



**Hochschule für Technik  
und Wirtschaft Berlin**

University of Applied Sciences

# **Anpassung einer ARMv4 Mikroarchitektur an XILINX Zynq FPGAs**

**Abschlussarbeit**

zur Erlangung des akademischen Grades  
**Bachelor of Engineering**

an der

Hochschule für Technik und Wirtschaft Berlin  
Fachbereich 1: Ingenieurwissenschaften - Energie und Information  
Studiengang Computer Engineering

1. Prüfer: Prof. Dr.-Ing. Carsten Gremzow
2. Prüfer: Dipl.-Ing. Ines Liepert

Eingereicht von: Fiona Brömer  
Matrikelnummer: 553052  
Datum der Abgabe: 20.08.2018

# Inhaltsverzeichnis

<b>Abstract</b>	<b>1</b>
<b>1 Einleitung</b>	<b>2</b>
<b>2 Theoretischer Hintergrund</b>	<b>4</b>
2.1 Rechnerarchitektur . . . . .	4
2.1.1 Abstraktionsebenen im Hardwareentwurf . . . . .	6
2.2 Von der Hardware zur Software . . . . .	7
2.2.1 Instruction Set Architecture . . . . .	8
2.2.2 Mikroarchitektur . . . . .	8
2.2.3 Spezifikationen der ARM-Prozessoren . . . . .	10
2.3 Digitaler Schaltungsentwurf . . . . .	12
2.3.1 Programmierbare Hardwarebausteine . . . . .	12
2.3.2 Hardwarebeschreibungssprache VHDL . . . . .	15
2.4 Multicore-Architektur . . . . .	16
2.4.1 Homogene Multicores . . . . .	16
2.4.2 Heterogene Multicores . . . . .	17
2.4.3 Softwareperspektive . . . . .	17
2.4.4 Multitasking und Multithreading . . . . .	18
<b>3 ARMv4 Prozessor als FPGA-Umsetzung</b>	<b>19</b>
3.1 Struktur . . . . .	19
3.2 Technische Besonderheiten . . . . .	20
3.3 Programmierung . . . . .	20
<b>4 Prozessoranpassung</b>	<b>22</b>
4.1 Inbetriebnahme auf Originalplattform Spartan 3 . . . . .	22
4.1.1 Originalhardware Spartan 3E . . . . .	22
4.1.2 Ersatzhardware Spartan 3AN . . . . .	23
4.2 Anpassung auf Zielplattform Zynq . . . . .	23
4.3 Notwendige Änderungen . . . . .	25
4.3.1 Block-RAM Speicher . . . . .	25
4.3.2 UART-Anschluss . . . . .	27
4.3.3 Anpassung des Systemtaktes . . . . .	30
<b>5 Konzept und Umsetzung einer Multicore-Architektur</b>	<b>31</b>
5.1 Konzepte . . . . .	31
5.1.1 Gemeinsamer Hauptspeicher . . . . .	31
5.1.2 Getrennter Hauptspeicher . . . . .	31
5.1.3 Coprozessor-Anbindung . . . . .	32
5.1.4 Transputer-Konzept . . . . .	32

5.1.5	Software für Multicore-Architekturen . . . . .	33
5.2	Umsetzung . . . . .	33
<b>6</b>	<b>Ergebnisse</b>	<b>36</b>
6.1	Funktionalität . . . . .	36
6.2	Ressourcennutzung . . . . .	36
<b>7</b>	<b>Fazit und Ausblick</b>	<b>38</b>
	<b>Eidesstattliche Erklärung</b>	<b>39</b>
	<b>Literatur</b>	<b>40</b>
	<b>Anhang</b>	<b>43</b>
<b>A</b>	<b>ARM-Architektur</b>	<b>43</b>
<b>B</b>	<b>VHDL</b>	<b>43</b>
<b>C</b>	<b>Auslastung des Zynq FPGAs</b>	<b>44</b>
C.1	Genutzte Fläche . . . . .	44
C.2	Genutzte Ressourcen . . . . .	44

## Abbildungsverzeichnis

1	Von-Neumann-Architektur mit Hervorhebung des Flaschenhalses . . . . .	4
2	Harvard-Architektur . . . . .	5
3	Abstraktionsebenen nach Golze [13] . . . . .	7
4	Abstraktionsebenen im Zusammenspiel von Hard- und Software [7] . . . . .	8
5	Grafische Darstellung der fünfstufigen Pipeline nach [12, S. 24] . . . . .	9
6	Aufbau einer konfigurierbaren logischen Matrix nach Hoffmann [17] . . . . .	13
7	Aufbau eines FPGA nach [2] . . . . .	14
8	Typische Speicheranbindung eines homogenen Multicores . . . . .	17
9	Aufruf und Ausgabe des Testprogramms auf dem Spartan 3AN-Board . . . . .	24
10	Vergleich der verschiedenen Xilinx-FPGAs [30] . . . . .	24
11	Block-RAM-Speicher im Vivado IP Generator . . . . .	25
12	Simulation des Block-RAM-Speichers in Vivado . . . . .	26
13	Flächenauslastung des Zynq-FPGAs bei einem Prozessorkern . . . . .	27
14	UART-Verbindung zwischen Zynq-Board und Adapter . . . . .	28
15	Messung der Übertragung des Programm-Binärfiles am Oszilloskop . . . . .	29
16	Struktur eines Transputers nach [18] . . . . .	33
17	Anpassung der Ports im Top-Modul bei mehreren Kernen . . . . .	34
18	Flächenauslastung des FPGA mit acht Kernen . . . . .	35
19	Übersicht über verschiedene ARM-Prozessoren zur Einordnung der Generation ARMv4 [4] . . . . .	43
20	Anpassung des Systemcontrollers zur Umsetzung mehrerer Kerne . . . . .	43
21	Flächenauslastung des Zynq-FPGAs bei zwei Prozessorkernen . . . . .	44
22	Ressourcennutzung des Spartan 3E . . . . .	44
23	Ressourcennutzung des Spartan 3AN . . . . .	45
24	Ressourcennutzung des Zynq-FPGAs bei einem Prozessorkern . . . . .	45
25	Ressourcennutzung des Zynq-FPGAs bei zwei Prozessorkernen . . . . .	46

## Abkürzungsverzeichnis

<b>AMP</b>	Asymetric Multiprocessing
<b>ARM</b>	Advanced RISC Machines
<b>CISC</b>	Complex Instruction Set Computer
<b>CLB</b>	Configurable Logic Block
<b>CPLD</b>	Complex PLD
<b>DCM</b>	Digital Clock Management
<b>FPGA</b>	Field Programmable Gate Array
<b>HDL</b>	Hardware Description Language
<b>IP</b>	Intellectual Property
<b>ISA</b>	Instruction Set Architecture
<b>ISE</b>	Integrated Synthesis Environment
<b>JTAG</b>	Joint Test Action Group
<b>LUT</b>	Look-Up-Table
<b>MMCM</b>	Mixed Mode Clock Manager
<b>PLD</b>	Programmable Logic Device
<b>RAM</b>	Random Access Memory
<b>RISC</b>	Reduced Instruction Set Computer
<b>RTL</b>	Register-Transfer-Logik
<b>SMP</b>	Symmetric Multiprocessing
<b>UART</b>	Universal Asynchronous Receiver Transmitter
<b>VHDL</b>	VHSIC Hardware Description Language
<b>ZED</b>	Zynq Evaluation and Development

### Abstract

Ziel dieser Arbeit ist es, eine Prozessorumsetzung in der Hardwarebeschreibungssprache VHDL an eine neue Hardwareplattform anzupassen. Die gegebene Umsetzung wurde implementiert auf einem FPGA-Board des Herstellers Xilinx und soll auf einem neueren Board-Modell mit Namen Zynq lauffähig gemacht werden. Beim Prozessortypen handelt es sich um das Modell ARMv4, welches auf der weit verbreiteten ARM-Architektur basiert.

Um die praktische Umsetzung nachvollziehbar zu machen, wird zunächst auf die theoretischen Grundlagen der Rechnerarchitektur eingegangen, mit Schwerpunkt auf die Hardware-Software-Schnittstelle und die Eigenschaften der ARM-Prozessoren. Außerdem wird die Funktionsweise der verwendeten Hardware und die dazu genutzte Hardwarebeschreibungssprache erklärt.

Die Prozessorimplementierung, welche angepasst wird, wird aus struktureller Sicht zusammengefasst, die technischen Besonderheiten und die Maßnahmen zum Programmieren auf dem fertigen Prozessor werden erläutert.

In der Umsetzung wurde zunächst die bestehende Implementierung auf einem Spartan 3AN-FPGA in Betrieb genommen und getestet. Um die Anpassung an die Zynq-Architektur zu ermöglichen, wurden kompatible BRAM-Speicher einzeln erzeugt und mithilfe einer Testbench getestet. Diese wurden anschließend in die Prozessorbeschreibung eingebunden und der Prozessor somit auf das neue FPGA-Board übertragen.

Um den Prozessor auch auf der neuen Plattform programmierbar zu machen, wurde außerdem die Verbindung mit einem Host-Rechner über eine UART-Schnittstelle angepasst und anschließend die Programmierbarkeit getestet.

Da der Zynq-FPGA über ein Vielfaches der Ressourcen der Spartan-Reihe verfügt, konnte mehr als ein Prozessorkern implementiert werden. Dafür musste die Takterzeugung aus den einzelnen Kernen in ein zentrales Modul ausgelagert werden und die entsprechenden Schnittstellen zu den Kernen und innerhalb dieser angepasst werden. Es konnten acht Kerne umgesetzt werden, welche die Ressourcen des Zynq-FPGAs nicht komplett auslasten. Diese sind alle voll funktionstüchtig, können aber nicht unabhängig von einander programmiert werden.

Abschließend werden die Ergebnisse zusammengefasst und Eigenschaften der originalen und neuen Implementierung verglichen. Dabei liegt der Schwerpunkt auf der Gegenüberstellung der genutzten Ressourcen und der dadurch möglichen Erweiterung zur Multicore-Architektur. Die Ressourcen des Zynq-FPGAs sind dabei nicht annähernd ausgelastet, selbst bei einer Achtkern-Architektur ergibt sich eine Belegung der LUTs von nur 54%.

# 1 Einleitung

Betrachtet man die Bestandteile eines Rechnersystems, so stellt sich schnell heraus, dass dem Prozessor als Kernelement eine wichtige Rolle zukommt. Auf viele Einzelteile kann verzichtet oder diese können verkleinert werden, aber ein Rechnersystem ohne Prozessor kann schon per Definition nicht existieren. Umso wichtiger ist es, die Funktionen und Bestandteile eines Prozessors in einem Studium der technischen Informatik kennen zu lernen und so gut wie möglich zu überblicken. Das geschieht im Bachelorstudium Computer Engineering an der Hochschule für Technik und Wirtschaft Berlin in mehreren Fächern, besonders aber im Modul Embedded Systems, welches sich hardwarenah mit einem bestimmten Prozessor beschäftigt, wobei Erweiterungen erschaffen werden und dieser Prozessor auch implementiert wird.

Aktuell handelt es sich dabei um den KCPSM3, auch bekannt unter dem Namen Picoblaze [29], welcher unter anderem um einen UART-Baustein erweitert und programmiert wird. Allerdings ist die Hardware dieses Prozessors kaum verbreitet und Anwendungsfälle sind von eher theoretischer Natur. In anderen Modulen des Studiengangs werden nach und nach immer öfter Prozessoren der ARM-Reihe verwendet, welche in der Praxis weit verbreitet sind und besonders im Bereich der Eingebetteten Systeme eine marktführende Position einnehmen [14].

Ein Prozessor des Modells ARMv4 wurde von Herrn Carsten Böhme im Rahmen einer Diplomarbeit an der Technischen Universität Berlin in der Hardwarebeschreibungssprache VHDL implementiert und somit für den Hardwareentwurf zugänglich gemacht. Diese Umsetzung soll im Rahmen dieser Arbeit an eine neue Entwicklungsplattform angepasst und erweitert werden und so für den Studiengang Computer Engineering nutzbar gemacht werden, da die konkrete Umsetzung eines Prozessors die beste Möglichkeit bietet, um Aufbau und Funktionsweise eines solchen zu verstehen.

Da die ARM-Architektur in vielen Fächern des Studiengangs eingesetzt und programmiert wird, kann eine frei zugängliche, vollständige und einsetzbare Hardwareumsetzung also zum Gesamtverständnis und zum Zusammenspiel der verschiedenen Module beitragen.

Dementsprechend stützt sich die Arbeit auf die im Studium erworbenen Kenntnisse in den Themenbereichen Rechnerarchitektur und digitaler Schaltungsentwurf. Der Schwerpunkt liegt hierbei auf den Eigenschaften und Unterschieden der verschiedenen FPGA-Plattformen, insbesondere bei der Speicheranbindung und der Kommunikation mit und Programmierung des entstandenen Prozessors.

Dabei bietet die Arbeit einen Einblick in die Umsetzung der ARMv4-Mikroarchitektur, die Anforderungen an und die Merkmale der genutzten Hardwareentwicklungsboards, einen Überblick über die zugrunde liegende Technologie der programmierbaren Logik und die Erweiterung des Prozessors zu einer Multicore-Architektur.

Hierfür werden zunächst die theoretischen Grundlagen erläutert, die vorausgehende Arbeit

## 1. EINLEITUNG

---

und dadurch entstandene Implementierung zusammengefasst und anschließend die im Rahmen dieser Arbeit entwickelten Anpassungen erklärt. Einen besonderen Stellenwert nimmt dabei die Umsetzung als Mehrkern-Prozessor ein, da diese durch ihre Komplexität besonders anspruchsvoll ist. Abschließend werden die Ergebnisse, besonders im Hinblick auf die genutzte Hardware und damit verbundene Ressourcennutzung, verglichen.

Als Hardwareplattform für die Entwicklung dient ein sogenanntes ZED-Board, das den Entwurf digitaler Schaltungen mit einem FPGA als konfigurierbarem Logikbaustein ermöglicht. Die Entwicklung erfolgt zugeschnitten auf die Hardwareentwurfsmodule im Studiengang mithilfe der Herstellersoftware, um die Nutzung in der Lehre leichter möglich zu machen.



## 2 Theoretischer Hintergrund

Wie bereits beschrieben, kommt dem Prozessor als Kernelement jedes Rechnersystems eine wichtige Bedeutung zu. Deshalb soll in diesem Kapitel eine Einführung in die theoretischen Grundlagen gegeben werden, sowohl in den Bereich der Rechnerarchitektur als auch in den Hardwareentwurf digitaler Schaltungen, wobei besonders auf die in dieser Arbeit verwendete Hardware der programmierbaren Logik und die jeweilige Hardwarebeschreibungssprache eingegangen wird.

### 2.1 Rechnerarchitektur

Die zentrale Aufgabe eines Prozessors ist es, Daten auf eine gewünschte Art und Weise zu verarbeiten und das Ergebnis auszugeben. Um das zu ermöglichen, werden drei zentrale Elemente benötigt: das Rechenwerk, welches die eigentliche Verarbeitung vornimmt, das Steuerwerk, welches das Decodieren von Instruktionen übernimmt, und die Register, in welchen Werte lokal gespeichert werden können. Bei letzteren wird zusätzlich unterschieden zwischen Universalregistern (auch General Purpose Register), die für beliebige Zwecke verwendet werden können, und Hilfsregistern, welche zur Steuerung verwendet werden und Werte wie den Instruktionszähler oder das Statusregister enthalten [17, S. 357].

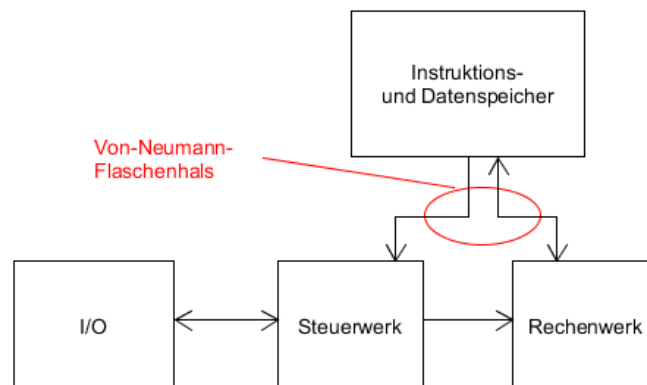


Abbildung 1: Von-Neumann-Architektur mit Hervorhebung des Flaschenhalses

Darüber hinaus ist der Prozessor auf die Anbindung an Speicherelemente angewiesen, aus denen Instruktionen und Daten gelesen und in die Daten geschrieben werden können. Hierbei können Daten- und Instruktionsspeicher entweder getrennt (Harvard-Architektur) oder gemeinsam (Von-Neumann-Architektur) vorliegen [17, S.352 ff.], wobei letzteres durch den so genannten Von-Neumann-Flaschenhals die Verarbeitungsgeschwindigkeit stark reduzieren kann. Beim Von-Neumann-Flaschenhals handelt es sich um die Engstelle bei der Speicheranbindung, welche entsteht, wenn nicht zwischen Daten- und Instruktionsspeicher getrennt wird. Dabei ergibt sich das Problem, dass niemals auf Befehle und Daten gleichzeitig zugegriffen werden kann, es erfolgt also bei jedem Schreiben oder Lesen eines Datums eine Verzögerung beim Lesen der Instruktionen, welche die gesamte Ausführungsgeschwindigkeit stark beeinträchtigt. Dies ist in Abbildung 1 dar-

## 2. THEORETISCHER HINTERGRUND

---

gestellt. Liegen beide Speicher getrennt vor, so wird auch getrennt darauf zugegriffen und der Flaschenhals entfällt (siehe Abbildung 2).

