



Hochschule Bochum
Bochum University
of Applied Sciences

Campus
Velbert/Heiligenhaus



Institut für Angewandte Elek-
trotechnik

Evaluierung des „Cadence Joules RTL Design Studios“ für verschiedene Schaltungsentwürfe

von

Lukas Hoffleit

Matrikelnummer: 183 335 27

Dokumentation des KIS-Projekt-3 im Studiengang
Technische Informatik

Eingereicht am: 23. August 2025

Prüfer: Prof. Dr.-Ing. Mohammad Ashfaq

Abstract

Das Ziel der vorliegenden KIS-Projektarbeit ist es, die Software „Joules RTL Design Studio“ von Cadence zu evaluieren. Hierfür wird es auf verschiedene ASIC-Entwürfe angewandt. Ein besonderes Augenmerk wird hierbei auf die Möglichkeit gelegt, die Leistungsaufnahme eines Schaltungsentwurfes frühzeitig abzuschätzen.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Aktueller Stand der ASIC-Entwicklung	1
1.2 Aufgabenstellung und Zielsetzung	1
1.3 Aufbau der Arbeit	1
2 Grundlagen	3
2.1 Traditioneller Design-Flow bei der ASIC-Entwicklung	3
2.1.1 Design Planning	4
2.1.2 Placement	4
2.1.3 Clock-Tree-Synthese	5
2.1.4 Routing	5
2.1.5 Signoff	6
2.2 Metriken bei der ASIC-Entwicklung	6
2.2.1 Power (Leistungsaufnahme)	6
2.2.2 Performance	8
2.2.3 Area (Größe)	8
2.2.4 Congestion	9
2.2.5 Zusammenhang zwischen den PPAC-Metriken	9
2.3 Joules RTL Design Studio	10
2.4 Tool-Command-Language	10
2.5 Verwendete Designs	11
2.5.1 Nick Asic	11
2.5.2 Cortex-A53 CPU	11
3 Prototype Design	12
3.1 Aufsetzen des Flows	12
3.2 Vergleich Low- und High-Effort am Nick-Asic	13
3.2.1 Begrenzung der Routing-Ressourcen	14
3.2.2 Erhöhung der Timing-Anforderungen	15
3.3 Ressourcennutzung	17
3.4 Vorteile für die Front-End Entwicklung	17
4 Nachvollziehen der vorhandenen Ergebnisse mit PTPX und Voltus	19
4.1 Methodik zur Ursachenanalyse	19
4.1.1 Berechnung der Internal Power	20
4.1.2 Exemplarische Berechnung der internen Leistung in Voltus	21
4.1.3 Exemplarische Berechnung der internen Leistung in PrimeTime	22
4.1.4 Berechnung der Toggle-Rate	22
4.1.5 Anpassen der Constraints	23
4.1.6 Toggle-Rate innerhalb eines Logik-Pfades	23
4.1.7 Mapfile	25
4.1.8 Angeben des Fensters	25
4.2 Ergebnisse der Maßnahmen	26
5 Implementierung mit traditioneller Software	27
5.1 Schritte zur Implementierung	27
5.2 Power-Analyse	29
5.2.1 Joules	29

5.2.2 Vergleich mit Innovus	31
6 Zusammenfassung und Ausblick	32
Literatur- und Quellenverzeichnis	33
Anhang	36

Nomenklatur

Abkürzungen

ADC	Analog-Digital-Converter
ASIC	Application Specific Integrated Circuit
CTS	Clock-Tree-Synthese
DRC	Design Rule Check
EDA	Electric-Design-Automation
GDSII	Graphic Data System II
GND	Massepotential
I/O	Input/Output
IR-Drop	Spannungsabfall durch Innenwiderstand
LEC	Logical Equivalence Check
LVS	Layout versus Schematic
MMMC	Multi-Mode-Multi-Corner
NDR	Non-Default Rules
PLL	Phase-Locked Loop
PPAC	Power, Performance, Area, Congestion
QoR	Quality of Results
REE	Renesas Electronics Europe
RTL	Register Transfer Level
SoC	System on Chip
SPEF	Standard Parasitic Exchange Format
STA	Static-Timing-Analysis
TCL	Tool-Command-Language
VDD	Versorgungsspannung
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Englische Fachbegriffe

Clock Buffer	Verstärkerzelle im Taktnetz
Clock Domain	Taktbereich innerhalb eines Designs
Clock Gating	Taktabschaltung zur Energieeinsparung
Clock Skew	Taktversatz zwischen Registern
Clock Tree	Verzweigtes Taktnetzwerk
Congestion	Verdrahtungsengpass im Layout
Density	Flächenauslastung im Chipdesign
Design Planning	Planungsschritt im physikalischen Design
Floorplan	Grundriss des Chiplayouts
Insertion Delay	Verzögerung des Taktsignals vom Ursprung bis zum Ziel
Mapfile	Datei zur Zuordnung von Instanznamen zwischen Netzlisten
Placement	Platzierung der Zellen im Chipdesign
Routing	Verdrahtung der Zellen im Chipdesign
Signoff	Abschlussprüfung vor Fertigung
Slack	Zeitreserve im Timing-Pfad

Timing Violation	Verletzung zeitlicher Anforderungen
Toggle Rate	Schaltfrequenz eines Signals
Yield	Ausbeute

Symbole

C	Kapazität
C_{int}	Interne Kapazität
C_{load}	Lastkapazität
f	Frequenz
I	Strom
I_{intsw}	Interner Schaltstrom
I_{leak}	Statischer Leckstrom
I_{SC}	Kurzschlussstrom
I_{sw}	Schaltstrom der Last
P	Power (Leistung)
P_{leak}	Leakage-Power
P_{sw}	Switching-Power
P_{intsw}	Interne Schaltleistung
P_{int}	Interne Leistung (Internal-Power)
P_{leak}	Leistung durch statischen Leckstrom (Leakage-Power)
P_{SC}	Kurzschlussleistung
P_{sw}	Schaltleistung (Switching-Power)
T	Periodendauer
T_D	Toggle-Rate einer Zelle
V_{dd}	Versorgungsspannung

Verwendete Software

Genus	Genus Synthesis Solution – Software von Cadence zur Synthese
Innovus	Innovus Implementation System – Software von Cadence für die Physikalische Implementierung (Invs)
Joules	Joules RTL Design Studio – Software von Cadence zum erstellen eines Prototypen
PrimeTime PX	Software von Synopsys zur Power-Analyse (PTPX)
Simvision	Waveform-Viewer zur Anzeige von Signalverläufen von Cadence
Voltus	Software von Cadence zur Power-Analyse

1 Einleitung

Im Laufe dieser Arbeit soll die Software *Cadence Joules RTL Design Studio*, nachfolgend Joules genannt, für die Entwicklung von Halbleitern im Unternehmen Renesas Electronics Europe (REE) evaluiert werden. Das Unternehmen REE stellt unter anderem ASICs (Application Specific Integrated Circuit) her, bei denen die Beschreibung der Logik in Form von RTL-Code (Register-Transfer-Level) vom Kunden geliefert wird, oder diese Beschreibung intern in Form von Hardwarebeschreibungssprachen wie Verilog- oder VHDL-Code stattfindet. Abschnitt 1.1 stellt die aktuellen Herausforderungen bei der Halbleiterentwicklung dar, welche mit Joules gelöst werden sollen. Abschnitt 1.2 erläutert, welche Schwerpunkte bei der Evaluierung im Verlauf dieser Arbeit gesetzt werden sollen. Abschließend umreißt Abschnitt 1.3 kurz den Aufbau dieser Arbeit.

1.1 Aktueller Stand der ASIC-Entwicklung

Traditionell sind bei der Entwicklung von Halbleitern die Abschnitte zwischen der logischen Beschreibung der Hardware und der physikalischen Implementierung strikt getrennt. So kann es dazu kommen, dass strukturelle Probleme erst spät im Entwicklungsprozess erkannt werden. Seit Jahren wird daher bereits versucht, physikalische Aspekte frühzeitig zugänglich zu machen, beispielsweise mit der physikalischen Synthese. Joules versucht dieses Problem zu lösen, indem es die Möglichkeit zur Verfügung stellt, während des Entwicklungsprozesses der Logik einen Prototypen anzufertigen, um eine grobe Implementierung nachzuahmen. So sollen mögliche Implementierungsprobleme schon vor der eigentlichen, physikalischen Implementierung aufgezeigt und für den Logikentwickler sichtbar gemacht werden.

1.2 Aufgabenstellung und Zielsetzung

Im Verlauf dieser Arbeit soll ergründet werden, inwiefern die Anwendung von Joules im Betrieb REE sinnvoll ist. Hierzu soll zunächst die Kernfunktionalität, die Erstellung eines Prototypen, evaluiert werden. Dabei soll geprüft werden, ob es mit Joules möglich ist, Probleme, die typischerweise erst während der Implementierung auffallen, schon frühzeitig, während des logischen Designs zu erkennen. Außerdem soll überprüft werden, wie akkurat die Funktionen der Power-Analyse in Joules sind. Hierfür soll eine Implementierung mit klassischen Tools durchgeführt werden. Das Ergebnis dieser Implementierung soll dann mithilfe von Sign-Off-Tools wie Cadence Voltus und Synopsys PrimeTime mit dem Prototypen aus Joules verglichen werden.

1.3 Aufbau der Arbeit

Diese Arbeit beschäftigt sich in Abschnitt 2 zunächst mit den für das Verständnis dieser Arbeit erforderlichen Grundlagen. Hier wird der traditionelle Design-Prozess mit den zugehörigen Metriken in der Halbleiterentwicklung sowie die Software Joules-RTL-Design-Studio vorgestellt. Daraufhin wird in Abschnitt 3 die Grundfunktionalität von Joules, das Erzeugen eines Prototypen, evaluiert. Abschnitt 4 beschäftigt sich mit dem Aufsetzen einer Power-Analyse und den Unterschieden zwischen Voltus und PrimeTime. Nachfolgend wird im Abschnitt 5 eine Implementierung mit der bei REE

schon bekannten Software Innovus durchgeführt und mit dem Prototypen aus Joules verglichen. Abschließend folgt in Abschnitt 6 eine Zusammenfassung und ein Fazit der Untersuchungen dieser Arbeit und ein Ausblick auf künftige Anwendungen.

2 Grundlagen

Dieses Kapitel beschäftigt sich mit den Grundlagen, welche für das Verständnis dieser Arbeit erforderlich sind. Darauffolgend beschreiben die Abschnitte 2.1 und 2.2 den traditionellen Design-Flow von ASICs, sowie zentrale Metriken, auf die während der Implementierung geachtet wird. Anschließend beschreibt der Abschnitt 2.3 die Software Joules und ihre Einordnung in das EDA-Ökosystem. Der Abschnitt 2.4 beschreibt die Programmiersprache TCL, welche als Schnittstelle für die Implementierungswerzeuge verwendet wird. Zuletzt werden die beiden für die Untersuchungen verwendeten Designs in Abschnitt 2.5 vorgestellt.

2.1 Traditioneller Design-Flow bei der ASIC-Entwicklung

Der Design-Prozess eines ASICs ist in der Regel in verschiedene Schritte aufgeteilt, welche von unterschiedlichen Abteilungen übernommen werden. In Abbildung 2.1 sind diese Schritte zu sehen. Die in Rot gekennzeichneten Schritte werden in der Regel vom *Front-End*-Team durchgeführt, während die in Blau gekennzeichneten Schritte vom *Back-End*-Team erledigt werden (Abhay Chopde, 2024). Somit kommt der Front-End-Designer im traditionellen Design-Flow nicht in Kontakt mit technologiespezifischen Details der physikalischen Implementierung.

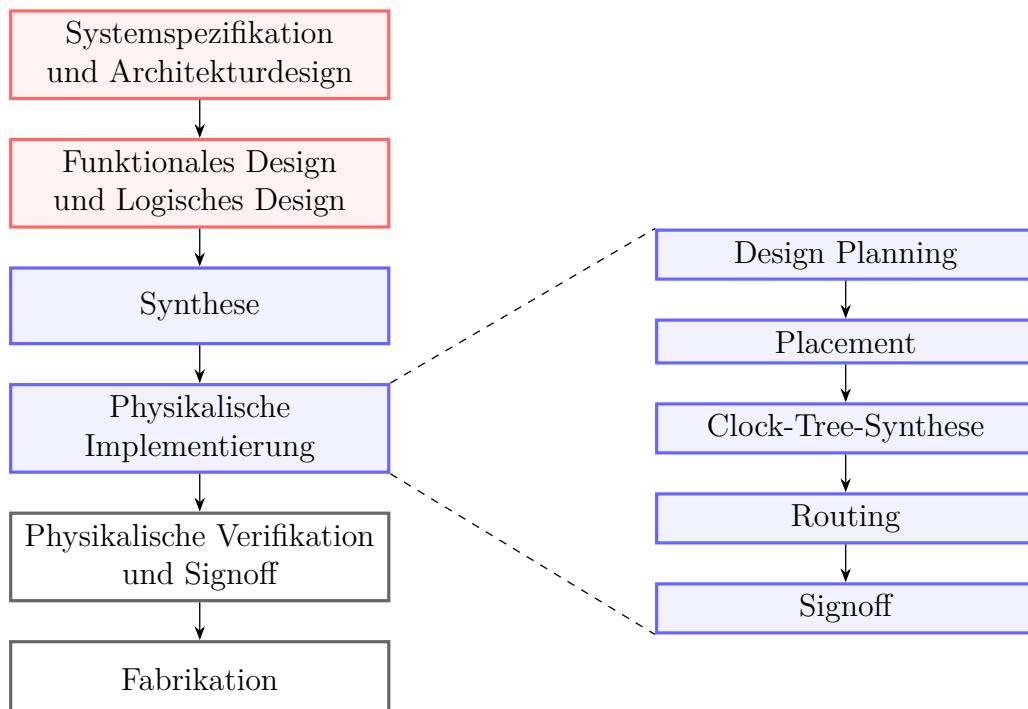


Abbildung 2.1: Die Schritte im Design-Flow, vereinfacht nach Andrew B. Kahng (2022, S. 6)

Zunächst wird das System spezifiziert und die Architektur erstellt. Hierbei werden die Ziele für das finale Produkt festgelegt, welche während des restlichen Ablaufes des Designs beachtet werden müssen. Darauf folgt das funktionale Design. Hier wird das Verhalten aller Module definiert. Im logischen Design wird dieses funktionale Design auf RTL-Ebene mit einer Hardwarebeschreibungssprache wie Verilog oder VHDL umgesetzt.

