Großer Beleg

ASIC-Synthese der SHAP-Mikroarchitektur

Andrej Olunczek

geboren am 20.02.1984 in Cottbus

Matrikelnummer: 3066276

30.03.2009

Technische Universität Dresden
Fakultät Informatik
Institut für Technische Informatik

Betreuender Hochschullehrer:	Prof. Dr.-Ing. habil. Rainer G. Spallek
Betreuer:	Dipl.-Inf. Martin Zabel

Inhaltsverzeichnis

1 Einleitung **1**

2 Vorbetrachtung **3**

2.1 Aktuelle eingebettete Prozessoren im Vergleich 3

2.2 Abhängigkeiten zwischen Leistungsdaten 4

2.3 Abhängigkeiten von der Cachegröße 6

3 Vorbereitung **7**

3.1 Analyse der Aufgabe 7

3.2 Analyse des Quelltextes 8

3.3 Speichermodule 9

3.3.1 Generierung von RAM-Macros 9

3.3.2 Stack 10

3.3.3 Methoden-Cache 10

3.3.4 Speicher des Garbage Collectors 10

3.3.5 Microtext 11

3.3.6 Microdata 12

3.3.7 Zusammenfassung der Speicher 12

3.4 Sonstige Anpassungen 13

4 Synthese **15**

4.1 Einflussnahme über Constraints 15

4.2 Ergebnisse 17

4.3 Probleme 18

5 Place & Route **21**

5.1 Vorgehensweise 21

5.2 Ergebnisse 22

5.3 Probleme 26

6 Auswertung **27**

7 Zusammenfassung und Ausblick **31**

A **A – 1**

A.1 Floorplan . . . . . A – 2
A.2 Finales Layout . . . . . A – 3

Tabellenverzeichnis

2.1 Vergleich aktueller Prozessoren 4
2.2 Abhängigkeiten von der Cachegröße 5

3.1 Zusammenfassung RAM-Macros 12

5.1 Timing-Ergebnisse 25

6.1 CaffeineMark-Ergebnisse 27
6.2 Vergleich SHAP mit anderen Prozessoren 28

Abbildungsverzeichnis

2.1 Abhängigkeiten zwischen Leistungsdaten 5

3.1 Design Flow 7
3.2 SHAP-Architektur 8
3.3 Generierung von RAM-Macros 9
3.4 Pad- vs. Core-limited 13

4.1 Fläche und Leistungsaufnahme nach Synthese 17

5.1 Floorplan mit Power-Netz 22
5.2 Zeitlicher Leistungsaufnahmeverlauf 23
5.3 Maximaler IRdrop 24
5.4 Verteilung der Komponenten 24

1 Einleitung

Viele Mikrokontroller und eingebettet Prozessoren haben einen eigenen RISC-Befehlssatz, oft auch Befehlssatzerweiterungen für andere Programmiersprachen und -konzepte. Die Programmierung erfolgt meist in C, oft wird dafür eine Reihe eigener Softwarewerkzeuge der Hersteller bereitgestellt. Der Quellcode ist meist wenig übersichtlich und Fehler lassen sich nur schwer finden. Im Gegensatz dazu steht Java, dass vom Konzept her von Anfang an auf strikte objektorientierte Programmierung setzt. Weitere Vorteile von Java sind vor allem das Konzept der strukturierten Ausnahmen zum Abfangen und Behandeln von Fehlern zur Laufzeit und die automatische Speicherverwaltung. Objekte, die im Speicher liegen und nicht mehr benötigt werden, müssen nicht mehr „von Hand“ entfernt werden, sondern diese Aufgabe wird automatisch vom Garbage Collector erledigt. Genau an dieser Stelle setzt die SHAP-Architektur an, indem ein eingebetteter Prozessor zur Verfügung gestellt wird, der die Konzepte der Java-Virtual-Maschine in vollem Umfang nativ unterstützt. Dadurch können auch im eingebetteten Bereich alle Vorteile von Java genutzt werden. Es können so die zahlreichen, oft frei erhältlichen und weit entwickelten, Softwarewerkzeuge genutzt werden, um die Software für den Prozessor zu schreiben.

Warum soll jetzt eine ASIC-Synthese durchgeführt werden? Bisher wurde die SHAP-Architektur nur auf FPGAs getestet, und so die Funktionsfähigkeit und Leistungsfähigkeit der Architektur vielfältig bewiesen. FPGAs sind zwar sehr flexibel, und können in ihrer logischen Funktion beliebig angepasst und geändert werden, aber das funktioniert natürlich nur auf Kosten der Geschwindigkeit und der Leistungsaufnahme. Ziel der Arbeit ist es nun, zum einen herauszufinden, welches leistungsmäßige Potential in der Architektur steckt, im Konkreten also, welche Taktfrequenz kann erreicht werden. Dabei ist auch wichtig zu wissen, welche Leistungsaufnahme zu erwarten ist, und welche Chipfläche benötigt wird, um zu sehen, wie sich die Architektur im Vergleich zu den Architekturen anderer Prozessoren verhält. Zum anderen ist es auch wichtig zu wissen, wie sich die ASIC-Implementierung im Verhältnis zur FPGA-Implementierung verhält, welcher Vorteil also zu erwarten ist.

2 Vorbetrachtung

2.1 Aktuelle eingebettete Prozessoren im Vergleich

Um einen Vergleich der SHAP-Mikroarchitektur mit anderen eingebetteten Prozessoren zu haben, sollen in diesem Abschnitt aktuelle Architekturen untersucht werden. Schwerpunkt ist dabei vor allem Chipfläche, Leistungsaufnahme, Taktfrequenz und auch Java-Unterstützung. Leider ist nicht zu jedem Prozessor eine verlässliche Angabe zu bekommen, wie viel Chipfläche benötigt wird. Erreichte Taktfrequenz, Chipfläche und Leistungsaufnahme ist dabei sehr stark abhängig von der verwendeten Prozess-Technologie, der Technologie-Bibliothek und auf welche Kriterien bezüglich Fläche oder Taktfrequenz optimiert wurde. In der folgenden Übersicht sind einige aktuelle Prozessoren aufgeführt, die Leistungsdaten sind in Tabelle 2.1 aufgelistet.

- aJile aj-100:
  - Java: nativ
  - Datenbreite: 32 Bit
  - Cache: 32K Datenspeicher + 16K Microcode
- Fujitsu MB86799 (picoJava-II-Core):
  - Java: nativ
  - Datenbreite: 32 Bit
  - Cache: 8K Instruction Cache + 8K Data Cache
- ARM926EJ-S:
  - Java: Jazelle-Befehlscode-Erweiterung für Bytecode-Unterstützung
  - Datenbreite: 32 Bit
  - Cache: 8K Instruction Cache + 8K Data Cache
- AVR32-AT32AP7000
  - Java: Erweiterung um Hardware-JVM möglich
  - Datenbreite: 32 Bit
  - Cache: 16K Instruction Cache + 16K Data Cache + 2x16K SRAM

CPU	Technologie	Taktrate	Benchmark[a]	Leistungsaufn.	Die Size
aJile aj-100	0,25 μm	100 MHz	2,75 CM/MHz	2,57 mW/MHz	
Fujitsu MB86799	0,25 μm	66 MHz	9,4 CM/MHz	5,4 mW/MHz	
ARM926EJ-S	0,13 μm	238 MHz	5,0 CM/MHz	0,36 mW/MHz	1,45 mm^2
AT32AP7000	0,18 μm	150 MHz		2,08 mW/MHz	

[a] Emb. CaffeineMark 3.0

Tabelle 2.1: Vergleich der Leistungsdaten aktueller Prozessoren

Es gibt bereits einige Prozessoren, die Java nativ beherrschen, wie zum Beispiel der aJ-100 oder der MB86799, wesentlich mehr beherrschen zumindest teilweise Java, indem sie nach dem Booten auch Java-Programme ausführen können, und dafür entsprechende Erweiterungen in Hardware haben. Eine Datenbreite von 32 Bit ist heute mehr oder weniger Standard, auch an entsprechendem integrierten Cache wird heutzutage nicht mehr gespart. Zum AVR32 ist zu sagen, dass es auch noch eine zweite Generation gibt, die in 0,13 μm gefertigt wird und 200 MHz erreicht. Bei dem Beispielprozessor von ARM ergeben sich die Daten aus der flächenoptimierten Variante, es gibt auch Varianten in 0,18 μm oder 90 nm, jeweils nach Fläche oder Geschwindigkeit optimiert. Die Daten für Fläche und Leistungsaufnahme beziehen sich dabei nur auf den Core ohne Cache und IO-Pads. Die in Tabelle 2.1 angegeben Werte zur Leistungsaufnahme sind mit Vorsicht zu betrachten. Zum einen hängen diese von der ausgeführten Anwendung ab, zum anderen bezieht sich die Angabe zum ARM nur auf den Core, währen bei den anderen ein Großteil der Leistung auch von den IO-Zellen aufgenommen wird.

2.2 Abhängigkeiten zwischen Leistungsdaten

Im Folgenden will ich nun eine kleine Fallstudie zu den Abhängigkeiten von Taktfrequenz, Chipfläche und Leistungsaufnahme bezüglich der Prozess-Technologie und der Optimierung hinsichtlich Taktfrequenz oder Chipfläche zeigen. Als Beispiel dient der Core des ARM Cortex-M3. In Abbildung 2.1 ist sehr schön zu erkennen, dass mit kleiner werdender Strukturgröße Chipflächenbedarf und Leistungsaufnahme sinken und gleichzeitig die Taktfrequenz steigt.

Das Sinken des Chipflächenbedarfs erklärt sich direkt mit den kleineren Strukturgrößen. Wird die Kanallänge der Transistoren kleiner, können die Transistoren als solche komplett in kleinerem Maßstab gefertigt werden. Das führt natürlich zu einer Kompaktierung der Chipfläche. Mit kleiner werdenden Strukturen sind auch die Gatekapazitäten der Transistoren kleiner. Das Umschalten der Transistoren geht also bei kleinerer Versorgungsspannung schneller vonstatten. Daraus resultiert dann zum einen eine geringere Leistungsaufnahme bei gleicher Taktrate. Man kann dadurch aber auch die Taktrate erhöhen, was dann wiederum zu einer steigenden Leistungsaufnahme führt.

Es ist auch deutlich der Einfluss der Optimierungsstrategie zu erkennen. Bei gleicher Strukturgröße führt die Flächenoptimierung auch zu geringerer Leistungsaufnahme, was positiv zu

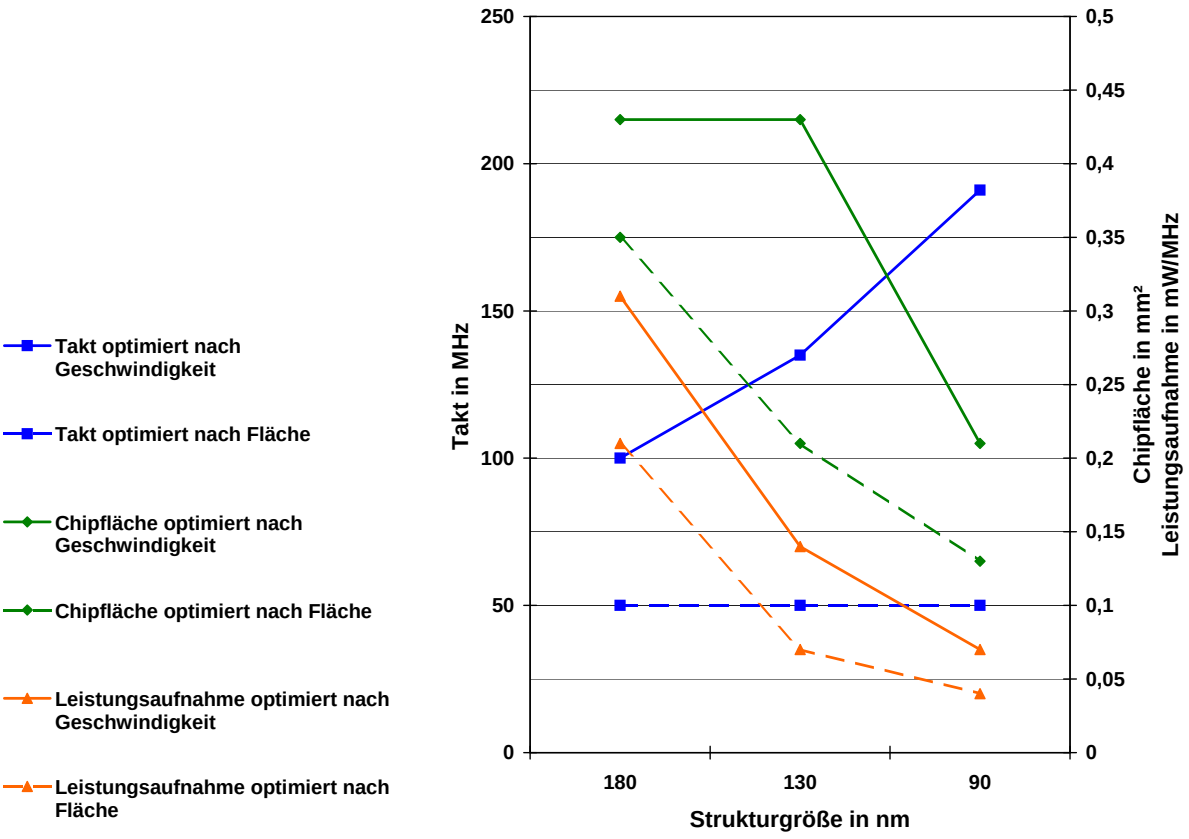


Abbildung 2.1: Technologiebedingte Abhängigkeiten zwischen Leistungsdaten

bewerten ist, dafür muss man aber leider eine geringere erreichbare Taktfrequenz in Kauf nehmen. Optimiert man in Richtung einer möglichst hohen Taktfrequenz, kann es vorkommen, dass trotz sinkender Strukturgröße die Chipfläche nicht kleiner wird. Eine Optimierung nach hoher Taktfrequenz führt automatisch auch zu einem höheren Chipflächenbedarf und einer höheren Leistungsaufnahme.