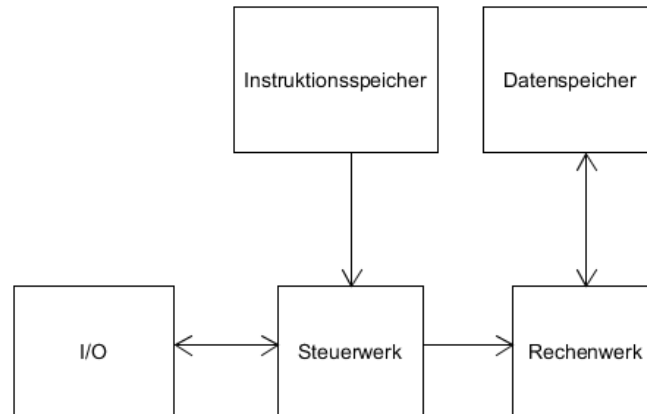


Abbildung 2: Harvard-Architektur

Außerdem ist ein Prozessor, um eine Funktionalität zu erfüllen, üblicherweise an Peripheriegeräte angebunden. Elementar ist hierbei externer Speicher wie beispielsweise Festplatten. Darüber hinaus kann es sich um Ausgabegeräte (wie Bildschirme) und Eingabegeräte (wie Maus und Tastatur) handeln, im Bereich der eingebetteten Systeme sind jedoch eher Sensoren, Aktoren und primitive Ein- und Ausgabegeräte wie Taster und LCD-Anzeigen verbreitet.

Zur Umsetzung dieser allgemeinen Anforderungen gibt es verschiedene Architekturarten, von denen die beiden am weitesten verbreiteten Grundkonzepte hier kurz erläutert werden.

### RISC-Prozessoren

Die Abkürzung RISC steht für Reduced Instruction Set Computer und bezieht sich auf den Instruktionssatz des Prozessors. Im Gegensatz zu CISC-Prozessoren umfasst dieser wenige und dafür kurze Instruktionen, was eine Speicher- und Zeitersparnis mit sich bringt. Der Grundgedanke dieser Architektur ist, dass sich jeder komplexe Befehl auch durch mehrere einfachere darstellen lässt.

Eine wichtige Eigenschaft von RISC-Prozessoren ist die Load-Store-Architektur. Das bedeutet, dass bestimmte Befehle - und nur diese - auf den Hauptspeicher (auch: Arbeitsspeicher) zugreifen können, für alle anderen Operationen muss der Operand bereits im Registerspeicher vorliegen. Das macht auch den direkten Transfer eines Datums innerhalb des Hauptspeichers unmöglich, hier muss der Zwischenschritt über den Registerspeicher gemacht werden. Die Vorgabe, dass alle Operanden im Registerspeicher vorliegen müssen, sorgt auch dafür, dass bei dieser Architektur in der Regel mehr Register vorhanden sind [17, S. 385].

Gerade im Bereich der mobilen Geräte und der eingebetteten Systeme ist dies die vorherrschende Architektur. Ein Grund dafür ist unter anderem die geringe Chipfläche, welche auf die

einfachere Logik zum Decodieren kurzer Instruktionen und die weniger komplexen Befehle als solche zurückgeht. Damit verbunden sind eine geringere Leistungsaufnahme und günstigere Fertigungskosten [17, S. 384].

Die ARM-Prozessoren, um die es in dieser Arbeit geht, sind als RISC-Prozessoren einzuordnen.

Eine weiterführende Darstellung des Konzepts mit allen typischen Eigenschaften findet sich bei [15, S. C-4] im Kapitel Basics of RISC Instruction Set.

### **CISC-Prozessoren**

Das dem RISC-Konzept entgegengesetzte Beispiel sind die CISC-Prozessoren. Die Befehle des Complex Instruction Set Computer sind komplex und dementsprechend lang. Das bedeutet, dass oft mehrere Operationen in einem Befehl zusammengefasst werden können, was Programme kürzer und übersichtlicher machen kann. Ob dieser Vorteil ins Gewicht fällt, hängt allerdings davon ab, ob überhaupt in Assembler programmiert werden soll, was nur noch in Ausnahmefällen benötigt wird. Bei der Nutzung eines Compilers werden im Schnitt nur 20% der komplexen Befehle genutzt, der Vorteil geht also verloren. Darüber hinaus sind die häufig benutzten Elementaroperationen durch das langsamere Dekodieren der Befehle etwas langsamer als bei RISC-Prozessoren [17, S. 384].

Dieser Instruktionssatz beinhaltet unter anderem den Transfer von Daten innerhalb des Hauptspeichers, aber auch die Manipulation von Adressen, wie beispielsweise das Inkrementieren des Stackpointers innerhalb eines anderen Befehls [17, S. 380].

Das bekannteste und am weitesten verbreitete Beispiel ist hier die x86-Architektur des Herstellers Intel.

#### **2.1.1 Abstraktionsebenen im Hardwareentwurf**

Ein Prozessor lässt sich in verschiedenen Abstraktionsebenen betrachten, wie in Abbildung 3 zu sehen ist [13, S. 19]. Die Layout-Ebene beschäftigt sich dabei mit der geometrischen Anordnung einzelner Elemente im Silizium, welche dadurch Transistoren ergeben. Auf der nächsten Ebene hingegen, der Transistorebene, wird die Umsetzung der Schaltung in Transistorlogik behandelt. Daraus ergeben sich im nächsten Schritt logische Gatter, wie das AND- oder OR-Gatter.

Die Logik- oder Gatterebene fasst die darunterliegenden Transistorschaltungen zu logischen Funktionselementen zusammen, welche auf digitalelektronischer Ebene als Logikschaltungen betrachtet werden können. Statt Spannungswechseln stehen hier 1- und 0-Werte im Vordergrund. Hier bietet es sich an, die Schaltung unter Zuhilfenahme von boolescher Algebra zu betrachten.

Die Register-Transfer-Ebene oder auch RTL-Ebene beschäftigt sich in erster Linie mit dem Transfer von Signalen zwischen Registern. Dadurch wird bereits stark von der elektrischen Schaltung abstrahiert, Konstrukte wie Speicher und Datenstrukturen werden ermöglicht. Auch die Betrachtung von Spannungswechseln als Signale trägt dazu bei. Dies vereinfacht den Entwurf komplexer Schaltungen und ermöglicht damit deutlich größere und schwierigere Projekte. Auf dieser Ebene sind auch Zustandsautomaten (State Machine) angesiedelt, welche zur Umsetzung




Abstraktionsebene	kleinste Strukturen	Beispiel
Systemebene	Architekturblöcke	Pipeline
Register-Transfer- oder RTL-Ebene	Register, kombinatorische Logik	$d = a \& c \mid d$
Logik- oder Gatterebene	Bibliothekszellen	
Transistorebene	Transistoren	
Layout-Ebene	Geometrien	

Abbildung 3: Abstraktionsebenen nach Golze [13]

sowohl Register zum Speichern von Zustandsvariablen als auch Logik zur Realisierung von Zustandsübergängen benötigen. Das ist die Ebene, auf der sich der praktische Teil dieser Arbeit bewegt.

Die Systemebene als höchste Abstraktionsebene befasst sich nicht mit der Hardwareumsetzung, sondern nutzt Funktionsblöcke als grafische Darstellungsmethode, um Funktionalitäten und Zusammenhänge zu erläutern.

In verschiedenen Abstraktionsstufen werden hier nicht nur die Bauteile betrachtet, sondern damit auch die dahinter stehenden Funktionen. So ist ein in der Layout- und Transistorebene erkennbarer Spannungswechsel in der Logikebene ein logischer Eins- oder Nullwert und in der RTL-Ebene ein Datenbit, möglicherweise auch ein Teil einer Datenstruktur, beispielsweise als Element eines Bitvektors. In der Systemebene hingegen spielt das einzelne Datenbit in der Regel keine Rolle mehr, ausgenommen sind an dieser Stelle natürlich Steuersignale wie Enable-Leitungen oder Interrupts. Hier werden eher, je nach Architektur, 8-Bit, 16-Bit oder 32-Bit breite Bitgruppen betrachtet.

### 2.2 Von der Hardware zur Software

Eine in Richtung der Software erweiterte Betrachtung der Abstraktion von physikalisch-elektrischen Bauelementen beschäftigt sich nicht in erster Linie mit dem Hardwareentwurf, sondern mit dem Zusammenspiel zwischen Software und Hardware (siehe Abb. 4). Unterhalb der Ebene des Betriebssystems, welche im Bereich der eingebetteten Systeme zunächst vernachlässigt werden kann<sup>1</sup>, findet sich die Schicht, die es überhaupt erst ermöglicht, Software für bestimmte Hardware zu entwickeln. Hierbei handelt es sich um die Instruction Set Architecture, kurz ISA.

Für den Programmierer eines Mikroprozessors ist diese von entscheidender Bedeutung. Um ein funktionsfähiges Programm zu erhalten, ist es nicht relevant, wie genau die Hardwareumset-

---

<sup>1</sup>Natürlich finden sich auch im Bereich der eingebetteten Systeme Betriebssysteme, aber sie sind nicht zwingend notwendig und in einigen Fällen sogar hinderlich, beispielsweise bei besonders zeitkritischen Anwendungen.

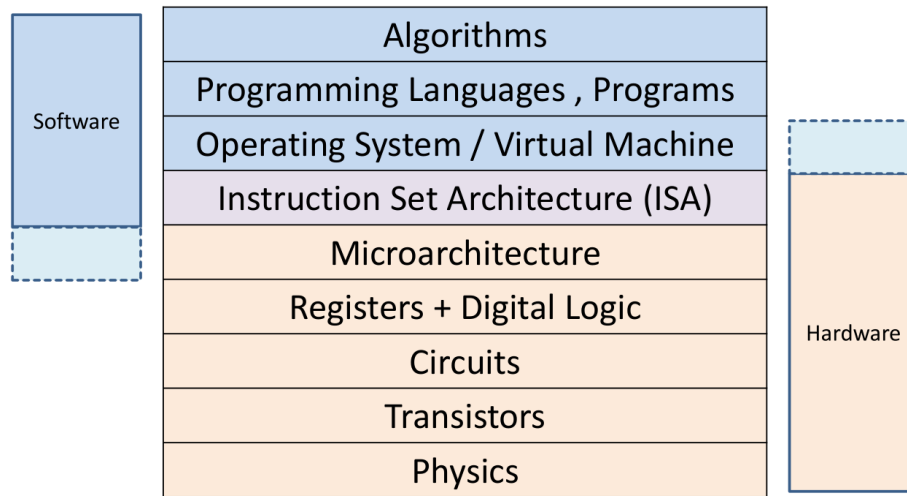


Abbildung 4: Abstraktionsebenen im Zusammenspiel von Hard- und Software [7]

zung aussieht, sondern wie diese Hardware angesprochen und gesteuert wird. Das ist in der ISA vereinbart und muss vom Hardware- genauso wie vom Software-Entwickler eingehalten werden und darüber hinaus dem verwendeten Compiler bekannt sein [16, S. 20].

### 2.2.1 Instruction Set Architecture

Die ISA oder auch Befehlssatzarchitektur beschreibt das Verhalten des Prozessors bei bestimmten Befehlen.

Nach Hoffmann wird der Befehlssatz definiert als [...]die Gesamtheit aller Maschinenbefehle, die ein Mikroprozessor verarbeiten kann [17, S. 416]. Allerdings erfasst diese Definition kaum ausreichend die Menge der hierbei erhaltenen Informationen. So wird beispielsweise festgelegt, wie viele Operanden einem Befehl übergeben werden sollen, ob es sich um eine Load-Store-Architektur handelt und wie breit die Adressen sind [20, S. 46].

Die ISA unterliegt allgemein wenigen Veränderungen, da einmal entwickelte Programme über eine längere Zeit und mehrere Prozessorgenerationen ausführbar sein sollen. Es können zwar neue Funktionalitäten in neueren Generationen hinzugefügt werden, aber die Abwärtskompatibilität muss durch das Beibehalten der bisherigen Konventionen sichergestellt werden.

Zusammengefasst lässt sich sagen, dass die ISA die Schnittstelle zwischen Hardware und Software definiert und damit in jedem Hardware- und Softwareentwurf eine wichtige Rolle einnimmt, auch wenn die Beachtung dieser dem Softwareentwickler zumeist vom Compiler abgenommen wird.

### 2.2.2 Mikroarchitektur

Die Mikroarchitektur ist noch eine Ebene näher an der tatsächlichen Hardware anzusiedeln als die ISA. Sie implementiert die Befehle als Hardwareumsetzung, also durch Logik- und Speicher-

## 2. THEORETISCHER HINTERGRUND

elemente. Im Gegensatz zur ISA ist die Mikroarchitektur flexibel, sie kann modifiziert, veränderten Bedürfnissen angepasst und im Laufe der Zeit auch optimiert werden, ohne die Kompatibilität mit vorhandener Software zu gefährden.

Außerdem können auf Mikroarchitekturebene viele Optimierungen stattfinden, da sie, anders als die ISA, in jeder neuen Generation verbessert und Bedürfnissen angepasst werden kann. Zu nennen sind hier das Prinzip des Pipelining, die Out-of-Order-Execution und die Sprungvorhersage (Branch Prediction) [7], wobei im Praxisteil dieser Arbeit nur das Pipelining eine Rolle spielt, da der Prozessor nicht in erster Linie auf Performance ausgelegt ist. Da die klassische fünfstufige Pipeline im hier verwendeten Prozessor Anwendung findet [8, S. 55], ist sie nachfolgend kurz dargestellt.

### Pipelining

Pipelining bedeutet, dass Abläufe beim Ausführen von Instruktionen in kleinere Abschnitte zerlegt werden und diese parallel ausgeführt werden können. Das ist vergleichbar mit der Fertigung von Gegenständen am Fließband, weshalb im Deutschen auch oft von Fließbandverarbeitung gesprochen wird. Diese Abschnitte umfassen in der einfachsten Form das Holen der nächsten Instruktion (Instruction Fetch), das Dekodieren dieser (Instruction Decode) und natürlich das Ausführen (Execute). Dazu kommen in der klassischen fünfstufigen Pipeline der Speicherzugriff (Memory Access) und das Zurückschreiben des Ergebnisses (Write Back Cycle) [15, S. C-6]. Der zeitliche Ablauf der Instruktionsbearbeitung in einer fünfstufigen Pipeline ist in Abbildung 5 dargestellt.

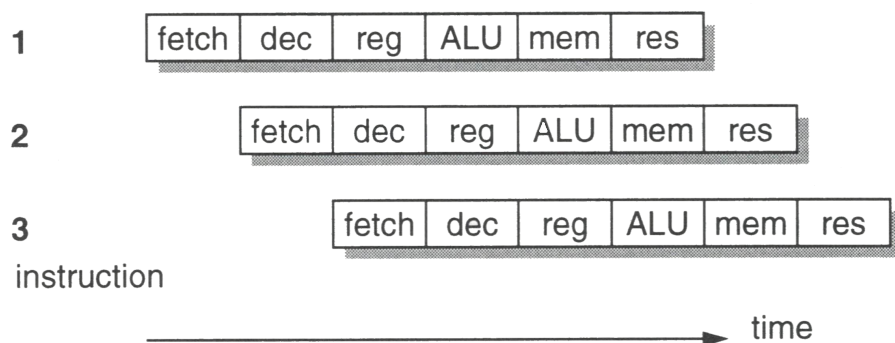


Abbildung 5: Grafische Darstellung der fünfstufigen Pipeline nach [12, S. 24]

Mit dieser Methode können verschiedene Instruktionen gleichzeitig bearbeitet werden, da in jeder Pipelinestufe eine Instruktion bearbeitet und außerdem der gesamte Systemtakt erhöht werden kann, weil bei der Ausführung eines Befehls nur noch auf den langsamsten Pipelineabschnitt und nicht mehr auf die gesamte Ausführungsdauer gewartet werden muss. Hierbei ist zu beachten, dass das System zwar insgesamt schneller wird, die Ausführungszeit eines einzelnen Befehls allerdings

nicht verkürzt wird [15, S. C-10].

Natürlich kann es beim Pipelining auch Probleme (sogenannte Hazards) geben, so gibt es beispielsweise Befehle, die länger als einen Takt zur Ausführung benötigen. Es kann zu Konflikten um Hardwareressourcen kommen (Structural Hazards), wenn beispielsweise mehr als ein Befehl auf eine Multiplikatoreinheit zugreifen will. Auch Datenkonflikte (Data Hazards) können auftreten, wenn beispielsweise eine Instruktion auf Ergebnisse einer anderen zugreifen muss, welche noch nicht zurückgeschrieben sind. Data Hazards werden weiterhin eingeteilt in verschiedene Arten von Konflikten, die zustande kommen können, indem sich die Ausführungszeiten verschiedener Instruktionen überschneiden:

- Read after Write (RAW): Eine Instruktion will ein Ergebnis lesen, welches eine andere noch nicht geschrieben hat
- Write after Write (WAW): Zwei Instruktionen wollen in das gleiche Zielregister schreiben; durch die Überschneidung in der Ausführungszeit kann es dazu kommen, dass sie das in der falschen Reihenfolge tun und das falsche Ergebnis sichtbar bleibt (ist nur möglich, wenn mehr als eine Pipelinestufe schreiben darf)
- Write after Read (WAR): Eine Instruktion überschreibt einen Wert, welcher noch von einer anderen benötigt wird, so dass dort fälschlicherweise der neue Wert gelesen wird

Des Weiteren kann es bei Sprüngen oder Verzweigungen im Programmablauf zu Control Hazards kommen; das bedeutet, es sind bereits Instruktionen in der Pipeline, welche aufgrund des Sprungs nicht benötigt werden. Das kann umgangen werden, indem festgelegt wird, dass eine bestimmte Anzahl Befehle nach einem Sprung immer ausgeführt wird, jedoch ist dies eine Änderung der ISA, was nicht immer gewünscht ist. Wenig effizient, aber wirkungsvoll ist der sogenannte Pipeline Flush, hierbei wird einfach die gesamte Pipeline geleert. Das ist weit verbreitet, da es einfach umzusetzen ist und den Konflikt zweifelsfrei auflöst. Es gibt komplexere Herangehensweisen, um das Problem ganz zu vermeiden, wie beispielsweise die Sprungvorhersage (Branch Prediction), welche an dieser Stelle jedoch nicht weiter erläutert werden. Weiterführende Informationen finden sich in [15].