Ein Tool zur Logiksynthese übersetzt das Design in eine Netzliste. Dabei handelt es sich um eine Gate-Level Netzliste, die die gewünschte Funktionalität mit Standard-Logik-Gattern (digitale Standardzellen) abbildet. Elemente wie RAM (Random-Access-Memory) Blöcke, Makros (z.B. ADCs, Serdes, PLL), I/O (Input/Output), Pad-Zellen und Ports werden manuell in dem RTL-Code instanziert und bei der Synthese übernommen.

Bei der physikalischen Implementierung werden die Komponenten auf dem Chip platziert und verbunden. Das physikalische Design kann in Unterabschnitte gegliedert werden, welche in den folgenden Abschnitten (2.1.1 - 2.1.5) beschrieben sind. Über die einzelnen Schritte wird iterativ optimiert, um am Ende eine spezifikationsgerechte Implementierung zu erhalten.

Bei der Verifikation wird überprüft, ob das Design die logischen, elektrischen und durch die Technologie bedingten Anforderungen erfüllt. Ist dies der Fall, kann der ASIC hergestellt werden (Andrew B. Kahng, 2022, S. 6-10).

2.1.1 Design Planning

Das Design Planning ist der erste Implementierungsschritt. Bei größeren Projekten wird das Design zunächst hierarchisch in kleinere Partitionen aufgeteilt, welche einzeln bearbeitet werden können. Daraufhin wird ein Floorplan erstellt. Im Floorplan werden die Größe, Form und Position der Partitionen, Makros wie IPs und Memory-Blöcke (RAM) und IO-Zellen festgelegt. Abschließend wird der Power-Plan erstellt. Die Power-Leitungen für ground (GND) und supply (VDD) werden über Externe Ports angebunden. Daraufhin werden die Spannungen in einem Gitter über den gesamten Die verteilt. In Abbildung 2.2 ist ein Floorplan vor und nach dem Power-Routing zu sehen.

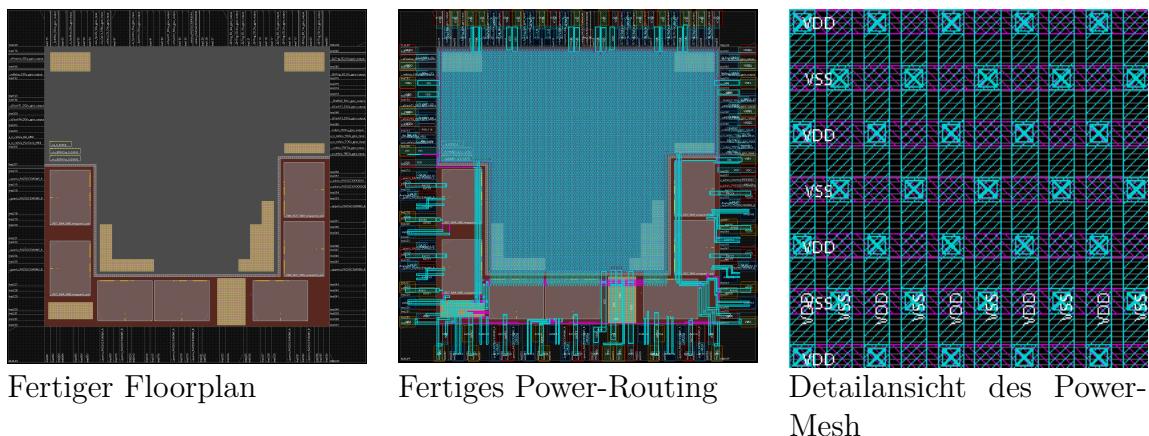


Abbildung 2.2: Ein Floorplan vor und nach dem Power-Routing

2.1.2 Placement

Beim Placement werden die Positionen der Standardzellen innerhalb des Designs festgelegt. Die Qualität der Platzierung der Zellen ist ausschlaggebend für das Ergebnis des Routings. Das Placement wird auf *Global Placement* und *Detailed Placement* aufgeteilt. Beim Global Placement werden Zellen grobe Positionen zugewiesen. Nach diesem Schritt ist das Design noch nicht valide, beispielsweise dürfen sich Zellen hier

noch überlappen. Beim Detailed Placement werden die Positionen der Zellen verfeinert, einzelne Zellen getauscht und auf valide Positionen platziert (Andrew B. Kahng, 2022, S. 95ff). Ein Beispiel für ein platziertes Design ist in Abbildung 2.3 zu sehen.

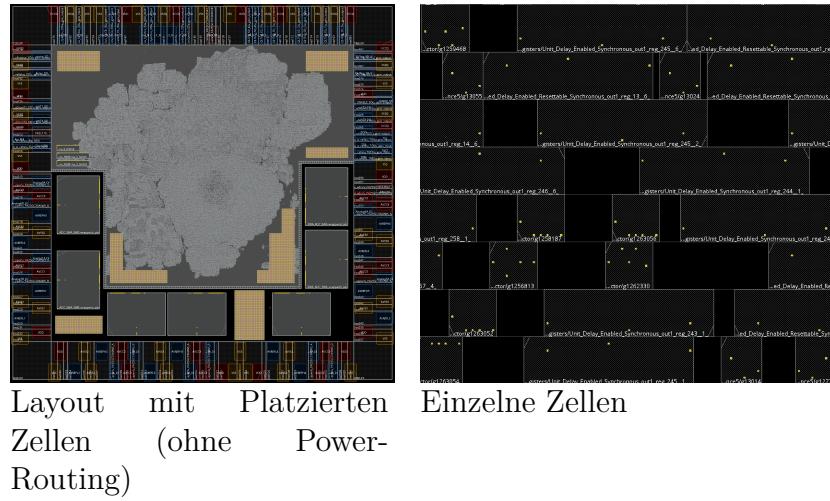


Abbildung 2.3: Ein Design nach dem Placement

2.1.3 Clock-Tree-Synthese

Ziel der Clock-Tree-Synthese (CTS) ist es, das Taktsignal von einer Quelle, wie einer PLL oder einem externen Takteingang, über eine strukturierte Topologie zu allen sequentiellen Elementen des Designs zu verteilen. Dabei wird besonderer Wert auf minimale Taktverzerrung (Clock Skew) und geringe Latenz gelegt, um die zeitliche Integrität des Designs sicherzustellen. Die CTS erfolgt durch das Einfügen von Buffer- und Inverterzellen entlang des Taktpfades. Um alle sequentiellen Elemente zu balancieren, können abhängig vom Design H- oder X-Bäume oder auch Clock-Mesh-Strukturen verwendet werden. Ein wesentliches Ziel der CTS ist die Minimierung des Insertion-Delay, also der Zeit, die das Taktsignal vom Ursprung bis zu den Flip-Flops benötigt. Gleichzeitig sollte die Differenz der Ankunftszeiten (Skew) zwischen Start- und Zielregister gering gehalten werden, um sowohl das Setup- als auch das Hold Timing erreichen zu können. Die CTS nutzt dazu spezielle Routingregeln (Non-Default Rules, NDR), bei denen Takteleitungen mit erhöhter Breite und reduzierter Widerstandsfähigkeit verlegt werden.

Die Qualität der CTS beeinflusst damit auch die maximal erreichbare Taktfrequenz eines Designs. Diese hängt ab von der Summe aus kombinatorischer Verzögerung, Setup-Zeit und Skew. Die Einhaltung dieser Timing-Anforderungen ist Voraussetzung für die funktionale Korrektheit und Zuverlässigkeit des Chips (Chakravarthi, 2022, S. 56 - 63).

2.1.4 Routing

Ziel des Routing-Schrittes ist die vollständige und regelkonforme Verbindung aller logischen Netze zwischen Standardzellen, Makros und Ein-/Ausgabepads innerhalb des Chiplayouts. Dabei erfolgt die physikalische Realisierung der Verbindungen durch leitende Segmente in mehreren Metallschichten, wobei die Routingregeln der jeweiligen

Fertigungstechnologie strikt einzuhalten sind. Die Einzelnen Routing-Schichten werden mit so genannten Vias verbunden.

Das Routing gliedert sich in zwei Hauptphasen: das *Global Routing* und das *Detail Routing*. Im Global Routing wird der Chip in rechteckige Regionen, sogenannte *gCells*, unterteilt. Innerhalb dieser Regionen werden Routingkanäle definiert, die eine grobe Planung der Signalverbindungen ermöglichen. Ziel ist es, eine optimale Verbindungsstrategie zu identifizieren, die sowohl die Designregeln als auch die Anforderungen an Timing und Signalintegrität berücksichtigt. Im anschließenden Detail Routing erfolgt die konkrete physikalische Umsetzung der geplanten Verbindungen. Dabei werden die Metallschichten entsprechend den zuvor definierten Tracks gefüllt. Beim Detail-Routing werden Pfade nur horizontal und vertikal verbunden (Chakravarthi, 2022, S. 65 ff).

2.1.5 Signoff

Die Signoff-Phase markiert den finalen Abschnitt im physikalischen Designprozess eines ASICs. In dieser Phase wird überprüft, ob das Design alle funktionalen, zeitlichen und physikalischen Anforderungen erfüllt, bevor es zur Fertigung freigegeben wird.

Zur Unterstützung des Signoff-Prozesses kommt spezialisierte EDA-Software zum Einsatz. Diese analysieren das Design auf Timing-Violations mithilfe einer Static-Timing-Analysis (STA), Design Rule Violations (DRC), Spannungsprobleme (IR-Drop) und Elektromigration (EM). Bei Bedarf generieren sie automatisch ECO-Vorschläge, die anschließend in das Design integriert werden können. Die Software ist in der Lage, sowohl inkrementelle Änderungen als auch vollständige Designverifikationen durchzuführen (Chakravarthi, 2022, S. 105ff). Ein weiterer wichtiger Bestandteil ist die formale Verifikation, insbesondere der Logical Equivalence Check (LEC) und die Layout-vs-Schematic (LVS)-Analyse. Diese Methoden stellen sicher, dass die funktionale Integrität des Designs über alle Transformationsstufen hinweg erhalten bleibt. Der LEC vergleicht die logische Struktur des synthetisierten Designs mit der finalen, physikalisch implementierten Version, während LVS die Übereinstimmung zwischen Layout und Schaltplan überprüft (Chakravarthi, 2022, S. 107ff). Der Signoff-Prozess endet mit der Erstellung der finalen Design-Datenbank im GDSII- oder OASIS-Format. Diese Daten werden an die Fertigungsstätte übermittelt. Erst nach erfolgreicher Validierung durch die Foundry wird das Design für die Produktion freigegeben (Chakravarthi, 2022, S. 119).

2.2 Metriken bei der ASIC-Entwicklung

In der ASIC-Entwicklung spricht man traditionell von den drei zentralen Metriken PPA: Power, Performance und Area. Mit PPAC erweitert Cadence nun dieses Konzept nun um den Aspekt der Congestion, um die Routbarkeit eines Designs besser beurteilen zu können. Die Begriffe *Power*, *Performance*, *Area* und *Congestion* werden in den folgenden Unterabschnitten näher erläutert (Patil, 2021).

2.2.1 Power (Leistungsaufnahme)

Power bezieht sich auf den Leistungsaufnahme eines Halbleiters. In der Halbleiterentwicklung wird es immer wichtiger die Leistungsaufnahme zu minimieren, um zum einen die Wärmeentwicklung zu reduzieren oder längere Batterielaufzeiten zu ermöglichen.

Dies ist besonders wichtig für mobile Geräte und andere Anwendungen, bei denen die Batterielebensdauer eine entscheidende Rolle spielt. Techniken wie die Optimierung der Schaltkreise und die Verwendung von energieeffizienten Materialien tragen dazu bei, den Energieverbrauch zu senken. Bei der Leistungsaufnahme eines Halbleiters wird zwischen statischer und dynamischer Power unterschieden. Die statische Power ist hierbei im Gegensatz zur dynamischen Power nicht von der Schaltaktivität einer Zelle abhängig. Statische Power wird auch Leakage-Power (P_{leak}) genannt. Die Dynamische Power setzt sich aus Internal-Power (P_{int}) und Switching-Power (P_{sw}) zusammen. Die Gesamtleistung eines Halbleiters ist die Summe aus statischer und dynamischer Power, wie in Gleichung 2.1 zu sehen (Synopsys, 2025).

$$P_{\text{ges}} = \underbrace{P_{\text{leak}}}_{\text{statisch}} + \underbrace{P_{\text{int}} + P_{\text{sw}}}_{\text{dynamisch}} \quad (2.1)$$

Die Leakage-Power bezeichnet die Leistung, welche durch Leckströme im Halbleitermaterial verursacht wird. Dabei spielen verschiedene Effekte wie *subthreshold leakage*, *gate leakage* und *junction leakage* eine Rolle. Die Leakage-Power kann vereinfacht durch Gleichung 2.2 in Abhängigkeit von V_{dd} und der Versorgungsspannung dargestellt werden (Weste und Harris, 2015, S. 217).

$$P_{\text{leak}} = I_{\text{leak}} \cdot V_{\text{dd}} \quad (2.2)$$

Bei der internen Leistung handelt es sich um die Leistung, welche ausschließlich von den Internen-Faktoren einer Zelle abhängt. Die interne Leistung lässt sich in zwei Teile aufteilen, wie in Gleichung 2.3 zu sehen ist.

$$P_{\text{int}} = P_{\text{SC}} + P_{\text{intsw}} \quad (2.3)$$

Hierfür wird die Interne Kurzschlussspannung während des Schaltvorgangs einer CMOS-Logikzelle (P_{SC}), wie in Gleichung 2.4 und die Leistung für das Laden und Entladen der internen Kapazität (P_{intsw}), wie in Gleichung 2.5 benötigt.

$$P_{\text{SC}} = V_{\text{dd}} \cdot I_{\text{SC}} \quad (2.4)$$

$$P_{\text{intsw}} = \frac{1}{2} \cdot C_{\text{int}} \cdot V_{\text{dd}}^2 \cdot T_D \quad (2.5)$$

Durch Einsetzen dieser Gleichungen in Gleichung 2.3 ist die interne Leistung entsprechend Gleichung 2.6 zu berechnen (Synopsys, 2006, S. 7).

$$P_{\text{int}} = \underbrace{V_{\text{dd}} \cdot I_{\text{SC}}}_{P_{\text{SC}}} + \underbrace{\frac{1}{2} \cdot C_{\text{int}} \cdot V_{\text{dd}}^2 \cdot T_D}_{P_{\text{intsw}}} \quad (2.6)$$

Bei der Switching-Power handelt es sich um die Leistung, welche beim Laden und Entladen der Lastkapazität am Ausgang der Zelle anfällt. Ihre Berechnung ist identisch

zu der Berechnung der internen Leistung, allerdings wird statt der internen Kapazität die Lastkapazität verwendet (Synopsys, 2006, S. 7).

$$P_{\text{sw}} = \frac{1}{2} \cdot C_{\text{load}} \cdot V_{\text{dd}}^2 \cdot T_D \quad (2.7)$$

Mit Zuhilfenahme der Gleichungen für Leakage-, Internal- und Switching-Power ergibt sich in Gleichung 2.8 die Gesamtleistung einer Zelle.

$$P_{\text{ges}} = \underbrace{I_{\text{leak}} \cdot V_{\text{dd}}}_{P_{\text{leak}}} + \underbrace{V_{\text{dd}} \cdot I_{\text{SC}} + \frac{1}{2} \cdot C_{\text{int}} \cdot V_{\text{dd}}^2 \cdot T_D}_{P_{\text{int}}} + \underbrace{\frac{1}{2} \cdot C_{\text{load}} \cdot V_{\text{dd}}^2 \cdot T_D}_{P_{\text{sw}}} \quad (2.8)$$

Am Beispiel eines Clock-Buffers sind die verwendeten Ströme und Kapazitäten in Abbildung 2.4 zu sehen.

