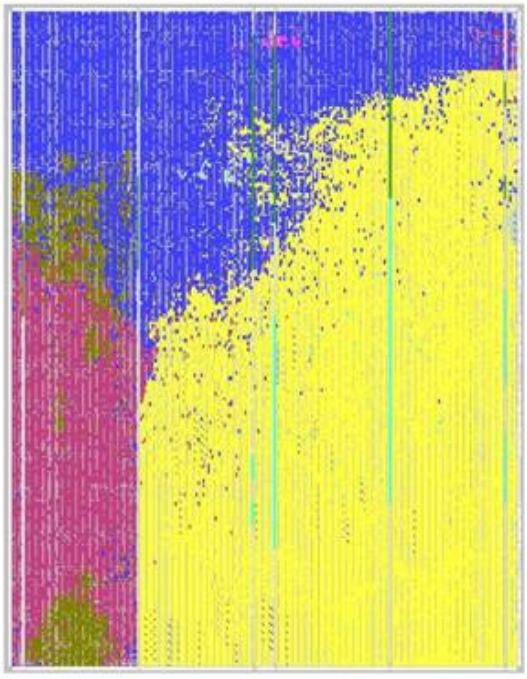
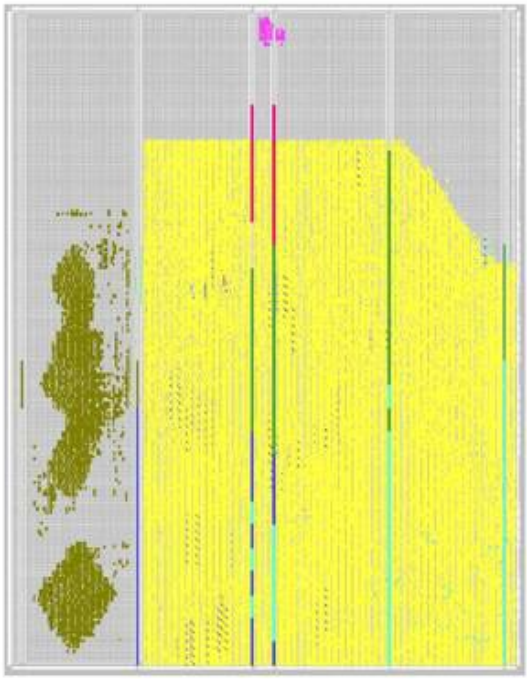
Stephen Schmitt

Integrierte Simulation und Emulation eingebetteter Hardware/Software-Systeme





Cuvillier Verlag Göttingen

Integrierte Simulation und Emulation eingebetteter Hardware/Software-Systeme

Dissertation
der Fakultät für Informations- und Kognitionswissenschaften
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Dipl.-Inform. Stephen Schmitt
aus Ruit auf den Fildern

Tübingen
2005

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.
1. Aufl. - Göttingen : Cuvillier, 2005
Zugl.: Tübingen, Univ., Diss., 2005
ISBN 3-86537-511-1

Tag der mündlichen Prüfung:	08.06.2005
Dekan	Prof. Dr. Michael Diehl
1. Berichterstatter:	Prof. Dr. Wolfgang Rosenstiel
2. Berichterstatter:	Prof. Dr.-Ing. Dr.-Ing. E.h. Wolfgang Straßer

Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

1. Auflage, 2005
Gedruckt auf säurefreiem Papier

ISBN 3-86537-511-1

Für meine Mutter und meinen Vater

Danksagung

Diese Arbeit entstand neben meiner Tätigkeit als Wissenschaftlicher Mitarbeiter im Arbeitsbereich Technische Informatik des Wilhelm-Schickard-Instituts für Informatik der Universität Tübingen. Mein besonderer Dank gilt meinem Doktorvater Herrn Professor Dr. Wolfgang Rosenstiel für die Betreuung der Arbeit und die exzellente Arbeitsumgebung, die er mir zur Durchführung meiner Arbeit zur Verfügung gestellt hat. Ein weiterer Dank gilt Herrn Professor Dr. Dr. E.h. Wolfgang Straßer für die Übernahme des Korreferats und seiner sorgfältigen Begutachtung der Arbeit. Darüber hinaus möchte ich mich bei meinen Projektpartnern von Infineon Technologies AG für die gute Zusammenarbeit bedanken.

Weiterhin bedanke ich mich bei Herrn Dr. Joachim Gerlach, Herrn Dipl.-Inform. Axel Braun und Herrn Dr. Klaus Beschorner für die fruchtbaren Diskussionen während meiner Arbeit sowie bei meiner Studentin Frau Katharina Weinberger und meinen Studenten Herrn Paulius Duplys und Herrn Johannes Brügmann für die Zusammenarbeit. Ein ganz besonderer Dank gilt meinen beiden Kollegen Herrn Dipl.-Phys. Werner Dreher und Herrn Dr. Walter Lange für die vielen hilfreichen Diskussionen und Anregungen für meine Arbeit. Insbesondere danke ich Herrn Dr. Walter Lange für die Korrekturhinweise zu meiner Ausarbeitung.

Bei meinen Kolleginnen und Kollegen bedanke ich mich für die freundliche Atmosphäre am Arbeitsbereich und die angenehmen Gespräche auch abseits der Arbeit. Insbesondere gilt mein Dank den Administratorkollegen Herrn Dr. Marcus Ritt, Herrn Dipl.-Inform. Carsten Schulz-Key, Herrn Dipl.-Inform. Gerald Heim und Herrn Dipl.-Inform. Thomas Schweizer für die gute Zusammenarbeit und die stets einwandfreie Rechnerinfrastruktur.

Mein herzlichster Dank aber gilt meiner Mutter und meinem Vater, die mich auf meinem Lebensweg immer unterstützt und die mir diese Arbeit durch eine erstklassige schulische und universitäre Ausbildung erst ermöglicht haben. Bei meinen Eltern habe ich zu jeder Zeit den Rückhalt, den ich zur Erlangung meiner Ziele benötige. Ohne die vielen kleinen und großen Hilfen meiner Eltern wäre ich nie so weit gekommen und ohne sie wäre ich nie das geworden, was ich heute bin.

Nürtingen, im Juni 2005 Stephen Schmitt

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programmverzeichnis

1 Einleitung

Nach dem Erfolg der Informationstechnologie in der Büro- und Arbeitswelt werden eingebettete Systeme als das wichtigste Anwendungsgebiet der Informatik in den kommenden Jahren betrachtet. In diesem Zusammenhang wird auch gerne von der *Nach-PC-Ära* gesprochen. Beispielsweise ist heute jede moderne Küche mit mehr Rechenkapazität ausgestattet als der Steuerrechner der Apollo Mondlandefähre im Jahre 1969.

Bedingt durch den stetigen Technologiefortschritt, der es ermöglicht, durch kleiner werdende Strukturgrößen immer mehr Funktionalität auf einem Chip unterzubringen, steigt die Anwendungsvielfalt und Komplexität der Systeme bei gleichzeitig sinkenden Entwurfszeiten und -kosten. Daraus ergibt sich ein komplexer Systementwurf der die Integration vieler unterschiedlicher Funktionen in ein Gesamtsystem auf einem Chip *(System-on-a-Chip (SoC))* unterstützen muss.

Insbesondere spielt die Kombination eines oder mehrerer Mikrokontroller mit zusätzlichen Peripheriekomponenten eine zentrale Rolle. Aufgrund der Komplexität zukünftiger eingebetteter Systeme wird die Wiederverwendung möglichst vieler Komponenten an Bedeutung gewinnen. Diese werden in Form von *Intellectual Property (IP)* als Hardware und Software zu einem Gesamtsystem kombiniert. Bei der Vernetzung eingebetteter Systeme können beispielsweise die Schnittstellen in Hardware realisiert und das System anwendungsspezifisch damit erweitert werden. Im Automobilbereich erlangt z.B. das Echtzeitbussystem FlexRay [32] zunehmend an Bedeutung, mit dem es zukünftig möglich sein wird, sicherheitskritische Komponenten miteinander zu vernetzen und die Sicherheit zu erhöhen.

Daneben kommt der Software in zukünftigen eingebetteten Systemen eine wichtige Rolle zu. In Abbildung 1.1 ist die Entwicklung des Anteils der Software an den Ausgaben für Hardware und Software in der Automobilindustrie für die nächsten Jahre dargestellt. In diesem Zusammenhang kann eine Parallele zum Moore'schen Gesetz formuliert werden, die besagt, dass sich für viele Produkte aus dem Verbraucherbereich die Größe des Codes alle zwei Jahre verdoppelt [100].

Insbesondere ist aus dem Bild ersichtlich, dass sich die Software zukünftiger Systeme modularer zusammensetzt. Wurde früher die Software noch als Ganzes gesehen, wird in Zukunft auf einem Betriebssystem (RTOS) *Basissoftware*, wie etwa Treiber für Hardware-Komponenten oder Middleware für die Vernetzung, zusammen mit unterschiedlichen Anwendungen implementiert werden, die unter Umstän-

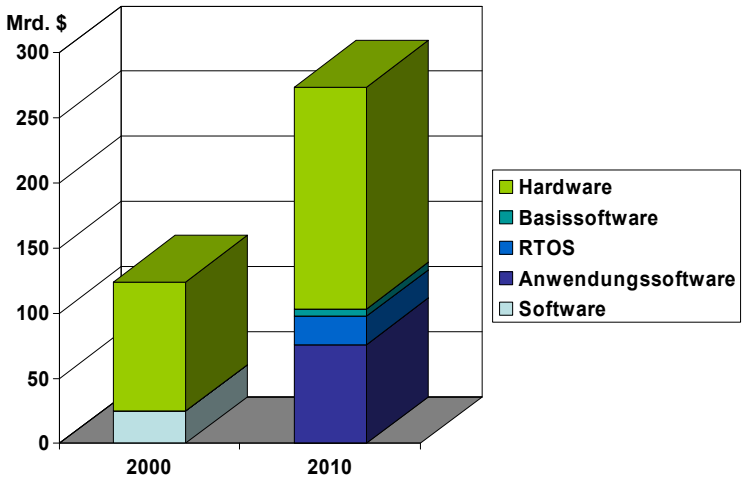


Abbildung 1.1: Entwicklung des Anteils der Software an den Ausgaben für Hardware und Software in der Automobilindustrie (Quelle: Mercer, 2003).

den auch dynamisch nachgeladen werden können.

Durch die Verwendung von Software lassen sich die Produkte mit unterschiedlichen Funktionen ausstatten, um sie von Konkurrenzprodukten abzuheben. Heute sind neue Produkte wie Mobiltelefone oder Digitalkameras für den Kunden nur noch dann interessant, wenn sie wesentliche Neuerungen gegenüber bereits bekannten Geräten bieten. Der Zeitpunkt, zu dem ein neues Produkt auf den Markt kommt, hat einen entscheidenden Einfluss auf den Erfolg. So verzeichnete Siemens 2004 einen Verlust von 152 Mio Euro unter anderem aufgrund der Tatsache, dass man die Entwicklung in Bezug auf die Integration von Digitalkameras in Mobiltelefonen nicht vorausgesehen hat [43].

Darüber hinaus spielt die Verifikation und Qualitätssicherung bei der zunehmenden Komplexität eingebetteter Systeme ein wichtige Rolle. Speziell im Bereich sicherheitskritischer Anwendungen sind ausgiebige Tests unter möglichst realistischen Umweltbedingungen unumgänglich. Treten bei solchen Systemen nach der Auslieferung Probleme auf, so sind die Firmen zu Rückrufaktionen gezwungen, die nicht nur sehr teuer werden, sondern auch ihrem Image schaden. Gleiches gilt auch für Produkte aus dem Verbraucherbereich wie beispielsweise Mobiltelefone, Digitalkameras oder MP3-Spieler.

Aus diesem Grund ist das *Prototyping* ein wichtiger Bestandteil des Entwurfsprozesses eingebetteter Systeme. Beim Prototyping wird versucht, die Funktionalität eines Systems anhand eines physikalischen Prototyps zu evaluieren. Oft lassen sich komplexe Effekte, die z.B. durch Umwelteinflüsse entstehen, nur anhand eines real existierenden physikalischen Modells des zu entwickelnden Systems genauer un-

tersuchen und bewerten. Deshalb wird das Prototyping unter anderem gerne in den Bereichen der Automobilentwicklung und Kommunikationstechnik eingesetzt.

Beim Prototyping moderner eingebetteter Systeme müssen unterschiedliche Entwicklungsmethoden und -werkzeuge kombiniert werden. Betrachtet man die Hardware-Komponenten eines Systems, so ist die Integration der verschiedenen funktionalen Blöcke sehr aufwändig. Beim Entwurf eingebetteter Software sind gegenüber der konventionellen Software-Entwicklung andere Anforderungen wichtig. Hier spielen Echtzeiteigenschaften, Speicherplatz, Energieverbrauch und Sicherheitsaspekte eine wichtige Rolle. Nicht zuletzt müssen die Schnittstellen zwischen den Hardware- und Software-Komponenten des eingebetteten Systems eine verlässliche und reibungslose Zusammenarbeit zwischen den einzelnen Komponenten garantieren.

1.1 Motivation und Problemstellung

Der bisherige Entwurfsprozess für eingebettete Systeme und speziell der SoCs ist stark auf die Verifikation der Hardware ausgerichtet. Die Integration und der Test neu entwickelter Hardware-Komponenten wird heute vor allem mit hardware-spezifischen Entwicklungsplattformen und Verifikationsmethoden durchgeführt. Wie im vorangegangenen Abschnitt allerdings bereits angedeutet wurde, steigt der Anteil der Software in eingebetteten Systemen rasant an, weshalb die Notwendigkeit besteht, neue, auch software-zentrierte Entwicklungs- und Verifikationsmethoden für eingebettete Hardware/Software-Systeme zu entwickeln.

Diese Arbeit entstand im Rahmen des Forschungsprojektes „Spezifikation und Algorithmus/Architektur-Codesign für hochkomplexe Anwendungen der Automobil- und Kommunikationstechnik (SpeAC)“ [94] in Zusammenarbeit mit der Firma Infineon Technologies AG. Ziel des Forschungsprojektes ist eine Produktivitätssteigerung des Entwurfs eingebetteter Hardware/Software-Systeme durch die Entwicklung einer Algorithmus/Architektur-Codesign-Methodik, die auf hohem Abstraktionsniveau beginnend, einen durchgängigen Entwurfsablauf ermöglicht und auf der Basis vordefinierter Hardware/Software-Plattformen angelegt ist.

Innerhalb dieser Entwurfsmethodik konzentrieren sich die in dieser Arbeit vorgestellten Konzepte auf die Systemintegration von Hardware und Software, das Hardware/Software-Co-Debugging und Rapid Prototyping komplexer Systementwürfe. Die Ziele dieser Arbeit lassen sich daher auch in zwei Teilbereiche untergliedern:

1. Entwicklung einer Entwurfsmethodik für eingebettete Software für SoC-Entwürfe auf der Basis eines schnellen und architekturgenauen Modells der SoC-Hardware.

In heutigen Systemen spielt die Verkürzung der Entwurfszeit eine entscheidende Rolle. Hier liegt die Software-Entwicklung auf dem kritischen Pfad, da für deren Entwicklung ein Modell des Mikrokontrollers verfügbar sein muss. Da der Umfang der Programme stetig ansteigt, muss deren Ausführungszeit signifikant verkürzt werden. Dazu ist die Integration von Software-Entwicklungswerkzeugen und insbesondere die Anbindung einer Software-Debuggingumgebung an das beschleunigte Hardware-Modell zwingend notwendig.

Für die Evaluierung von Software-Algorithmen und Performanzabschätzungen des Zielsystems ist darüber hinaus ein möglichst architekturgenaues Modell der Hardware notwendig. An dieser Stelle weisen heutige Entwurfsmethoden Defizite auf, da die Ausführung der Software auf detaillierten Hardware-Modellen sehr viel Zeit in Anspruch nimmt. Da an der Software-Entwicklung darüber hinaus mehrere Entwickler arbeiten können, sollte die Entwicklungsplattform keine hohen Kosten verursachen, damit eine parallele Entwicklung unterschiedlicher Teile der Software mit mehreren Systemen möglich ist.

Teile dieser Software müssen dabei für neue Hardware-Komponenten entwickelt werden, die die Funktionalität des SoC anwendungsspezifisch erweitern. Ein zweites Ziel dieser Arbeit ist deshalb:

2. Bereitstellung einer Entwurfsmethodik, für die parallele Entwicklung und den Test komplexer Hardware-Komponenten und der dazugehörigen Software[1].

Durch die parallele Entwicklung von Hardware und Software lässt sich die Entwurfszeit drastisch reduzieren. Insbesondere entfällt durch das Testen der Hardware-Komponenten durch die Software auf dem Prozessor-Modell die Erstellung einer Hardware-Testumgebung. Die frühe Evaluierung von Hardware und dazugehöriger Software ist wichtig, da in dieser Entwicklungsphase Fehler in der Hardware-Komponente und der Hardware/Software-Schnittstelle noch einfach behoben werden können. Nach dem Tape-out des Chips ist dies nur durch eine neue Implementierung möglich, was Kosten in Millionenhöhe verursachen kann.

Die Entwurfsmethodik muss detaillierte Einblicke in die internen Abläufe der Hardware-Komponenten bis auf Signalebene zulassen. Ferner muss das Debugging der Hardware-Komponenten ermöglicht und die Turn-around-Zeiten für das Wiederaufsetzen der Entwicklungsumgebung nach Beseitigung eines Entwurfsfehlers verkürzt werden.

Das in dieser Arbeit vorgestellte Konzept ermöglicht es einem Hardware-IP-Anbieter darüber hinaus, Details seiner IP vor dem Kunden zu verstecken und so den

[1] Im Folgenden wird die Software, die für die Kontrolle einer Hardware-Komponente verwendete wird, als *Treiber* oder hardware-abhängige Software bezeichnet.

Quelltext seiner IP zu schützen. Dies stellt ein nicht unerhebliches Kriterium für eine Entwicklungsplattform für SoC-Entwürfe dar, da die Kosten für die IP selbst im siebenstelligen Bereich liegen können.

Insgesamt wird in dieser Arbeit ein *ganzheitlicher* Ansatz entwickelt und umgesetzt, der eine architekturgenaue und detaillierte Sicht auf ein zu entwickelndes System zulässt und dennoch die Ausführungszeiten für die zu entwickelnde Software auf dem Hardware-Modell beschleunigt. Diese Beschleunigung der Hardware-Modelle, die in einer Hardware-Beschreibungssprache wie z.B. VHDL [7] vorliegen, ist durch Emulation möglich. Heutige Emulations-Systeme sind allerdings sehr teuer und stellen dadurch eine Hürde für die Evaluierung neuer Technologien durch einen IP-Kunden dar.

Die **Idee** dieser Arbeit besteht deshalb darin, für den Entwurf eingebetteter Hardware/Software-Systeme eine *kostengünstige* rekonfigurierbare Entwicklungsumgebung zu verwenden, die auf einer FPGA-basierten Emulationsplattform basiert. Die Entwicklungsumgebung bietet dem Entwickler ein *Soft*-Evaluierungsboard, auf dem die SoC-Hardware integriert und durch die Kopplung mit einem Software-Debugger darüber hinaus auch die Entwicklung hardware-naher Software ermöglicht wird. Dadurch lässt sich eine signifikante Beschleunigung des Hardware-Modells und damit der Anwendungsentwicklung insgesamt bei dennoch moderaten Kosten für die Entwicklungsumgebung erreichen.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit lässt sich in vier Teile untergliedern. Neben den Grundlagen und einer Diskussion des Stands der Technik wird ein neues Konzept zur Entwicklung eingebetteter Hardware/Software-Systeme vorgestellt. Dieses Konzept wird anhand eines ausführlichen Beispiels aus der Praxis näher erläutert und die Relevanz der entwickelten Methoden wird anhand von Ergebnissen gezeigt.

In Kapitel 2 werden zunächst Grundlagen behandelt, die für das Verständnis der in dieser Arbeit angesprochenen Begriffe und Methoden notwendig sind. Kapitel 3 enthält eine Zusammenfassung des Stands der Technik bei der Entwicklung von Hardware und Software von SoCs. Abschnitt 3.1 behandelt zunächst den Entwurf eingebetteter Software auf Hardware-Modellen bevor in Abschnitt 3.2 bisherige Verfahren für die parallele Entwicklung von Hardware-Komponenten und deren Software für SoCs vorgestellt werden. In Kapitel 4 findet sich eine Bewertung des Stands der Technik. Es werden Defizite aktueller Methoden identifiziert und in Abschnitt 4.3 die Ziele der vorliegenden Arbeit definiert.

Im zweiten Teil dieser Arbeit wird in Kapitel 5 ein neues Konzept des Entwurfs eingebetteter Hardware/Software-Systeme für SoCs vorgestellt. Dort wird zunächst

in Abschnitt 5.1 ein Überblick über die Grundideen des neuen Konzepts gegeben. Abschnitt 5.2 enthält eine Methodik für die Beschleunigung der Software-Entwicklung für SoCs auf der Basis eines emulationsbasierten Mikrokontroller-IP-Kerns. Dieses Konzept wird in Abschnitt 5.3 zu einer Methodik zur parallelen Entwicklung neuer Hardware-Komponenten und deren hardware-naher Software ausgebaut.

Kapitel 6 zeigt mit Hilfe eines praxisrelevanten Beispiels, wie die in dieser Arbeit vorgestellten neuen Konzepte umgesetzt werden können. Es wird anhand des TriCore1®-Mikrokontrollers gezeigt, wie dieser auf einem FPGA-basierten Prototyping-System integriert und als SoC-Plattform für die Entwicklung neuer Hardware-Komponenten und ihrer dazugehörigen hardware-nahen Software und Anwendungen verwendet werden kann.

In Kapitel 7 werden Ergebnisse präsentiert, die mit dem vorgestellten Konzept erzielt werden können. Diese beziehen sich zum Einen auf Laufzeitmessungen, die mit einem architekturgenauen Modell einer SoC-Hardware gemacht wurden und zum Anderen auf die Darstellung der Entwicklung und Verifikation von Hardware-Komponenten und deren Software. Kapitel 8 enhält eine Zusammenfassung der im Rahmen dieser Arbeit entwickelten Konzepte und Methoden und der daraus gewonnenen Erkenntnisse.

2 Grundlagen

In diesem Kapitel werden die grundlegende Themen behandelt, die für das Verständnis dieser Arbeit wichtig sind. Dazu wird zunächst in Abschnitt 2.1 ein Überblick über Entwurfsmethoden für eingebettete Systeme gegeben. Danach werden in zwei weiteren Abschnitten die Entwicklungsdomänen Hardware und Software genauer betrachtet. Abschnitt 2.2 behandelt dabei die Entwicklung der Hardware und Abschnitt 2.3 die Entwicklung der Software eingebetteter Hardware/Software-Systeme. Im letzten Abschnitt dieses Kapitels wird dann detailliert auf die Emulation eingebetteter Systeme mit rekonfigurierbaren Hardware-Bausteinen eingegangen, da diese im Verlauf dieser Arbeit eine zentrale Rolle spielen werden.

2.1 Entwurfsmethoden für eingebettete Systeme

In diesem Abschnitt werden Entwurfsmethoden für die Entwicklung eingebetteter Systeme näher vorgestellt. Da eingebettete Systeme aus Hardware- und Software-Komponenten bestehen, müssen sowohl Entwicklungsmethoden für Hardware als auch Software betrachtet werden. Entwurfsmethoden sind wichtig, um die Entwicklung fehlerfreier Systeme zu gewährleisten. Dabei wird versucht, den Entwurfsprozess in Phasen aufzuteilen, deren Aufgaben genau spezifiziert sind. Darüber hinaus finden an den Übergängen zwischen den einzelnen Entwicklungsphasen Reviews statt.

2.1.1 Allgemeine Entwurfsmethoden für Software

In der Literatur wird zwischen *Top-down*- und *Bottom-up*-Entwurfsmethoden unterschieden [10]. Beim Top-down-Ansatz wird mit einer Systemspezifikation begonnen und diese im Rahmen des Entwurfs immer weiter verfeinert. Man spricht auch von einem Vorgehen vom Abstrakten zum Konkreten. Beim Bottom-up-Ansatz wird dagegen vom Konkreten zum Abstrakten vorgegangen.

Ein bekannter Vertreter des Top-down-Entwurfs ist das Wasserfallmodell, welches erstmals von Boehm [13] eingeführt wurde. Das Wasserfallmodell definiert bestimmte, aufeinanderfolgende Phasen, die beim Systementwurf durchlaufen werden. Am Ende einer jeden Entwurfsphase stehen Meilensteine, die in Form von Dokumenten

die Ergebnisse festhalten. Anhand der Meilensteine kann eine Terminplanung durchgeführt und eine Koordination unterschiedlicher Entwickler vorgenommen werden. Eines der Probleme des Wasserfallmodells ist, dass aufgrund der linearen Vorgehensweise auf Änderungen der Spezifikation nicht reagiert werden kann. Anforderungen können in der Regel aber nicht im Voraus ermittelt werden und ändern sich zudem während der Entwicklung.

Beim Wasserfallmodell spielt die Verifikation und Validation noch eine untergeordnete Rolle. Diese Entwicklungsphasen wurden beim V-Modell [17] explizit eingeführt. Das V-Modell war ursprünglich für die Entwicklung eingebetteter militärischer Anlagen vorgesehen. Beim V-Modell stehen den einzelnen Entwicklungsschritten explizite Testschritte gegenüber.

Bottom-up-Methoden gewinnen vor allem durch die heute übliche Wiederverwendung von Systemteilen an Bedeutung. Sie werden deshalb unter anderem auch in der objektorientierten Analyse von Software-Systemen angewendet. Dort wird von Klassen und ihren Beziehungen (Bottom) auf Subsysteme (Top) abstrahiert.

Allen bisher beschriebenen Methoden ist gemein, dass sie nicht auf Änderungen in der Spezifikation reagieren können. Von Boehm wurde deshalb das Spiralmodell [14] eingeführt, welches die Entwicklung eines Produkts in mehrere Prototypenphasen unterteilt. Beim Spiralmodell werden die einzelnen Phasen der Entwicklung in einem zyklisch angeordneten Prozess mehrfach durchlaufen. Der Entwurf ist dabei in vier Schritte unterteilt, die als die vier Quadranten eines Koordinatensystems betrachtet werden können. Begonnen wird im nordwestlichen Quadranten mit der Spezifikation des Produktes und der Festlegung von Entwurfszielen. Im nordöstlichen Quadranten werden dann Lösungsvarianten evaluiert, an deren Ende jeweils ein Prototyp des Produktes steht. Im südöstlichen Quadranten findet die Implementierung und Validierung des Produktes statt. Am Ende des Zyklus wird dann mit der Planung des nächsten Entwicklungsschrittes fortgefahren (südwestlicher Quadrant).

Die Vorteile des Spiralmodells liegen darin, dass dem Lernprozess des Entwicklers während der Produktentwicklung Rechnung getragen wird und das Produkt anhand von Prototypen in einer frühen Phase des Entwurfs überprüft werden kann. Die Kosten für die Produktentwicklung entsprechen der Fläche der Spirale. Nachteile des Spiralmodells liegen zum Einen in dem hohen Aufwand für das Projektmanagement, da oft neue Entscheidungen über den weiteren Prozessablauf getroffen werden müssen. Zum Anderen ist es für kleinere und mittlere Projekte wenig geeignet [10].

2.1.2 Entwurfsmethoden für Hardware/Software-Systeme

Methoden für den Entwurf eingebetteter Systeme unterscheiden sich von den oben beschriebenen Techniken dadurch, dass neben der Entwicklung der Software noch

die Entwicklung der Hardware durchgeführt werden muss. Dadurch ergeben sich mehr Freiheitsgrade für den Entwickler, da dieser entscheiden muss, ob eine Funktion in Hardware oder Software realisiert werden soll. Abbildung 2.1 zeigt eine schematische Darstellung für den Hardware/Software-Entwurf.

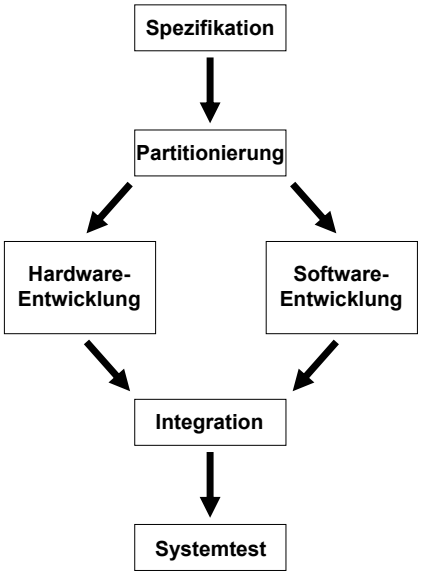


Abbildung 2.1: Entwicklung eingebetteter Hardware/Software-Systeme.

Beim Entwurf eingebetteter Hardware/Software-Systeme wird, ausgehend von einer Systemspezifikation, der Entwurf zunächst partitioniert. Dabei wird entschieden, welche Funktionen in Hardware und welche Funktionen in Software realisiert werden sollen. Hier spielen unter anderem Faktoren wie Chipfläche, Leistungsverbrauch, Taktfrequenz und Flexibilität eine wichtige Rolle. Die Entwicklung der Hardware und Software erfolgt dann idealerweise parallel, so dass sich die Entwicklungszeit des Gesamtsystems reduziert. Da für die Entwicklung der Software ein Modell der Hardware benötigt wird, ist diese parallele Entwicklung nur schwer zu erreichen. Ein weiteres Problem stellt die Integration der Hardware und Software und der Systemtest dar. Werden erst an dieser Stelle Probleme im System oder Fehler in der Spezifikation entdeckt, so muss mit einem erneuten Entwurf fortgefahren werden.

Aufgrund der zunehmenden Komplexität eingebetteter Systeme und der damit einhergehenden Vielzahl der Funktionen ist darüber hinaus eine komplette Neuentwicklung eines eingebetteten Systems nicht möglich. Aus diesem Grund wird von Sangiovanni-Vincentelli et al. [78, 55] eine plattformbasierte Entwurfsmethodik vorgeschlagen.

2.1.3 Plattformbasierter Entwurf

Abbildung 2.2 zeigt die Entwicklung der Entwurfsmethoden für die Hardware eingebetteter Systeme. Wurde früher die Hardware eines eingebetteten Systems komplett neu entwickelt, hat sich dies heute grundlegend gewandelt. Aufgrund des Kostendrucks und der kürzer werdenden Entwicklungszeiten wird die Hardware heutiger SoCs aus einigen wenigen Hardware-IP-Komponenten kombiniert mit einer speziell auf die Anwendung zugeschnittenen Logik entwickelt (blockbasierter Entwurf). Eine IP-Komponente ist dabei eine funktionale Einheit, die als eigenständige Subkomponente in einem kompletten SoC verwendet werden kann.

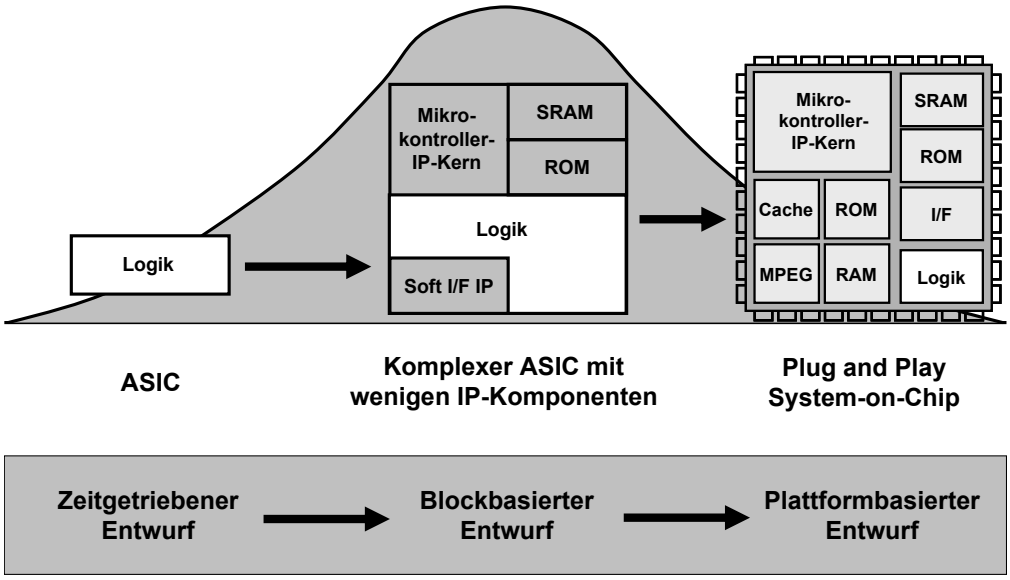


Abbildung 2.2: Entwurfsmethoden integrierter Schaltungen und SoCs [22].

Wie aus Abbildung 2.2 ersichtlich ist, können heute z.B. ganze Speicherstrukturen auf dem Chip untergebracht werden. Dabei kommen unterschiedliche Speicherbausteine zum Einsatz. ROM-Speicher (*Read only memory*) können von einem Mikrokontroller lediglich ausgelesen aber nicht geschrieben werden. RAM-Speicher (*Random access memory*) können dagegen gelesen und geschrieben werden. Es gibt zwei Arten von RAM-Speichern. DRAM-Speicher (*Dynamic RAM*) sind in Zeilen und Spalten organisiert und verlieren ihren Speicherinhalt beim Auslesen sowie durch Leckströme. Sie müssen deshalb immer wieder neu beschrieben werden. SRAM-Bausteine (*Static RAM*) dagegen behalten ihren Speicherinhalt solange die Spannungsversorgung nicht unterbrochen wird. Synchrone DRAMs (SDRAMs) sowie synchrone SRAMs (SSRAMs) sind taktsynchrone Speicherbausteine.

Die IP-Komponenten für die Chipentwicklung können als *Hardmakros* oder *Soft-*

makros vorliegen. Ein Hardmakro ist eine IP-Komponente, die dem SoC-Entwickler als GDSII-Datei[1] vorliegt. Vom Hersteller der Komponente wird diese vollständig entwickelt, platziert und verdrahtet ausgeliefert. Im Gegensatz dazu wird ein Softmakro vom Hersteller in synthetisierbarem RTL-Code geliefert und muss vom SoC-Entwickler neu synthetisiert werden [54]. Softmakros können intern wiederum Makros enthalten, die vom IP-Integrator entweder durch eigene Komponenten gleicher Funktionalität ersetzt oder durch entsprechende IP-Bibliotheken eingebunden werden müssen. Die Hardware-Synthese für Softmarkos wird in Abschnitt 2.2.2 noch näher behandelt.

Durch die systematische Wiederverwendung auf Blockebene konnten die Entwurfskosten Ende der neunziger Jahre innerhalb von zwei Jahren um den Faktor vier gesenkt werden [93]. Aufgrund der hohen Integrationsdichte zukünftiger Halbleitertechnologien reicht aber dieser Grad der Wiederverwendung von Hardware-IP bei weitem nicht aus. Beim plattformbasierten Entwurf wird deshalb von einer ganzen Plattform für ein bestimmtes Anwendungsgebiet ausgegangen und der Entwickler entwirft nur noch einen sehr kleinen Teil der Logik neu, um die Hardware-Plattform seinem konkreten Anwendungsfall anzupassen.

Der plattformbasierte Entwurf unterscheidet sich von den oben beschriebenen Entwurfsverfahren dadurch, dass das Zielsystem nicht mehr in einem reinen Top-down- oder Bottom-up-Entwurfsprozess entwickelt wird, sondern bei der Umsetzung einer bestimmten Funktionalität eine weitgehend vorgegebene Plattform mit zu berücksichtigen ist. Diese Plattform richtet sich dabei maßgeblich nach einem bestimmten Anwendungsgebiet, wie etwa der Automobil- oder Kommunikationselektronik.

Neben der Entwicklung der Hardware wird dabei auch die Entwicklung der Software aus der Sichtweise einer Plattform, wie in Abbildung 2.3 dargestellt, betrachtet. Wie aus der Abbildung ersichtlich ist, stellt der plattformbasierte Entwurf einen *Meet-in-the-Middle-Ansatz* dar, bei dem die Funktion und die die Plattform repräsentierende Architektur eines Systems parallel spezifiziert und anschließend Funktionsblöcke auf Architekturkomponenten abgebildet werden. Die Methodik kann auf unterschiedlichen Abstraktionsebenen angewendet werden. So stellt die Systemplattform an der Schnittstelle zwischen Anwendung und Architektur eine Plattform mit einer Programmierschnittstelle (API) für den Entwickler dar, die ein abstraktes Modell der Hardware kapselt und auf der eine erste Entwicklung der Software stattfinden kann.

Während des Entwurfsprozesses ist es aber auch wichtig, eine Bewertung der Plattform hinsichtlich ihrer Eignung für die gewünschte Funktionalität durchzuführen und diese Plattform in einem iterativen Prozess zu verbessern. Hier spielen unter

[1] GDSII ist ein binäres Dateiformat, das als Austauschformat von Maskendaten zwischen SoC-Entwickler und Chipfabrik verwendet wird.

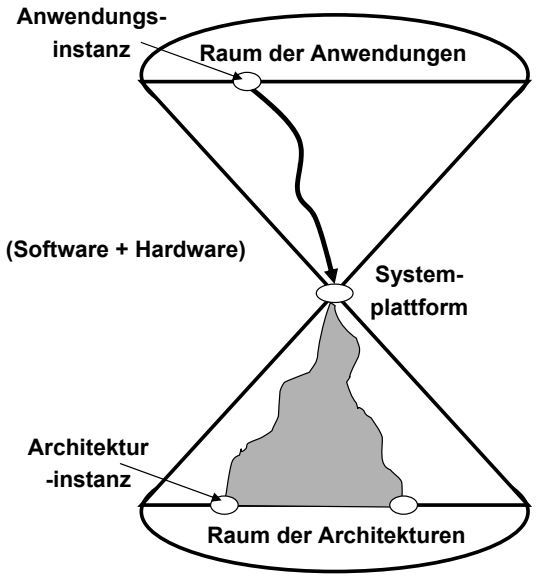


Abbildung 2.3: Hardware/Software-Entwicklung beim plattformbasierten Entwurf [78].

anderem auch zeitliche und physikalische Eigenschaften eine Rolle, die nur anhand eines realen Prototyps bewertet werden können.

2.1.4 Prototyping

Das Prototyping ist einer der wesentlichen Teile im Entwurfsprozess eingebetteter Systeme. Das Prinzip des Prototyping ist in Abbildung 2.4 dargestellt. Beim Systementwurf wird ausgehend von einer Systemspezifikation in der Analysephase ein Demonstrationsmodell (Prototyp) erstellt, welches möglichst alle Eigenschaften des geplanten Systems besitzt. Da in eingebetteten Systemen eine Interaktion mit der Umgebung des Systems notwendig ist, ist der Prototyp idealerweise ein physikalisches Modell, welches in ein Zielsystem eingebaut werden kann.

Durch das Prototyping können Annahmen und Entwurfsentscheidungen, die in der Spezifikationsphase getroffen wurden, überprüft werden. Ohne einen Prototyp können Fehler der Spezifikation während der Systemintegrationsphase zutage treten. Ist dies der Fall, dann ist ein aufwändiger Neuentwurf notwendig.

Es kann zwischen unterschiedlichen Arten des Prototyping unterschieden werden [25]. Beim *Rapid Prototyping* wird versucht, anhand eines schnell erstellten Modells möglichst alle Eigenschaften eines Systems nachzubilden. Das Ergebnis wird dann im weiteren Verlauf der Entwicklung nicht mehr weiterverwendet, sondern dient ausschließlich dazu, möglichst schnell einen vollständigen Überblick über

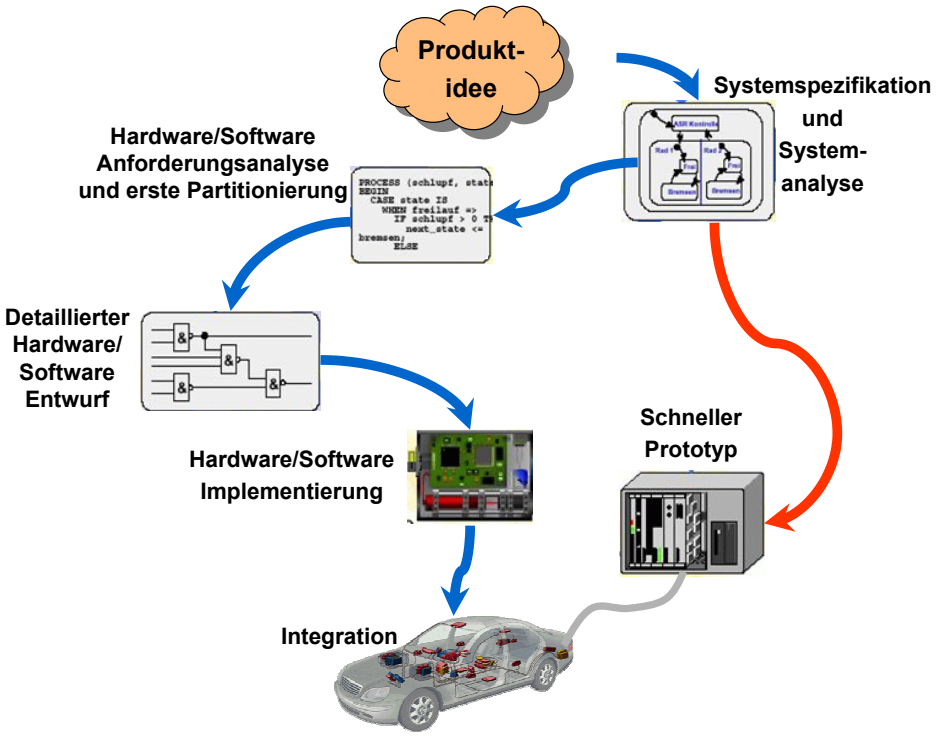


Abbildung 2.4: Systementwicklung mit Rapid Prototyping.

das zu entwickelnde System zu erhalten.

Beim *normalen* Prototyping wird der Prototyp dagegen systematisch zu einem Endprodukt weiterentwickelt. Dies ist beispielsweise bei dem in Abschnitt 2.1.1 bereits vorgestellten Spiralmodell der Fall. Da in dieser Arbeit eine Entwurfsmethodik für eingebettete Hardware/Software-Systeme vorgestellt wird und diese explizit die Entwicklung von Hardware-Komponenten mit einschließt, wird im Folgenden mit Prototyping immer die Erstellung eines Prototyps von Hardware und Software bezeichnet, der gemäß der Definition des normalen Prototypings weiterentwickelt wird. Da in SoC-Entwürfen die Hardware letzten Endes immer aus einem Chip besteht, muss beim Prototyping der Hardware nach geeigneten Modellen gesucht werden, die das Verhalten des Chips nachbilden. In diesem Zusammenhang wird die Hardware der Systeme oft emuliert. Die Emulation wird in Abschnitt 2.4 noch näher beschrieben.

Ein gutes Beispiel für das Rapid Prototyping im Bereich eingebetteter Systeme stellen Steuerungsanwendungen aus dem Automobilbereich dar [66]. Diese werden von unterschiedlichen Fahrern in unterschiedlichen Umgebungen und unter allen erdenklichen Umwelteinflüssen auf Herz und Nieren getestet. Dabei kommen oft große und komplexe Prototyping-Systeme zum Einsatz, die sich *beinahe* wie das Zielsys-

tem verhalten. Deren Nachteil ist, dass sie sehr kostenintensiv und unhandlich sind.

2.2 Hardware-Entwicklung für eingebettete Systeme

In diesem Abschnitt werden Methoden zur Hardware-Entwicklung vorgestellt. Dabei wird auf System- und Hardware-Beschreibungssprachen sowie die automatisierte Umsetzung dieser Beschreibungen in eine Hardware-Implementierung eingegangen. Im letzten Abschnitt wird die Verifikation der Hardware näher beleuchtet, da diese beim heutigen Entwurfsprozess eine wichtige Rolle spielt.

2.2.1 Hardware-Beschreibungssprachen

Aufgrund der Entwicklungen in der Chipfabrikation ist es in den letzten Jahren möglich geworden, immer komplexere Systeme auf einem Chip zu integrieren. Um der dadurch steigenden Entwurfskomplexität Herr zu werden, musste die Chipentwicklung automatisiert werden. Zu Beginn der 70er Jahre des 20ten Jahrhunderts treten vermehrt solche Werkzeuge auf [77]. Mit der Einführung von Hardware-Beschreibungssprachen (HDLs) Ende der 80er Jahre erhielt die Entwurfsautomatisierung nochmals einen Schub. Hier haben sich die beiden HDLs VHDL und Verilog durchgesetzt. Diese stellen bis heute die dominierenden Beschreibungssprachen für den Hardware-Entwurf dar.