CPU	Architektur	Technologie	Taktrate	Leistungsaufnahme	Die Size
ARM920T	2 * 16K Cache	0,18 μm	200 MHz	0,8 mW/MHz	11,8 mm^2
ARM920T	2 * 16K Cache	0,13 μm	250 MHz	0,25 mW/MHz	4,7 mm^2
ARM922T	2 * 8k Cache	0,18 μm	200 MHz	0,8 mW/MHz	8,1 mm^2
ARM922T	2 * 8k Cache	0,13 μm	250 MHz	0,25 mW/MHz	3,2 mm^2
ARM9[a]	ohne Cache	0,18 μm	200 MHz	0,8 mW/MHz	4,4 mm^2
ARM9	ohne Cache	0,13 μm	250 MHz	0,25 mW/MHz	1,7 mm^2

[a] theoretische Betrachtung, kein real existierender ARM-Core

Tabelle 2.2: Leistungsdaten in Abhängigkeit der Cachegröße

2.3 Abhängigkeiten von der Cachegröße

Einen wesentlichen Einfluss auf die Chipfläche hat auch der Cache. Bei vielen Prozessoren werden große Teile der Chipfläche vom Cache belegt. Anhand des ARM9(20T/22T) ein kleines Beispiel dazu. In Tabelle 2.2 ist sehr schön zu erkennen, dass integrierter Speicher einen Großteil der Chipfläche einnimmt, also bei der Produktion ein wichtiger Kostenfaktor ist. Zum anderen ist aber auch zu sehen, dass die Speichergröße keinen Einfluss auf die Leistungsaufnahme hat. Zu beachten ist in dem Beispiel aber, dass in dem fiktiven Fall von 0 KB Cache nur die Differenz der beiden anderen Fälle abgezogen wurde. Die ganze Ansteuer- und Kontrolllogik des Caches in noch vorhanden, braucht demzufolge auch Strom. In sofern sind die Daten zur Leistungsaufnahme nur bedingt relevant, real beträgt die Leistungsaufnahme der Speicher einige Prozent an der gesamten Leistungsaufnahme.

3 Vorbereitung

3.1 Analyse der Aufgabe

Als erstes ist nun zu analysieren, welche Arbeitsschritte die Aufgabenstellung erfordert. In Abbildung 3.1 ist ein vereinfachter Ablauf der Arbeitsschritte zu sehen. Im groben Verlauf ist aus der gegebenen VHDL-Beschreibung der SHAP-Architektur eine Netzliste zu synthetisieren. Dazu werden dem Synthesetool verschiedene Parameter, so genannte Constraints übergeben, in denen bestimmte Optimierungswünsche bezüglich zu erreichender Taktfrequenz und/oder Leistungsaufnahme definiert sind. Näheres zur Synthese wird in Kapitel 4 beschrieben. Wie in den folgenden Abschnitten in diesem Kapitel beschrieben, sind aber nicht alle VHDL-Beschreibungen synthetisierbar, so dass noch etwas Vorarbeit nötig ist. Die Funktion der synthetisierten Netzliste ist dann über Simulationen zu überprüfen. Dazu wird das Laden und Ausführen eines kleinen Testprogramms durch den synthetisierten Prozessor simuliert.

Ist die Synthese erfolgreich geht es mit dem „Place & Route“ weiter. Dort werden die Elemente der synthetisierten Netzliste platziert (Place) und physikalisch verbunden (Route). Dabei werden die Positionen von IO-Zellen und Macros, wie z.B. RAM vorgegeben. Die eigentliche Logik, die Standardzellen der Netzliste werden von dem Tool automatisch platziert, um möglichst effektiv die einzelnen Zellen mit einander verbinden zu können. Beeinflussen kann man das ganze wieder über vielfältige Parameter. Der „Place & Route“-Schritt wird in Kapitel 5 genauer beschrieben. Das Ergebnis wird dann natürlich wieder per Simulation überprüft, und mit Hilfe der extrahierten physikalischen Daten die Leistungsaufnahme berechnet.

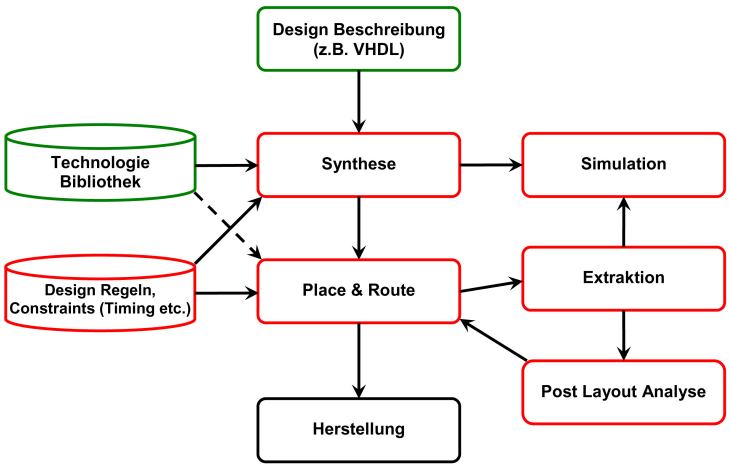


Abbildung 3.1: Der ASIC-Design-Flow vom HDL-Code zum Layout

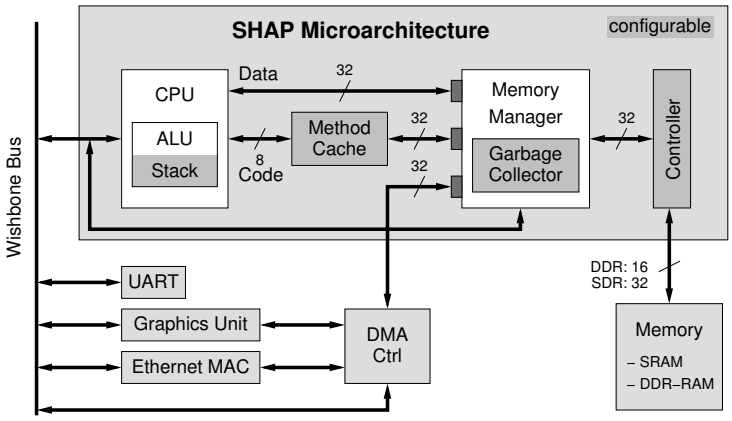


Abbildung 3.2: Überblick über die SHAP-Architektur

3.2 Analyse des Quelltextes

Um nun herauszufinden, wie sich SHAP als ASIC verhält, welche Leistungsdaten zu erwarten sind und wie er sich im Vergleich zu anderen Lösungen schlägt, ist nun erst einmal zu analysieren, welchen HDL-Code man ohne Probleme weiterverwenden kann, und was man gegen ASIC-spezifischen Code oder Module tauschen muss. Der bisherige Code war für eine Implementierung auf FPGAs ausgelegt. Prinzipiell kann allgemeiner HDL-Code sowohl auf die CLBs eines FPGAs als auch auf die Logik, bzw. Standardzellen eines ASICs abgebildet, sprich synthetisiert werden. Die Besonderheit eines FPGAs ist unter anderem auch, dass er spezielle voroptimierte Funktionseinheiten enthält, die er mit nutzen kann. Dazu gehören neben fertigen Multiplizierern, IO-Einheiten und Taktgeneratoren vor allem integrierte Speichermodule. Für Xilinx-FPGAs sind das z.B. die BlockRAM-Module. Dabei gibt es zwei Arten, wie man solchen Speicher in den eigenen HDL-Code integrieren kann. Entweder man instanziiert die RAM-Module direkt als RAM-Komponenten, oder man beschreibt den Speicher mit HDL-Code formal und die Synthese Tools erkennen die Code-Struktur als Speicher und ersetzen ihn durch RAM-Module.

Die Synthesetools des ASIC-Workflows kennen nun weder die instanziierten BlockRAM-Module, noch können sie entsprechenden HDL-Code in RAM-Macros überführen. Für einen Speicher würde das Tool für jede Speicherzelle entsprechende Register instanziieren und diese über einen riesigen MUX-Baum oder ähnliche Logikstrukturen ansteuern. Das wäre in höchstem Maße ineffizient, und würde meist zu keinem funktionierenden Ergebnis führen. Diese Methode kann nur für Speicher von wenigen Byte angewandt werden. Die Hauptaufgabe der Vorbereitung des HDL-Codes ist es nun, solche RAM-Macros zu erstellen, und in den bestehenden Code zu integrieren. Die RAM-Macros bestehen aus dicht gepackten SRAM-Zellen, sind also ab einer bestimmten Größe wesentlich kompakter, schneller und sparsamer als ein vergleichbares Konstrukt aus Registern.

Betrachten wir nun einmal die SHAP-Architektur (siehe Abbildung 3.2), wo es überall größere Speicher gibt. Das sind natürlich zum einen der Stack in der CPU und der Methoden-Cache,

aber auch der Garbage Collector enthält Speichermodule, um Referenzmarkierungen zu speichern. Dazu kommen noch ein ROM-Modul für den Microtext für die Befehlsdekodierung und ein Konstantenspeicher, den Microdata.

3.3 Speichermodule

3.3.1 Generierung von RAM-Macros

Zur Generierung von RAM-Macros wird vom Anbieter der Standardzellenbiblitothek ein Tool, der so genannte „Memaker“, zur Verfügung gestellt, mit dessen Hilfe man verschiedene Speichertypen unterschiedlicher Größe in Form von Macros generieren kann. Es gab verschiedene Prozesstechnologien zur Auswahl, die hinsichtlich Geschwindigkeit oder Leistungsaufnahme optimiert sind. Ich habe mich hier für den „High Speed Logic Process“ entschieden. Zur Generierung von RAM-Macros wählt man zuerst den gewünschten Speichertyp aus, und gibt im Anschluss ein, wie viel Wörter ein Speicher enthält, und wie viel Bit ein Wort groß ist. Der Speicher-Generator stellt einem dann verschiedene passende Bauweisen zur Verfügung (siehe Abbildung 3.3). Diese unterscheiden sich im so genannten „Column-Mux“, kurz CM. Je nach Bauweise werden 2, 4, 8 oder sogar 16 Werte gleichzeitig aus dem Speicher gelesen, und über einen Mux der richtige Wert ausgegeben. Ein kleiner CM bedeutet dabei einen höheren Aufwand in der Adressierung, der Speicher wird langsamer. Gleichzeitig werden aber auch weniger Speicherzellen aktiv, der Stromverbrauch sinkt. Bei einem großen CM ist es genau anders herum, weniger Adressierungsaufwand und damit schneller, dafür mehr aktive Speicherzellen, die Leistungsaufnahme steigt. Es ist also ein Kompromiss zwischen Schnelligkeit und Leistungsaufnahme zu finden.

Für den Fall, dass man Dual-Port- oder Two-Port-Speicher (Bezeichnung im Memaker, bei Xilinx z.B. Simple-Dual-Port genannt) nutzt, ist es von Interesse, was passiert, wenn man auf einem Port von der Adresse liest, auf der man gleichzeitig auf dem anderen Port schreibt. Prinzipiell gibt es drei Möglichkeiten, „read first“, „write first“ oder „don’t care“. Im Fall von „read first“ wird auf dem lesenden Port noch der alte Wert ausgegeben, bevor der Neue geschrieben wird. Bei „write first“ wird erst der neue Wert geschrieben und dieser dann auf dem lesenden Port ausgegeben. Die generierten RAM-Macros beherrschen nur den dritten Fall, das „don’t care“, dabei wird auf dem lesenden Port ein unbestimmter Wert ausgegeben, eine Weiterverar-

FSC0H_D_SJ (Sync. High Density DPRAM)

Column Mux (Aspect Ratio)	Gate Count	Taa(ns) BC,TC,WC loading=0.01pF ckslew=0.016ns	DC Power (uA)	AC Power (mA/MHz)	Area (mm_sq)	Width (um)	Height (um)
4 (1024x32x1)	65040	1.445,2.125,3.556	59.738	0.020	0.266	385.200	691.600
8 (1024x32x1)	62138	1.092,1.602,2.670	40.275	0.021	0.255	662.800	384.000
16 (1024x32x1)	71392	0.957,1.407,2.359	31.228	0.028	0.292	1202.400	243.200

Abbildung 3.3: Generierung und Auswahl von RAM-Macros

beitung des Ausgangssignales ist also nicht sinnvoll möglich. Da die Dual-Port-RAM-Macros aber am Ausgang des schreibenden Ports den neuen Wert ausgeben, kann man über eine einfache Logik ein „write first“-Verhalten erreichen, in dem man den Ausgangswert des schreibenden Ports auf den Ausgang des Lesenden umleitet.