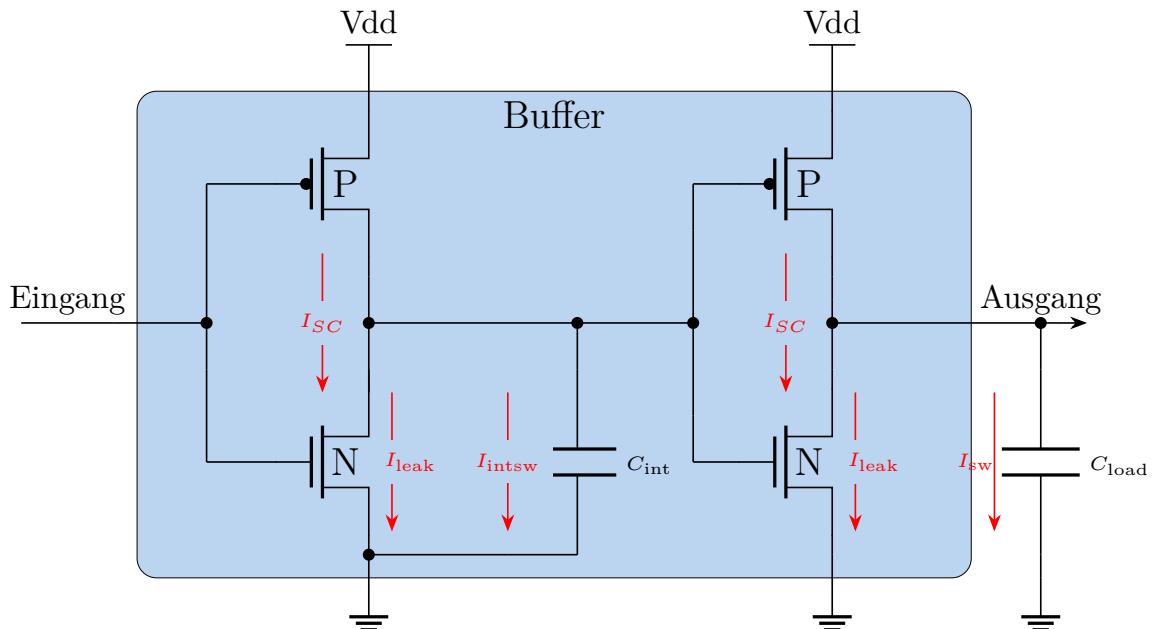


Abbildung 2.4: Der Schaltplan eines Buffers mit eingezeichneten Strömen, nach Synopsys (2006)

2.2.2 Performance

Performance beschreibt die Geschwindigkeit, mit der ein Halbleiter seine Aufgaben ausführt. Ein hoher Datendurchsatz ist entscheidend für Anwendungen, die schnelle Datenverarbeitung und hohe Rechenleistung erfordern, wie z.B. in Hochleistungsrechnern und Grafikprozessoren. Die Leistung kann durch verschiedene Faktoren beeinflusst werden, darunter die Taktfrequenz (interne Schaltvorgänge pro Sekunde) und die Architektur des Chips.

2.2.3 Area (Größe)

Area bezieht sich auf die physische Größe des Chips. Grundsätzlich angestrebt wird eine möglichst kleine Chipfläche, um die Herstellungskosten zu reduzieren. Dabei führt

eine kleiner Chipfläche zu einer geringeren Fehlerrate und damit höheren Ausbeute (Yield) sowie einer höheren Anzahl an Chips pro Wafer.

2.2.4 Congestion

Routing-Congestion beschreibt das Phänomen, dass die Nachfrage nach Routing-Ressourcen in bestimmten Bereichen eines Designs das verfügbare Angebot übersteigt. Die Hauptursachen liegen vor allem in modernen Fertigungsprozessen und der zunehmenden Designkomplexität.

Moderne Prozesse führen zu einer höheren Transistordichte, was kleinere Standardzellen ermöglicht. Dadurch sinkt jedoch die verfügbare Routing-Fläche pro Zellbereich. Gleichzeitig erschweren komplexere Designregeln wie Mindestabstände zwischen Vias oder Einschränkungen bei Layer-Übergängen das Routing zusätzlich. Hinzu kommt, dass die Routing-Ressourcen selbst limitiert sind: Während untere Metalllagen für lokale Verbindungen genutzt werden, sind obere Lagen für globale Stromversorgungsnetze und übergreifendes Routing reserviert. Die effektiv nutzbare Routing-Fläche nimmt dadurch weiter ab.

Parallel dazu steigt die Komplexität der Designs. Mehr IP-Blöcke, größere Speicher und umfangreichere Logik führen zu einer stark wachsenden Anzahl an Netzen und Pins. Dies resultiert in einem exponentiellen Anstieg der Verbindungsanforderungen, wodurch immer mehr Verbindungen um dieselben bevorzugten Routing-Ressourcen konkurrieren.

Routing-Congestion hat direkte Auswirkungen auf die Herstellbarkeit und die Performance eines Chips. Designs, die nicht ohne Kurzschlüsse verdrahtet werden können, sind nicht produzierbar. Selbst wenn eine Verdrahtung technisch möglich ist, erfordert sie oft einen hohen Aufwand und führt zu längeren Leitungswegen. Dies wirkt sich negativ auf das Timing-Verhalten aus und kann die Gesamtleistung des Chips erheblich beeinträchtigen. Mit fortschreitender Technologie und wachsender Designkomplexität verschärft sich das Problem weiter. Die Anzahl der Transistoren wächst schneller als die verfügbaren Routing-Schichten und die physikalischen Eigenschaften der Leitungen verschlechtern sich mit neueren Prozessgenerationen.

Um Routing-Congestion zu begegnen, ist eine frühzeitige Abschätzung und gezielte Optimierung notwendig. Dazu gehören Metriken zur Bewertung der Congestion sowie Strategien zur Vermeidung und Umleitung von Engpässen. Diese sollten in allen Phasen des Design-Flows berücksichtigt werden. Ziel ist es, klassische Designziele wie minimale Verzögerung und geringe Fläche mit einer hohen Routbarkeit zu vereinen (Saxena et al., 2007, S. 3 - 28).

2.2.5 Zusammenhang zwischen den PPAC-Metriken

Die Metriken Power, Performance, Area und Congestion stehen miteinander in direkter Konkurrenz, sodass bei der Entwicklung Prioritäten gesetzt werden müssen. Alle diese Metriken haben Einfluss auf die Gesamtkosten des Chips. Beispielsweise müssen für einen Chip mit maximaler Performance größere Gates verwendet werden, was den Chip größer macht und mehr Power benötigt wird (Weste und Harris, 2015, S. 205). Eine größere Chipgröße geht dann automatisch mit geringerem Yield (Ausbeute) einher,

was wiederum die Kosten steigert (Melzner, 2006). Auch bei einer zu hohen Routing-Congestion muss die Chip-Größe erhöht werden. Durch Routing-Congestion kann sich auch die Leistung eines Designs erheblich verschlechtern, da kritische Netze gezwungen sind, längere oder resistivere Wege zu nehmen, was zu höheren Verzögerungen führt (Saxena et al., 2007, S. 24f).

2.3 Joules RTL Design Studio

Das *Cadence Joules RTL Design Studio* soll eine Lösung zur Analyse und Optimierung von Register-Transfer-Level (RTL)-Designs darstellen und ist Teil der digitalen Design-Toolchain von Cadence. Ziel des Tools ist es, die Produktivität von Front-End-Designern signifikant zu steigern und die Qualität der Ergebnisse (Quality of Results, QoR) bereits vor der Übergabe an die Implementierungsphase zu verbessern. Ein zentrales Merkmal des Joules RTL Design Studio ist die Sichtbarkeit auf physikalische Designmetriken wie PPAC. Diese Metriken stehen den Designern nun bereits in der frühen Designphase zur Verfügung, was eine fundierte Entscheidungsfindung ermöglicht und Iterationen reduziert. Die Integration eines intelligenten Debugging-Assistenten erlaubt die Analyse von logischen, physikalischen und implementierungsbezogenen Aspekten des Designs. Zusätzlich gibt es die Möglichkeit, diese Metriken über den RTL-Entwicklungsprozess zu verfolgen und somit verschiedene RTL-Versionen zu vergleichen (Cadence, 2023).

Technologisch basiert das Tool auf bewährten Engines aus dem Cadence-Ökosystem, darunter das *Innovus Implementation System*, die *Genus Synthesis Solution* sowie die *Joules RTL Power Solution*. Diese gemeinsame Basis ermöglicht eine konsistente und effiziente Nutzung innerhalb eines einheitlichen Interfaces (Cadence, 2023a).

In der Praxis konnte das Tool laut Herstellerangaben die RTL-Konvergenz um den Faktor 5 beschleunigen und eine Verbesserung der QoR um bis zu 25 % erzielen. Damit soll Joules RTL Design Studio einen wesentlichen Beitrag zur Verkürzung der Entwicklungszyklen und zur Steigerung der Designqualität in modernen SoC-Projekten leisten (Cadence, 2023).

2.4 Tool-Command-Language

Die Tool-Command-Language (TCL) ist eine Skript-Sprache, welche von John Ousterhout entworfen wurde. Sie ist open-source und wurde auf verschiedenste Betriebssysteme portiert. Vom Aufbau der Sprache ähnelt TCL anderen UNIX-Shell-Sprachen, wie beispielsweise der Bourne-Shell oder der C-Shell (Welch et al., 2003, S. xlxi - li). Variablen können in TCL mit dem Schlüsselwort `set name` gesetzt werden und mit `$name` wieder abgerufen werden. Eigene Funktionen können mit dem Schlüsselwort `proc` definiert werden. Parameter können einer Funktion mit Leerzeichen getrennt übergeben werden.

TCL wird in den verschiedensten Bereichen angewandt. So ist es beispielsweise möglich, eine Web-Applikation basierend auf TCL zu schreiben, gleichzeitig ermöglicht das GUI-Framework TCL-TK grafische Benutzeroberflächen zu gestalten. Da es sich bei TCL um eine kompakte Sprache handelt, ist es einfach diese in andere Software einzubetten. So ist es beispielsweise für Electric-Design-Automation (EDA) Tools üblich, ein TCL-Interface bereitzustellen (tcl-lang.org, 2025). TCL wird in den Tools mit einer interaktiven Shell verwendet. Eine interaktive Shell ist im Laufe dieser Arbeit daran

zu erkennen, dass der Name des Tools als Kommentar (#) voransteht. Ausgaben der interaktiven Shell werden mit => gekennzeichnet. Dies ist auch bei den in dieser Arbeit verwendeten Werkzeugen von Cadence und Synopsys der Fall. Die verwendeten Tools von Cadence verfügen über das Kommando `get_db`, mit dem die internen Variablen des Tools, wie beispielsweise Informationen über das Design wie beispielsweise Designobjekte, abgefragt werden können (Cadence, 2025, S. 79).

2.5 Verwendete Designs

Im Verlauf dieser Arbeit werden zur Evaluierung von Joules zwei verschiedene Designs verwendet. Eine grobe Beschreibung dieser ist in den folgenden Unterabschnitten zu finden.

2.5.1 Nick Asic

Beim Nick Asic handelt es sich um einen ASIC mit 7 ADC-Instanzen (Analog-Digital-Converter). Bei den ADCs handelt es sich um Hardmakros. Interne Erfahrungen zeigen, dass dieses Design einfach zu implementieren ist, womit es sich für den Einstieg in Joules und erste Analysen eignet. Die Beschreibung des ASICs ist vollständig in VHDL und Verilog vorhanden.

2.5.2 Cortex-A53 CPU

Bei der Cortex-A53 CPU (im Folgenden auch `ca_53_cpu` genannt) handelt es sich um einen CPU-Kern der Firma ARM. Dieser wird als Partition innerhalb eines größeren Designs verwendet. Für diese CPU liegen aus der Vergangenheit Power-Analysen mit VCD-File vor, was sich dazu eignet, die Leistungsabschätzung von Joules zu evaluieren. Neben Standardzellen werden hier außerdem RAM-Makros verwendet.

3 Prototype Design

In diesem Abschnitt wird die Funktion `prototype_design` von Joules evaluiert, mit der es möglich ist, in kurzer Zeit einen Prototypen für ein Design anzufertigen. Abschnitt 3.1 erläutert die Funktion und beschreibt den Aufbau des Flows. Daraufhin wird in Abschnitt 3.2 die Qualität des Prototypen mit unterschiedlichen Optimierungsstufen evaluiert. Abschließend wird die Ressourcennutzung in Abschnitt 3.3 aufgezeigt.

3.1 Aufsetzen des Flows

Joules verfügt mit dem Befehl `prototype_design` über die Funktion, eine physikalische Synthese mit verschiedenen Abstraktionsebenen durchzuführen. Über den Parameter `effort_level` kann beeinflusst werden, wie stark während der Synthese optimiert wird. Dies hat Auswirkung auf die Qualität des Ergebnisses und die Laufzeit. Ein höheres Effort-Level soll ein besseres Ergebnis liefern, aber auch eine längere Laufzeit mit sich bringen. Ein detaillierter Vergleich dieser Optionen ist in Abschnitt 3.2 zu finden. Mithilfe dieses Befehls kann ein logischer und ein physikalischer Prototyp erstellt werden. Im Gegensatz zum physikalischen Prototypen werden beim logischen Prototypen keine Zellen platziert, was für eine geringere Genauigkeit sorgt (Cadence, 2023b). In dieser Arbeit wird der Fokus auf die physikalische Implementierung gelegt, da nur dort die vollständigen PPAC-Metriken zur Verfügung stehen. Die sichtbaren Metriken im logischen Flow sind begrenzt, für erste Flächen- und Power-Abschätzung kann diese aber reichen.