In Gajski et al. [34] wird für die Entwicklung der Hardware ein Y-Diagramm eingeführt, dessen Achsen aus der Verhaltens-, Struktur- und Geometriesicht bestehen. Innerhalb dieser Sichten lassen sich unterschiedliche Abstraktionsebenen definieren. Zum Beispiel werden in der Verhaltenssicht die algorithmische Ebene oder Register-Transfer-Ebene (RTL) unterschieden. Wie oben bereits erwähnt wurde, werden heute als Softmakros verfügbare IP's hauptsächlich als RTL-Modelle ausgeliefert.

Auf der RT-Ebene wird das zeitliche Verhalten des Entwurfs in diskreten Taktzyklen beschrieben. Bei einem synchronen Entwurf werden die Daten, je nach Spezifikation, mit fallender oder steigender Taktflanke in Speicherelemente übernommen. Der Entwurf selbst wird in der Strukturbeschreibung als eine Architektur von Registern, Signalleitungen und Verarbeitungseinheiten wie beispielsweise Addierer oder Multiplizierer dargestellt. Die Struktur aus Komponenten inklusive deren Verbindungen wird als Datenpfad bezeichnet. Der zeitliche Ablauf der Schaltung wird mit endlichen Automaten, den sogenannten Steuerwerken, gesteuert.

Um der steigenden Komplexität eingebetteter Systeme gerecht zu werden, reichen die durch die Sprachen VHDL und Verilog bereitgestellten Hilfsmittel zur Hardware-Beschreibung allerdings nicht mehr aus. Die Spezifikation muss auf höhere Abstraktionsebenen verlagert werden, wo dann sogenannte Systembeschreibungssprachen

zum Einsatz kommen. Hierfür werden unter anderem auch objektorientierte Methoden und Hilfsmittel verwendet. In [61] wird der objektorientierte Hardware-Entwurf am Beispiel der Programmiersprache Java aufgezeigt. Andere Systembeschreibungssprachen wie etwa SystemC [38] oder SpecC [35] wurden für die Beschreibung der Schaltungen in C entwickelt.

Neben diesen neu aufkommenden Beschreibungssprachen sind in den letzten Jahren auch Weiterentwicklungen der Sprachen Verilog und VHDL für die Spezifikation auf Systemebene entstanden. Es ist dies zum Einen SystemVerilog [96] und Objective VHDL [75]. Zum Anderen wird versucht, auch graphische Modellierungssprachen für die Hardware-Entwicklung heranzuziehen. Ein wichtiger Vertreter ist hier die UML [16] sowie ihre Erweiterung „Echtzeit UML“ [28] zur Spezifikation von Echtzeitsystemen. Aufgrund der Tatsache, dass UML ursprünglich für die Beschreibung komplexer Software-Systeme entwickelt wurde, sind deren Hilfsmittel zur Modellierung von Hardware beschränkt. Aus diesem Grund gewinnt die Systemmodellierungssprache SysML [103] an Bedeutung.

2.2.2 Hardware-Synthese

Wie oben bereits beschrieben wurde, wird mit Hilfe der Entwurfsautomatisierung versucht, Rechensysteme zum Chipentwurf zu nutzen. Der Vorgang der Umwandlung einer in einer Hochsprache wie etwa VHDL beschriebenen Schaltung in logische Schaltkreise wird als *Hardware-Synthese* bezeichnet. Abbildung 2.5 zeigt einen VHDL-basierten ASIC-Entwurfsablauf nach [76].

Ausgehend von einer Systemspezifikation wird ein Verhaltensmodell des zu entwickelnden Systems erstellt, welches zu einer Beschreibung auf RT-Ebene verfeinert wird. Diese Verfeinerung kann automatisiert durchgeführt werden und wird deshalb als *High-Level-Synthese* bezeichnet. Aufgrund des komplexen Entwurfsraums werden mit der High-Level-Synthese oft nur suboptimale Implementierungen generiert, weshalb in der Praxis der Schritt zur RT-Ebene meist von Hand vollzogen wird.

Ab dieser Stelle beginnt dann die automatische Synthese des Entwurfs. Dabei ist allerdings darauf zu achten, dass nur die Teilmenge der Sprachkonstrukte der verwendeten Hardware-Beschreibungssprache verwendet wird, die von einem Synthesewerkzeug in Hardware umgesetzt werden kann (synthetisierbare Untermenge). Nachdem von dem Synthesewerkzeug aus einer RTL-Beschreibung eine Netzliste auf Gatterebene erzeugt wurde, können in diese Teststrukturen eingebaut werden, die es später ermöglichen, den Chip zu testen. Nach der Platzierung und Verdrahtung der Gatter auf einer vorgegebenen Fläche und mit einer vorgegebenen Verzögerung, wird dann ein Prototyp erstellt, der mit Testmustern getestet wird. Die Implementierungen der höheren Abstraktionsebenen werden mit HDL-Simulatoren verifiziert, wobei die

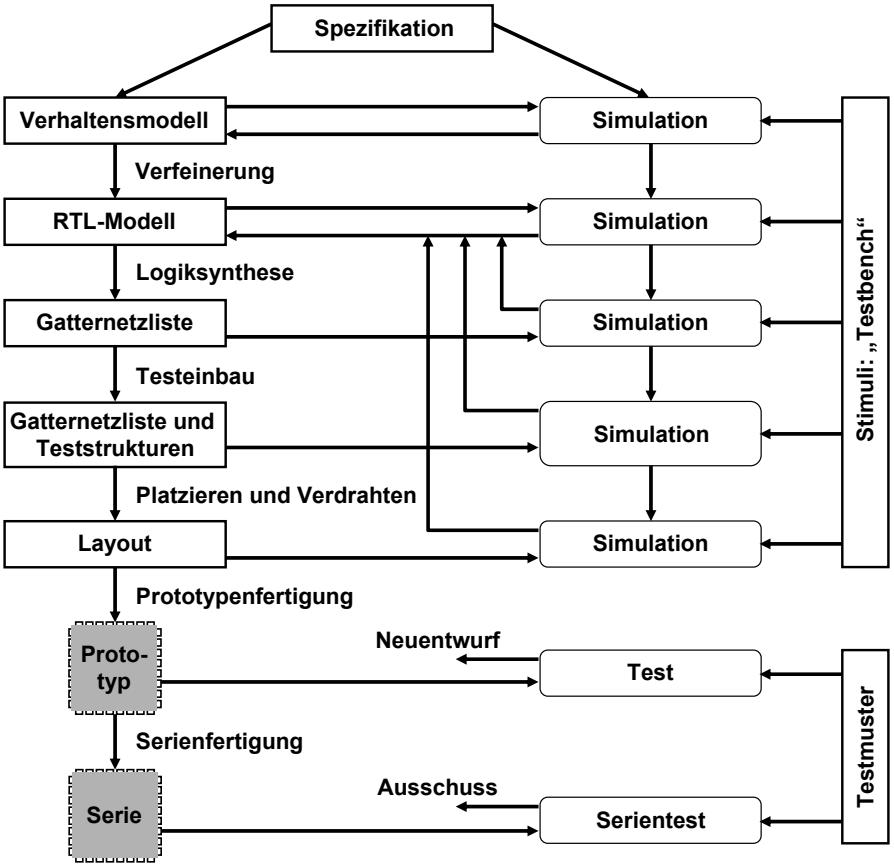


Abbildung 2.5: VHDL-basierter ASIC-Entwurfsablauf [76].

Schaltungen in der *Testbench* stimuliert und deren Ausgaben überprüft werden.

Ist der Prototyp ausreichend getestet, kann der Chip in Serie produziert werden. Da die Chipherstellung einen erheblichen Kostenfaktor darstellt, sollte ein Prototyp der Hardware erst dann hergestellt werden, wenn der Entwurf bereits einen hohen Reifegrad aufweist. Da für große Entwürfe die Simulation einen erheblichen Zeitaufwand darstellt, muss die Verifikation beschleunigt werden. Dies kann dadurch geschehen, dass anstelle eines eigens erzeugten Prototypen-Chips rekonfigurierbare Hardware-Bausteine verwendet werden. Hier kommen heute hauptsächlich *Field Programmable Gate Arrays (FPGAs)* zum Einsatz, die in Abschnitt 2.4 noch gesondert betrachtet werden. Da FPGAs sich in ihrer Hardware-Architektur wesentlich von ASICs unterscheiden, ändert sich der Synthesefluss entsprechend.

Bei der Synthese für FPGAs wird ausgehend von der Gatternetzliste, welche noch mit herkömmlichen Synthesewerkzeugen erstellt werden kann, zunächst eine Abbildung der Netzliste auf Komponenten des FPGAs vorgenommen. Danach werden diese entsprechend der Architektur des FPGAs platziert und verdrahtet.

Bei der Chipentwicklung kommen inzwischen auch Mischformen zwischen ASICs und FPGAs vor. Wird bei den *Standardzellen-ASICs* der Chip noch mit mehreren IP-Komponenten sowie spezieller Hardware entwickelt, so werden bei den *Structured ASICs* Plattformen bestehend aus festen IP-Komponenten kombiniert mit einem integrierten FPGA für die Hardware-Entwicklung verwendet. Auf der anderen Seite stehen FPGAs, in die in Zukunft immer mehr feste Hardware-IP-Komponenten integriert werden.

2.2.3 Verifikation der Hardware

Testbenches

Wie oben bereits angesprochen wurde, kann das Verhalten einer Hardware-Schaltung auf allen Abstraktionsebenen simuliert werden. Dabei wird der zu testende Entwurf in eine Umgebung eingebettetet, die die Eingänge des Entwurfs stimuliert und dessen Ausgaben prüft. Diese Umgebung wird *Testbench* genannt und kann in derselben Sprache implementiert werden, in der auch der Entwurf implementiert ist. Wird für die Testbench nur die synthetisierbare Untermenge der Hardware-Beschreibungssprache verwendet, kann die Testbench sogar mit dem Entwurf zusammen in ein FPGA synthetisiert und so die Verifikation nochmals beschleunigt werden.

Da für komplexere Systeme die sprachlichen Mittel von Hardware-Beschreibungssprachen für die Implementierung von Testbenches nicht mehr ausreichen, wurden zu diesem Zweck spezielle Sprachen wie *e* [101] oder VERA [40] entwickelt. Diese Sprachen unterstützen beispielsweise die Wiederverwendung von Code, können automatische und zufällige Testvektoren generieren und unterstützen die Implementierung der Testbench auf unterschiedlichen Abstraktionsebenen.

JTAG

Mit *JTAG* wird der Standard IEEE 1149.1 der *Joint Test Action Group* bezeichnet, der unter anderem für das Testen von Hardware verwendet werden kann. Mit JTAG lässt sich ein Chip über Konfigurations- und Kontrollsequenzen, die sich zu komplexen Befehlen ausbauen lassen, kontrollieren. Darüber hinaus wird JTAG unter anderem für die Konfiguration von FPGAs sowie als Debugging-Schnittstelle für Mikrokontroller verwendet. Darauf wird in Abschnitt 2.3.2 noch näher eingegangen.

Eine andere Bezeichnung für JTAG ist *Boundary scan*, da es zum sequentiellen Testen mit Hilfe sogenannter Scann-Ketten verwendet wird. Dabei werden spezielle Register innerhalb eines Chips zu einer sequentiellen Kette zusammengeschaltet. Die Register besitzen zwei Operationsmodi. Im Scann-Modus agiert das Register als

Schieberegister, aus dem der aktuelle Status heraus und ein neuer Status hineingeschoben wird. So kann der Zustand des Chips ausgelesen und analysiert, sowie von außen verändert werden. Auf diese Weise können auch mehrere Chips untereinander in einer Kette verbunden und geprüft werden.

2.3 Software-Entwicklung für eingebettete Systeme

In diesem Abschnitt wird die Software-Entwicklung für eingebettete Systeme behandelt. Dazu wird zunächst der allgemeine Aufbau der Entwicklungsumgebungen näher betrachtet, bevor dann im letzten Abschnitt ein weit verbreiteter Schnittstellenstandard von Prozessoren für Software-Debugger vorgestellt wird.

2.3.1 Entwicklungsumgebungen

Da eingebettete Systeme auf spezielle Anwendungen hin zugeschnitten sind, haben sie spezielle Eigenschaften, die eine direkte Software-Entwicklung auf dem System selbst unmöglich machen. So steht beispielsweise oft nicht genügend Rechenleistung für die Kompilierung großer Programmpakete oder genügend Speicherplatz zur Verfügung. Viele eingebettete Systeme besitzen darüber hinaus keine Tastatur oder einen Monitor. Aus diesem Grund werden *Cross-Entwicklungsumgebungen* für die Software-Entwicklung verwendet. Abbildung 2.6 zeigt ein Beispiel einer solchen Entwicklungsumgebung.

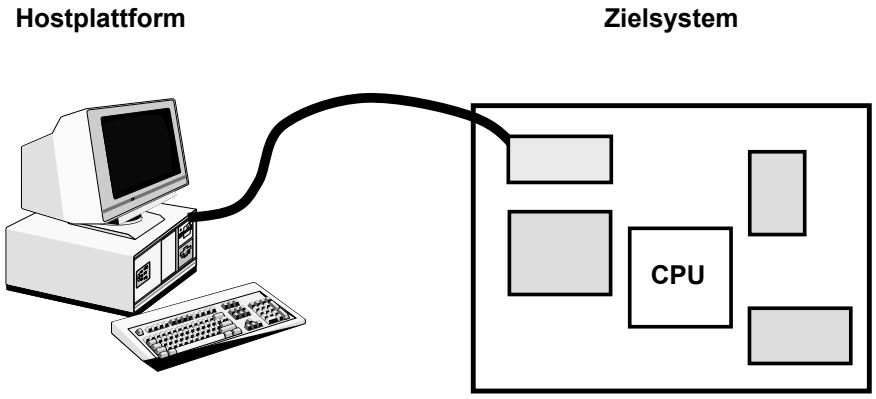


Abbildung 2.6: Anwendungsentwicklung mit einer Cross-Entwicklungsumgebung.

Die Entwicklung der Software wird auf einer Hostplattform durchgeführt, die über genügend Systemressourcen verfügt, um komplexe Entwicklungswerkzeuge verwenden zu können. Für die Übersetzung des Quelltextes werden Cross-Compiler und

-Assembler verwendet, d.h. der Compiler übersetzt die Programme nicht in Maschinencode des Hostprozessors sondern in Maschinencode der CPU des Zielsystems. Das Zielsystem wiederum ist mit dem Mikrokontroller sowie Peripheriekomponenten ausgestattet, die für das eingebettete System gebraucht werden. Für die Kommunikation zwischen Hostplattform und Zielsystem kann eine serielle, parallele, JTAG- oder Ethernet-Schnittstelle zum Einsatz kommen.

2.3.2 Debugging der Software

Über die im vorangegangenen Abschnitt beschriebene Schnittstelle zwischen Hostplattform und Zielsystem ist ein Cross-Debugger mit dem eingebetteten Mikrokontroller verbunden und kann für das Herunterladen und Debugging von Programmen sowie die Kontrolle des Zielsystems verwendet werden. Für das Debugging beginnt sich der Nexus-Standard (IEEE-ISTO 5001) durchzusetzen, der auf einer JTAG-Schnittstelle aufbaut [95]. Vor allem die Automobilindustrie drängt auf diesem Gebiet auf einheitliche Standards. Der Standard definiert eine Mehrzweckschnittstelle, die neben der Software-Entwicklung und dem Debugging auch die Performanzanalyse adressiert. Der Nexus-Standard definiert vier hierarchische Klassen mit unterschiedlichen Funktionen. Die Funktionen werden dabei sowohl von Seiten der Werkzeuge als auch mit speziellen Hardware-Modulen auf den Chips implementiert. Die Klasse eins bietet minimale und die Klasse vier eine maximale Debugging-Funktionalität an.

Beim Software-Debugging spielen *Breakpoints* eine wichtige Rolle. Ein Breakpoint ist eine Adresse im Programmcode, an der die Ausführung des Programms gestoppt werden soll. Breakpoints können auf unterschiedliche Art und Weise implementiert werden. Bei Software-Breakpoints wird das Programm dahingehend verändert, dass an der Adresse des Breakpoints der eigentliche Befehl entweder durch einen Unterprogrammaufruf zu einem Debugging-Monitor oder einen speziellen Debugging-Befehl ersetzt wird. Der Debugging-Monitor gibt dann den aktuellen Status des Systems an den Debugger weiter, der diesen anzeigen kann. Bei speziellen Debugging-Befehlen wird der Debugger nur über das Erreichen des Breakpoints informiert und ist selbst für das Auslesen der Informationen über den Systemstatus verantwortlich. Software-Breakpoints sind einfach zu implementieren, haben aber den Nachteil, dass sie den Programmcode verändern. Aus diesem Grund implementieren viele Mikrokontroller Hardware-Breakpoints, die den Programmzähler überwachen. Hardware-Breakpoints verändern zwar nicht den Programmcode, benötigen aber zusätzliche Logik, weshalb sie in ihrer Anzahl beschränkt sind. Mit dem Nexus-Standard konforme Mikrokontroller müssen mindestens zwei Hardware-Breakpoints implementieren.

2.4 Hardware-Emulation

Wie in Abschnitt 2.2.2 bereits angesprochen wurde, verursacht die Verifikation einer Schaltung mit Chip-Prototypen sehr hohe Kosten für dessen Produktion. Um allerdings dennoch gegenüber der Simulation einen Geschwindigkeitsvorteil bei der Verifikation zu erhalten, wird die *Emulation* eingesetzt. Der Begriff der Emulation wurde ursprünglich im Zusammenhang mit dem Wechsel von Rechenanlagen benutzt, bei dem wichtig war, dass alle vorhandenen Programme auf der neuen Anlage ablaufen konnten. In [25] findet sich folgende Definition:

Emulation: Die Anpassung und Abarbeitung des Befehlsvorrates einer Rechenanlage A durch geeignete Mikroprogrammierung in einer anderen Rechenanlage B heißt Emulation. Man sagt auch, B emuliert A. Bei der Emulation verhält sich die Rechenanlage B so, als ob sie gleich A wäre.

Diese Definition, die ursprünglich für Software getroffen wurde, lässt sich auf die Emulation von Hardware erweitern. In [58] wird dazu die Spezifikation einer Schaltung als Programm betrachtet und die dafür vorgesehene Zieltechnologie als Rechenanlage A. Ein Emulator ist dann eine Anlage, die in einer von der Zieltechnologie abweichenden Technologie implementiert ist, aber eine Spezifikation, die für die Zieltechnologie erstellt wurde, ausführen kann.

Neben dem Geschwindigkeitsvorteil können Emulatoren auch oft direkt mit der Umwelt verbunden und der Entwurf unter realen Einsatzbedingungen getestet werden. Dies ist bei Simulatoren schwierig, da nicht die geeigneten Schnittstellen zur Verfügung stehen.

Für die Emulation einer Schaltung können unterschiedliche Emulator-Technologien zum Einsatz kommen. Bei der prozessorbasierten Emulation wird eine Rechenanlage verwendet, die aus einer Vielzahl spezieller Prozessorelemente besteht. Die Register innerhalb dieser Prozessorelemente werden als Speicherelemente für die Register der emulierten Schaltung verwendet. Die Prozessorelemente selbst emulieren die kombinatorische Logik zwischen den Registern der Schaltung. In [42] wird ein Emulations-System vorgestellt, welches auf einem VLIW-Prozessor basiert. Mit einem solchen System ist die Emulation synthetisierter RT-Verhaltensbeschreibungen möglich. RT-Verhaltensbeschreibungen sind das Ergebnis der High-Level-Synthese und weisen ähnliche Strukturen wie VLIW-Prozessoren auf.

Eine andere Möglichkeit der Emulation besteht darin, rekonfigurierbare Hardware-Bausteine, wie etwa *Field Programmable Gate Arrays (FPGAs)*, zu verwenden. Die FPGAs bestehen aus einer Reihe konfigurierbarer Logikblöcke sowie eines konfigurierbaren Verbindungsnetzwerkes. Im Folgenden werden FPGAs näher beschrieben,

da diese als Grundlage für die in dieser Arbeit vorgestellte Entwurfsmethodik für eingebettete Hardware/Software-Systeme dienen.

2.4.1 Überblick über FPGAs

Aktuell existieren am Markt zwei unterschiedliche Technologien für FPGAs. Einmalig programmierbare FPGAs besitzen sehr viele programmierbare sich kreuzende, isolierte Leitungen, welche durch einen gezielten Stromstoss verbunden werden können. Auf diese Weise entsteht eine Schaltung, die irreversibel im FPGA verbleibt. Der Vorteil dieser FPGAs besteht darin, dass die Hardware-Schaltung direkt beim Booten des Systems zur Verfügung steht. Nachteil dieser Technologie ist allerdings, dass Fehler in der Schaltung nicht mehr rückgängig gemacht werden können.

Rekonfigurierbare FPGAs bestehen dagegen aus statischen SRAM-Zellen, die unterschiedliche logische Funktionen speichern können. Die Verdrahtung dieser SRAM-Zellen untereinander wird ebenfalls mit gespeicherten Bits realisiert. Dabei kann die Programmierung der Bits für die SRAM-Zellen und die Verbindungsstruktur über JTAG erfolgen. Vorteil dieser FPGAs ist ihre Wiederverwendbarkeit. Sie haben aber den Nachteil, dass die Schaltung beim Booten des Systems zunächst in das FPGA geladen werden muss, was bei großen FPGAs einige Zeit in Anspruch nehmen kann. Da dies beim Prototyping von Hardware allerdings keine Rolle spielt, wird diese Technolgie für die Emulation verwendet und deshalb im Folgenden näher betrachtet.

Bei SRAM-basierten FPGAs wird zwischen drei unterschiedlichen Konfigurationsarten unterschieden. Bei der Konfiguration zur Übersetzungszeit (*compile time reconfiguration*) wird die Schaltung zunächst mit einem Synthesewerkzeug in eine Konfiguration für das FPGA übersetzt und beim Booten dann geladen. Bei der globalen Rekonfiguration zur Laufzeit (*global run time reconfiguration*) wird das FPGA zur Laufzeit, z.B. von einem Mikrokontroller, komplett neu konfiguriert. Bei der partiellen Rekonfiguration zur Laufzeit (*partial run time reconfiguration*) werden nur Teile des FPGAs neu konfiguriert. In den folgenden Abschnitten wird die Architektur von rekonfigurierbaren FPGAs anhand Xilinx Virtex-II-FPGAs näher beschrieben.

2.4.2 Funktionsweise SRAM-basierter FPGAs

Als Beispiel eines SRAM-basierten FPGAs soll hier die Architektur der konfigurierbaren Logikblöcke des im weiteren Verlauf dieser Arbeit verwendeten Virtex-II-FPGAs von Xilinx besprochen werden. Ein konfigurierbarer Logikblock besteht aus zwei *Slices*, die übereinander angeordnet sind. Abbildung 2.7 zeigt den Aufbau eines Slices.

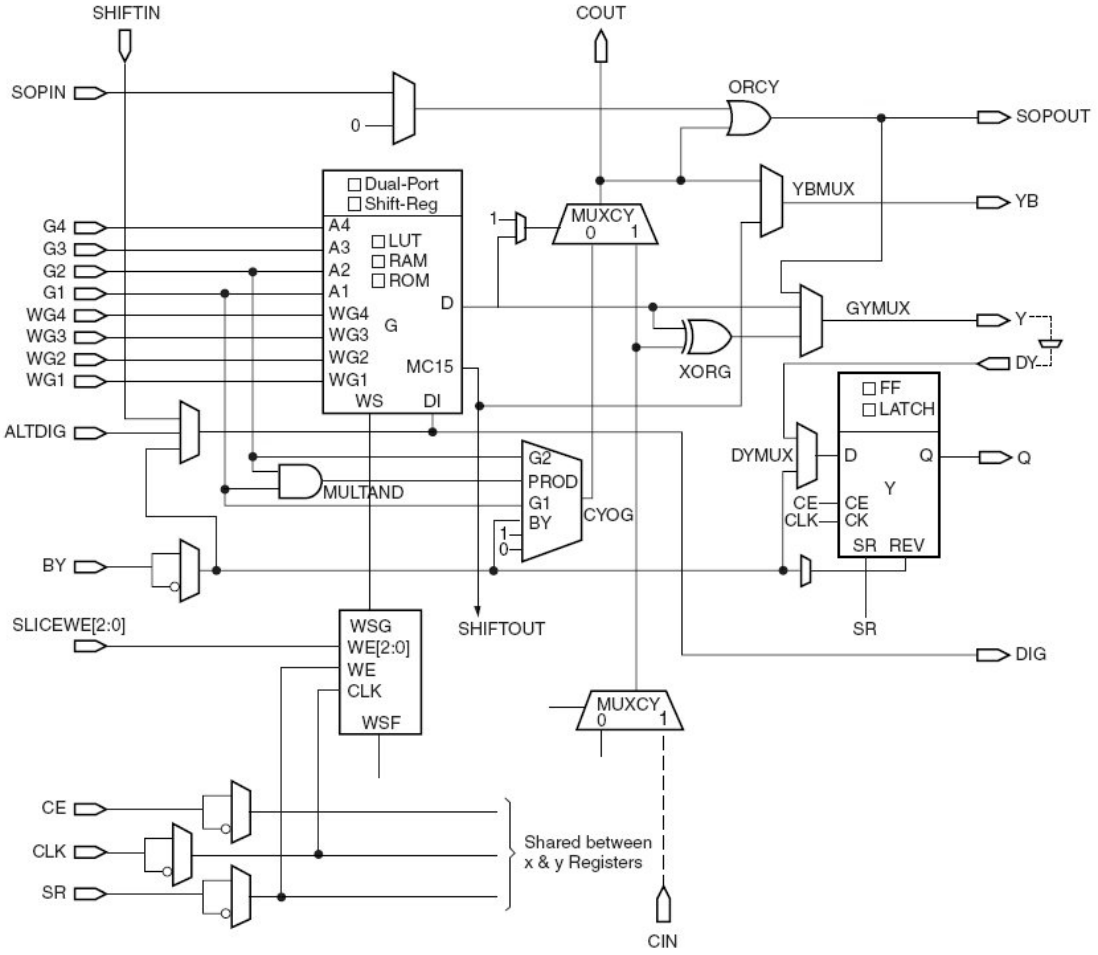


Abbildung 2.7: Aufbau der oberen Hälfte eines FPGA-Logikblocks [113].

Den Kern des Slices bilden der Funktionsgenerator (*Lookup table (LUT)*) und ein Register, welches sowohl als flankengesteuertes Flip-Flop als auch als pegelgesteuertes Latch verwendet werden kann. Mit dem Funktionsgenerator können Funktionen aus bis zu acht Variablen (G1-G4 und WG1-WG4) erzeugt werden. Die LUT kann aber auch als RAM oder ROM verwendet werden. Die Konfigurationsdaten werden in SRAM-Zellen gehalten, die bei Xilinx FPGAs als eine Art großes Schieberegister implementiert sind. Auf diese Art kann das FPGA mit einer Takt- und einer Datenleitung konfiguriert werden.

Mit speziellen Setz- und Rücksetz- sowie Takt- und Freischaltleitungen kann das Register des Slices kontrolliert werden. Darüber hinaus kann über zusätzliche Steuerleitungen der Datenpfad des Slices über Multiplexer geändert und so beispielsweise der Funktionsgenerator umgangen werden.

2.4.3 Hardware-Architektur von FPGAs

Mit den oben beschriebenen rekonfigurierbaren Logikblöcken lassen sich beliebige Schaltungen emulieren. Durch ihre generische Struktur benötigen sie allerdings sehr viel Chipfläche gegenüber der Implementierung der gleichen Funktionalität mit herkömmlichen ASICs. Aus diesem Grund sind FPGA-Hersteller dazu übergegangen, häufig beim Systementwurf benötigte Komponenten als Hardmakros zu implementieren. Dazu zählen z.B. Addierer, Multiplizierer, RAM-Bausteine und ganze Prozessoren. Die in Xilinx FPGAs integrierten 18-bit Multiplizierer eignen sich beispielsweise sehr gut zur Realisierung von DSP-Anwendungen. Die Virtex-II-Pro-Serie von Xilinx enthält je nach FPGA-Typ einen oder zwei eingebettete PowerPC405 von IBM als Hardmakro, wie in Abbildung 2.8 dargestellt. Diese neuen FPGA-Architekturen werden auch als Plattform-FPGAs bezeichnet [113]. Die Firma Xilinx geht auf diesem Gebiet inzwischen so weit, dass sie mit ihrer neuesten FPGA-Familie unterschiedliche FPGA-Typen für einzelne Marktsegmente anbietet [65].

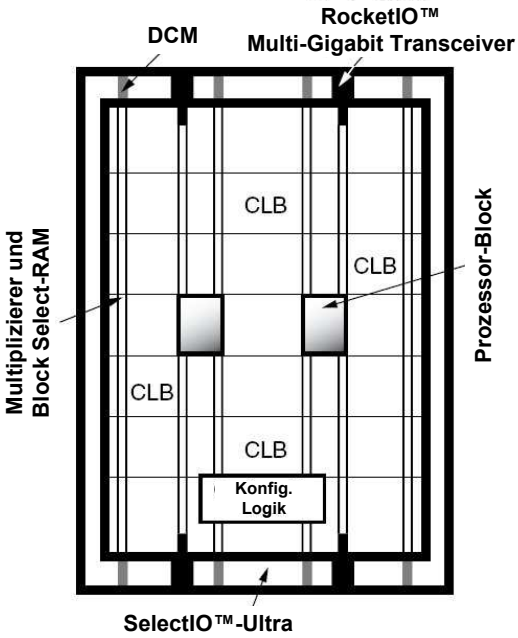


Abbildung 2.8: Architektur des Virtex-II-Pro-FPGAs von Xilinx [113].

Abbildung 2.9 zeigt die Architektur des Virtex-II FPGAs von Xilinx. Neben den rekonfigurierbaren Logikblöcken (CLBs) sind Block Select-RAMs (Block-RAMs) und Multiplizierer als lange vertikale Streifen auf dem Chip zu erkennen. Daneben gibt es digitale Taktmanager (DCMs) für die Implementierung unterschiedlicher Taktdomänen sowie spezielle E/A-Anschlüsse (EAB) an den Rändern des FPGAs.

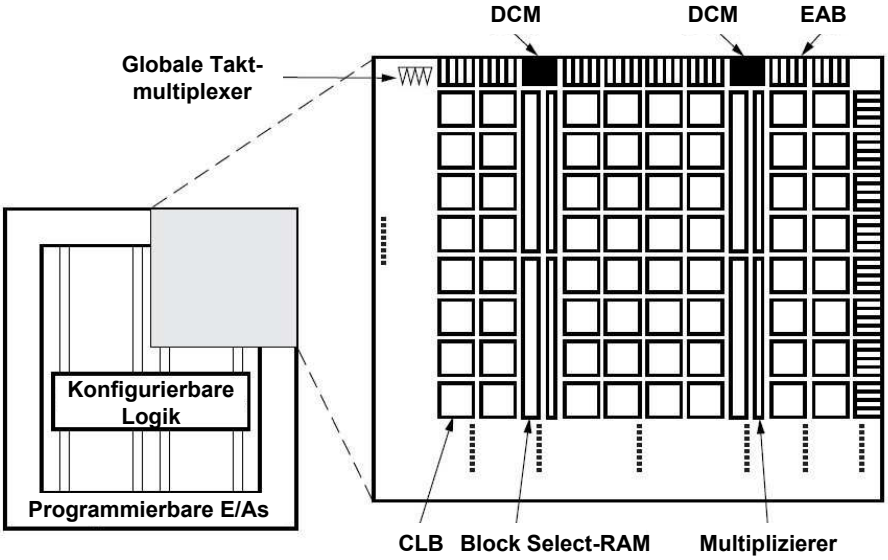


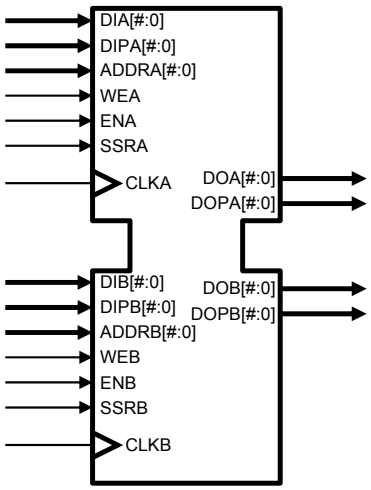
Abbildung 2.9: Architektur des Virtex-II-FPGAs von Xilinx [113].

Integrierte RAM-Module

Mittlerweile bieten alle am Markt verfügbaren FPGAs dem Entwickler spezielle Speicher-Module auf dem Chip an. Bei Xilinx FPGAs wird zwischen zwei unterschiedlichen On-chip-Speicher-Arten unterschieden. Die Block-RAMs sind auf dem FPGA in vertikalen Reihen angeordnet (siehe Abbildung 2.9). Jedes Block-RAM bietet bis zu 18KBit Speicher, von denen 2KB zur Speicherung der Daten und die restlichen Bits zur Speicherung der Datenparität genutzt werden. Dies ergibt für das größte Virtex-II-FPGA mit 168 Block-RAM-Modulen eine Gesamtspeicherkapazität von 336KB.

Abbildung 2.10 (a) zeigt die Schnittstellen des Virtex-II Block-RAMs. Das Block-RAM kann mit zwei voneinander unabhängigen Ports A und B betrieben werden. Darüber hinaus ist die Breite des Adress- und Datenbusses flexibel einstellbar. Diese sind in der Tabelle in Abbildung 2.10 (b) dargestellt. Durch diese hohe Flexibilität und Antwortzeiten, die innerhalb eines Taktes liegen, sind die Block-RAMs sehr vielseitig einsetzbar.

Die in den einzelnen Logikblöcken für die Realisierung boolscher Funktionen zuständigen LUTs können für die Implementierung verteilte Select-RAM-Blöcke verwendet werden. Sie bilden dann ein 16 x 1-Bit synchrones RAM. Werden LUTs als verteiltes Select-RAM verwendet, dann fehlt allerdings diese Ressource zur Implementierung boolscher Logik der zu emulierenden Schaltung.



(a)

Adressbus	Datenbus	Anzahl der Paritätsbits
ADDR[13:0]	DATA[0]	Nicht verfügbar
ADDR[12:0]	DATA[1:0]	Nicht verfügbar
ADDR[11:0]	DATA[3:0]	Nicht verfügbar
ADDR[10:0]	DATA[7:0]	DP[0]
ADDR[9:0]	DATA[15:0]	DP[1:0]
ADDR[8:0]	DATA[31:0]	DP[3:0]

(b)

Abbildung 2.10: Aufbau und Schnittstellen eines Virtex-II Block-RAM-Bausteins.

FPGA-E/A-Blöcke

Für die Kommunikation mit Peripheriekomponenten sind die Xilinx-FPGAs mit E/A-Blöcken ausgestattet, die unterschiedliche Funktionen bereitstellen. Die E/A-Blöcke können sowohl als einfache Ein- oder Ausgabe-Leitungen fungieren, oder paarweise als differentieller Ein- oder Ausgang. Der Aufbau eines Virtex-II E/A-Blocks ist in Abbildung 2.11 dargestellt.

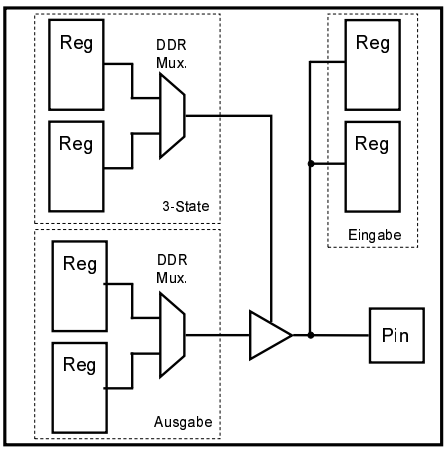


Abbildung 2.11: Aufbau eines Virtex-II E/A-Blocks [113].

Die in den E/A-Blöcken vorhandenen Tristate-Schaltkreise bieten die Möglichkeit zur Implementierung von Bussen. Die doppelt ausgelegten Eingabe- sowie Ausgabe-Register in Kombination mit den DDR-Multiplexern (*double data rate*) können zur Erhöhung des Datendurchsatzes bei der Ein- bzw. Ausgabe dienen. Hierbei werden die Registerpaare mit unterschiedlichen Taktflanken betrieben, um eine doppelte Datenrate an den jeweiligen Ausgängen zu erreichen.

Integrierte Taktverwaltung

Mit Hilfe der digitalen Taktverwaltung (*Digital Clock Manager (DCM)*) lassen sich verschiedene Taktdomänen innerhalb des FPGAs realisieren. In Virtex-II-FPGAs stehen dafür 16 Taktpuffer zur Verfügung, mit denen 16 Taktdomänen implementiert werden können. Da die Taktpuffer als 2:1 Multiplexer realisiert sind, können allerdings nur bis zu acht Taktnetze global im FPGA verwendet werden. Diese acht Taktnetze sind mit sämtlichen synchronen Elementen wie Register, Block-RAM, Multiplizierer oder E/A-Blöcken verbunden. Das Synthesewerkzeug von Xilinx kann für diese acht Taktnetze eine automatische Platzierung und Verdrahtung des Entwurfs vornehmen.

Sind für den Prototyp eines Entwurfs mehr als acht Taktdomänen notwendig, muss der Entwurf manuell im FPGA platziert werden. Das FPGA wird dafür in vier Quadranten eingeteilt. Die 16 globalen Taktmultiplexer sind in acht primäre und acht sekundäre Taktmultiplexer aufgeteilt, die sich jeweils die Eingänge und den Zugriff auf die vier Quadranten des FPGA teilen. In Abbildung 2.12 ist die Schaltung für zwei primäre (0P/1P) und zwei sekundäre (0S/1S) Taktmultiplexer dargestellt.

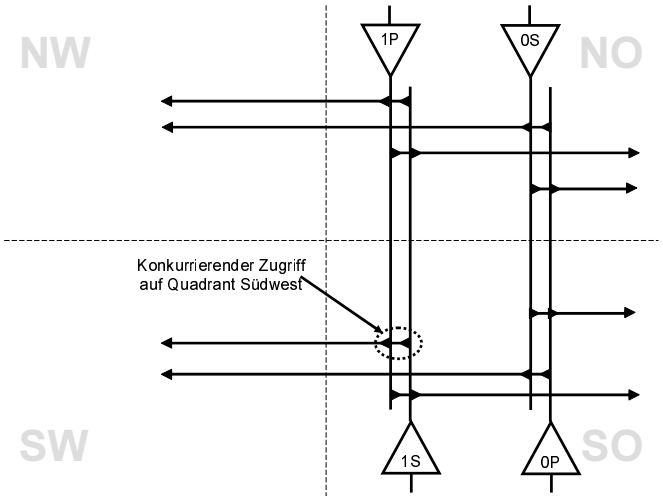


Abbildung 2.12: Konkurrierende Ansteuerung der vier Quadranten eines FPGAs durch Taktmultiplexer [113].

Aus der Anordnung der Taktmultiplexer ergibt sich, dass die vier Quadranten jeweils nur von einem der beiden gegenüberliegenden primären/sekundären Taktmultiplexer angesteuert werden können. Ein Beispiel für eine Einteilung eines Entwurfs mit 16 Taktdomänen $CLK_i : i = 0..15$ ist in Tabelle 2.1 dargestellt.

Taktnetz (FPGA oben)	CLK_0	CLK_1	CLK_2	CLK_3
Taktpuffer	7P	6S	5P	4S
Taktnetz (FPGA unten)	CLK_8	CLK_9	CLK_{10}	CLK_{11}
Taktpuffer	7S	6P	5S	4P
Quadrant NW	CLK_0	CLK_1	CLK_2	-
Quadrant SW	CLK_0	-	CLK_2	CLK_{11}
Quadrant NO	CLK_8	CLK_1	CLK_{10}	CLK_3
Quadrant SO	CLK_0	CLK_9	-	-
Taktnetz (FPGA oben)	CLK_4	CLK_5	CLK_6	CLK_7
Taktpuffer	3P	2S	1P	0S
Taktnetz (FPGA unten)	CLK_{12}	CLK_{13}	CLK_{14}	CLK_{15}
Taktpuffer	3S	2P	1S	0P
Quadrant NW	-	CLK_5	-	CLK_{15}
Quadrant SW	CLK_{12}	CLK_{13}	CLK_6	CLK_7
Quadrant NO	CLK_4	CLK_{13}	CLK_{14}	CLK_7
Quadrant SO	-	-	-	CLK_{15}

Tabelle 2.1: Platzierung eines Entwurfs mit 16 Taktdomänen mit acht primären und acht sekundären Taktmultiplexern[113].

Werden mehr als 16 Taktdomänen für eine korrekte Implementierung eines ASIC-IP-Kerns notwendig, stehen im FPGA noch sogenannte *Backbone-Netze* zur Verfügung. Dies sind 24 vertikal und horizontal verlaufende Leitungen, die als zusätzliche Taktnetze verwendet werden können. Bei der Verwendung dieser Netze ist allerdings eine genauere Analyse des zeitlichen Verhaltens der Schaltung notwendig, da diese Netze nicht speziell für die Implementierung von Takten vorgesehen sind.

2.4.4 Debugging mit integrierten Logikanalysatoren

Für das Debugging der Hardware können bei Xilinx FPGAs neben der Verwendung herkömmlicher externer Logikanalysatoren auch *interne* Logikanalysatoren verwendet werden. Diese sind in dem Werkzeug ChipScope Pro von Xilinx implementiert. Abbildung 2.13 zeigt das Debugging mit ChipScope Pro.

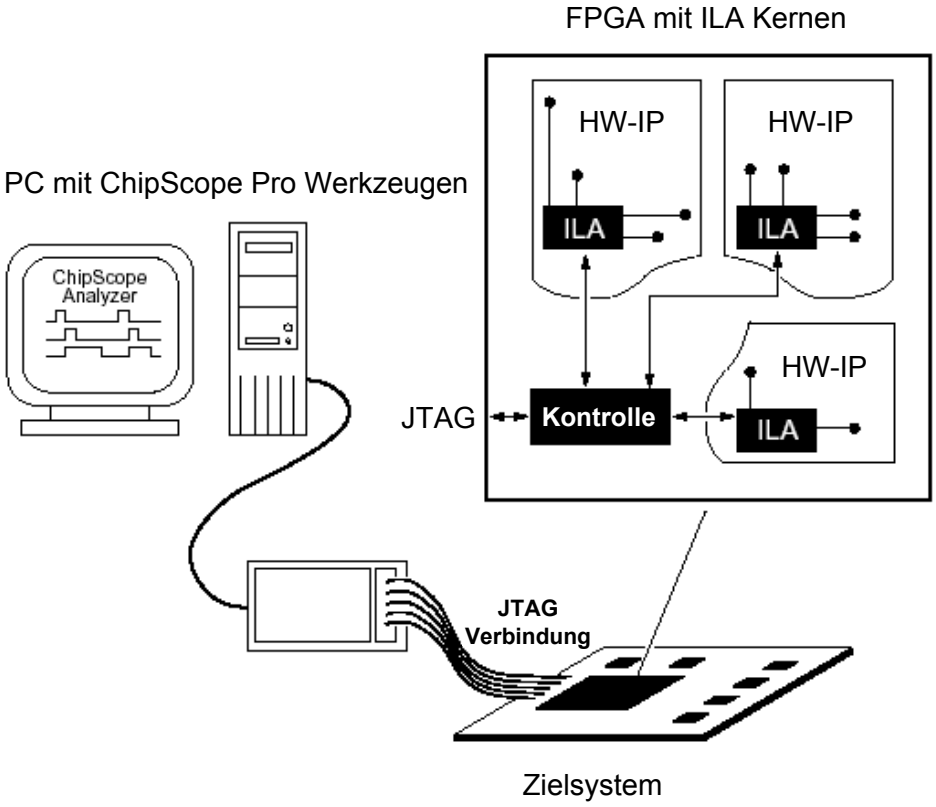


Abbildung 2.13: Debugging von Hardware-IP-Komponenten mit in FPGAs integrierten Logikanalysatoren [114].

ChipScope Pro besteht aus Hardware- und Software-Komponenten, die über die JTAG-Schnittstelle des FPGAs miteinander kommunizieren. Die Hardware-Komponenten stellen Trigger-Logik zur Verfügung und zeichnen interne Signale der zu untersuchenden IP in den internen Block-RAMs des FPGAs auf. Diese Signale können dann entweder manuell abgerufen und auf einem PC dargestellt, oder automatisch zum PC übertragen werden, wenn die Kapazität der Block-RAMs erschöpft ist.