### 2.2.3 Spezifikationen der ARM-Prozessoren

Die Abkürzung ARM steht für Advanced RISC Machines und bezeichnet ursprünglich die Firma ARM Limited [3], wird aber auch als Benennung für die Prozessoren dieses Herstellers genutzt. Dabei handelt es sich nicht um einen Hersteller im klassischen Sinne, ARM entwirft Architekturen und Chipdesigns und lizenziert diese für andere Firmen, welche daraus wiederum Hardware herstellen und diese entweder verkaufen oder in eigene Produkte einbauen.

Alle älteren ARM-Prozessoren besitzen eine Befehlslänge von 32 Bit [15, S. 14], was das Interpretieren der Instruktionen stark vereinfacht. ARM-Prozessoren mit einer 64-Bit-Architektur sind seit 2011 bekannt [1], werden hier aber nicht näher betrachtet.

ARM-Prozessoren gehören zur Gruppe der RISC-Prozessoren, was bedeutet, dass sie kurze Befehle haben, deren Anzahl dadurch überschaubar ist und die für komplexere Operationen gegebenenfalls kombiniert werden müssen (siehe auch Kapitel 2.1). Die ARM-Architektur ist eine Load-Store-Architektur, es kann also nur über bestimmte Befehle auf den Speicher zugegriffen werden. Bei den Befehlen handelt es sich um 3-Adress-Instruktionen, es werden also zwei Parameter und ein Ziel übergeben. Dabei handelt es sich nicht um konkrete Adressen im Hauptspeicher, sondern um Bezeichner für Register, in denen die jeweiligen Werte stehen [12, S. 17].

Um Software für die ARM-Prozessoren zu entwickeln, ist es nicht zwangsläufig nötig, sich detailliert mit der ISA zu beschäftigen. Durch die weite Verbreitung dieser Architektur sind mit der GNU Toolchain diverse Werkzeuge verfügbar, die die Softwareentwicklung in Hochsprachen ermöglichen und über das Cross-Compilieren<sup>2</sup> auch die Nutzung komplexer, für eingebettete Systeme sonst nicht geeigneter Entwicklungsumgebungen unterstützen [5].

Darüber hinaus gibt es einige Besonderheiten, die an dieser Stelle erläutert werden.

### **Thumb-Instruktionen**

Der Thumb-Instruktionssatz ist ein verkürzter Instruktionssatz mit nur 16 Bit Befehlslänge, welcher genutzt werden kann, um besonders kurze und speicheroptimierte Programme zu schreiben. Es handelt sich dabei nicht um eine komplette eigenständige Architektur, sondern um eine Erweiterung, welche flexibel mit den normalen Befehlen gemischt werden kann. Die Thumb-Befehle beinhalten beispielsweise keine Ausnahmebehandlung und benötigen dafür den normalen Befehlssatz.

Um diese kurzen Befehle zu ermöglichen müssen einige Einschränkungen gemacht werden, so nutzen die Thumb-Instruktionen oft ein 2-Adress-Befehlsformat, bei dem eins der Operandenregister gleichzeitig das Zielregister ist [12, S. 202]. Durch das Fehlen einiger Instruktionen braucht das gleiche Programm typischerweise mehr Befehle im Thumb-Format als im klassischen Format, da jeder Befehl aber nur halb so lang ist, wird dennoch Speicher eingespart; außerdem kann die Ausführung auch signifikant schneller werden [12, S. 215].

### **Coprozessor-Anbindung**

Die ARM-Architektur unterstützt die Anbindung zusätzlicher externer Hardware, welche Coprozessoren genannt werden [12, S. 101]. Entgegen dem Namen muss es sich nicht zwangsläufig um echte Prozessoren handeln, sondern allgemein um Hardware, die die Datenverarbeitung des Prozessors in irgendeiner Form unterstützt. Ein Beispiel ist die externe Verarbeitung von Rechnungen mit Fließkommazahlen [12, S. 141]. Als Schnittstelle zu den Coprozessoren dienen bestimmte Registersätze.

---

<sup>2</sup>Erzeugen eines Programms für eine bestimmte Architektur auf einem Host-Computer mit anderer Architektur



### 2.3 Digitaler Schaltungsentwurf

Beim Entwurf digitaler Schaltungen gibt es zwei wesentliche Punkte: den eigentlichen Entwurfs- und Designprozess und die praktische Umsetzung.

Die praktische Umsetzung gliedert sich in zwei Methoden für sehr verschiedene Anwendungsfälle. Die klassische Herstellung integrierter Schaltkreise in Silizium ist sehr aufwendig, kostspielig und nur geeignet, wenn eine sehr große Anzahl Schaltkreise hergestellt werden soll oder ein fast unbegrenztes Budget zur Verfügung steht. Die andere Methode ist die Nutzung von konfigurierbaren Logikbausteinen. Diese sind deutlich preisgünstiger und wiederverwendbar, wodurch sie sich vor allem für Entwicklungsaufgaben und den Entwurf von Prototypen eignen.

Der Entwurfs- und Designprozess kann auf den bereits erläuterten Abstraktionsebenen stattfinden, wobei die mögliche Komplexität der entworfenen Schaltung mit der Abstraktion steigt. So wird kaum ein komplexes Schaltungsprojekt Transistor für Transistor zusammengesteckt, stattdessen ist das Verhalten der Schaltung der Ausgangspunkt des Entwurfs. Hier übernehmen die Hardwarebeschreibungssprachen mit den dazugehörigen Werkzeugen die Aufgabe, die Verhaltensbeschreibungen in Schaltungen zu übersetzen.

Die Übersetzung, auch Synthese genannt, ermöglicht die Umsetzung von mehr und größeren Entwurfsprojekten, da der Entwickler durch diese Automatisierung nicht mit der Schaltung an sich, sondern eher mit der Funktionalität des Gesamtsystems beschäftigt ist.

#### 2.3.1 Programmierbare Hardwarebausteine

Es gibt verschiedene Arten programmierbarer oder konfigurierbarer Logikbausteine. Sie sind entweder einmalig oder mehrfach konfigurierbar und können so flexibel für verschiedene Aufgabenstellungen eingesetzt werden. Einmal konfigurierbare Bausteine werden hier nicht näher behandelt, da sie in der Regel vom Hersteller vorkonfiguriert sind und sich nicht für Entwicklungszwecke eignen.

Allen Arten gemein ist, dass sie darauf basieren, die Gesamtfunktion einer Schaltung in kleinere, logische Funktionseinheiten zu unterteilen. Diese Gesamtfunktion lässt sich fast immer aufteilen in Speicher und kombinatorische Schaltungen, wobei die kombinatorischen Schaltungen auch durch boolesche Gleichungen oder Wahrheitstabellen dargestellt werden können. Für die Darstellung als Gleichung bietet sich eine Normalform an, da hierbei eine einheitliche Darstellung festgelegt wird. Üblicherweise wird die disjunktive Normalform gewählt. Eine umfangreiche Erklärung der verschiedenen Darstellungsarten findet sich beispielsweise in [17, S. 119].

#### PLD

Liegt eine boolesche Gleichung in disjunktiver Normalform vor, so sind die Eingangswerte in einer festgelegten Struktur nur über die elementaren Funktionen AND und OR verknüpft. Diese Struktur macht sich das PLD (Programmable Logic Device) zunutze. Während alle möglichen

## 2. THEORETISCHER HINTERGRUND

Verknüpfungen zwischen den negiert und nicht negiert vorliegenden Eingangswerten umsetzbar sind, werden nur die Verbindungen der Matrix tatsächlich konfiguriert.

Diese Verbindungen (in Abbildung 6 als Schrägstrich dargestellt) können entweder über einen EPROM-Brenner konfiguriert werden oder über eine ISP-Schnittstelle, die Abkürzung steht für In System Programming [21].

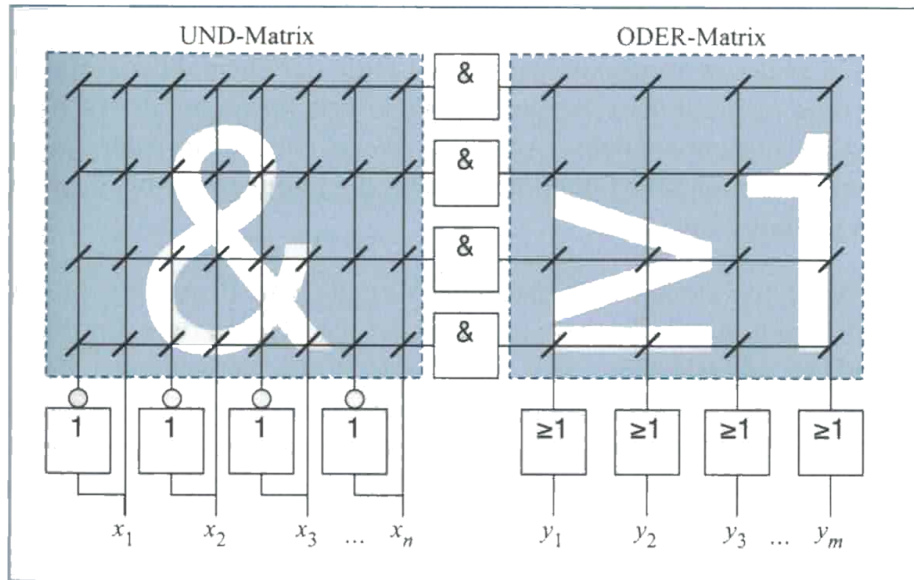


Abbildung 6: Aufbau einer konfigurierbaren logischen Matrix nach Hoffmann [17]

### CPLD

Die CPLDs (Complex PLDs) stellen eine Erweiterung der PLDs dar. Hierbei werden mehrere PLDs über eine Verbindungsmatrix zusammengeschaltet [24, S. 711], wobei nicht nur die Bausteine selbst konfigurierbar sind, sondern auch die Matrix selbst.

Ein Vorteil, abgesehen von der größeren umsetzbaren Komplexität, ist, dass Eingangssignale direkt an den Ausgang weitergeleitet werden können und anders als beim PLD nicht die Matrizen durchlaufen müssen, wenn das Signal nicht verändert werden soll. Das kann den gesamten Systemtakt positiv beeinflussen, da die Signallaufzeit um die übersprungenen Gatterlaufzeiten reduziert wird und sich der Systemtakt immer nach der längsten Signallaufzeit richten muss.

Die Anbindung des gesamten CPLDs nach außen verläuft über IOBs (Input Output Blocks), die nicht nur das jeweils benötigte Signal nach außen leiten, sondern auch die Anpassung des Spannungspegels ermöglichen [21].

### FPGA

Beim FPGA (Field Programmable Gate Array) spielt die Darstellung der Funktion als Wahrheitstabelle eine große Rolle [24, S. 716]. Diese wird hardwaretechnisch als LUT (Look-Up-Table) umgesetzt, wörtlich übersetzt also als Nachschlagetabelle. Diese Nachschlagetabelle ist implementiert als Speicher, bei dem die Eingangswerte der Wahrheitstabelle als Adresse interpretiert werden und der Ausgangswert oder die Ausgangswerte an der entsprechenden Adresse hinterlegt sind [21]. Der Speicher kann dabei flüchtig (SRAM) oder nicht flüchtig (Flashspeicher) sein, abhängig davon sind auch die Wahrheitstabellen flüchtig oder nicht [21].

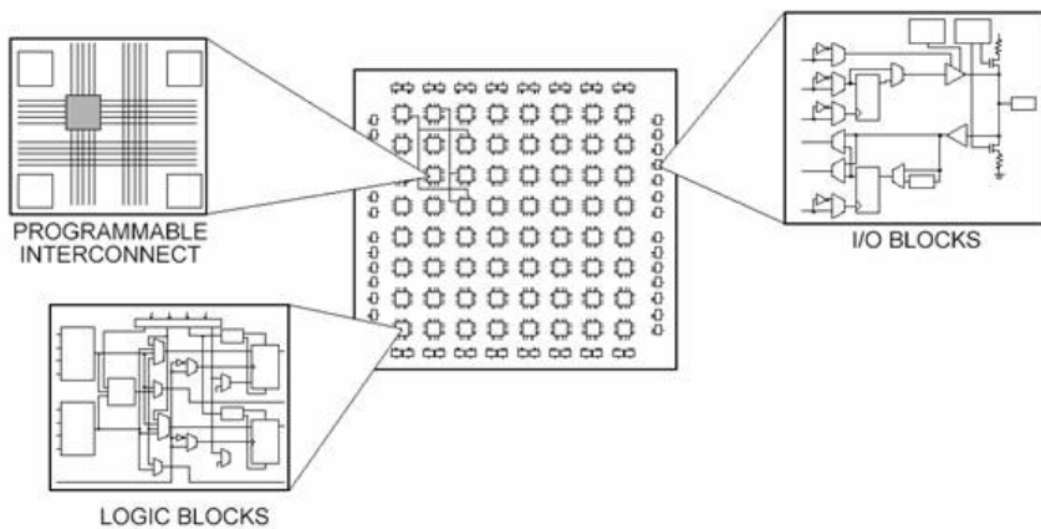


Abbildung 7: Aufbau eines FPGA nach [2]

Diese LUTs ergeben zusammen mit logischen Matrizen und Flipflops als Speicher sogenannte Configurable Logic Blocks (CLBs), welche das Kernelement eines FPGAs sind. Typischerweise werden diese CLBs um Speicher, I/O-Blöcke, Multiplizierer und ein DCM (Digital Clock Management) erweitert; je nach Hersteller und Bedarf können auch fertige Prozessoren zur Signalverarbeitung, Netzwerkschnittstellen zur Kommunikation nach außen oder andere zusätzliche Hardware vorhanden sein [21]. Dieses gesamte System wird in einem einzigen Baustein als SoC (System on Chip) zur Verfügung gestellt, der Einbau in ein Entwicklungsboard mit zusätzlicher Peripherie ist hier noch nicht berücksichtigt.

Das genannte DCM übernimmt dabei die wichtige Rolle der Taktverteilung. Da die Laufzeiten der einzelnen internen Bausteine bei wenigen Nanosekunden liegen, ist es wichtig, dass das Taktsignal die Bausteine möglichst synchron erreicht, es soll eine größtmögliche Gleichzeitigkeit hergestellt werden. Dadurch kann das gesamte System mit einer möglichst hohen Taktfrequenz betrieben werden. Um das zu realisieren wird der Takt in einer verzweigten Baumstruktur verteilt. Dies ist auch der Grund dafür, warum auf einem FPGA nicht unbegrenzt viele Taktsignale vorhanden sein dürfen.

Die Schnittstelle, mit der FPGAs konfiguriert und getestet werden können, ist der JTAG-Standard der IEEE [22, S. 190]. Diese als großes Schieberegister ausgelegte Testschnittstelle ist in der Lage, Konfigurationsdaten in den FPGA zu schreiben oder Analysedaten herauszulesen. Weiterführende Informationen hierzu finden sich beispielsweise bei [12, S. 221].

### **Anwendung konfigurierbarer Logik**

Da FPGAs oft in der Entwicklung und zu Bildungszwecken eingesetzt werden, sind sie häufig in entsprechende Entwicklungsboards eingebunden. Diese nutzen verschiedene Peripherieelemente wie LEDs, Schalter und Buttons, um dem Nutzer Ein- und Ausgaben zu ermöglichen und Testfälle zu entwickeln. Außerdem können sie bequem mit den dazugehörigen Entwicklungsumgebungen konfiguriert werden und mit Schnittstellen versorgt sein, die eine Kommunikation mit externen Geräten ermöglichen. Eine übliche Schnittstelle ist beispielsweise der Standard RS232, welcher die elektrischen und mechanischen Spezifikationen zur Übertragung vorgibt [10].

Grundsätzlich empfiehlt es sich, die Struktur der Hardwareplattform bei der Entwicklung der Hardwarebeschreibung zu berücksichtigen. Sind nutzbare zusätzliche Ressourcen vorhanden, können diese unter Umständen nur bei expliziter Benennung oder der Einhaltung bestimmter Formvorgaben genutzt werden. Zusätzlich sind Synthesewerkzeuge nicht unbedingt in der Lage, eine unnötig lange und schlecht strukturierte Beschreibung zu optimieren; die genutzte Hardware und die dementsprechend genutzte Entwicklungsumgebung sollten also berücksichtigt werden [21].

Bekannte Hersteller sind hier unter anderem Xilinx, deren Hard- und Software auch in dieser Arbeit verwendet werden, Altera und Actel [22, S. 190].