3.3.2 Stack

Der Stack ist mit, je nach Konfiguration, 4 - 8,25 KB einer der beiden größten Speicher in der SHAP-Architektur. Der Speicher ist ein klassischer Dual-Port-Speicher, wobei einer der beiden Schreib/Lese-Ports nur zum Lesen genutzt wird, die zweite Schreibeinheit liegt also quasi ungenutzt auf dem Chip. Für den Fall dass zur selben Zeit auf der selben Adresse gelesen und geschrieben wird ist für den Stack das Verhalten egal, entspricht also mit einem „don’t care“-Verhalten der standardmäßigen Funktionsweise des generierten Dual-Port-RAMs. Für den Stack sind Konfigurationen für 10 oder 11 Adressbits bei jeweils 33 Bit Wortlänge vorgesehen. Für die umzusetzende SHAP-Konfiguration wird die Variante mit 11 Adressbits, also 2048 Einträgen genutzt, also 8,25 KB. Der Stack hat damit eine maximale Zugriffszeit von 3,7 *ns* bei einer Stromaufnahme von 25 $\mu A/MHz$ und einem Flächenbedarf von 0,453 mm^2.

3.3.3 Methoden-Cache

Der Methoden Cache hat je nach Konfiguration 2 - 16 KB bei einer Wortlänge von 32 Bit, und ist damit der zweite große Speicher der SHAP-Architektur. Wie der Stack ist auch der Methoden-Cache ein Dual-Port-Speicher. Wie beim Stack dient auch hier nur ein Port zum Schreiben und Lesen, vom anderen Port wird nur gelesen. Die Besonderheit an dem reinen Leseport ist, das von diesem nur 8-Bit-Wörter gelesen werden. Da die RAM-Macros keine unterschiedlichen Wortbreiten bei den beiden Ports unterstützen, wurde hinter den Ausgang ein Mux geschaltet, der aus den unteren beiden Adressbits das jeweilige Byte auswählt und ausgibt. Auch der Methoden Cache benötigt für den Fall des gleichzeitigen Lesen und Schreiben auf die gleiche Adresse nur das standardmäßige „don’t care“-Verhalten der RAM-Macros. An Konfigurationen sind nur variable Adressenlängen von 9 bis 12 Bit für den Schreib-Lese-Port vorgesehen, der zweite Lese-Port hat dementsprechend zwei Bit mehr. Die umzusetzende Konfiguration verwendet 9 Bitadressen, bei einer Wortlänge von 32 Bit ist der Speicher also 2 KB groß. Der Methoden Cache hat damit eine maximale Zugriffszeit von 2,15 *ns* bei einer Stromaufnahme von 19 $\mu A/MHz$ und einem Flächenbedarf von 0,159 mm^2.

3.3.4 Speicher des Garbage Collectors

Der Speicher im Garbage Collector ist mit 0,25 - 2 KB ein relativ kleiner Speicher, wird aber in doppelter Ausführung benötigt. Der Speicher hat zwei Besonderheiten, zum einen wird am ersten Port Bitweise geschrieben und gelesen, am zweiten Port werden 16-Bit-Wörter gelesen, die Schreibfunktion des zweiten Ports wird nicht genutzt. Die zweite Besonderheit ist, dass im Fal-

le des gleichzeitigen Lesens und Schreibens auf die gleiche Adresse ein „read first“-Verhalten gefordert ist. Wie oben beschrieben, beherrschen die RAM-Macros nur das „don’t care“- oder mit kleiner Erweiterung das „write first“-Verhalten. Um ein „read first“-Verhalten zu erreichen, müssen wir also erst den alten Wert lesen, und danach den neuen Wert schreiben, beides aber innerhalb eines Taktzykluses. Dazu wird normal auf die steigende Taktflanke gelesen, und direkt auf die folgende fallende Taktflanke geschrieben. Da beim Schreiben aber der neue Wert auf den Ausgang propagiert wird, würde das soeben gelesene Signal noch vor der nächsten steigenden Taktflanke wieder ungültig. Wir können also nicht auf den Port schreiben, von dem wir gelesen haben. Wir brauchen also einen Port zum Schreiben, und einen zweiten zum Lesen. Da es keinen Tri-Port-Speicher gibt, bleibt uns nur der Ausweg, den Speicher zu verdoppeln. Dafür können wir Two-Port-Speicher nutzen. Anders als Dual-Port-Speicher, bei dem man auf beiden Ports Lesen und Schreiben kann, gibt es beim Two-Port-Speicher einen Port auf dem man nur Schreiben kann, und einen zweiten Port, auf dem man nur Lesen kann. Wir lesen also mit der steigenden Taktflanke die alten Werte aus den Lese-Ports der beiden Speicher, und schreiben danach auf die folgende fallende Flanke synchron in beide Speicher den selben Wert. Beide Speicher haben also zu jedem Zeitpunkt den identischen Inhalt. Nachteil dieser Lösung ist, dass man die doppelte Fläche benötigt.

Auch für die andere Besonderheit, das man auf einen Port Bitweise zugreift, auf den anderen in 16-Bit-Wörtern, verlangt ein paar Anpassungen. Der Speicher ist für 16-Bit-Wörter erstellt, aber mit der Erweiterung, dass man nur Teile des Wortes beschreiben kann. Dafür hat der Speicher für jedes der 16 Bits ein eigens Write-enable-Signal. Vor den Speicher wurde ein kleiner Encoder geschaltet, der aus den unteren 4 Bit der Adresse eine Bitmaske für die Write-enable-Signale generiert. Da jeweils immer nur ein Bit beschrieben wird, entspricht die Kodierung einer one-hot-Kodierung. Für den 1-Bit-Ausgang ist dementsprechend auch wieder ein Mux geschaltet, der aus dem 16-Bit-Wort das richtige Bit ausgibt. Für die Speicher des Garbage Collectors sind Konfigurationen von 7 bis 10 Adressbits für die 16-Bit-Wörter vorgesehen. Für den hier zu betrachtenden Fall wird ein Speicher mit 7 Adressbits benötigt. Bei 16 Bit Wortlänge entspricht dass 0,25 KB. Die maximale Zugriffszeit beträgt 1,3 *ns* bei einer Stromaufnahme von 6 $\mu A/MHz$ und einem Flächenbedarf von 0,037 mm^2. Da der Speicher innerhalb eines Modules aufgrund der „read first“-Problematik verdoppelt werden muss und der Garbage Collector zwei solcher Module benötigt, werden also vier dieser RAM-Macros gebraucht. Das entspricht 1 KB Speicher, bei einem Gesamtstromverbrauch von 24 $\mu A/MHz$ und einem Gesamtflächenbedarf von 0,148 mm^2.

3.3.5 Microtext

Der Microtext ist ein einfacher ROM mit einer Adresse von 11 Bit und einer Wortlänge von 9 Bit, insgesamt also 2,25 KB. Einzige Erweiterung an dem ROM ist das Reset-Verhalten. Wird ein Reset durchgeführt, gibt der Microtext solange eine „0“ aus, bis der erste Wert gelesen wird. Die „0“ entspricht dabei einem NOP, also „no operation“. Der Microtext hat eine maximale Zugriffszeit von 2,56 *ns* bei einer Stromaufnahme von 4 $\mu A/MHz$ und einem Flächenbedarf

von 0,033 mm^2.

3.3.6 Microdata

Der Microdata ist ein ganz spezieller Speicher. Er enthält zum einen 104 ROM-Einträge, zum anderen aber auch 16 variable, beschreibbare Speicherplätze. Im FPGA wird auch für einen ROM ein BlockRAM-Modul genutzt, das mit festen Werten initialisiert ist. Im ASIC ist das nicht mehr möglich. Entweder alles ROM oder alles RAM, aber ohne vorhersehbaren Startwert. In einem ersten Versuch wurden dazu ein ROM- und ein Dualport-RAM-Modul miteinander so verbunden, dass bestimmte Adressen auf den RAM, die anderen Adressen auf den ROM umgeleitet wurden. Da für die benötigte Wortlänge von 33 Bit der kleinstmögliche Dual-Port-RAM 32 Einträge enthält, und der kleinstmögliche ROM 512 Einträge, wurde sehr viel Platz vergeudet. Nach einigen Anpassungen und Änderungen in der Ansteuerung des Microdata-Modules wurde für den beschreibbaren Teil nur noch ein 16 Einträge enthaltender Two-Port-RAM benötigt, der ROM-Teil wurde als großer MUX umgeschrieben. So wird vermieden, dass leerer, ungenutzter Speicher auf dem Chip implementiert werden muss. Der genutzte Two-Port-Speicher hat eine maximal Zugriffszeit von 1,23 *ns* bei einer Stromaufnahme von 8 $\mu A/MHz$ und einem Flächenbedarf von 0,027 mm^2.

3.3.7 Zusammenfassung der Speicher

In Tabelle 3.1 sind die physikalischen Eckdaten der Speicher noch einmal zusammen gefasst. Die angegeben Zeiten (Taa(WC) in der Tabelle) sind die Zugriffszeiten von der Taktflanke bis Ausgabe der Daten. Die Zeiten sind für den Fall des „worst case“ also unter den schlechtesten Bedingungen. Die Werte für die Stromaufnahme beziehen sich auf den Fall, dass pro Takt alle Ports beschrieben oder gelesen werden. Die Angaben für die Flächen der Macros beinhalten schon einen Ring von pauschal 10 μm um die Macros herum, der später für das Routing der Spannungsversorgung benötigt wird.

Speicher	**Taa(WC)**	**AC Power**	**Area**
Stack	3,7 *ns*	25 $\mu A/MHz$	0,453 mm^2
Methoden Cache	2,15 *ns*	19 $\mu A/MHz$	0,159 mm^2
Garbage Collector	1,3 *ns*	24 $\mu A/MHz$	0,148 mm^2
Microtext	2,56 *ns*	4 $\mu A/MHz$	0,033 mm^2
Microdata	1,23 *ns*	8 $\mu A/MHz$	0,027 mm^2
Summe		80 $\mu A/MHz$	0,82 mm^2

Tabelle 3.1: Zusammenfassung der RAM-Macros

3.4 Sonstige Anpassungen

Weitere Elemente und Strukturen, die in einem FPGA schon vorhanden sind, aber für einen ASIC erst angelegt werden müssen, sind IO-Zellen, Addierer, Multiplizierer und der Taktbaum. Der Taktbaum wird während des Place&Route-Designschrittes durch die Tools automatisch angelegt und optimiert. Durch verschiedene Parameter kann dies noch etwas beeinflusst werden. Addierer und Multiplizierer werden während der Synthese von den Tools automatisch angelegt und, abhängig von der zu erreichenden Taktfrequenz, mehr oder weniger stark optimiert.

Die IO-Pads sind von Hand in die HDL-Beschreibung einzufügen. Vom Anbieter der Standardzellen-Bibliothek werden Verschiedene zur Verfügung gestellt, die im Design zu instanziieren sind. Es gib drei Arten von IO-Zellen, Input, Output und bidirektional. Diese werden nochmals unterschieden in breitere und weniger tiefe Zellen für Core-limited Design und schmale, dafür aber tiefere Zellen für Pad-limited Design. Core-limited bedeutet, um den Logik Core haben mehr IO-Zellen Platz, als eigentlich benötigt werden, der Logik-Kern ist also für die Größe des gesamten Chip entscheidend. Beim Pad-limited Design benötigen die IO-Zellen soviel Platz, dass im Inneren mehr Logik Platz hätte, als eigentlich nötig ist. Die Anzahl und Verteilung der IO-Zellen ist also für die Größe des Chips entscheidend.