Für die Integration der Logikanalysator-Hardware-Module werden bei ChipScope Pro die Werkzeuge *ChipScope Pro Core Generator* und *ChipScope Pro Core Inserter* verwendet. Diese fügen die Debugging-Module in die Netzliste des zu untersuchenden Entwurfs ein. Diese Methode hat den Vorteil, dass die Signale, die beobachtet werden sollen, nicht auf Anschlüsse des FPGAs herausgeführt werden müssen, son-

dern intern untersucht werden können. Auf der anderen Seite werden für die Integration der Debugging-Module FPGA-Ressourcen wie Takt- und Signalnetze sowie Block-RAMs benötigt. Darüber hinaus ist nur eine Signalaufzeichnung mit dem Takt des FPGAs möglich.

Externe Logikanalysatoren können dagegen die Signale mit einer höheren Taktrate abtasten. Dadurch sind genauere Aussagen bezüglich des zeitlichen Verhaltens des Entwurfs möglich. Der Nachteil der Lösung besteht darin, dass für das Herausführen der zu untersuchenden Signale Anschlüsse des FPGAs benötigt werden, die dann nicht für andere Zwecke, wie etwa Kommunikation mit externen Komponenten, zur Verfügung stehen. Die Verwendung eines FPGAs zur Emulation hat gegenüber einem Mikrokontroller-Chip aber immer den Vorteil, dass *beliebige* Signale zum Debugging herausgeführt werden können.

3 Stand der Technik

In diesem Kapitel werden Methoden vorgestellt, die beim Entwurf von Hardware und Software für SoCs zum Einsatz kommen. Speziell für die Software ist nach einem Bericht der Vereinigung internationaler Halbleiterhersteller (*Semiconductor Industry Association (SIA)*) davon auszugehen, dass deren Entwicklung für zukünftige Systeme 80% der Entwurfszeit in Anspruch nimmt [93]. Damit ist erkennbar, dass die Software bei der SoC-Entwicklung auf dem kritischen Pfad liegen wird und es neben der *Entwurfslücke* für Hardware [70] auch zu einer Entwurfslücke für die Entwicklung der eingebetteten Software [53] kommen kann. Insbesondere verlängert sich die Entwurfszeit für die Systeme, wenn mit der Software-Entwicklung erst nach Fertigstellung der Hardware begonnen wird. In Abschnitt 3.1 werden deshalb Verfahren zur Software-Entwicklung in einer frühen Phase des Entwurfs auf Hardware-Modellen vorgestellt.

Für die Diversifizierung eines SoC am Markt ist es darüber hinaus aber auch notwendig, vorhandene Hardware-Plattformen mit neu zu entwickelnden Hardware-Komponenten zu erweitern. Für diese Hardware-Komponenten muss dann auch Software entwickelt werden. In Abschnitt 3.2 werden Entwurfsmethoden betrachtet, die sich mit der parallelen Entwicklung von Hardware-Komponenten und deren Software für SoCs beschäftigen.

3.1 Software-Entwicklung in frühen Entwurfsphasen

In diesem Abschnitt werden Methoden für die Software-Entwicklung in einem frühen Stadium des Entwurfsprozesses vorgestellt. Dazu zählen Techniken, die auf der Modellierung der Hardware beruhen und Techniken, die mit einer konkreten Realisierung der Hardware arbeiten. Bei der Modellierung können einerseits Prozessoren mit unterschiedlichen Befehlssätzen nachgebildet werden. Diese Modelle werden als Befehlssatzsimulatoren (BSS) bezeichnet. Es kann aber auch ein virtuelles Prozessormodell, wie das der *Java Virtual Machine (JVM)*, verwendet werden, welches eine Abstraktion der eigentlichen Rechnerarchitektur darstellt und einen einheitlichen Befehlssatz über alle Hardware-Plattformen hinweg besitzt. Für die Software-Entwicklung können auch konkrete Hardware-Plattformen verwendet werden, die in Abschnitt 3.1.4 näher vorgestellt werden.

3.1.1 Entwicklung von Software-Algorithmen mit Befehlssatzsimulatoren

Bei der Befehlssatzsimulation wird ein Prozessor in Software nachgebildet. Dabei ist zwischen zwei unterschiedlichen Ansätzen zu unterscheiden. Der binäre Programmcode kann entweder interpretiert [50] oder für die Simulationsplattform neu übersetzt werden [23]. Die Übersetzung des Binärcodes kann dabei statisch oder dynamisch erfolgen. Dadurch wird zwar die Ausführungsgeschwindigkeit des Codes gegenüber der Interpretierung erhöht, es entsteht aber der Nachteil, dass kein selbstmodifizierender Code und keine Aufrufe anderer Programme verwendet werden können. Darüber hinaus ist die Modellierung des zeitlichen Verhaltens des Programms schwieriger.

BSS können auf unterschiedlichen Abstraktionsebenen implementiert werden. Ein sehr ungenaues aber dafür schnelles Modell bildet nur die Auswirkungen der Befehle nach, ohne deren zeitliches Verhalten zu berücksichtigen. Auf der *Transaktionsebene* werden dagegen Lese- und Schreibzugriffe auf dem Bus und die Kommunikation zwischen Komponenten modelliert und bei der zyklengenauen Befehlssatzsimulation kann die genaue Anzahl der Taktzyklen für eine Anwendung bestimmt werden [66].

Ein Beispiel eines zyklengenauen BSS ist der TSIM2 für den TriCore1-Mikrokontroller [50]. Er stellt eine Simulationsumgebung dar, bei dem Speicher, Unterbrechungsbehandlung und Adressabbildung der Peripheriegeräte programmierbar sind. Der TSIM2 kann für die Überprüfung der Software-Funktionalität, das Software-Debugging, die Performanzanalyse und für Vergleiche unterschiedlicher Software-Algorithmen eingesetzt werden.

Bei der Ausführung der Befehle werden allerdings das Pipelining oder Busprotokolle nicht berücksichtigt. Dadurch ist die angegebene Anzahl an Zyklen nur eine Abschätzung, die von der Anwendung abhängig ist. Der Simulator kann beispielsweise weder CPU-Stillstände (*CPU-Stalls*) aufgrund von Busaktivitäten simulieren noch achtet er auf Portlimitierungen des Registersatzes sowie Datenabhängigkeiten im Befehlsstrom.

Die Simulation von Peripheriegeräten ist darüber hinaus nur sehr rudimentär implementiert. Hier kann nur deren Adresse im Adressraum des Prozessors und ihr Unterbrechungsverhalten simuliert werden. Eine weiterführende Kommunikation mit der Peripheriekomponente ist nicht möglich.

3.1.2 Einsatz von Java in eingebetteten Systemen

Im vorangegangenen Abschnitt wurde bereits erwähnt, dass ein BSS den Befehlssatz eines Prozessors interpretiert. Der Befehlssatz ist dabei von Prozessor zu Prozessor unterschiedlich. Mit der Einführung der Java-Technologie wurde ein Ansatz verfolgt,

bei dem über der eigentlichen Prozessorarchitektur ein virtueller Befehlssatz realisiert wird, der die Portabilität von Programmen über unterschiedliche Hardware- und Betriebssystemplattformen hinweg garantiert. Damit ist es auch möglich, Software für eine Zielplattform zu entwickeln, die noch nicht in Hardware vorliegt, da für die Anwendungsentwicklung die virtuelle Maschine des Entwicklungssystems verwendet werden kann. Ursprünglich zur Programmierung eingebetteter Systeme entwickelt [19], wurde die Sprache aufgrund ihrer Plattformunabhängigkeit schnell für das Internet entdeckt. Ein beliebtes Anwendungsgebiet der Java-Technologie ist deshalb die internetbasierte Kontrolle eingebetteter Systeme [82].

Da bei Java in erster Linie ein virtueller Prozessor nachgebildet und von der darunterliegenden Hardware abstrahiert wird, ist die Ansteuerung von Hardware-Komponenten schwierig. Dies gilt einerseits für den Zugriff auf die Hardware selbst. Andererseits gibt es Probleme, wenn die Hardware-Komponenten auf dem Entwicklungssystem nicht verfügbar sind. Aus diesem Grund wurde in der Echtzeit-Spezifikation für Java (*The Real-time specification for Java (RTSJ)*) [15, 84] eine Klasse definiert, die Zugriff auf physikalischen Speicher erlaubt. Mit der RTSJ sollen darüber hinaus Probleme gelöst werden, die den Einsatz einer herkömmliche Java-Plattform in echtzeitkritischen eingebetteten Systemen verbieten [83]. Entsprechende echtzeitfähige JVMs stehen heute zur Verfügung [2].

Eine andere Möglichkeit wäre der Einsatz von Jini [30] zur Kapselung der Funktionalität von Hardware-Komponenten [81, 85]. Bei Jini werden Dienste hinter einer Schnittstelle verborgen. Auf diese Weise könnte ein Simulationsmodell eines Hardware-Moduls hinter einer Jini-Schnittstelle versteckt werden. Auf dem Entwicklungsrechner kommuniziert die Software dann mit dem Simulationsmodell und auf der Zielplattform mit der echten Hardware-Komponente.

3.1.3 Transaktionsbasierte Modellierung auf hohen Abstraktionsebenen

Für die Entwicklung komplexer eingebetteter Systeme kommen heute auch Methoden zum Einsatz, die auf sehr hohen Abstraktionsebenen ansetzen. Bei der transaktionsbasierten Modellierung mit SystemC (*Transaction Level Modeling (TLM)*) wird die Interaktion zwischen Hardware-Modulen mit Methodenaufrufen in C++ realisiert. Auf diese Art und Weise lässt sich ein komplexes System einfach und schnell aufbauen und es können Modelle der Hardware auf unterschiedlichen Abstraktionsebenen hinter den Schnittstellen der Module und der Kommunikationskanäle verborgen werden.

Mit VISTA [71] ist eine frühe Evaluierung der Performanz eines SoC auf Transaktionsebene möglich. Dort wird eine Anwendung für ein Hardware-Modell ent-

wickelt, welches aus zyklengenauen IP-Komponenten besteht. Es wird dort allerdings kein Mikrokontroller-Modell oder BSS verwendet, sondern der Quelltext wird auf einer *Black-box* mit Maschinencode des Hostrechners ausgeführt. Durch Cross-Kompilierung und Ausführung der Software auf der Zielplattform werden zeitliche Informationen gewonnen, die dem Simulationsmodell des SoC annotiert werden können. Das System ist nicht für das Software-Debugging des SoC geeignet, da zum Einen kein Software-Debugger in VISTA integriert ist und zum Anderen die Anwendung nicht für den Zielprozessor kompiliert wird. Dadurch kann der Entwickler den Programmverlauf auf Assembler-Ebene nicht verifizieren. Mit VISTA können 0.5 Sekunden in Echtzeit innerhalb von 20 Sekunden simuliert werden [71].

Durch die Kapselung der Verbindung zwischen Modulen in Kommunikationskanälen ist deren Kombination auf unterschiedlichen Abstraktionsebenen möglich. Allerdings ist eine blockbasierte Verifikation des SoCs auf RT-Ebene mit SystemC-TLM nicht sinnvoll [27]. Dies liegt im Wesentlichen darin begründet, dass eine architekturgenaue Simulation der Hardware in SystemC sehr viel Zeit in Anspruch nimmt.

Insgesamt konzentriert sich SystemC mehr auf die Entwicklung der Hardware. Dies ist unter anderem auch daran zu erkennen, dass die Version SystemC 3.0, die die Modellierung von Software im Allgemeinen und von Echtzeitbetriebssystemen im Besonderen beinhalten soll, schon seit Jahren diskutiert [37] aber immer noch nicht eingeführt wurde.

3.1.4 Software-Entwicklung mit Evaluierungsboards

Sowohl die Verwendung von BSS, Java und TL-Modellen der Hardware ist mit Problemen behaftet. Vor allem mangelnde Performanz kann die effiziente Software-Entwicklung behindern. Aus diesem Grund bieten alle nahmhaften Mikroprozessor-Hersteller Evaluierungsboards für ihre Produkte an [6, 44, 48].

Ein Evaluierungsboard stellt eine vollständige Software-Entwicklungsumgebung für einen Mikroprozessor dar. Auf dem Board ist der Mikroprozessor als Chip implementiert. Zusätzlich werden verschiedene Schnittstellen für die Anwendungsentwicklung sowie sehr große Speicherressourcen zur Verfügung gestellt. Darüber hinaus ist eine Schnittstelle für einen Software-Debugger implementiert. Für das Debugging der Hardware verfügt der Mikroprozessor-Chip oft über zusätzliche E/A-Anschlüsse, die den internen Status des Prozessors nach außen hin sichtbar machen. Die Einsicht ist allerdings meist auf den Prozessorbus beschränkt.

Ein Problem der Evaluierungsboards ist, dass die Architektur des Mikrokontrollers festgelegt ist. Dies gilt beispielsweise für die Größe der Caches oder für die Peripheriekomponenten. Bei einfacheren Kontrollern können diese in speziellen steckbaren

Gehäusen gefertigt und so auf dem Evaluierungsboard ausgetauscht werden. Beim AVR [8] 8-Bit-Mikrokontroller der Firma Atmel sind solche Module mit einer Anschlussanzahl von 8-44 erhältlich. Bei größeren Chips, die 16- und 32-Bit-Mikrokontroller implementieren, ist dies nicht mehr möglich. Dort ist der Chip fester Bestandteil des Entwicklungsboards.

Abbildung 3.1 zeigt beispielsweise das Evaluierungsboard *TriBoard TC1920A* der Firma Infineon Technologies AG mit einem Derivat des TriCore1-Mikrokontrollers. Der TC1920 basiert auf der TriCore1.3-Architektur und ist speziell für Infotainment-Anwendungen im Automobil zugeschnitten. Dafür kombiniert er sowohl Peripheriekomponenten wie etwa CAN und J1850 aus dem Automobilbereich als auch Standardschnittstellen wie ADC, SSC/SPI, UART, IrDA, I^2S und I^2C auf einem Chip. Die Anwendungen reichen von Navigationssystemen, Internet-Radios bis hin zu Notrufsystemen in Automobilen.

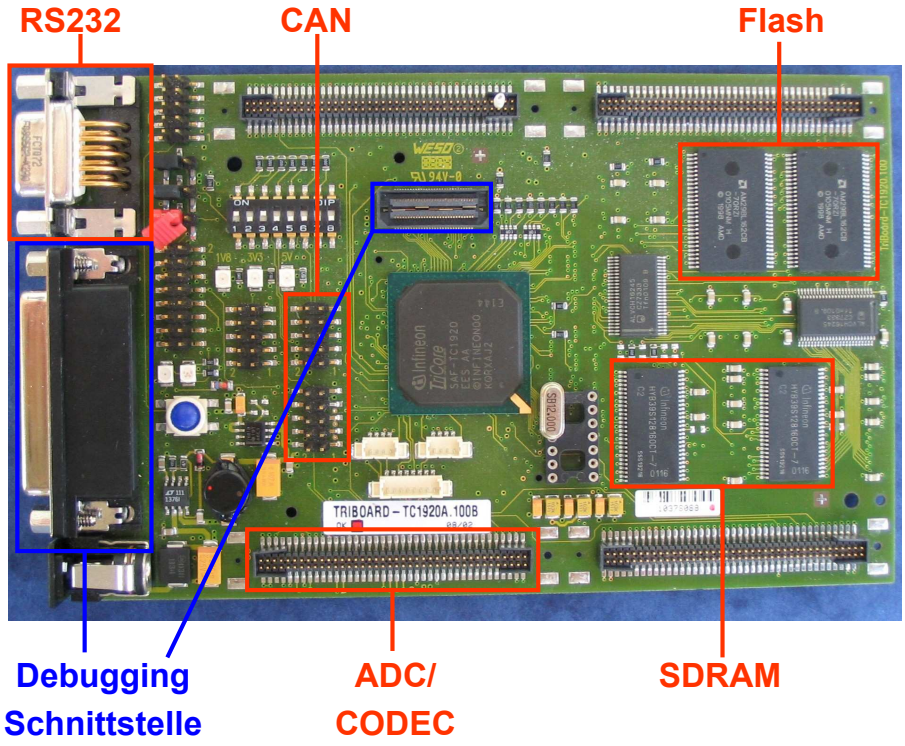


Abbildung 3.1: Evaluierungsboard TriBoard TC1920A.

3.2 Entwicklung eingebetteter Hardware/Software-Systeme

In diesem Abschnitt wird die Entwicklung von Hardware-Komponenten und deren Software für SoCs näher beleuchtet. In diesem Bereich ist immer noch ein Fokus

der Entwicklungswerkzeuge auf die Entwicklung der Hardware zu beobachten. Dies spiegelt sich z.B. bei der Entwicklung neuer Spezifikationssprachen auf Systemebene wieder, die Ende der 90er Jahre entwickelt wurden. Dort wurden neue Sprachen für die Hardware-Verifikation entwickelt, die neue Programmierparadigmen unterstützen [96, 40, 101]. Es wurde aber auch dazu übergegangen, aus der Software-Entwicklung bekannte Sprachen wie Java oder C++ für die Systementwicklung heranzuziehen [61, 33, 38].

In Abschnitt 3.2.1 wird zunächst die Hardware-Simulation vorgestellt und es werden auch Methoden zu deren Erweiterung für die Software-Entwicklung aufgezeigt. Abschnitt 3.2.3 beschäftigt sich dann mit der Emualtion von Hardware, mit der Geschwindigkeitsdefizite der Simulation ausgeglichen werden soll. Die formale Verifikation beschließt diesen Abschnitt.

3.2.1 Simulation von Hardware-Komponenten

Die Simulation der Hardware zählt zu den ältesten Verifikationsmethoden von Hardwarekomponenten überhaupt. Sie kann auf System-, algorithmischer, RT-, Logik- und elektrischer Ebene durchgeführt werden [34]. Bei der Simulation werden die funktionalen Eigenschaften einer Schaltung gegenüber ihrer Spezifikation getestet.

An dieser Stelle soll die Simulation auf Register-Transfer-Ebene betrachtet werden, da sie den optimalen Abstraktionsgrad für die Entwicklung eines architekturgenauen SoC bietet, ohne die Simulation zu sehr zu verlangsamen, im Gegensatz zur Simulation auf Logik-Ebene, die sehr viel mehr Ausführungszeit benötigt.

Betrachtet man die funktionale Verifikation von Mikrokontroller-IP-Komponenten, so kommen spezielle Entwicklungsumgebungen zum Einsatz. In Farrall et al. [56] wird eine Entwicklungsumgebung für den TriCore1-Mikrokontroller von Infineon Technologies AG beschrieben. Für die Verifikation der Hardware kann der Mikrokontroller unterschiedlich konfiguriert und um zusätzliche IP-Komponenten erweitert werden. Die Verifikation geschieht mit einer Testbench, die Code in die Modelle des Speichers lädt, der dann vom Mikrokontroller ausführt wird.

Ein ähnliches Konzept wird auch bei der DesignWare-StarIP-Komponente des TriCore1-Mikrokontrollers [52] verfolgt, deren Simulationsumgebung in Abbildung 3.2 dargestellt ist. Der Prozessor ist dort in eine Testbench eingebettet, die in der Verifikationssprache VERA [40] implementiert ist. Die Testbench stellt sowohl eine Laufzeitumgebung in Form von Speicher (z.B. Modul `Lokaler Speicher`) als auch Verifikationskomponenten, die beispielsweise Bustransfers aufzeichnen, zur Verfügung. Die Software wird im *Motorola-SRecords-Format* (`.sre`) zu Beginn der Simulation von der Testbench in die Speichermodelle eingelesen und dann ausgeführt. Im unteren Teil der Abbildung sind Möglichkeiten für das Software-Debugging

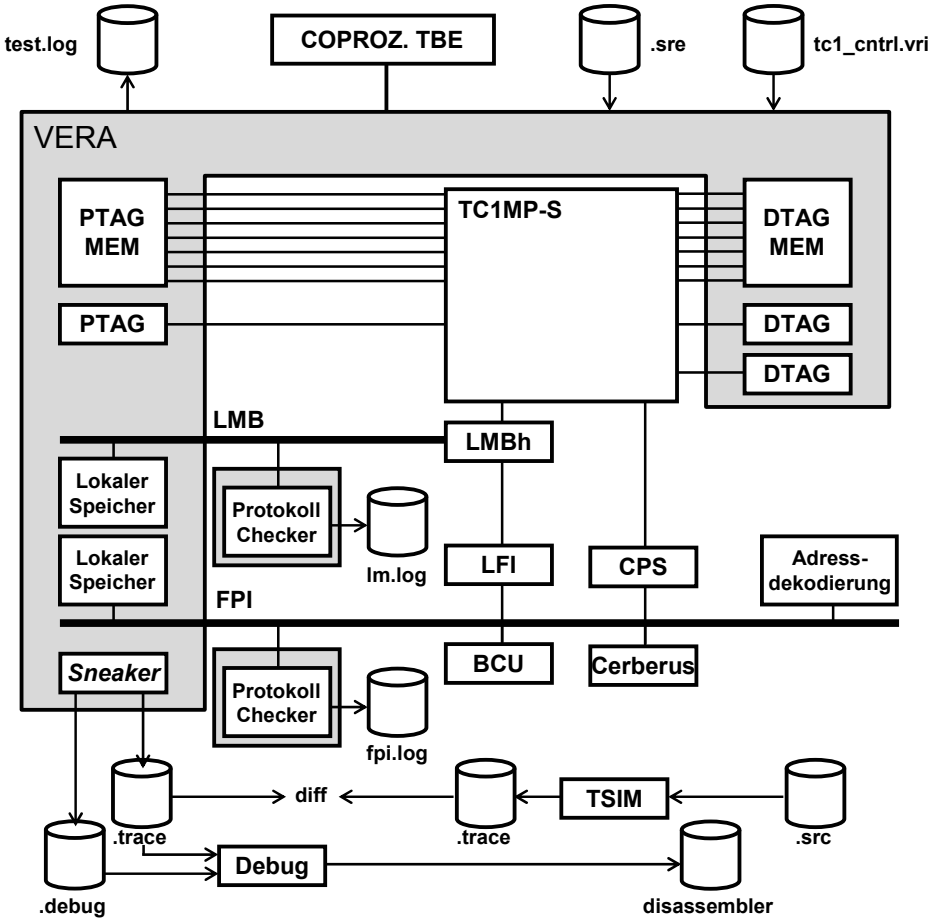


Abbildung 3.2: Entwicklungsumgebung für die TriCore1-DesignWare-StarIP-Komponente [52].

dargestellt. Diese beschränken sich zum Einen auf den Vergleich zwischen Traces des Hardware-Modells und des Befehlssatzsimulators TSIM [50] und zum Anderen auf die Disassemblierung der Traces zusammen mit Debugging-Informationen.

Beiden Ansätzen ist gemein, dass ihre Entwicklungsumgebungen für die Entwicklung und Simulation von Hardware ausgelegt sind und die Anbindung einer Software-Entwicklungsumgebung nicht oder nur sehr rudimentär vorgesehen ist. Das Beispiel der Synopsys-DesignWare-Komponente des TriCore1-IP-Kerns zeigt auch eine der Schwierigkeiten der simulationsbasierten Systementwicklung. Die Sprachen für Simulationsumgebung und Entwurf können sich unterscheiden, wodurch verschiedene Entwurfswerkzeuge kombiniert werden müssen. Ein weiterer Nachteil besteht darin, dass die Simulationsumgebung oft nicht synthetisiert werden kann, was ihren Einsatz für die im Folgenden vorgestellten Hardware-Emulatoren einschränkt. Darüber hinaus dauert die Simulation der 70 kleinen Assemblerprogramme, die mit der

DesignWare-Komponente mitgeliefert werden, auf einer heute verfügbaren Workstation bis zu zehn Stunden.

3.2.2 Kombination von BSS mit HDL-Simulator

Die in Abschnitt 3.1.1 vorgestellten BSS beschleunigen die Software-Entwicklung gegenüber einem Mikrokontroller-Modell, welches in einem HDL-Simulator simuliert wird, erheblich. Durch eine Kombination eines BSS mit einem HDL-Simulator kann das Zusammenspiel zwischen BSS und zusätzlichen Hardware-Modulen genauer untersucht werden [99]. In Abbildung 3.3 ist das Prinzip dieser Entwicklungsmethodik dargestellt. Ein neuerer Vertreter dieser Co-Verifikationsmethode ist das Seamless-Werkzeug CVE™ von Mentor [69].

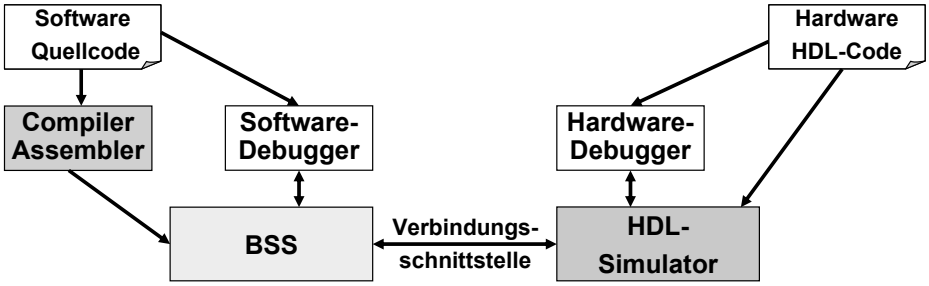


Abbildung 3.3: Entwicklung eingebetteter Hardware/Software-Systeme durch Kombination aus BSS und HDL-Simulator [99].

Die Kombination des BSS mit einem HDL-Simulator hat den Nachteil, dass der Geschwindigkeitsvorteil des BSS durch die Simulation der Hardware zunichte gemacht wird. Der Simulator kann nur ca. 10 Befehle pro Sekunde ausführen. Deshalb sollte die Methodik nur für besonders kritische Module angewendet werden [67]. Darüber hinaus ist die Kopplung und Synchronisation beider Werkzeuge schwierig, da der HDL-Simulator unter Umständen mehrere Zyklen simulieren muss, während der BSS einen Befehl in einem Schritt ausführt. Dies hat dazu geführt, dass die Kombination aus BSS und HDL-Simulator sich nie richtig am Markt durchgesetzt hat. So wurde beispielsweise das Werkzeug *Eagle* von Synopsys nicht mehr weiterentwickelt [36].

Zur Beschleunigung der Simulation kann der HDL-Code kompiliert werden. Bei SystemC™ wird der Entwurf zunächst mit C++ beschrieben und dann zusammen mit einer Simulationsbibliothek übersetzt. Dies führt zu Performanzsteigerung gegenüber der herkömmlichen Simulation [91]. Der so beschleunigte Entwurf kann dann mit einem BSS kombiniert werden. In [80] wurde um einen, mit einer pro-

prietären Beschreibungssprache erzeugten BSS ein SystemC-Wrapper geschrieben, der mit einem SystemC-Modell zusätzlicher Hardware-Komponenten kommunizieren kann und eine Performanz von 75k-90kZyklen/Sek. aufweist.

3.2.3 Emulation von Hardware-Komponenten

Aufgund der hohen Integrationsdichte moderner Mikrochips ist es möglich, zunehmend komplexere Schaltungen zu implementieren. Aus diesem Grund werden die Entwürfe häufig in kleinere Module aufgeteilt und getrennt voneinander simuliert. Eine Simulation des Gesamtssytstems, bei der beispielsweise die Zusammenarbeit eines Mikrokontrollers mit Peripheriekomponenten getestet werden soll, ist dagegen sehr zeitaufwändig.

Deshalb wurde nach Möglichkeiten zur Beschleunigung der Simulation gesucht. Die heute gebräuchlichste Methode ist die Emulation der Hardware, die beispielsweise mit FPGAs durchgeführt werden kann [41]. Die funktionale Verifikation des Systems kann damit um einen Faktor 100-1000 gegenüber der Hardware-Simulation beschleunigt werden [63]. Bei den Prototyping-Plattformen kann dabei zwischen großen Hardware-Emulations-Systemen und Prototyping-Boards unterschieden werden.

Hardware-Emulations-Systeme

Unter großen Hardware-Emulations-Systemen sollen im Folgenden Systeme verstanden werden, die umfangreiche Hardware-Ressourcen zur Verfügung stellen. Die Systeme von Aptix [5] oder Cadence [20] fallen beispielsweise in diese Kategorie. Aufgrund der sehr skalierbaren und generischen Architektur der Systeme müssen diese eine sehr aufwändige Verbindungstopologie bereitstellen, um mehrere Boards miteinander kombinieren zu können.

Für die Synthese ist dann eine komplexe und teure Werkzeugumgebung notwendig. Dazu gehört beispielsweise Partitionierungs-Software, die den Entwurf auf mehrere FPGAs aufteilen kann. In [60] wird eine Entwicklungsumgebung auf Basis des Aptix System Explorer™ beschrieben, auf den ein großer SoC-Entwurf abgebildet wird. Die Prototyping-Plattform ist in Abbildung 3.4 dargestellt.

In der Arbeit werden für das Prototyping von Mikrokontrollern von ST Microelectronics zwischen 12 und 14 FPGAs verwendet, die durch eine aufwändige Verbindungsstruktur miteinander verbunden sind. Für die Abbildung der Prozessoren mussten darüber hinaus spezielle Software-Werkzeuge des Emulations-Systems, wie beispielsweise das *Aptix Multiplexing Tool* für E/A-Anschluss-Multiplexing und das *Aptix Logic AggreGater Tool* für die Partitionierung, herangezogen werden.

Abbildung 3.4: Hardware-Emulations-System von Aptix [5].

Die Verifikation des Entwurfs wird mit einer synthetisierbaren Hardware-Testbench durchgeführt. Das macht die Systementwicklung sehr hardware-lastig und verhindert ein effizientes Software-Debugging. Darüber hinaus wird die Prototyping-Plattform durch die verwendete Hardware und Software sehr teuer und stellt damit für den Entwickler eine Hürde für den Einstieg in diese Technologie dar.

Emulations-Boards

Im Gegensatz zu den im vorangegangenen Abschnitt beschriebenen Hardware-Emulations-Systemen gibt es auf dem Markt auch relativ günstige Prototyping-Boards [109, 74]. Diese werden hauptsächlich für die Entwicklung einzelner Hardware-Komponenten sowie für die Beschleunigung einzelner Teile eines Gesamtsystems eingesetzt [79].

Da bei diesen Systemen, genau wie bei den Hardware-Emulations-Systemen, ein Entwurf unter Umständen auf mehrere FPGAs aufgeteilt werden muss, werden auch hier spezielle Boards mit speziellen Verbindungsnetzwerken entwickelt [63] oder es werden Anschluss-Multiplexing-Verfahren im FPGA selbst implementiert [9]. Dies macht die Integration der Hardware-Komponenten aufwändiger, fehleranfälliger, be-

nötigt zusätzliche FPGA-Ressourcen und verlangsamt die Emulationsgeschwindigkeit.

Ein Problem der Verifikation von Hardware-Komponenten mit Emulations-Boards ist die Anbindung der Testbench. Diese wird oft mit in den Prototyp synthetisiert [60], oder in einem Simulator betrieben [79]. Da für Testbenches allerdings zum Einen beim Entwurf objektorientierte Methoden verwendet werden und zum Anderen spezielle Verifikationssprachen wie beispielsweise *e* oder VERA zum Einsatz kommen, ist deren Synthese mit Problemen behaftet. Hier gibt es bereits erste Methoden [62] und Werkzeuge für deren Synthese [101].

Allen Ansätzen ist gemein, dass sie sich hauptsächlich mit der Verifikation von einzelnen Hardware-Komponenten beschäftigen. In [79] wird die Verifikation eines kleinen RISC-Prozessors teilweise auf einem Prototyping-Board und teilweise in einem HDL-Simulator durchgeführt.

3.2.4 Kombination von BSS mit Hardware-Emulator

Ein eher auf die Software-Entwicklung ausgelegter Ansatz ist dagegen die Kombination eines BSS mit Emulations-Systemen [36, 79, 99]. Abbildung 3.5 zeigt die Architektur einer solchen Entwicklungsplattform.

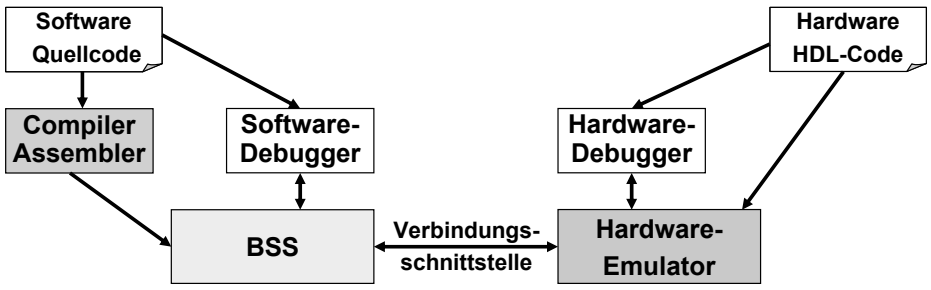


Abbildung 3.5: Entwicklung eingebetteter Hardware/Software-Systeme durch Kombination aus BSS und Hardware-Emulator [99].

Auch bei diesen Lösungen können an der Schnittstelle von BSS und Emulator Synchronisationsprobleme auftreten. Insbesondere wird hier der BSS unter Umständen langsamer als der Emulator sein. Eine Lösung wäre dann, die Taktrate des Emulators entsprechend herunterzusetzen. Neuere Ansätze versuchen dagegen die Ausführungsgeschwindigkeit des BSS zu erhöhen [90], um eine bessere Performanz des Gesamtsystems zu erreichen.

3.2.5 Kombination aus Prozessor-ASIC und FPGA

Eine Möglichkeit, die Geschwindigkeitsnachteile eines BSS auszugleichen, ist die Kombination eines Prozessor-ASIC mit rekonfigurierbarer Hardware [105, 72, 57, 46]. In [57] wird beispielsweise ein *microSPARC-IIep* Prozessor-Chip mit Xilinx FPGAs auf einem Board kombiniert und für die Entwicklung eingebetteter Hardware/Software-Systeme verwendet.

Wird für das Prototyping *ein* Entwicklungsboard verwendet, so ist man in der Auswahl des Mikrokontrollers und dessen interner Struktur festgelegt. Soll ein anderer Mikrokontroller evaluiert werden, muss das gesamte Board neu entwickelt werden. Ein anderer Ansatz sieht vor, dass zwei Boards miteinander kombiniert werden, von dem das eine Board mit einem Mikrokontroller und das andere Board mit einem rekonfigurierbaren Chip ausgestattet ist. Abbildung 3.6 zeigt das Spyder-System [105, 72]. Bei dieser Lösung kann der Mikrokontroller gewechselt werden, ohne dass das FPGA-Board verändert werden muss. Die Ankopplung an den Peripheriebus des Mikrokontrollers wird über programmierbare Logikbausteine realisiert, die sich auf dem Prozessor-Board befinden. Allerdings muss auch bei diesem Ansatz das Prozessor-Board neu entwickelt und produziert werden, wenn ein anderer Mikrokontroller evaluiert werden soll.

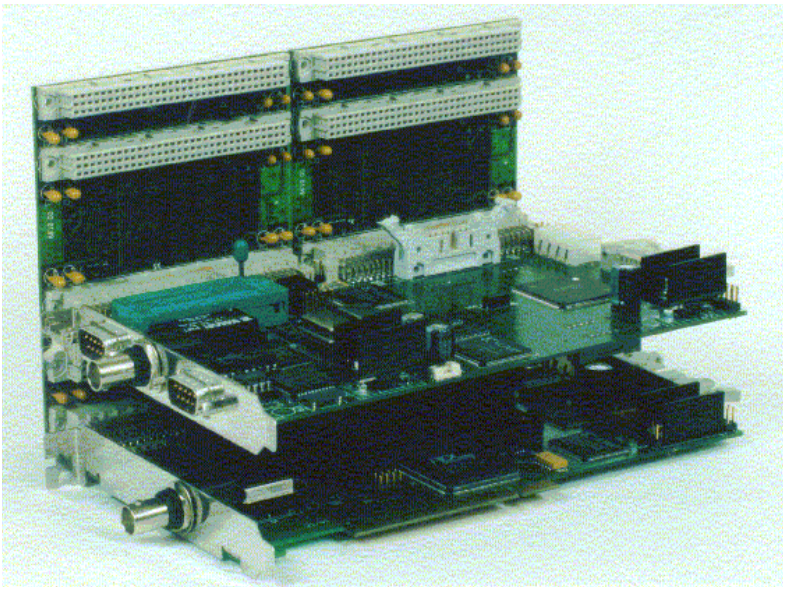


Abbildung 3.6: Spyder-System mit Hitachi-SH3-Mikrokontroller-Board (oben) und Xilinx Virtex-FPGA-Board.

3.2.6 Integration von Prozessoren in FPGAs

Eine andere Möglichkeit, einen Prozessor mit rekonfigurierbarer Hardware zu kombinieren, besteht darin, den Prozessor mit in das FPGA zu integrieren. Diese Lösungen werden hauptsächlich für die Produktentwicklung verwendet, d.h. das FPGA mit dem Prozessor wird sich später auch im Produkt wiederfinden. Prinzipiell sind diese Entwicklungsumgebungen aber auch für das Prototyping geeignet.

Der Prozessor kann dabei als Hard- oder als Softmakro innerhalb des FPGAs realisiert werden. In Xilinx Virtex-II-Pro-FPGAs sind beispielsweise bis zu zwei *Embedded PowerPC™ 405* Mikrokontroller von IBM integriert. Abbildung 3.7 zeigt die Architektur eines Xilinx Virtex-II-Pro-FPGA mit den beiden integrierten Prozessoren.

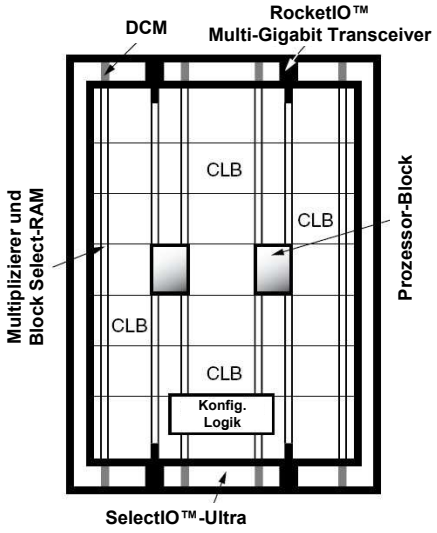


Abbildung 3.7: Architektur des Virtex-II-Pro-FPGAs von Xilinx.

In [107] wird ein *Altera Excalibur* FPGA [4] für die Entwicklung einer Antriebssteuerung mit FlexRay beschrieben. Dort ist ein ARM-Mikrokontroller mit einem im FPGA implementierten FlexRay-Hardware-Modul gekoppelt. Das System wurde dort allerdings speziell für FlexRay entwickelt und ist deshalb für das Prototyping beliebiger Hardware/Software-Systeme nicht geeignet.

Aufgrund der Auslegung des Prozessors als Hardmakro sind bei den bisher beschriebenen Lösungen die Prozessoren nicht konfigurierbar. Xilinx bietet zwei Prozessoren als Softmakros an, die speziell für FPGAs entwickelt wurden. Es sind dies der 8-Bit-Mikrokontroller PicoBlaze™ [112] und der 32-Bit-Mikrokontroller MicroBlaze™ [111].

Bei dieser Lösung besteht das Problem, dass die Prozessoren speziell für FPGAs

entwickelt wurden. Für die Integration in einem ASIC muss der Code unter Umständen für ASICs angepasst werden, wenn beispielsweise spezielle Komponenten des FPGAs in den Entwurf mit einbezogen wurden. In [64] wurde der Motorola M-Core Prozessor auf diese Art zunächst für FPGAs neu implementiert und dann wieder als ASIC gefertigt. Ein anderes Beispiel ist der LEON2-Prozessor, der die SPARC V8 Architektur für FPGAs neu implementiert [24]. Weitere als Softmakros verfügbare FPGA-Prozessor-Kerne finden sich unter [73].

3.2.7 Formale Verifikation

Aufgrund der immer komplexer werdenden Systeme ist ein erschöpfender funktionaler Test eines Systems heute kaum mehr möglich. Beispielsweise würden zur vollständigen Verifikation des TriCore2-Mikroprozessors von Infineon mit ca. 900.000 Gattern durch Simulation 20 Personenjahre benötigt [12].

Um dennoch Aussagen über die Korrektheit eines Entwurfs machen zu können, werden in den kommenden Jahren formale Methoden an Bedeutung gewinnen. In dem Forschungsprojekt *Verisoft* [102] sollen formale Methoden zur Verifikation des TriCore2 untersucht werden, die den Validierungsaufwand auf 1,5 Personenjahre reduzieren [12].

Bei der formalen Verifikation wird die Implementierung einer Schaltung mit formalen Methoden gegen die Spezifikation der Schaltung überprüft [59]. Ein guter Überblick über die formale Verifikation findet sich in [39].

4 Bewertung des Stands der Technik

In diesem Kapitel sollen der Stand der Technik hinsichtlich der beiden mit dieser Arbeit verfolgten Ziele bewertet werden. In Abschnitt 4.1 werden Verfahren zur Software-Entwicklung in frühen Entwurfsphasen und in Abschnitt 4.2 Verfahren zur parallelen Entwicklung von Hardware-Komponenten und hardware-naher Software beurteilt. Abschnitt 4.3 gibt dann quantitative Ziele vor, die mit der in dieser Arbeit vorgestellten Entwurfsmethodik erreicht werden sollen.

4.1 Bewertung der Entwurfsmethoden für die Software-Entwicklung

Der Software wird bei der Entwicklung eingebetteter Systeme in Zukunft eine wichtige Rolle zukommen [93]. Aufgrund des hohen Anteils der Software wird diese den Zeitpunkt der Fertigstellung eines Produktes bestimmen und es sollte daher möglichst früh mit der Entwicklung der Software begonnen werden.

BSS sind ein klassisches Beispiel für die Software-Entwicklung mit einem Prozessor-Modell. Diese sind oft Bestandteil der Entwicklungsumgebung für einen Debugger deren Preis sich im vier bis fünfstelligen Eurobereich bewegt. Allerdings stellt die Genauigkeit der Rechnerarchitektur beim BSS ein Problem dar. Wenn spezielle Merkmale moderner Prozessoren wie Caches, Pipelining, Multithreading, Sprungvorhersage, Unterbrechungsbehandlung sowie die Anbindung zusätzlicher Hardware-Komponenten modelliert werden müssen, um ein architekturgenaues Modell des Prozessors zu erhalten, hat dies sehr lange Simulationszeiten zur Folge. Proprietäre schnellere Lösungen benötigen darüber hinaus eine spezielle Anbindung des Software-Debuggers, die zunächst implementiert werden muss und mit Fehlern behaftet sein kann. Auch müsste ein architekturgenaues Modell des BSS unter Umständen erst implementiert werden.

Eine weitere kostengünstige Alternative für die Software-Entwicklung stellt die Embedded Java-Technologie dar. Sie ist in der Referenzimplementierung von Sun kostenlos zu bekommen und es fallen nur Kosten für den Entwicklungsrechner an. Die Java-Technologie konnte sich im Bereich eingebetteter Systeme allerdings nie richtig durchsetzen, da sie hier viele Probleme mit sich bringt. Dies zeigt beispielsweise auch der Rückzug von IBM aus diesem Marktsegment, in dem ursprünglich die *VisualAge Micro Edition* als Entwicklungsumgebung angeboten wurde [45]. Vor al-

lem die Echtzeitfähigkeit und Einbindung spezieller Hardware-Komponenten ist mit Problemen verbunden. Diese müssten beispielsweise als Simulationsmodelle vorliegen, wenn die Software nicht direkt auf der Zielhardware entwickelt wird. Darüber hinaus besitzt Java gegenüber nativ kompilierten Programmen oft einen Performanznachteil, der sich in eingebetteten Systemen, in denen gegenüber dem Desktop-Bereich in der Regel leistungsärmere Prozessoren eingesetzt werden, noch verstärkt.

Werden dagegen Evaluierungsboards für die Software-Entwicklung verwendet, so steht ein schnelles und architekturgenaues Modell der Hardware, welches trotzdem sehr kostengünstig ist, zur Verfügung. Aufgrund der vorgegebenen Architektur der Evaluierungsboards mit Peripheriekomponenten für bestimmte Anwendungsgebiete ist eine Anwendungsentwicklung nur für diese Peripheriekomponenten möglich. Die Integration zusätzlicher Hardware-Komponenten zur Erweiterung des Funktionsumfangs ist bei Evaluierungsboards nicht möglich. Dies gilt auch für die Untersuchung der Auswirkungen unterschiedlicher Cachegrößen auf das Verhalten der Anwendung.

Abbildung 4.1 zeigt eine Einordnung der Performanz bei der Software-Entwicklung für unterschiedliche Entwicklungsumgebungen. Aus der Abbildung ist ersichtlich, dass die Emulation in Bezug auf die Performanz eine Alternative zu Evaluierungsboards darstellt und gleichzeitig eine Implementierung unterschiedlicher Rechnerarchitekturen zulässt. Der Nachteil dieser Lösung liegt allerdings in den hohen Kosten für die Hardware-Emulations-Systeme, die im mittleren sechstelligen Eurobereich liegen können.