Um eine Vorhersage des Designs zu erhalten, müssen vorher einige Schritte durchgeführt werden. Mit der Funktion `prot_design_physical` in Listing 3.1 können diese Schritte durchgeführt werden. Dieser Prototyp kann daraufhin in der Benutzeroberfläche analysiert werden.

```

1 proc prot_design_physical {effort db_dir} {
2     read_mmmc $mmmc_file
3     read_physical -lef $libs
4     set rtl_files [glob "${rtl_path}/*.v"]
5     read_hdl -language sv $rtl_files
6     elaborate $CIRCUIT
7     init_design
8     read_def $def
9
10    set TIME_start [clock seconds]
11    prototype_design -synthesis_type physical -effort $effort
12    set TIME_taken [expr ([clock seconds] - $TIME_start)]
13    puts "Prototype design, physical $effort effort took $TIME_taken seconds"
14    write_db "${db_dir}/phys_${effort}.db"
15 }
```

Listing 3.1: Die Funktion `prot_design_physical`

Diese Funktion liest das Design und führt das Prototyping mit hohem oder niedrigem Aufwand durch. Die benötigte Zeit des Prototyping wird gemessen und ausgegeben. In den Variablen `mmmc_file`, `rtl_path` und `libs` sind die Pfade zu den entsprechenden Dateien gespeichert. Die variable `CIRCUIT` enthält den Namen des Designs.

Hier wird zunächst ein Multi-Mode-Multi-Corner-Setup (MMMC) durchgeführt. Dieses Setup dient dazu ein Design unter verschiedenen Bedingungen zu analysieren. Dabei können mehrere *Constraint Modes*, wie beispielsweise unterschiedliche Betriebszustände (Test/Normalbetrieb) mit verschiedenen *Delay Corners*, wie beispielsweise unterschiedliche Spannungs- und Temperaturvarianten kombiniert werden, um die *Analysis Views* zu bilden. Für das Erstellen eines physikalischen Prototypen wird ein MMMC-Setup benötigt, da dies die Grundlage für die nachfolgenden Optimierungen während des erstellen des Prototypen verwendet wird (Cadence, 2025).

Danach werden die physikalischen Daten eingelesen. Darin enthalten sind Layer-, Via-, Placement- und Makro-Definitionen. Das Laden der physikalischen Daten sorgt dafür, dass Zellen korrekt platziert werden können, sodass beim Routing die Designregeln eingehalten werden und im Anschluss eine physikalische Verifikation erfolgen kann (Cadence, 2017, S. 10). Erst dann kann die beschriebene Hardware in Form einer Hardwarebeschreibungssprache wie Verilog- oder VHDL-Dateien eingelesen werden und das Design wird initialisiert. In diesen Dateien ist die Logik und Funktionalität der Schaltung definiert und Makros instanziert. Vor der Initialisierung des Designs mit dem Befehl `init_design`, wird der Befehl `elaborate` verwendet. Beim Elaborate-Schritt erstellt das Tool unter anderem Datenstrukturen, verbindet Register und führt HDL-Optimierungen durch (Cadence, 2025, S. 137).

Für ein realistisches Prototyping müssen nun noch die Daten des Design-Plannings über ein def-file geladen werden. Dieses enthält physikalische Parameter wie die Größe (äußerer Rand) des Chips. Außerdem sind hier Makro-Instanzen definiert und platziert. Beim Nick-Asic sind dies beispielsweise die ADCs, bei der ca-53-CPU die RAMs. Diese Instanzen können während der Synthese nicht bewegt werden. Auch das Power-Routing sollte für eine Abschätzung der Congestion bereits enthalten sein.

Mit dem initialisierten Design wird nun durch den Befehl `prototype_design` ein Prototyp erstellt. Die benötigte Zeit für den Prozess wird in der Variable `TIME_taken` gespeichert und ausgegeben. Die resultierende Datenbasis wird anschließend gespeichert.

3.2 Vergleich Low- und High-Effort am Nick-Asic

Die in den folgenden Abschnitten beschriebenen Tests dienen dazu, die Qualität des Prototypen aus Joules zu beurteilen. In beiden Tests wird die Komplexität des Designs künstlich erschwert, um den Unterschied zwischen High- und Low-Effort beurteilen zu können und um gezielt Routing-Congestion sowie schwer erreichbares Timing zu provozieren. Bei den hier angegebenen Werten für die Congestion handelt es sich um den höchsten Congestion-Hotspot innerhalb des Designs. Die Congestion kann dem Tool aus der Grafischen Benutzeroberfläche entnommen werden. Die Funktion `getUtil` in Listing 3.2 berechnet die Utilization bzw. Density.

Die Density berechnet sich aus der Fläche der Summe aller Instanzen geteilt durch die verfügbare Fläche. Die Fläche jeder Instanz sowie die Grenze des Designs kann von Joules mithilfe von `get_db` abgefragt werden.

```

1 proc getUtil {} {
2     set area 0
3     foreach inst [get_db insts *] {
4         set inst_area [get_db $inst .area]
5         set area [expr $area + $inst_area]
6     }
7     set boundary [get_db current_design .boundary.area]
8     set util [expr $area / $boundary]
9     return $util
10 }
```

Listing 3.2: Die Funktion `getUtil`

3.2.1 Begrenzung der Routing-Ressourcen

Standardmäßig sind im Nick-Asic 5 Metalllagen zum Verdrahten des Designs vorhanden. Da es sich beim Nick-Asic um ein einfaches Design handelt, sind für den Test die Metalllagen reduziert worden, bis nur noch zwei Lagen zum Verdrahten übrig bleiben. Somit werden die Routing-Ressourcen limitiert, was zu steigenden Anforderungen an den Routing-Algorithmus führt. Es ist zu erwarten, dass die Congestion mit Abnahme der Metalllagen steigt, bis das Design nicht mehr verdrahtbar ist. Die Ergebnisse dieses Tests sind in Tabelle 3.1 zu sehen.

Schichten	Low Effort		High Effort	
	Congestion	Density	Congestion	Density
5	0	47.65	0	46.0
4	0.45	48.17	0.17	46.11
3	3.86	48.15	2.34	46.11
2	77720	47.93	12027.30	46.04

Tabelle 3.1: Ergebnisse des Prototypen für unterschiedliche Anzahl an Metalllagen mit High- und Low-Effort

Der Test bestätigt die Erwartung. Bei den Durchläufen mit allen 5 Metalllagen ist die Congestion nicht auffällig, was die gute Routbarkeit des Designs bestätigt. Bei 4 und 3 Metalllagen steigt die Congestion langsam an, bleibt aber in einem Bereich, der für die physikalische Implementierung noch akzeptabel wäre. Erst bei nur 2 Metalllagen ist ersichtlich, dass das Design nicht routbar ist. Die Density bleibt über die Durchläufe nahezu konstant. Dies war zu erwarten, da zusätzliche Zellen die Routing-Probleme nur verschärfen würden.

Die Layouts nach der Implementierung der in der Tabelle zu sehenden Durchläufe sind in Abbildung 3.1 zu sehen.

Durch die Density-Abbildungen ist zu sehen, dass Joules bei weniger Ressourcen versucht, die Zellen an dem Rand des Designs zu platzieren, um in der Mitte mehr Routing-Ressourcen zu erhalten. Mit sinkender Anzahl an Metalllagen ist auch optisch das Ansteigen der Congestion in beiden Modi zu sehen. Im Low-Effort sind es jedoch sichtbar mehr und verteilt über das gesamte Design, während die Anzahl der Hotspots im High-Effort geringer und nur im Zentrum des Designs zu sehen sind. Es ist zu beobachten, dass der High-Effort durch alle Tests hinweg bessere Ergebnisse erzielt,

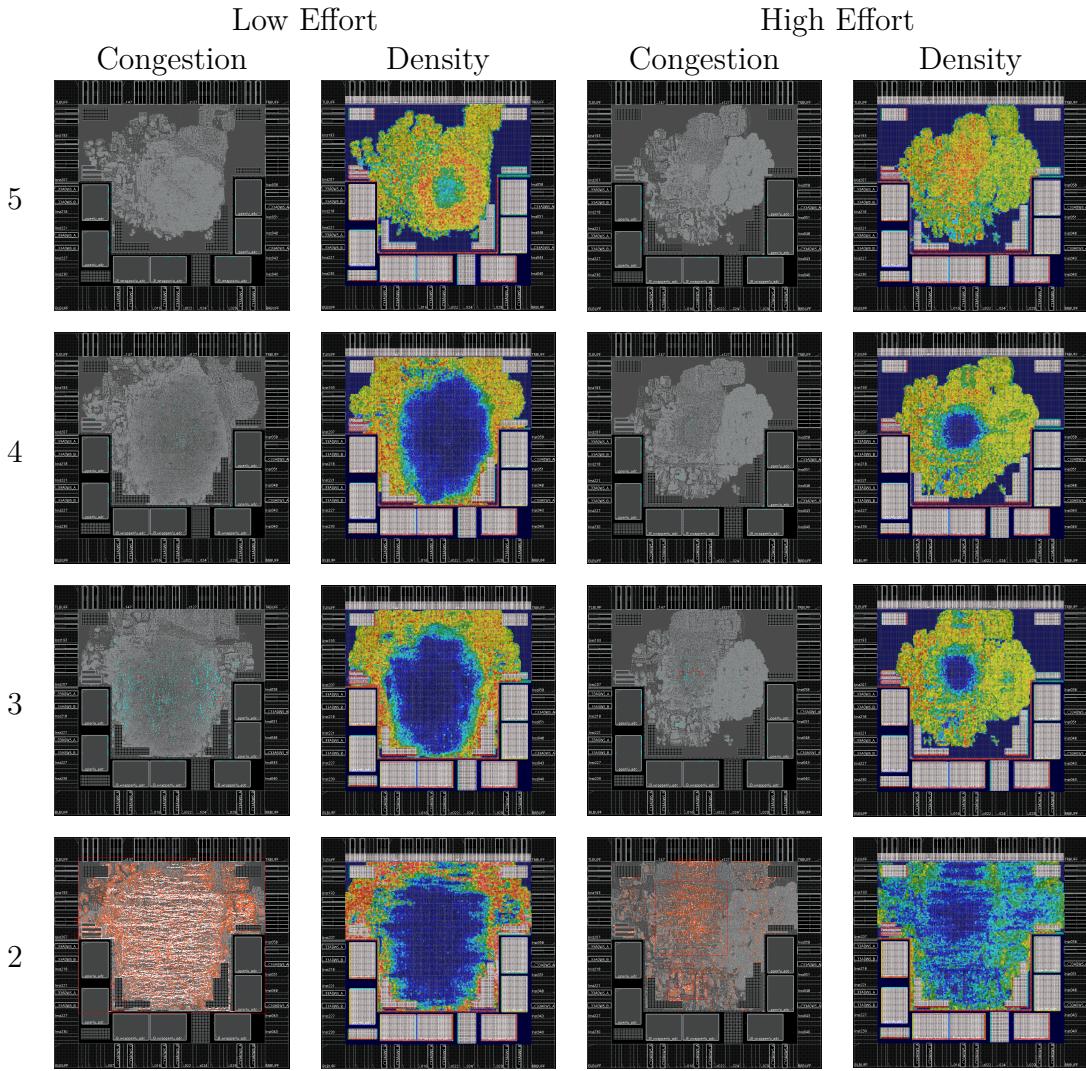


Abbildung 3.1: Layouts für verschiedene Anzahl an Metalllagen

es ist weniger Congestion zu sehen. Besonders bei nur 2 Lagen ist zu sehen, dass durch eine optimiertere Platzierung der Zellen beim High-Effort weniger Routing-Congestion erreicht wird.

3.2.2 Erhöhung der Timing-Anforderungen

Eine andere Variable, welche verändert werden kann, ist die Taktfrequenz. Die originale Periodendauer der Clock ist 16 ns, daraus folgt eine Frequenz von 6.25 MHz. Durch ein sukzessives Verringern der Periodendauer in 2-ns-Schritten bis auf 4 ns steigt die Frequenz bis auf 25 MHz. Dies simuliert ein Design, mit höheren Anforderungen an die Clock. Zu erwarten ist, dass die Density steigt, da größere und schnellere Zellen mit erhöhter Treiberstärke benutzt werden, um den gestiegenen Timing-Anforderungen gerecht zu werden. Als Metrik wird hier außerdem der *Slack* des kritischsten Pfades verwendet. Slack bezeichnet die Differenz zwischen der erreichten Ankunftszeit und der erforderlichen Ankunftszeit eines Signals. Bei einem Positiven Slack sind die Anforderungen erfüllt, ein Negativer Slack stellt eine so genannte *Timing-Violation* dar. Als Kritisches Pfade wird der Pfad mit geringstem Slack verwendet. Es ist zu erwarten, dass der Slack mit steigender Clock-Frequenz abnimmt. Tabelle 3.2 zeigt die Ergebnisse dieses

Tests, Abbildung 3.2 visualisiert diese Ergebnisse.