In der hier zugrunde liegenden Technologie haben die Pad-limited IO-Zellen ein Breite von 34,8 μm und eine Tiefe von 218,4 μm. Rechnet man mit einem moderaten Abstand von 80 μm zwischen zwei Bond-Pads (von Pad-Mitte bis Pad-Mitte gemessen) und legt man ein staggered Bond-Pad-Layout zugrunde, in dem die Bond-Pads in zwei Reihen jeweils um die Hälfte versetzt platziert werden, ergibt das einen mittleren Abstand zwischen zwei IO-Zellen von 40 μm. Wir haben 112 IO-Zellen zuzüglich mindestens sechs Zellen für die Spannungsversorgung. Für die Spannungsversorgung wird mindestens je eine IO-Zelle für die Core-Spannung von 1,2 V und zwei IO-Zellen für die IO-Spannung von 3,3 V zuzüglich der jeweiligen dazu gehörigen Masse-IO-Zellen. Von den zwei IO-Zellen für die IO-Spannung wird einer für die Eingangspuffer und Ausgangs-Vorstufen-Treiber gebraucht, und einer für die Ausgangs-Haupt-Treiber

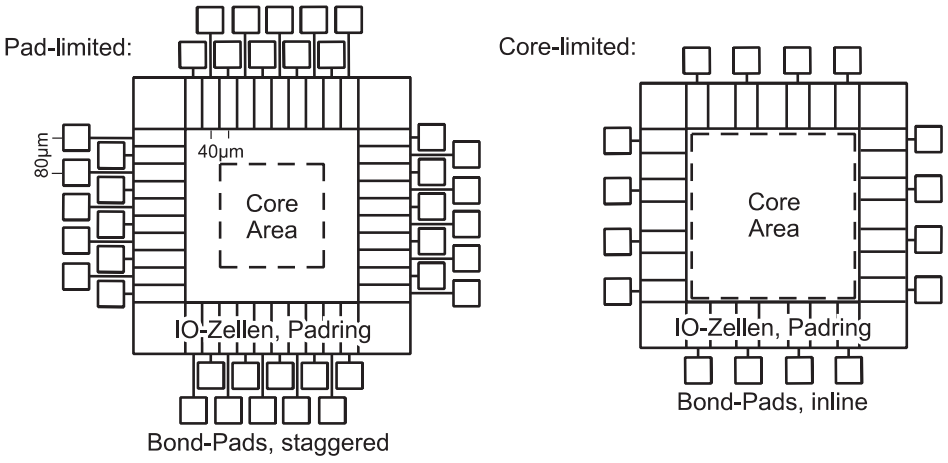


Abbildung 3.4: Vergleich zwischen Pad-limited und Core-limited Layout

und die Schutzschaltung gegen elektrostatische Entladungen. Insgesamt gibt es also mindestens 118 IO-Zellen. Im Pad-limited Design hätten wir somit einen inneren Umfang der IO-Zellen von 4720 μm. Bei einem quadratischen Layout wären das also 1180 μm zum Quadrat für die Core-Fläche, also rund 1,39 mm^2. Für die gesamte Chipfläche muss man auf allen Seiten noch die Tiefe der IO-Zellen und zwei Reihen Bond-Pads von je 80 μm und etwas Platz für das Routing hinzurechnen. Man kommt so auf 1940 μm zum Quadrat, also einer Gesamtfläche von 3,76 mm^2.

Die IO-Zellen für das Core-limited Layout haben eine Breite von 60 μm und eine Tiefe von 152 μm. Würde man wieder das staggered Bond-Pad-Layout nutzen, könnte man diese Zellen ohne Lücke dicht zusammen packen. Der innere Umfang wäre somit 7080 μm, bei einem quadratischen Layout also 3,13 mm^2. Die Gesamtfläche wäre somit inklusive IO-Zellen und zwei Reihen Bond-Pads 5,73 mm^2. Als Alternative wäre auch ein inline Bond-Pad-Layout möglich, wo die Bond-Pads nur in einer Reihe angeordnet sind. In diesem Fall wäre aber ein mittlerer Abstand von 80 μm zu nutzen. Der innere Umfang wäre dann 9440 μm, was zu einer Core-Fläche von 5,57 mm^2 führen würde. Inklusive IO-Zellen und einer Reihe Bond-Pads ergibt sich eine Gesamtfläche von 7,97 mm^2. Da die RAM-Macros mit einer Gesamtfläche von 0,82 mm^2 vermutlich den größeren Teil des Cores belegen werden (siehe Abschnitt 2.3), erscheint nur die Variante mit den schmalen IO-Zellen für Pad-limited Design sinnvoll. Deswegen hab ich diese Art der IO-Zellen instanziiert. Die IO-Zellen sind frei konfigurierbar, ich habe die Konfigurations-Pins so belegt, dass die Ausgangs-IO-Zellen eine Treiberstärke von 8 *mA* haben und die „Slew Rate“ auf schnell eingestellt ist. Für die Eingangs-IO-Zellen sind Schmitt-Trigger, Pull-Up- und Pull-Down-Widerstände deaktiviert.

4 Synthese

4.1 Einflussnahme über Constraints

Sind alle nicht synthetisierbaren Module durch entsprechende Macros und Instanziierungen ersetzt, kann die eigentliche Synthese erfolgen. Dazu habe ich den „Design Compiler“ von Synopsys genutzt. Während der Synthese übersetzt das Tool den VHDL-Code in einen Logikschaltplan, und bildet diese Logik auf die vorhandenen Standardzellen ab. Standardzellen können einfache Logikfunktionen ausführen wie Inverter, NAND- oder OR-Gatter. Aber auch komplexere Funktionen wie kombinierte AND-OR-Gatter, Multiplexer, 1-Bit-Volladdierer und Register sind in der Standardzellenbibliothek vorhanden. Aus diesen Standardzellen wird letztendlich von dem Synthese-Tool eine Netzliste erstellt. Das Ergebnis ist natürlich nicht fest, sondern abhängig, nach welchen Parametern optimiert werden soll.

Das Grundprinzip sieht so aus, dass die Taktfrequenz in Form der Periodenlänge vorgegeben wird, und der Compiler die Logik so optimiert, dass die Taktfrequenz gerade so erreicht wird, dabei aber so wenig wie möglich Fläche benötigt wird, und die Leistungsaufnahme so gering wie möglich gehalten wird. Dabei gibt es vielfältige Möglichkeiten den Takt so realistisch wie möglich zu beschreiben. Dies dient dazu, schon vor dem „Place & Route“ zu berechnen, welches realistische zeitliche Verhalten sich später einstellen wird. So kann man unter anderem Latenzen angeben, die der Taktbaum später voraussichtlich haben wird. Leider beeinflussen diese Einstellungen auch die finalen Timing-Berechnungen nach dem „Place & Route“-Schritt, so dass es besser ist, mit einem ideal angenommen Takt weiter zu arbeiten.

Es hat sich herausgestellt, dass es günstiger ist, erst einen Takt zu modellieren, um das Syntheseverhalten zu beurteilen und realistische Timing-Berichte zu bekommen, für die finale Synthese aber nur einen idealen Takt vorzugeben. Die eigentlichen Takeverzögerungen ergeben sich erst mit dem „Place & Route“ durch die Generierung des Taktbaumes. Den einzigen zusätzlichen Wert, den ich im weiteren Verlauf mitgenommen habe ist eine „Taktunschärfe“ von 0,1 ns.

Es gibt standardmäßig drei Pfad-Gruppen, die für das Timing betrachtet werden, Register zu Register, Input zu Register und Register zu Output. Theoretisch kommt auch noch Input zu Output hinzu, aber diese Pfadgruppe gibt es im SHAP nicht. Für die Pfadgruppen Input zu Register und Register zu Output gibt es jeweils eine große Unbekannte: die Signallaufzeit außerhalb des Chips. Für den Input-Pfad kann eine lange externe Verzögerung, von fast der ganzen Taktlänge geplant werden, da diese Zeit für die externe Peripherie, wie zum Beispiel SRAM zu Verfügung stehen muss. Innerhalb des Chips müssen nur die Setup- und Hold-Zeiten

des Eingangsregisters eingehalten werden. Ich habe pauschal Werte von der Taktperiode, um 0,5 *ns* reduziert, angenommen.

Für den Outputpfad müssen wesentlich kürzere Laufzeiten eingeplant werden. Zum einen haben die Ausgangstreiber sehr hohe Schaltzeiten, die schon, abhängig von Treiberstärke und angehängter Kapazität, von anderthalb *ns* bis hin zu einer halben Taktperiode dauern können. Zum anderen müssen die Signale auch erst von den Ausgangsregistern zu den eigentlichen Ports gerouted werden. Außerhalb des Chips müssen dann aber nur noch die Setupzeiten der peripheren Geräte eingehalten werden. Ich habe eine pauschale externe Laufzeit von 0,5 *ns* verwendet.

Ein weiterer wichtiger Punkt für die Optimierung des Timings ist das „auto-ungrouping“. Dabei werden kleinere Module die im zeitlich kritischen Pfad liegen in die nächst höhere Hierarchie aufgelöst und dort in das Logiknetzwerk integriert. Das hat den Vorteil, dass sich ein besseres zeitliches und flächenmäßiges Ergebnis erreichen lässt. Der Nachteil ist aber, das sich diese Module bei späteren Simulationen nicht wieder finden lassen, und so eine Fehlersuche extrem erschwert wird.

Ein weiteres Kriterium, nach dem optimiert werden soll, ist die Leistungsaufnahme. Unterschieden wird nach dynamischer und statischer Verlustleistung. Dynamische Verlustleistung wir nur währen des Umschaltens von Signalen aufgenommen. Statische Verlustleistung wird dauerhaft aufgenommen, hauptsächlich in Form von Leckströmen. In der CMOS-Technik ist die dynamische Verlustleistung der Hauptfaktor, da diese auch stark von der Frequenz abhängig ist. In vereinfachter Annahme kann man davon ausgehen, dass bei doppelter Frequenz auch die doppelte Leistung aufgenommen wird. Es gilt somit auch, je häufiger sich ein Signal ändert, desto mehr trägt es zu der Gesamtverlustleistung bei.

Da der Taktbaum einen erheblichen Anteil an Standardzellen im Gesamtentwurf einbringen wird, und sich der Takt zudem auch zweimal pro Taktperiode ändert, ist dort auch der größte Teil an Leistungsaufnahme zu erwarten. Es ist demzufolge angebracht an diesem Punkt zu optimieren. Am Ende eines jeden Zweiges des Taktbaumes steht ein Register, oder der Takteingang eines RAM-Macros. Aber nicht jedes Register ändert bei jeder Taktflanke seinen Wert. Viele Register machen ihre Änderung von einem Enable-Signal abhängig, das eine Änderung zulässt, oder eben auch nicht. Oft sind es ganze Registerbänke für Bus-Signale die auf ein Enable-Signal reagieren. Es ist also unnütz den ganzen Taktbaum für diese Register aktiv lassen, wenn sich am Register kein Signal ändern soll.

Genau an dieser Stelle setzt das so genannt Clock-Gating an. Der benötigte Teil des Taktbaumes wird nur dann aktiviert, wenn sich auch der Inhalt der Register ändern soll. Dazu wird ein so genanntes Clock-Gating-Latch in den Taktbaum gesetzt, das Taktflanken nur dann weiter gibt, wenn das Enable-Signal aktiv ist. Auf diese Weise sind ganze Teile des Taktbaumes inaktiv, wenn diese nicht benötigt werden, und nehmen demzufolge auch keine dynamische Verlustleistung auf. Diese Aufgabe des Platzierens von entsprechenden Clock-Gating-Latches übernimmt, bei geeigneter Konfiguration, auch der Design-Compiler. Er erkennt die Enable-Strukturen und kann den Taktbaum in Teilbäume aufteilen und entsprechend die Latches einfügen. Das Script ist so konfiguriert, dass es nur dann Clock-Gating-Elemente einfügt, wenn mindestens drei Re-

gister von einem Enable-Signal abhängig sind.

4.2 Ergebnisse

Welche Ergebnisse hat nun die Synthese gebracht? In Abbildung 4.1 ist der Flächenbedarf und die Leistungsaufnahme in Abhängigkeit der zu erreichenden Taktfrequenz dargestellt. Die Werte sind natürlich nur geschätzte Werte. Zum einen ist die Fläche nur eine Summe der Flächen der Standardzellen und RAM-Macros, es ist also noch keine Verteilung der Zellen und das Routing mit einberechnet. Zum andern ist auch die Leistungsaufnahme nur eine Summe der Werte der einzelnen Netzlistenelemente. Es ist weder der Taktbaum, der erst im „Place & Route“-Schritt erstellt wird, noch ist der Einfluss des Routing beinhaltet. Auch ist die Leistungsaufnahme sehr von der Art des abgearbeiteten Programms abhängig. Die Werte, wie oft welches Signale umschaltet, sind also sehr spekulativ und haben wenig mit der späteren „Realität“ zu tun. Ein paar wichtige Erkenntnisse können wir aber aus der Synthese mitnehmen.