4.2 Bewertung der Entwurfsmethoden für die Entwicklung komplexer Hardware-Komponenten

Für die Entwicklung komplexer Hardware-Komponenten und der dazugehörigen hardware-nahen Software spielen unterschiedliche Faktoren eine Rolle. Zum Einen muss ein Prozessor-Modell zur Verfügung stehen, auf dem die Software implementiert werden kann und zum Anderen muss sowohl das Debugging der zu entwickelnden Hardware-Komponente und der dazugehörigen Software gewährleistet sein.

Simulation, Emulation und formale Verifikation sind auf die Entwicklung der Hardware ausgerichtet und berücksichtigen eine parallele Software-Entwicklung nur ungenügend. Dafür garantieren diese Methoden, allen voran die Simulation auf RT-Ebene, detaillierte Einblicke in die internen Abläufe der Hardware. Allerdings sind die Simulation und formale Verifikation für sehr große und komplexe SoCs aufgrund ihrer schlechten Performanz ungeeignet und können darüber hinaus nicht in ein Zielsystem integriert werden.

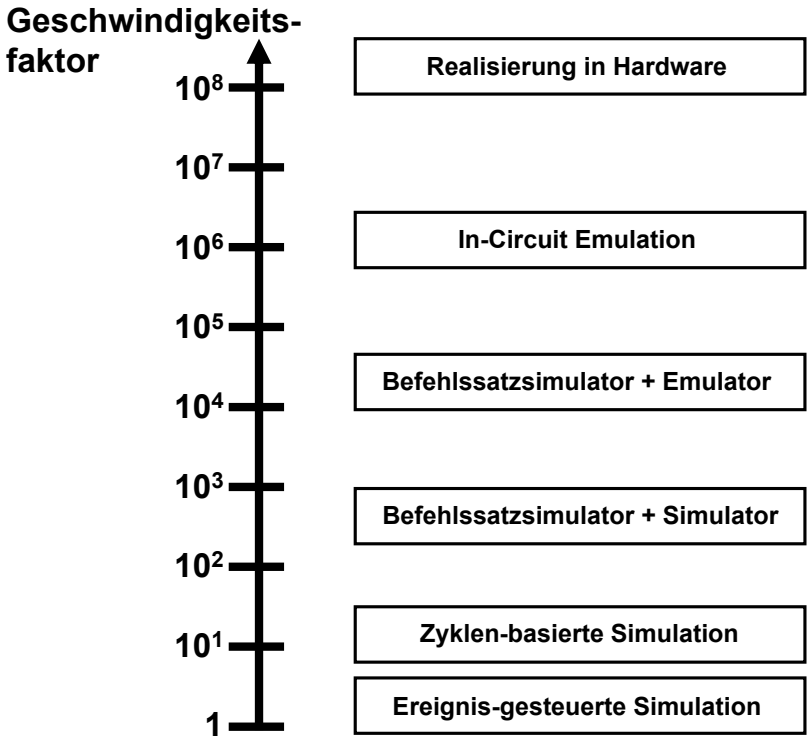


Abbildung 4.1: Laufzeitvergleich verschiedener Modelle für die Entwicklung eingebetteter Software [99].

Wie aus Abbildung 4.1 ersichtlich ist, ist die Simulation um einen Faktor 10^5 langsamer als die Emulation einer Hardware-Komponente. Aus diesem Grund werden in Zukunft emulationsbasierte Methoden an Bedeutung gewinnen. Insbesondere wird innerhalb des Entwicklungsprozesses eine „FPGA-zentrierte Methodik für Debugging und Verifikation von eingebetteten Systemen benutzt werden“ [11]. Die dort gebräuchlichen Entwicklungsumgebungen basieren heute allerdings entweder auf sehr teurer Hardware [60] oder zielen nur auf eine Beschleunigung der Hardware-Verifikation ab [101].

Die Co-Simulation mit einem BSS ist dagegen auch für die Hardware/Software-Co-Verifikation geeignet. Sie ist allerdings mit Problemen behaftet, wenn das zeitliche Verhalten eines Systems verifiziert werden soll [108]. Dies ist darauf zurückzuführen, dass die Werkzeuge für die Hardware- und Software-Simulation eine unterschiedliche Granularität besitzen. Beispielsweise kann ein einziger Schritt eines BSS auf Befehlsebene sehr viele Schritte in einem HDL-Simulator in Anspruch nehmen, in dem ein Ein-/Ausgabe-Gerät für die CPU simuliert wird. Für die Beschleunigung der HDL-Simulation der Hardware-Komponente kann Emulation verwendet werden. Hier ist dann aber unter Umständen der BSS der Teil, der das Gesamtsystem verlang-

samt. In Schnerr et al. [89] wird eine Methode vorgestellt, wie der BSS beschleunigt werden kann. Insgesamt kann es aber zu Synchronisationsproblemen zwischen BSS und Emulator kommen. Darüber hinaus sind interne Abläufe des Prozessors nicht sichtbar und komplexe Rechnerarchitekturen sind nur schwierig und zeitaufwändig zu modellieren.

Für ein architekturgenaues Prototyping muss ein möglichst genaues Modell der Hardware zur Verfügung stehen. Eine detaillierte Analyse ist notwendig, damit nicht von vorneherein ein unter- oder überdimensioniertes System entwickelt wird. Genügt das SoC später nicht den Ansprüchen der Anwendung, so ist es unbrauchbar, ist es überdimensioniert, so wird das Produkt für den Verbraucher zu teuer. Eine genaue und dennoch schnelle Analyse kann zwar mit Prozessor-ASICs oder mit in FPGAs integrierten Prozessoren durchgeführt werden. Für beide Technologien steht dann allerdings die Art und Ausprägung der CPU fest und kann nur schwer durch andere Architekturen ausgetauscht werden.

4.3 Zielsetzung einer neuen Entwurfsmethodik für eingebettete Hardware/Software-Systeme

In Tabelle 4.1 sind die Ergebnisse der in Abschnitt 4.1 getroffen Bewertung nochmals quantitativ und qualitativ zusammengefasst. Die Werte für die Laufzeiten sind der Abbildung 4.1 entnommen und auf den BSS normiert worden. Die Laufzeit von Embedded Java wird mit eins angegeben, da auch dort wie beim BSS der Befehlssatz interpretiert wird. Durch JIT-Compilierung kann diese zwar gesteigert werden, das System weist dann allerdings keine Echtzeiteigenschaften mehr auf. Die Laufzeiten für SystemC werden als „nicht bekannt“ angegeben, da diese Technologie in [99] nicht berücksichtigt ist und deshalb nicht in Relation zu den anderen Verfahren gesetzt werden kann. In [71, 80] finden sich Angaben zu Laufzeiten für SystemC. Insgesamt werden diese zwischen BSS und Hardware-Emulations-Systemen liegen.

Die Entwurfsverfahren zeichnen sich durch eine unterschiedliche Genauigkeit der Modellierung aus. Die auf Simulation ausgelegten Verfahren BSS, Embedded Java und SystemC-TLM arbeiten alle auf einer höheren Abstraktionsebene. Werden bei diesen Verfahren detailliertere Modelle verwendet, so vermindert sich die angegebene Performanz. Evaluierungsboards und Hardware-Emulations-Systeme bilden das SoC dagegen architekturgenau nach.

Tabelle 4.1 enthält auch eine Bewertung für die Flexibilität der Entwicklungsplattform im Hinblick auf die Integration neuer Hardware-Komponenten, da diese für die zweite Zielsetzung dieser Arbeit eine herausragende Rolle spielt und die Vorteile der Evaluierungsboards relativiert. Diese sind in ihrer Hardware-Architektur fest

Tabelle 4.1: Bewertung der Entwurfsverfahren für eingebettete Software.

	BSS	Embedded Java	SystemC-TLM	Evaluierungs-boards	Hardware-Emulations-Systeme	**Ziel**
Geschwindigkeitsfaktor	1	$\sim$1	nicht bekannt	10^4	10^2	50
Genauigkeit	Zyklen	Befehl	Transaktion	Arch.	Arch.	Arch.
Flexibilität	nein	nein	ja	nein	ja	ja
Kosten	ca. Euro 2.000	ca. Euro 2.000	ca. Euro 2.000	ca. Euro 2.400	ca. Euro 600.000	ca. Euro $\leq$ 20.000

und lassen sich nicht um zusätzliche Hardware-Komponenten erweitern. Bei BSS und Embedded Java ist dies nur schwer möglich.

Für die Kosten der einzelnen Entwurfsverfahren ist nur der Preis für die Hardware und etwaige dafür notwendige spezielle Entwicklungs-Software angegeben. Beim BSS, Embedded Java und SystemC-TLM entsprechen die Kosten dem Preis für einen Entwicklungs-PC. Beim Evaluierungsboard kommen zusätzlich Kosten für das Board hinzu [47]. Die Kosten für ein Hardware-Emulations-System sind aus [21] entnommen. Dort sind neben den Kosten für die Hardware noch Kosten für Partitionierungs- und spezielle Synthese-Software enthalten. Kosten für Software-Entwicklungswerkzeuge sind in der Aufstellung nicht enthalten.

Wie aus Tabelle 4.1 ersichtlich ist, haben die kostengünstigen Lösungen auf der einen Seite Performanzprobleme und sind für eine architekturgenaue Analyse des zu entwickelnden Gesamtsystems nicht geeignet. Evaluierungsboards und Hardware-Emulations-Systeme sind dagegen sehr performant, dafür aber entweder teuer oder in ihrer Hardware-Architektur festgelegt. Die Integration neuer Hardware-Komponenten ist mit Evaluierungsboards nicht möglich.

Ein **Ziel dieser Arbeit** ist es deshalb, eine Methodik für die Software-Entwicklung auf einem Hardware-Modell zu entwickeln, die die Performanz eines BSS um einen Faktor 50 steigert. Darüber hinaus sollen allerdings zusätzlich architekturgenaue Einblicke in das Gesamtsystem möglich sein und die Kosten sollen im Rahmen der Evaluierungsboards liegen. Die Kosteneinsparung bei der Entwicklungsumgebung führt dazu, dass der Geschwindigkeitsfaktor 10^2 der Hardware-Emulations-Systeme nicht

Tabelle 4.2: Bewertung der Entwurfsverfahren für Hardware-Komponenten und hardware-nahe Software.

	HDL-Simulation	BSS + Simulation	BSS + Emulation	Hardware-Emulations-Systeme	ASIC + FPGA	Integr. FPGA	Evaluierungsboard	**Ziel**
Geschwindigkeitsfaktor	1	10^2	10^3	10^5	10^5	10^5	10^7	10^4
Hardware-Debugging	ja	Teile	Teile	ja	Teile	Teile	Teile	ja
Software-Debugging	nein	ja	ja	nein	ja	ja	ja	ja
Flexibilität	ja	nein	nein	ja	nein	nein	nein	ja

erreicht werden kann. Der Entwickler soll dabei nicht von vorne herein auf eine Architektur der SoC-Plattform festgelegt sein, sondern neue Hardware-Komponenten integrieren können.

Tabelle 4.2 zeigt einen Überblick über bisher verfügbare Entwurfsverfahren für komplexe Hardware-Komponenten mit dazugehöriger hardware-naher Software. Die Werte für die Laufzeiten sind der Abbildung 4.1 entnommen und auf den HDL-Simulator normiert worden.

Die HDL-Simulation, die eine Beobachtung der Hardware bis auf Signalebene ermöglicht, ist nicht für das Software-Debugging ausgelegt und zeichnet sich durch sehr langsame Laufzeiten aus. Beim BSS wird dem Entwickler das Software-Debugging zwar ermöglicht, hier sind aber keine Einblicke in interne Abläufe des Prozessors möglich und auch die Laufzeiten sind gegenüber der Emulation um einen Faktor 10^2 langsamer. Bei den Kombinationen eines FPGAs mit Prozessor-ASIC oder der Integration einer CPU in das FPGA (Integr. FPGA) ist die Architektur der CPU fest und es sind keine Einblicke in interne Abläufe der CPU möglich. Wird für die Entwicklung hardware-naher Software gar ein Prototyp des SoC als Chip implementiert, dann belaufen sich die Kosten für Masken und Prototypen-Platine auf bis zu 5 Millionen Dollar [55]. Bei Evaluierungsboards steht ein solcher Chip zwar zur Verfügung, dort können interne Abläufe im Prozessorkern aber nur teilweise beobachtet werden und er kann nicht um neue Hardware-Komponenten erweitert werden.

Eine volle Flexibilität bezüglich der SoC-Plattform ist nur bei der HDL-Simulation und den Hardware-Emulations-Systemen gegeben. Bei den anderen Verfahren ist der Entwickler entweder auf einen Mikrokontroller-Typ festgelegt, oder es müssen ent-

sprechende BSS-Modelle oder Entwicklungsboards/FPGAs produziert werden, die die gewünschte Rechnerarchitektur implementieren.

Das zweite **Ziel dieser Arbeit** ist deshalb, die parallele Entwicklung von Hardware-Komponenten und hardware-abhängiger Software zu unterstützen, bei der Einblicke bis auf Signalebene *aller* Teile des SoC möglich sind. Dabei soll die Anwendungsentwicklung bis zu einem Faktor 10^4 gegenüber einer HDL-Simulation des Gesamtsystems beschleunigt und das Evaluierungssystem nach dem Beheben eines Fehlers schnell wieder für weitere Debugging-Aufgaben zur Verfügung stehen. Darüber hinaus muss aufgrund der immer wichtiger werdenden eingebetteten Software die Anbindung einer Software-Entwicklungsumgebung mit Software-Debugger zur Verfügung gestellt werden. Auch sollen die Kosten für die Entwicklungsumgebung wieder im niederen fünfstelligen Euro-Bereich liegen, weshalb Abstriche bei der Performanzsteigerung gegenüber Hardware-Emulations-Systemen in Kauf genommen werden können.

5 Konzept einer Entwicklungsumgebung für den Entwurf eingebetteter Hardware/Software-Systeme

In diesem Kapitel wird eine Entwurfsmethodik vorgestellt, mit der die in Abschnitt 4.3 abgeleiteten Ziele für die Entwicklung komplexer SoC-Entwürfe erreicht werden. Die Methode beruht auf einer rekonfigurierbaren Hardware-Plattform, die als Hardware-Evaluierungsumgebung und für die Software-Entwicklung komplexer SoC-Entwürfe genutzt werden kann. Durch die Emulation des eingebetteten Hardware/-Software-Systems auf einer rekonfigurierbaren Hardware-Plattform lässt sich einerseits die Programmausführung beschleunigen und andererseits ist die Möglichkeit zur Integration komplexer Hardware-Komponenten gegeben, für die parallel Software entwickelt werden muss.

In Abschnitt 5.1 wird zunächst ein Überblick über die Entwurfsmethodik gegeben, die in dieser Arbeit vorgestellt wird. Abschnitt 5.2 zeigt dann, wie die Entwicklung eingebetteter Software auf der Basis eines architekturgenauen Hardware-Modells eines Mikrokontroller-IP-Kerns beschleunigt werden kann. Das dort vorgestellte Modell wird in Abschnitt 5.3 dahingehend erweitert, dass der Entwurf neuer komplexer Hardware-Komponenten und Software zusammen mit einer vorgegebenen SoC-Plattform unterstützt wird. Die in diesem Kapitel vorgestellten Konzepte und Methoden werden in Abschnitt 5.4 nochmals kurz zusammengefasst.

5.1 Grundkonzept der Entwurfsmethodik

Mit der in dieser Arbeit vorgestellten Entwurfsmethodik sollen zwei Ziele bei der Entwicklung von Hardware und Software für SoCs verfolgt werden. Das erste Ziel besteht darin, die Software-Entwicklung auf einem architekturgenauen Hardware-Modell zu beschleunigen. Für die Entwicklung eingebetteter Software in einem frühen Stadium des Entwurfsprozesses spielt das Prozessor-Modell eine entscheidende Rolle. Wie in Abschnitt 3.1 aufgezeigt wurde, kommen hier entweder BSS oder Evaluierungsboards zum Einsatz. Da sich Evaluierungsboards gegenüber BSS durch eine deutlich bessere Performanz auszeichnen, ist in Abbildung 5.1 (a) die Software-Entwicklung mit einem Evaluierungsboard dargestellt.

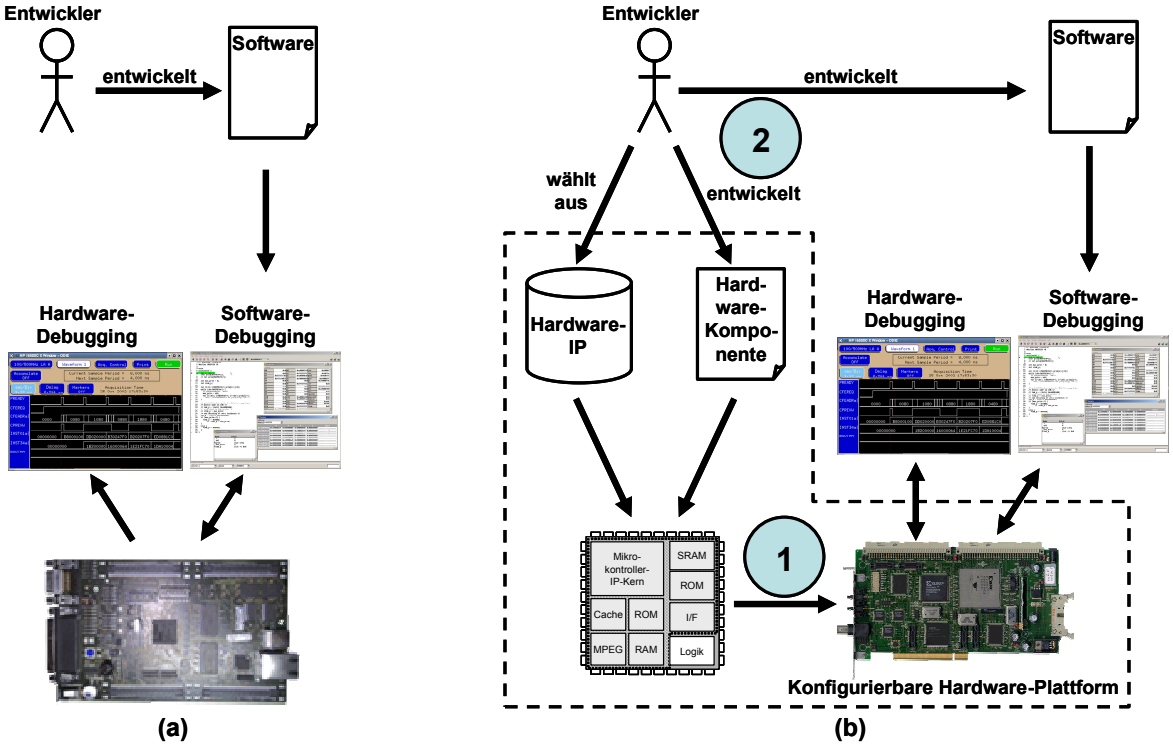


Abbildung 5.1: Klassische Entwicklungsumgebung für eingebettete Software (a) und neue konfigurierbare Entwicklungsumgebung für eingebettete Hardware/Software-Systeme (b).

Die Mikrokontroller sind auf diesen Boards als Chip integriert und deshalb in ihrer Hardware-Funktionalität auf die dort jeweils gewählte Realisierung beschränkt. Auch ist das Debugging der Hardware nur mit den Mitteln möglich, die vom Chip zur Verfügung gestellt werden. Die Plattform zeichnet sich allerdings durch sehr schnelle Ausführungszeiten und geringe Kosten aus.

Die mit dieser Arbeit vorgeschlagene Entwurfsmethodik ist in Abbildung 5.1 (b) dargestellt. Um kurze Laufzeiten zu erreichen, wird der Mikrokontroller-IP-Kern emuliert. Die Entwurfsumgebung wird um einen Zweig für die Hardware-Entwicklung erweitert, die die Kombination einer vorgegebenen SoC-Plattform mit zusätzlichen Hardware-Komponenten ermöglicht. Durch die Emulation *aller* Teile des SoC ist ein Einblick in *alle* Teile der Hardware gegeben. Um die Kosten für die Entwicklungsumgebung gering zu halten, wird ein kostengünstiges FPGA-Board verwendet.

Die beiden Aufgaben, die in dieser Arbeit gelöst werden sind in Abbildung 5.1 (b) durch die beiden Kreise gekennzeichnet. Diese Zweiteilung spiegelt sich auch in der Gliederung des Kapitels wieder. In Abschnitt 5.2 wird gezeigt, wie ein komplexes SoC auf einem Emulations-System integriert und damit die Software-Entwicklung

durchgeführt werden kann (Kreis 1). Abschnitt 5.3 zeigt dann, wie mit dem in dieser Arbeit vorgestellten Konzept eine parallele Entwicklung von Hardware und Software möglich ist. Ziel ist es, dem Entwickler eine transparente Sicht auf das zu entwickelnde System zu ermöglichen. Letzten Endes kommuniziert der Entwickler, wie von bisherigen Evaluierungsboards gewohnt, mit einem Software-Debugger, der es ihm erlaubt, Programme auf dem rekonfigurierbaren System zu entwickeln.

5.2 Beschleunigung der Anwendungsentwicklung durch Emulation

In diesem Abschnitt wird die Entwicklung eingebetteter Software auf der Basis eines emulationsbasierten Mikrokontroller-IP-Kerns näher beschrieben. Beim Einsatz eines Emulations-Systems sind verschiedene Probleme zu lösen. So muss ein Mikrokontroller-IP-Kern im FPGA integriert werden. Dieser IP-Kern liegt in der Regel in einer HDL vor und ist für die Realisierung als ASIC optimiert. Die Integration solcher ASIC-IP-Komponenten in FPGAs gestaltet sich dabei in vielerlei Hinsicht als schwierig. Weiter muss für die Software-Entwicklung die Anbindung einer Debugging-Umgebung an das Emulations-System möglich sein.

5.2.1 Prototyping von ASIC-IP-Kernen auf FPGAs

Die Probleme beim Prototyping von ASIC-IP-Kernen auf FPGAs lassen sich grob in zwei Klassen einteilen. Es sind dies Probleme, die sich aus den technologischen Unterschieden von ASICs und FPGAs ergeben und Probleme, die auf die begrenzten Ressourcen zurückzuführen sind, die FPGAs bieten. In Abbildung 5.2 ist diese Aufteilung in Problemklassen dargestellt [88].

Probleme aufgrund technologischer Unterschiede

Beim Prototyping von ASIC-IP-Kernen tritt das Problem auf, dass die Beschreibung der Module mit einem Synthesewerkzeug für FPGAs verarbeiten werden muss. Dies führt zu einem nicht unerheblichen Portierungsaufwand, der die Entwicklung des Prototyps verzögern kann. Oft müssen Syntheseskripte und Dateien mit Entwurfsbedingungen dem FPGA-Synthesewerkzeug angepasst werden. Beispielsweise wird für einen picoJava-Prozessorkern mit 46.376 LOC ein Syntheseskript mit 1.755 Zeilen verwendet. Die Datei mit den Entwurfsbedingungen für einen picoJava-Prozessorkern enthält 1.619 Zeilen [26].

Neben der Anpassung der Syntheseskripte müssen unter Umständen auch Veränderungen im Quelltext des IP-Kerns vorgenommen werden. Verschiedene Synthe-

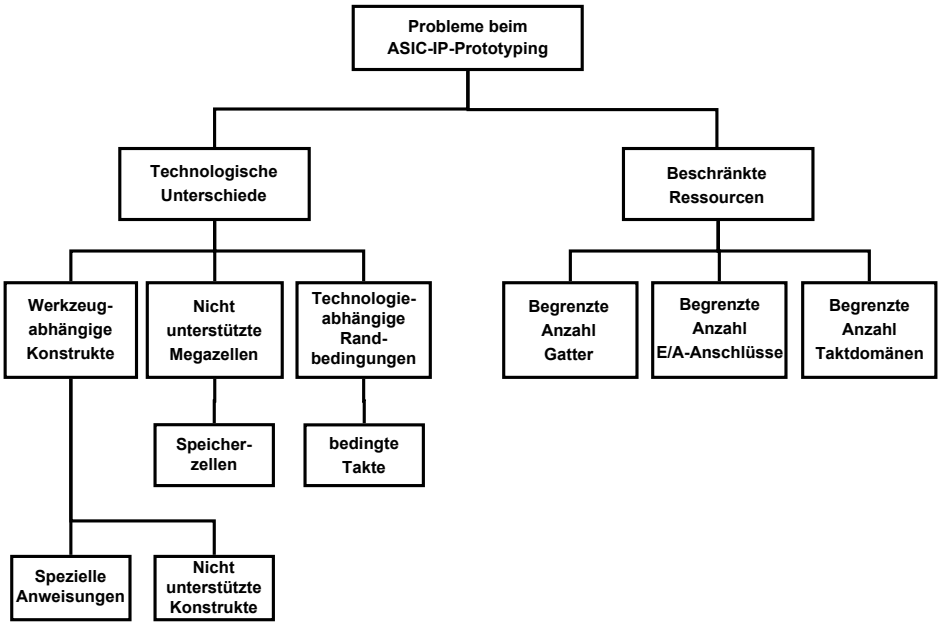


Abbildung 5.2: Problemklassifikation beim Prototyping von ASIC-IP-Kernen mit FPGAs.

sewerkzeuge können eine unterschiedliche Untermenge an Sprachkonstrukten einer Hardware-Beschreibungssprache verarbeiten. Wenn IP-Hersteller und IP-Integrator unterschiedliche Synthesewerkzeuge verwenden, so muss der IP-Integrator unter Umständen den Quelltext des IP-Kerns modifizieren, um diesen synthetisieren zu können. Bei VHDL unterstützen beispielsweise nicht alle Synthesewerkzeuge das Sprachkonstrukt `generic` in seiner vollen Definition. Das Konstrukt `generic` wird dazu verwendet, einen Entwurf parametrisierbar zu machen. Der IP-Integrator muss in diesem Fall die Parameter im `generic`-Befehl durch globale Konstanten ersetzen.

Im Bereich der Mikrokontroller werden für Peripheriekomponenten wie z.B. On-chip-Speicher Standardzellen aus Bibliotheken der Mikrokontroller-Hersteller verwendet. Diese stehen allerdings für FPGAs nicht zur Verfügung und müssen deshalb nachgebildet werden. Dabei ist darauf zu achten, dass sich die nachgebildeten Komponenten wie ihre Pendants aus dem echten Entwurf verhalten.

Eine dritte Schwierigkeit beim Prototyping von ASIC-IP-Komponenten liegt in der vorgegebenen Architektur der FPGAs begründet. Beim ASIC-Entwurf stehen dem Entwickler wesentlich mehr Freiheitsgrade bezüglich der Realisierung eines Schaltkreises zur Erbringung einer gewissen Funktionalität zur Verfügung. Ein Entwickler kann für seinen Entwurf unterschiedliche Taktdomänen definieren. Um den Leistungsverbrauch zu minimieren, werden z.B. einzelne Module zur Laufzeit abgeschaltet. Unterschiedliche Taktdomänen werden auch verwendet, um langsamere Pe-

ripheriegeräte von schnelleren Prozessoren abzukoppeln. In FPGAs stehen aufgrund ihrer generischen Struktur dagegen nur eine begrenzte Anzahl an Taktdomänen zur Verfügung.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity clk_gate is
  port (
    clk         : in std_logic;
    gate        : in std_logic;
    g_clk       : out std_logic;
);

architecture rtl of clk_gate is

begin

  g_clk <= clk and gate;

end rtl;
```

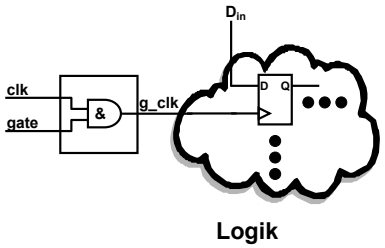


Abbildung 5.3: Abschalten des Taktes mit Hilfe eines *gate*-Signals in ASICs.

In Abbildung 5.3 ist die Beschreibung eines Moduls zur Kontrolle des Taktes in VHDL und die dazugehörige Implementierung der Schaltung für ASICs zu sehen. Für die Realisierung dieser Schaltung in einem FPGA müssen Änderungen an der Schaltung vorgenommen werden, um nicht Laufzeitprobleme mit dem Taktsignal zu bekommen. Eine Möglichkeit ist hier, die Clock-enable-Eingänge der Flip-Flops im FPGA zu verwenden. Eine mögliche Realisierung ist in Abbildung 5.4 in VHDL und als Implementierung zu sehen.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity clk_enable is
  port (
    clk         : in std_logic;
    gate        : in std_logic;
    Din         : in std_logic;
    Q           : out std_logic;
);

architecture rtl of clk_gate is

begin

  ce_pr : process(clk, gate, Din, Q)
  begin
    if (clk'event and clk = '1') then
      if (gate = '1') then
        Q <= Din;
      else
        Q <= Q;
      end if;
    end if;
  end process;

end rtl;
```

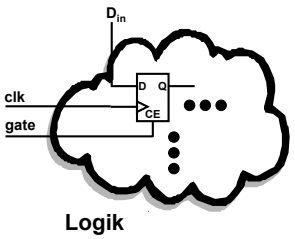


Abbildung 5.4: Umwandlung eines bedingten Taktes für FPGAs.

Da bei komplexen SoC-Entwürfen leicht einige hundert dieser Taktdomänen vorhanden sein können und der Takt bis tief in die Modulstruktur des SoC verteilt wird,

ist eine Transformation des VHDL-Quelltextes von Hand nicht mehr durchführbar. Hierfür ist dann Werkzeugunterstützung notwendig [98]. Eine andere Möglichkeit ist die Abbildung der Taktdomänen der ASIC-IP auf die Taktdomänen des FPGAs. Auf diese Lösung wird weiter unten noch näher eingegangen.

Probleme aufgrund eingeschränkter Ressourcen in FPGAs

Da es sich bei FPGAs um Hardware-Bausteine handelt, die prinzipiell jede beliebige Schaltungsfunktion realisieren können, verfügen sie über eine generische Architektur. Dies hat zur Folge, dass sie bezüglich ihrer Gatterkapazitäten immer hinter ASICs zurück bleiben werden. In Abbildung 5.5 ist diese *Kapazitätslücke* exemplarisch für Xilinx FPGAs dargestellt. Beispielsweise bieten heute verfügbare FPGAs von Xilinx dem Entwickler bis zu 8 Millionen Gatter an, die allerdings aus Gründen der Platzierung, Verdrahtung und der nicht optimalen Integration der Funktionen auf LUTs nicht vollständig für das Prototyping genutzt werden können. Auf ASIC-Gatter bezogen, müssen die angegebenen Gatterzahlen um den Faktor 4-5 geteilt werden [106].

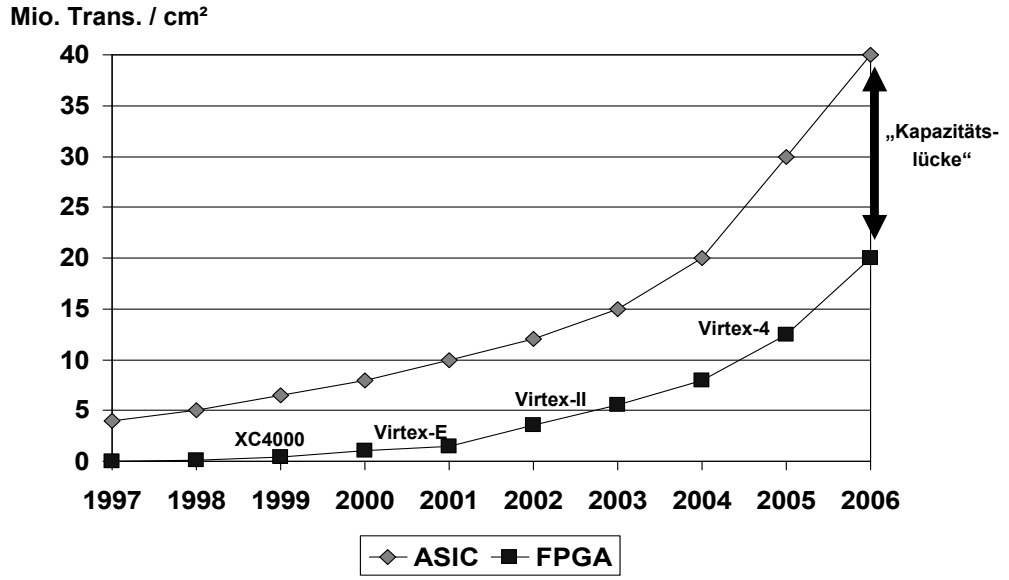


Abbildung 5.5: Entwicklung der Transistorkapazitäten von ASICs [92] und FPGAs [3].

Um dieses Problem zu lösen, werden bei Hardware-Emulations-Systemen mehrere FPGAs kombiniert. Da auch die Verbindungen der FPGAs untereinander durch spezielle Hardware optimiert wird, stehen für den Prototyp ausreichend Ressourcen zur Verfügung. Oft wird hier für jedes Modul des SoC ein eigenes FPGA verwen-

det. Durch die großen Hardware-Ressourcen und die Notwendigkeit das SoC mittels aufwändiger Partitionierungs-Software zu partitionieren, verursacht das Emulations-System sehr hohe Kosten.

Ein weiteres Ressourcenproblem von FPGAs sind die Anschlüsse des Chips. Da es wegen der eingeschränkten Anzahl an Gattern oft nicht möglich ist, das gesamte SoC auf einem FPGA unterzubringen, muss der Entwurf partitioniert werden. Da auf RT-Ebene die Module untereinander unter Umständen über sehr viele Verbindungen kommunizieren, können die Anschlüsse nicht ausreichen. Lösungen hierfür sehen ein Zeit-Multiplexing der Signale entweder innerhalb des FPGAs oder auf der Verbindungsplatine selbst vor.

Zusammenfassend lässt sich sagen, dass beim Einsatz kostengünstiger FPGA-basierter Emulations-Systeme das Ressourcenproblem gelöst werden muss. Im Folgenden wird hierfür eine Methodik sowie die Architektur einer Hardware-Entwicklungsplattform vorgestellt, die dieses Problem löst.

5.2.2 Methodik zur Lösung des Ressourcenproblems

Wie aus dem vorangegangenen Abschnitt deutlich wurde, stellt das Problem der begrenzten Ressourcen eine ernsthafte Hürde beim Prototyping von ASIC-IP-Kernen auf FPGAs dar. Die bisher vorgeschlagenen Methoden sind in Abbildung 5.6 bezüglich des Geschwindigkeitsgewinns, der Komplexität der Verbindungen und der Kosten qualitativ bewertet.

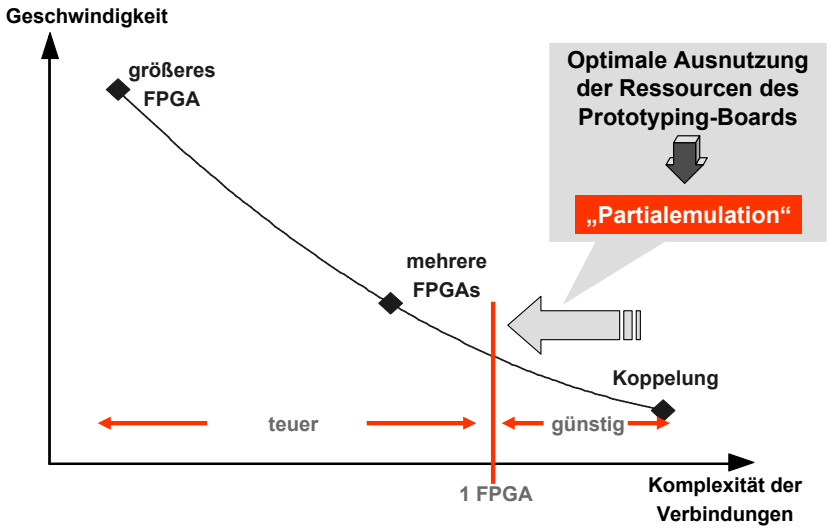


Abbildung 5.6: Kosteneinsparung bei der Prototyping-Plattform durch Partialemulation.

Die Kurve ist in zwei Bereiche unterteilt. Die Trennlinie wird durch die maximale Ausführungsgeschwindigkeit festgesetzt, die sich mit einem kostengünstigen FPGA-basierten Emulations-Board erreichen lässt. Die Laufzeit der Software auf der jeweiligen Prototyping-Plattform nimmt ab, da sich die Komplexität der Verbindungen und damit die Laufzeit der Signale zwischen den einzelnen Modulen erhöht, wenn der Entwurf partitioniert werden muss. Bei der Verwendung mehrerer FPGAs verringert sich die Geschwindigkeit aufgrund der Signallaufzeiten zwischen den FPGAs. Bei der Kopplung eines FPGAs mit einem HDL-Simulator wird Zeit für die Kommunikation zwischen beiden Teilen benötigt und der Simulator bremst die Emulation auf dem FPGA.

Außerdem zeichnen sich die Lösungen auf der linken Seite der Trennlinie dadurch aus, dass sie sehr teuer werden können, da hier spezielle Verbindungsnetzwerke zwischen den FPGAs verwendet werden und darüber hinaus spezielle Software-Werkzeuge für die Abbildung des Entwurfs auf die Prototyping-Plattform benötigt wird. Auf der rechten Seite der Trennlinie stehen dagegen kostengünstige Prototyping-Systeme.

Die Idee der *Partialemulation* ist es, durch geschickte Ausnutzung *aller* Ressourcen, die von einem Prototyping-System zur Verfügung gestellt werden, so nahe wie möglich an die Trennlinie zu den teuren Prototyping-Lösungen zu gelangen, ohne diese zu überschreiten [87]. Dadurch kann die Ausführungsgeschwindigkeit der Software gesteigert werden, ohne die Kosten für die Prototyping-Umgebung zu erhöhen.

Bei der Partialemulation wird die Integration von SoCs auf FPGA-basierte Prototyping-Boards als Abbildungsprozess definiert, welcher die Komponenten des SoC auf Ressourcen des Prototyping-Systems abbildet. Die Ressourcen lassen sich dabei in die Teilmenge der Ressourcen des FPGAs und die Teilmenge der Ressourcen, die auf dem Prototyping-Board selbst zur Verfügung gestellt werden, aufteilen.

5.2.3 Architektur der rekonfigurierbaren Entwicklungsumgebung

Um das Prinzip der Partialemulation anwenden zu können, muss das Prototyping-System besondere Eigenschaften haben. Sowohl das Prototyping-Board als auch das FPGA müssen bestimmte Ressourcen zur Verfügung stellen, um ein SoC integrieren zu können. Diese Eigenschaften werden in den folgenden beiden Abschnitten behandelt.

Ressourcen auf dem Prototyping-Board

In Kapitel 2 wurden die *Field Programmable Gate Arrays (FPGAs)* vorgestellt, die für die Emulation fast beliebiger Hardware-Schaltungen verwendet werden können.

Diese Chips sind zentraler Bestandteil der hier vorgeschlagenen Entwurfsumgebung von SoC-Entwürfen. Neben dem FPGA sind allerdings für die Integration eines komplexen SoC noch weitere Komponenten notwendig. In Abbildung 5.7 ist eine generische Prototyping-Plattform dargestellt, die diese Voraussetzungen erfüllt und die im Folgenden näher beschrieben werden soll.

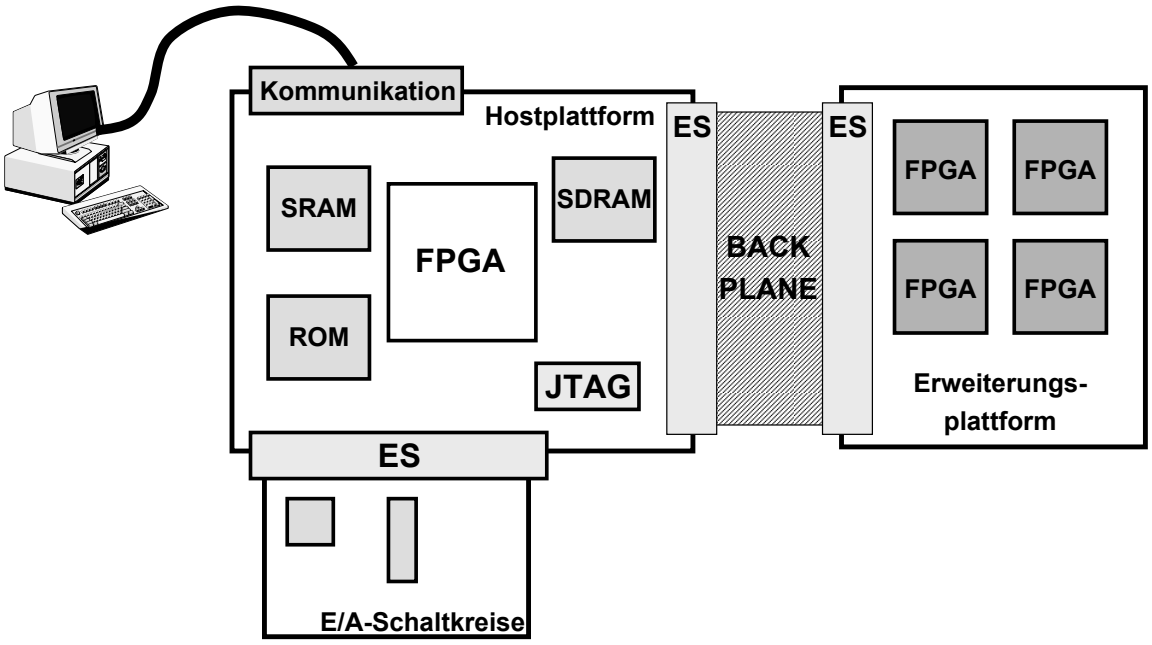


Abbildung 5.7: Generische Prototyping-Plattform zur Integration komplexer SoCs.

Daneben sind für die Software-Entwicklung vor allem große Speicherbausteine wichtig, die an Stelle der im FPGA implementierten Speicher verwendet werden können. Als Speicherbausteine sollten sowohl synchrone statische als auch synchrone dynamische RAM-Bausteine vorhanden sein. Da für synchrone SRAMs keine allzu große zusätzliche Logik im FPGA benötigt wird, empfehlen sich diese besonders für die Emulation von sehr ressourcenintensiven Systemen. Die SDRAM-Bausteine können dagegen zum Testen eines SDRAM-Kontrollers des zu emulierenden SoC verwendet werden.

Darüber hinaus sollte die Entwicklungsplattform zahlreiche Schnittstellen zur Verfügung stellen. Zur Kontrolle des Systems ist es günstig, eine schnelle Kommunikationsschnittstelle mit einem PC zur Verfügung zu haben. In eingebetteten Systemen wird die Kommunikation oft über eine serielle Schnittstelle realisiert. Diese ist zwar einfach zu implementieren, verfügt aber nur über eine begrenzte Bandbreite. Die Realisierung einer PCI-Schnittstelle und der Betrieb des Boards als PCI-Einsteckkarte bietet dagegen einige Vorteile. Über die Schnittstelle ist es dann so-

wohl möglich, das FPGA zu konfigurieren als auch mit dem FPGA zu kommunizieren. In Abschnitt 5.2.5 wird beispielsweise beschrieben, wie mit Hilfe einer PCI-Schnittstelle eine eingeschränkte Software-Entwicklung durchgeführt werden kann.

Andere Erweiterungsstecker (ES) werden dazu benutzt, das Prototyping-System um anwendungsspezifische Komponenten des SoC zu ergänzen. Die Erweiterungsstecker sollten dabei generisch aufgebaut sein, um die Entwicklungsumgebung nicht einzuschränken. Ein Stecker sollte es beispielsweise möglich machen, entsprechende Schaltelemente anzusteueren, die auf einer zusätzlichen Platine realisiert sind und über die das SoC mit der Außenwelt kommunizieren kann.

Eine andere Schnittstelle wird benötigt, um die Anbindung weiterer FPGA-Boards zu ermöglichen. Diese sollen dem Entwickler vor allem zusätzliche Logikressourcen für die Implementierung zusätzlicher Hardware-Komponenten des SoC zur Verfügung stellen. Da der Mikrokontroller und dessen Speicherhierarchie auf der Hostplattform implementiert ist, kann es sich bei den Erweiterungsplattformen um einfache Boards mit günstigen FPGA-Bausteinen handeln. Auf diese Weise werden zusätzlich Kosten für das Prototyping-System eingespart.