Periode	Frequenz	Low Effort		High Effort	
		Density	Slack	Density	Slack
16 ns	6.25 MHz	48.06	0.278	45.99	1.615
14 ns	7.14 MHz	47.87	0.328	46.00	1.343
12 ns	8.33 MHz	48.08	0.281	46.07	1.073
10 ns	10.0 MHz	48.48	0.183	46.39	0.413
8 ns	12.5 MHz	49.09	0.193	46.70	0.362
6 ns	16.7 MHz	49.64	-0.506	47.46	-0.450
4 ns	25.0 MHz	50.64	-1.450	48.83	-1.437

Tabelle 3.2: Ergebnisse des Prototypen für unterschiedliche Clock-Frequenzen mit High- und Low-Effort

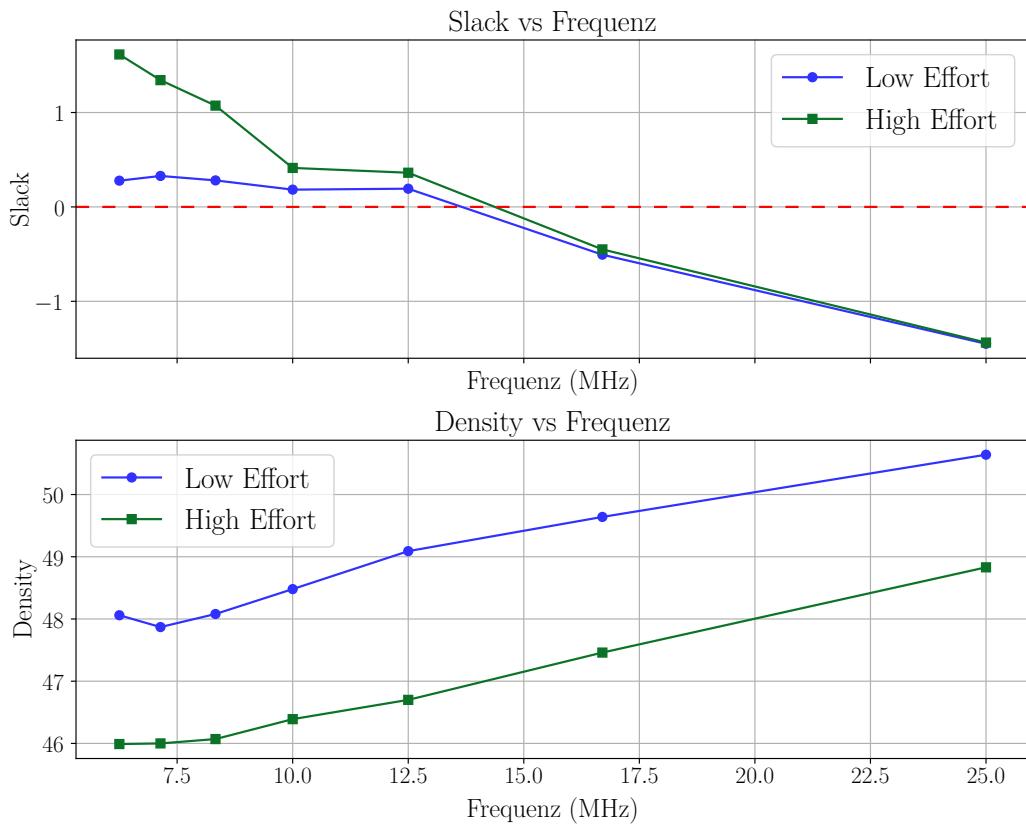


Abbildung 3.2: Visualisierung der Ergebnisse für unterschiedliche Clock-Frequenzen mit High- und Low-Effort

Auch hier erfüllen sich die Erwartungen. Die Variation der Clock-Frequenzen zeigt deutliche Trends in Bezug auf Slack und Density. Mit abnehmender Periodendauer, also steigender Frequenz verschärfen sich die Anforderungen des Designs. Es ist klar zu erkennen, dass der minimale Slack mit steigender Frequenz kontinuierlich abnimmt. Während bei niedrigen Frequenzen noch Positive Slack-Werte erreicht werden, sind ab 16.7 MHz Timing-Violations in beiden effort-leveln zu sehen. Die High-Effort Optimierung zeigt dabei durchweg bessere Slack-Werte als die Low-Effort Optimierung.

Auch die Entwicklungen der Density in Abhängigkeit zur Frequenz erfüllen die Erwartungen. Insgesamt ist die Density in der High-Effort Optimierung geringer, was für

eine Effektivere Nutzung der Zellen spricht.

3.3 Ressourcennutzung

Die Erstellung eines Prototypen wird hier als Grundlage genommen, um den Ressourcenverbrauch von Joules zu bewerten. Der Ressourcenverbrauch kann anhand von zwei Parametern bewertet werden: Der Benötigten Zeit und dem Benötigten Speicher. Ein Arbeitsschritt kann in der Praxis statt auf einer CPU auf mehreren Kernen ausgeführt werden, um den Prozess zu beschleunigen. Der Einfachheit halber wird im Folgenden die Summe aller CPU-Zeiten als *Laufzeit* beschrieben. Der von der Applikation verwendete Speicher variiert im Laufe der Durchführung, daher wird hier der zu einem Zeitpunkt maximal genutzte Speicher aufgeführt. In Tabelle 3.3 sind diese Beiden Parameter für logische und physische Prototypen mit hoher und geringer Optimierung aufgetragen. Die Laufzeit und Speichernutzung von Joules werden zwischen den verschiedenen Arten des Prototypen (logisch und physisch) und den verschiedenen Optimierungsstufen (gering und hoch) verglichen.

	logisch-low	logisch-high	physisch-low	physisch-high
Laufzeit	28:40	35:36	34:57	41:14
Speicher	152.4 GB	159.5 GB	149.1 GB	155.9 GB

Tabelle 3.3: Ressourcenverbrauch der Unterschiedlichen Prototypen und Innovus

Die höhere Optimierung beim Erstellen des Prototypen erhöht die Laufzeit um jeweils 6 bis 7 Stunden. Verglichen mit dem physischen Flow ist der logische Flow insgesamt etwa 6 Stunden schneller. Die zeitlichen Unterschiede zwischen den Optimierungsstufen und Typen sind zu erwarten, bei höherer Optimierung werden, wie in den vorherigen Abschnitten erläutert, bessere Ergebnisse erreicht und der Physikalische Flow übernimmt mehr Aufgaben als der logische Flow, wie beispielsweise das Platzieren der Zellen. Die Unterschiede in der Speichernutzung sind gering und können vernachlässigt werden.

3.4 Vorteile für die Front-End Entwicklung

Für einen Front-End-Designer kann die Nutzung von Joules vorteilhaft sein. Wie in den vorangegangenen Abschnitten erläutert, bietet Joules die Möglichkeit, im Designprozess einen Prototypen anzufertigen. Somit kann ein Designer über die PPAC-Metriken seine Logik beurteilen.

Außerdem kann der Designer eine Momentaufnahme (Snapshot) seines aktuellen Fortschrittes in eine im Vorhinein angelegte Datenbasis zu speichern.

Eine solche Datenbasis wird durch den Befehl

```
track_ppac -config_id $id -create
```

erstellt. Ein Snapshot kann dann nach dem Erstellen eines Prototypen mit

```
track_ppac -config_id $id -capture -tag "aktueller status"
```

angelegt werden. Die id selbst ist hier frei wählbar.

Somit ist es möglich, die PPAC-Metriken im Verlauf der Entwicklung des Designs zu betrachten und so die Auswirkungen von RTL-Änderungen auf das Design zu beurteilen.

Außerdem hat der Designer durch das RTL Debug Assistance System (RDAS) die Möglichkeit, das Timing, die Congestion und die Struktur des Designs zu analysieren und den Grund für Violations im RTL zu finden.

4 Nachvollziehen der vorhandenen Ergebnisse mit PTPX und Voltus

Aus der Vergangenheit ist für die ca-53-cpu eine VCD-Datei zur Simulation vorhanden. Mithilfe dieser kann eine dynamische Power-Analyse durchgeführt werden. Zum Zeitpunkt der Entwicklung des Designs wurde zur Power-Analyse PrimeTime zur Durchführung dieser Analyse verwendet. Da zur aktuellen Zeit Voltus das Signoff-Tool ist, sollen die Ergebnisse der beiden Tools verglichen werden.

Die beiden Tools Voltus von Cadence und PrimeTime von Synopsys sind Tools, welche anhand der Netzliste und eines SPEFs (Standard Parasitic Exchange Format) die Leistungsaufnahme eines Designs berechnen können. Das SPEF enthält die relevanten Kapazitäten und Widerstände aller Verbindungen des Designs nach der Implementierung und wird benötigt, um die Leistungsaufnahme möglichst akkurat zu berechnen. Um zunächst die beiden Tools selbst zu vergleichen, wird eine alte Netzliste verwendet, welche in der Vergangenheit schon mit PTPX verwendet wurde. Es ist mit beiden Tools sowohl möglich eine statische Analyse durchzuführen als auch eine dynamische Analyse, bei der eine Aktivitätsdatei (VCD) geladen wird, welche die Aktivitäten der einzelnen Zellen enthält. Zusätzlich werden die Constraints (SDC) eingelesen, um zusätzliche Vorgaben, wie beispielsweise konstante Signale an einigen Ports, welche nur zur Testzwecken benötigt werden, mitzuteilen. Ziel ist es, den Flow zur Berechnung der Power zu verifizieren, um im nachfolgenden Abschnitt 5 die Leistungsaufnahme des Joules-Prototypen mit einer vollständigen Implementierung in Innovus zu vergleichen. Für PrimeTime ist bereits eine Analyse aufgesetzt, welche verwendet werden kann. Für Voltus wurde eine ähnliche Analyse durchgeführt. Nach dem Aufsetzen der beiden Tools sind in den Ergebnissen der Berechnungen starke Unterschiede zu sehen. Im Abschnitt 4.1 sind die Schritte dokumentiert, welche unternommen wurden, um die Unterschiede zwischen den beiden Tools zu identifizieren und zu beseitigen. Daraufhin folgen in Abschnitt 4.2 die Ergebnisse dieser Maßnahmen.

4.1 Methodik zur Ursachenanalyse

Nach dem Aufsetzen der beiden Tools unterscheiden sich die berechneten Leistungswerte dieser stark, wie in Tabelle 4.1 zu sehen ist.

Metrik	Voltus [mW]	PTPX [mW]	Abweichung
P_{int}	224.9	74.8	200.7 %
P_{sw}	115.4	75.8	52.3 %
P_{leak}	76.6	78.1	1.9 %
P_{ges}	417.0	228.7	82.3 %

Tabelle 4.1: Unterschied der berechneten Leistungswerte von Voltus und PTPX

Hier wurde, wie im weiteren Verlauf dieses Abschnittes, die Abweichung der Tools wie in Gleichung 4.1 berechnet. PTPX wird also als Referenz betrachtet, da dieses Tool zur Zeit der Implementierung des Designs verwendet wurde.

$$\text{Abweichung} = \left| \frac{P_{\text{PTPX}} - P_{\text{Voltus}}}{P_{\text{PTPX}}} \cdot 100 \right| \quad (4.1)$$

Besonders die Komponenten der Dynamischen Leistung (P_{int} und P_{sw}) unterscheiden sich deutlich. Aus Abschnitt 2.2.1 ist bekannt, dass die Komponenten der dynamischen Leistung im Gegensatz zur statischen Leistung (P_{leak}) von der Schaltaktivität der Zelle abhängt. Daher liegt die Vermutung nahe, dass die Ursachen der Unterschiede dort liegen.

Um die Berechnung der internen Leistung nachvollziehen zu können, ist zunächst in Abschnitt 4.1.1 beschrieben, wie diese in der Praxis berechnet werden kann. Daraufhin wird in Abschnitt 4.1.2 eine Berechnung mit den Werten von Voltus und anschließend in Abschnitt 4.1.3 mit den Werten von PTPX. Da die Unterschiede dieser Zelle auf die Taktfrequenz zurückzuführen sind, untersucht der Abschnitt 4.1.5 die Auswirkungen der Constraints auf die Ergebnisse beider Tools.

Bei der Analyse einer weiteren Zelle in Abschnitt 4.1.6 deuten die Ergebnisse darauf hin, dass einige Zellen nicht korrekt annotiert werden. Diese Annotation lässt sich auf Fehler im Mapfile zurückführen, welche in Abschnitt 4.1.7 korrigiert werden.

Da weiterhin Unterschiede bestehen, wird abschließend in Abschnitt 4.1.8 untersucht, ob das Zeitfenster beim Einlesen des VCDs Auswirkung auf die Ergebnisse hat.

4.1.1 Berechnung der Internal Power

Aus Abschnitt 2.2.1 ist bekannt, dass die Interne Leistung aus der Summe der internen Kurzschlussspannung und der internen Schaltspannung besteht. In der Praxis wird die interne Leistung über eine Lookup-Tabelle mit Abhängigkeiten von Eingangsflanke (Slew-Rate) und Lastkapazität enthalten, da diese den internen Kurzschlussstrom bestimmen. Voltus unterstützt die Einstellung eines festen Slew-Werts. Damit kann mithilfe der Gleichung 4.2 zwischen zwei Punkten linear interpoliert werden.

$$f(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \cdot (x - x_1) \quad (4.2)$$

Die lineare Interpolation ist ein Hilfsmittel, mit dem die Rise- und Fall-Power berechnet werden kann. Wie Voltus und PTPX zwischen beiden Werten interpolieren, ist nicht bekannt, daher ist eine Abweichung in den Ergebnissen zwischen den berechneten Werten und der Ausgabe der Tools plausibel. Aus dem Durchschnitt dieser beiden Werte ergibt sich die Gesamtenergie, wie in Gleichung 4.3 zu sehen ist.

$$E_{\text{int}} = \frac{P_{\text{rise}} + P_{\text{fall}}}{2} \quad (4.3)$$

Die interne Leistung eines Pfades berechnet sich nun entsprechend der Gleichung 4.4, wobei T_D die Toggle-Rate ist.

$$P_{\text{internal}} = T_D \cdot E_{\text{int}} \quad (4.4)$$

In den folgenden Abschnitten wird die Interne Leistung in Voltus und Primetime berechnet. Dies wird exemplarisch für die Zelle `cts_buf_13227436055` durchgeführt. Diese Zelle eignet sich für eine manuelle Analyse, da es sich hier um einen Clock-Buffer handelt. Vor einem solchen Buffer befindet sich keine Logik, welche die Toggle-Rate beeinflussen kann. Wie in Abbildung 4.1 zu sehen, befinden sich in der Schaltung vor der Zelle nur weitere Clock-Buffer, welche letztendlich vom Clock-Pin getrieben werden.

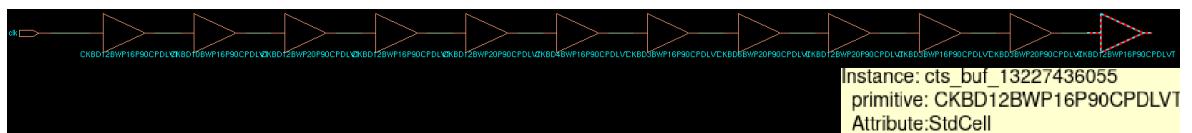


Abbildung 4.1: Der Pfad vom Clock-Eingang zum Clock-Buffer `cts_buf_13227436055`

4.1.2 Exemplarische Berechnung der internen Leistung in Voltus

Um die Berechnung der internen Leistung in Voltus nachvollziehen zu können, wird der Wert für die Slew-Rate auf den von Voltus verwendeten Wert (0.0726576) festgesetzt. Für die Kapazität ist dies nicht möglich. Deshalb wird über die Lineare Interpolation, wie in Gleichung 4.2 die Rise-Power (4.5) und die Fall-power (4.6) berechnet werden.

$$P_{\text{rise}} = 0.0140322 + \frac{0.0106516 - 0.0140322}{0.0426555 - 0.000320476} \cdot (0.01063456 - 0.000320476) = 13.2086 \text{ fJ} \quad (4.5)$$

$$P_{\text{fall}} = 0.0168469 + \frac{0.0168469 - 0.0168469}{0.0426555 - 0.000320476} \cdot (0.01063456 - 0.000320476) = 16.8469 \text{ fJ} \quad (4.6)$$

Aus dem Mittelwert aus Rise- und Fall-Power ergibt sich die Gesamtenergie von 15.03 fJ in Gleichung 4.7. Dieser Wert ist nicht identisch mit dem Wert, welcher von Voltus berechnet wird (14.6 fJ). Diese geringe Abweichung von ca. 7 % lässt sich auf die Art der Interpolation zurückführen, welche in Voltus nicht bekannt ist. Für das Nachvollziehen der Berechnung reicht das Ergebnis allerdings aus.

$$E = \frac{013.2086 \text{ fJ} + 16.8469 \text{ fJ}}{2} = 15.03 \text{ fJ} \quad (4.7)$$

Mithilfe der Energie kann die interne Leistung wie in Gleichung 4.8 berechnet werden. Hierzu kann die Toggle-Rate aus Voltus abgefragt werden. Diese beträgt 2.6.

$$P_{\text{internal}} = T_D \cdot E = 2.6 \text{ s}^{-1} \cdot 15.03 \text{ fJ} = 39.1 \mu\text{W} \quad (4.8)$$

Die manuell berechnete Leistung deckt sich mit dem Wert aus Voltus.