Aus dem Diagramm ist gut zu erkennen, dass sich die Fläche mit steigender vorgegebener maximaler Taktfrequenz nur unwesentlich erhöht. Auch ist der Anteil an wirklicher kombinatorischer Logik nur im Bereich von etwa 10 Prozent am Gesamtanteil der Fläche. Dass heißt, die RAM-Macros, die einen Großteil der nichtkombinatorischen Fläche einnehmen, haben, wie

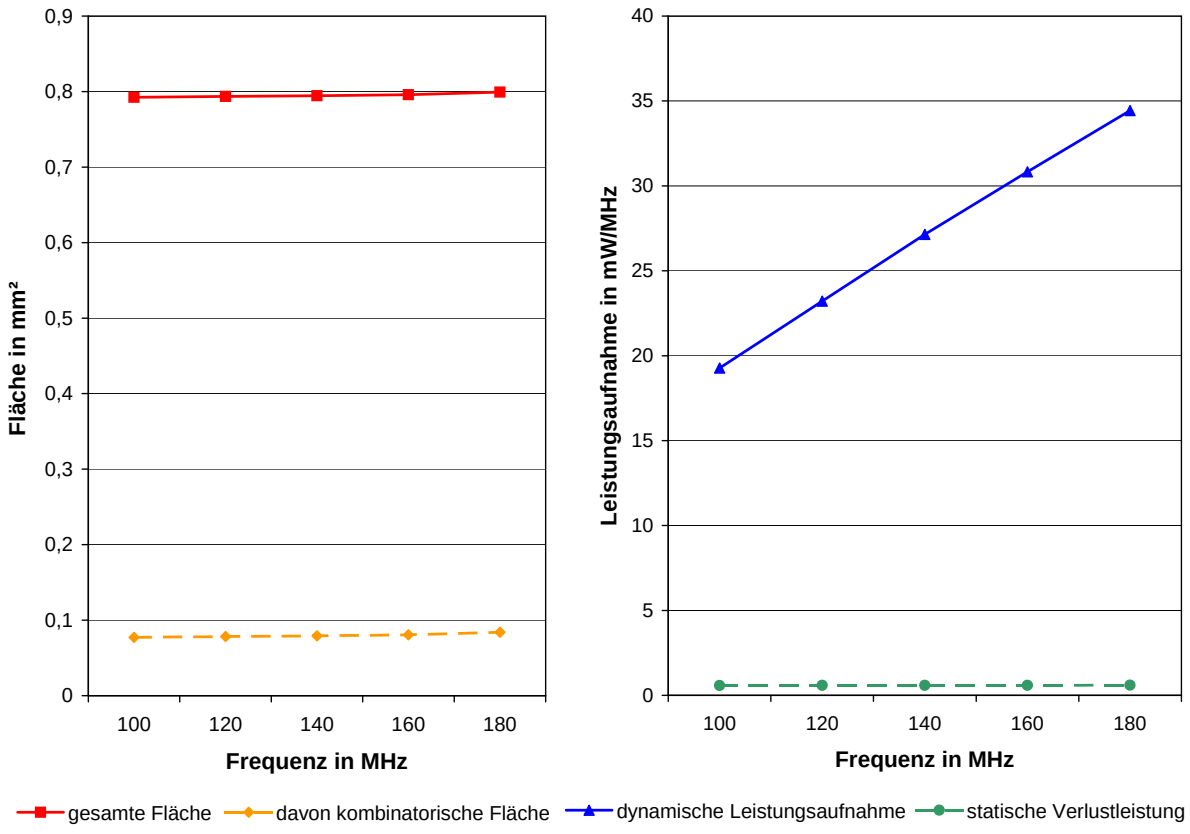


Abbildung 4.1: Fläche und Leistungsaufnahme in Abhängigkeit der Taktfrequenz

schon in Abschnitt 2.3 angedeutet, auch den Hauptanteil an der Gesamtfläche.

Auch wenn die Werte der Leistungsaufnahme nur spekulativ sind, erkennt man deutlich, dass die dynamische Leistungsaufnahme linear von der Taktfrequenz abhängt. Das ist auch zu erwarten, da die Leistungsaufnahme direkt davon abhängt, wie oft die Signal zwischen „0“ und „1“ wechseln. Bei doppelter Taktfrequenz erfolgt das logischer weise doppelt so oft, weswegen auch die Leistungsaufnahme doppelt so hoch ist. Innerhalb der Standardzellen und RAM-Macros fließen aber auch Leckströme, die, unabhängig von der Taktfrequenz, immer vorhanden sind, und zu geringen Verlusten führen. Diese fallen aber mit knapp 0,6 mW sehr gering aus. Aufgrund der Tatsache, das es in der Fläche kaum Abweichungen gibt, und die Leistungsaufnahme linear von der Taktfrequenz abhängt, führen ich das „Place & Route“ mit der maximal möglichen Taktfrequenz aus.

Das Clock-Gating konnte in großem Umfang realisiert werden. Von 3270 Registern ist für 2999 Register das Taktsignal abschaltbar. Es können somit 91,7 % der Register vom Takt getrennt werden. Dafür wurden 204 Clock-Gating-Elemente eingefügt.

4.3 Probleme

Erste Simulationen der synthetisierten Netzliste schlugen fehl. Wie sich herausstellte gab es Probleme beim Reset der Schaltung. Einige Register wurden beim Reset nicht richtig zurückgesetzt. Zum einen weil sie laut VHDL-Code auf „don’t care“ zurückgesetzt wurden, zum anderen, weil die Logikoptimierung, das Signal nicht als Reset erkennt und als normales Signal in die Kombinatorik einfließen lässt. Das führte bei einigen wenigen Registern zu Fehlern beim Reset. Das „don’t care“-Verhalten wurde vom Synthese-Tool entfernt, die Register blieben in einem ungültigen Zustand. Es gibt aber Compiler-Anweisungen, die man in den HDL-Code einbauen kann, die dann definieren, welches Signal als Reset zu nutzen ist. Diese Compiler-Anweisung habe ich für alle resetbaren Register genutzt.

Aber auch in den RAM-Macros gab es Probleme. Im FPGA sind die RAM-Macros standardmäßig mit „0“ initialisiert, im ASIC sind sie aber nur undefiniert. So kam es vor, dass der Garbage Collector beim Überprüfen des Speichers undefinierte Werte ausgelesen hat, und so der ganze Schaltkreis in einen unbestimmten Zustand gelangt ist. Dieses Problem trat beim Microdata und beim Stack auf. Der Microdata ist noch ein relativ kleiner Speicher mit nur 16 Speicherplätzen. Die Lösung dafür ist relativ einfach, es wurde quasi ein Statusregister zu jedem Speicherplatz hinzugefügt, in dem gespeichert wird, ob der Platz beschrieben ist, oder nicht. Wird ein Wert aus dem Speicher gelesen, wird gleichzeitig in das Statusregister geschaut, und, wenn der Platz nicht beschrieben ist ein „0“-Wert ausgegeben. Anders sieht es beim Stack aus. Mit 2048 Einträgen ist er zu groß, um für alle Einträge ein Statusregister bereit zuhalten und diese auch zeitnah zu adressieren, um den Status parallel zum Speichern im Stack zu verändern. Der Steuerungsautomat des Stack enthält aber schon eine Reset-Phase, welche dann dafür genutzt wurde, um beim Starten des SHAP-Controllers, den Stack mit „0“-Werten zu initialisieren.

Ein weiterer Fehler der nicht geklärt werden konnte war vermutlich auf Fehler im Compiler zurück zu führen. Eine Simulation mit einer auf 100 MHz compilierten Netzliste lief durch, bei 175 MHz gab es nach einer Laufzeit von weit über 500.000 Takten Rechenfehler, deren Ursache nicht abschließend geklärt werden konnten. Nach Änderungen am Server und unter Nutzung einer neueren Version des Design Compilers trat der Fehler nicht mehr auf.

5 Place & Route

5.1 Vorgehensweise

Nach der erfolgreichen Synthese einer Netzliste aus dem Quellcode steht nun der wichtigste Arbeitsschritt, das „Place & Route“ an. Dieser Arbeitsschritt unterteilt sich in das Floorplanning, Placing, Clock Tree Specification (CTS), Routing und finale Arbeitsschritte. Vor und nach der CTS-Phase und nach dem Routing erfolgen noch Optimierungsphasen des Timings. Die Phasen werden scriptgesteuert durch das Tool „Cadence Encounter“ abgearbeitet. Durch das modifizieren der Scripte kann man das Verhalten des Tools und damit das Ergebnis beeinflussen.

Die Phase des Floorplanning ist die Phase mit den meisten Möglichkeiten der Beeinflussung. In dieser Phase wird nicht nur festgelegt, welche Ausmaße der Chip hat und wo die einzelnen IO-Zellen und die RAM-Macros liegen, sondern auch das Powerplanning erfolgt in diesem Schritt, dass heißt es wird das vollständige Routing der Versorgungsspannung (V_{cc}) und der Masse (*Gnd*) durchgeführt.

Doch zu allererst werden die Ausmaße des Chips festgelegt. Da, wie im Abschnitt 3.4 schon beschrieben, die Pad-Zellen entscheidend für die Ausmaße sein werden, ist der Umfang des Cores vorgegeben. Bei gegebenen Umfang wäre das Quadrat der größte Flächeninhalt, dass heißt je größer das Seitenverhältnis, desto kleiner die Fläche bei gleichem Umfang. Da Fläche in der Produktion Geld kostet, ist man natürlich bestrebt, so wenig wie möglich Platz zu verbrauchen. Aus diesem Grund sind die Ausmaße so gewählt, das der größte Speicher, der Stack, gerade so noch von der Höhe her in den Core passt. Ein detaillierter Floorplan ist im Anhang A.1 zu sehen.

Um einen Anschluss aller RAM-Macros und Standardzellen an die Spannungsversorgung zu gewährleisten, werden um den Core und die RAM-Macros so genannte Power-Ringe gezogen. Dabei wird je eine Leitung der Versorgungsspannung und der Masse um den Core und um jedes RAM-Macro gezogen. Das braucht natürlich auch Platz. Aus diesem Grund werden alle RAM-Macros an den Rand des Cores gelegt. So fallen die Power-Ringe des Cores und der Macros zusammen, was letztendlich Platz spart.

Ein wichtiger Punkt bei der Planung dieser Spannungsversorgungsstrukturen ist der so genannte IRdrop. Das ist ein Wert für den Spannungsabfall über den Versorgungsleitungen und berechnet sich aus der Stromstärke I und dem Leitungswiderstand R. Ist der Spannungsabfall über der Versorgungsleitung zu groß, wird die Spannung an den Standardzellen und RAM-Macros zu klein. Damit wäre die richtige Funktionsfähigkeit nicht mehr gewährleistet. Um den IRdrop zu senken muss man entweder die Stromstärke I oder den Widerstand R senken.

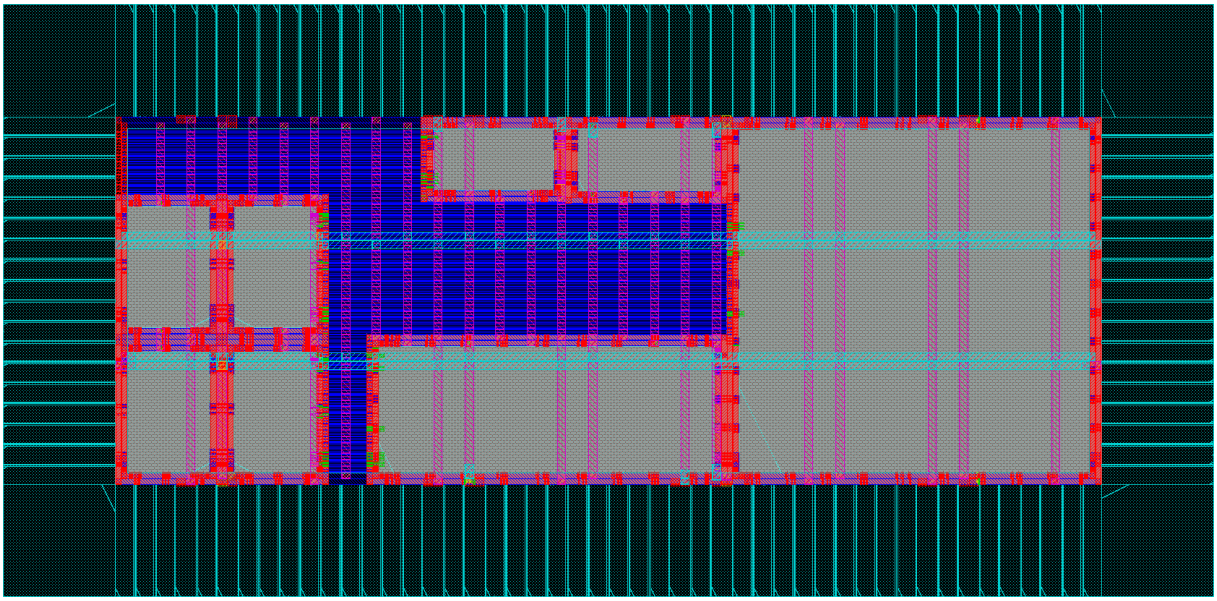


Abbildung 5.1: Floorplan mit fertig geroutetem Power-Netz

Die Stromstärke lässt sich nicht beeinflussen, aber durch mehrere Spannungsversorgungs-Pads kann der Stromfluss auf mehrere Wege verteilt werden, so dass die Stromstärke in einer Leitung geringer ist. Zu dem kann man den Leitungswiderstand an sich verkleinern. Der Leitungswiderstand berechnet sich zu $R = \rho \frac{l}{A}$. An der Material-Konstante ρ kann man nichts ändern, aber am Leitungsquerschnitt A und an der Länge l. Da die Stärke der Leitungsbahnen von der Technologie vorgegeben ist, kann man nur über die Breite der Leitungen Einfluss nehmen. Die Länge der Leitungen kann man zum einen verkürzen, indem man mehr Spannungsversorgungspads vorsieht, so wird der Weg zum nächsten Pad kürzer. Zum andern kann man auch ein zusätzliches Spannungsversorgungsnetz über die Strukturen legen. Dazu werden Leitungen in höheren Verdrahtungsebenen, so genannte Power-Stripes, orthogonal über den Chip gezogen und an Kreuzungspunkten mit den tiefer liegenden direkten Zuleitungen zu den einzelnen Zellen verbunden. Die direkten Zuleitungen der Standardzellen sind sehr schmal und haben somit einen hohen Widerstand. Das obere Spannungsnetz hat einen wesentlichen geringeren Widerstand, da die oberen Metallebenen dicker sind als die niedrigen und außerdem auch viel breiter gezogen werden können.