Der Nachteil der FPGAs gegenüber gefertigten Schaltkreisen ist, dass keine Möglichkeit besteht, die Chipfläche zu optimieren, da diese beim FPGA fest vorgegeben ist, lediglich die Auslastung dieser Fläche ist variabel. Außerdem lässt sich die Geschwindigkeit nur bis zu einem bestimmten Grad beeinflussen, da die maximale Frequenz durch die Clock Domain des FPGAs und die auf dem Entwicklungsboard vorhandenen Oszillatoren begrenzt ist.

Ein Vorteil dieser Technologie, der sich aus den bereits Genannten ergibt, ist die Rekonfigurierbarkeit der Hardware. So kann, oft auch bei fest verbauten FPGAs, die Hardwarekonfiguration unter dem Eindruck neuer Anforderungen oder zur Fehlerbehebung geändert werden, teilweise sogar aus großer Distanz. Ein bemerkenswertes Beispiel dafür wird in [19] vorgestellt, wobei hier die Hauptfunktionalitäten eines Satelliten in der Beschreibungssprache VHDL auf einem FPGA implementiert sind und von Bodenstationen aus neu konfiguriert werden können.

### **2.3.2 Hardwarebeschreibungssprache VHDL**

Eine Hardwarebeschreibungssprache (HDL, Hardware Description Language) ist eine Sprache, mit der Schaltungen beschrieben werden können. Hierbei muss klar zwischen der Hardwarebeschreibung und einer Programmierung mit einer Programmiersprache unterschieden werden. Während

mithilfe einer Programmiersprache, eines Compilers und eines Linkers ein Programm generiert wird<sup>3</sup>, wird mit einer HDL eine abstrakte Beschreibung einer digitalen Schaltung erzeugt. Aus dieser Beschreibung entsteht dann mithilfe eines Synthesewerkzeugs eine Konfigurationsdatei, welche einen FPGA so konfiguriert, dass er sich wie die beschriebene Schaltung verhält. Wenn in dieser Arbeit die Begriffe Programm oder Programmierung verwendet werden, beziehen sie sich entsprechend der Erläuterung immer auf die Software für den Prozessor und nicht auf dessen Hardwarebeschreibung.

Auch wenn es sich nicht um eine Programmiersprache handelt, so bietet es sich dennoch an, zum Verständnis Parallelen zwischen beidem zu ziehen und Unterschiede hervorzuheben. Die verwendeten Strukturen weisen oft eine große Ähnlichkeit auf, so dass die Vermutung naheliegt, dass Hardwarebeschreibungssprachen historisch auch von verbreiteten Programmiersprachen beeinflusst wurden.

Es gibt zwei verbreitete Hardwarebeschreibungssprachen, eine davon ist Verilog, die andere, welche hier verwendet wird, ist VHDL. Die Abkürzung VHDL steht für VHSIC Hardware Description Language, wobei VHSIC die Abkürzung für Very High Speed Integrated Circuit ist.

### 2.4 Multicore-Architektur

Da es mit den vorhandenen Ressourcen möglich sein wird, mehr als einen Prozessorkern zu erzeugen, werden die Grundzüge der Multicore-Architektur an dieser Stelle erläutert.

Zunächst muss unterschieden werden zwischen den Begriffen Multicore und Multiprozessor. Bei Multiprozessorarchitekturen sind tatsächlich mehrere physikalische Prozessoren vorhanden, welche komplett unabhängig voneinander arbeiten können und jeder über eigene Schnittstellen nach außen und eigenen Speicher verfügt. Bei jedem einzelnen kann es sich wiederum um einen Multicore handeln. Multicore hingegen bedeutet, dass mehrere Prozessorkerne auf einem Chip vorhanden sind.

Diese Unterscheidung wird in der Literatur nicht überall oder nur teilweise gemacht, oft werden die Begriffe auch synonym verwendet, gemeint ist hierbei üblicherweise, wie in [15, S. 345], das Vorhandensein von mehreren Kernen auf einem Chip.

Damit die Kerne miteinander kommunizieren können, sind mehrere Methoden möglich. Grundsätzlich verläuft die Kommunikation über gemeinsamen Speicher; dabei kann es sich um den Hauptspeicher handeln, um geteilten Cache oder über einzelne Registerspeicher, über die Nachrichten ausgetauscht werden können [23].

#### 2.4.1 Homogene Multicores

Bei homogenen Multicores ist der gleiche Prozessorkern mehrfach vorhanden, es liegen also mehrere identische Kerne vor. Diese können entweder verschiedene Aufgaben lösen oder zur Lösung einer Aufgabe zusammenarbeiten. Dazu muss entweder die Software schon in der Programmierung

---

<sup>3</sup>Den genauen Ablauf der Entstehung eines Programms hier zu erläutern wäre zu umfangreich, in der Fachliteratur finden sich dazu umfangreiche Informationen, zum Beispiel in [26].

diese Parallelisierung unterstützen oder der Compiler muss das Programm entsprechend übersetzen. Durch die gleiche Struktur aller Kerne ist diese Art Multicore-Architektur am einfachsten in Hardware umzusetzen und auf Softwareseite zu nutzen [23].

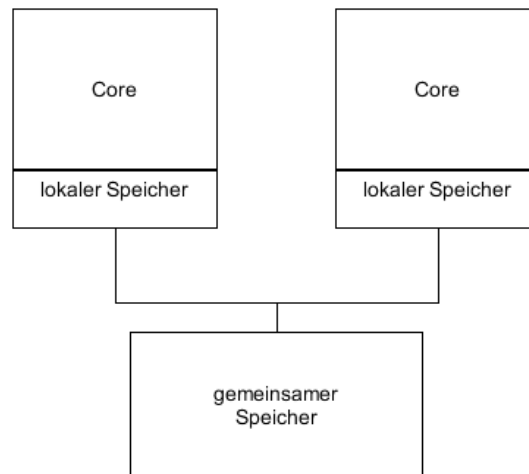


Abbildung 8: Typische Speicheranbindung eines homogenen Multicores

In Abbildung 8 ist eine solche Struktur mit zwei Kernen und einem gemeinsamen Cache-Speicher dargestellt.

### 2.4.2 Heterogene Multicores

Im Gegensatz zu homogenen Multicores sind bei heterogenen Multicores verschiedene Arten von Kernen auf einem Chip zusammengefasst. Das macht den Aufbau des Systems komplexer, ermöglicht aber die Lösung spezieller Probleme durch darauf ausgerichtete Hardware. Zu nennen sind hier kryptographische Aufgaben [23] und die parallele Verarbeitung von großen Datenmengen wie Bild- oder Videomaterial.

### 2.4.3 Softwareperspektive

Aus der Perspektive des Softwareentwicklers wird unterschieden zwischen AMP (Asymmetric Multiprocessing) und SMP (Symmetric Multiprocessing) [25]. Einem AMP-System können sowohl homogene als auch heterogene Multicores zugrunde liegen. Es werden bei diesem Ansatz verschiedene Aufgaben an verschiedene Kerne verteilt, wobei diese entweder auf die jeweiligen Aufgaben spezialisiert sind (heterogen) oder alle gleich aufgebaut sind (homogen). Dabei kann die Differenzierung so weit gehen, dass auf jedem Kern ein unterschiedliches Betriebssystem unabhängig von den anderen Kernen läuft. Auch geteilter Speicher muss nicht zwangsläufig vorhanden sein [25].

Ein SMP-System hingegen muss ein homogenes Multicoresystem sein. Die Cores müssen sich gemeinsamen Speicher teilen, können zusätzlich aber noch eigenen haben. Daraus folgt, dass jeder Core jede Teilaufgabe übernehmen könnte und die Aufgabenverteilung oft sinnvoll durch ein

Betriebssystem gelöst werden kann [25].

### 2.4.4 Multitasking und Multithreading

Um das Vorhandensein mehrerer Kerne nutzen zu können, ist es nötig, mehrere Aufgaben gleichzeitig ausführbar zu machen. Obwohl hier umgangssprachlich oft von Multitasking gesprochen wird, handelt es sich nur darum, wenn je Kern höchstens eine Aufgabe ausgeführt wird. Bei mehr gleichzeitigen Aufgaben ist keine echte Parallelität gegeben, sondern es müssen Kontextwechsel zwischen Aufgaben durchgeführt werden, welche ein Betriebssystem oder mindestens einen Scheduler nötig machen. Das soll hier vermieden werden, da der vorliegende Prozessor hardwareseitig nicht auf Geschwindigkeit optimiert ist [8] und die Nutzung eines Betriebssystems weder nötig noch sinnvoll wäre. Der erzeugte Overhead würde vermutlich zu enormen Geschwindigkeitseinbußen führen, was in Kombination mit der nicht geschwindigkeitsoptimierten Hardware zu einem extrem langsamen bis unbenutzbaren Gesamtsystem führen würde.

Außerdem ist anzuzweifeln, ob selbst ein einfaches Betriebssystem mit den gegebenen Ressourcen lauffähig wäre.

Darüber hinaus handelt es sich um eine Umsetzung für eingebettete Systeme, welche ein Betriebssystem nicht zwingend erforderlich macht, sondern eher für einfache, hardwarenahe Aufgaben verwendet wird. Das fällt besonders ins Gewicht, da der erzeugte Prozessor in der Lehre zur Entwicklung und Anbindung von Hardwarebausteinen in VHDL verwendet werden soll, wobei kein Betriebssystem verwendet wird und nicht einmal zwangsläufig in eine höhere Programmiersprache abstrahiert werden soll, sondern durchaus auch Assembler-Programme zum Einsatz kommen können.

## 3 ARMv4 Prozessor als FPGA-Umsetzung

Nach der theoretischen Einführung beschäftigt sich dieses Kapitel mit der konkreten Umsetzung eines ARM-Prozessors, der in dieser Arbeit verwendet wird.

Die ARMv4-Mikroarchitektur, auf der diese Arbeit basiert, wurde von Herrn Carsten Böhme im Rahmen einer Diplomarbeit an der Technischen Universität umgesetzt. Die Implementierung erfolgte in VHDL für das Entwicklungsboard Spartan 3E der Firma Xilinx. Daher sind alle in diesem Kapitel erläuterten Spezifikationen dieser Umsetzung der genannten Diplomarbeit und der beigelegten Dokumentation entnommen [8].

In diesem Kapitel wird die Struktur des implementierten Prozessors als Ausgangsstatus erläutert, außerdem werden technische Besonderheiten erklärt und auf die Programmierung des entstandenen Prozessors eingegangen.

### 3.1 Struktur

Die Implementierung gliedert sich in verschiedene Funktionsblöcke, welche im Modul `ArmTop` zusammengefasst sind. Das Modul `ArmCore` umfasst dabei den eigentlichen Prozessorkern, das `ArmMemInterface`, kurz für Memory Interface, übernimmt die Schnittstelle zum Hauptspeicher. Die Kommunikation nach außen übernimmt das Modul `ArmRS232Interface`, welches es ermöglicht, ein Programm in Form einer Binärdatei in den Speicher zu schreiben und Ausgaben auf einen UART-Anschluss (näher erläutert in Kapitel 4.3.2) umleitet. Die Funktion des Controllers und Bus Arbiters übernimmt das Modul `ArmSystemController`, außerdem beinhaltet es den Taktgenerator `ArmClkGen`, welcher aus dem externen Takt einen internen und einen internen invertierten Takt erzeugt.

Jeder dieser Funktionsblöcke ist in sich weiter untergliedert, wo es nötig oder sinnvoll ist. So enthält beispielsweise das Memory Interface einen Wrapper als Schnittstelle zum Speicher, welcher wiederum die einzelnen Speicherblöcke instanziiert. Das Wort Wrapper bedeutet wörtlich übersetzt Hülle oder Verpackung; der Name beschreibt die Funktion, die eigentliche Speicheranbindung zu verbergen und vom Rest der Implementierung unabhängig zu machen.

Diese Kapselung nach Funktionen macht nicht nur die Struktur übersichtlich und leichter nachvollziehbar, sie trägt auch maßgeblich dazu bei, das System wartbar und anpassungsfähig zu machen. Soll ein bestimmtes Modul verändert werden, so muss nur dieses Modul ausgetauscht und gegebenenfalls die Schnittstelle dazu angepasst werden. Auch die Fehlersuche bei Problemen reduziert sich auf einen Funktionsblock und die dazugehörigen Schnittstellen. Beim Beispiel des Speichers müssen also, wenn der komplette Hauptspeicher ausgetauscht werden soll, nur die eigentlichen Speicherblöcke ersetzt und die Schnittstelle angepasst werden.

Es kann allerdings der Fall auftreten, dass die Weitergabe von Signalen und die Kommunikation innerhalb des Systems komplexer wird. Müssen Module Signale austauschen, welche innerhalb von anderen Funktionsblöcken liegen, so müssen diese zunächst bis zur höchsten Ebene durchgereicht werden und auf der anderen Seite an den empfangenen Funktionsblock weitergegeben werden. Andererseits führt dieser Aufwand dazu, dass keine unbenötigten Signale weitergegeben werden,



da diese Signale auch in der späteren Umsetzung zu Problemen führen könnten.

#### 3.2 Technische Besonderheiten

Die Umsetzung in VHDL verfügt über einige technische Besonderheiten oder Abweichungen gegenüber normalen ARM-Prozessoren, welche nachfolgend erläutert werden.

Ein wichtiges Merkmal ist das Fehlen des gesamten Thumb-Instruktionssatzes. Diese im theoretischen Teil der Arbeit erklärten Kurzbefehle, welche Geschwindigkeits- und Speicheroptimierungen von Programmen durch die kurze Befehlslänge von 16 statt 32 Bit ermöglichen, wurde in der VHDL-Umsetzung nicht implementiert. Der Grund dafür ist, dass der Hersteller ARM Limited die Implementierung des Thumb-Instruktionssatzes im Rahmen der ursprünglichen Arbeit auf Anfrage nicht erlaubt hat [8, S. 6].

Obwohl der vorliegende Prozessor in seiner Hardwareumsetzung nicht auf eine höchstmögliche Ausführungsgeschwindigkeit ausgelegt ist, lässt sich eine leichte Optimierung nicht vermeiden, wenn eine vertretbare Ausführungsgeschwindigkeit erreicht werden soll. Von den im Kapitel 2.2.2 erläuterten Optimierungsmethoden wurde hier das Pipelining umgesetzt, wobei fünf Pipelinestufen umgesetzt wurden [8, S. 80], angelehnt an die klassischen ARM-Pipelinestufen.

Eine weitere Besonderheit ist die Implementierung der Coprozessorschnittstelle. Diese Möglichkeit, zusätzliche Hardware über Kommunikationsregister anzubinden, wurde nicht nur umgesetzt, sondern auch mit Hilfe eines existierenden Coprozessors realisiert. Dieser NoC (Network on Chip) Prozessor ermöglicht die Anbindung an ein Netzwerk.

#### 3.3 Programmierung

Die ISA der Implementierung des ARM-Prozessors orientiert sich recht streng an der offiziellen ARM-ISA, sodass sich aus Sicht des Software-Entwicklers kaum Unterschiede ergeben. Es kann ein normaler ARM-Crosscompiler verwendet werden, welcher beispielsweise unter Linux auf einem Host-Rechner eine ausführbare Binärdatei für einen ARM-Prozessor erzeugt.

Um diese zu übertragen, gibt es mehrere Möglichkeiten. Zunächst muss eine hardwareseitige Verbindung zwischen dem Entwicklungsboard und dem Host-Rechner, auf welchem die Binärdatei erzeugt wurde, hergestellt werden. Das erfolgt beim Spartan-Board über eine RS232-Verbindung mit DSub-Steckern. Beim Zynq-Board gestaltet sich diese Verbindung in Ermangelung anderer Anschlüsse etwas komplizierter, Details dazu sind in Kapitel 4.3.2 zu finden.

Verläuft die Übertragung erfolgreich, so schreibt der Prozessor zunächst die empfangenen Daten zurück und führt anschließend das entsprechende Programm aus.

Vom Rechner, welcher das auszuführende Programm überträgt, wird auf die UART-Verbindung über USB mittels des tty-Ports zugegriffen, dabei kommt ein Adapter zum Einsatz.

Auf den tty-Port<sup>4</sup> kann unter Linux entweder direkt über die Linux-Konsole oder über Terminalprogramme wie minicom zugegriffen werden. Unter Windows besteht diese Möglichkeit mit

---

<sup>4</sup>Die Abkürzung steht aus historischen Gründen für Teletypewriter und bezeichnet serielle Schnittstellen, auf welche unter Linux vom Nutzer zugegriffen werden kann [9, S. 26].

### 3. ARMV4 PROZESSOR ALS FPGA-UMSETZUNG

---

Programmen wie PuTTY. Da sich hier allerdings viele Probleme mit Konfigurationseinstellungen wie einem Software-Handshake, der Baudrate oder der Bitbelegung ergeben können, wurde von Herrn Prof. Carsten Gremzow ein Linux-Terminalprogramm entwickelt, welches nur die für diese Umsetzung nötigen Funktionen erfüllt; es schreibt die Binärdatei bitweise auf den dafür vorgesehenen Ausgang und gibt empfangene Daten auf der Konsole aus. Mithilfe dieses Programms wird auch unnötiger Overhead wie beispielsweise ein Software-Handshake vermieden, welcher vom Prozessor nicht behandelt werden könnte.

