllers mit VHDL und SpecChart“. Fachhochschule Köln, 2000.

[15] Exemplar Logic. *LeonardoSpectrum for Altera Reference Manual*. 2001.

[16] Mentor. *LeonardoSpectrum User Guide*. 2019.

[17] J. Bhasker (auth.) Rakesh Chadha. *Static timing analysis for nanometer designs: a practical approach*. 1. Aufl. Springer US, 2009.

[18] Richard Sohnius. *Standard Cell Characterization*. 2003. URL: http://ra.ziti.uni-heidelberg.de/pages/student_work/seminar/ws0304/richard_sohnius/praesentation.pdf.

[19] Jos Budi Sulistyo. „On the Characterization of Library Cells“. Virginia Polytechnic Institute und State University, 2000.

[20] Synopsys. *Liberty User Guide Volume 1*. 2019.

[21] Synopsys. *Power Compiler User Guide*.

[22] Synopsys. *Synopsys Timing Constraints and Optimization User Guide*. 2019.

[23] Dr. Adam Teman. *Digital VLSI Design*. URL: http://www.eng.biu.ac.il/temanad/digital-vlsi-design/.

[24] Clifford Wolf. *Yosys Manual*. 2012. URL: http://www.clifford.at/yosys/files/yosys_manual.pdf.

[25] Xilinx. *Vivado Design Suite User Guide*. 2019.