Für die weiteren Schritte nach dem Floorplanning habe ich die Scripte nur in sofern verändert, dass ein möglichst hoher Erfolg beim Erreichen des gewünschten Timings zustande kommt. Ansonsten verlaufen die weiteren Arbeitsschritte des „Place & Route“ weitgehend automatisch.

5.2 Ergebnisse

Beim „Place & Route“ waren die 180 MHz von der Synthese nicht mehr ganz zu halten. Im Endergebnis werden aber immer noch gute 175 MHz erreicht. Die Gesamtfläche des Chips inklusive der IO-Zellen, aber ohne Bond-Pad-Strukturen beträgt 2355,6 μm mal 1151,6 μm,

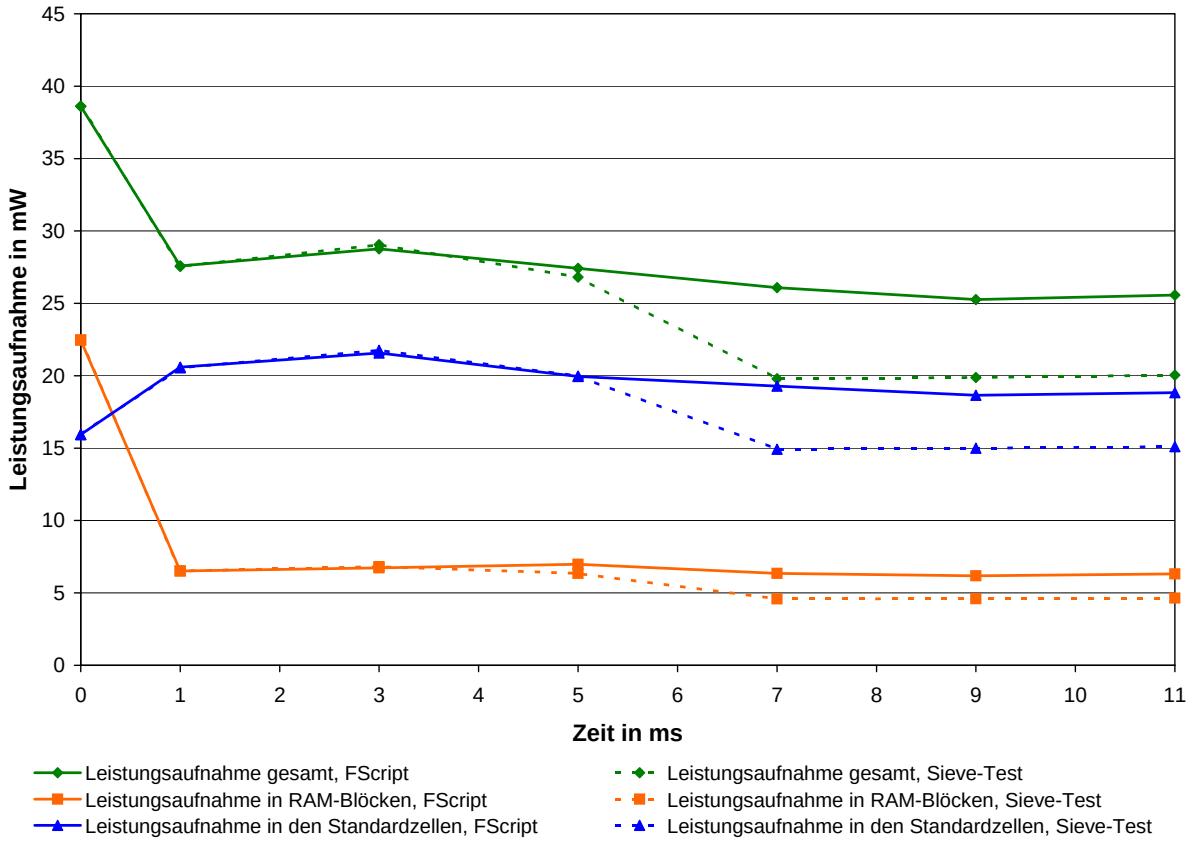


Abbildung 5.2: Leistungsaufnahme im zeitlichen Verlauf

also 2,71 mm^2. Die Fläche des reinen Cores, ohne Pad-Zellen beträgt 1918,8 μm mal 714,8 μm, also 1,37 mm^2. Die Pad-Zellen belegen demnach in etwa genau soviel Fläche, wie der Core. Für die Standardzellen steht eine Fläche von 0,288 mm^2 zur Verfügung. Daraus errechnet sich, dass 79% Der Core-Fläche für die RAM-Macros und die Powerringe benötigt werden, und 21% für die Standardzellen zur Verfügung stehen. Da die Standardzellen nur mit einer mittleren Dichte von 82% verteilt sind, belegen sie insgesamt sogar nur 0,236 mm^2, also nur 17% der Core-Fläche.

Ein weiterer wichtiger Punkt in der Auswertung der Ergebnisse ist die zu erwartende Leistungsaufnahme. Im Abschnitt 4.1 habe ich ja schon die Einsparmöglichkeiten des Clock-Gating erläutert. Nach dem „Place & Route“ kann man die realen Vorteile dieses Verfahrens berechnen, indem man einmal die Leistungsaufnahme eines Layouts ohne Clock-Gating berechnet, und einmal mit aktiviertem Clock-Gating. Dazu habe ich die durchschnittliche Leistungsaufnahme von 1,0 ms bis 1,1 ms nach dem „Anschalten“ des Chips berechnet. Ohne Clock-Gating lag die mittlere Leistungsaufnahme bei 34,43 mW, mit Clock-Gating nur noch bei 27,59 mW. Die Leistungsaufnahme wurde durch das Clock-Gating also um etwa 20% verringert. Interessanter Nebeneffekt, die Dichte der Standardzellen fällt von 84,4 % auf 82 %. Es lässt sich also auch geringfügig Platz sparen.

In Abbildung 5.2 ist eine grafische Visualisierung der Leistungsaufnahme dargestellt. Dafür

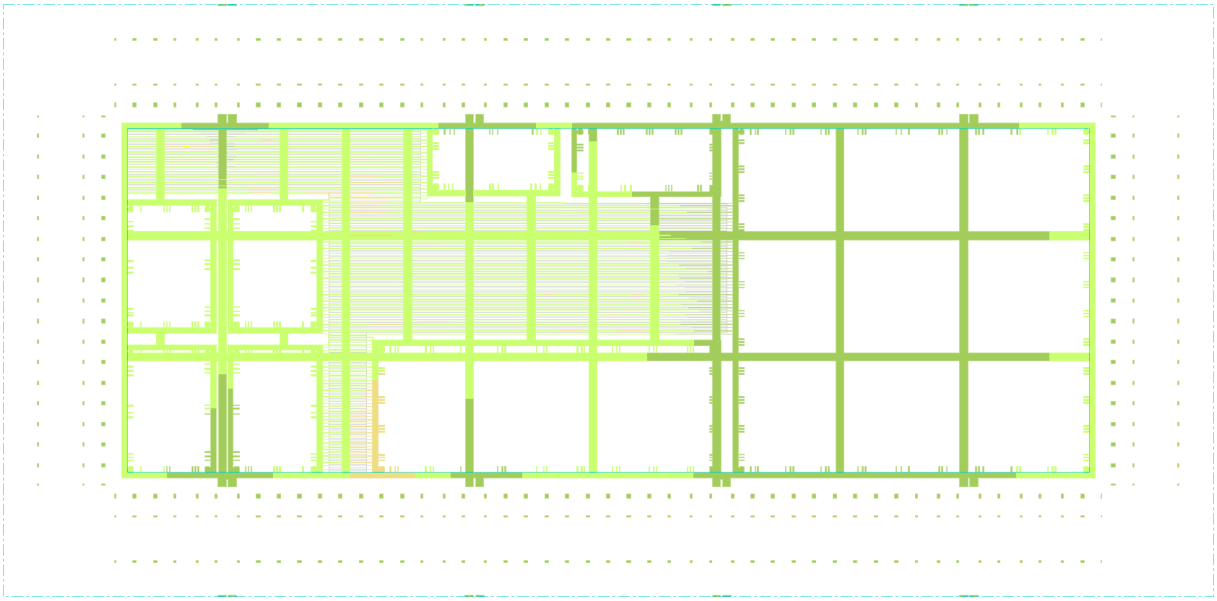


Abbildung 5.3: grafische Verteilung des IRdrop von 0 *mV* (grün) über 50 *mV* (gelb) bis 120 *mV* (rot, nicht genutzt)

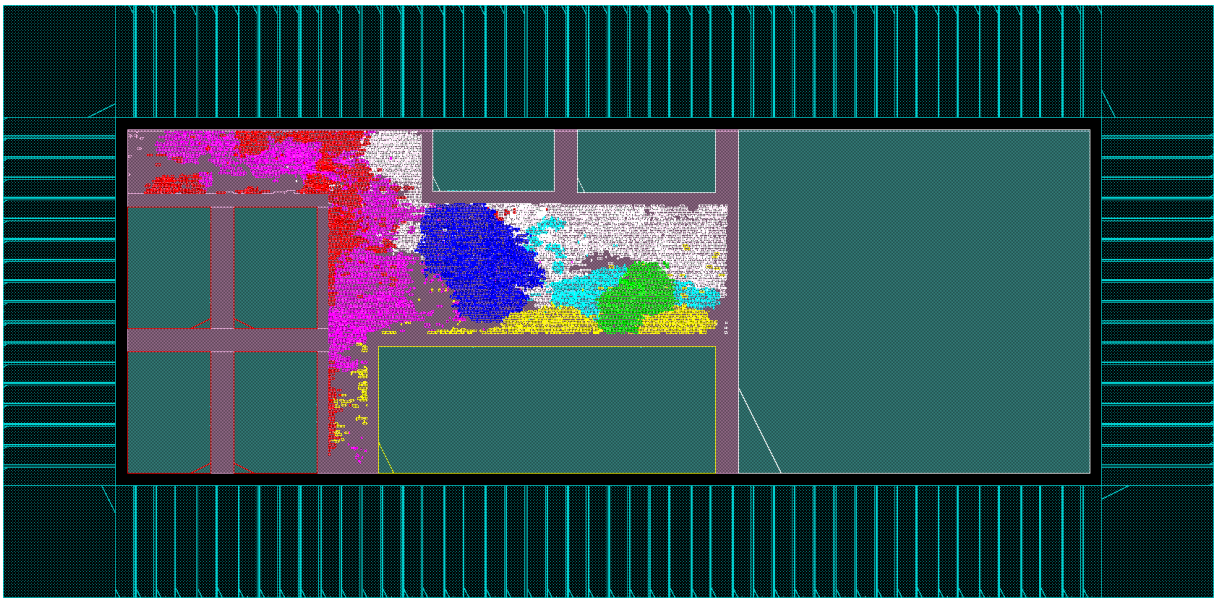


Abbildung 5.4: Verteilung der Einzelkomponenten: CPU (weiß/blau/türkis/grün) mit ALU (blau), Dividierer (türkis) und AMem des Stacks (grün); Methoden Cache (gelb); MMU (violett/rot) mit Garbage Collector (rot)

Setup-Pfad	reg2reg	in2reg	reg2out	clkgate
schlechtester „Slack“[a]	0,152 *ns*	0,734 *ns*	0,021 *ns*	0,245 *ns*
Anzahl Pfade	4306	110	67	191

[a] maximale mögliche Signallaufzeit minus errechnete Signallaufzeit

Tabelle 5.1: Ergebnisse der Timinganalyse nach der letzten Optimierung

wurde zu den Messpunkten die durchschnittliche Leistungsaufnahme über die darauf folgenden 100 μs dargestellt. Die Messungen wurden zweimal ausgeführt, einmal mit dem Sieve-Test aus dem Caffeine Benchmark, und einmal mit einem FScript-Test. Der Sieve-Test sind sehr rechenintensiv, der FScript-Test beansprucht die internen Speicher und den Heap sehr stark. Direkt nach dem Start ist die Leistungsaufnahme besonders hoch. Das lässt sich damit begründen, dass am Anfang alle Register erst einmal initialisiert werden müssen, also im gesamten Chip Schaltvorgänge stattfinden, was sich dann in der Stromaufnahme bemerkbar macht. Bis etwa 6 *ms* dauert die Boot-Phase wo die Programminstruktionen über die externe Schnittstelle eingelesen werden. Diese Phase ist stark optimiert, um möglichst schnell das Programm starten zu können, was sich in einer Leistungsaufnahme von bis zu etwa 30 *mW* bemerkbar macht. Wenn das Programm läuft, beträgt die mittlere Leistungsaufnahme je nach ausgeführtem Programm 20 - 25 *mW*. Der darin enthaltene Anteil der durch Leckströme verursachten Leistungsaufnahme beträgt konstant 0,157 *mW*. Die hier angegebene Leistungsaufnahme ist aber nur die des Cores. Die Pad-Zellen sind nicht mit eingerechnet, da sie ein Großteil der Leistung auch über die 3,3 *V* Leitung beziehen. Die Leistungsaufnahme der Pad-Zellen ist auch abhängig von den später angeschlossenen externen Geräten. Im Datenblatt sind pauschale Werte von 0.385 $\mu W/MHz$ für die Eingangs-Pads und 15,87 $\mu W/MHz$ für die Ausgangs-Pads angegeben. Es gibt insgesamt 44 Eingangs- und 68 Ausgangs-Pads. Rechnet man mit einer, pessimistisch angenommen, durchschnittlichen Signaländerung aller 10 Takte an den Pads ergibt sich eine zusätzliche Leistungsaufnahme für die IO-Pads von 19,2 *mW*.