### 4 Prozessoranpassung

Dieses Kapitel beschäftigt sich mit der praktischen Umsetzung der Anpassung des ARM-Prozessors. Dafür muss zunächst die gegebene Originalimplementierung auf einem Spartan-3-Board in Betrieb genommen und getestet werden. Anschließend kann die Anpassung auf den Xilinx Zynq FPGA, eingebettet in ein Entwicklungsboard, vorgenommen werden. Hierbei liegt der Schwerpunkt auf der Anpassung des Block-RAM-Speichers, aber auch andere Besonderheiten müssen beachtet werden.

Darüber hinaus ergeben sich viele Aufgaben erst im Laufe des Anpassungsprozesses durch die Unterschiede der Entwicklungsumgebungen. Während die Entwicklung der Reihe Spartan-3 von der Xilinx ISE Design Suite [27] unterstützt wird, gibt es für die neueren Zynq-Boards eine entsprechend neuere Software vom gleichen Hersteller mit dem Namen Vivado. Die Software unterscheidet sich nicht nur durch die verschiedenen Board-Generationen, sondern auch durch unterschiedliche Features und die nicht gleichwertig vorhandene Umsetzung der VHDL-Standards, wie sich in den nachfolgenden Abschnitten zeigt.

#### 4.1 Inbetriebnahme auf Originalplattform Spartan 3

Die Reihe der Spartan-FPGA-Boards vom Hersteller Xilinx umfasst die Board-Familien 3, 6 und 7, welche jeweils mehrere Starter Kits beinhalten. Diese sind für verschiedene Zwecke optimiert, beispielsweise das Spartan 3A DSP für die Nutzung von integrierten DSPs (Digitale Signalprozessoren). Die hier näher erläuterten Plattformen sind logikoptimiert (3E) bzw. für Anwendungsfälle, bei denen nicht-flüchtiger Speicher benötigt wird (3AN - non-volatile) [28].

Allerdings ist diese Board-Familie inzwischen schlicht veraltet, die ältesten verfügbaren Dokumente darüber sind aus dem Jahr 2003 [31], der Händler Digilent als Partner von Xilinx [11] bietet die Spartan-3-Reihe nicht mehr an; begründet wird dies mit dem Vermerk *Retired* [32], was sich als *im Ruhestand* übersetzen lässt.

Der Hersteller stellt zur Hardwareentwicklung in VHDL, zur Synthese, zum Implementieren und zum Konfigurieren der Hardware die Entwicklungsumgebung Xilinx ISE (Integrated Synthesis Environment) bereit.

Diese Boards werden im Bachelorstudium Computer Engineering für einige Module des Hardwareentwurfs verwendet, sind aber aufgrund ihres Alters teilweise nicht mehr funktionstüchtig; auch die dazugehörige Entwicklungssoftware wird auf absehbare Zeit nicht mehr unterstützt werden und kann schon jetzt zu Kompatibilitätsproblemen mit anderen Programmen und dem Betriebssystem führen. Der Hersteller Xilinx empfiehlt für neuere Boards und Projekte dementsprechend die Verwendung der neueren Software [27].

##### 4.1.1 Originalhardware Spartan 3E

Die ursprünglich verwendete Hardwareplattform Spartan 3E Starter Kit stellte sich nach einigen Tests als nicht funktionstüchtig heraus. So ließ sich nicht nur der Prozessor nicht nutzen, auch

einfachste Testkonfigurationen, welche lediglich Schalterpositionen auf den Board-LEDs ausgeben sollten, waren nicht funktionsfähig. Es konnte nicht abschließend ermittelt werden, was die Ursache der Fehlfunktionen ist, da dies unverhältnismäßig viel Zeit in Anspruch genommen hätte.

Stattdessen wurde auf Entwicklungsboards der Reihe Spartan 3A und Spartan 3AN zurückgegriffen, da diese an der Hochschule eingesetzt werden und deshalb im Hardwarelabor ausreichend vorhanden sind. Ein Vorteil hiervon ist, dass diese Boards dem ursprünglich verwendeten sehr ähnlich sind und unter anderem die gleiche Entwicklungsumgebung und den gleichen Block-RAM-Speicher (siehe Kapitel 4.3.1) verwenden. Davon ausgehend musste lediglich die ucf-Datei, in welcher die Pinbelegung zur Verbindung mit Input- und Outputelementen und anderen externen Bausteinen wie dem taktgebenden Quarz festgelegt ist, modifiziert werden.

### 4.1.2 Ersatzhardware Spartan 3AN

Das Board Spartan 3AN konnte wie erwartet nach Anpassung der ucf-Datei ohne weitere Anpassungen in Betrieb genommen werden.

Nachdem sich die synthetisierte Konfigurationsdatei auf das Board aufspielen ließ und sich, soweit das an den Status-LEDs erkennbar ist, erwartungsgemäß verhielt, konnte der Prozessor selbst getestet werden. Hierfür wurde ein kurzes Programm verwendet, welches als Binärdatei übersetzt und mittels UART (siehe Kapitel 4.3.2) auf den FPGA überspielt wurde. Um das Aufspielen zu ermöglichen, muss zunächst ein Schalter umgelegt sein, welcher in der Pin-Belegung des Boards spezifiziert wird. Danach reagiert der Prozessor, indem er die empfangenen Daten zurück sendet und danach das Programm ausführt. Die empfangenen und zurück gespielten Daten sind nicht leserlich, da es sich um eine als Text interpretierte Binärdatei handelt, welcher in dieser Form keinen sinnvollen Output erzeugt, aber es lässt sich daran erkennen, dass etwas empfangen wurde und welche Länge es hat.

Das Programm verursacht eine Ausgabe der Wörter Hallo Welt über UART, welche in einer Endlosschleife wiederholt wird. Die Ausgabe erfolgt kontinuierlich beim Empfänger, bis entweder der Empfang unterbrochen wird oder das Senden mittels Drücken des Reset-Knopfes und Umlegen des Empfangsschalters unterbunden wird.

In Abbildung 9 ist diese Ausgabe zu erkennen.

Zur Unterstützung des Tests dienten die erwähnten Board-LEDs des Entwicklungsboards, da diese unabhängig vom aufgespielten Programm einzelne Statusbits oder auch bis zu acht verschiedene Binärwerte ausgeben können. Sinnvoll ist dies vor allem, wenn ein bestimmter Zustand überwacht werden soll oder ein Ereignis erwartet wird. Im konkreten Test wurden die LEDs genutzt, um den Beginn der UART-Übertragung anzuzeigen und auszugeben, ob die Übertragung vollständig war.

## 4.2 Anpassung auf Zielplattform Zynq

Die Nachfolgenerationen des Spartan-3, Spartan-6 und Spartan-7, welche für den Einsatz im Automobilbereich entworfen wurden, werden laut Herstellerangaben bis mindestens 2027 verkauft

#### 4. PROZESSORANPASSUNG

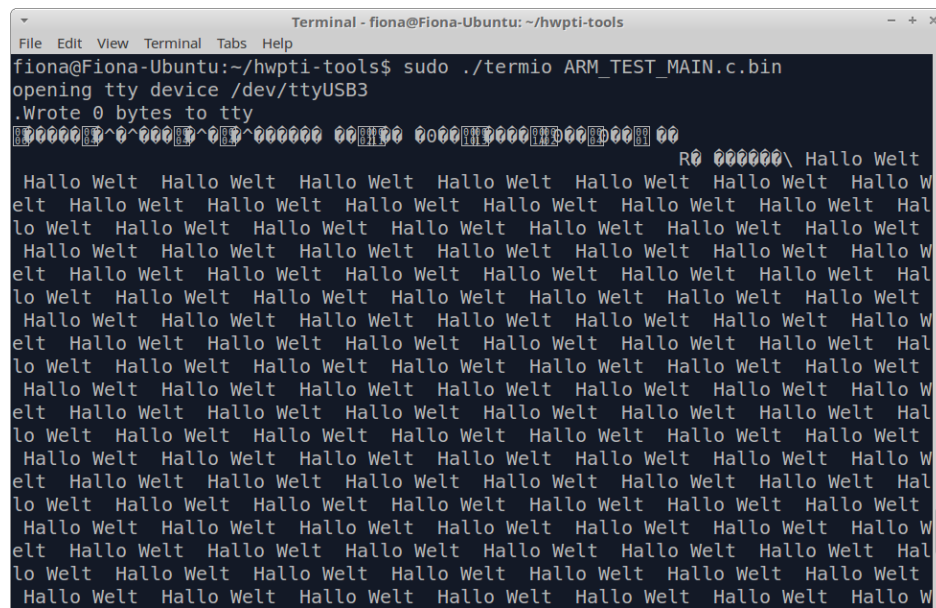


Abbildung 9: Aufruf und Ausgabe des Testprogramms auf dem Spartan 3AN-Board

[33]. Die Spartan-Familie gehört zu den kostenoptimierten Entwicklungsboards von Xilinx, genau wie die Zynq-Familie.

	Spartan-7	Spartan-6	Artix-7	Zynq-7000
Max Logic Cells (K)	102	147	215	444
Max Memory (Mb)	4.2	4.8	13	26.5
Max DSP Slices	160	180	740	2,020
Max Transceiver Speed (Gb/s)	--	3.2	6.6	12.5
Max I/O Pins	400	576	500	250

Abbildung 10: Vergleich der verschiedenen Xilinx-FPGAs [30]

Allerdings fällt im direkten Vergleich auf, dass die Zynq-FPGAs in fast allen Kategorien mehr Ressourcen zur Verfügung stellen als die Alternativprodukte, lediglich im Bereich der I/O-Pins bieten die Alternativen höhere Werte.

Der FPGA, welcher letztendlich verwendet werden soll, ist vom Typ Zynq-7000 XC7Z020-CLG484-1, entwickelt ebenfalls vom Hersteller Xilinx. Eingebettet ist dieser in ein ZED-Board, welches als Entwicklungsboard zum Implementieren und Testen von Prototypen gedacht ist [6]. Die Abkürzung ZED steht für Zynq Evaluation and Development.

Für die Entwicklung mit VHDL bietet der Hersteller die Entwicklungsumgebung Vivado an, welche prinzipiell die gleichen Funktionalitäten wie die ISE bietet, sich in der Umsetzung aber teilweise stark davon unterscheidet.

### 4.3 Notwendige Änderungen

Die wichtigste Änderung, um einen lauffähigen Prozessor zu erhalten, ist die Erzeugung und Anbindung kompatiblen Speichers. Außerdem muss, um den Prozessor programmierbar zu machen, eine Möglichkeit zur Kommunikation zwischen ARM-Prozessor und Host-Rechner geschaffen werden. Darüber hinaus konnte im Rahmen der Anpassungen eine Verdopplung des Systemtaktes erreicht werden.

Diese Änderungen werden hier im Detail erläutert.

#### 4.3.1 Block-RAM Speicher

Der Speicher, der vom Prozessor verwendet wird, lässt sich unterteilen in den Hauptspeicher, welcher größer, aber langsamer ist, und den Registerspeicher, welcher direkt an den Prozessor angebunden ist und daher über schnelle Zugriffszeiten verfügt. Dieser ist jedoch deutlich kleiner.

Der Registerspeicher ist in VHDL in Form von Flipflops realisiert, das Synthesewerkzeug kann diese somit passend zum jeweils verwendeten FPGA generieren, eine weitere Anpassung ist nicht notwendig.

Der Block-Ram Speicher kann hingegen nicht einfach in Form von Bibliotheken oder Ähnlichem in die VHDL-Beschreibung eingebunden werden, da es sich nicht um auf dem FPGA erzeugte Schaltungen handelt, sondern um hardwareseitig schon vorhandenen Speicher, der eingebunden wird. Um das zu tun, bieten sowohl die ISE als auch Vivado ein gesondertes Tool an. In ISE handelt es sich um den Core Generator, in Vivado um den IP Generator, wobei IP hier für Intellectual Property (Geistiges Eigentum) steht und besagt, dass der Hersteller die Anbindung des internen Speichers und die genaue Implementierung hiervon nicht freigibt.

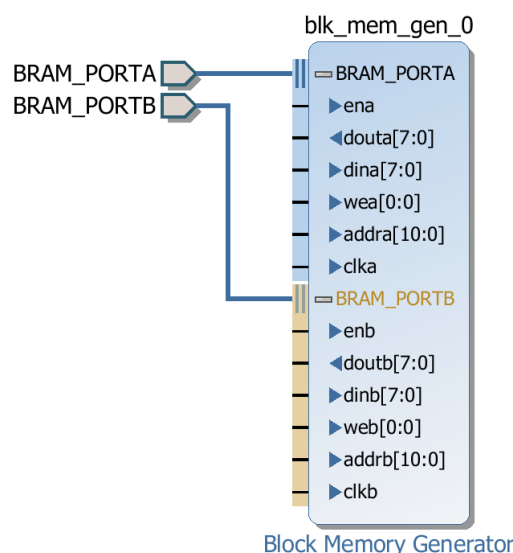


Abbildung 11: Block-RAM-Speicher im Vivado IP Generator

Der angepasste Block-RAM Speicher entspricht genau den Spezifikationen des Originalspeichers. Es muss bei der Schnittstellenimplementierung lediglich die Benennung der einzelnen An-

#### 4. PROZESSORANPASSUNG

schlüsse angepasst werden, da diese in Vivado automatisch anders ist als in der Vorgängersoftware. Der Speicher ist vom Typ True Dual Port, das heißt, es gibt zwei Anschlüsse, über die gelesen und geschrieben werden kann, hier mit der Benennung A und B. Beide Ports haben als Eingänge den Takt, die Adresse, ein Enable-Signal, ein Write-Enable-Signal und den Dateneingang. Der einzige Ausgang ist der Datenausgang.

Beim Testen des Speichers fiel ein interessantes Problem auf: Der durch Vivado generierte und dadurch nur eingeschränkt anpassbare Wrapper enthielt als Write-Enable-Eingang einen `STD_LOGIC_VECTOR(0 to 0)`. Das ist für beide Ports A und B identisch und in Abbildung 11 erkennbar, die Ports sind hier als `wea` und `web` benannt.

Vom Verständnis her handelt es sich hier um ein einzelnes Bit, genau wie der Datentyp `STD_LOGIC`, jedoch umgesetzt als Vektor mit nur einem Element. Das führt dazu, dass dem Eingang nicht ohne Weiteres ein Signal vom Datentyp `STD_LOGIC` zugewiesen werden kann, was die normale Beschreibung eines einzelnen Bits wäre. Dieses Problem ist dem Hersteller seit spätestens 2006 bekannt [35], es wird jedoch keine Lösung, sondern nur eine Umgehung des Problems angeboten. Dieser Workaround sieht die Nutzung eines Hilfssignals vor, um durch den Zugriff auf Index 0 des Write-Enabled-Eingangs das Write-Enable-Bit setzen zu können.

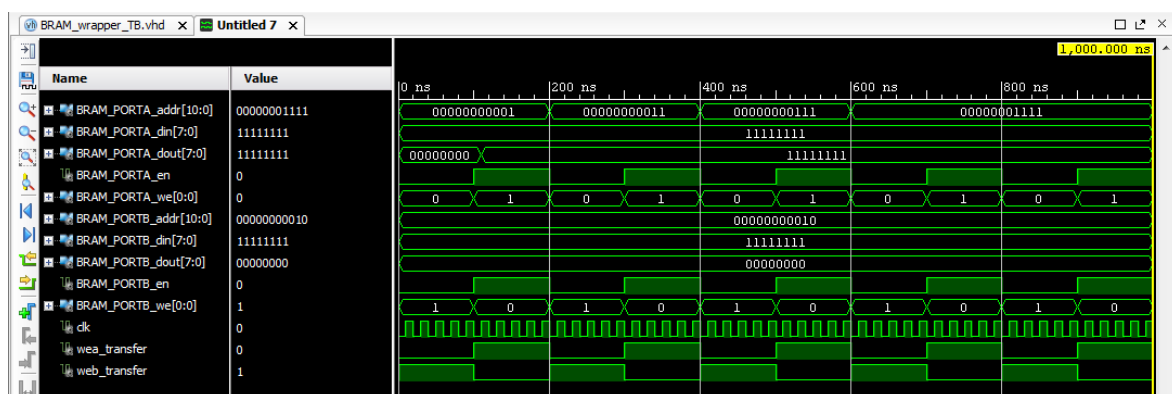


Abbildung 12: Simulation des Block-RAM-Speichers in Vivado

Das Verhalten des Speichers wurde mithilfe einer Testbench in einer Simulation getestet. Hierbei werden die Eingangssignale softwareseitig eingegeben, die Simulationsumgebung bildet das Verhalten der beschriebenen Hardware nach und gibt die dabei erzeugten Ausgangssignale grafisch aus. In Abbildung 12 sind die Simulationsergebnisse dargestellt. Es lässt sich erkennen, dass der Speicher sich wie erwartet verhält, es wird nur geschrieben, wenn der Enable-Eingang und der Write-Enable-Eingang gleichzeitig aktiv sind.