4.1.3 Exemplarische Berechnung der internen Leistung in PrimeTime

Der von PTPX berechnete Wert für die interne Leistung der betrachteten Zelle ist $28.8 \mu\text{W}$ und unterscheidet sich somit um ca. 27 % von dem Ergebnis in Voltus. PTPX unterstützt weniger Möglichkeiten zum Nachvollziehen der berechneten Leistung, weshalb einige Werte von Voltus verwendet werden, um die Berechnung von PTPX nachzuvollziehen. Die Toggle-Rate kann jedoch aus PTPX abgefragt werden und beträgt 1.98. Verwendet man Gleichung 4.4 nun mit der Toggle-Rate aus PTPX und der Energie aus Voltus, wie in Gleichung 4.9, deckt sich das Ergebnis mit dem von PTPX ausgegebenen Wert.

$$P_{\text{internal}} = T_{D_{\text{PTPX}}} \cdot E_{\text{int}_{\text{voltus}}} = 1.98 \cdot 10^9 \cdot 14.6 \text{ fJ} = 28.9 \mu\text{W} \quad (4.9)$$

PTPX berechnet die interne Leistung also analog zur Berechnung in Voltus, wie in Abschnitt 4.1.2. Die beiden Tools unterscheiden sich in der Berechnung Zelle nur in der verwendeten Toggle-Rate.

4.1.4 Berechnung der Toggle-Rate

Aus den vorherigen beiden Abschnitten wird klar, dass die Toggle-Raten der beiden Tools nicht korrelieren. Im Folgenden wird die Toggle-Rate manuell mit den Werten aus den Constraints durchgeführt. Zunächst wird die Frequenz aus der im SDC beschriebenen Periode $T = 0.769\,231 \text{ ns}$ berechnet.

$$f = \frac{1}{T} = \frac{1}{0.769\,231 \text{ ns}} = 1.299\,999\,61 \text{ GHz} \quad (4.10)$$

Mithilfe der berechneten Frequenz ist die Toggle-Rate T_D wie in Gleichung 4.11 zu berechnen.

$$T_D = 2 \cdot f = 2 \cdot 1.299\,999\,61 \text{ GHz} = 2.599\,999\,22 \text{ ns}^{-1} \approx 2.6 \text{ ns}^{-1} \quad (4.11)$$

Somit ist die berechnete Toggle-Rate auch die Toggle-Rate, welche in Voltus verwendet wird.

Eine Veränderung der Clock im SDC bei PTPX scheint diese Rate nicht zu verändern.

Aus PTPX ist abzulesen, dass die Toggle-Rate in PTPX nicht mit der berechneten Toggle-Rate übereinstimmt. Dies ist auch für die erste Zelle des Clock-Pfades (`clk_IN_PORT_BUF`) der Fall. Es ist also kein Problem, welches durch die Propagierung des Signals zustande kommt. In dem VCD ist mithilfe eines Waveform-Viewers zu erkennen, dass die Clock-Frequenz innerhalb des VCDs 1 GHz ist (vgl. Abbildung 4.2).

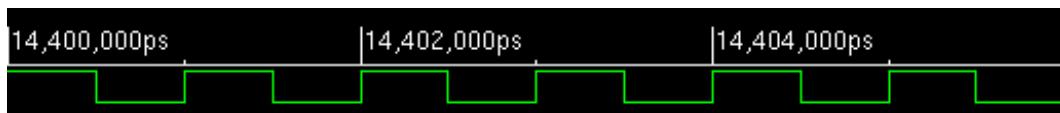


Abbildung 4.2: Das Clock-Signal des VCDs mit der Frequenz 1 GHz in Simvision

Das verwendete VCD zur Simulation verwendet eine falsche Clock-Frequenz. Da es nicht möglich ist, sie im VCD zu ändern, wird im folgenden Abschnitt die Clock in den Constraints auf 1 GHz angepasst, um unterschiedliche Interpretationen von SDC und VCD zwischen den Tools zu vermeiden.

4.1.5 Anpassen der Constraints

Die Clock im VCD läuft mit der Frequenz 1 GHz. PTPX verwendet die Clock-Frequenz aus dem SDC-File, während Voltus sie direkt aus dem VCD übernimmt. Zur Angleichung beider Tools wird die Frequenz im SDC entsprechend dem VCD auf 1 GHz gesetzt. Damit ist die Simulation weniger realitätsnah, sollte jedoch die Abweichung der Tools verringern. In Tabelle 4.2 ist an dem vorher berechneten Beispiel des Clock-Buffers und dem ersten Clock-Buffer zu sehen, dass sich beide Tools angenähert haben.

Zelle	1.3 GHz [μW]			1 GHz [μW]		
	Voltus	PTPX	Abw.	Voltus	PTPX	Abw.
cts_buf_13227436055	32.4	25.2	28.6 %	24.9	25.2	1.2 %
clk_IN_PORT_BUF	21.2	15.3	38.6 %	16.3	15.3	6.5 %

Tabelle 4.2: Auflistung von P_{int} beider Tools mit unterschiedlichen Frequenzen im SDC für beide Clock-Buffer

Eine Auflistung aller Werte ist in Tabelle 4.3 zu sehen. Das bestätigt die Annäherung der Tools, allerdings sind vor allem bei der Internal-Power weiterhin große Unterschiede zu sehen.

Metrik	1.3 GHz[mW]			1 GHz[mW]		
	Voltus	PTPX	Abw.	Voltus	PTPX	Abw.
P_{int}	224.9	74.8	200.7 %	125.8	74.2	69.6 %
P_{sw}	115.4	75.8	52.3 %	84.2	74.8	12.6 %
P_{leak}	76.65	78.1	1.9 %	76.5	78.1	2.0 %
P_{ges}	417.0	228.7	82.3 %	288.04	227.1	26.8 %

Tabelle 4.3: Auflistung der Gesamtleistungen beider Tools nach Frequenz im SDC

4.1.6 Toggle-Rate innerhalb eines Logik-Pfades

Die Toggle-Rate von Clock-Zellen wurde durch das Setzen der Clock auf 1 GHz angeglichen. Da die Leistungswerte der Tools weiterhin unterscheiden, werden im Folgenden die Toggle-Rate einiger Zellen in einem Logikpfad verglichen. Als Beispiel wird der Pfad zur Zelle u_ca53_noram/u_ca53dpu/HFSBUF_16478 genommen. Verglichen werden nun die Werte der ersten Clock-Gating-Zelle. Die Toggle-Rate am Ausgang dieser ist im Normalbetrieb abhängig von der Toggle-Rate am Clock-Pin sowie der Toggle-Rate am Enable-Pin. Wie in Tabelle 4.4 zu sehen ist, unterscheiden sich schon die Ausgangswerte dieser Zelle.

Tool	Q (Ausgang)	CP	E
Voltus	$1.05 \cdot 10^9$	$2 \cdot 10^9$	$0.1 \cdot 10^9$
PTPX	$1.76 \cdot 10^9$	$2 \cdot 10^9$	$0.27 \cdot 10^9$

Tabelle 4.4: Vergleich der Ein- und Ausgänge der betrachteten Clock-Gating-Zelle

Da die Werte für die Eingangsclock (CP) gleich sind (da es sich um die erste Clock-Gating-Zelle im Pfad handelt, ist die Eingangsclock die im SDC gesetzte Clock), muss der Unterschied durch die verschiedenen Werte des Enable-Pins (E) kommen.

Die Zelle `u_ca53_noram/u_ca53dpu/FPU1_u_dpu_fp_cg/fp_mul_en_reg_0` treibt den Enable-Pin. In PTPX ist ersichtlich, dass der Ausgang dieser Zelle (Q) nicht aus dem VCD stammt, wie in Listing 4.1 zu sehen ist.

```
# PTPX
pt_shell> get_switching_activity -toggle_rate [get_pins
  u_ca53_noram/u_ca53dpu/FPU1_u_dpu_fp_cg/fp_mul_en_reg_0/Q]
=> {"u_ca53_noram/u_ca53dpu/FPU1_u_dpu_fp_cg/fp_mul_en_reg_0/Q"
  0.273267 propagated}
```

Listing 4.1: PTPX-Befehl zum Auslesen der Toggle-Rate

Auch in Voltus scheint die Annotation allerdings nicht funktioniert zu haben. Der Befehl

```
report_annotationed_check
  u_ca53_noram/u_ca53dpu/FPU1_u_dpu_fp_cg/fp_mul_en_reg_0
```

zeigt keinen Annotierten Pfad.

Beim Analysieren des VCDs in einem Waveform-Viewer (Simvision) wird wie in Abbildung 4.3 ersichtlich, dass das Signal insgesamt 112 mal schaltet.

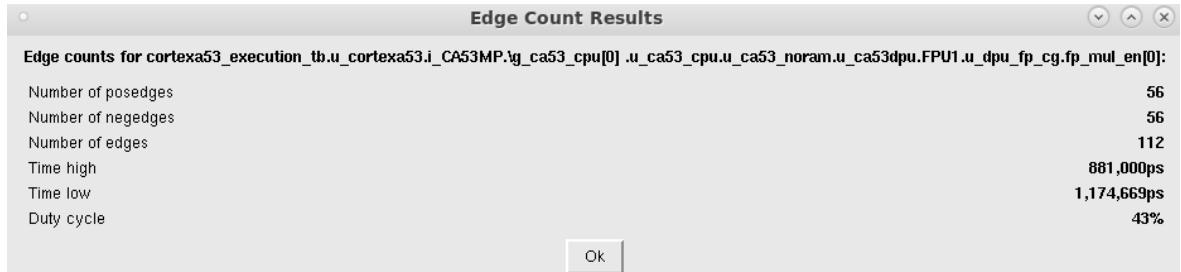


Abbildung 4.3: Ausgabe von Simvision zum Toggle-Count der Zelle `fp_mul_en[0]`

In Gleichung 4.12 ist die Berechnung der Frequenz aus der festgestellten Toggle-Rate zu sehen. Daraus ergibt sich eine erwartete Toggle-Rate von 0.112 ns^{-1} . Bei den 1000 ns handelt es sich um das Zeitfenster im VCD, in der Aktivität stattfindet.

$$T_D = \frac{56 \cdot 2}{1000 \text{ ns}} = 0.112 \text{ ns}^{-1} \quad (4.12)$$

Der Ausgang (Q) ist in Voltus aber $1.05 \cdot 10^9$ und in PTPX $2 \cdot 10^9$. Somit ist klar, dass die Annotation, welche im VCD vorhanden ist, in PTPX und Voltus nicht übernommen wird. Die beiden Tools nehmen somit intern berechnete, unterschiedliche Toggle-Raten an, was zu Unterschieden in den berechneten Leistungswerten führt. Die Ursache dafür wird im nachfolgenden Abschnitt erläutert.

4.1.7 Mapfile

Die Namensgebung von Instanzen kann sich zwischen einer RTL-Netzliste und einer Gate-Level-Netzliste unterscheiden. Ein Mapfile wird verwendet, um die Instanzzuordnung bei geänderten Namen zu gewährleisten. (Cadence, 2025, S. 745). Cadence und Synopsys verwenden unterschiedliche Syntaxen, welche nicht miteinander kompatibel sind. Ein Mapfile lässt sich allerdings verlustfrei zwischen den beiden Tools konvertieren. Im Folgenden Abschnitt wird die Syntax von Synopsys verwendet. Das Mapfile scheint im PTPX nicht zu funktionieren. Nach genauerer Untersuchung fällt auf, dass das verwendete Mapfile fehlerhaft ist. Die Veränderung des Mapfiles ist exemplarisch für die betrachtete Zelle in Listing 4.2 zu sehen.

```

1 # Alt
2 set_rtl_to_gate_name \
3   -rtl u_ca53_noram/u_ca53dpu/FPU1.u_dpu_fp_cg/fp_mul_en[0] \
4   -gate u_ca53_noram/u_ca53dpu/FPU1_u_dpu_fp_cg/fp_mul_en_reg_0/Q \
5 # Neu
6 set_rtl_to_gate_name \
7   -rtl u_ca53_noram/u_ca53dpu/FPU1/u_dpu_fp_cg/fp_mul_en[0] \
8   -gate u_ca53_noram/u_ca53dpu/FPU1_u_dpu_fp_cg/fp_mul_en_reg_0/Q \

```

Listing 4.2: Änderung des Mapfiles

Nach der Änderung des Mapfiles wird die Zelle in beiden Tools richtig erkannt und die Werte für die interne Leistung der beiden Tools nähern sich weiter an, wie in Tabelle 4.5 zu sehen ist. Es bestehen allerdings weiterhin größere Unterschiede in der Internal- und Switching-Power.

Metrik	Voltus [mW]	PTPX [mW]	Abweichung
P_{int}	63.6	89.5	28.9 %
P_{sw}	31.3	92.8	66.3 %
P_{leak}	77.4	78.1	0.9 %
P_{ges}	172.3	260.5	33.8 %

Tabelle 4.5: Leistungswerte nach Änderung des Mapfiles

4.1.8 Angeben des Fensters

Nach weiterer Betrachtung des VCDs fällt auf, dass die Simulation nicht sofort am Anfang des VCDs beginnt. Wie in Abbildung 4.4 am Beispiel des Clock-Signals zu sehen ist, beginnt die Simulation im VCD-File nicht am Anfang der Datei, sondern startet erst nach 14400 Nanosekunden.