Zusammen mit der Analyse der Leistungsaufnahme wird auch der maximale IRdrop berechnet. Dieser beträgt im finalen Layout 52,4 *mV*. Die benötigte Spannung für den Core liegt bei 1,2 *V*. Der kleinste erlaubte Wert liegt bei 1,08 *V*. Der maximale IRdrop darf also höchstens 120 *mV* betragen. Von diesem Wert erreichen wir nicht einmal die Hälfte, die hier berechneten 52,4 *mV* sind also ein sehr guter Wert. In Abbildung 5.3 ist eine Verteilung des IRdrop über den Chip grafisch dargestellt. Um diesen Wert zu erreichen wurden je acht Spannungsversorgungs-Pads für die 1,2 *V* des Cores und der dazugehörigen Masse benötigt. Diese sind gleichmäßig an der oberen und unteren Seite des Chips verteilt, die genaue Anordnung der Pads ist im Anhang A.1 zu sehen. Alle Power-Pads sind so angeordnet, das sie bei der Platzierung der Bond-Pads mit der äußeren Bond-Pad-Reihe verbunden werden können. Das ist nötig, um beim Bonden, dem Verbinden von Chip und Chip-Träger, ein Kreuzen, und somit ein Kurzschluss von Leitungen zu verhindern, da die Spannungsversorgung auf dem Chip-Träger in der Regel innen die Leitungsstrukturen hat, die IO-Pins aber außen. Die Power-Ringe haben eine Breite von 10,6 μm,

die Power-Stripes sind 16 μm breit, das Power-Netz ist in Abbildung 5.2 dargestellt.

In Abbildung 5.4 ist eine beispielhafte Übersicht, welche Module wo platziert wurden. Der in der Abbildung grün markierte AMem des Stack ist ein „read first“-Dual-Port-Speicher mit 32 7-Bit Wörtern.

In Tabelle 5.1 sind die Werte für das Timing nach der letzten Optimierung angegeben. Der kritischste Pfad zwischen zwei Registern führt vom Datenausgang des Microtext zum Adresseingang des Microtext, über einen Logik-Pfad durch die decode- und fetch-Module. Bei den Eingängen stehen sehr große Reserven zur Verfügung, dafür fallen diese bei den Ausgängen sehr knapp aus. Die kritischsten Pfade sind dort am Boot-Interface zu finden.

5.3 Probleme

Für die Platzierung der Standardzellen ist es mit entscheidend, wo die RAM-Macros platziert sind. Zum einen sind die Pins der Macros mit dem Netz der Standardzellen verbunden und entscheiden so mit, wo einzelne Module der Logik platziert werden. Zum andern, entscheidet die Lage der Macros aber auch, wo überhaupt noch Platz zur Platzierung der Standardzellen ist. Dabei hat sich heraus gestellt, dass das Platzieren in den meisten Fällen nur dann erfolgreich ist, wenn die zur Verfügung stehende Fläche ein größerer zusammenhängender Bereich ist. Wurden die Macros so platziert, dass zwischen dem Rand des Cores und den Macros, oder zwischen zwei Macros ein schmaler Streifen entstand, auf dem die Zellen zu platzieren waren, gab es sehr oft Routing- oder Timing-Probleme. Es stand nicht mehr genügend Platz für Buffer des Taktbaumes oder zum Bereinigen von Verletzungen der Hold-Zeiten zur Verfügung.

Auch der IRdrop bereitete einige Probleme. Nach ersten Berechnungen sah der Wert mit 0,1 V noch gut aus. Die Berechnung beruhte auf einer statistischen Analyse, nicht auf einer per Netzlisten-Simulation berechneten Analyse. Zu dem war der errechnete Wert nur der zeitliche Durchschnitt, nicht der maximal Spitzenwert, der für die Funktion des Chips aber im Zweifelsfall entscheidend ist. Nach der ersten Analyse des maximalen IRdrops mit einer auf Simulationsdaten beruhenden Analyse stellte sich ein Wert von über 5 V ein. Das ist natürlich ein extrem zu hoher Wert. Um den Wert auf ein verträgliches Maß zu bekommen mussten die Power-Ringe und das obere Netz aus breiten Spannungsversorgungsleitungen erweitert werden. Zudem wurde die Anzahl an Spannungsversorgungs-Pads für den Core auf acht Stück erhöht. Am Ende brachte die Verbindung der senkrechten Power-Stripes mit zwei Waagerechten den Erfolg.

6 Auswertung

Nun stellt sich die Frage, wie die Ergebnisse zu bewerten sind. Dazu will ich die ermittelten Werte der SHAP-Architektur mit den Werten der Tabelle 2.1 aus Abschnitt 2.1 vergleichen. Dazu ist aber erste einmal zu ermitteln, wie viel CaffeineMark-Punkte die SHAP-Architektur erreichen kann. Ein Durchlauf des Caffeine Benchmark dauert mehrere Sekunden, da die Simulation einer Sekunde schon über 10 Tage dauert, würde eine komplette Simulation des Caffeine Benchmarks mehrere Monate dauern. Aus diesem Grunde nehme ich die auf verschiedenen FPGAs ermittelten Werte. Da der Float-Test auf der SHAP-Architektur nicht läuft, stehen nur die Werte der einzelnen Tests zur Verfügung. In Tabelle 6.1 sind diese aufgelistet.

Der Gesamtwert errechnet sich zu $exp[\frac{1}{n}\sum_{i=1}^{n}\ln(Testwert_i)]$. Das entspricht dem geometrischen Mittel der Einzelwerte. Die Sparten-3 FPGAs laufen mit 50 MHz der Virtex-5 mit 66,7 MHz. Auf dem Virtex-5 sind theoretisch bis zu 80 MHz möglich, was aber noch nicht einmal der Hälfte der Taktfrequenz entspricht die die ASIC-Variante hier erreichen würde. Um jetzt auf einen Wert in CM/MHz zu kommen habe ich die jeweiligen Gesamtwerte aus Tabelle 6.1 durch die jeweilige Taktfrequenz dividiert und den arithmetischen Mittelwert aus allen drei Werten ermittelt. Es ergibt sich ein Wert von 4,7 CM/MHz.

In Tabelle 6.2 sind noch einmal die Werte aus Abschnitt 2.1 aufgelistet, und die neu ermittelten Werte für die SHAP-Architektur hinzugefügt. Bei der Taktfrequenz können, in Anbetracht der Technologie, nicht ganz die Werte anderer Prozessoren erreicht werden. Der AT32AP7000 erreicht mit einer 0,13-μm-Technologie 200 MHz. Im Leistungsvergleich über den Caffeine Benchmark kann die SHAP-Architektur durchaus mithalten. Die Werte schwanken zwischen 4,2 CM/MHz beim Spartan-3E und 5,18 CM/MHz beim Spartan-3. Mit durchschnittlich 4,7 CM/MHz ist dass ein guter Wert. Es gibt aber auch einen gravierenden Nachteil. Die SHAP-Architektur kann den Float-Test nicht ausführen, da noch nicht alle Funktionen vollständig

FPGA-Board	**Spartan-3**	**Spartan-3E**	**Virtex-5 ML505**
Sieve-Test	155	124	189
Loop-Test	193	136	221
Logic-Test	336	328	445
String-Test	442	348	535
Method-Test	262	211	322
Gesamtwert	259	210	317

Tabelle 6.1: Ergebnisse des embedded CaffeineMark auf verschiedenen FPGA-Plattformen

CPU	Technologie	Taktrate	Benchmark[a]	Leistungsaufn.	Die Size
aJile aj-100	0,25 μm	100 MHz	2,75 CM/MHz	2,57 mW/MHz	
Fujitsu MB86799	0,25 μm	66 MHz	9,4 CM/MHz	5,4 mW/MHz	
ARM926EJ-S	0,13 μm	238 MHz	5,0 CM/MHz	0,36 mW/MHz	1,45 mm^2
AT32AP7000	0,18 μm	150 MHz		2,08 mW/MHz	
SHAP	0,13 μm	175 MHz	4,7 CM/MHz	0,14 mW/MHz	1,37 mm^2

[a]Emb. CaffeineMark 3.0, bei SHAP ohne den Float-Test

Tabelle 6.2: Vergleich der Leistungsdaten aktueller Prozessoren mit den errechneten Daten von SHAP

implementiert sind. Der Float-Test ist deswegen im Gesamtwert des Caffeine Benchmark nicht mit berücksichtigt.

Die Werte für die Leistungsaufnahme lassen sich nur mit dem ARM926EJ-S sinnvoll vergleichen, da die Werte in der Tabelle die Werte für den reinen Core ohne IO-Zellen und Daten-Cache sind. Der Wert von 0,14 mW/MHz für die SHAP-Architektur ergibt sich aus 25 mW bei 175 MHz (siehe Abschnitt 5.2). Rechnet man noch die prognostizierten 19,2 mW für die IO-Zellen hinzu, ergibt sich immer noch ein sehr guter Wert von 0,25 mW/MHz. Auch die Core-Fläche ist mit 1,37 mm^2 in der aktuellen Konfiguration noch vergleichsweise gering.

Die Gesamtflächenangabe von 2,71 mm^2 ist natürlich nur die Fläche von Core und Pad-Ring. Für die Herstellung müssen noch zwei Reihen Bond-Pads um den Pad-Ring gelegt und verdrahtet werden. Zudem ist ein Abstand zwischen Bond-Pads und Chipkante einzuhalten, um während des Zerteilens des Wafers in Einzelchips eine mechanische Verletzung der Logik zu vermeiden. Legt man einen mittleren Abstand von 80 μm zwischen zwei Bond-Pads zugrunde, und nimmt einen Abstand zwischen Bond-Pads und Chip-Kante von noch einmal 160 μm an, muss man um die äußeren Abmessungen, nach Anhang A.2, noch mal einen Ring von 320 μm einplanen. Die genauen Werte sind allerdings herstellerabhängig. Man kommt so auf eine Chipfläche von etwa 5,4 mm^2.

Dieser große Anteil der IO-Zellen und Bond-Pads an der Gesamtfläche lässt sich durch andere Konnektierungstechnologien verringern. Zum einen gibt es die Möglichkeit, die Bond-Pads über die IO-Zellen und zum Teil auch über die Core-Logik zu legen. Diese Verfahren wird oft mit CUP abgekürzt (Circuit-Under-Pad). Eine weitere Methode ist das Flip-Chip-Verfahren. Dabei werden die IO-Zellen samt der darauf befindlichen Kontakt-Pads innerhalb des Cores verteilt. Es wird nicht mehr herkömmlich durch Draht-Bonden der Chip mit dem Chip-Träger verbunden, sondern der Chip wird kopfüber (deswegen Flip-Chip) auf den Träger aufgebracht, und die Kontakte durch kleine Lötkügelchen hergestellt.

Die hier genutzte Konfiguration der SHAP-Logik ist natürlich nur beispielhaft. Für einen realen Chip würden noch Module für grafisches Interface, USB- und Netzwerk-Schnittstelle und weitere Module hinzukommen. Diese belegen dementsprechend noch zusätzliche Fläche und erhöhen auch die Leistungsaufnahme. Die Schnittstelle zum externen RAM ist hier auch noch

mit zweimal 32-Bit-Datenbus ausgeführt. Real würde man nur einen 32-Bit-Bus nehmen, und diesen wechselseitig als Ein- und Ausgang nutzen. Die frei gewordenen 32 IO-Zellen können dann für die zusätzlichen Module genutzt werden. Da die IO-Zellen auch frei konfigurierbar sind, ist es auch denkbar, der Architektur ein paar frei belegbare, Software-konfigurierbare, IO-Pins hinzuzufügen.