Darüber hinaus ist es beispielsweise möglich, den Mikrokontroller-IP-Kern auf der Hostplattform fest zu implementieren und dem Kunden nur die Schnittstellen am Erweiterungsstecker bekannt zu machen. Dadurch kann der Kunde eigene Hardware-Komponenten auf der Erweiterungsplattform integrieren, ohne dass er die Quellen des Mikrokontroller-IP-Kerns für die Synthese benötigt. Die IP-Konfiguration kann in diesem Fall beispielsweise aus einem ROM-Baustein beim Booten der Hostplattform in das FPGA geladen werden. Eine JTAG-Schnittstelle für das Hardware-Debugging komplettiert die Menge der Komponenten des Prototyping-Boards.

Ressourcen im FPGA

Wie in Abbildung 5.5 gezeigt wurde, werden die von FPGAs bereitgestellten generischen Logikressourcen denen, die bei ASICs vorhanden sind, immer im Nachteil sein, da diese zur Emulation beliebiger Hardware-Schaltungen verwendet werden können. Aus diesem Grund sind die FPGA-Hersteller dazu übergegangen, neben den generischen Teilen auch dedizierte Hardware-Blöcke in die FPGAs zu integrieren. Diese beanspruchen aufgrund ihrer Ausführung als Hardmakro weniger Platz als eine Realisierung in konfigurierbaren Logikressourcen. Dazu zählen unter anderem auch spezielle Addierer und Multiplizierer die beispielsweise sehr gut zur Realisierung von DSP-Anwendungen geeignet sind.

Darüber hinaus verfügen moderne FPGAs über mehrere globale Taktnetze, mit denen unterschiedliche Taktdomänen für einen Entwurf implementiert werden kön-

nen und es gibt spezielle E/A-Anschlüsse, die eine effiziente Kommunikation mit der FPGA-Umgebung erlauben.

Mittlerweile bieten alle am Markt verfügbaren FPGAs dem Entwickler spezielle RAM-Module auf dem Chip an. Da diese für sehr unterschiedliche Anwendungsgebiete benutzt werden sollen, besitzen sie eine sehr generische Architektur und haben aufgrund ihrer Integration im Chip sehr schnelle Antwortzeiten. Darüber hinaus gibt es, wie bereits in Kapitel 3.2 angesprochen, FPGAs, in denen ganze Mikrokontroller-Kerne integriert sind. Diese sind als Hardmakro realisiert und können deshalb mit einer sehr hohen Taktfrequenz betrieben werden.

In dieser Arbeit sollen die auf den FPGAs angebotenen Ressourcen dazu verwendet werden, verschiedene Teile eines SoC zu emulieren, um damit mehr Ressourcen für das Prototyping größerer Systeme freizumachen. Da beim Prototyping von SoC-Entwürfen auch andere Mikrokontroller außer den in den FPGAs implementierten Hardmakros verwendet werden sollen und die Hardmakros einen wesentlichen Teil des Chips einnehmen, eignen sich an dieser Stelle nur FPGAs in denen keiner dieser Mikrokontroller-Kerne implementiert ist.

5.2.4 Integration eines System-on-Chip auf dem System

In diesem Abschnitt wird aufgezeigt, wie mit Hilfe der Methode der Partialemulation ein komplexes SoC auf der oben beschriebenen Hardware-Plattform integriert werden kann. Zur Verdeutlichung der in dieser Arbeit entwickelten Methodik soll ein generisches SoC verwendet werden, das in Abbildung 5.8 dargestellt ist.

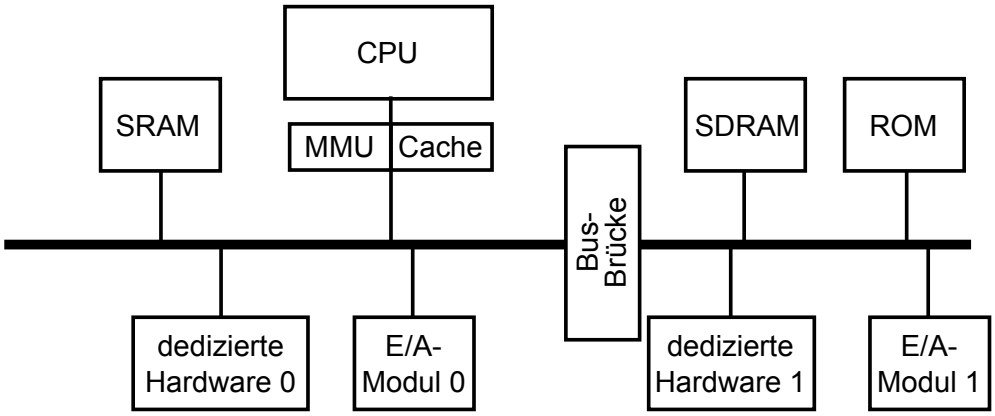


Abbildung 5.8: Aufbau eines generischen SoC mit Mikrokontroller, Speicherhierarchie und Peripheriekomponenten.

Moderne SoCs bestehen heute aus einer Kombination eines oder mehrerer Mikrokontroller/-prozessoren mit dazugehöriger Speicherhierarchie und zusätzlichen Pe-

ripheriekomponenten, die eine bestimmte Funktionalität implementieren. Mikroprozessoren haben gegenüber dedizierten ASIC-Bausteinen den Vorteil, dass sie eine effiziente Möglichkeit der Realisierung eines eingebetteten Systems bieten und dass es mit Mikrokontrollern einfacher ist, eine bestimmte Familie von Geräten mit unterschiedlicher Funktionalität und damit unterschiedlichem Preisniveau anzubieten bzw. diese durch neue Funktionalität weiter auszubauen [108].

Wie oben bereits angesprochen wurde, können die hier untersuchten Systeme mehrere unterschiedliche Taktdomänen haben. Beispielsweise wird der Prozessorbus oft durch eine Bus-Brücke von einem Bus für Peripheriekomponenten entkoppelt. Dies hat den Vorteil, dass der Prozessorbus nicht von langsamen Peripheriekomponenten, die einige Wartezyklen für die Beantwortung von Anfragen benötigen, verlangsamt wird.

Auf der linken Seite der Abbildung sind alle Module des SoC dargestellt, die sich mit der CPU in der gleichen Taktdomäne befinden. Es sind dies ein Cache und eine MMU für den Prozessor sowie ein synchrones RAM (SRAM) zur Speicherung der Software. Darüber hinaus sind hier auch zwei Peripheriekomponenten implementiert. Die dedizierte Hardware 0 (*DHW0*) kann beispielsweise ein Co-Prozessor für Gleitkommaarithmetik sein und das E/A-Modul 0 (*EAM0*) könnte beispielsweise eine Anbindung eines Gerätes sein, welches sicherheitskritische Aufgaben erfüllt und deshalb schnell angesprochen werden muss. Die Busbrücke entkoppelt die schnelle Taktdomäne der CPU von langsameren Peripheriekomponenten. Das synchrone dynamische RAM (SDRAM) und ROM speichern Daten und die dedizierte Hardware 1 (*DHW1*) und E/A-Modul 1 (*EAM1*) erfüllen anwendungsspezifische Aufgaben.

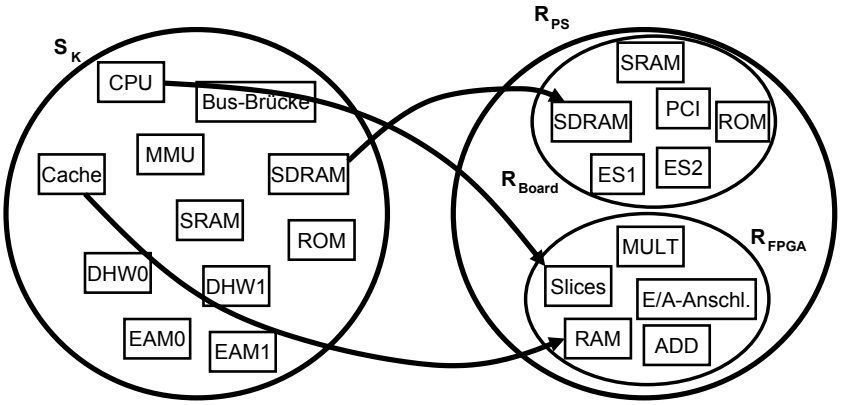


Abbildung 5.9: Abbildung einer Menge S_K von SoC-Komponenten auf die Ressourcen R_{PS} eines Prototyping-Systems.

Abbildung 5.9 zeigt die beiden Mengen S_K der Komponenten des SoC aus Abbildung 5.8 und R_{PS} der Ressourcen des in Abschnitt 5.2.3 beschriebenen generischen

Prototyping-Systems. Gesucht ist eine Abbildung für die Komponenten des SoC auf das Prototyping-System.

Aufgrund der beschränkten Ressourcen des Prototyping-Systems müssen dabei für die jeweiligen Komponenten des SoC Prioritäten vergeben werden, nach denen die Reihenfolge bestimmt wird, in der die Funktionsblöcke integriert werden. Dadurch ist es möglich, durch die Integration eines minimalen Teilsystems des SoC ein lauffähiges Hardware-Modell zu implementieren. Tabelle 5.1 zeigt die Zuweisung von Prioritäten für Komponenten des oben beschriebenen SoC. Komponenten mit niederen Zahlen haben eine höhere Prioritätsstufe.

Priorität	Komponenten
0	CPU, SRAM
1	DHW0, EAM0
2	Bus-Brücke
3	ROM, DHW1, EAM1
4	SDRAM
5	Cache, MMU

Tabelle 5.1: Einteilung der Komponenten eines generischen SoC in Prioritätsklassen.

Die höchste Priorität wird der CPU des Mikrokontrollers sowie dem direkt mit ihr verbundenen Speicher zugeordnet, da ohne diese beiden Komponenten keine Software ausgeführt und evaluiert werden kann. Um zunächst Ressourcen einzusparen, werden den Cache und der MMU die niedersten Prioritäten zugewiesen, da sie zur Software-Entwicklung nicht unbedingt benötigt werden. Soll allerdings ein architekturgenaues Modell des SoC implementiert werden, ist ihre Integration unabdingbar. Andere Komponenten, die sich mit der CPU in der gleichen Taktdomäne befinden, werden als nächstes ausgewählt. Welche der beiden am Prozessorbus angeschlossenen Komponenten *DHW0* und *EAM0* als erstes implementiert wird, kann vom Entwickler entschieden werden, je nachdem welche Funktionalität überprüft werden soll. Handelt es sich bei der Komponente *EAM0* um ein Modul zur Kommunikation mit einem Software-Debugger, so sollte dieses in jedem Fall als nächstes eingeplant werden.

Es folgt die Bus-Brücke, da diese eine natürliche Schnittstelle des SoC darstellt. Sie ist oft gut dokumentiert und besitzt darüber hinaus weniger Schnittstellensignale als interne Module. Die Bus-Brücke ist deshalb ein Modul, welches als Schnittstelle des Prototyping-Systems zu den Erweiterungsboards verwendet werden kann. Da-

nach werden wieder je nach gewünschter Funktionalität, die überprüft werden soll, zunächst die Module *DHW1* und *EAM1* eingeplant, bevor das System um zusätzlichen Speicher erweitert wird.

Nach erfolgter Zuordnung der Prioritäten zu Funktionsblöcken des SoC wird dann eine Abbildung auf die Ressourcen des Prototyping-Systems vorgenommen. Wie im vorangegangenen Abschnitt bereits angedeutet wurde, bieten sich die Block-RAMs des FPGAs an, schnellen On-chip-Speicher des SoC nachzubilden. Die Block-RAMs sind leicht anzusteuern und es sind Schreib- und Lesezugriffe innerhalb eines Taktes möglich. Bei der Integration eines SoC werden deshalb die Caches, schneller On-chip-Speicher sowie das ROM auf die Block-RAMs des FPGA abgebildet. Dies bietet darüber hinaus den Vorteil, dass das ROM des Systems bei der Synthese mit einem Startprogramm gefüllt werden kann, welches nach dem Reset vom Prozessor ausgeführt wird und so das System in einen definierten Zustand bringt.

Für die CPU werden die konfigurierbaren Logikblöcke des FPGAs verwendet. Die dedizierten Multiplizierer des FPGAs bieten sich zwar für die Implementierung von Teilen der arithmetischen Einheit der CPU an, sind aber aufgrund ihrer speziellen Eigenschaften nur schwer integrierbar. Der Glue-code für eine CPU würde hier eher stören, weshalb die Multiplizierer zu diesem Zweck nicht verwendet werden sollten. Die Multiplizierer des FPGA sind für die Implementierung komplexer DSP-Funktionalität besser geeignet. Für die Implementierung eines DSP-Kerns des SoC könnten die Multiplizierer deshalb herangezogen werden.

Sieht man von der Integration eines DSP-Kerns ab, werden auch andere dedizierte Hardware-Module sowie Ein-/Ausgabe-Bausteine mit Hilfe der konfigurierbaren Logikblöcke des FPGAs implementiert. Für die Abbildung der Speicherhierarchie des SoC auf das Prototyping-System ist es notwendig, Schnittstellen zu implementieren, die die Ansteuersignale für die RAM-Chips des Boards und die Block-RAMs des FPGAs erzeugen. Auch diese Schnittstellen werden mit Logikressourcen des FPGAs implementiert.

Generell sollte das Prototyping-System dem Entwickler große und schnelle Speicherchips zur Verfügung stellen, die als RAM für das SoC verwendet werden können. Dadurch werden weniger Block-RAM Ressourcen des FPGAs benötigt und es können größere Programme verifiziert werden. Die RAM Chips sollten sich durch eine möglichst einfache Ansteuerlogik auszeichnen, um Logikressourcen des FPGAs für die Kontrolle der RAMs zu sparen. SDRAMs sollten nur dann verwendet werden, wenn das SoC einen entsprechenden Kontroller besitzt. Das Ergebnis der Abbildung des in Bild 5.8 eingeführten SoC auf die in Abschnitt 5.2.3 vorgestellte generische Prototyping-Plattform, ist in Abbildung 5.10 dargestellt.

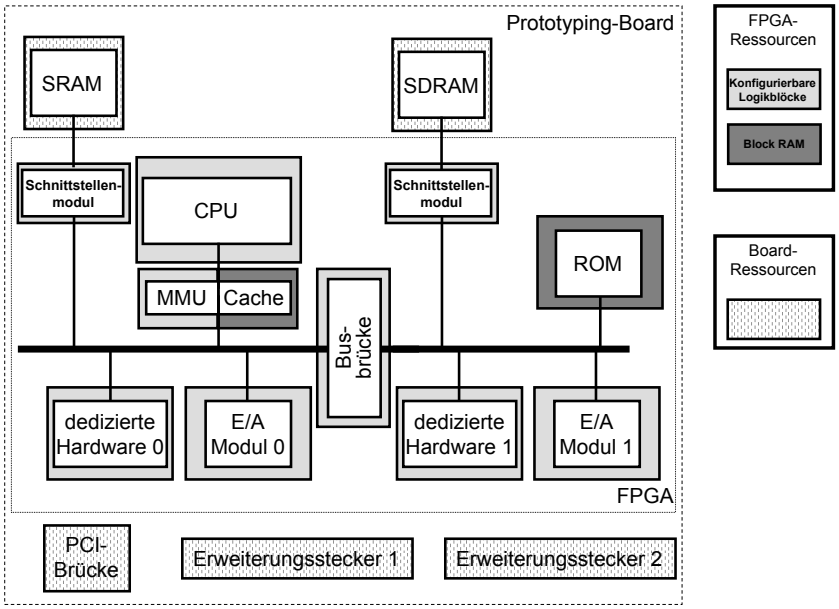


Abbildung 5.10: Abbildung eines SoC auf ein FPGA-basiertes Prototyping-System.

5.2.5 Hardware-nahe Software-Entwicklung unter harten Ressourceneinschränkungen

Aufgrund der eingeschränkten Ressourcen, die in FPGA-basierten Prototyping-Umgebungen herrschen, können für die oben erwähnte Debugger-Infrastruktur hardwareseitig unter Umständen nicht mehr genügend Ressourcen zur Verfügung stehen. Um unter diesen Bedingungen trotzdem Software testen zu können, ist eine alternative Methode zum Debugging der Software notwendig.

In Abschnitt 5.2.2 wurde die Partialemulation vorgestellt, mit der das Ressourcenproblem von FPGAs beseitigt werden kann. Da es sich bei den Hardware-IP-Modulen zur Anbindung von Software-Debuggern um sehr spezielle Implementierungen handelt, ist in der Regel eine Emulation dieser Teile eines SoC nur mit Logikressourcen des Prototyping-Systems möglich.

Reichen diese Logikressourcen nicht aus, ist mit einem dreistufigen Entwicklungsansatz [87] dennoch eine sehr hardware-nahe Software-Entwicklung zu erreichen. Der Ansatz ist in Abbildung 5.11 dargestellt. Im Rahmen dieses dreistufigen Entwicklungsansatzes wurden Werkzeuge entwickelt und prototypisch realisiert, die die Ausführung von Programmen ermöglichen und eine Fehlererkennung auf unterer Ebene bieten. Zunächst muss allerdings das Laden der Software auf das System, welches normalerweise vom Software-Debugger übernommen wird, ermöglicht werden.

Die Kommunikation des Werkzeugs mit dem Prototyping-Board erfolgt über die

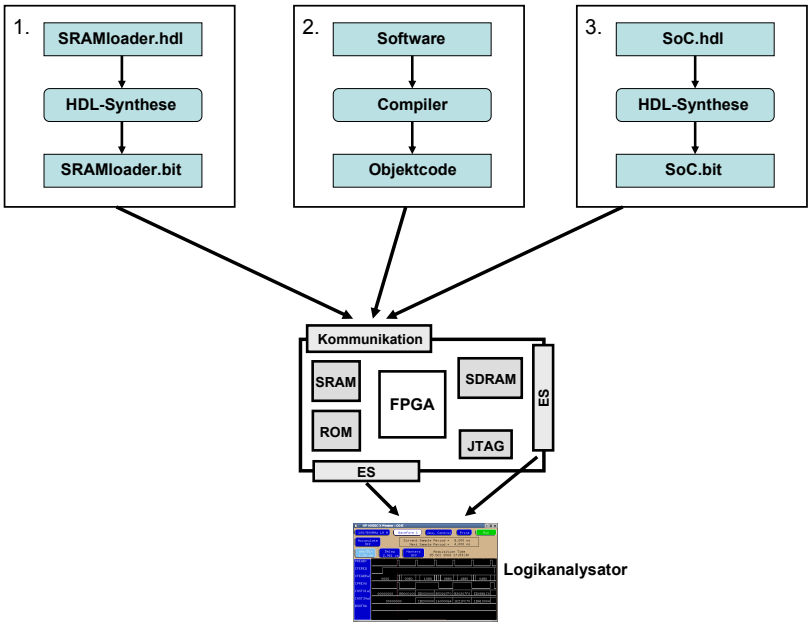


Abbildung 5.11: Software-Entwicklung unter harten Ressourceneinschränkungen.

Kommunikationsschnittstelle, die aus Performanzgründen als PCI-Schnittstelle realisiert werden sollte. In einem ersten Schritt wird eine entwickelte Hardware-Schaltung in das FPGA geladen, die Daten von der Kommunikationsschnittstelle des PC übernehmen und in das SRAM des Prototyping-Boards schreiben kann. In einem zweiten Schritt wird dann über Low-level-PCI-Treiber und die Transferschaltung im FPGA der zuvor erzeugte Objektcode des zu analysierenden Programms in den SRAM-Chip transferiert.

Im dritten Schritt kann dann die Bitdatei des synthetisierten SoC in das FPGA geladen werden. Der Prozessor des SoC muss dabei so implementiert werden, dass er von einer Adresse bootet, die im Adressbereich des SRAMs liegt. Auf diese Weise wird das zu testende Programm nach Beendigung des Herunterladens der Bitdatei sofort ausgeführt.

Trotz der eingeschränkten Ressourcen kann das Debugging der Software auf einer unteren Abstraktionsebene durchgeführt werden. So ist es zwar nicht möglich, Befehle im Einzelschrittmodus auszuführen und sich dabei die Registerinhalte direkt anzeigen zu lassen. Es ist aber möglich, mit Hilfe kleiner Debugging-Module den Status des Programms auf einem Logikanalysator anzuzeigen. Darüber hinaus besteht die Möglichkeit, über eine serielle Schnittstelle am Prozessorbus Informationen auf einem Terminal eines PCs auszugeben [18]. Da beliebige Signale der Hardware auf dem Logikanalysator angezeigt werden können, kann für das Debugging auch der Registersatz des Prozessors herausgeführt werden.

Die selbst entwickelten Debugging-Module werden direkt an den Prozessorbus angeschlossen und sind in den Adressraum des Mikrokontrollers eingeblendet. Programmabschnitt 5.1 verdeutlicht, wie die Debugging-Marken realisiert werden können.

```
1  main() {
2    unsigned int *lmb_p = (unsigned int*) 0xf0000900;
3    unsigned int  result ;
4
5     result  =  algorithm_to_test () ;
6
7    if  ( result ==7) {
8      *lmb_p = 0x900d;
9    }
10   else  {
11     *lmb_p = 0xdead;
12   }
13 }
```

Programm 5.1: Ansteuerung des Debugging-Moduls.

In Zeile 2 wird zunächst ein Zeiger definiert, der auf die Adresse des Debugging-Moduls zeigt. Im Beispiel ist dies die Adresse `0xf0000900`. Dann wird das Ergebnis eines Algorithmus berechnet und mit einem erwarteten Ergebnis, im Beispiel der Wert 7, verglichen. Danach kann dann ein Wert für Erfolg (`0x900d`) oder Misserfolg (`0xdead`) ausgegeben werden. Diese Methode kann auch angewendet werden, um den Programmablauf zu dokumentieren, indem dem Debugging-Modul fortlaufende Nummern übergeben werden. Die Informationen werden dann vom Debugging-Modul an Anschlüsse des FPGAs gelegt, von wo sie mit einem Logikanalysator ausgelesen werden können. Wenn die Ressourcen ausreichen, kann auch eine serielle Schnittstelle am Prozessorbus implementiert werden, die Informationen auf einem Hyperterminal ausgibt.

Aufgrund der statischen Eigenschaft von SRAMs ist es darüber hinaus möglich, in einem vierten Schritt den Inhalt der SRAMs auszulesen, und auf diese Weise einen Speicherdump zu erzeugen, welcher weiter analysiert werden kann.

5.3 Parallele Entwicklung von Hardware-Komponenten und hardware-naher Software

Das zweite Ziel dieser Arbeit besteht darin, die Entwicklung und Integration komplexer Hardware-Komponenten in vorhandene SoC-Plattformen zu verbessern. Darüber hinaus soll die parallele Entwicklung hardware-naher Software für diese neuen Komponenten unterstützt werden, um die Entwurfszeit für SoCs zu verkürzen.

5.3.1 Überblick

Wie in Kapitel 2 bereits erwähnt wurde, werden zukünftige SoCs aus Kostengründen und aufgrund der immer komplexer werdenden Anwendungen nicht mehr von Grund auf neu entwickelt. Stattdessen wird die Hardware-Plattform aus IP-Komponenten aufgebaut und um zusätzliche, neu entwickelte Hardware-Komponenten erweitert, die eine spezielle Funktion erfüllen. Gleiches gilt für die Software, die möglichst früh um hardware-nahe Software für die zusätzlichen Hardware-Komponenten ergänzt werden muss.

Die in dieser Arbeit vorgestellte Entwicklungsmethodik unterstützt dieses Entwurfsparadigma durch die Erweiterung der in Abschnitt 5.2.4 geschilderten Menge R_{PS} der Ressourcen des Prototyping-Systems um Erweiterungsplattformen, die für die Implementierung dieser zusätzlichen Hardware-Komponenten verwendet werden. Das Prinzip der Integration zusätzlicher Hardware-Komponenten auf das vorgestellte Prototyping-System ist in Abbildung 5.12 dargestellt.

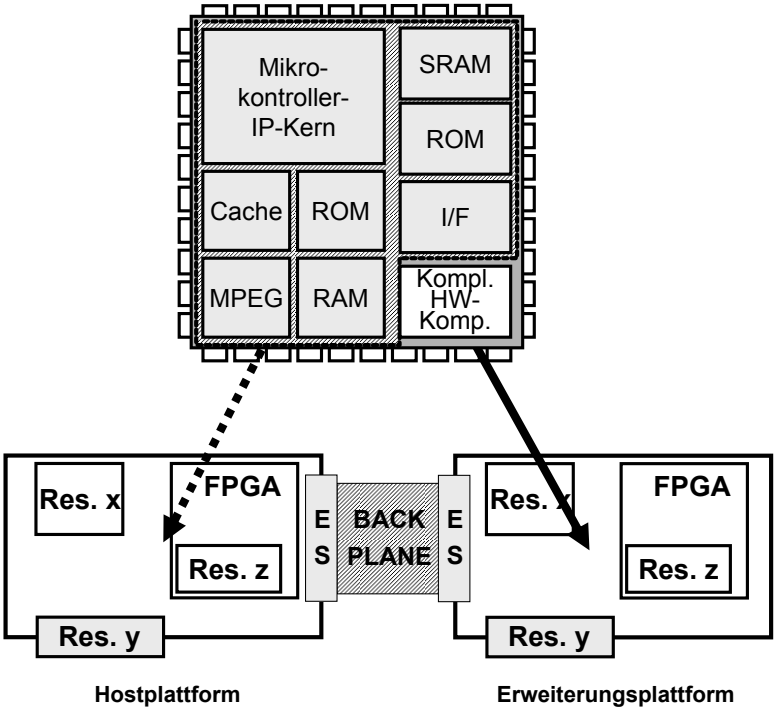


Abbildung 5.12: Erweiterung der rekonfigurierbaren Entwicklungsumgebung um Erweiterungsplattformen.

Die Verwendung einer Erweiterungsplattform zur Implementierung zusätzlicher Hardware-Komponenten hat viele Vorteile. Zum Einen ist das aus verifizierten IP-Modulen bestehende SoC auf der Hostplattform geschützt, da die Kunden, die zu-

sätzliche Hardware-Komponenten implementieren wollen, nur die Schnittstellen am Erweiterungsstecker sehen und nicht den gesamten Quelltext für die Synthese des Gesamtsystems benötigen. Es wird aber auch Synthesezeit gespart, da nur die neu zu entwickelnde Komponente nach dem Auffinden eines Fehlers neu synthetisiert werden muss. Darüber hinaus können sehr schnell unterschiedliche Realisierungen des SoC auf der Hostplattform zusammen mit der neuen Hardware-Komponente evaluiert werden, indem umgekehrt schon synthetisierte SoC-Plattformen, die sich beispielsweise durch unterschiedliche Cachegrößen auszeichnen, ausgetauscht werden. Insgesamt ergibt sich der in Abbildung 5.13 dargestellte Entwurfsablauf.

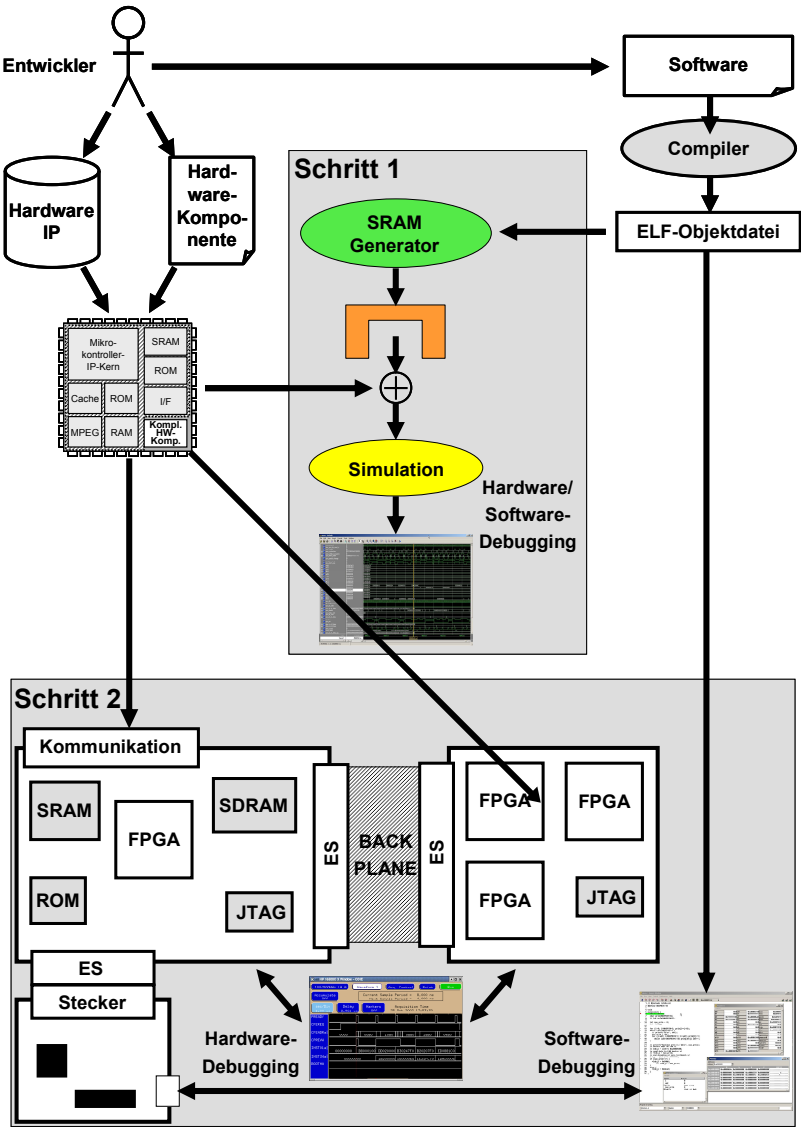


Abbildung 5.13: Entwurfsablauf für die Entwicklung eingebetteter Hardware/Software-Systeme.

Der Ablauf des Entwurfs gliedert sich in zwei Schritte. Da sich neue Hardware-Komponenten am Anfang ihrer Entwicklung noch in einem sehr instabilen Zustand befinden und unter Umständen noch nicht alle Funktionen implementiert sind, wird das Gesamtsystem in einem ersten Schritt simuliert. Damit wird zunächst die Grundfunktionalität der neuen Hardware-Komponente überprüft. Eine Beschreibung der Simulationsumgebung findet sich in Abschnitt 5.3.2.

Im zweiten Schritt wird das gesamte SoC dann auf dem Prototyping-Board integriert und ausgiebigen Tests unterzogen. Die für die Tests verwendete Software kann später als Grundlage für die weitere Software-Entwicklung dienen. In Abschnitt 5.3.3 wird zunächst die Integration des SoC mit zusätzlichen Hardware-Komponenten auf einem Prototyping-Board beschrieben. Daran anschließend folgt eine Diskussion des Prototypings von SoCs mit Erweiterungsplattformen. Die Emulation und das Debugging der Hardware ist Gegenstand von Abschnitt 5.3.5.

Die Software-Entwicklung mit der hier vorgestellten rekonfigurierbaren Evaluierungsumgebung kann auf unterschiedlichen Abstraktionsebenen durchgeführt werden, angefangen von einer einfachen Benachrichtigung des Benutzers über die Terminierung des Programmablaufs bis hin zu einem transparenten Debugging von Software-Programmen. Dies hängt im Wesentlichen damit zusammen, wie effizient die Hardware des SoC auf das Prototyping-System abgebildet werden kann, da für die Anbindung eines Software-Debuggers spezielle Hardware-IP-Module erforderlich sind, die auch auf dem Prototyping-System implementiert werden müssen. Eine Methode zum Software-Debugging unter eingeschränkten Ressourcen war bereits Gegenstand von Abschnitt 5.2.5. In Abschnitt 5.3.6 wird beschrieben, wie die Software-Entwicklung und das Debugging mit der vorgestellten Prototyping-Plattform und einem herkömmlichen Software-Debugger durchgeführt werden kann.

5.3.2 Simulation des eingebetteten Hardware/Software-Systems

Die in dieser Arbeit vorgestellte Prototyping-Umgebung zeichnet sich vor allem durch eine Beschleunigung des Hardware-Modells des zu entwickelnden Systems aus. Es wird sowohl das Hardware- als auch das Software-Debugging unterstützt. Im Gegensatz zum Software-Debugging sind allerdings die Turn-around-Zeiten für das Hardware-Debugging höher, da hier das System neu synthetisiert werden muss, falls andere Signale für den Logikanalysator aus dem FPGA herausgeführt werden oder ein Fehler im HDL-Quelltext beseitigt wurde.

Aus diesem Grund ist es nicht sinnvoll, eine neue Hardware-Komponente sofort auf das Prototyping-System zu bringen. Vielmehr wird zunächst eine Simulation des Gesamtsystems durchgeführt, um die grundlegende Funktionsfähigkeit der neuen Hardwarekomponente sicherzustellen. Hierfür wurde eine Simulationsumgebung

für die Systementwicklung aufgesetzt. Nach erfolgreicher Simulation einiger Basisfunktionen kann das System dann mit Hilfe von Stress-Tests vollständig verifiziert werden.

Da in der hier vorgeschlagenen Methodik Ressourcen des Prototyping-Systems verwendet werden, um Teile des eingebetteten Systems zu emulieren, müssen für diese Ressourcen entsprechende Hardware-Modelle vorliegen, die mit dem zu entwickelnden System kombiniert werden können. Die Simulationsbibliotheken für Komponenten des FPGAs werden in der Regel von den FPGA-Herstellern selbst angeboten. Darüber hinaus sollten Speicherbausteine für das Prototying-Board ausgewählt werden, für die vom Hersteller auch Simulationsmodelle in einer HDL zur Verfügung gestellt werden.

Für die Simulation von Software-Programmen muss es möglich sein, den Code in das HDL-Simulationsmodell zu integrieren. Dies kann durch die Implementierung des RAMs als HDL-Feld geschehen, welches mit den Programmdaten gefüllt wird. Im Rahmen der Arbeit wurde das Werkzeug *tc4spyder* entwickelt, welches unterschiedliche Objektcode-Formate einlesen und daraus die entsprechenden RAM-Inhalte generieren kann. Abbildung 5.14 zeigt beispielhaft die Integration des Codes in das Hardware-Modell des SoC.

Da es sich bei der vorgestellten Analyse des SoC durch Simulation um ein eingebettetes Hardware/Software-System handelt, tritt das Problem auf, dass im HDL-Simulator der Programmablauf nicht sichtbar ist. Der Befehlsstrom wird dort in Form von Signalen und Binär- oder Hexadezimalwerten dargestellt. Es muss deshalb sichergestellt werden, dass der Entwickler Zeitpunkte innerhalb der HDL-Simulation Abschnitte des Software-Programms zuordnen kann, um ein Verständnis für den Programmablauf zu bekommen.

Hierfür wird ein *Trigger-Modul* verwendet, welches an den Peripheriebus des Mikrokontrollers angeschlossen und in dessen Adressbereich eingeblendet ist. Das Trigger-Modul treibt ein Trigger-Signal sowie eine oder mehrere Statusleitungen, die in der Simulation aber auch in der Emulation sichtbar gemacht werden können.

Im Programm werden dann an markanten Punkten dem Trigger-Modul Informationen zum Programmablauf übergeben, welches dieses dann auf seine Signal- und Statusleitungen schreibt. Diese können sowohl in der Simulation als auch der Emulation sichtbar gemacht werden und der Entwickler kann so Abschnitte der Simulation den Abschnitten im Programmablauf zuordnen. Dadurch wird in der Simulation das Debugging der Hardware erheblich vereinfacht.

In den folgenden Abschnitten wird gezeigt, wie ein komplexes SoC und zusätzliche neue Hardware-Komponenten auf einem Prototyping-System integriert und parallel dazu Software entwickelt werden kann.

Verhaltensmodell der SRAM-Chips:

```
ARCHITECTURE behave OF SRAM_CHIP IS

BEGIN

    main : PROCESS
        -- Memory Array
        VARIABLE bank0, bank1, bank2, bank3 : pmemory;
        .
        .
        .
BEGIN
        if (GNET2_GRESET_0='0') then
          bank0 := CREATE_PRAM_Bank0(0);
          bank1 := CREATE_PRAM_Bank1(0);
          bank2 := CREATE_PRAM_Bank2(0);
          bank3 := CREATE_PRAM_Bank3(0);
        end if;
        WAIT ON Clk;
        IF Clk'EVENT AND Clk = '1' AND Zz = '0' THEN
          .
          .
          .
```

ELF-Objektcode:

```
simu_uart.elf:     file format elf32-tricore

Disassembly of section .text:

a0000000 <_start>:
a0000000:       00 00           nop
a0000002:       00 00           nop
a0000004:       91 00 00 ab     movh.a %sp,45056
a0000008:       d9 aa 68 66     lea %sp,[%sp]26024
a000000c:       91 10 00 0b     movh.a %a0,45057
```

Speicherinhalt der SRAM-Chips:

```
-- pmem_array-p.vhd
-- automatically generated by tc4spyder analyzer
-- Do not edit!
library ieee;
use ieee.std_logic_1164.all;

package pmem_array is
  constant addr_bits : integer := 19;  -- size of program memory
  constant data_bits : integer := 32;   -- bit width of memory
  type pmemory is array (2 ** addr_bits - 1 DOWNTO 0) of
       std_logic_vector(data_bits/4 - 1 downto 0);

  function CREATE_PRAM_Bank0(dummy: in integer) return pmemory;
  …
end pmem_array;

package body pmem_array is
  function CREATE_PRAM_Bank0(dummy: in integer) return pmemory is
    variable pmem_array_inst : pmemory;
    variable index : integer := 0;
  begin
    pmem_array_inst(0)  := to_stdlogicvector(X"00");
    pmem_array_inst(1)  := to_stdlogicvector(X"91");
    …
    return pmem_array_inst;
  end;
  function CREATE_PRAM_Bank1(dummy: in integer) return pmemory is
    …
  begin
    pmem_array_inst(0)  := to_stdlogicvector(X"00");
    pmem_array_inst(1)  := to_stdlogicvector(X"00");
    …
    return pmem_array_inst;
  end;
  function CREATE_PRAM_Bank2(dummy: in integer) return pmemory is
    …
  begin
    pmem_array_inst(0)  := to_stdlogicvector(X"00");
    pmem_array_inst(1)  := to_stdlogicvector(X"00");
    …
    return pmem_array_inst;
  end;
  function CREATE_PRAM_Bank3(dummy: in integer) return pmemory is
  …
  begin
    pmem_array_inst(0)  := to_stdlogicvector(X"00");
    pmem_array_inst(1)  := to_stdlogicvector(X"ab");
    …
    return pmem_array_inst;
  end;
end pmem_array;
```

Abbildung 5.14: Integration des Objektcodes in das SRAM-HDL-Modell der SRAMs des generischen Prototyping-Systems.

5.3.3 Entwicklung von Hardware-Komponenten auf einem Board

Für Entwicklung, Integration und Test zusätzlicher Hardware-Komponenten gibt es zwei Möglichkeiten. Zum Einen können die neuen Komponenten zusammen mit dem schon vorhandenen SoC auf der Hostplattform implementiert werden. Zum Anderen ist es möglich, die Hostplattform über eine Backplane um weitere Prototyping-Boards zu erweitern.

Bei beiden Möglichkeiten wird der Entwurf hierarchisch in mehrere Teile aufgeteilt. Abbildung 5.15 zeigt eine Darstellung der Modulhierarchie des generischen SoC aus Abbildung 5.8 auf Seite 63. An dieser Stelle soll die dedizierte Hardware-Komponente 1 (DHW1) als neu zu entwickelnde Komponente in das System integriert werden.

Für die Integration des SoC auf einem Board wird der Entwurf in zwei Teilmengen aufgeteilt. In der ersten Teilmenge sind alle Komponenten zusammengefasst,

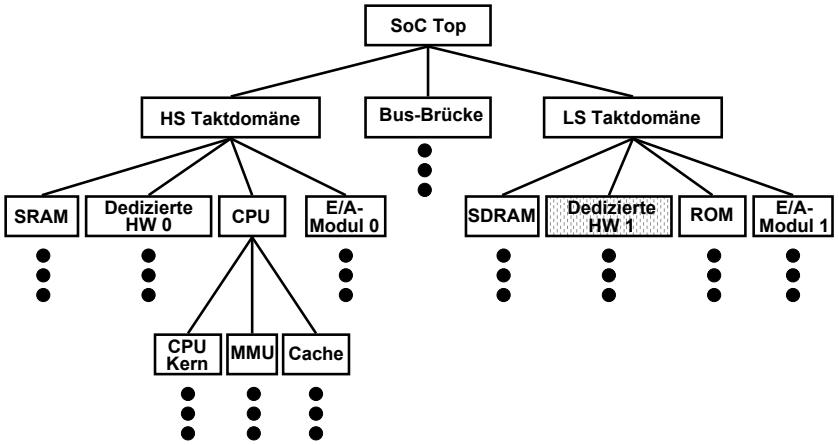


Abbildung 5.15: Hierarchie des generischen SoC aus Abbildung 5.8.

die als Hardware-Plattform für ein Anwendungsgebiet verwendet werden können. In Abbildung 5.16 ist dieser Teil schraffiert hinterlegt. Die neu entwickelten Hardware-Komponenten werden ausgegliedert und sind unter einem Submodul *Peripherie* zusammengefasst.

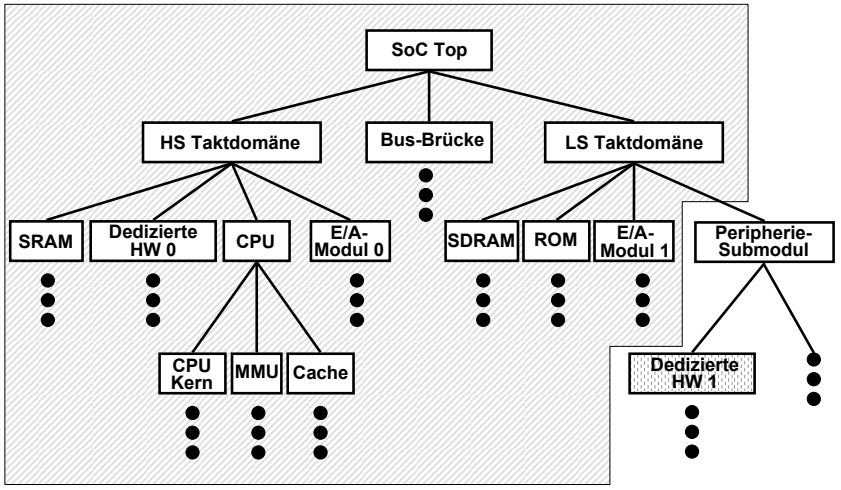


Abbildung 5.16: Architektur des SoC-Prototypen mit dem Peripherie-Submodul.

Mit Hilfe dieses Aufbaus lässt sich die für die Entwicklung eingebetteter Systeme notwendige Flexibilität bezüglich der Hardware-Plattform erreichen. Diese kann beispielsweise unterschiedliche Cachegrößen oder -ersetzungsstrategien, eine MMU oder eine FPU anbieten. Der Vorteil der Integration des Gesamtsystems auf einem Board liegt darin, dass der Prototyp unter Umständen mit einer höheren Taktgeschwindigkeit betrieben werden kann, da Signale nicht über eine Backplane geleitet

werden müssen.

Bei der Entwicklung von Hardware-Komponenten, deren Implementierung in der Regel noch nicht ausgereift ist, kommt es bei dieser Lösung allerdings zu hohen Turn-around-Zeiten, da das gesamte System neu synthetisiert werden muss. Außerdem ergibt sich durch die Integration der zusätzlichen Hardware-Komponenten in demselben FPGA, in dem sich auch die SoC-Plattform befindet, ein höherer Ressourcenbedarf in Bezug auf Logik und Taktdomänen, was unter Umständen dazu führt, dass das Gesamtsystem in einem FPGA nicht mehr platziert und verdrahtet werden kann. Darüber hinaus kann eine zu große Auslastung des FPGAs zu einer verminderten Taktfrequenz führen. Zur Reduktion der Taktdomänen muss gar in die Architektur eingegriffen und diese wie in Abbildung 5.4 auf Seite 57 verändert werden.

Ein weiterer Nachteil der Integration des Gesamtsystems auf einem Board besteht darin, dass für die Synthese des kompletten SoC der HDL-Quelltext aller Komponenten erforderlich ist. Dies stellt für den IP-Hersteller ein Problem dar, da er dem Kunden schon zu Testzwecken den Quelltext der IP-Komponente preisgeben muss.

5.3.4 Hardware-Entwicklung mit Hilfe von Erweiterungsplattformen

Um die im vorangegangenen Abschnitt aufgezeigten Defizite beim SoC-Entwurf durch Emulation zu beseitigen, wird in dieser Arbeit eine Entwicklung neuer Hardware-Komponenten mit Hilfe von Erweiterungsplattformen vorgeschlagen. Durch die Kombination mehrerer Prototyping-Systeme können einige der oben beschriebenen Probleme gelöst werden.