Da die Schnittstelle zwischen Prozessor und Speicher in der Originalversion in einer Wrapper-Datei gekapselt ist, musste lediglich diese angepasst, die alte Speicherimplementierung entfernt und die neue eingebunden werden. In der Umsetzung bedeutet das, dass die Component-Deklaration neu erstellt und die Verdrahtung der Eingänge an diese angepasst werden musste. In dieser einfachsten Variante hat der Speicher die gleiche Größe wie im Originalsystem. Allerdings wird nur ein Bruchteil der auf dem FPGA vorhandenen Fläche genutzt, wie in Abbildung 13 zu erkennen

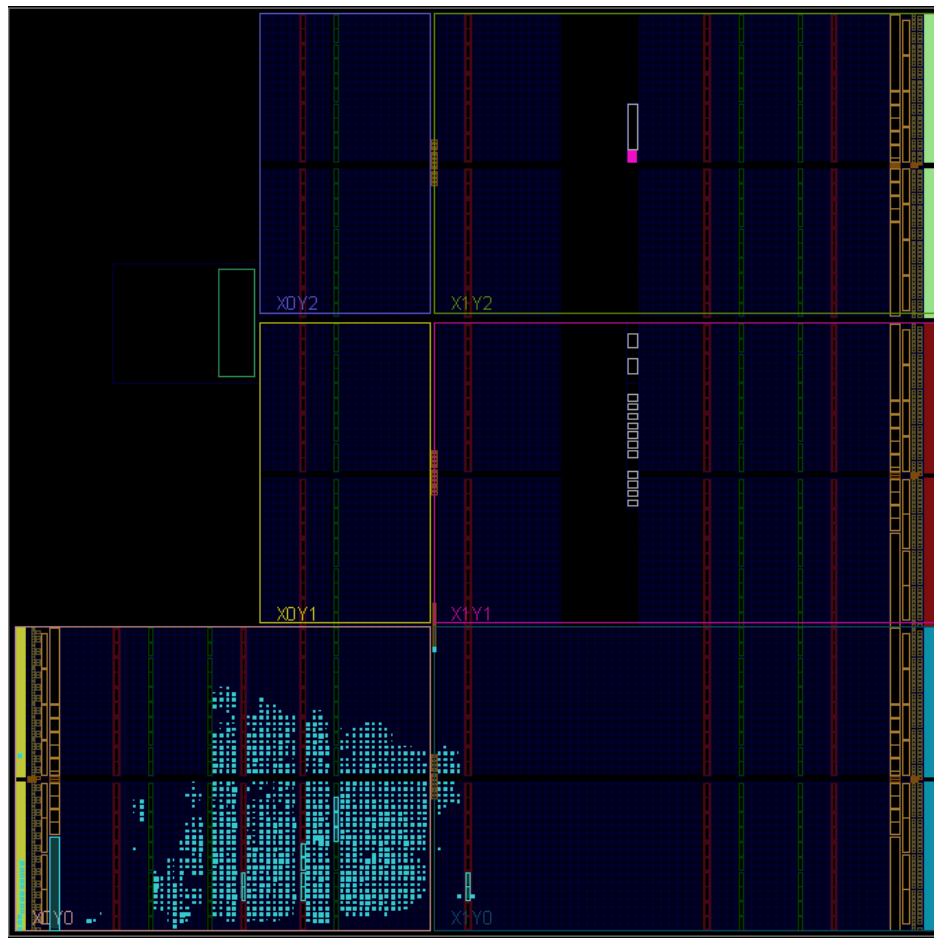


Abbildung 13: Flächenauslastung des Zynq-FPGAs bei einem Prozessorkern

ist. Diese grafische Darstellung der Auslastung wird im Schritt der Implementierung für einen spezifischen FPGA von Vivado erzeugt.

### 4.3.2 UART-Anschluss

Zum Übertragen der Binärdatei (vgl. Kapitel 3.3), welche aus dem Programmcode erstellt wurde, kann in der Originalversion ein RS232-Anschluss genutzt werden, um via UART Daten zu senden und zu empfangen. UART steht für Universal Asynchronous Receiver Transmitter und definiert die Logik der Datenübertragung, während RS232 (RS steht für Recommended Standard) die Spannungspegel und den eigentlichen Übertragungsstandard festlegt.

Bei UART handelt es sich um eine serielle Schnittstelle; das heißt, Datenbits werden nicht gleichzeitig, sondern eins nach dem anderen übertragen. Beginn und Ende einer Übertragung werden durch sogenannte Start- und Stopbits gekennzeichnet. Die Bezeichnung Asynchronus bedeutet, dass Sender und Empfänger nicht auf den gleichen Systemtakt zurückgreifen, sondern unabhängig voneinander senden und empfangen können. Um hierbei eine Kommunikation bezüglich der Datenübertragung zu ermöglichen, also beispielsweise mitzuteilen, dass der Empfänger



#### 4. PROZESSORANPASSUNG

keine Daten mehr aufnehmen kann, gibt es weitere Leitungen, die solche Signale austauschen. Diese werden hier allerdings nicht verwendet, da sie bei den zu übertragenden Datenmengen schlicht nicht nötig sind.

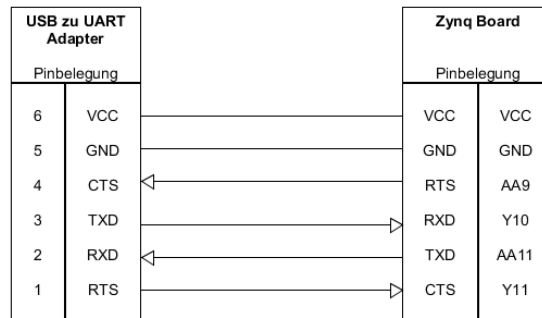


Abbildung 14: UART-Verbindung zwischen Zynq-Board und Adapter

Während bei den Spartan-Boards ein RS232-Anschluss vorhanden ist, um über UART Programme auf den Prozessor zu spielen, fehlt ein Anschluss in dieser Form bei den Zynq-Boards. Er ist zwar physisch vorhanden, aber mit dem bereits vorhandenen UART-Anschluss des auf dem Entwicklungsboard vorhandenen ARM-Prozessors fest verbunden. Daher kann er von Seiten des FPGAs nicht genutzt werden. Eine Alternative bieten allerdings die GPIO-Anschlüsse, welche vom FPGA aus beliebig belegt werden können. Diese Möglichkeit wurde genutzt, um die UART-Anschlüsse des ARM-Prozessors nach außen zu legen. Hierbei sind von Seiten des Prozessors nur die Grundfunktionalitäten RX (Receive Data, Empfangen) und TX (Transmit Data, Senden) benötigt. Da der Adapter vom Hersteller Pmod allerdings einen Hardware-Handshake nutzt, sind die Handshake-Leitungen zusätzlich belegt worden. Hierbei teilt der jeweilige Sender über den RTS-Pin (Ready To Send) seine Sendebereitschaft mit und empfängt über CTS (Clear To Send) die Sendebereitschaft des anderen Kommunikationsteilnehmers. Alle Leitungen sind Low-aktiv, das heißt, ein niedriger Pegel steht für den logischen Wert eins.

Aufgrund der oberflächlichen Dokumentation des Adapters mussten zunächst durch Messungen mit dem Oszilloskop die Pinbelegungen ermittelt werden. Danach stellte sich heraus, dass trotz empfangener Daten keine Antwort erfolgte. Der Adapter konnte als Fehlerquelle ausgeschlossen werden, ebenso der Speicher, welcher wie beschrieben getestet worden war. Die Übertragung des Programms über die serielle Schnittstelle ist in Abbildung 15 zu erkennen.

Auch die eigentliche Prozessorimplementierung war grundsätzlich korrekt, da diese ja in der Originalumsetzung funktionstüchtig ist. Mithilfe der Ausgabe von entsprechenden Informationen beim Senden und Empfangen auf die Board-LEDs konnte festgestellt werden, dass zwar Daten empfangen, aber nicht gesendet werden.

Als Fehlerquelle wurden in VHDL implementierte globale Signale identifiziert. Diese werden für die originale RS232-Kommunikation benutzt. Globale Signale wurden in der Xilinx ISE unterstützt, in Vivado allerdings nicht. Das Problem tritt bei der Portierung von Beschreibungen in die neue Entwicklungsumgebung auf und sorgt dafür, dass keine fehlerfreie Synthese mehr möglich ist. Es



Abbildung 15: Messung der Übertragung des Programm-Binärfiles am Oszilloskop

handelt sich hierbei um ein reines Software- und Beschreibungsproblem, die Hardware der Boards ist unabhängig hiervon. Die globale Verwendung von Signalen ist zwar in VHDL vorgesehen, allerdings hält Xilinx diese für riskant; die fehlende Unterstützung in Vivado ist demnach beabsichtigt. Darüber hinaus gibt es laut offizieller Stellungnahme auch keine Pläne, diese Funktion zukünftig zu unterstützen [34].

Um die nicht mehr vorhandenen globalen Signale dort verfügbar zu machen, wo sie benötigt werden, gibt es mehrere Möglichkeiten. Entweder werden aus den Signalen Konstanten gemacht, welche Vivado auch global unterstützt oder die Werte, sofern sie sich nicht ändern, werden in jeder Datei, in der sie benötigt werden, lokal verfügbar gemacht. Diese beiden Varianten funktionieren nur, wenn sich der dem Signal zugewiesene Wert innerhalb der Beschreibung nicht ändert, also die Signalfunktion eigentlich nicht benötigt wird. Ändert sich der zugewiesene Wert, so muss das Signal bei allen Modulen, die es verwenden, zu den Ein- und Ausgabeports hinzugefügt und entsprechend verdrahtet werden. Das führt dazu, dass gegebenenfalls auch übergeordnete Module, die das Signal nicht verwenden, aber ein Modul enthalten, welches es verwendet, das Signal in der Portbeschreibung enthalten müssen. Allerdings ändert sich im Anwendungsfall der zugewiesene Wert nicht, es handelt sich um die Konfiguration der für die UART-Kommunikation verwendeten Registeradressen. Diese werden ursprünglich im Konfigurationsfile festgelegt und können mit einer der beschriebenen Methoden geändert werden.

Darüber hinaus muss auf Seiten des Rechners, von dem das Programm auf den Prozessor aufgespielt wird, die Baudrate von den originalen 57600 Baud auf 115200 Baud angepasst werden, da das Zynq-Board mit der doppelten Geschwindigkeit betrieben wird, wie im folgenden Abschnitt beschrieben wird.

Nach diesen Anpassungen konnte der Prozessor auch auf dem neueren Entwicklungsboard in

Betrieb genommen und über UART programmiert werden. Es wurde das gleiche Testprogramm wie zur Inbetriebnahme des Spartan-Boards verwendet, die Ausgabe ist dementsprechend identisch zu der vorherigen.

### 4.3.3 Anpassung des Systemtaktes

Die Entwicklungsboards der Spartan-Reihe verfügen über einen 50 MHz Oszillator, welche über die Pinconfiguration mit dem FPGA verbunden wird. Dieser kann als externer Taktgeber des Prozessors genutzt werden, das entsprechende Signal in der Implementierung wird als EXT\_CLK (extern Clock) bezeichnet.

Der auf dem ZED-Board zum Einsatz kommende Oszillator erzeugt einen Takt von 100 MHz, also der doppelten Betriebsfrequenz. Diese konnte ohne weitere Anpassungen verwendet werden, die neuere Hardware ermöglicht auch für die kombinatorischen Schaltungen eine höhere Betriebsfrequenz.

Genau lässt sich das mithilfe einer Timing-Analyse des synthetisierten Designs begründen. Diese wurde für die alte Version mit der Xilinx ISE durchgeführt, da Vivado keine Maximalfrequenz eines synthetisierten Designs errechnet. Dafür mussten manuelle Definitionen, sogenannte Timing-Constraints, verwendet werden, in denen festgelegt wird, welches Signal als Taktsignal zu betrachten ist. Wird das nicht gemacht, wertet die ISE das Signal EXT\_CLK aus, welches intern im Prozessor nicht verwendet wird, weshalb absurd hohe Zielfrequenzen angegeben werden, die praktisch der Maximalfrequenz des FPGAs entsprechen.

Für das neue Design musste in Vivado ebenfalls mit Constraints gearbeitet werden, diese geben allerdings keine Maximalfrequenz an, sondern nur, ob die manuell festgelegte Zielfrequenz eingehalten wird. Als Zielfrequenz wurden 100 MHz angegeben, welche laut Vivado erreicht werden können. Die Definition der Zielfrequenz erfolgte mit dem Befehl

```
create_clock -period 10.000 -name SYS_CLK -waveform {0.000 5.000}
```

Hierbei ist nicht die Zielfrequenz direkt angegeben, sondern die Periodendauer in Nanosekunden (hier 10 ns).

### 5 Konzept und Umsetzung einer Multicore-Architektur

Durch die auf dem Zynq-Board vorhandenen Ressourcen ist es möglich, mehr als einen Prozessor zu implementieren. Das kritische Element beim Spartan-Board sind die LUTs, welche im Vergleich zum Spartan auf dem Zynq-Board reichlich vorhanden sind.

#### 5.1 Konzepte

Die Prozessorkerne müssen miteinander kommunizieren, was in irgendeiner Form über gemeinsamen Speicher realisiert werden muss. Dabei kann es sich um den Hauptspeicher, bestimmte Speichersegmente oder auch einzelne Register oder Flipflops handeln. Es sind verschiedene Konzepte für die Umsetzung möglich, welche hier erklärt werden.

##### 5.1.1 Gemeinsamer Hauptspeicher

Entscheidet man sich dafür, den Hauptspeicher für mehrere Prozessorkerne erreichbar zu machen, stellt sich die Frage der Synchronisation. Es darf nicht der Fall auftreten, dass mehrere Kerne zeitgleich oder kurz nacheinander auf die gleiche Hauptspeicheradresse zugreifen und dadurch ein ungültiges Datum an dieser Adresse steht. Alternativ ist es möglich, für bestimmte Adressen eine Bypass-Regel umzusetzen; das heißt, dass beim Zugriff auf diese Adresse nicht auf den Hauptspeicher zugegriffen wird, sondern direkt von einem Register des einen Kerns in ein Zielregister des anderen Kerns geschrieben wird. Hierdurch werden lange Lese- und Schreibzeiten eingespart, da der Hauptspeicherzugriff eine der langsamsten Operationen ist, und außerdem ein Teil des Synchronisationskonfliktes gelöst, da die Kommunikation über den Hauptspeicher entfällt. Allerdings muss auch bei einer Bypass-Lösung der Speicher in irgendeiner Form aufgeteilt werden; das kann zeitlich (über eine Synchronisation) oder räumlich (über die Zuordnung bestimmter Adressbereiche) geschehen.

Es handelt sich bei dieser Umsetzung um eine homogene Multicore-Architektur, allerdings mit der Einschränkung, dass der Zugriff auf den Hauptspeicher für jeden Kern individuell angepasst werden muss.

Der entscheidende Nachteil dieses Konzeptes ist die genannte Synchronisation von Speicherzugriffen und die aufwendige Aufteilung des Speichers, weshalb von der Umsetzung dieses Konzeptes abgesehen wurde.

##### 5.1.2 Getrennter Hauptspeicher

Alternativ bietet sich die Möglichkeit, für jeden Kern einen eigenen Block-RAM-Speicher anzulegen, welcher genauso groß ist wie in der ursprünglichen Prozessorversion. Die Kommunikation erfolgt dabei über spezielle Register.

Der Vorteil hierbei ist die einfache Umsetzbarkeit und der geringe Verwaltungsaufwand, da keine weiteren Synchronisationsmechanismen notwendig sind. Der Nachteil ist, dass der Austausch von großen Datenmengen zwischen den Kernen verlangsamt wird, da die zur Kommunikation

genutzten Register hier einen Flaschenhals darstellen. Als Kompromiss wäre es auch möglich, zusätzlich zum jeweils eigenen Speicher einen größeren gemeinsamen Speicher zum Transfer großer Datenmengen anzulegen.

Es handelt sich bei dieser Umsetzung ebenfalls um eine homogene Multicore-Architektur, da alle Prozessorkerne identisch sind.

Wegen der übersichtlichen Struktur wurde dieses Konzept für die Umsetzung gewählt, allerdings in einer Variation, welche später ausführlich erläutert wird.

### 5.1.3 Coprozessor-Anbindung

Eine andere Möglichkeit, weitere Prozessorkerne anzubinden, bietet sich durch eine Besonderheit der ARM-Architektur, die Coprozessorschnittstelle. Diese ermöglicht es, dem Prozessor für spezielle Aufgaben zusätzliche Hardware zur Seite zu stellen.

Anstelle von spezialisierten Hardwarebausteinen lässt sich hier auch ein zweiter, identischer ARM-Prozessor anbinden, der seine Befehle über die Coprozessorschnittstelle erhält. So übernimmt der zentrale Prozessor lediglich die Verwaltung und Steuerung und verteilt arithmetisch-logische Aufgaben einfach weiter. Dieses Konzept ist bei dem vorliegenden Prozessor in der maximalen Ausbaustufe mit zwei Coprozessoren denkbar. Coprozessoren teilen 16 Register mit dem Hauptprozessor, die Implementierung verfügt jedoch über 32 Register, sodass zwei Coprozessorschnittstellen zu einem Coprozessor zugeordnet werden. Werden lediglich 16 Register für die Kommunikation genutzt, so könnten sogar vier Coprozessoren angebunden werden; die Bandbreite der Kommunikation ändert sich dadurch allerdings nicht.

Dieser Sonderfall ist nur durch die spezielle ARM-ISA möglich. Genau genommen handelt es sich dabei nicht um eine homogene Multicore-Architektur, da die Kerne nicht gleichberechtigt sind und auch softwareseitig nicht gleich behandelt werden dürfen. Es handelt sich vielmehr um einen zentralen Hauptprozessor mit zusätzlichen Recheneinheiten, auch wenn alle diese Elemente grundsätzlich die gleiche Struktur haben.

### 5.1.4 Transputer-Konzept

Eine Variante einer Multicore-Architektur mit getrenntem Hauptspeicher ist die sogenannte Transputer-Architektur.