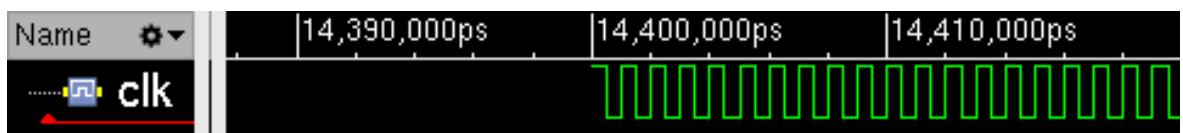


Abbildung 4.4: Beginn des Clock-Signals in Simvision

Eine Vermutung ist, dass die Tools dieses Zeitfenster unterschiedlich erkennen. Um dies auszuschließen, wird das Einlesen des VCD-Files mit einem expliziten Zeitfenster

(Startpunkt bis Ende) angegeben. In Listing 4.3 sind die Befehle zum Einlesen der VCDs für beide Tools zu sehen. In der Variable `VCD` ist der Pfad zur VCD-Datei gespeichert. Die Variable `VCD_TOP` enthält die Top-Hierarchie des VCDs. Die Angabe des Zeitfensters hat keinen Einfluss auf die endgültigen Leistungswerte beider Tools, was darauf schließen lässt, dass die Tools das effektive Zeitfenster der Simulation automatisch finden.

```
1 # Voltus:
2 read_activity_file -format VCD -start 14400000ps -end 15399000ps $VCD
   -scope $VCD_TOP
3 # PTPX:
4 read_vcd $VCD -strip_path $VCD_TOP -rtl -time {14400 15399}
```

Listing 4.3: Befehle zum Einlesen der VCDs in TCL

4.2 Ergebnisse der Maßnahmen

Das Anpassen der Frequenz in den Constraints und die Korrektur des Mapfiles führen zu einer Annäherung der Ergebnisse von PrimeTime und Voltus, eine vollständige Korrelation konnte allerdings im zeitlichen Rahmen dieser Arbeit nicht vollzogen werden. Durch die implementierten Anpassungen konnte die ursprüngliche Abweichung von etwa 82 % auf rund 33 % reduziert werden.

Da Voltus zum aktuellen Zeitpunkt als Signoff-Tool eingesetzt wird, und somit die komplette Implementierung und Signoff-Analyse mit Cadence-Tools stattfindet, basieren alle nachfolgenden Bewertungen auf den mit Voltus gewonnenen Ergebnissen.

5 Implementierung mit traditioneller Software

Um die Ergebnisse des Prototypen aus Joules einordnen zu können, wird im folgenden Kapitel die Implementierung in einer traditionellen Software beschrieben. In den folgenden Abschnitten werden die Ergebnisse der Implementierung verglichen. Der Flow für Joules und Innovus wird mit einer Taktfrequenz von 1 GHz in den Constraints durchgeführt, um Konflikte zwischen VCD und SDC zu umgehen. Die Schritte, welche für die Implementierung in Innovus durchlaufen werden, sind in Abschnitt 5.1 erläutert. Diese Implementierung dient als Grundlage zum Vergleich mit dem Joules-Prototypen. Ein Vergleich der Power-Werte ist in Abschnitt 5.2 zu finden.

5.1 Schritte zur Implementierung

In diesem Abschnitt wird die Implementierung der ca-53-CPU durchgeführt. Bei den hier besprochenen Schritten handelt es sich um einen vereinfachten Prozess. Da Innovus keine Hardwarebeschreibungssprache direkt Einlesen kann, wird ein Prototyp mit Joules erstellt. Dieser Prototyp dient als Ausgangszustand und wird im Innovus geladen. Zunächst wird das Design geladen und die darin enthaltenen Zellen platziert. Dies geschieht in Innovus mit dem Befehl `place_opt_design`.

Dem Clock-Signal wird bei der Implementierung besondere Aufmerksamkeit zugewandt. Um ein sauberes Clock-Signal zu erhalten, wird für dieses Signal mit dem Befehl `clock_opt_design` ein sogenannter Clock-Tree aufgebaut, wie in Abschnitt 2.1.3 erläutert. Das Ergebnis der Clock-Tree-Synthese in Innovus ist in Abbildung 5.1 zu sehen. Bei den grünen Dreiecken handelt es sich um Buffer, bei den türkisen Kreisen handelt es sich um Clock-Gating-Zellen. Die roten Vierecke an den Blättern des Clock-Trees sind *sinks*, beispielsweise die eigentlichen Register der CPU.

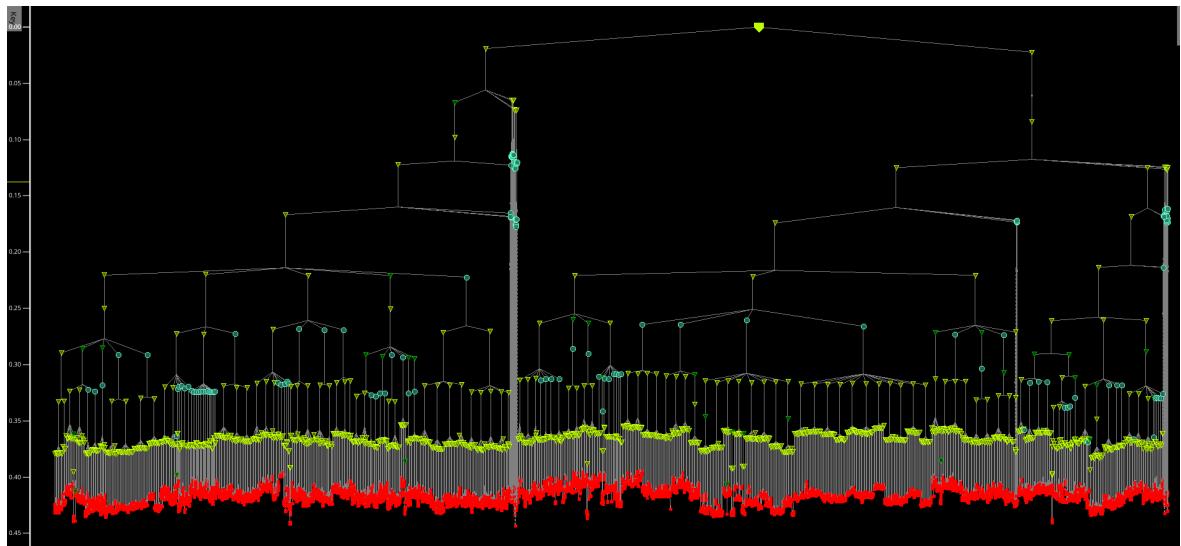


Abbildung 5.1: Der Clock-Tree nach der Clock-Tree-Synthese

Mit dem Befehl `route_opt_design` wird das Design abschließend verdrahtet. Das Ergebnis der Post-Route-Implementierung in Innovus und einem Prototypen aus Joules ist in Abbildung 5.2 zu sehen.

Das Erstellen des Prototypen in Joules fand analog zum Nick-Asic in Abschnitt 3 statt. Es wurde der physische Flow mit hoher Optimierung verwendet. Im Vergleich

zu Joules sind hier nicht nur die Platzierung der Zellen (5.2c, 5.2a) und die Density-Map (5.2d, 5.2d), sondern in der letzten Abbildung (5.2e) auch das Routing zu sehen. Für eine bessere Übersicht zeigt diese Abbildung keine Metalllagen des Power-Routing. In der Abbildung mit Routing sind einige DRC-Violations (weiße Kreuze) zu sehen. Diese müssten für ein reales Design ausgebessert werden, für die Untersuchungen dieser Arbeit können diese vernachlässigt werden. Durch die Density-Map der beiden Designs ist zu sehen, dass die Zellen von Innovus durch eine höhere Optimierung gleichmäßiger verteilt werden und somit weniger Flächen mit hoher Dichte entstehen.

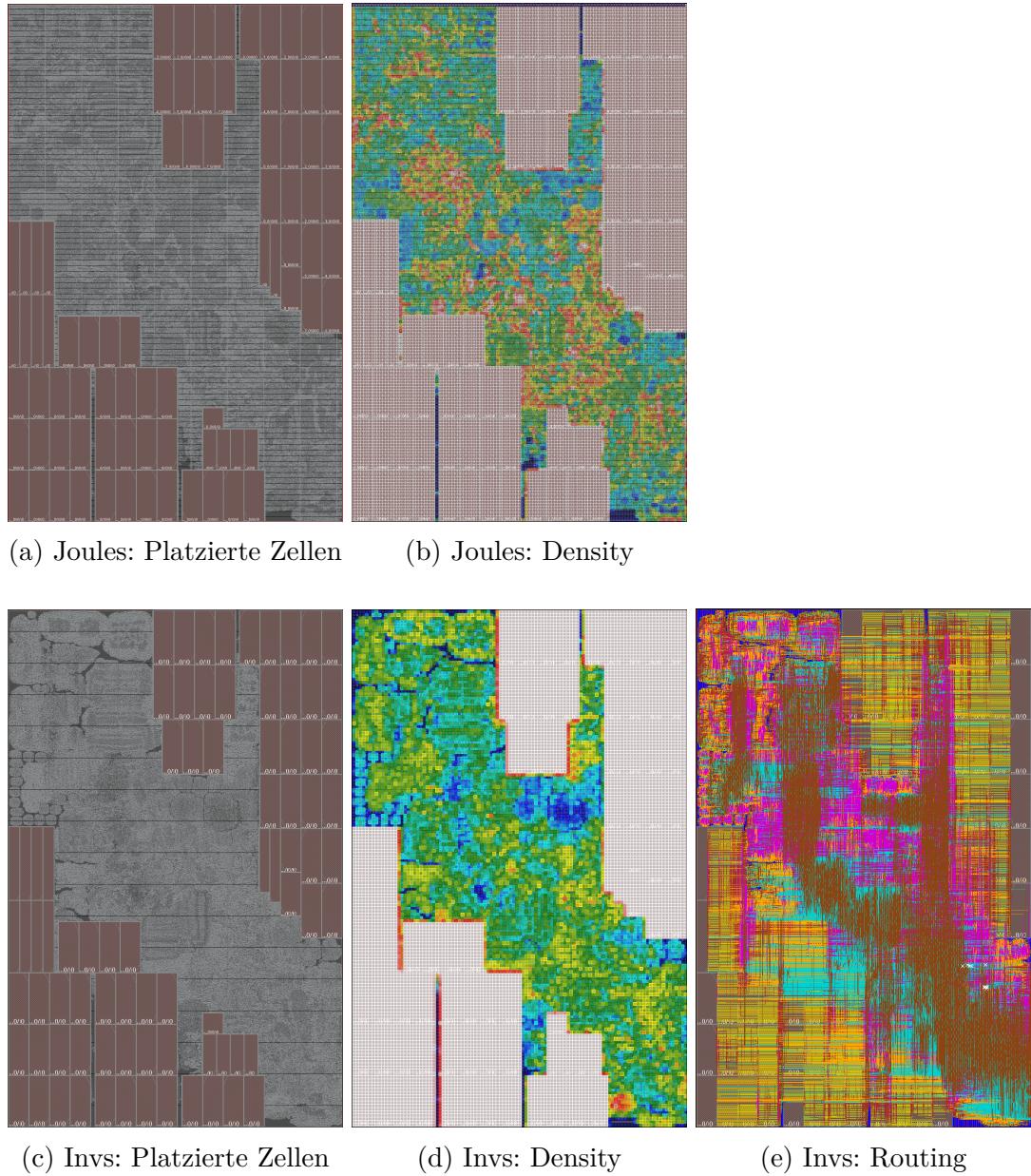


Abbildung 5.2: Ergebnisse der Implementierung und des Joules-Prototypen

Tabelle 5.1 zeigt die Unterschiede zwischen dem Joules-Prototypen im physischen Flow mit hoher Optimierung und der Innovus-Implementierung. Hier ist zu sehen, dass die Density im Verlauf des Implementierungsflows leicht ansteigt, was auf die zusätzlichen Zellen durch die Clock-Tree-Synthese zurückzuführen ist. Nach dem Platzierungsschritt ist keine Congestion mehr vorhanden, was für eine bessere Optimierung spricht. Nach der Clock-Tree-Synthese steigt die Congestion wieder leicht an, was auf die zusätzlichen

Zellen zurückzuführen ist.

Metrik	Joules	Invs-Place	Invs-CTS	Invs-Route
Congestion Hotspot	5.8103	0	0.17344	0.17344
Density	56.84 %	56.62 %	57.07 %	57.24 %

Tabelle 5.1: Auflistung von Density und Congestion des Joules Prototypen und der Implementierung in Innovus

5.2 Power-Analyse

Da nun mit Innovus eine Implementierung durchgeführt wurde, ist es möglich die Power-Analyse zwischen dem Prototypen aus Joules und der vollständigen Implementierung von Innovus zu vergleichen. Abschnitt 5.2.1 beschreibt die Schritte zur Power-Analyse mit Joules und den Vergleich dieser mit Voltus. Daraufhin werden die Ergebnisse der Implementierung in Innovus mit den von Joules in Abschnitt 5.2.2 verglichen.

5.2.1 Joules

Joules unterstützt die Power-Analyse durch die Verwendung von durchschnittlichen Schaltraten der Zellen und durch Einlesen eines Stimulus. Auch hier kann der Stimulus in verschiedenen Dateiformaten vorliegen, im weiteren Verlauf wird die gleiche VCD-Datei verwendet, welche auch für PTPX und Voltus verwendet wird. Bei Joules ergibt sich die Möglichkeit das VCD vor und nach dem erstellen des Prototypen einzulesen, um das VCD während der Erstellung des Prototypen bei Optimierungen zu berücksichtigen. Mit dem Befehl `compute_power` wird eine Power-Berechnung durchgeführt. Tabelle 5.2 zeigt die Unterschiede zwischen einem Prototypen mit angenommenen Schaltraten ohne VCD, dem Einlesen eines VCDs vor und nach dem erstellen des Prototypen. Die Werte sind im physikalischen Modus mit hoher Optimierung entstanden.

Metrik	VCD vor Syn		VCD nach Syn		Ohne VCD	
	Leistung [mW]		Leistung [mW]	Abw.	Leistung [mW]	Abw.
P_{leak}	5.69		5.83	2.46 %	4.47	21.44 %
P_{sw}	33.51		34.59	3.22 %	124.73	272.22 %
P_{int}	73.38		74.28	1.23 %	211.32	187.98 %
P_{ges}	112.57		114.68	1.87 %	340.51	202.49 %

Tabelle 5.2: Vergleich verschiedener Leistungsmetriken mit und ohne Simulation

Bei dem Vergleich der Werte fällt auf, dass zwischen der Simulation vor und nach der Synthese nur ein geringer Unterschied zu sehen ist, die Maximale Abweichung ist bei der Switching-Power mit ca. 3 Prozent zu sehen. Die Ergebnisse mit VCD vor der Synthese sind leicht besser, was darauf schließen lässt, dass Joules das VCD während der Synthese berücksichtigt. Zwischen einer Simulation mit VCD und einer Simulation ohne VCD ist jedoch besonders bei der Switching-Power und der Internen Power ein Unterschied zu sehen. Dies ist zu erwarten, da diese Metriken von der Schaltaktivität der Zelle abhängen, welche ohne das Einlesen einer Schaltaktivität vom Tool angenommen werden. Im weiteren Verlauf wird nur noch das Einlesen vor der Synthese betrachtet, da dies die besten Ergebnisse liefert.