Die grundlegende Idee des Prototypings mit Erweiterungsplattformen ist in Abbildung 5.17 dargestellt. Dabei wird die Hardware des zu entwickelnden eingebetteten Hardware/Software-Systems in zwei Teile aufgeteilt. Einen Teil bildet die SoC-Plattform, die in der Regel aus einem Mikrokontroller, Speicherhierarchie sowie Hardware-IP-Komponenten besteht. Diese SoC-Plattform stellt die Grundlage für ein bestimmtes Anwendungsgebiet dar. Der zweite Teil besteht aus Hardware-Komponenten, die neu entwickelt werden sollen. Diese werden getrennt von der SoC-Plattform auf dem Erweiterungsboard implementiert.

Mit der Hinzunahme des Erweiterungsboards stehen somit zusätzliche Ressourcen für die Integration der Hardware-Komponente zur Verfügung, die in der Abbildung als Menge R_{EP} bezeichnet wird. Wie aus der Abbildung ersichtlich ist, kann die Erweiterungsplattform dabei genau wie die Hostplattform voll ausgestattet sein. D.h. dem Entwickler stehen auch Speicherchips oder eine PCI-Schnittstelle für das Prototyping zur Verfügung. Sie kann aber auch mit günstigeren FPGA-Bausteinen bestückt werden, die viele programmierbare Logikgatter enthalten und damit einen

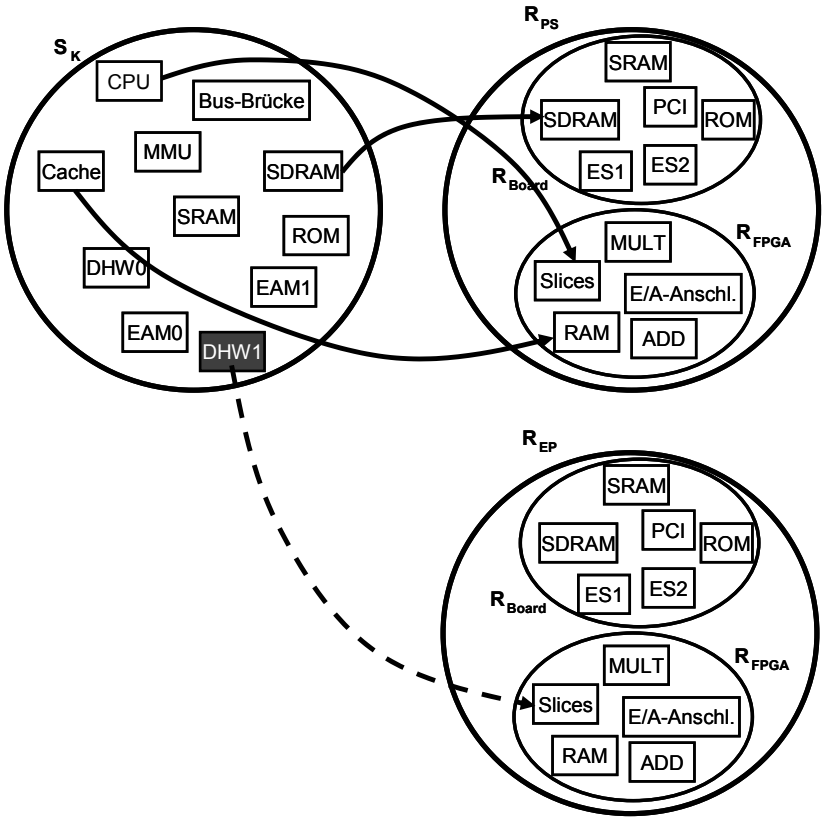


Abbildung 5.17: Integration komplexer Hardware-Komponenten mit einer Erweiterungsplattform.

sogenannten *Sea of Gates* implementieren. An dieser Stelle erhöht sich dann auch die Anzahl der zur Verfügung stehenden Taktdomänen.

Aufgrund der Trennung der beiden Prototyping-Systeme verkürzt sich die Synthese für die neu hinzugefügte Hardware-Komponente. Dies führt zu einer Verkürzung der Turn-around-Zeiten für deren Entwicklung. Auch gestaltet sich die Platzierung und Verdrahtung einfacher und beansprucht weniger Zeit. Weiter können auf der Hostplattform verschiedene Ausprägungen der Prozessorarchitektur, wie z.B. unterschiedliche Cachegrößen, schnell gegeneinander ausgetauscht und mit der neuen Hardware-Komponente kombiniert werden. Darüber hinaus ist durch die Trennung der Plattform mit Mikrokontroller und Speicherhierarchie sowie anwendungsspezifischen Hardware-Modulen von den neu entwickelten Hardwarekomponenten der Schutz der IP gewährleistet. Ein Kunde benötigt für die Integration zusätzlicher Hardware-Komponenten nur Kenntnisse der Schnittstelle zur Hostplattform.

Daraus ergibt sich eine geänderte Modulstruktur für das SoC. Diese ist in Abbildung 5.18 dargestellt. Im Gegensatz zur Integration des Systems in einem FPGA

wird hier das Submodul *Peripherie* durch ein Submodul *FPGA-Bus-Brücke* erweitert, welches die oben beschriebene Schnittstelle zur SoC-Plattform einkapselt. Die Bus-Brücke findet sich sowohl im Entwurf des Hostmoduls als auch auf der Erweiterungsplattform und ist für die Kommunikation verantwortlich. Das FPGA-Bus-Brücken-Modul kann beispielsweise eine Abbildung des Peripheriebusses des SoC auf die FPGA-E/A-Anschlüsse sowie die Verbindungsstruktur des Prototyping-Systems implementieren. Ein Beispiel hierfür wird in Kapitel 6.4.3 näher beschrieben.

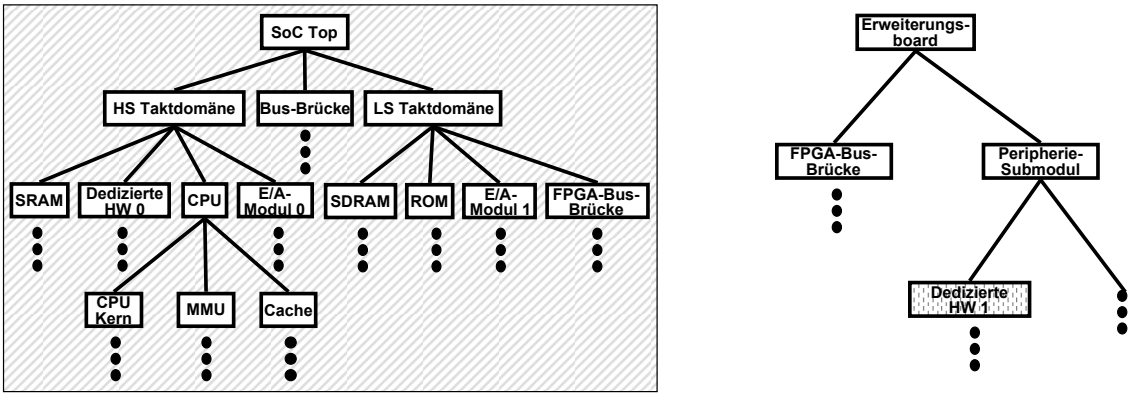


Abbildung 5.18: Partitionierung des SoC mit Erweiterungsplattform.

Der Peripheriebus des SoC ist an der Schnittstelle zur Erweiterungsplattform sichtbar. Damit stehen dessen Signale dort zum Debugging des Busprotokolls zur Verfügung. Das Herausführen des Peripheriebusses hat auch Nachteile. Zum Einen kann sich dadurch der maximal mögliche Systemtakt reduzieren und zum Anderen kann die Anzahl der zur Verfügung stehenden Anschlüsse am FPGA und Signale auf der Backplane für die Implementierung des Peripheriebusses nicht ausreichen. Die hier notwendigen Anpassungen können allerdings in dem FPGA-Bus-Brücken-Modul gekapselt und als Hardware-IP in den Entwurf eingebunden werden.

5.3.5 Emulation und Hardware-Debugging des SoC

In diesem Abschnitt wird die Emulation und das Debugging der zusätzlichen Hardware-Komponenten auf dem Prototyping-Board vorgestellt. Um die neuen Hardware-Komponenten in Betrieb nehmen zu können, müssen diese in ein Gesamtsystem eingebunden werden. Dazu muss zwischen verschiedenen Arten von Hardware-Komponenten unterschieden werden. Bei E/A-Komponenten steht die Interaktion mit der Umwelt im Vordergrund. Hier muss dann eine entsprechende Gegenstelle zur Verfügung stehen, mit der die E/A-Komponente kommuniziert. Zweitens können Hardware-Komponenten dazu verwendet werden, einen Algorithmus zu beschleu-

nigen. Beispielsweise könnte ein MPEG-Encoder in Hardware wesentlich schneller Datenströme kodieren, als dies in Software möglich ist.

Für die Entwicklung und Verifikation von Komponenten, die für die Beschleunigung von Algorithmen zuständig sind, kann Software verwendet werden, die auf dem Mikrokontroller des SoC ausgeführt wird. Diese schickt einen Datenstrom zur Hardware-Komponente und überprüft das Resultat mit dem theoretisch erwarteten Ergebnis. Stimmt das Ergebnis nicht, so arbeitet die neue Hardware-Komponente fehlerhaft.

Die Implementierung der Gegenstellen für die Verifikation von E/A-Modulen kann auf zwei unterschiedliche Arten erfolgen. Das in Abschnitt 5.2.3 vorgestellte generische Prototyping-System verfügt über Erweiterungsstecker, an denen über eine Zusatzplatine eine physikalische Verbindung mit einem Endgerät hergestellt werden kann, welches mit der zu entwickelnden E/A-Komponente kommuniziert. Eine zweite Möglichkeit besteht darin, zusammen mit dem SoC Module zu integrieren, die die E/A-Signale des SoC auffangen bzw. treiben. Um bestimmte Muster an den Eingängen des SoC anzulegen, können dieses Module über die Kommunikationsschnittstelle des Prototyping-Boards Befehle vom PC entgegen nehmen (siehe Abbildung 5.7 auf Seite 61).

Die Kommunikation sowie interne Abläufe der neuen Hardware-Komponente können dann mit *externen* oder *internen* Logikanalysatoren untersucht werden. Für die Anbindung eines externen Logikanalysators stehen die oben bereits erwähnten Erweiterungsstecker zur Verfügung. Neben der Verwendung eines externen Logikanalysators können aber auch interne Logikanalysatoren eingesetzt werden, wenn diese vom FPGA-Hersteller angeboten werden. In Abschnitt 2.4.4 wurde bereits ein interner Logikanalysator für Xilinx FPGAs vorgestellt.

Die Verwendung interner Logikanalysatoren hat den Vorteil, dass die Signale, die untersucht werden sollen, nicht auf die Erweiterungsstecker des Prototyping-Boards herausgeführt werden müssen, sondern intern untersucht werden können. Auf der anderen Seite werden für die Integration der Debugging-Module FPGA-Ressourcen wie Takt- und Signalnetze sowie Block-RAMs benötigt. Darüber hinaus ist nur eine Signalaufzeichnung mit dem Takt des FPGAs möglich.

Externe Logikanalysatoren können dagegen die Signale mit einer höheren Taktrate abtasten. Dadurch sind genauere Aussagen bezüglich des zeitleichen Verhaltens des SoC möglich. Der Nachteil der Lösung besteht darin, dass für das Herausführen der zu untersuchenden Signale Anschlüsse des FPGAs und der Erweiterungsstecker benötigt werden, die dann nicht für andere Zwecke wie etwa Kommunikation mit externen Komponenten zur Verfügung stehen. Die Verwendung eines FPGAs zur Emulation hat gegenüber einem Mikrokontroller-Chip aber immer den Vorteil, dass *beliebige* Signale zum Debugging herausgeführt werden können.

5.3.6 Software-Entwicklung mit einer Cross-Entwicklungsumgebung

Mit der in dieser Arbeit vorgestellten Methodik für eingebettete Hardware/Software-Systeme wird eine schnelle Entwicklung hardware-naher Software in einer frühen Phase des Entwurfsprozesses parallel zum Entwurf der entsprechenden Hardware-Komponente ermöglicht. Dies führt dazu, dass der Entwickler frühzeitig tiefe Einblicke in das Systemverhalten bekommt und so Fehler schnell erkennen und beheben kann. Die Software kann in ihrem Einsatzumfeld unter realen Bedinungen auf einem architekturgenauen Mikrokontroller-Modell getestet und entwickelt werden. Dabei können durch den modularen Aufbau der Prototyping-Umgebung schnell unterschiedliche Varianten einer Mikrokontroller-Plattformen auf ihr zeitliches Verhalten hin untersucht werden.

In Abschnitt 5.2.5 wurde bereits ein Verfahren vorgestellt, mit dem eine eingeschränkte Software-Entwicklung mit einem emulationsbasierten Mikrokontroller-IP-Kern durchgeführt werden kann. Die Einschränkungen des Debuggings können sich aufgrund eingeschränkter Ressourcen des FPGAs ergeben. Dadurch könnte eine Implementierung der Module, die für die Anbindung des Software-Debuggers in der Hardware realisiert werden müssen, unmöglich sein.

Wenn genügend Ressourcen für die Implementierung dieser Debugger-IP-Module im FPGA zur Verfügung stehen, kann mit der Prototyping-Umgebung eine transparente Software-Entwicklung mit Hilfe einer Cross-Entwicklungsumgebung durchgeführt werden [86]. Abbildung 5.19 zeigt die Anbindung einer Cross-Entwicklungsumgebung an das in Abschnitt 5.2.3 vorgestellte generische Prototyping-System.

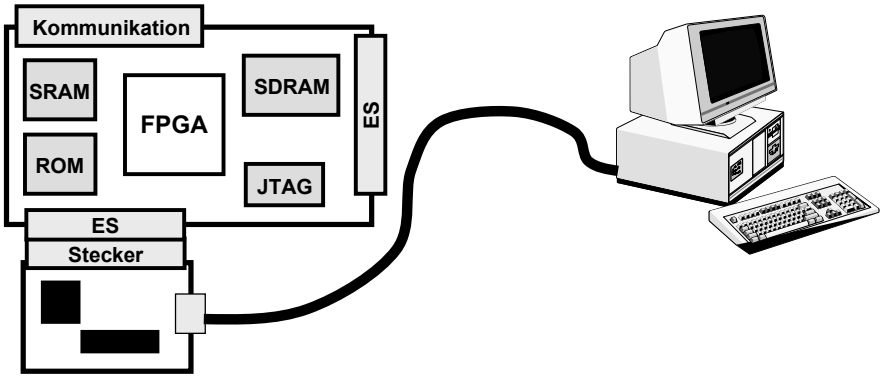


Abbildung 5.19: Anbindung einer Cross-Entwicklungsumgebung an ein generisches Prototyping-System.

In eingebetteten Systemen wird die Anbindung einer Entwicklungsumgebung mit Hilfe paralleler oder serieller Schnittstellen implementiert. Dazu werden zum Einen im SoC selbst Module benötigt, die Signale des Debuggingwerkzeugs in Befehle

für den Prozessor übersetzen und zum Anderen ist eine physikalische Schnittstelle notwendig, welche die Signale dieser Debugging-Module in Signale für die PC-Schnittsstelle übersetzt. In Abbildung 5.19 ist die physikalische Schnittstelle am Erweiterungsstecker der Prototyping-Umgebung direkt mit Anschlüssen am FPGA verbunden. Auf der anderen Seite kann der PC beispielsweise mit einem seriellen Kabel verbunden werden. Die Debugger-IP-Module des SoC selbst werden dann im FPGA implementiert.

Abbildung 5.20 zeigt die Software-Entwicklung mit diesem System. In einem ersten Schritt wird zunächst eine Bitdatei mit der Hardware-Realisierung des SoC auf das FPGA heruntergeladen. Damit sich der Prozessor nach dem Herunterladen in einem definierten Zustand befindet, muss ein Boot-ROM implementiert werden, welches statisch Code zur Initialisierung des Systems enthält. Da für die Initialisierung des Prozessors keine größeren Speicherressourcen benötigt werden, kann das Boot-ROM mit Block-RAM Ressourcen des FPGAs realisiert werden. Für Xilinx FPGAs steht dafür das Werkzeug `data2MEM` zur Verfügung, welches sowohl ein Programm im ELF-Objektformat in entsprechende VHDL- oder Verilog-Dateien für Simulation und Synthese umwandeln als auch in eine Bitdatei integrierte Block-RAMs nachträglich füllen kann. Alternativ dazu könnte zum Booten des Systems auch der ROM-Chip der Prototyping-Plattform selbst verwendet werden.

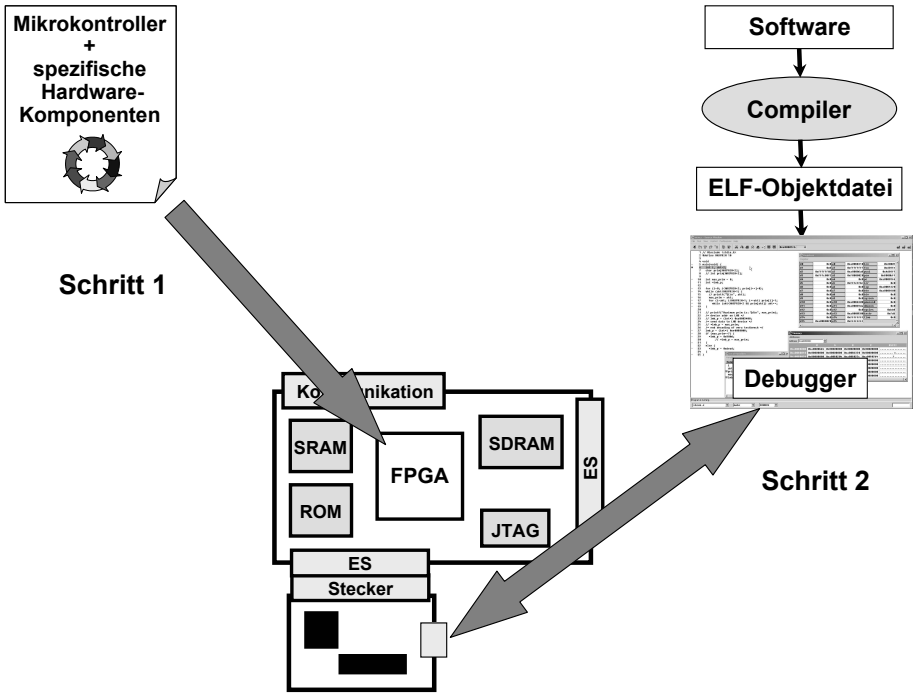


Abbildung 5.20: Software-Entwicklung mit der Cross-Entwicklungsumgebung.

Das Booten des Initialisierungscodes aus dem Block-RAM bietet gegenüber dem Booten aus den SRAMs, wie in Abschnitt 5.2.5 beschrieben, zwei Vorteile. Zum Einen kann das System auch im Standalone-Modus ohne PCI-Anbindung betrieben werden, da das Programm nicht über die PCI-Schnittstelle in die SRAMs heruntergeladen werden muss. Zum Anderen kann der Prozessor durch Betätigen des Reset-Tasters am Prototyping-Board wieder in einen definierten Zustand gebracht werden. Dies kann bei direktem Booten aus den SRAMs heraus zu Problemen führen, wenn deren Speicherinhalt verändert wurde.

Nachdem der Prozessor sowie die Peripheriekomponenten aus dem Boot-ROM heraus initialisiert wurden, kann über die externe Debugger-Schnittstelle von einem Software-Debugger aus auf das System zugegriffen werden. Über diese Verbindung ist es dann möglich, in Maschinencode übersetzte Programme in die SRAMs des Prototyping-Boards herunterzuladen und auszuführen. Dazu muss noch der Inhalt des Programmzählers mit Hilfe des Debuggers so abgeändert werden, dass er auf die Anfangsadresse des SRAM-Chips zeigt, in dem der Programmcode abgelegt ist.

Auf diese Art und Weise ist eine transparente Entwicklungsumgebung geschaffen worden, die dem Entwickler die gewohnte Sicht auf das Mikrokontroller-System gibt, die er von Evaluierungsboards kennt und die so flexibel ist, dass das SoC jederzeit einfach verändert und um zusätzliche neue Hardware-Komponenten erweitert werden kann.

5.4 Zusammenfassung

In diesem Kapitel wurde eine Entwurfsmethodik für eingebettete Hardware/Software-Systeme vorgestellt, die im Wesentlichen zwei Ziele verfolgt. So soll zum Einen die Software-Entwicklung auf einem architekturgenauen Hardware-Modell eines SoCs beschleunigt und zum Anderen die parallele Entwicklung komplexer Hardware-Komponenten und deren hardware-naher Software ermöglicht werden. Da Software bisher erst nach Fertigstellung der Hardware entwickelt wurde, trägt dies zu einer erheblichen Verkürzung der Entwurfszeit für komplexe SoCs bei.

Im Gegensatz zu den bisher bekannten Entwicklungsumgebungen, die entweder auf der Verwendung von mit ASICs ausgestatteten Evaluierungsboards oder der Kombination von Mikrokontroller-Chips und rekonfigurierbarer Hardware beruhen, wird bei der hier vorgeschlagenen Lösung das *komplette* SoC in einem FPGA realisiert. Durch die Emulation wird die Software-Entwicklung wesentlich beschleunigt. Darüber hinaus entsteht eine *Soft-Evaluierungsplattform*, die sehr leicht um zusätzliche neue Hardware-Komponenten erweitert und für die dann parallel Software entwickelt werden kann.

Da die Kosten für die Entwicklungsplattform eine wichtige Rolle spielen, wird die Evaluierungsplattform mit kostengünstigen Emulations-Boards realisiert. Es wurde gezeigt, wie sich die daraus ergebenden Ressourcenprobleme mit der Methode der Partialemulation lösen lassen. Kernidee der Partialemulation ist die effiziente Ausnutzung *aller* Ressourcen des Prototyping-Systems, sowohl im FPGA als auch auf dem Board selbst. Es wurde auch die Möglichkeit eines eingeschränkten Software-Debuggings vorgestellt, mit der Ressourcen des FPGAs eingespart werden können, indem die für das Software-Debugging notwendigen Hardware-IP-Module des SoC nicht implementiert werden.

Im zweiten Teil des Kapitels wurde dann gezeigt, wie ein Prototyping-System erweitert werden muss, damit eine parallele Entwicklung neuer Hardware-Komponenten und hardware-naher Software möglich wird. Hier kommen *Erweiterungsplattformen* zum Einsatz, die für die Integration neuer Hardware-Komponenten verwendet werden. Das Hardware-Debugging dieser Komponenten ist aufgrund detaillierter Einblicke mit Logikanalysatoren gut möglich. Die durch die architekturgenaue Nachbildung der Hardware beobachteten Geschwindigkeitseinbußen bisheriger Entwicklungssysteme konnten durch das vorgestellte neue Konzept beseitigt werden. Die Anbindung einer vollständigen Cross-Entwicklungsumgebung über Erweiterungsstecker des Prototyping-Systems erlauben eine transparente Software-Entwicklung.

Mit den vorgeschlagenen Erweiterungsplattformen, die auch aus wesentlich günstigeren FPGAs bestehen können, die nur als *Sea-of-Gates* dienen und damit mehr Emulationsressourcen anbieten, können schnell neu zu entwickelnde Hardware-Komponenten integriert und evaluiert werden. Da beim Auftreten eines Fehlers in dieser neuen Hardware-Komponente nur ein kleinerer Teil des SoCs neu synthetisiert werden muss, sinken die Turn-around-Zeiten für die Systementwicklung. Darüber hinaus ist die IP des Herstellers auf der Hostplattform gekapselt und somit geschützt. Durch die Verwendung unterschiedlicher Konfigurationsdateien können darüber hinaus sehr schnell unterschiedliche Konfigurationen der Mikrokontroller-Plattform miteinander verglichen werden. Auch hat man an den Schnittstellen zwischen Mikrokontroller-Plattform und Erweiterungsplattform einen detaillierten Einblick in die Hardware/-Software-Schnittstelle der zu integrierenden neuen Hardware-Komponente.

6 Hardware/Software-Entwicklung mit einem emulationsbasierten Mikrokontroller-IP-Kern

Die in dieser Arbeit vorgestellte Entwicklungsmethodik für Hardware und Software komplexer SoCs wird in diesem Kapitel anhand eines praxisrelevanten Beispiels angewendet. Dazu wird das Softmakro eines typischen, komplexen Mikrokontroller-IP-Kerns für SoC-Entwürfe auf einem Prototyping-System integriert, welches in dem in Abschnitt 4.3 anvisierten Marktsegment liegt. Bei dem Mikrokontroller-IP-Kern handelt es sich um den TriCore1 der Firma Infineon Technologies AG, der Teil der DesignWare®-StarIP-Bibliothek der Firma Synopsys [97] ist. Die Bibliothek enthält außerdem Mikrokontroller-IP-Kerne der Firmen IBM (PowerPC 440), MIPS (MIPS32 4KE), NEC (V850E) sowie den C166, ebenfalls von Infineon. Der TriCore1 wird in Abschnitt 6.1 und das Prototyping-System in Abschnitt 6.2 vorgestellt.

Die Integration des TriCore1-Mikrokontrollers auf dem Spyder-System sowie die Anbindung einer Software-Entwicklungsumgebung an das Prototyping-Board ist Gegenstand von Abschnitt 6.3. Dort wird in Abschnitt 6.3.2 auch beschrieben, wie die Ressourcenprobleme FPGA-basierter Emulations-Boards mit Hilfe der Partialemulation gelöst und eine eingeschränkte Entwicklung von Hardware-Komponenten und Software für den TriCore1-IP-Kern möglich ist.

Die parallele Entwicklung von Hardware-Komponenten und ihrer hardware-nahen Software für TriCore1-basierte SoC-Entwürfe wird in Abschnitt 6.4 beschrieben. Dort wird die Anbindung von Erweiterungsplattformen an das Prototyping-System sowie die Abbildung des SoC-Busses auf diese Anbindung aufgezeigt. In Abschnitt 6.4.4 wird auf das Debugging der Hardware-Komponenten näher eingegangen. Das Kapitel schließt mit einer Zusammenfassung.

6.1 TriCore1-Mikrokontroller

Die TriCore® Familie von Infineon bedient das 32-Bit-Marktsegment für eingebettete Systeme und ist in unterschiedlichen Ausprägungen für die Anwendungsfelder Automobiltechnologie, Industrieautomatisierung, Telekommunikationstechnologie und Verbrauchergeräte gedacht.

Im TriCore sind die drei Architekturmerkmale eines Mikrokontrollers, DSPs und RISC-Prozessors zu einem sehr effizienten Gesamtsystem vereinigt. Der TriCore ist

zur Zeit in den beiden Varianten TriCore Version 1.3 und TriCore Version 2 verfügbar, wobei zur Zeit am Markt hauptsächlich noch Derivate der Version 1.3 eingesetzt werden. Der TriCore2 hat eine wesentlich komplexere Busarchitektur und bietet Hardware-Unterstützung für Hyperthreading.

In dieser Arbeit wird als Beispiel der TriCore Version 1.3 (TriCore1) verwendet, da ein Derivat dieser Architektur als DesignWare-StarIP-Komponente verfügbar ist, und somit als idealer Vertreter eines komplexen SoC-Mikrokontroller-IP-Kerns angesehen werden kann. Darüber hinaus treten bei dieser Variante typische Probleme des ASIC-IP-Prototypings auf FPGAs auf, da der Mikrokontroller für ASICs entwickelt wird.

6.1.1 Überblick

Der TriCore1 besitzt eine 32-Bit Lade-/Speicher-Harvard-Architektur und superskalare Ausführungseinheiten. Die Architektur ist in vier Pipelinestufen aufgeteilt und hat eine spezielle Hardware-Unterstützung, um die Antwortzeiten auf Unterbrechungen zu minimieren. Der Prozessor bietet einen 4GB großen Adressraum, wobei Peripheriemodule in den Adressraum eingeblendet werden. Es stehen 16 und 32-Bit-Befehle zur Verfügung, was zu einer effizienteren Speicherausnutzung führt.

Der Kern kann durch eine Gleitkommaeinheit (FPU), eine Memory Management Unit (MMU) oder über eine spezielle Co-Prozessorschnittstelle flexibel um zusätzliche Funktionseinheiten erweitert werden. Darüber hinaus verfügt der Kern über zwei 16x16-Multiply-Accumulate-Funktionseinheiten und spezielle Befehle für DSP-Anwendungen.

Für den Mikrokontroller stehen Peripheriemodule sowie Module, welche ein effizientes Software-Debugging ermöglichen, zur Verfügung. Auf das Software-Debugging wird in Abschnitt 6.3.4 noch näher eingegangen. Der Prozessor besitzt außerdem Module, die seinen Einsatz in einem Multiprozessor-System möglich machen.

6.1.2 Architektur

Ein Überblick über die Architektur des TriCore1-Mikrokontroller-Kerns ist in Abbildung 6.1 dargestellt. Wie oben bereits erwähnt wurde, besitzt der TriCore1 getrennte Schnittstellen für Programm- (PMI) und Datenspeicher (DMI). Daran angeschlossen sind die beiden schnellen On-chip-Speicher P-MEM und D-MEM, die auch als *Scratchpad-RAMs (SPR)* bezeichnet werden. Die Caches des Prozessors nehmen Teile des Scratchpad-RAMs ein.

Die Größe der SPRs sowie der Caches und die Verfügbarkeit einer MMU ist bei dem IP-Kern frei konfigurierbar. Die SPRs können eine Größe von maximal 64KB

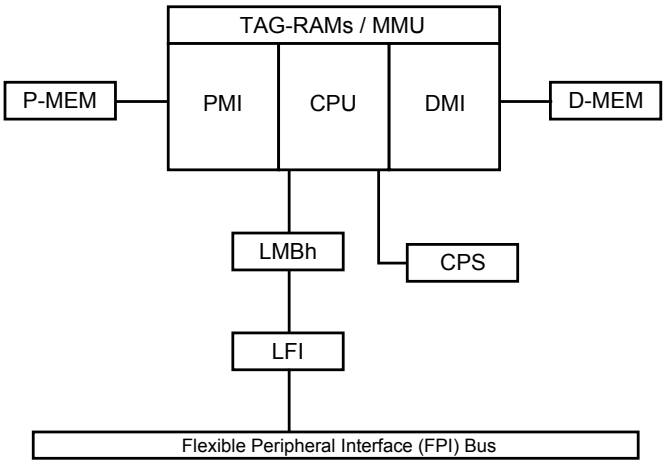


Abbildung 6.1: Architektur des TriCore1-Mikrokontroller-IP-Kerns [51].

annehmen. Davon werden dann je nach Größe der Caches 4, 8 oder 16KB für Caches benötigt.

Der Mikrokontroller verfügt über eine zweistufige Busarchitektur, den *Local Memory Bus (LMB)* und den *Flexible Peripheral Interface (FPI)* Bus. Der LM-Bus ist als Hub (*Local Memory Bus Hub (LMBh)*) implementiert und direkt an die beiden Schnittstellen PMI und DMI angeschlossen. Dieser ist auf Geschwindigkeit optimiert und kann deshalb Module, die schnelle Antwortzeiten benötigen, bedienen. Speicher, der hier angeschlossen ist, kann beispielsweise als 2nd-Level-Cache verwendet werden. Der Bus hat 32-Bit Adress- und 64-Bit Datenleitungen. Die Arbitrierung wird in einem Takt durchgeführt. Der LMBh kann mehrere Master haben sowie im Burst-Modus betrieben werden. Sind langsamere Module angeschlossen, so können diese Wartezyklen einführen.

Der *CPU Processor Slave (CPS)* besteht aus zwei Funktionsblöcken. Ein Block ist für die Unterbrechungsbehandlung (*Interrupt Control Unit (ICU)*) zuständig und bietet eine Schnittstelle zwischen CPU und Unterbrechungssystem an. Die ICU kann bis zu 255 Unterbrechungen behandeln. Der zweite Block fungiert als Debugging-Schnittstelle für die Software-Entwicklung. Der Block kann dabei Echtzeittraces des Programms und der Daten über eine spezielle Schnittstelle nach außen liefern.

Das *LMB to FPI Interface (LFI)* schließlich entkoppelt den LM-Bus und den FPI-Bus über FIFOs. Dadurch können beide Busse mit einer unterschiedlichen Taktfrequenz betrieben werden. Das Modul bietet Einzel- und Burstzugriffe vom LMB zum FPI und umgekehrt, sowie Transaktionen im Pipeline-Modus auf beiden Bussen an.

6.1.3 Anbindung von Peripheriemodulen

Wie oben bereits erwähnt wurde, können zusätzliche Peripheriemodule sowohl am LMBh als auch am FPI-Bus angeschlossen werden. Da langsamere Module am LMBh allerdings das Gesamtsystem verlangsamen können, sollten diese besser am FPI-Bus angebunden werden. Dieser kann wahlweise mit halber oder voller CPU-Frequenz betrieben werden. Dies ist möglich, da er durch das LFI vom LMBh entkoppelt ist.

Abbildung 6.2 zeigt beispielhaft den Aufbau eines FPI-basierten Bus-Systems mit mehreren Modulen. Der FPI-Bus ist ein On-chip-Bus, der Datentransfers zwischen unterschiedlichen Busagenten zulässt. Der Adressbus ist 32 Bit breit. Die Breite des Datenbusses ist konfigurierbar und kann 16, 32 oder 64 Bit betragen. Am FPI-Bus können beliebig viele Peripheriemodule angeschlossen werden. Bis zu 16 von ihnen können auch als Master auf dem Bus agieren, d.h. sie können Transaktionen auf dem Bus starten. Es sind 8, 16, 32, 64 Bit sowie Blocktransfers von 2, 4 oder 8 Worten möglich.

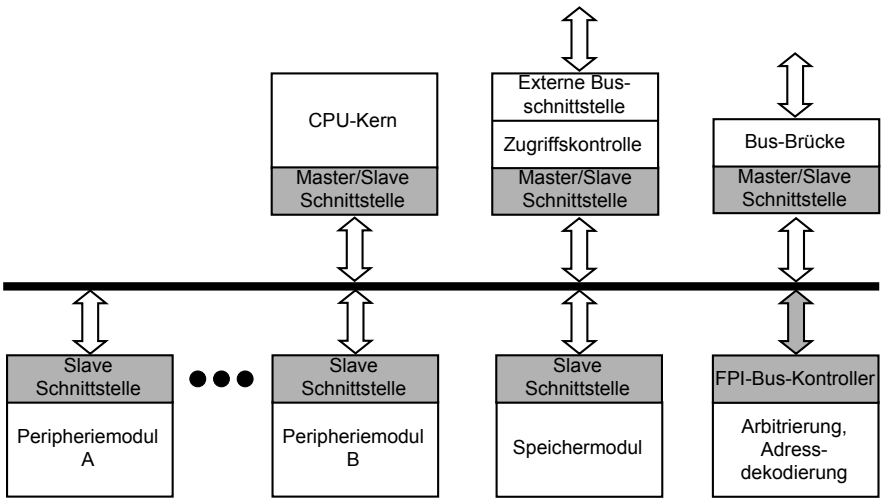


Abbildung 6.2: Beispiel eines FPI-Bus-basierten SoC mit unterschiedlichen Peripheriemodulen [51].

In Abbildung 6.2 ist ein FPI-Bus-System mit einem Busmaster, 3 gemischten Master/Slave-Modulen und 3 reinen Slave-Modulen dargestellt. Der FPI-Bus-Kontroller ist ein reines Master-Modul und kontrolliert die Bustransfers auf dem FPI-Bus. Er ist für die Arbitrierung, Adressdekodierung und Kollisionsauflösung zuständig. Master/Slave-Module sind Module, die sowohl als Master als auch als Slave auf dem Bus agieren können. Als Master agierend können sie Transaktionen initiieren, Transaktionen aufsplitten sowie eine Fehlerbehandlung durchführen.

Holt die CPU ein Datum aus dem Speicher, so agiert sie als Master am FPI-

Bus und initiiert einen entsprechenden Lesebefehl auf dem Bus. Liest ein Software-Debugger ein Register der CPU, dann handelt die CPU als Slave, der die Leseanfrage beantwortet. Gleiches gilt für Schreibzugriffe auf dem Bus. Die DesignWare-StarIP-Komponente des TriCore1 besitzt eine AMBA-Brücke für den bei SoCs weit verbreiteten AMBA-Bus. Diese Bus-Brücke kann ebenfalls als Master oder Slave am FPI-Bus agieren.

Slave-Module schließlich können keine Transaktionen auf dem FPI-Bus durchführen. Sie können lediglich Lese- oder Schreibzugriffe beantworten. Beispiele für Slave-Module sind Speicher, E/A-Schnittstellen oder Zeitgeber.

6.1.4 Programmiermodell und Unterbrechungsbehandlung

Der TriCore1 besitzt 16 Adress- und Datenregister. Die Adress- und Datenregister Nummer 15 werden als implizite Register verwendet. Das Adressregister 10 nimmt den aktuellen Stackpointer und Adressregister 11 die aktuelle Rücksprungadresse eines Unterprogrammaufrufs auf. Programmstatusinformationen werden in zwei 32 Bit Registern, mit Namen: *Previous Context Information (PCXI)* und *Programme Status Word (PSW)* gehalten. Darüber hinaus gibt es einen Programmzähler (*PC*). Der Befehlssatz unterstützt absolute, pre- sowie postinkrementelle Adresssierung und die Adressierung über Basisadresse und Offsets.

Abbildung 6.3 zeigt den Adressraum des TriCore1. Der Prozessor hat bis zu 4GB einheitlichen Programm- und E/A-Speicher, der in 16 Segmente aufgeteilt ist. Die oberen vier Bits der Adresse bestimmen das Segment.

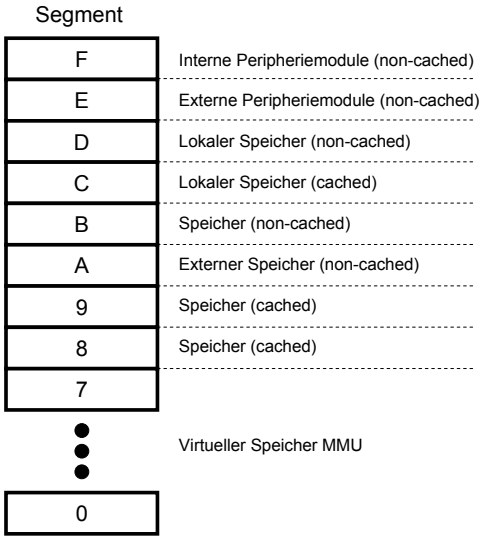


Abbildung 6.3: Aufteilung des Adressraums des TriCore1-Prozessors.

Das TriCore1-Unterbrechungssystem ist sehr effizient aufgebaut. Es besteht aus den programmierbaren *Service Request Nodes (SRN)* und der ICU des oben erwähnten *CPU Processor Slave (CPS)*. Eine Unterbrechnungsanfrage kann von einem Onchip-Modul, externer Hardware oder durch Software initiiert werden. Die Unterbrechungsanfrage wird dann durch einen Sprung in eine Vektortabelle im Programmspeicher beantwortet. Da Kontextwechsel beim TriCore1 durch die Hardware unterstützt werden, reduzieren sich die Antwortzeiten für Unterbrechungsanfragen um drei Taktzyklen im Vergleich zu herkömmlichen Verfahren. Insgesamt stehen dem Programmierer bis zu 255 unterschiedliche Unterbrechungsquellen zur Verfügung. Dabei sind die Unterbrechungsprioritäten frei programmierbar. Diese reichen von einer Prioritätsstufe mit 255 Quellen bis zu 255 Prioritätsstufen mit jeweils einer Quelle.

6.1.5 TriCore1-Plattformen

Wie oben bereits erwähnt wurde, handelt es sich beim TriCore1 um einen Prozessorkern, der am FPI-Bus flexibel um zusätzliche Module erweitert werden kann. Dadurch lassen sich für verschiedene Anwendungsgebiete unterschiedliche Plattformen definieren. Die Chips sind allerdings allesamt als Hardmakro ausgeführt, wodurch sie nicht durch neue Hardware-Komponenten erweitert werden können. Als Beispiele sollen hier der TC1920 und der TC1130 dienen.

Hardmakros

Das TC1920-Hardmakro wird beispielsweise im Automobilsektor als 32 Bit Telematikkontroller eingesetzt. Dementsprechend verfügt der Chip über eine CAN-Schnittstelle. Für die Wandlung von Analogsignalen stehen ADC-Module zur Verfügung. Für die Kommunikation mit mobilen Endgeräten ist eine IrDa-Schnittstelle implementiert worden.

Das TC1130-Hardmakro soll dagegen in der Industrieautomation und Telekommunikation eingesetzt werden. Es bietet eine Gleitkommaeinheit (FPU) und eine Memory Managament Unit (MMU) an. Dadurch ist es möglich, auf dem Prozessor das Betriebssystem Linux zu betreiben. Für die Kommunikation werden Ethernet und USB unterstützt.

Softmakros

Der TriCore1 ist darüber hinaus als Softmakro, d.h. als synthetisierbarer Prozessorkern (TC1MP-S), verfügbar. Das Softmakro lässt sich sowohl in Bezug auf die Größe des SPR sowie der Caches als auch bezüglich der Integration einer MMU kon-

figurieren. Durch eine AMBA-Bus-Brücke ist die Anbindung weiterer Hardware-Komponenten auf AMBA-Bus-Basis möglich.

Damit stellt das TC1MP-S-Softmakro ein ideales Beispiel dar, anhand dessen sich die Vorzüge der in Kapitel 5 vorgestellten Methoden zur Entwicklung eingebetteter Hardware/Software-Systeme zeigen lassen. Beim TC1MP-S handelt es sich um einen komplexen ASIC-IP-Kern, für den einem potentiellen Kunden eine relativ kostengünstige Entwicklungsplattform zur Verfügung gestellt werden muss, um dessen Akzeptanz am SoC-IP-Markt zu steigern.

Darüber hinaus muss die effiziente parallele Entwicklung neuer Hardware-Komponenten und ihrer hardware-nahen Software für TriCore1-basierte SoCs ermöglicht werden. Für die Anbindung zusätzlicher Hardware-Komponenten besitzt der Prozessor den FPI-Bus. Für die Software-Entwicklung sollen herkömmliche Cross-Entwicklungsumgebungen verwendet werden, um dem Entwickler eine transparente Sicht auf das System zu bieten.

6.2 Rapid-Prototyping-System Spyder

Auf dem Markt sind einige Boards unterschiedlicher Architektur verfügbar, die sich mehr oder weniger gut für das in dieser Arbeit behandelte Themengebiet eignen. Boards, die beispielsweise mit mehreren FPGAs bestückt sind, zeichnen sich durch hohe Kosten aus, da die FPGAs einen wesentlichen Teil der Gesamtkosten des Prototyping-Systems ausmachen.

Die optimale Architektur eines Prototyping-Systems für SoC-Entwürfe, die in Abschnitt 5.2.3 eingeführt wurde, soll exemplarisch am Beispiel des Spyder-Systems gezeigt werden [109]. Abbildung 6.4 zeigt ein Blockschaltbild des Spyder Rapid-Prototyping-Systems in der Version 2.

Die Architektur der Version 2 des Spyder-Boards unterscheidet sich im Wesentlichen nur durch den Tausch der FPGA-Technologie von der Version 1. In der Version 1 des Systems ist das Board maximal mit einem Xilinx VirtexE-XCV2000E-FPGA verfügbar. Wie dieses System für das Prototyping einer komplexen Mikrokontroller-IP-Komponente eingesetzt werden kann, wird in Abschnitt 6.3.2 näher beschrieben.

An dieser Stelle soll das Spyder-Systems in der Version 2 betrachtet werden, da dieses mit der neuesten Generation verfügbarer Xilinx FPGAs ausgerüstet ist. Dabei handelt es sich um Xilinx Virtex-II-FPGAs, der Baureihen XC2V3000-FF1152 bis XC2V8000-FF1152.