Dieses Konzept aus den 80er Jahren verbindet mehrere Computer über ein Kommunikations-Interface (Link Interface) zu einer Recheneinheit. Dabei hat jeder Computer einen eigenen Prozessorkern und eigenen Arbeitsspeicher, lediglich Signale wie der Takt oder der Reset-Befehl werden zentral verteilt. Das Link Interface sind dabei prozessornahe Register zur Kommunikation, welche entweder als Netzwerk (jeder kommuniziert mit seinen Nachbarn) oder über Knoten (jeder kommuniziert mit jedem) realisiert werden können [18, S. 3 ff.].

Dieses Konzept kann jedoch als FPGA-Umsetzung auch als boardinternes Netzwerk auf einem Chip realisiert werden, sodass lediglich das Kommunikationskonzept über ein Link Interface aus

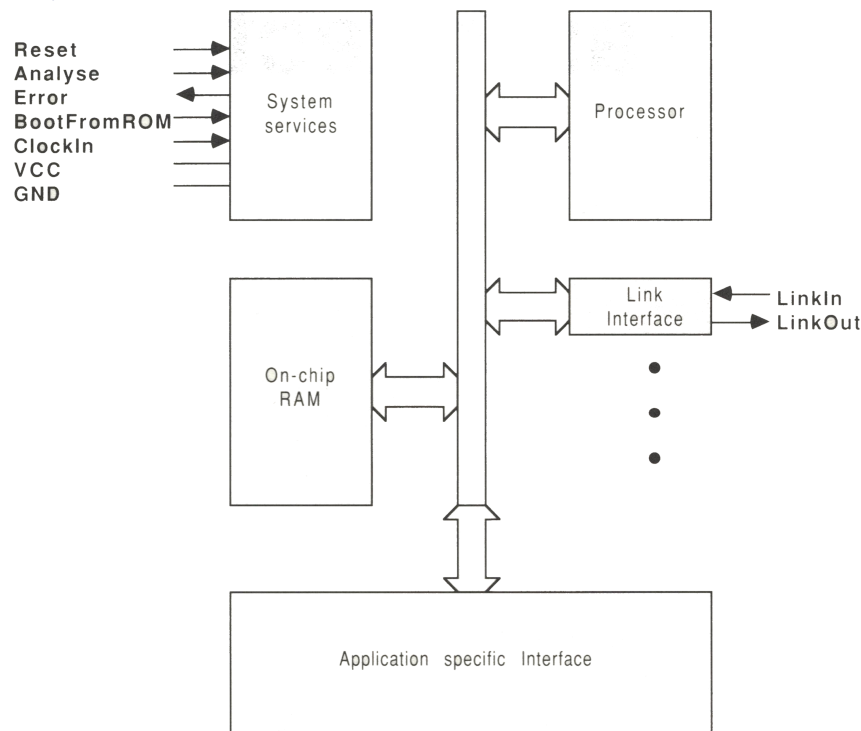


Abbildung 16: Struktur eines Transputers nach [18]

prozessornahen Registern übernommen wird. Der Aufbau ist in Abbildung 16 dargestellt.

### 5.1.5 Software für Multicore-Architekturen

Zusätzlich stellt sich die Frage, wie die Software an die Multicore-Architektur angepasst wird. Das Vorhandensein mehrerer Kerne bringt keine Vorteile, wenn nicht auch mehrere Aufgaben gleichzeitig ausgeführt werden können. Die Ausführung mehrerer Aufgaben auf mehreren Kernen, also ein echtes Multitasking, sollte möglich gemacht werden können. Da besonders die Mechanismen der Speichersynchronisation weit über diese Arbeit hinaus gehen würden, wird das Thema an dieser Stelle nicht weiter behandelt.

## 5.2 Umsetzung

Für die Umsetzung wurde das Transputer-Konzept gewählt. Dafür wurden zunächst acht identische, eigenständige Kerne des Prozessors in der Hardwarebeschreibung implementiert.

```

48 -- interne clk
49 signal INT_CLK      : std_logic;
50 signal INT_INV_CLK  : std_logic;
51 signal DCM_LOCKED   : std_logic;
52
53 component ArmTop is
54     Port (
55         EXT_RST : in  std_logic;
56         INT_CLK : in  std_logic;
57         INT_INV_CLK: in std_logic;
58         EXT_LDP : in  std_logic;
59         EXT_RXD : in  std_logic;
60         EXT_TXD : out std_logic;
61         EXT_LED : out std_logic;
62         DCM_LOCKED: in std_logic
63     );
64 end component;
65
66 component ArmClkGen
67     port(
68         CLKIN_IN      : in std_logic;
69         RST_IN        : in std_logic;
70         CLKFX_OUT      : out std_logic;
71         CLKFX180_OUT   : out std_logic;
72         CLKIN_IBUFG_OUT : out std_logic;
73         LOCKED_OUT     : out std_logic
74     );
75 end component;

```

Abbildung 17: Anpassung der Ports im Top-Modul bei mehreren Kernen

Die oberste Ebene ist dabei das Modul `TransputerTop`, welches acht Module vom Typ `ArmTop` instanziiert und verdrahtet. Außerdem wurde der Clock-Generator, welcher aus dem externen Takt einen internen Takt und einen invertierten internen Takt erzeugt, instanziiert. Ursprünglich war dieser in den `SystemController` innerhalb des Kerns integriert. Das hätte aber die Anzahl der Kerne auf vier limitiert, da das Synthesewerkzeug nur maximal vier verschiedene Takt-domänen auf dem ZED-Board umsetzen kann.

Zusätzlich wurden auch am Kern-Modul selbst einige Veränderungen vorgenommen. Jedem Kern steht jetzt eine Status-LED zur Verfügung, da auf dem ZED-Board nur insgesamt acht LEDs vorhanden sind. Außerdem mussten die beiden nun globalen Taktsignale über das Top-Modul an den Systemcontroller weiter gereicht werden, da dieser die Signale benötigt. Das fügt den Ports vom Systemcontroller und vom Top-Modul einige Signale hinzu. Gleichzeitig wird das `EXT_CLK`-Signal nicht länger benötigt, da der systeminterne Takt nun global erzeugt und den jeweiligen Kernen zur Verfügung gestellt wird. Die Änderung der Ports im Top-Modul ist in Abbildung 17 dargestellt.

Außerdem erzeugt der Clock-Generator ein Signal mit der Bezeichnung `DCM_LOCKED`. Dieses ist ebenfalls in der Abbildung zu erkennen, da es zusammen mit den Taktsignalen weiter gegeben werden muss. Es zeigt anderen Modulen, insbesondere dem Systemcontroller, an, ob der generierte

Takt stabil ist [8, S. 126].

Die Bezeichnung für das nun global erzeugte Taktsignal ist INT\_CLK beziehungsweise INT\_INV\_CLK, da es sich um den internen und internen invertierten Takt handelt.

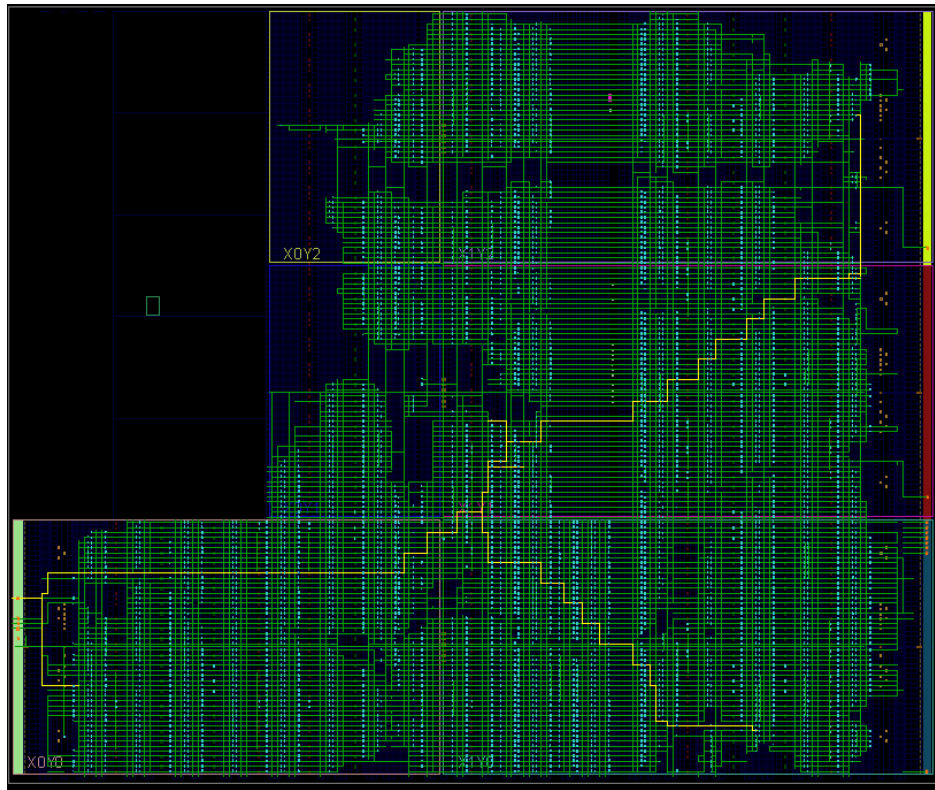


Abbildung 18: Flächenauslastung des FPGA mit acht Kernen

In Abbildung 18 ist erkennbar, dass die Fläche des FPGA mit acht Kernen deutlich besser ausgelastet ist als in der Originalumsetzung. Es konnte somit eine effizientere Nutzung der vorhandenen Ressourcen erreicht werden.

Die erzeugten Kerne funktionieren bislang unabhängig voneinander, aber einzeln fehlerfrei. Durch das Multiplexen der UART-Signale können die Kerne jeweils einzeln angesprochen und getestet werden. Hierbei bietet es sich an, den Multiplexer manuell über die nicht verwendeten Schalter des Entwicklungsboards zu steuern.



## 6 Ergebnisse

### 6.1 Funktionalität

Der Prozessor konnte entsprechend der Anforderungen so angepasst werden, dass er auf dem ZED-Board voll funktionstüchtig ist. Programme, die für die Vorgängerversion oder allgemein ISA-konform entwickelt wurden, sind lauffähig, können jedoch doppelt so schnell ausgeführt werden, da der Systemtakt entsprechend höher ist. Die Kommunikation funktioniert bei angepasster Baudrate des Host-Rechners genau wie beim Vorgängermodell.

Außerdem wurde die Entwicklung komplett mit Vivado durchgeführt, so dass eine Anpassung, Weiterentwicklung oder Nutzung in der Lehre mit dem Werkzeug möglich ist.

Darüber hinaus wurden in einer Variante acht Kerne auf einem Board realisiert, welche alle funktionstüchtig sind und über UART angesprochen werden können.

### 6.2 Ressourcennutzung

Die genutzten Ressourcen, geordnet nach der verwendeten Hardware, werden in Tabelle 1 dargestellt. Dabei werden die im Originalprojekt angegebenen Werte genannt, diese sind [8, S. 140] entnommen. Außerdem wird zur besseren Vergleichbarkeit die Auslastung des Spartan 3E im eigenen Projekt genannt. Obwohl es sich dabei um dieselbe VHDL-Beschreibung handelt, können sich die Werte unterscheiden, da die Synthese mit neueren Softwareversionen optimiert worden ist. Die im Ursprungsprojekt verwendete Version der Xilinx ISE war 9.2 [8, S. 42], aktuell ist Version 14.7 [27]. Außerdem können unterschiedliche Projekteinstellungen das Ergebnis beeinflussen und auch schon kleinere Optimierungen zu unterschiedlichen Ergebnissen führen, da weniger ausgelastete Bausteine oftmals besser verdrahtet werden können.

Des Weiteren wird die Ressourcennutzung auf dem Spartan 3AN angegeben. Auf diesem Board sind zwar nicht bedeutend mehr Ressourcen vorhanden, aber da hiermit die Inbetriebnahme erfolgte, ist die Angabe nötig.

Tabelle 1: Vergleich der Ressourcennutzung auf verschiedenen FPGA-Boards

	Spartan 3E original	Spartan 3E neu	Spartan 3AN	Zynq 1 Kern	Zynq 8 Kerne
Slices	75%	68%	54%	-	-
LUTs	74,9%	62%	50%	7%	54%
Flipflops	19,5%	20%	16%	2%	14%
MMCM	-	-	-	25%	25%
BRAM	-	40%	40%	3%	23%

Schließlich ist die Ressourcenauslastung in der angepassten Version auf dem Zynq-Board angegeben, sowohl in der einfachen Version als auch in der Umsetzung mit acht Kernen. Nicht angegebene Analysewerte werden von der jeweiligen Entwicklungsumgebung nicht zur Verfügung

## 6. ERGEBNISSE

---

gestellt. Die Abkürzung MMCM steht für Mixed Mode Clock Manager und beschreibt die Anzahl der genutzten Bäume zur Taktverteilung, wobei es sich immer nur um einen handelt.

Es ist zu erkennen, dass trotz acht umgesetzter Kerne in der neuen Version das Zynq-Board nicht vollständig ausgelastet ist.

### 7 Fazit und Ausblick

Die ARMv4 Mikroarchitektur in der VHDL-Implementierung konnte erfolgreich an den Zynq-FPGA angepasst werden. Dafür wurde der Speicher neu implementiert und die Speicherschnittstelle entsprechend geändert. Außerdem wurde die UART-Kommunikationsschnittstelle an die neue Hardware angepasst. Auch eine Verdopplung des Systemtaktes konnte erreicht werden.

Der modulare, gekapselte Aufbau des Gesamtsystems konnte dabei beibehalten werden, sodass eine möglichst große Anpassbarkeit nach wie vor gegeben ist. Dies ist nötig, da durch die Nutzung von Entwicklungsumgebungen wie Vivado und damit verbundene IP immer eine Bindung an einen bestimmten FPGA erfolgt und so auch zukünftige Anpassungen an weitere Boards möglich sein müssen.

Außerdem ermöglichte die größere Menge an Ressourcen auf dem Zynq-FPGA die Umsetzung einer Multicore-Architektur mit acht Kernen. Diese ist zwar funktionstüchtig, kann aber noch nicht programmiert werden, was in zukünftigen Projekten realisiert werden könnte.

Darüber hinaus besteht die Möglichkeit, in einer weiterentwickelten Version den einzelnen Prozessorkernen mehr BRAM-Speicher zur Verfügung zu stellen, da selbst in der Multicore-Variante der Speicher des FPGAs nicht annähernd voll ausgenutzt wird.

## **Eidesstattliche Erklärung**

Ich erkläre hiermit an Eides statt, dass

- ich die vorliegende wissenschaftliche Arbeit selbstständig und ohne unerlaubte Hilfe angefertigt habe,
- ich andere als die angegebenen Quellen und Hilfsmittel nicht benutzt habe,
- ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe,
- die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfbehörde vorgelegen hat.

Berlin, 20.08.2018

### Literatur

- [1] AppliedMicro Showcases World's First 64-bit ARM v8 Core. In: *BusinessWire* (2011). – URL <https://www.businesswire.com/news/home/20111027006673/en/AppliedMicro-Showcases-World%E2%80%99s-64-bit-ARM-v8-Core>. – abgerufen am 16.08.2018
- [2] *Wie funktionieren FPGAs?* 2013. – URL <http://www.ni.com/white-paper/6983/de/>. – abgerufen am 16.08.2018
- [3] ARM LIMITED: *architecting a secure and connected world*. 2018. – URL <https://www.arm.com/company>. – abgerufen am 16.08.2018
- [4] ARM LIMITED: *Arm Cortex Processors*. 2018. – URL <https://www.arm.com/products/processors>. – abgerufen am 16.08.2018
- [5] ARM LIMITED: *GNU Toolchain for Arm processors*. 2018. – URL <https://developer.arm.com/open-source/gnu-toolchain>. – abgerufen am 16.08.2018
- [6] AVNET: *Zedboard Datasheet*. URL [http://zedboard.org/sites/default/files/product\\_briefs/5066-PB-AES-Z7EV-7Z020-G-V3c%20%281%29\\_0.pdf](http://zedboard.org/sites/default/files/product_briefs/5066-PB-AES-Z7EV-7Z020-G-V3c%20%281%29_0.pdf). – abgerufen am 16.08.2018
- [7] BAUERNÖPPEL, Frank: *Vorlesung Rechnerorganisation - Mikroarchitektur*. Berlin : Hochschule für Technik und Wirtschaft Berlin, 2016
- [8] BÖHME, Carsten: *Entwurf und Implementierung einer ARMv4 konformen Mikroarchitektur in VHDL und Validierung auf einem FPGA*. Berlin : Technische Universität Berlin, 2010
- [9] BLUM, Richard ; BRESNAHAN, Christine: *Linux Command Line and Shell Scripting Bible*. John Wiley and Sons, Inc., 2015. – URL <https://books.google.de/books?id=ywUaBgAAQBAJ>. – abgerufen am 10.08.2018
- [10] DEMBOWSKI, Klaus: *Das Addison-Wesley Handbuch der Hardwareprogrammierung*. München : Addison-Wesley Verlag, 2006
- [11] DIGILENT INC.: *Technology Partners*. – URL <https://store.digilentinc.com/technology-partners/>. – abgerufen am 20.07.2018
- [12] FURBER, Steve: *ARM System Architecture*. Essex : Addison Wesley Longman, 1996
- [13] GOLZE, Ulrich: *VLSI-Entwurf eines RISC-Prozessors: Eine Einführung in das Design großer Chips und die Hardware-Beschreibungssprache VERILOG HDL*. Vieweg+Teubner Verlag, 2013 (Lehrbuch Informatik). – URL <https://books.google.de/books?id=3BrQBgAAQBAJ>. – abgerufen am 16.08.2018