Um die erhaltenen Werte einordnen zu können, wird die aus Joules resultierende Datenbasis in Voltus geladen. Es wird erwartet, dass Voltus ähnliche Ergebnisse wie Joules liefert, da laut dem Hersteller im Hintergrund eine vergleichbare Berechnungsengine des Herstellers läuft.

Die Ergebnisse sind in Tabelle 5.3 und Abbildung 5.3 zu sehen und erfüllen diese Erwartungen nicht.

Metrik	log-low [mW]		log-high [mW]		phys-low [mW]		phys-high [mW]	
	Joules	Voltus	Joules	Voltus	Joules	Voltus	Joules	Voltus
P_{leak}	8.5	8.5	6.6	6.6	6.9	6.9	5.7	5.7
P_{sw}	28.5	42.9	23.3	36.7	29.5	37.9	33.5	27.8
P_{int}	73.0	99.2	70.1	93.4	73.6	98.5	73.4	90.6
P_{ges}	109.9	150.6	99.9	136.6	120.1	143.2	112.6	124.0

Tabelle 5.3: Vergleich der Leistungswerte für unterschiedliche Optimierungsstufen in Joules mit Voltus

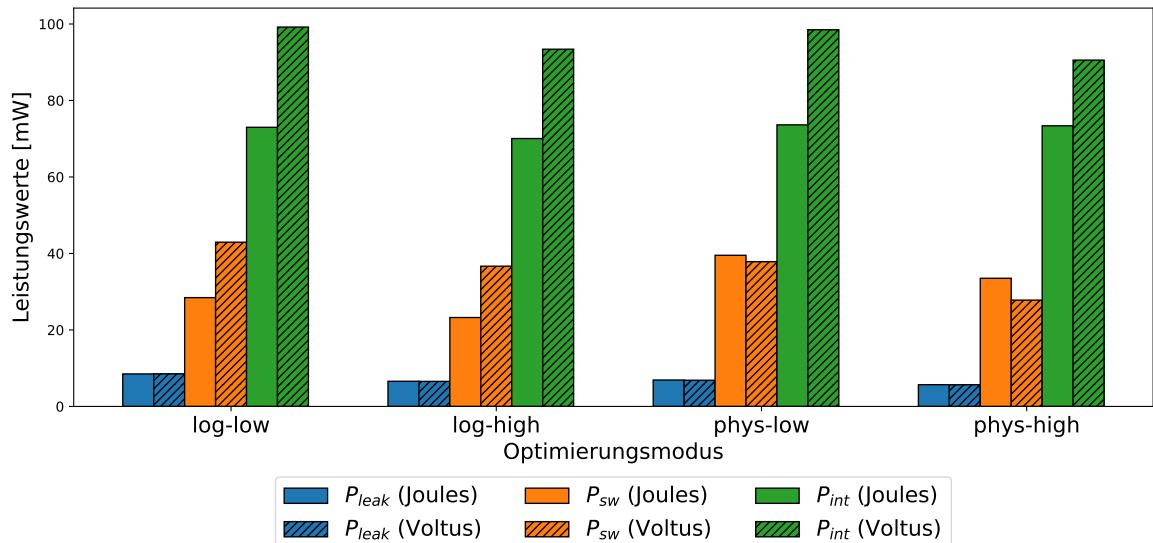


Abbildung 5.3: Visualisierung der Leistungswerte für unterschiedliche Optimierungsstufen in Joules mit Voltus

Hierbei fällt zunächst auf, dass sich der statische Anteil der Leistungsaufnahme (P_{leak}) zwischen Voltus und Joules kaum unterscheidet. Beim Dynamischen Anteil der Leistung (P_{sw} und P_{int}) lässt sich allerdings ein deutlicher Unterschied feststellen. Bei den physischen Optimierungsstufen ist der Wert für die interne Leistung in Voltus höher als in Joules. Die Schaltleistung verhält sich entgegengesetzt, der Wert ist in Voltus geringer als in Joules. Bei den logischen Optimierungsstufen sowohl der Wert für die interne Leistung als auch der Wert der Schaltleistung in Voltus höher. Insgesamt ist die Gesamtleistungsaufnahme der CPU in Voltus im Durchschnitt über alle Optimierungsmodi in Voltus ca. 22 Prozent höher als in Joules. Eine mögliche Ursache hierfür könnte sein, dass Voltus eine genauere Berechnung durchführt. Während die Berechnung mit `compute_power` in Joules nur wenige Minuten benötigt, dauert die Berechnung in Voltus ca. eine halbe Stunde, wobei beide Tools 8 CPU-Kerne zur Verfügung haben.

Insgesamt sind diese Ergebnisse unerwartet und es Bedarf weiterer Analyse. Hier ist eine enge Zusammenarbeit mit dem Hersteller notwendig, was aus zeitlichen Gründen im Rahmen dieser Arbeit nicht realisiert werden konnte.

5.2.2 Vergleich mit Innovus

Die Datenbasis des implementierten Designs nach dem Route-Schritt aus Innovus wird direkt mit `read_db` in Voltus geladen. Daraufhin wird mit dem vorhandenen VCD eine Power-Analyse durchgeführt. Da als Grundlage für die Implementierung in Innovus der Joules-Prototyp verwendet wird, kann das in Joules durch den Befehl `write_name_mapping` erstellte Mapfile geladen werden.

In Tabelle 5.4 sind die Leistungswerte für Joules und Innovus aufgelistet. Um Unterschiede in der Art der Power-Analyse auszuschließen, werden beide Datenbasen gleichermaßen in Voltus eingelesen und die Analyse von Joules selbst ignoriert, da wie im vorherigen Abschnitt erläutert, unerwartet hohe Unterschiede zwischen Joules und Voltus bestehen.

Metrik	Joules [mW]	Innovus [mW]	Abweichung
P_{leak}	5.7	4.6	18.2 %
P_{sw}	27.8	46.3	66.5 %
P_{int}	90.6	104.4	15.3 %
P_{ges}	124.0	155.3	25.2 %

Tabelle 5.4: Vergleich der Leistungswerte zwischen Joules und Voltus

Insgesamt ist zu sehen, dass bei einer realen Implementierung etwa 25 Prozent mehr Leistungsaufnahme erreicht wird, als von Joules vorhergesagt wird. Besonders auffällig sind die Unterschiede in der Switching-Power. Zwei Vermutungen können aufgestellt werden, benötigen aber einer weiteren, detaillierteren Analyse. Da bei Joules kein Routing durchgeführt wird, sondern lediglich eine frühe Routing-Abschätzung erfolgt, ist unklar, welche Kapazitäten und Widerstände hier für die Signalleitungen angenommen werden. Außerdem wird in Joules kein Clock-Tree aufgebaut, es fehlt also ein erheblicher Anteil an Clock-Buffer und Inverterzellen, wodurch die Leistung des Clock-Netzwerkes deutlich zu gering abgeschätzt werden könnte. Beide Punkte sollten zu einer höheren Switching-Power führen und somit die Ergebnisse weiter annähern. Aus Zeitgründen konnte eine detaillierte Analyse dieser Punkte im Rahmen dieser Arbeit nicht durchgeführt werden.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde das „Cadence Joules RTL Design Studio“ im Kontext der ASIC-Entwicklung bei Renesas Electronics Europe evaluiert. Ziel war es, die Eignung des Tools zur frühzeitigen Erkennung von Designproblemen sowie zur Abschätzung der Leistungsaufnahme zu untersuchen.

Dazu wurden verschiedene Testszenarien mit unterschiedlichen Optimierungsstufen, Ressourcenbeschränkungen und Timing-Anforderungen durchgeführt. Die Ergebnisse zeigen, dass Joules im physikalischen Prototyping eine realitätsnahe Einschätzung der Designmetriken wie Congestion, Density und Timing ermöglicht. Der Vergleich mit klassischen Tools wie Innovus zeigt, dass Joules eine signifikante Reduktion der Laufzeit für eine Abschätzung des Front-End-Designers ermöglicht.

Ein weiterer Schwerpunkt lag auf der Analyse der Power-Schätzungen von Joules im Vergleich zu etablierten Signoff-Tools wie Voltus und PTPX. Trotz Bemühungen zur Angleichung der Analyseparameter blieben signifikante Abweichungen bestehen.

Auch die Ergebnisse der Power-Analyse zwischen Joules und Voltus sind überraschend. Hier war eine geringe Abweichung zwischen den beiden Tools zu erwarten. Trotzdem verblieben nennenswerte Unterschiede zwischen den beiden Tools.

Beim Power-Vergleich des Joules-Prototypen und der Innovus-Implementierung fällt auf, dass besonders die Switching-Power durch den Joules-Prototypen noch ungenau approximiert wird. Die anderen Metriken können mit einer Abweichung von weniger als 25 Prozent aber eine gute Möglichkeit darstellen, die Leistung frühzeitig abzuschätzen.

Insgesamt eignet sich Joules zum aktuellen Zeitpunkt als Tool zur Abschätzung, besonders von Congestion und Timing, für den Front-End-Designer. Die Analysefunktionen zur Leistungsaufnahme des ASICs benötigen weitere Untersuchungen, welche unter anderem vom Hersteller des Tools unterstützt werden müssen.

Literatur- und Quellenverzeichnis

- Abhay Chopde, A. M. K. (2024). Physical Design: Methodologies and Developments. <https://arxiv.org/html/2409.04726v1> [Zugriff am 4.06.2025].
- Andrew B. Kahng, Jens Lienig, I. L. M. J. H. (2022). *VLSI Physical Design: From Graph Partitioning to Timing Closure*.
- Cadence (2017). Lef/DEF 5.8 Language Reference.
- Cadence (2023). Cadence Unveils Joules RTL Design Studio, Delivering Breakthrough Gains in RTL Productivity and Quality of Results. https://www.cadence.com/en_US/home/company/newsroom/press-releases/pr/2023/cadence-unveils-joules-rtl-design-studio-delivering-breakthrough.html [Zugriff am 1.06.2025].
- Cadence (2023a). Joules rtl design studio. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/rtl-analysis/joules-rtl-design-studio.html. Zugriff am 18. Juli 2025.
- Cadence (2023b). Prototype Design Flow in Joules RTL Design Studio.
- Cadence (2025). Genus user guide.
- Cadence (2025). Timing and mode setup. Zugriff am 01.08.2025.
- Cadence (2025). Voltus Stylus Common UI Text Reference Manual.
- Chakravarthi, Veena S, K. S. R. (2022). *SoC Physical Design: A Comprehensive Guide*. Springer.
- Melzner, H. (2006). Smaller is better? maximization of good chips per wafer by co-optimization of yield and chip area. In *The 17th Annual SEMI/IEEE ASMC 2006 Conference*, pages 372–379.
- Patil, C. A. (2021). The PPA Management In Semiconductor Product Development.
- Saxena, P., Shelar, R. S., und Sapatnekar, S. (2007). *Routing Congestion in VLSI Circuits: Estimation and Optimization*. Springer Science & Business Media.
- Synopsys (2006). Primetime px recommended methodology for power analysis.
- Synopsys (2025). Synopsys multivoltage implementation and verification overview. https://spdocs.synopsys.com/dow_retrieve/qsc-w/dg/fcolh/W-2024.09-SP3/fcolh/smfvfug/low_power_design_strategies/dynamic_static_power.html.
- tcl-lang.org (2025). Uses for Tcl/Tk. <https://www.tcl-lang.org/about/uses.html> [Zugriff am 4.06.2025].
- Welch, B. B., Jones, K., und Hobbs, J. (2003). *Practical programming in Tcl and Tk*. Prentice Hall Professional.
- Weste, N. H. und Harris, D. (2015). *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India.

Abbildungsverzeichnis

2.1	Die Schritte im Design-Flow, vereinfacht nach Andrew B. Kahng (2022, S. 6)	3
2.2	Ein Floorplan vor und nach dem Power-Routing	4
2.3	Ein Design nach dem Placement	5
2.4	Der Schaltplan eines Buffers mit eingezeichneten Strömen, nach Synopsys (2006)	8
3.1	Layouts für verschiedene Anzahl an Metalllagen	15
3.2	Visualisierung der Ergebnisse für unterschiedliche Clock-Frequenzen mit High- und Low-Effort	16
4.1	Der Pfad vom Clock-Eingang zum Clock-Buffer <code>cts_buf_13227436055</code> .	21
4.2	Das Clock-Signal des VCDs mit der Frequenz 1 GHz in Simvision . .	22
4.3	Ausgabe von Simvision zum Toggle-Count der Zelle <code>fp_mul_en[0]</code> . .	24
4.4	Beginn des Clock-Signals in Simvision	25
5.1	Der Clock-Tree nach der Clock-Tree-Synthese	27
5.2	Ergebnisse der Implementierung und des Joules-Prototypen	28
5.3	Visualisierung der Leistungswerte für unterschiedliche Optimierungsstufen in Joules mit Voltus	30

Tabellenverzeichnis

3.1	Ergebnisse des Prototypen für unterschiedliche Anzahl an Metalllagen mit High- und Low-Effort	14
3.2	Ergebnisse des Prototypen für unterschiedliche Clock-Frequenzen mit High- und Low-Effort	16
3.3	Ressourcenverbrauch der Unterschiedlichen Prototypen und Innovus . .	17
4.1	Unterschied der berechneten Leistungswerte von Voltus und PTPX . .	19
4.2	Auflistung von P_{int} beider Tools mit unterschiedlichen Frequenzen im SDC für beide Clock-Buffer	23
4.3	Auflistung der Gesamtleistungen beider Tools nach Frequenz im SDC .	23
4.4	Vergleich der Ein- und Ausgänge der betrachteten Clock-Gating-Zelle .	23
4.5	Leistungswerte nach Änderung des Mapfiles	25
5.1	Auflistung von Density und Congestion des Joules Prototypen und der Implementierung in Innovus	29
5.2	Vergleich verschiedener Leistungsmetriken mit und ohne Simulation .	29
5.3	Vergleich der Leistungswerte für unterschiedliche Optimierungsstufen in Joules mit Voltus	30
5.4	Vergleich der Leistungswerte zwischen Joules und Voltus	31

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit selbständig verfasst und keinen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Die Regelungen der geltenden Prüfungsordnung zu Versäumnis, Rücktritt, Täuschung und Ordnungsverstoß habe ich zur Kenntnis genommen.

Diese Arbeit hat in gleicher oder ähnlicher Form keiner Prüfungsbehörde vorgelegen.

Heiligenhaus, den 23. August 2025



Unterschrift