Die zusätzlichen Komponenten des Systems lassen sich in Gruppen einteilen. Für die Emulation von Speicher eines SoC-Entwurfs lassen sich jeweils zwei Bänke mit synchronem SRAM und SDRAM verwenden. Da für das synchrone SRAM keine all-

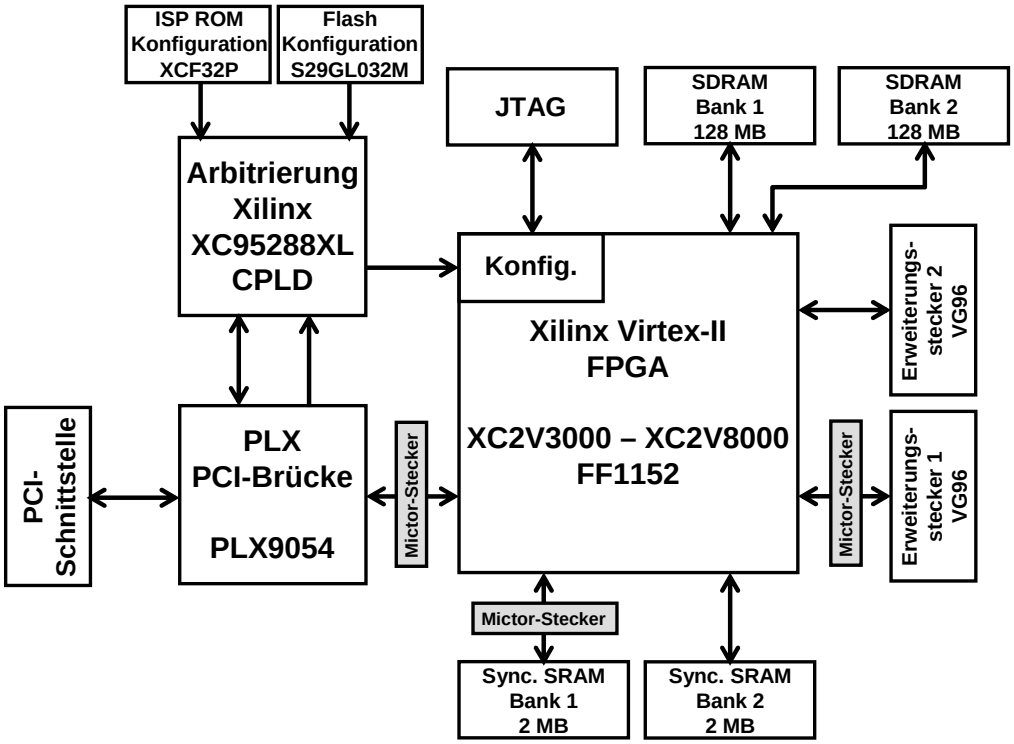


Abbildung 6.4: Blockschaltbild des Spyder Rapid-Prototyping-Systems in der Version 2.

zu große zusätzliche Logik im FPGA benötigt wird, empfiehlt sich dieses besonders für die Emulation von sehr ressourcenintensiven Systemen.

Das Board kann über eine PCI-Schnittstelle in einem PC betrieben werden. Über die Schnittstelle ist es sowohl möglich, das FPGA zu konfigurieren, als auch mit dem FPGA zu kommunizieren. In Abschnitt 5.2.5 wurde bereits beschrieben, wie mit Hilfe einer PCI-Schnittstelle eine eingeschränkte Software-Entwicklung durchgeführt werden kann. Diese wurde exemplarisch für das Spyder-System implementiert [87].

Das Spyder-System verfügt über zwei Erweiterungsstecker, über die das System um zusätzliche Hardware oder zusätzliche Spyder-Boards erweitert werden kann. Die Schnittstelle kann aber auch für das Debugging mit einem Logikanalysator verwendet werden. Weitere Debugging-Möglichkeiten für die Hardware bestehen über JTAG mit der ChipScope Pro Software von Xilinx [110] sowie über Mictor-Stecker von Agilent [1] für externe Logikanalysatoren.

Das FPGA kann neben der oben bereits erwähnten PCI-Schnittstelle auch über JTAG sowie über die auf dem Board verfügbaren ISP-ROM-Chips konfiguriert wer-

den. Die ISP-ROM-Chips bieten dabei die Möglichkeit, verschiedene Konfigurationen des SoC-Entwurfs zu speichern. Die Auswahl welcher Entwurf beim Booten des Systems geladen wird, erfolgt dann über Dip-Schalter.

6.3 Emulation des TriCore1-Mikrokontroller-IP-Kerns

In diesem Abschnitt wird die Entwicklung eingebetteter Software mit dem TC1MP-S auf der Basis des Prototyping-Systems Spyder beschrieben. Dazu muss der Mikrokontroller-IP-Kern auf dem Spyder-System integriert und eine Software-Entwicklungsumgebung angebunden werden. In Abschnitt 5.2 wurden Probleme aufgezeigt, die beim Prototyping von ASIC-IP-Komponenten auf FPGAs auftreten können. Abschnitt 6.3.1 schildert zunächst technologische Anpassungen des TC1MP-S für die Integration auf dem Spyder-System und Abschnitt 6.3.2 beschäftigt sich dann mit der Lösung des Ressourcenproblems bei der Verwendung eines Spyder-Boards in der Version 1. Für die transparente Software-Entwicklung wurde der komplette TriCore1-IP-Kern auf einem Spyder-System der Version 2 integriert und eine Software-Entwicklungsumgebung an das Prototyping-Board angebunden. Diese Integration und die Software-Entwicklung ist Gegenstand von Abschnitt 6.3.4.

6.3.1 Technologische Anpassung des ASIC-IP-Kerns

Einige der in Abbildung 5.2 auf Seite 56 aufgezeigten Probleme aufgrund technologischer Unterschiede zwischen ASICs und FPGAs treten auch beim TC1MP-S auf. Der TC1MP-S ist in VHDL spezifiziert. Der Mikrokontroller-Kern besteht aus über 400 einzelnen Modulen, die sich grob in zwei Submodule unterteilen lassen. In Abbildung 6.5 ist die grobe Modulstruktur des TC1MP-S dargestellt. Die *RedBox* enthält den inneren Kern mit der CPU, den Schnittstellen zu den Programm- und Datenspeichern sowie die TAG-RAMs für Caches und die MMU, falls vorhanden.

In der *BlueBox* befinden sich Module zur Kontrolle des LM-Busses (LMBh), und das CPS-Modul (*CPU Slave*) für die Unterbrechungsbehandlung und das Software-Debugging. Das Modul LFI stellt eine Brücke zwischen LM-Bus und dem Peripheriebus LFI dar. Der FPI-Bus-Kontroller (*Bus Control Unit (BCU)*) und der Cerberus werden für die Kontrolle des FPI-Busses bzw. das Software-Debugging verwendet.

Insgesamt beinhalten die 400 Module zusammen ca. 180.000 Code-Zeilen und die Anzahl der E/A-Anschlüsse des Kerns beläuft sich auf über 3.300. Dementsprechend komplex sind auch die Syntheseskripte für den DesignCompiler® der Firma Synopsys, die zusammen ca. 111.992 Zeilen aufweisen.

Auf der rechten Seite von Abbildung 6.6 ist die Syntheseumgebung für die Implementierung von Hardware-Komponenten auf dem Spyder-System dargestellt. Der

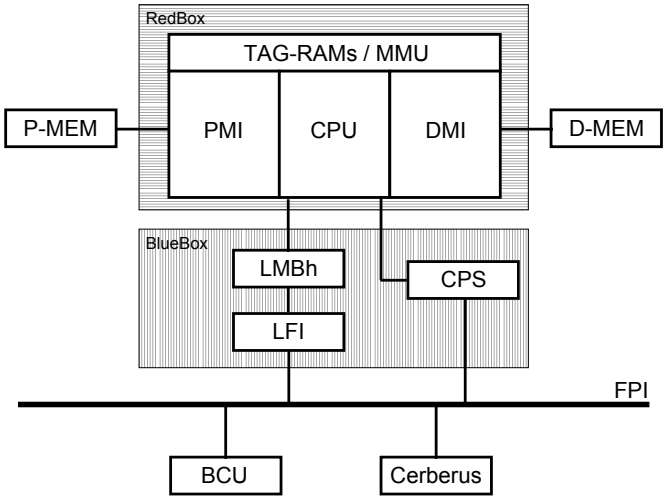


Abbildung 6.5: Modulstruktur des TriCore1 TC1MP-S-Softmakros.

Syntheseablauf für den TC1MP-S-Kern auf der linken Seite musste auf diese Syntheseumgebung angepasst werden. Als Frontend für die VHDL-Quelldateien wurde der FPGA Compiler II der Firma Synopsys verwendet, der speziell für die Synthese für FPGAs entwickelt wurde. Dieser liefert als Ausgabe eine Netzliste, die schon FPGA-spezifische Eigenschaften aufweist. An dieser Stelle der Synthese können bereits Aussagen über die maximal mögliche Taktfrequenz oder die Anzahl der benötigten Taktnetze getroffen werden.

Die Netzliste wird dann von den Backend-Werkzeugen der Firma Xilinx weiterverarbeitet und in eine Konfigurationsdatei für das FPGA überführt. `ngdbuild` übersetzt Netzlisten in ein internes Format von Xilinx, überprüft die Zuweisung der E/A-Anschlüsse in einer *User Constraint File (UCF)*-Datei und fügt Module hinzu, die innerhalb des VHDL-Quelltextes instanziiert, aber nicht spezifiziert wurden. Dies können z.B. Block-RAMs der VirtexII-FPGAs sein. Mit Hilfe der UCF-Datei lassen sich auch Rahmenbedingungen bezüglich der Platzierung einzelner Module auf dem FPGA machen. `map` bildet dann zunächst logisch die einzelnen Funktionseinheiten der Implementierung auf Funktionseinheiten des FPGAs ab. Anhand der Ergebnisse dieser Synthesestufe lassen sich dann Aussagen über die Auslastung des FPGAs machen. `par` platziert und verdrahtet schließlich die Funktionseinheiten innerhalb des FPGAs. Erst nach dieser Stufe steht fest, ob der Entwurf bei vorgegebener Taktfrequenz auf dem FPGA korrekt realisiert werden kann oder nicht. Mit `bitgen` wird dann noch der synthetisierte Entwurf in eine Konfigurationsdatei für die jeweilige FPGA-Familie überführt.

Im Rahmen des oben geschilderten Syntheseablaufs können in jeder Stufe Pro-

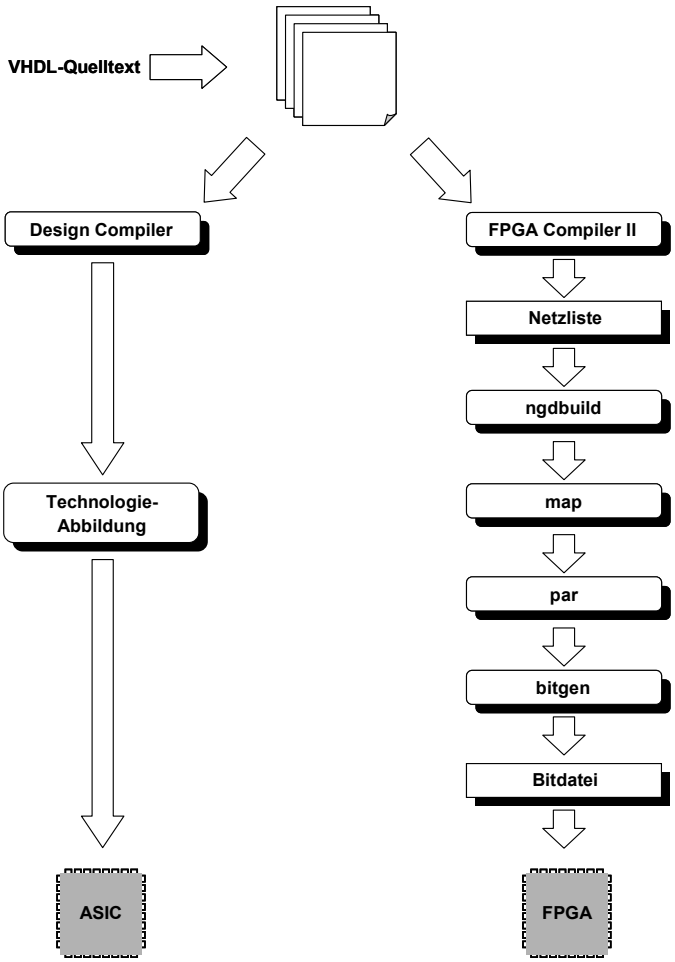


Abbildung 6.6: Synthese des TriCore1 VHDL-Quelltextes für Xilinx VirtexII-FPGAs.

bleme auftreten, die bei der Intergration eines ASIC-IP-Kerns auf einem FPGA typisch sind. So können beispielsweise im VHDL-Quelltext Konstrukte verwendet werden, die das Frontend, in diesem Fall der FPGA Compiler II, nicht synthetisieren kann. Im Fall des TC1MP-S werden für die Implementierung des Kerns VHDL `configurations` verwendet, die der FPGA Compiler II nicht verarbeitet. Der Quelltext musste dahingehend verändert werden, dass keine Konfigurationen mehr verwendet werden. Dazu wurden die Konfigurationen aus dem Quelltext entfernt und eine eindeutige Zuordnung der Schnittstellen (`entity`) eines Moduls zu ihrer Architektur (`architecture`) vorgenommen.

Der FPGA Compiler II kann darüber hinaus keine komplexen Datentypen verarbeiten, die im `generic`-Teil einer `entity` spezifiziert sind. Um dieses Problem zu umgehen, wurden die `generic`-Teile aus dem VHDL-Quelltext entfernt und durch globale Konstanten in einem `package` ersetzt. Beispielsweise wird beim TC1MP-S

die Größe der Caches mit Hilfe von `generics` definiert. Im Quelltext des Kerns für die FPGAs werden nun je nach gewünschter Cachegröße bei der Synthese unterschiedliche `packages` verwendet.

Für die Scratchpad-RAMs werden spezielle Speicher von Infineon verwendet, die in einer Bibliothek abgelegt sind. Diese Speicherbausteine werden im TC1MP-S-VHDL-Quelltext referenziert und müssen deshalb für FPGAs ersetzt werden. Im vorliegenden Fall wurden die Speicherbausteine durch die schnellen On-chip-FPGA Block-RAMs ersetzt.

Als wesentlich aufwändiger gestaltet sich die Transformation von VHDL-Quelltext, der ASIC-spezifische Architekturmerkmale beschreibt. So gibt es beispielsweise unterschiedliche Taktdomänen für den schnelleren Prozessorkern sowie für den langsameren Peripheriebus FPI. An der Schnittstelle zwischen beiden Taktdomänen (LFI) muss gewährleistet werden, dass die Daten zwischen beiden Domänen korrekt übergeben werden. Für die Implementierung verschiedener Taktdomänen stehen dem Entwickler bei ASICs spezielle Synthesewerkzeuge zur Verfügung, die dies bewerkstelligen. Speziell beim TriCore1 können die beiden unterschiedlichen Taktdomänen des Prozessorkerns und Peripheriebusses auf zwei der acht bei VirtexII-FPGAs vorhandenen dedizierten Taktdomänen abgebildet werden.

Wesentlich komplexer sind dagegen bedingte Takte, die bereits in Kapitel 5, Abschnitt 5.2.1, erwähnt wurden. Beim TriCore1 kann der Takt für verschiedene Komponenten zur Stromeinsparung abgeschaltet werden. Dies ist beim TriCore1 über bedingte Takte realisiert. Aufgrund der Größe des VHDL-Quelltextes ist eine manuelle Konvertierung so, dass die Taktstruktur korrekt in FPGAs umgesetzt wird, mit vertretbarem Aufwand nicht möglich. Hierfür ist dann Werkzeugunterstützung notwendig [98]. Da solche Werkzeuge im vorliegenden Fall nicht vorhanden waren, musste eine andere Lösung gefunden werden. Deshalb wurden die Signale zum Abschalten des Taktes im VHDL-Entwurf abgefangen und dahingehend fest verdrahtet, dass sie sich immer im inaktiven Zustand befinden.

6.3.2 Realisierung komplexer SoC-Prototypen unter harten Ressourcenbeschränkungen

Das Spyder-System der ersten Generation ist mit Xilinx VirtexE-FPGAs ausgestattet. Mit dem aus dieser Familie größten FPGA, einem XCV2000E-Chip, sind theoretisch maximal 19.200 Slices, 4 Taktdomänen und 404 E/A-Anschlüsse für das Prototyping verfügbar. Aufgrund der generischen Architektur von FPGAs und dem festen Layout des Boards stellen diese Werte allerdings nur die theoretische Obergrenze dar. Praktisch kann beispielsweise eine hundertprozentige Auslastung der Slices des FPGAs nicht erreicht werden, da die Platzierung und Verdrahtung der Funktionseinheiten bei

diesem hohen Integrationsgrad nicht mehr möglich ist. Dies wird durch die Tatsache noch verstärkt, dass bestimmte E/A-Anschlüsse durch ihre Verbindung mit externen Komponenten (z.B. SRAM-Chips) nicht frei wählbar sind.

In Tabelle 6.1 sind Syntheseergebnisse für einige wichtige Submodule und die gesamte DesignWare-StarIP-Komponente des TriCore1 angegeben. Zeile 1 zeigt den Ressourcenverbrauch des gesamten IP-Kerns. In Zeile 2 ist der Ressourcenverbrauch des inneren CPU-Kerns und in Zeile 3 für wichtige Peripheriekomponenten des TriCore1 dargestellt. Die Anzahl der Slices repräsentiert die benötigte Menge der frei programmierbaren Logikressourcen. Die Spalte der Gatteräquivalente gibt an, wieviele Gatter die Komponente als ASIC-Realisierung benötigen würde. Die Zahlen werden von dem Werkzeug `map` von Xilinx ausgegeben. Bei der Synthese wurde auf Geschwindigkeit (G) und auf Fläche (F) optimiert.

	# Slices	# E/A-Signale	# Taktdomänen	Gatteräquivalente
TC1MP-S	23.650(G) 22.615(F)	3.376	4	369.707(G) 357.166(F)
TC1 RedBox	18.954(G) 18.207(F)	2.815	1	275.855(G) 267.271(F)
TC1 BlueBox	3.917(G) 3.636(F)	1.149	4	60.840(G) 57.819(F)

Tabelle 6.1: Syntheseergebnisse für den TC1MP-S und Submodule für Xilinx Virtex-2000E-FPGAs.

Wie aus der Tabelle leicht ersichtlich ist, reichen die Ressourcen eines XCV2000E FPGAs nicht aus, um eine minimale Version eines TC1MP-S-Kerns zu implementieren. Eine Partitionierung des Designs gestaltet sich aufgrund der hohen Anzahl der E/A-Signale der RedBox und BlueBox als sehr schwierig. Eine alternative Lösung für das hier auftretende Ressourcenproblem stellt die Integration des TC1MP-S-Kerns mit Hilfe der Partialemulation dar [88]. Die Idee der Partialemulation beruht auf der optimalen Nutzung der Ressourcen des Prototyping-Boards. Ressourcen stehen im FPGA in Form von Block-RAMs, Mutliplizierern, speziellen E/A-Komponenten und Taktdomänen zur Verfügung. Außerhalb des FPGAs bietet das Spyder-System RAM-Bausteine, Schnittstellen, Erweiterungsstecker und unterschiedliche Taktgeber an.

Integration der TriCore1-IP auf dem Spyder-Virtex X2E V1.3 Board

Da es von vorneherein äußerst schwierig ist, die RedBox oder BlueBox des TriCore1 auf dem System zu integrieren, musste zunächst eine geeignete Startimplementierung gewählt werden, die dann sukzessive um weitere Module ergänzt werden kann. Die kleinste sinnvolle Funktionseinheit eines Mikrokontrollers ist die CPU selbst. Da eine CPU ohne Programm- und Datenspeicher keine Programme ausführen kann, muss das CPU-Modul um entsprechende Baugruppen erweitert werden. Werden die Speicher mit den programmierbaren Logikressourcen des FPGAs implementiert, werden sehr viele Ressourcen verbraucht, da ein Slice nur 2 Bits an Daten halten kann.

FPGAs der neueren Generation haben für diesen Fall spezielle RAM-Module. Mit den 160 Select-RAM-Blöcken der XCV2000E-FPGAs können 82KB an Speicher implementiert werden. Für die TriCore1-CPU wurden deshalb spezielle Speicherschnittstellen implementiert, die auf das Block-RAM der FPGAs zugreifen können. Die initiale Architektur ist in Abbildung 6.7 (a) aufgezeigt. Die schraffierten Module sind die aus der TC1MP-S-Komponente übernommenen Teile.

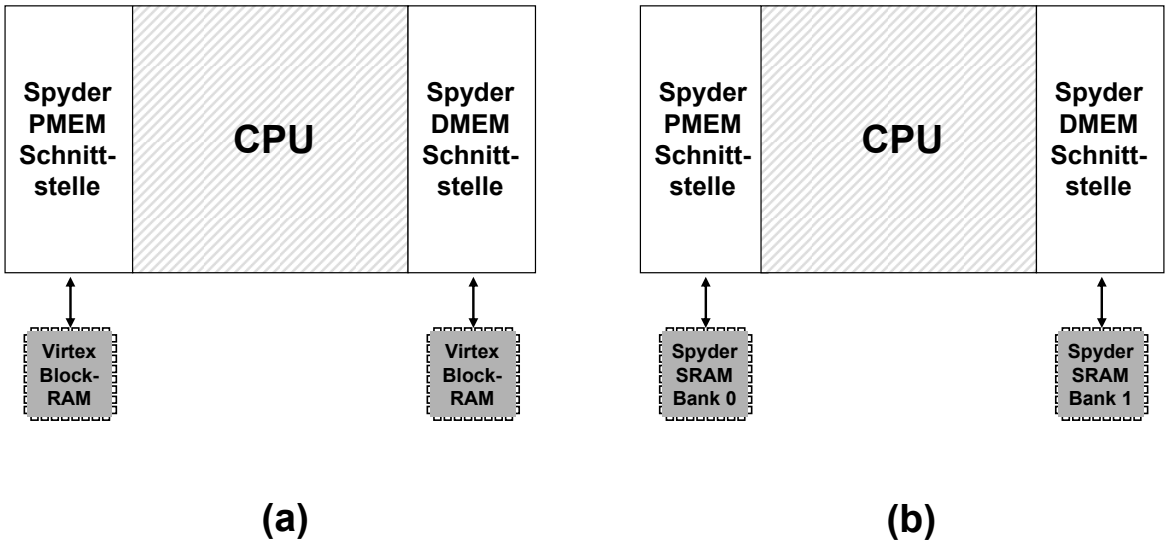


Abbildung 6.7: TriCore1-CPU-Modul mit unterschiedlichen Speicheranschlüssen.

Abbildung 6.7 (b) zeigt eine alternative Lösung einer minimalen TriCore1-Integration auf dem Spyder-System der ersten Generation. Die Verwendung der Spyder SRAMs bietet gegenüber der Variante 6.7 (a) einige Vorteile. Zunächst müssen für die Emulation unterschiedlicher Programme die Block-RAMs mit unterschiedlichen Daten gefüllt werden. Dies kann vor der Synthese durch Verwendung des Programms `data2mem` erfolgen. Das Programm `data2mem` kann aber auch eine bereits erzeugte Bitdatei nachträglich mit neuen Daten konfigurieren. Beide Varianten sind

allerdings recht aufwändig, da für die Block-RAMs Konfigurationsdateien erzeugt werden müssen. Darüber hinaus ist die Kapazität der Block-RAMs auf 82KB begrenzt, wodurch die Emulation großer Programme nicht möglich ist. Ein weiterer Nachteil, der durch die Verwendung der Block-RAMs entsteht, ist die erschwerte Verdrahtung des Gesamtsystems. Darüber hinaus können die Block-RAMs nicht für das Hardware-Debugging mit integrierten Logikanalysatoren, wie beispielsweise ChipScope Pro, verwendet werden.

Die Spyder SRAMs bieten dagegen pro Bank bis zu 2MB an Speicherkapazität an. Es wurde deshalb für die SRAM-Chips Schnittstellencode geschrieben, der die CPU des TriCore1 bedient. Die Emulation von Programmen mit diesem System führte zu einer signifikanten Geschwindigkeitssteigerung gegenüber der RTL-Simulation, hat allerdings noch einige Nachteile.

Zum Einen sind die Schnittstellen zwischen der CPU und den Speicherbausteinen sehr komplex und schlecht dokumentiert, was die Implementierung der Schnittstellenmodule `SpyderPMEM` und `SpyderDMEM` sehr fehleranfällig macht. Zum Anderen fehlt bei den in Abbildung 6.7 dargestellten Lösungen eine Schnittstelle für die Anbindung zusätzlicher Peripheriemodule, um das Prototyping eines komplexen SoC zu ermöglichen. Für die Entwicklung von Software steht darüber hinaus keine Debugging-Möglichkeit zur Verfügung und der Entwickler bekommt auch keinerlei Rückmeldung vom System. Das Debugging ist so auf die Hardware beschränkt. Ein Modul zur Unterstützung von Unterbrechungen fehlt ebenfalls.

Die in Abbildung 6.7 (b) vorgestellte Lösung benötigt bei einer flächenoptimierten Synthese 13.632 der 19.200 bei XCV2000-FPGAs verfügbaren Slices. Da ein wesentlicher Bestandteil eines SoC neben dem Mikrokontroller zusätzliche Peripheriemodule sind, wurde das oben vorgestellte System um zusätzliche Module erweitert, die deren Anbindung ermöglichen sollen.

Beim TriCore1 werden Peripheriemodule in den Speicherbereich des Kontrollers eingeblendet. Deshalb mussten Veränderungen auf der Datenseite der CPU vorgenommen werden. Da die Datenspeicherschnittstelle (*Data Memory Interface (DMI)*) des TriCore1 eine LMB-Schnittstelle besitzt, wurde in einem nächsten Schritt dieses Modul hinzugefügt. Die Architektur dieser Lösung ist in Abbildung 6.8 dargestellt.

Vor die `SpyderDMEM`-Schnittstelle und das DMI wurde ein Multiplexer eingefügt, der Lese- und Schreib-Anforderungen der CPU auf Adressen `0xD0xx_xxxx` an die `SpyderDMEM`-Schnittstelle weiterleitet. Die Adresse entspricht dem Bereich des Scratchpad-RAMs des Prozessors (siehe auch Abbildung 6.3). Entsprechend werden Anfragen auf Adressen `0xAxxx_xxxx` an die DMI-Schnittstelle gegeben, die dann die entsprechenden LM-Bus-Signale erzeugt, da dieser Adressbereich auch bei der TriCore1-IP auf dem LM-Bus liegt. Aufgrund der Tatsache, dass das Scratchpad-RAM an der `SpyderDMEM`-Schnittstelle implementiert ist, mussten die SPR-

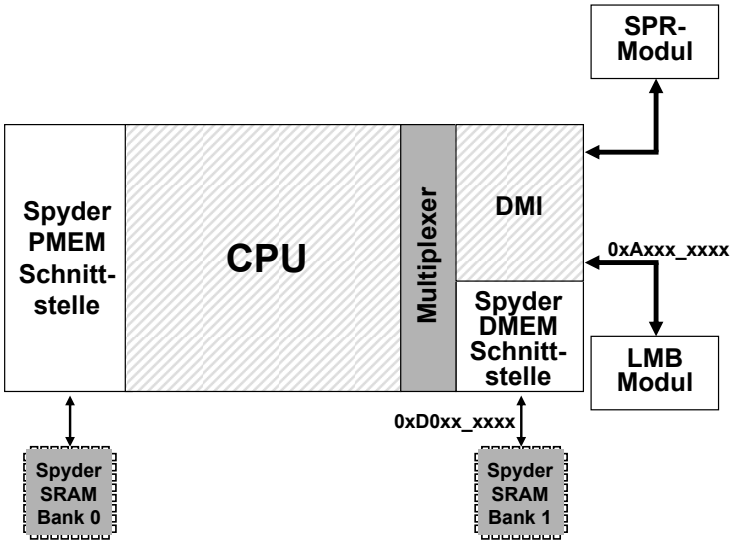


Abbildung 6.8: TriCore1-CPU-Modul mit einfacher LMB-Schnittstelle.

Signale des DMI mit einem speziellen SPR-Modul abgefangen bzw. mit vorgegebenen Werten getrieben werden. Das SPR-Modul stellt an dieser Stelle noch keine Implementierung der Scratchpad-RAMs des Mikrokontrollers dar.

An die DMI-Schnittstelle kann ein LMB-Modul angeschlossen werden, welches unter anderem mit der Außenwelt kommunizieren kann. Dadurch ist es möglich, im Programm auf eine entsprechende Adresse des LM-Bus zu schreiben und so Signale aus dem System auszugeben. Der Entwickler erhält dadurch beispielsweise die Möglichkeit, Informationen zum Ende eines Programmes aus dem System auszulesen. Die Lösung hat allerdings den Nachteil, dass keine weiteren Module an den LM-Bus angeschlossen werden können. Darüber hinaus ist der Schnittstellencode zwischen CPU und Datenschnittstelle durch die Hinzunahme des Multiplexers komplexer geworden. Ein effizientes Software-Debugging, sowie die Verwendung von Unterbrechungen, ist auch mit dieser Lösung nicht möglich.

Da die in Abbildung 6.8 dargestellte Lösung 15.904 Slices benötigt, kann die Implementierung um zusätzliche Funktionseinheiten erweitert werden. Um den Entwickler einen vollständigen LM-Bus zur Verfügung zu stellen, wurde die LMBh-Einheit aus dem BlueBox-Submodule des TC1MP-S herausgelöst und an das vorhandene System angebunden. Da das LMBh-Modul auch eine Schnittstelle zum Programmspeicher-Modul (*Programme Memory Interface (PMI)*) der CPU besitzt, wurde dieses Modul ebenfalls in den Entwurf mit aufgenommen. Der Multiplexer zwischen CPU, DMI und `SpyderDMEM`-Schnittstelle wurde dagegen entfernt, um Ressourcen einzusparen. Die Architektur des neuen Systems zeigt Abbildung 6.9.

Die in den bisherigen Lösungen verwendeten speziellen Anbindungen der Spyder

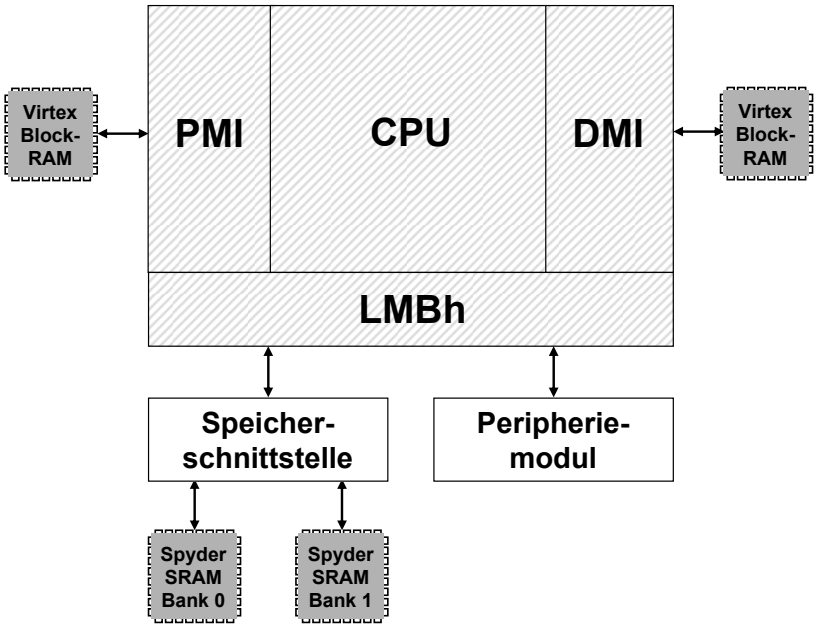


Abbildung 6.9: TriCore1-CPU-Modul mit LMBh-Anbindung und einfacher Speicherhierarchie.

SRAMs an die CPU sind in der neuen Implementierung durch eine einheitliche Speicherschnittstelle ersetzt worden, die die Spyder SRAMs ansteuert und an den LMB-Bus angeschlossen ist. Der Speicher ist dabei in zwei Bänke aufgeteilt, die von der Speicherschnittstelle über unterschiedliche Basisadressen angesprochen werden. Dadurch ergibt sich eine Aufteilung der Programme in Programm- und Datensegmente. An einer weiteren Schnittstelle des LMBh kann ein zusätzliches Peripheriemodul (LMB-PM) angeschlossen werden.

Da die in Abbildung 6.9 dargestellte Lösung die ursprünglichen PMI- und DMI-Module verwendet und diese über E/A-Signale zu den Scratchpad-RAMs des TriCore1 verfügen, werden die SPRs des TriCore1 durch die internen Block-RAMs des XCV2000E emuliert.

In Tabelle 6.2 ist ein Überblick über die für die Implementierung des TC1MP-S-Kerns gewählte Vorgehensweise zu finden. Die Komponenten des Kerns wurden in der Abbildung 6.5 auf Seite 94 bereits vorgestellt. Die Wahl der zu implementierenden Komponenten wird gemäß der Prioritäten durchgeführt. Die Speicherschnittstelle für die Spyder SRAMs sowie das einfache Debugging-Modul bekommen eine höhere Prioritätsklasse als z.B. die Komponente *LFI*, da diese nicht mehr auf dem Prototyping-System Spyder der Version 1 implementiert werden kann. Insgesamt können aufgrund der eingeschränkten Ressourcen nur die Komponenten der Prioritätsklassen drei und kleiner implementiert werden.

Priorität	Komponenten
0	CPU, PMEM, DMEM
1	DMI, PMI
2	LMBh
3	SRAM, LMB-PM
4	LFI, BCU
5	CPS, Cerberus
6	FPI-Peripheriemodule (FPI-PM)
7	TAG-RAMs, MMU

Tabelle 6.2: Einteilung der Komponenten des TC1MP-S-Kerns in Prioritätsklassen.

Ablauf der Emulation

Mit der in Abbildung 6.9 vorgestellten Lösungen können nun Programme entwickelt werden. Da es aufgrund der fehlenden Software-Debugging-Module nicht möglich ist, Programme über eine direkte Verbindung mit einem Software-Debugger auf das Prototyping-System herunterzuladen, wurde eine alternative Entwurfsumgebung implementiert [87]. Diese ist in Abbildung 6.10 dargestellt.

Das in Abschnitt 6.2 vorgestellte Rapid-Prototyping-System Spyder kann über einen PCI-Bus an einen PC angeschlossen werden. Über diesen PCI-Bus ist es sowohl möglich, Konfigurationsdateien für das FPGA herunterzuladen als auch über spezielle PCI-Treiber mit dem FPGA zu kommunizieren. Dazu wird das FPGA in den Adressraum des PCs eingeblendet. Der auf dem FPGA-Board befindliche PCI-Chip übersetzt die PCI-Signale dann in Adress-, Daten- und Kontrollsignale, die an Anschlüssen des FPGAs sichtbar sind.

Das Programm `SpyderSRAMLoader` lädt die SRAMs des Prototyping-Boards über den PCI-Bus mit den Daten aus der ELF-Objektdatei, die zuvor von einem Cross-Compiler für den TriCore1 erzeugt wurde. Dazu muss eine Hardware-Schaltung ins FPGA geladen werden, die die Daten an der PCI-Schnittstelle des FPGAs in Empfang nimmt und in die entsprechende SRAM-Bank schreibt (Abbildung 6.10 (1)). Da die statischen RAMs des Spyder-Systems die Daten halten, solange die Spannungsversorgung aufrecht erhalten bleibt, kann danach die TriCore1-CPU in das FPGA geladen und das Programm aus den SRAMs heraus ausgeführt werden (Abbildung 6.10 (2)).

Abbildung 6.10 zeigt im linken Teil eine Simulationsumgebung, die ebenfalls entwickelt wurde, um das System in der Anfangsphase des Entwurfsprozesses simulieren zu können. Mit Hilfe des Programms `SpyderSRAMGenerator` kann ein

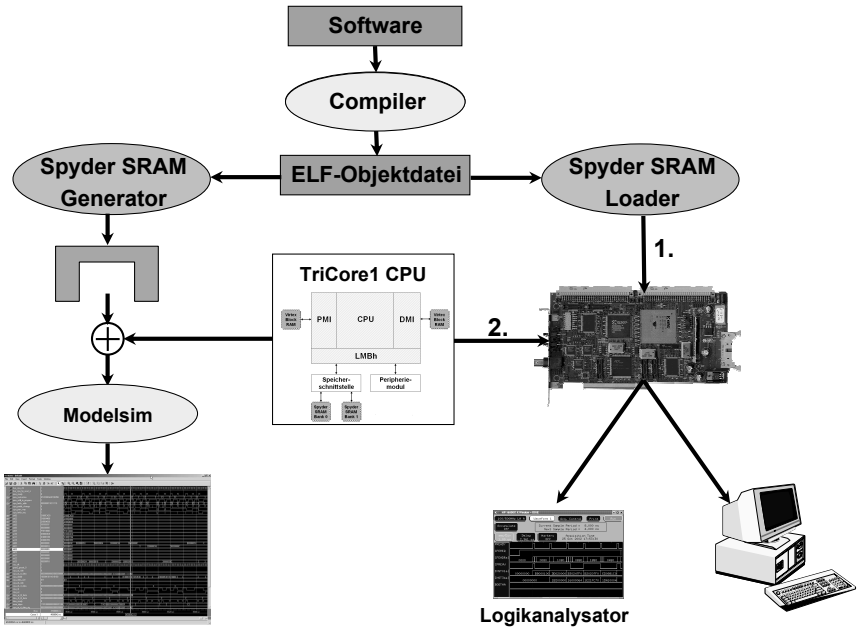


Abbildung 6.10: Entwicklungsumgebung für ein eingeschränktes TriCore1-basiertes SoC.

Programm in VHDL-Simulationsmodelle der Spyder-SRAMs eingefügt werden. Danach kann dann der TriCore1 zusammen mit der so erstellten Testbench in einem kommerziellen HDL-Simulator, wie z.B. Modelsim [68], simuliert werden.

Möglichkeiten des Hardware- und Software-Debugging

Die in diesem Abschnitt aufgezeigte Lösung lässt sich mit den aus dem Hardware-Bereich bekannten klassischen Lösungen des Hardware-Debugging auf Fehler hin untersuchen. So ist es möglich, an den Erweiterungssteckern des Spyder-Boards Signale des Systems mit Hilfe eines Logikanalysators abzugreifen, die im Entwurf herausgeführt wurden. Darüber hinaus kann mit dem von Xilinx verfügbaren internen Logikanalysator ChipScope Pro die Aufzeichnung interner Signale im FPGA selbst vorgenommen werden. Die Realisierung der Entwurfsumgebung stellt diesbezüglich keine Einschränkung dar, sondern bietet darüber hinaus Vorteile. Durch die Implementierung der RAMs in externen Speicherchips sind mehr interne Block-RAMs des FPGAs für die ChipScope Pro Software verfügbar.

Das Debugging von Software ist im Gegensatz zum Debugging der Hardware nur eingeschränkt möglich, da die für das Software-Debugging notwendigen Module des TriCore1 aufgrund der eingeschränkten Ressourcen nicht mit implementiert werden konnten. Dennoch gibt es eine Reihe von Debugging-Möglichkeiten.

Zunächst kann über ein spezielles Modul am LM-Bus des Prozessors der Programmstatus ausgegeben werden, indem auf eine bestimmte Adresse des TriCore1 geschrieben wird, die im Adressraum des LM-Bus liegt. Dieses Modul kann entweder durch an den Erweiterungssteckern herausgeführte Signale einen Logikanalysator ansteuern, oder über den PCI-Bus des Spyder-Boards eine Unterbrechung im PC auslösen. Damit ist es möglich, einfache Statusmeldungen an den Entwickler zu übermitteln.

Eine andere Möglichkeit ist eine serielle Schnittstelle, die am freien Port des LM-Bus angebracht werden kann. Dadurch können komplexere Informationen in einem Hyperterminal auf einem PC angezeigt werden. So können beispielsweise Ergebnisse und Informationen über den Programmablauf ausgegeben werden. Die dafür entwickelte serielle Schnittstelle beansprucht dabei nur 549 Slices und benötigt für die Erzeugung der Baudrate eine zusätzliche Taktdomäne sowie zwei Anschlüsse des FPGAs für Sende- und Empfangsleitung [18].

Eine dritte Möglichkeit stellt die Erweiterung des Emulationsablaufs aus Abbildung 6.10 um eine dritte Stufe dar, in der die Inhalte der beiden SRAMs über den PCI-Bus des Spyder-Boards ausgelesen werden können. Dies ergibt einen Speicherdump des TriCore1 nach der Ausführung eines Programms.

6.3.3 Zwischenergebnis

Mit der durch Partialemulation implementierten Entwicklungsumgebung für eingebettete Software konnte eine signifikante Steigerung der Ausführungsgeschwindigkeit von Programmen erzielt werden. Tabelle 6.3 zeigt Laufzeitmessungen für einige Benchmarks im Vergleich zwischen der RTL-Simulation und der Emulation auf dem Spyder-System.

	Euklid's Algorithmus	Fibonacci-Zahlen	Sieb des Eratosthenes
Anzahl ausgeführter Befehle	1.484	41.419	20.779
RTL-Simulation	ca. 28 Sec.	ca.10 Min.	ca. 18 Min.
Emulation auf dem Spyder-System	312 μs	3.9 ms	21.8 ms

Tabelle 6.3: Laufzeitvergleich von Software-Benchmarks zwischen RTL-Simulator und dem TriCore1 auf dem Spyder-Board der ersten Generation.

Die Messungen wurden im Fall des RTL-Simulators auf einer Sun Blade® 100 mit einem UltraSPARC-2e Prozessor mit 500 MHz aufgenommen. Die Emulation des TriCore1 auf dem Spyder-Board wurde mit einer Taktfrequenz von 8 MHz durchgeführt, die maximal erzielt werden konnte.

Euklid's Algorithmus berechnet den größten gemeinsamen Teiler zweier Zahlen. Im vorliegenden Fall wurden insgesamt 1.484 Befehle ausgeführt. Der zweite Benchmark berechnet die ersten 10.000 Fibonacci-Zahlen. Der Algorithmus Sieb des Eratosthenes berechnet die ersten 1.000 Primzahlen, indem er in einem 1.000 Einträge großen Feld im Speicher alle Vielfachen der ersten 500 Zahlen markiert. Die in dem Feld nicht markierten Einträge repräsentieren am Ende die Primzahlen. Durch die Vielzahl der Speicherzugriffe erklärt sich die längere Laufzeit des Algorithmus gegenüber Fibonacci, der hauptsächlich in Registern des Mikrokontrollers arbeitet.

Abbildung 6.11 zeigt einen Laufzeitvergleich der oben beschriebenen Algorithmen zwischen einem Befehlssatzsimulator und der Emulation auf dem Spyder-System. Um aussagekräftige Werte für die Messungen zu erhalten, musste die Anzahl der Befehle gegenüber der Ausführung mit dem RTL-Simulator drastisch erhöht werden. Euklid's Algorithmus führt 605.341, Fibonacci 153.740 und Eratosthenes 1.668.824 Befehle aus. Die Messungen für den Befehlssatzsimulator wurden auf einem Intel Pentium 4 mit 2,8 GHz durchgeführt. Für die Emulation wurde wiederum das Spyder-Board mit einer Taktfrequenz von 8 MHz verwendet.

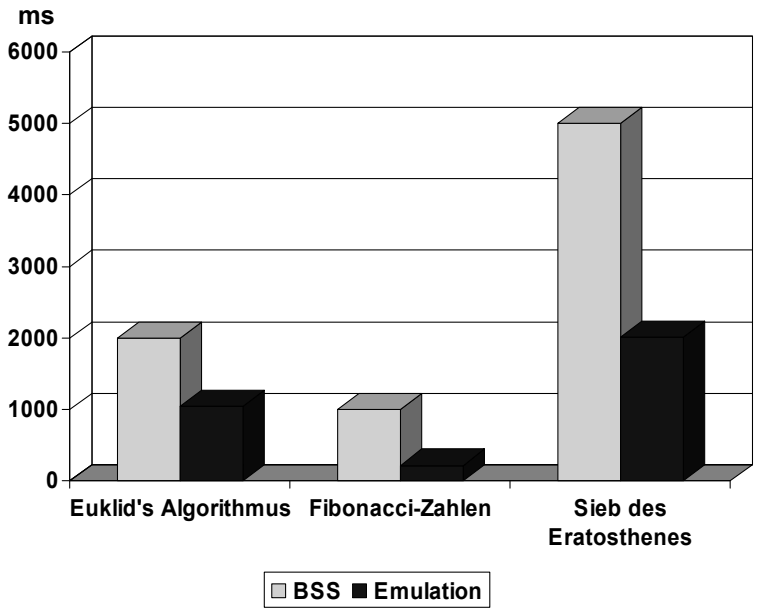


Abbildung 6.11: Laufzeitvergleich von Software-Benchmarks zwischen Befehlssatzsimulator und dem TriCore1 auf dem Spyder-Board.

Abbbildung 6.12 zeigt Syntheseergebnisse verschiedener TriCore1-Varianten für Xilinx Virtex-2000E-FPGAs. Lösung eins ist in Abbildung 6.7 (b) dargestellt. Lösung zwei wurde in Abbildung 6.8 und Lösung drei in Abbildung 6.9 vorgestellt.