- [14] HELLER, Holger: 6 Chiphersteller machen ARM zur führenden MCU-Architektur. In: *Elektronik Praxis* (2011). – URL <https://www.elektronikpraxis.vogel.de/6-chiphersteller-machen-arm-zur-fuehrenden-mcu-architektur-a-325687/>. – abgerufen am 16.08.2018
- [15] HENNESSY, John L. ; PATTERSON, David A.: *Computer Architecture - A Quantitative Approach*. San Francisco, California : Morgan Kaufmann, 2012
- [16] HENNESSY, John L. ; PATTERSON, David A.: *Rechnerorganisation und Rechnerentwurf - Die Hardware/Software-Schnittstelle*. Berlin/ Boston : Walter de Gruyter GmbH, 2016. – 5. Auflage
- [17] HOFFMANN, Dirk W.: *Grundlagen der Technischen Informatik*. München : Carl Hanser Verlag, 2016. – 5. aktualisierte Auflage
- [18] INMOS LIMITED: *Transputer Reference Manual*. Cambridge : Prentice Hall International Ltd, 1988
- [19] KUWAHARA, Toshinori: *FPGA-based Reconfigurable On-board Computing Systems for Space Applications*. Stuttgart : Universität Stuttgart, 2010. – URL <http://dx.doi.org/10.18419/opus-3830>. – abgerufen am 03.07.2018
- [20] MENGE, Matthias: *Moderne Prozessorarchitekturen - Prinzipien und ihre Realisierungen*. Berlin : Springer Verlag, 2005
- [21] PUSCHMANN, Peter: *Einführung Programmierbare Schaltkreise*. – Video, ist dieser Arbeit in elektronischer Form beigelegt mit freundlicher Genehmigung des Urhebers
- [22] REICHARDT, Jürgen ; SCHWARZ, Bernd: *VHDL-Synthese*. Berlin/Boston : Walter de Gruyter GmbH, 2015. – 7. Auflage
- [23] ROLOFF, Sascha: *Multicore-Architekturen*. 2009. – URL [www2.in.tum.de/hp/file?fid=311](http://www2.in.tum.de/hp/file?fid=311)
- [24] TIETZE, Ulrich ; SCHENK, Cristoph ; GAMM, Eberhard: *Halbleiter-Schaltungstechnik*. Berlin : Springer Verlag, 2016. – 15. Auflage
- [25] WALLS, Colin: *Multicore Basics: AMP and SMP*. 2014. – URL <https://www.embedded.com/design/mcus-processors-and-socs/4429496/Multicore-basics>. – abgerufen am 16.08.2018
- [26] WOLF, Jürgen: *C von A bis Z*. Rheinwerk Computing, 2009. – URL [http://openbook.rheinwerk-verlag.de/c\\_von\\_a\\_bis\\_z/](http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/). – 3. Auflage
- [27] XILINX INC.: *ISE Design Suite*. – URL <https://www.xilinx.com/products/design-tools/ise-design-suite.html>. – abgerufen am 31.07.2018

- [28] XILINX INC.: *Multiple Domain-Optimized Platforms Spartan-3 Generation*. – URL <https://www.xilinx.com/products/silicon-devices/fpga/spartan-3.html>. – abgerufen am 20.07.2018
- [29] XILINX INC.: *PicoBlaze 8-bit Microcontroller*. – URL <https://www.xilinx.com/products/intellectual-property/picoblaze.html>. – abgerufen am 02.08.2018
- [30] XILINX INC.: *Product Tables and Product Selection Guides*. – URL <https://www.xilinx.com/products/silicon-devices/fpga.html>. – abgerufen am 20.07.2018
- [31] XILINX INC.: *Spartan-3*. – URL <https://www.xilinx.com/support/documentation-navigation/silicon-devices/mature-products/spartan-3.html>. – abgerufen am 20.07.2018
- [32] XILINX INC.: *Spartan-3 Board (RETIRED)*. – URL <https://store.digilentinc.com/spartan-3-board-retired/>. – abgerufen am 20.07.2018
- [33] XILINX INC.: *Spartan-6 Highest I/O per Logic Cell*. – URL <https://www.xilinx.com/products/silicon-devices/fpga/spartan-6.html>. – abgerufen am 20.07.2018
- [34] XILINX INC.: *Using VHDL global signals (signals declared in package)*. – URL <https://forums.xilinx.com/t5/Synthesis/Using-VHDL-global-signals-signals-declared-in-a-package/td-p/495356>. – abgerufen am 24.07.2018
- [35] XILINX INC.: *LogiCORE Block Memory Generator - When running simulation or HDL Synthesis tools, an error occurs stating that WE, WEA, or WEB has an incorrect type of std\_logic\_vector(0 downto 0)*. 2006. – URL <https://www.xilinx.com/support/answers/24048.html>

# Anhang

## A ARM-Architektur

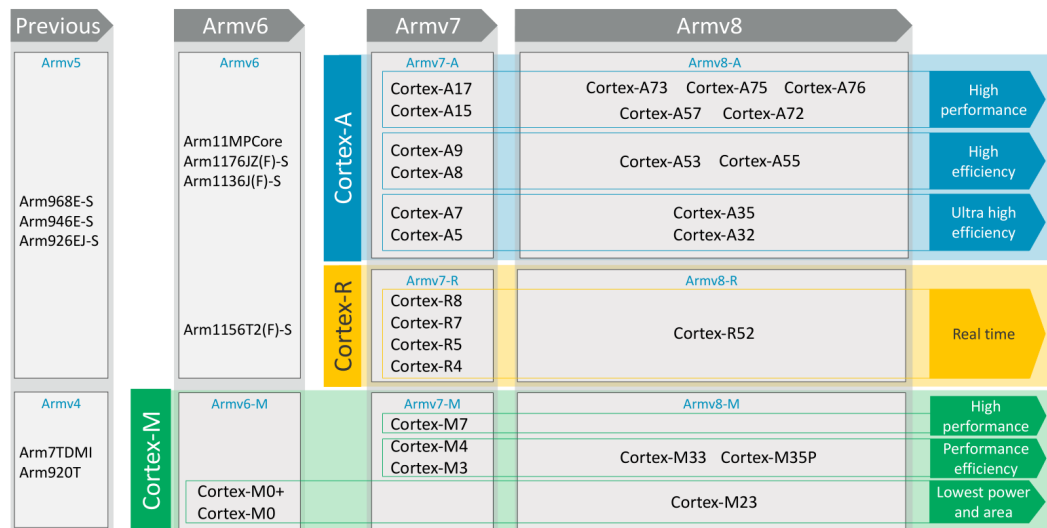


Abbildung 19: Übersicht über verschiedene ARM-Prozessoren zur Einordnung der Generation ARMv4 [4]

## B VHDL

```

46
47 entity ArmSystemController is
48   Port ( EXT_RST      : in STD_LOGIC;
49         SYS_RST      : out STD_LOGIC;
50         CTRL_DnRW    : out STD_LOGIC;
51         CTRL_DMAS    : out STD_LOGIC_VECTOR(1 downto 0);
52         CTRL_DA      : out STD_LOGIC_VECTOR(31 downto 0);
53         CTRL_DDIN    : in  STD_LOGIC_VECTOR(31 downto 0);
54         CTRL_DDOUT   : out STD_LOGIC_VECTOR(31 downto 0);
55         CTRL_DABORT  : in  STD_LOGIC;
56         CTRL_DEN     : out std_logic;
57 -- Adresseingang fuer die Anzeige der aktuellen Adresse (Bits 9:2) auf dem Instruktionsbus
58         CTRL_IA      : in std_logic_vector(9 downto 2);
59         CTRL_LDP     : in std_logic;
60         CTRL_STATUS_LED : OUT STD_LOGIC_VECTOR(7 downto 0);
61         CTRL_WAIT    : out std_logic;
62 --Zusaetzliche Inputs aus ausgelagertem Clk Generator
63         EXT_DCM_LOCKED : in std_logic;
64         INT_CLK       : in std_logic;
65         INT_INV_CLK  : in std_logic
66     );
67 end ArmSystemController;

```

Abbildung 20: Anpassung des Systemcontrollers zur Umsetzung mehrerer Kerne



## C Auslastung des Zynq FPGAs

### C.1 Genutzte Fläche

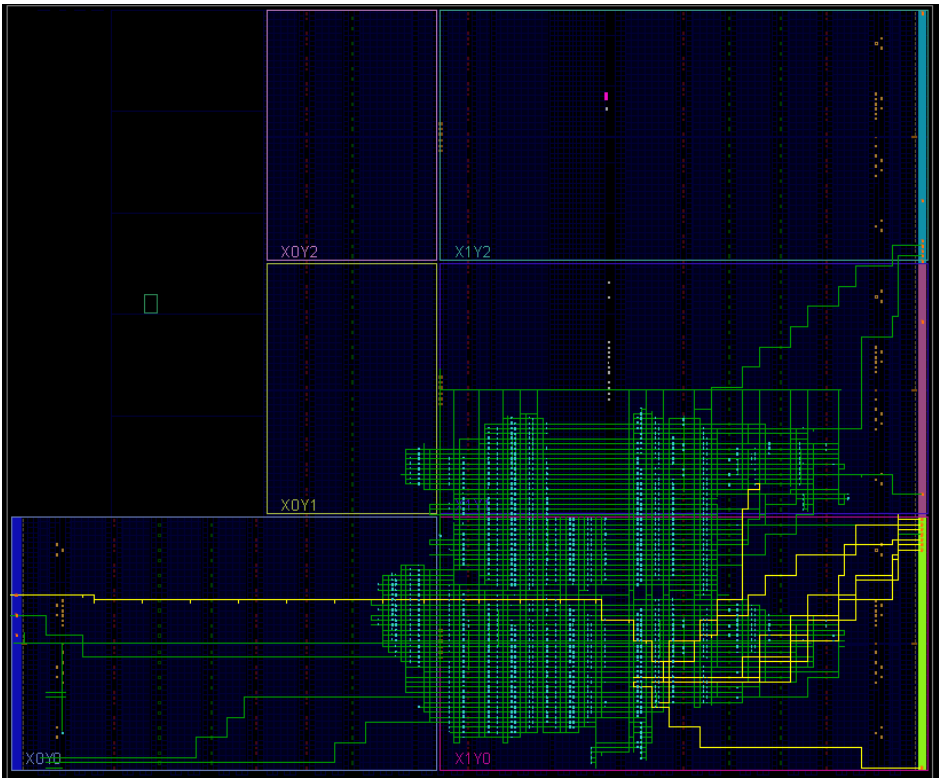


Abbildung 21: Flächenauslastung des Zynq-FPGAs bei zwei Prozessorkernen

### C.2 Genutzte Ressourcen

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	1,903	9,312	20%
Number of 4 input LUTs	5,850	9,312	62%
Number of occupied Slices	3,180	4,656	68%
Number of Slices containing only related logic	3,180	3,180	100%
Number of Slices containing unrelated logic	0	3,180	0%
Total Number of 4 input LUTs	6,004	9,312	64%
Number used as logic	5,814		
Number used as a route-thru	154		
Number used as 16x1 RAMs	35		
Number used as Shift registers	1		
Number of bonded IOBs	13	232	5%
Number of RAMB16s	8	20	40%
Number of BUFGMUXs	2	24	8%
Number of DCMs	1	4	25%
Number of MULT18X18SIOs	3	20	15%
Average Fanout of Non-Clock Nets	4.36		

Abbildung 22: Ressourcennutzung des Spartan 3E

## C. AUSLASTUNG DES ZYNQ FPGAS

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	1,907	11,776	16%
Number of 4 input LUTs	5,892	11,776	50%
Number of occupied Slices	3,205	5,888	54%
Number of Slices containing only related logic	3,205	3,205	100%
Number of Slices containing unrelated logic	0	3,205	0%
Total Number of 4 input LUTs	6,046	11,776	51%
Number used as logic	5,856		
Number used as a route-thru	154		
Number used as 16x1 RAMs	35		
Number used as Shift registers	1		
Number of bonded IOBs	13	372	3%
Number of BUFGMUXs	2	24	8%
Number of DCMs	1	8	12%
Number of MULT18X18SIOs	3	20	15%
Number of RAMB16BWEs	8	20	40%
Average Fanout of Non-Clock Nets	4.37		

Abbildung 23: Ressourcennutzung des Spartan 3AN

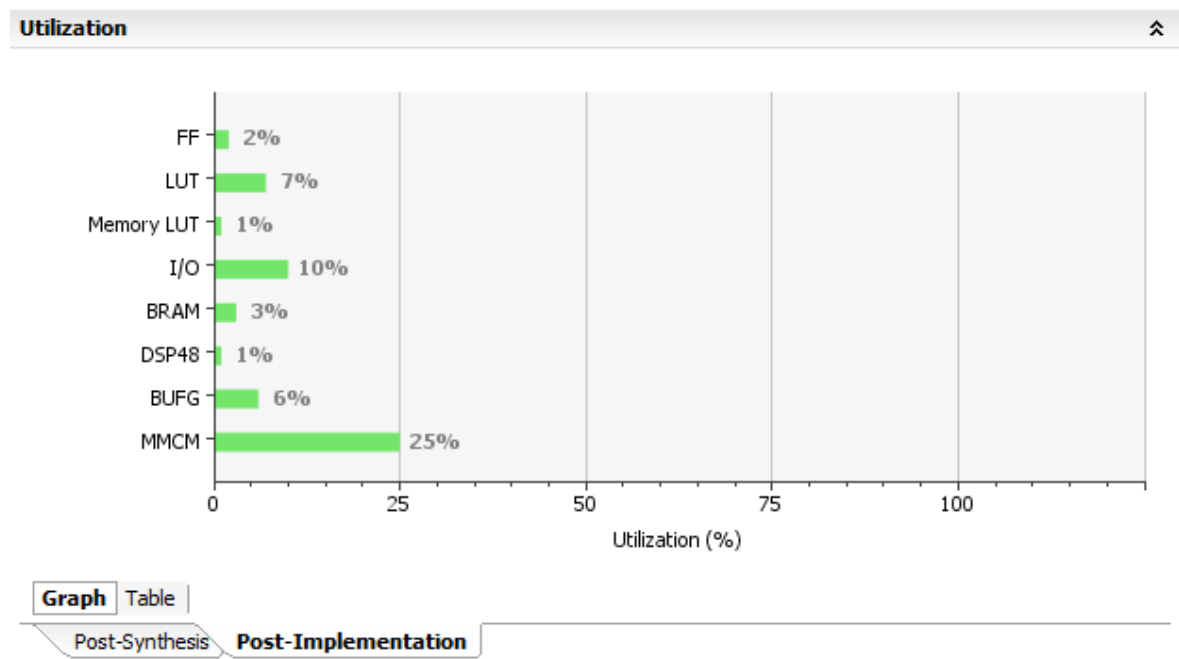


Abbildung 24: Ressourcennutzung des Zynq-FPGAs bei einem Prozessorkern

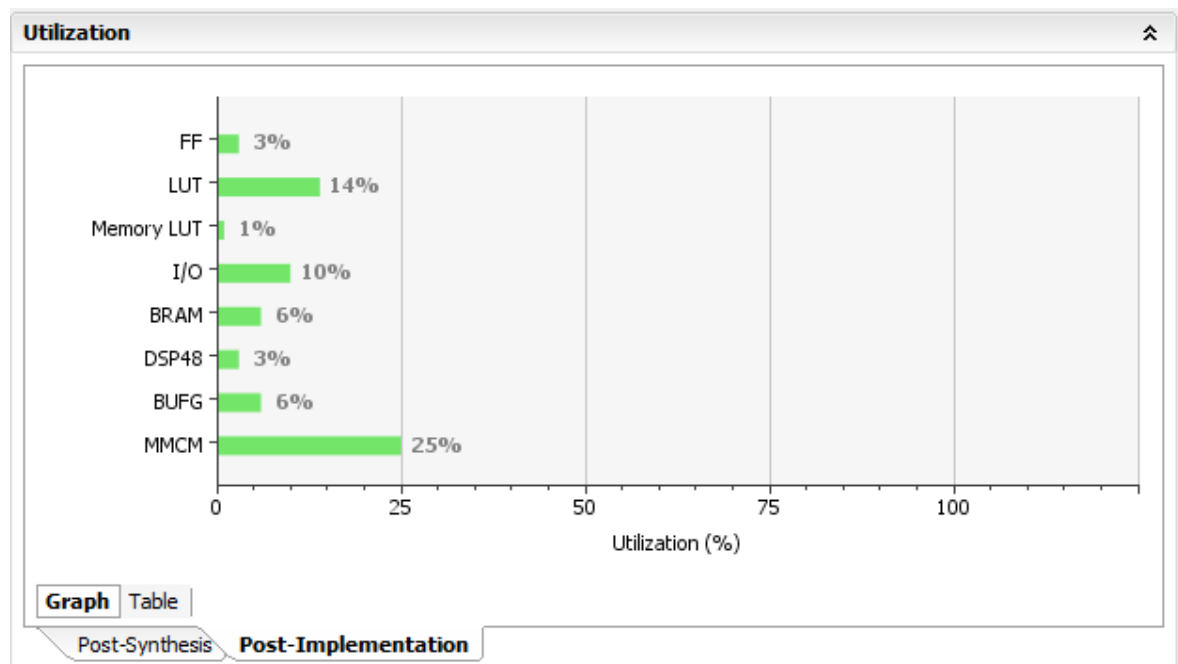


Abbildung 25: Ressourcennutzung des Zynq-FPGAs bei zwei Prozessorkernen