7 Zusammenfassung und Ausblick

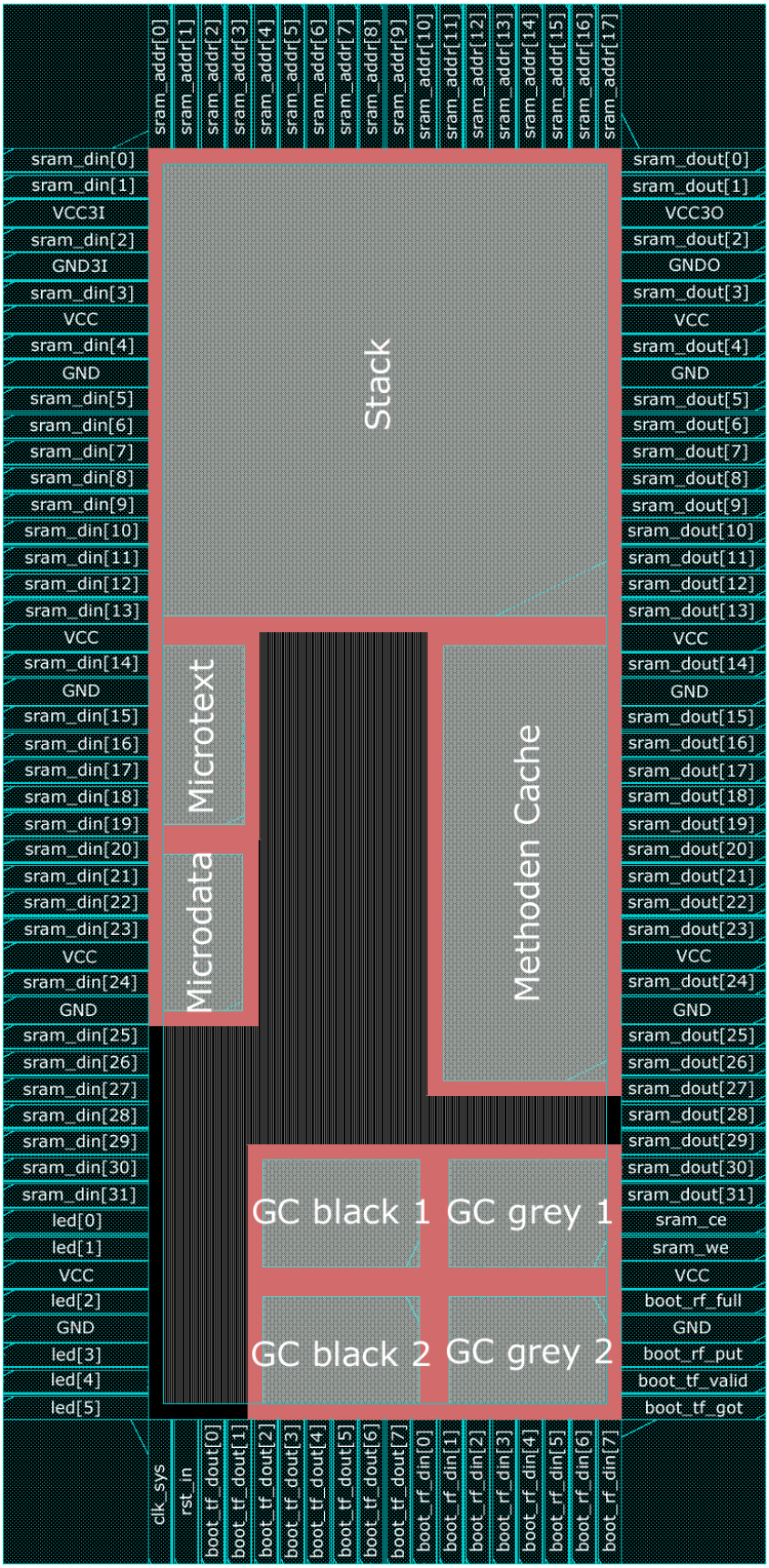
In dieser Arbeit wurden die Möglichkeiten der ASIC-Synthese der SHAP-Architektur untersucht. Es wurde eine beispielhafte Konfiguration in ein ASIC-Layout überführt und ermittelt, welche Taktfrequenz möglich ist und welche Leistungsaufnahme und Chipfläche nötig ist. Die Leistungsaufnahme wurde durch die Simulation des rechenintensiven Sieve-Tests des Caffeine Benchmark und des Speicherintensiven FScript-Tests berechnet. Im Vergleich mit anderen Prozessoren erreicht die SHAP-Architektur bei Leistungsaufnahme und Chipflächenbedarf sehr gute Werte. Bei der maximal erreichbaren Taktfrequenz werden nicht ganz die vergleichbaren Werte anderer Prozessoren erreicht.

Durch Optimierungen am kritischen Pfad sind im Core mit etwas mehr Aufwand sicher auch 200 MHz potentiell erreichbar. Da sich aber die Ausgangs-Pads vom Takt her am Limit befinden, sind auch dort Änderungen nötig, um die Taktfrequenz steigern zu können. Da für die Reduzierung der Leistungsaufnahme schon Technologien wie Clock-Gating genutzt werden, sehe ich dort nur noch wenig Einsparpotential. Die Chip-Fläche ist natürlich stark von der Anzahl und Größe der internen Speicher anhängig. Durch eine Änderung des Garbage Collectors, so dass dieser keinen „read first“-Speicher mehr benötigt, lässt sich zum Beispiel noch einiges einsparen. Da dann der Speicher nicht mehr doppelt angelegt werden müsste, könnte man die Hälfte der Fläche, 0,074 mm^2, sparen. Auch über die Größe des Stacks und des Methoden-Caches lässt sich der Flächenbedarf reduzieren, oder, wenn mehr Speicher nötig ist, auch vergrößern. Sämtliche Parameter lassen sich natürlich auch durch eine kleinere Strukturgröße von zum Beispiel 90 nm verbessern.

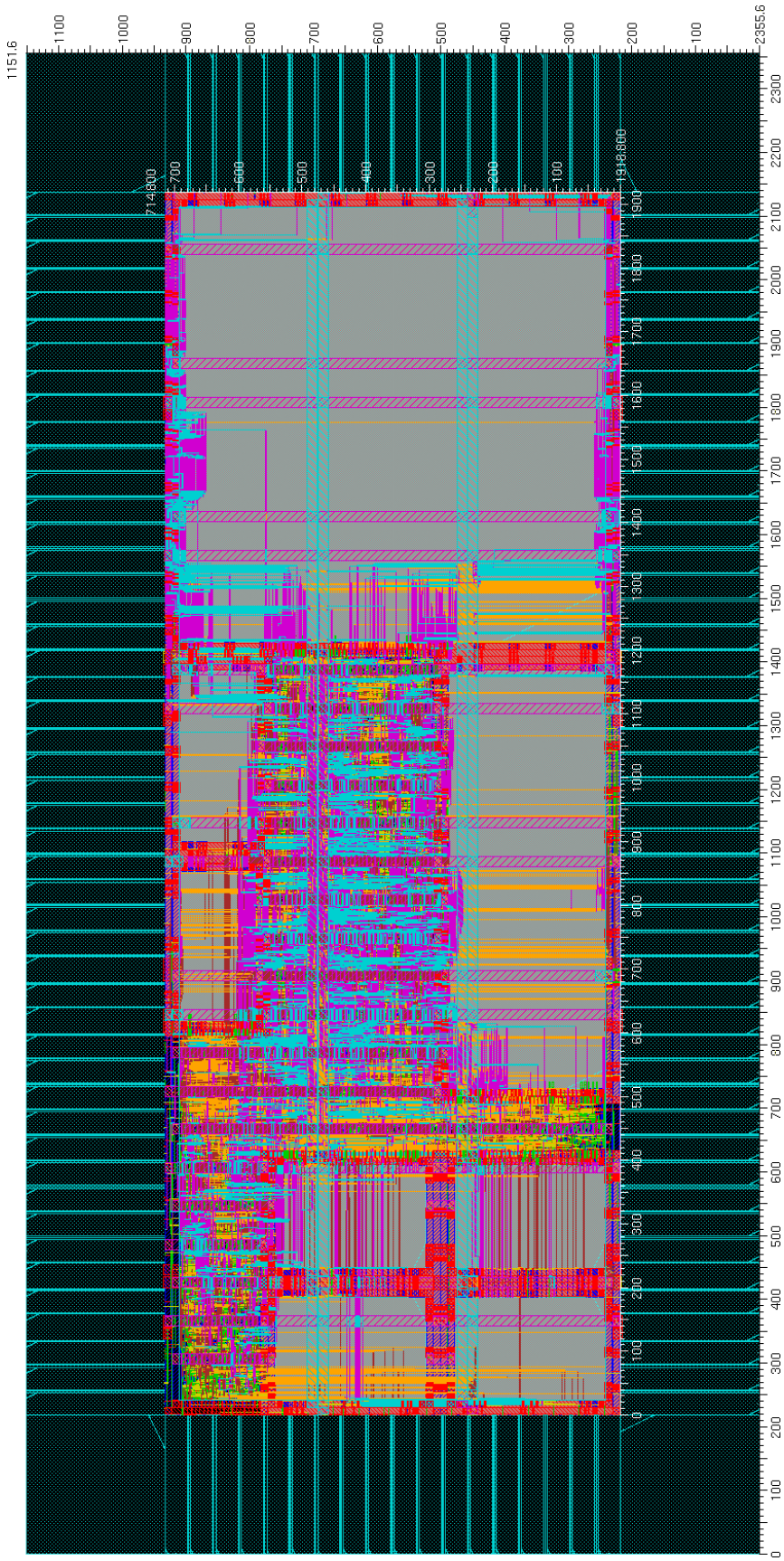
Bisher wurde die Funktionsfähigkeit des ASIC-Layouts nur durch die Simulation von Testprogrammen belegt. Praktisch müsste der volle Funktionsumfang aber noch durch eine formale Verifikation nachgewiesen werden. Dazu gibt es verbreitete Methoden und Tools, die es zu untersuchen und zu nutzen gilt.

Anhang A

A.1 Floorplan



A.2 Finales Layout



Literaturverzeichnis

[aJil] AJILE: *aJile Technology - Frequently Asked Questions*. http://www.ajile.com/index.php?option=com_easyfaq&Itemid=48. – Stand: 29.03.2009

[aJil00] AJILE: *aJ-100TM Real-time Low Power JavaTM Processor*. http://datasheet.digchip.com/552/aJ100Datasheet_1-3.pdf. 11/2000. – Stand: 23.02.2009

[ARMa] ARM: *ARM Cortex-M3*. http://www.arm.com/products/CPUs/ARM_Cortex-M3.html. – Stand: 29.03.2009

[ARMb] ARM: *ARM926EJ-S*. http://www.arm.com/products/CPUs/ARM926EJ-S.html. – Stand 23.02.2009

[ARM04] ARM: *ARM9 FAMILY*. http://www.arm.com/pdfs/ARM9_family_flyer_34_5.pdf. 09/2004. – Stand: 29.03.2009

[Cad05] Cadence Design Systems: *Encounter User Guide*. 4.1.5. 05/2005

[Cad07] Cadence Design Systems: *Encounter Text Command Reference*. 6.2.2. 08/2007

[Cum+03] CUMMINGS, C. E.; MILLS, D.; GOLSON, S. *Asynchronous & Synchronous Reset Design Techniques - Part Deux*. SNUG Boston. 09/2003. Rev 1.2

[Embe06] EMBEDDED TECHNOLOGY JOURNAL: *AVR32-Based SoC Offers High Throughput Solution for Compute-Intensive Embedded Control Applications*. http://www.embeddedtechjournal.com/news_2006/04/20060403_06.htm. 04/2006. – Stand: 29.03.2009

[Far04a] Faraday Technology: *Faraday 0.13µm Standard Cell Library FSC0H_D Layout / P&R Guideline*. 1.2. 08/2004

[Far04b] Faraday Technology: *FSC0H_D 0.13 µm Standard Cell*. 1.1. 03/2004

[Far05] Faraday Technology: *0.13µm (FSC0H_D) Standard Cell Library ESD Application Note*. 1.0. 09/2005

[Far06] Faraday Technology: *0.13 µm Memory Generator*. 1.5. 10/2006

[Mars01] MARSH, D.: *Silicon variety vanquishes embedded-Java taboos*. http://www.edn.com/article/CA61880.html. 01/2001. – Stand: 23.02.2009

[MSC08] MSC: *The World of AVR32 Series RISC Microcontrollers*. http://www.farnell.com/datasheets/113311.pdf. 2008. – Stand: 25.02.2009

[Pend] PENDRAGON SOFTWARE CORPORATION: *CaffeineMark 3.0 Information*. http://www.benchmarkhq.ru/cm30/info.html#The%20Embedded%20CaffeineMark. – Stand: 29.03.2009

[Pohl] POHL, M.: *Rechnerstrukturen - Bericht zu dem Vortrag über Java-Prozessoren*. http://www.weblearn.hs-bremen.de/risse/RST/WS02/javaproc.pdf. – Stand 23.02.2009

[Rus+03] RUSTON, M.; TRAN, T. A.; YONG, L.; YOUNGBLOOD, A.; RAVENSCRAFT, D.; HARUN, F.; MUI, K. W.: *Assembly Challenges Related to Fine Pitch In-line and Staggered Bond Pad Devices*. In: *Electronic Components and Technology Conference* IEEE, 2003. – http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01216468, S. 1334 –1343

[Schw] SCHWOPE, A.: *Chip Aufbau - Der IO-Bereich eines Chips*. http://www.andreas-schwope.de/ASIC_s/Aufbau/Aufbau-IO/body_aufbau-io.html. – Stand 29.03.2009

[Sha+07] SHAH, P.; SHELKE, R.; SHANBHAG, P.; DAS, G.: *A STAGGERED CUP IO METHODOLOGY USING ENCOUNTER PLATFORM*. In: *CDNLive! Conference* Cadence Design Systems, Ammos Software Technologies, 10/2007. – http://www.cadence.com/rl/Resources/conference_papers/3.5_presentationIndia.pdf

[Stif] STIFTUNGSLEHRSTUHL HOCHPARALLELE VLSI-SYSTEME UND NEURO-MIKROELEKTRONIK DER TU DRESDEN: *ICPRO - Wiki*. http://hpsn.et.tu-dresden.de/scpm/icpro/wiki. – Stand 29.03.2009

[Syn07] Synopsys: *Synopsys Online Documentation*. Z-2007.06. 2007

[unbe] UNBEKANNT: *ASIC How-tos*. http://www.asichowto.com/. – Stand 29.03.2009