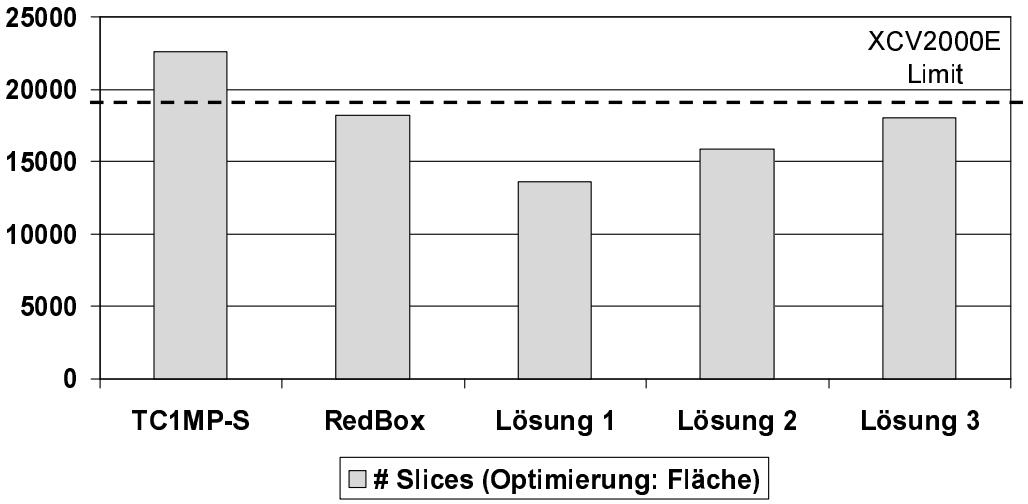


Abbildung 6.12: Syntheseergebnisse verschiedener TriCore1-Vaianten für Xilinx Virtex-2000E-FPGAs.

Mit Hilfe der Partialemulation konnte ein minimales TriCore1-System auf dem Spyder-Board implementiert werden, mit dessen Hilfe sich ein signifikanter Performanzgewinn gegenüber der RTL-Simulation erzielen ließ. Im Gegensatz zu einer reinen Implementierung der RedBox des Mikrokontrollers besitzt die aufgezeigte Lösung 3 eine Schnittstelle des LM-Bus, an der noch ein zusätzliches kleineres Modul mit dem TriCore1 kombiniert werden kann. Durch diese Schnittstelle ist die Anbindung zusätzlicher Hardware-Komponenten möglich, wodurch Vorteile gegenüber dem Befehlssatzsimulator entstehen.

Über ein spezielles Modul am LM-Bus ist es darüber hinaus möglich, Statusinformationen über den Programmablauf sowie den Erfolg oder Misserfolg eines Programmes auszugeben. Detaillierte Informationen können über eine serielle Schnittstelle an ein Hyperterminal übermittelt werden. Für die Programme und Daten stehen bis zu 4MB an statischem externem RAM zur Verfügung.

Durch die Einbindung des Kerns in das Spyder-System konnte die Anzahl der E/A-Anschlüsse drastisch reduziert werden. So verringern sich die ca. 3.000 Signale des TC1MP-S bzw. der RedBox auf 244 Signale des LM-Bus. Mit der vorgeschlagenen Lösung ist es möglich, den LM-Bus über die Erweiterungsstecker des Boards auf ein zweites Board zu leiten, auf dem dann größere IP-Kerne an den TriCore1 angeschlossen werden können. Da hier allerdings immer noch 244 Signale über die Stecker geleitet werden müssen, diese aber nur 171 freie Anschlüsse besitzen, müsste

ein Multiplexing der Signale durchgeführt werden.

Die Nachteile der vorgeschlagenen Lösung liegen vor allem in der eingeschränkten Funktionalität des Kerns. So ist es nicht möglich, zusätzliche IP-Module über den FPI-Bus anzuschließen, da die LFI-Brücke aus Platzgründen nicht implementiert werden konnte. Dadurch kann keine Entkopplung des schnellen Prozessorkerns von langsamen Peripheriemodulen erfolgen. Diese verlangsamen den Prozessor durch ihre Präsenz am LM-Bus. Da aus Platzgründen auch das CPS-Modul nicht implementiert werden konnte, steht darüber hinaus keine Unterbrechungsbehandlung für den Mikrokontroller zur Verfügung. Eine für den Entwickler transparente Software-Debugging-Umgebung konnte wegen des Fehlens des CPS-Moduls und FPI-Busses ebenfalls nicht implementiert werden.

6.3.4 Integration des TriCore1-Mikrokontroller-IP-Kerns auf dem Spyder-System

Um die im vorangegangenen Abschnitt angesprochenen Defizite vor allem im Hinblick auf die Software-Entwicklung auszugleichen, sind zusätzliche Module aus der *BlueBox* des TC1MP-S notwendig. Diese können unter der Verwendung des Spyder-Systems der zweiten Generation hinzugefügt werden. In diesem Abschnitt wird die Integration des gesamten TC1MP-S-Mikrokontroller-IP-Kerns sowie die damit mögliche umfassende Software-Entwicklung für den Mikrokontroller näher erläutert.

Integration des TC1MP-S

Um die volle Funktionalität des TriCore1-Mikrokontrollers ausschöpfen zu können, sind neben den oben bereits erwähnten Komponenten aus der BlueBox des TC1MP-S-Kerns noch weitere Hardware-Module notwendig, die im FPGA integriert werden müssen. Abbildung 6.13 zeigt den Aufbau eines minimalen TriCore1-Systems und dessen Schnittstellen zur Außenwelt.

Neben den oben bereits beschriebenen Schnittstellen für die Programm- und Daten-Scratchpad-RAMs besitzt der TriCore1 TAG-RAMs zur Verwaltung der Caches. Die Caches selbst sind als Teil der Scratchpad-RAMs implementiert. Daneben gibt es Schnittstellen für einen Co-Prozessor (COP) und das On-chip-Debugging (*On-chip-Debugging Standard (OCDS)*). Die OCDS-Schnittstelle ist ein speziell von Infineon definierter Standard, der Debugging auf drei unterschiedlichen Ebenen (L1, L2 und L3) zulässt. Der OCDS baut auf der in Abschnitt 2.2.3 beschriebenen JTAG-Schnittstelle auf. Auch hier werden, wie beim Nexus-Standard (Abschnitt 2.3.2), komplexe Befehle an den Mikrokontroller übermittelt bzw. Daten ausgelesen.

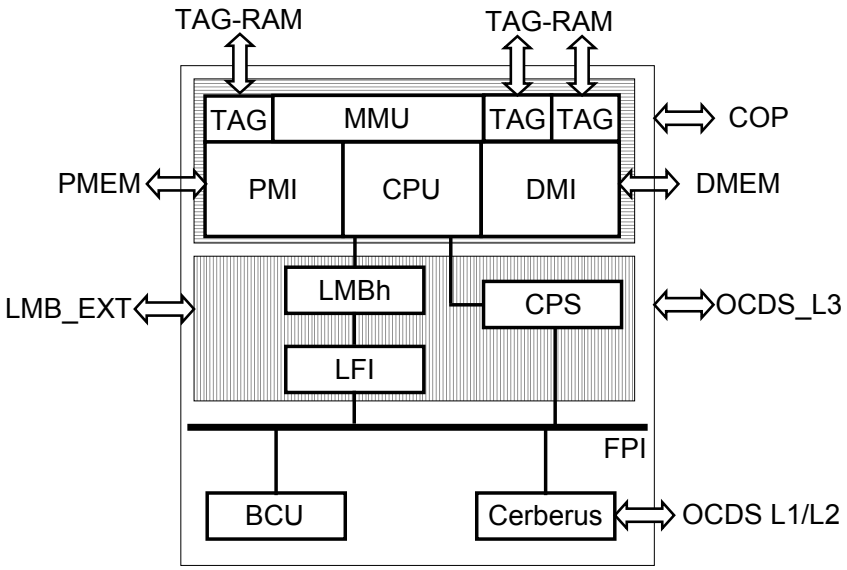


Abbildung 6.13: Schnittstellen eines minimalen TC1MP-S-Systems.

Die Anschlüsse der OCDS Level 1 und Level 2 werden an den Erweiterungssteckern des Spyder-Systems herausgeführt. Eine genaue Beschreibung der Schnittstelle für das Software-Debugging erfolgt unten. Die OCDS Level 3 Schnittstelle wird im aktuellen Entwurf nicht verwendet, kann aber prinzipiell auch auf die Erweiterungsstecker gelegt werden. Die Level 3 Signale werden wie die Signale für die Co-Prozessorschnittstelle mit vorbelegten Werten von Hilfs-Modulen getrieben.

Für die sich direkt an den CPU-Speicherschnittstellen befindenden Scratchpad-RAMs und TAG-RAMs kommt nur Speicher in Frage, der schnelle Antwortzeiten garantiert und eine einfache, schnelle und flexible Ansteuerung besitzt. Die Speicher wurden deshalb mit Hilfe von Virtex-II Block-RAMs realisiert. Diese sind für unterschiedliche Adress- und Datenbreiten konfigurierbar. Von den 18 KBit Speicherkapazität werden 2 KBit als Paritäts- und 16 KBit als Datenbits verwendet. Diese 16 KBit können flexibel bitweise oder bis zu einer Wortbreite von 32 Bit adressiert werden. Demzufolge schwankt auch die Breite des Adressbusses von 9 bis 14 Bit. Die Speicherblöcke sind somit ideal an die Bedürfnisse der unterschiedlichen Speicher des TriCore1 anpassbar.

Darüber hinaus wurden die Virtex-II Block-RAMs dazu verwendet, ein Boot-ROM für den TriCore1-Prozessorkern zu implementieren. Das Programm für das Boot-ROM wird mit in den Entwurf synthetisiert und bringt den Prozessor nach dem Reset in einen definierten Zustand. Der mit dem TriCore1-Mikrokontroller-System implementierte FPI-Bus besitzt eine Adress- und Datenbreite von 32 Bit. Da die Kapazität eines der Virtex-II Block-RAMs auf maximal 2KB beschränkt ist, wurden für

das Boot-ROM vier Block-RAMs verwendet, um den Speicher auf 8KB auszubauen. Dies reicht aus, um eine einfache Bootnachricht über eine serielle Schnittstelle zu senden. Durch diese Aufteilung in vier Bänke, kann jede Bank über 11 Bit breite Adressleitungen und 8 Bit breite Datenleitungen angesteuert werden.

Für die Realisierung größerer Speicher wurden, wie schon bei der Implementierung des limitierten TriCore1-Kerns, die beiden Spyder SRAM-Bänke verwendet. Dazu musste ein Peripheriemodul am FPI-Bus implementiert werden, welches Anfragen auf dem FPI-Bus in Anfragen an die SRAMs übersetzt.

In Abbildung 6.14 ist die Architektur der TC1MP-S-Plattform auf dem Spyder-System nochmals graphisch dargestellt. Der FPI-Bus ist in einem separaten Modul implementiert, in dem sich auch die Submodule für das Boot-ROM sowie die Spyder SRAM-Ansteuerung befinden. Die SRAM-Ansteuerung `FPI_SRAM_MOD` besitzt wiederum zwei Submodule für die Kontrolle über die zwei getrennten Bänke der Spyder SRAMs. Module, die mit Hilfe von Virtex-II-FPGA Block-RAMs implementiert wurden, sind grau hinterlegt. Am Boot-ROM sind beispielsweise die vier in FPGA Block-RAMs implementierten Boot-ROM-Bänke dargestellt.

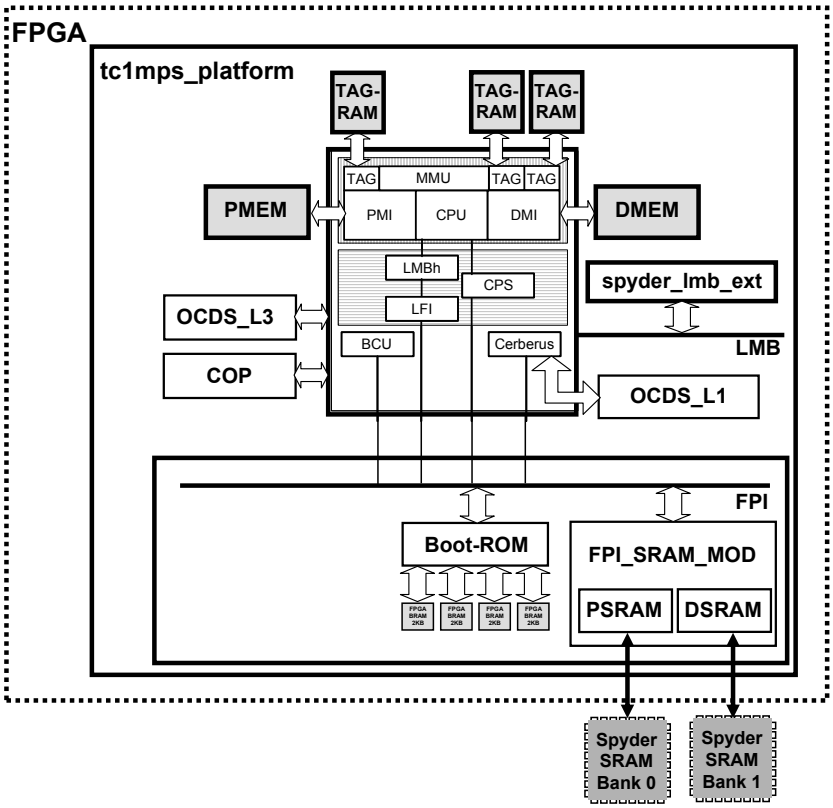


Abbildung 6.14: Implementierung der TC1MP-S-Plattform auf dem Spyder-System.

Die beiden Module `OCDS_L3` und `COP` sind zwei Hilfs-Module, die die Eingabesignale des Prozessorkerns mit Standardwerten treiben bzw. Ausgabesignale abfangen. So kann die Anzahl der E/A-Anschlüsse des Kerns drastisch reduziert werden.

Dasselbe Prinzip wurde auch bei den zwei Modulen `MMU` und `spyder_lmb_ext` angewandt. Für die Schnittstelle der CPU mit der MMU werden von Infineon zwei Module geliefert, die je nach Konfiguration der IP entweder eine MMU oder einen Stub implementieren, der die Schnittstelle zur CPU mit Standardwerten belegt. Das Modul `spyder_lmb_ext` kann durch ein Peripheriemodul ersetzt werden, welches am LM-Bus des Prozessors angeschlossen ist.

Für die Einbindung von Peripheriekomponenten wurde der Mikrokontroller-IP-Kern durch ein Submodul erweitert. In dem Submodul ist auch die Logik des FPI-Busses implementiert. In dem Submodul sind die Peripheriemodule für das Boot-ROM des Kontrollers und die SRAM-Ansteuerung implementiert. Das Modul kann dann noch um weitere Komponenten erweitert werden, die dem Entwickler eine Plattform für ein spezielles Aufgabengebiet zur Verfügung stellen. So ist es beispielsweise möglich, eine System-Steuerungs-Modul (*System Control Unit (SCU)*), eine serielle Schnittstelle (*Asynchronous Synchronous Control (ASC)*) und einen Systemzeitgeber (*System Timer Unit (STM)*) zusätzlich zu integrieren.

Die so vorgegebene Plattform kann dann durch die Hinzunahme weiterer Hardware-Komponenten auf die jeweilige Aufgabenstellung exakt zugeschnitten werden. Eine genaue Beschreibung der Kombination der vorgegebenen Mikrokontroller-Plattform um weitere Module ist Gegenstand von Abschnitt 6.4.

Software-Entwicklung mit dem Mikrokontroller-IP-Kern

Mit der in den vorangegangenen Abschnitten vorgestellten rekonfigurierbaren Hardware-Plattform ist es möglich, Hardware und Software für eingebettete Hardware/-Software-Systeme zu entwickeln. In diesem Abschnitt wird näher auf die Software-Entwicklung mit dem System eingegangen.

Für die Software-Entwicklung ist es wichtig, dass der Entwickler sich nicht mit speziellen Eigenschaften der Entwicklungsplattform beschäftigen muss. Mit der vorgeschlagenen Entwicklungsumgebung ist es möglich, dass der Hersteller einer Mikrokontroller-IP-Komponente einem Kunden das Prototyping-System zu Testzwecken überlässt und der Kunde das Board, wie mit Evaluierungsboards gewohnt, bedienen kann. Aufgrund der Rekonfigurierbarkeit der Entwicklungsplattform ist es aber sowohl dem Kunden als auch dem Hardware-IP-Hersteller möglich, die Hardware-Architektur des SoC zu ändern und um zusätzliche neue Hardware-Komponenten zu erweitern.

Um eine transparente Sicht des Entwicklers auf das Prototyping-System zu gewährleisten, müssen die aus dem Umfeld der Evaluierungsboards bekannten Software-Entwicklungswerkzeuge verwendet werden können. Dies drückt sich in der Entwicklungsumgebung für das System aus, welches in Abbildung 6.15 dargestellt ist.

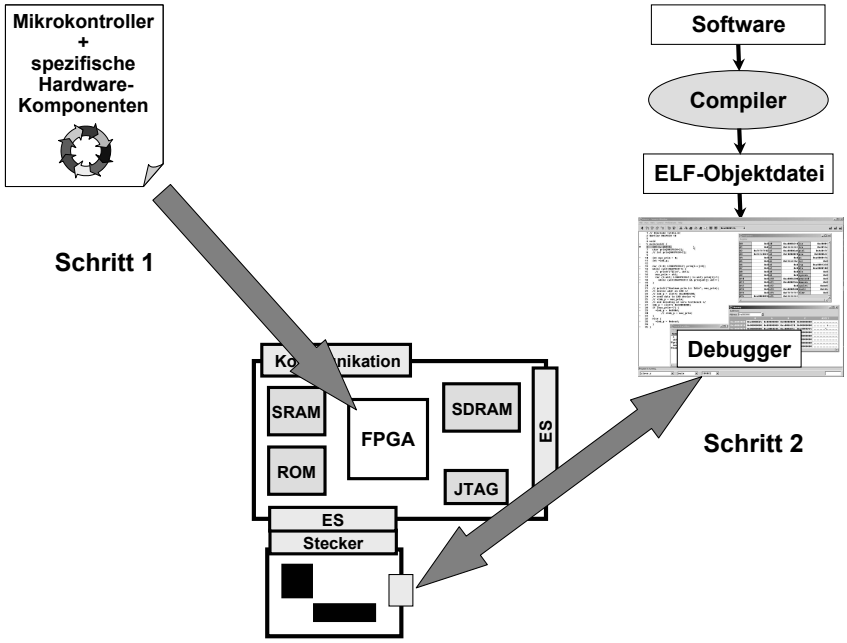


Abbildung 6.15: Software-Entwicklung mit der rekonfigurierbaren Entwicklungsumgebung.

Im Unterschied zu der in Abschnitt 6.3.2 beschriebenen Entwicklungsumgebung, bei der Programme zunächst über die PCI-Schnittstelle in die SRAMs des Boards geladen werden, werden bei der hier vorgeschlagenen Lösung die Programme wie gewohnt über einen angeschlossenen Software-Debugger in die SRAMs geladen. Der Debugger ermöglicht auch die volle Kontrolle über das System, indem beispielsweise der Programmzähler, Register oder Speicherbereiche zur Laufzeit des Programmes manipuliert werden können.

Um mit dem Software-Debugger die Kontrolle über das Mikrokontroller-System zu bekommen, wird dieser nach dem Herunterladen der FPGA-Konfigurationsdatei mit Hilfe eines Programms im Boot-ROM in einen definierten Zustand gebracht und dann durch eine Endlosschleife in diesem Zustand gehalten.

Für die Anbindung eines Software-Debuggers mit OCDS Level 1 sind mehrere Komponenten notwendig, die sich im FPGA und auf dem Entwicklungsrechner befinden. Eine Darstellung der Debugger-Werkzeugkette für verschiedene Mikrokontroller der Firma Infineon Technologies AG zeigt Abbildung 6.16.

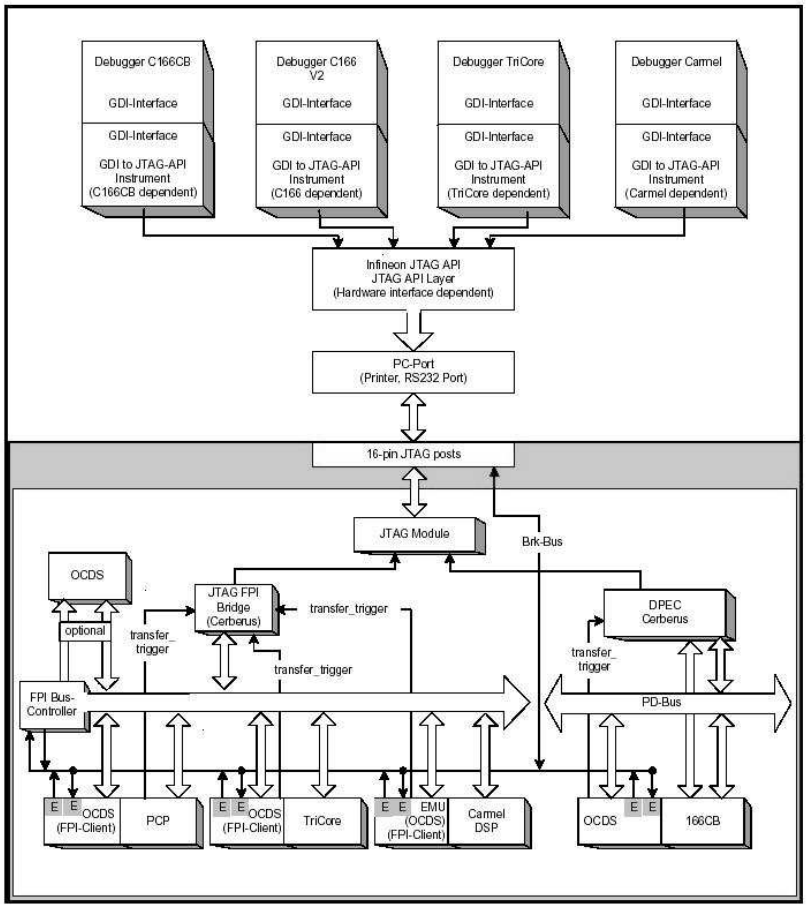


Abbildung 6.16: Software-Debugger-Anbindung für Mikrokontroller von Infineon Technologies AG [49].

Die Debugger greifen dabei über die einheitliche Schnittstelle *Infineon JTAG API* auf die PC-Hardware zu [49]. Die Anbindung erfolgt beim Spyder-System über die Parallelschnittstelle des PCs. Die Befehle des Debuggers werden in spezielle JTAG-Signale umgewandelt. Um die Kommunikation mit den Mikrokontrollern herzustellen, wurde das JTAG-Protokoll von Infineon um spezielle Befehle für die Debugging-Module auf dem Chip erweitert [52].

Abbildung 6.17 zeigt die On-chip-Debugging-Module für den TriCore1-Mikrokontroller. Die für das Software-Debugging notwendigen JTAG-Signale werden von der Komponente `JTAG_Module` in Steuersignale für das `Cerberus`-Modul übersetzt, welches sowohl eine Master- als auch eine Slave-Schnittstelle am FPI-Bus besitzt. Dadurch ist es für den Cerberus möglich, auf dem FPI-Bus Anfragen an Slave-Module wie beispielsweise das CPS, LFI oder einen Speicherbaustein abzusetzen.

Die Module reagieren dann entsprechend auf die Anfragen des Cerberus. Die Re-

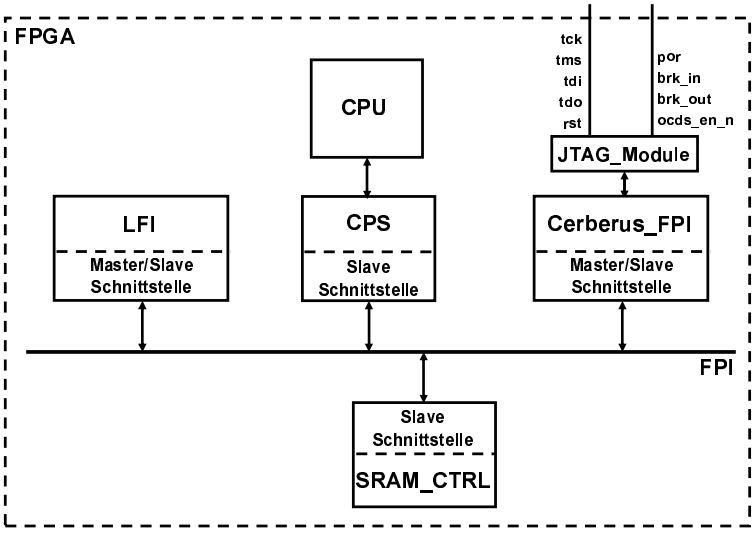


Abbildung 6.17: Hardware-Module des TriCore1 für das Software-Debugging.

gister des TriCore1 sind über das CPS in den FPI-Adressraum eingeblendet und können so manipuliert oder gelesen werden. Der LM-Bus und die Scratchpad-RAMs des TriCore1 können über die LFI-Brücke angesprochen werden und die Speicherbausteine sind direkt über ihren Adressbereich im FPI-Adressraum manipulierbar.

Der Zugriff auf die Speicherbausteine des TriCore1 ist wichtig, da mit Hilfe spezieller Debugging-Befehle Software-Breakpoints implementiert werden. Der Entwickler kann durch Anwahl einer Zeile im Quelltext-Fenster einen Breakpoint im Programm setzen. Der Debugger tauscht daraufhin den Befehl, an welchem der Breakpoint-Mechanismus aktiviert werden soll, durch einen Debugging-Befehl aus. Kommt der Prozessor an die entsprechende Stelle im Programm, wird über den Cerberus und das JTAG-Modul eine Nachricht an den Debugger gesendet. Dieser kann dann durch Auslesen des Registersatzes und der Speicher den aktuellen Status des Mikrokontroller-Kerns bestimmen und in der Benutzeroberfläche anzeigen.

6.3.5 Zwischenergebnis

In Abbildung 6.18 sind Syntheseergebnisse für einige Konfigurationen des TriCore1-Mikrokontroller-Kerns dargestellt. Die prinzipielle Architektur des Systems ist in Abbildung 6.14 bereits vorgestellt worden. Aufgrund des größeren FPGAs konnte der Entwurf im Gegensatz zur in Abschnitt 6.3.2 beschriebenen Implementierung für eine Taktfrequenz von 12 MHz synthetisiert und auf Geschwindigkeit optimiert werden.

Für das Boot-ROM sind 8KB Speicher implementiert worden. Die Scratchpad-

RAMs für Programm und Daten sind jeweils 32KB groß. Für Programm- und Datenspeicher wurden die Spyder SRAM-Chips verwendet, wodurch für den Speicher nur die FPI-Bus-Schnittstelle und Ansteuerlogik für die SRAMs im FPGA implementiert werden mussten.

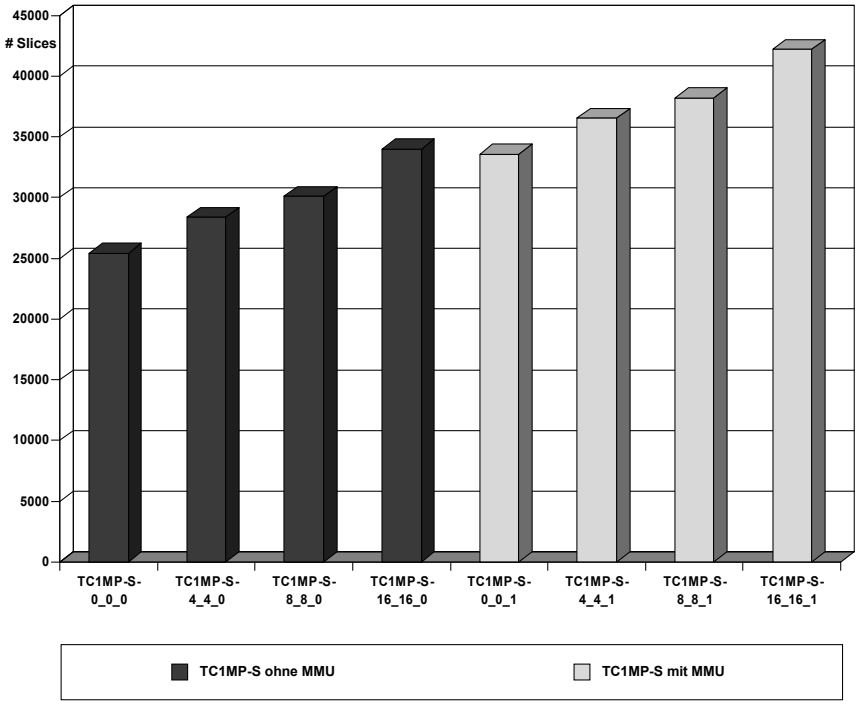


Abbildung 6.18: Syntheseergebnisse einiger beim TriCore1-Mikrokontroller verfügbarer Hardware-Konfigurationen.

Bei den unterschiedlichen Konfigurationen konnten verschiedene Cachegrößen sowie die Präsenz einer MMU eingestellt werden. Das Syntheseergebnis *TC1MP-S_4_4_0* repräsentiert beispielsweise einen TriCore1-Mikrokontroller-Kern mit 4KB Programm- und Datencache ohne MMU. Die Werte des *TC1MP-S_8_8_1*-Entwurfs spiegeln das Ergebnis für einen TriCore1-Kern mit 8KB Programm-, 8KB Datencache und einer MMU wieder.

Bei einer Limitierung der Anzahl der Slices pro FPGA von 33.792 für ein VirtexII-6000-FPGA und 46.592 für ein VirtexII-8000-FPGA wird klar, dass ein 6000er FPGA für die Implementierung einer TriCore1-Plattform mit 16KB Programm- und Datencache nicht mehr ausreicht. Da für die Implementierung zusätzlicher Peripheriemodule weitere Ressourcen des FPGAs benötigt werden, sollte für den TriCore1 in jedem Fall ein Virtex-II-8000-FPGA eingesetzt werden.

Bei einer Konfiguration des TriCore1 mit 16KB Programm- und Datencache sowie MMU ist das größte zur Zeit verfügbare FPGA zu 90% ausgelastet. Für das Pro-

totyping zusätzlicher Hardware-Komponenten muss das System deshalb erweiterbar sein. Diese Erweiterung des Spyder-Systems und die Integration zusätzlicher Hardware-Komponenten ist Gegenstand des nächsten Abschnitts.

6.4 Entwicklung eingebetteter Hardware/Software-Systeme

Mit der in dieser Arbeit vorgestellten Entwicklungsmethode wird eine parallele Entwicklung von Hardware und Software in einer frühen Phase des Entwurfs möglich. In diesem Abschnitt wird beschrieben, wie die TriCore1-basierte Mikrokontroller-Plattform um neue Hardware-Komponenten erweitert werden kann. Diese können dann zusammen mit den dazugehörigen Treibern getestet werden. Fehler in der Hardware-Komponente sowie der Schnittstelle zwischen Hardware und Software können so vor dem Tape-out des Chips erkannt und behoben werden.

Eine Anbindung von Hardware-Komponenten erfolgt beim TriCore1 am FPI-Bus. Dieser wird in Abschnitt 6.4.1 vorgestellt. In Abschnitt 6.4.2 wird beschrieben, wie die neue Hardware-Komponente zusammen mit der TC1MP-S-Plattform in einem FPGA integriert werden kann. Da diese Lösung einige Nachteile besitzt, wird in Abschnitt 6.4.3 die Verwendung von Erweiterungsplattformen für die Integration zusätzlicher Hardware-Komponenten vorgestellt.

6.4.1 Beispiel eines typischen SoC-Bussystems

Als Beispiel für ein typisches SoC-Bussystem wird an dieser Stelle der *Flexible Peripheral Interface (FPI)* Bus beschrieben, der in Abschnitt 6.1 bereits vorgestellt wurde. An dieser Stelle werden technologische Besonderheiten sowie eine technische Realisierung des Busses aufgezeigt. Die Signale des Busses lassen sich in Adress-, Daten- und Kontrollsignale aufteilen. In Tabelle 6.4 sind die Signale näher aufgeführt.

Bei den Schnittstellen der Module zum FPI-Bus muss zwischen Master- und Slave-Schnittstelle unterschieden werden. Ein Master hat zusätzlich zu den Dateneingängen und -ausgängen noch Ausgänge für das Anlegen von Adressen auf dem Bus. Slave-Schnittstellen benötigen dagegen nur einen Adresseingang.

Die Signale `fpi_req_n` und `fpi_gnt_n` werden für die Arbitrierung verwendet, für die die BCU verantwortlich ist. Master können mit den Lese- und Schreibsignalen Anfragen auf dem Bus absetzen und Anfragen mit dem Signal `fpi_abort_n` vorzeitig abbrechen. Mit dem Signal `fpi_opc` kann die Bitbreite des gewünschten Datenzugriffs angezeigt werden. Es sind auch Zugriffe auf ganze Blöcke von 2, 4, und 8 Worten möglich. Der Slave kann über das Signal `fpi_rdy` Wartezyklen für die Datenübertragung einfügen, wenn er Daten nicht in einem Takt übernehmen oder

Signal	Master	Slave	BCU	Signalbreite	Beschreibung
fpi_res_n	E	E	E	2	Reset
fpi_clk	E	E	E	1	Takt
fpi_req_n	A	-	E	# Master	Anfrage Master
fpi_gnt_n	E	-	A	# Master	Busfreigabe
fpi_lock_n	A	-	E	# Master	Sperren des Busses
fpi_rd_n	A	E	A	1	Leseanfrage
fpi_wr_n	A	E	A	1	Schreibanfrage
fpi_abort_n	A	E	A	1	Abbruch des Buszugriffs
fpi_opc	A	E	A	4	Befehl für Buszugriff
fpi_a	A	E	E/A	32	Adressen
fpi_d	E/A	E/A	A	16, 32, 64	Daten
fpi_tag	A	E	A	4	ID des Masters
fpi_ack	E/A	A	E/A	2	Anwortcode
fpi_rdy	E/A	E/A	E/A	1	Wartezyklen einfügen
fpi_sel_n	-	E	A	1	Slave Auswahl
fpi_tout	E	E	A	1	Time-out Signal
fpi_svm	A	E	A	1	Bus im Supervisor Modus

Tabelle 6.4: FPI-Bus-Signale des TriCore1-Mikrokontrollers.

herausgeben kann. Mit dem Signal `fpi_ack` ist es darüber hinaus möglich, dem Master eine Nachricht über Erfolg oder Misserfolg einer Transaktion zukommen zu lassen.

In Abbildung 6.19 ist die Realisierung des FPI-Busses auf dem Spyder-System für drei konkrete Komponenten sowie den Adressbus und das für die Einfügung von Wartezyklen zuständige `fpi_rdy`-Signal dargestellt. Das `Cerberus`-Modul besitzt sowohl eine Master- als auch eine Slave-Schnittstelle für den FPI-Bus wohingegen die Module `SRAM_CTRL` und `CPS` nur als Slave an den Bus angeschlossen sind. Deshalb besitzt auch nur das `Cerberus`-Modul einen Adressausgang und -eingang. Das Speicherkontroll-Modul und das `CPS`-Modul sind dagegen nur passiv an den Adressbus angeschlossen.

Für die Peripheriekomponenten am FPI-Bus gibt es eine Standardschnittstelle, die je nach Art des Moduls die entsprechenden Ein- und Ausgänge zur Verfügung stellt. Für jeden Ausgang wird von den Hardware-Komponenten noch ein Steuersignal getrieben, welches einen Multiplexer bedient, der bei aktivem Modul die Signale durchschaltet und bei einem inaktiven Modul entsprechend die Pegel der Signale auf eine

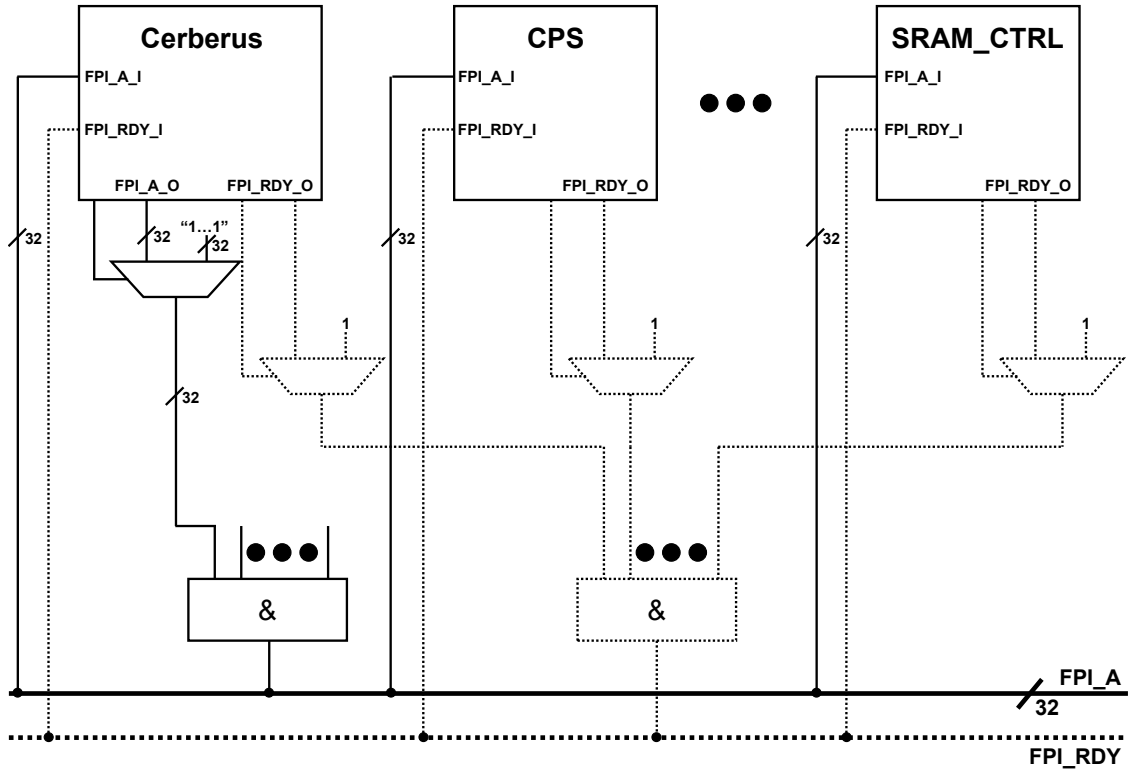


Abbildung 6.19: Implementierung des FPI-Busses auf dem Spyder-System.

logische ’1’ legt. *Alle* Signale der Peripheriemodule werden nach den Multiplexern durch ein *Und*-Gatter geleitet und formen so den FPI-Bus. Dieser wird wiederum direkt auf die entsprechenden Eingänge der Hardware-Komponenten gelegt.

6.4.2 Entwicklung und Integration neuer Hardware-Komponenten

Eine der Stärken der rekonfigurierbaren Entwicklungsplattform für SoCs ist die Flexibilität ihrer Hardware-Architektur. Sie bietet die Möglichkeit, eine bestehende TriCore1-Plattform leicht um neue Hardware-Komponenten zu erweitern. Die Hardware-Architektur einer TriCore1-Plattform wurde in Abschnitt 6.3.4 bereits näher beschrieben. Für die Integration der neuen Hardware-Komponenten gibt es grundsätzlich zwei Möglichkeiten. Die zusätzlichen Komponenten können einerseits gemeinsam mit der TriCore1-Plattform auf einem Prototyping-Board integriert werden. Die Vorteile einer Implementierung des Entwurfs auf einem Prototyping-Board liegen zum Einen darin, dass das System klein und kompakt und damit weniger anfällig gegen mechanische Einflüße an den Steckverbindern ist. Zum Anderen kann das Gesamtsystem unter Umständen mit einer höheren Taktfrequenz betrieben werden, da die Signale nicht aus dem FPGA herausgeführt werden müssen.

In Abbildung 6.20 sind Implementierungsmöglichkeiten für die Integration zusätzlicher Hardware-Komponenten dargestellt. Diese können zum Einen zusammen mit den Peripheriemodulen, die die Hostplattform des TC1MP-S-Systems aufspannen, in einem gemeinsamen Submodul implementiert werden (Lösung 1). Zum Anderen können diese in einem zweiten, externen Submodul integriert werden (Lösung 2). Durch die Kapselung der Peripheriemodule in einem externen Submodul kann die Komplexität der Schnittstellen des TriCore1 vor dem Entwickler verborgen werden.

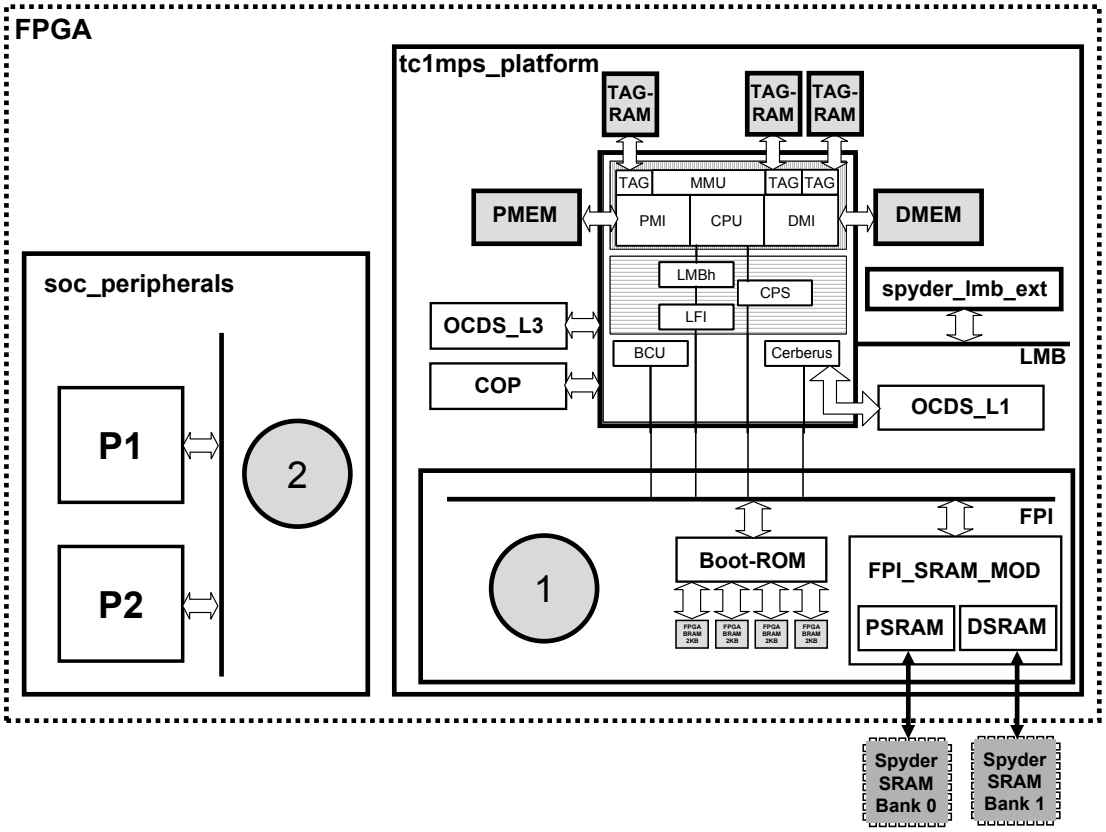


Abbildung 6.20: Implementierungsmöglichkeiten für die Einbindung neuer Hardware-Komponenten.

In diesem Zusammenhang wird die Frage aufgeworfen, in welchem Modul der FPI-Bus implementiert werden sollte, da über diesen Peripheriemodule an den Mikrokontroller angebunden werden. Im Falle von Lösung eins wird der FPI-Bus in dem Submodul für die Peripheriekomponenten implementiert. Bei Lösung zwei gibt es mehrere Alternativen.

Der FPI-Bus könnte zum Einen innerhalb des Submoduls implementiert werden in dem die Peripheriemodule der TC1MP-S-Plattform implementiert sind. Dies hat allerdings den Nachteil, dass der Entwickler weit in die TC1MP-S-Komponente ein-

greifen muss. Eine andere Möglichkeit bestünde darin, den FPI-Bus auf einer höheren Ebene außerhalb der TC1MP-S-Plattform zu implementieren. In diesem Fall führen dann allerdings sehr viele Signale aus der Plattform heraus und in sie hinein.

Aus den oben genannten Gründen wird für die Lösungsmöglichkeit zwei zur Integration zusätzlicher Hardware-Komponenten eine Implementierung gewählt, bei der der FPI-Bus in zwei Submodulen implementiert wird. Es sind dies die beiden Module `soc_peripherals` und `tc1mps_peripherals`. Diese Lösung ist in Abbildung 6.21 dargestellt. Um dennoch einen gemeinsamen FPI-Bus zu realsieren, werden allerdings die zu einem Ausgangsbus vereinigten FPI-Bus-Signale des Submoduls `soc_peripherals` in der *UND*-Stufe des Submoduls `tc1mps_peripherals` noch mit den Ausgangssignalen der Peripheriemodule der TC1MP-S-Plattform kombiniert.

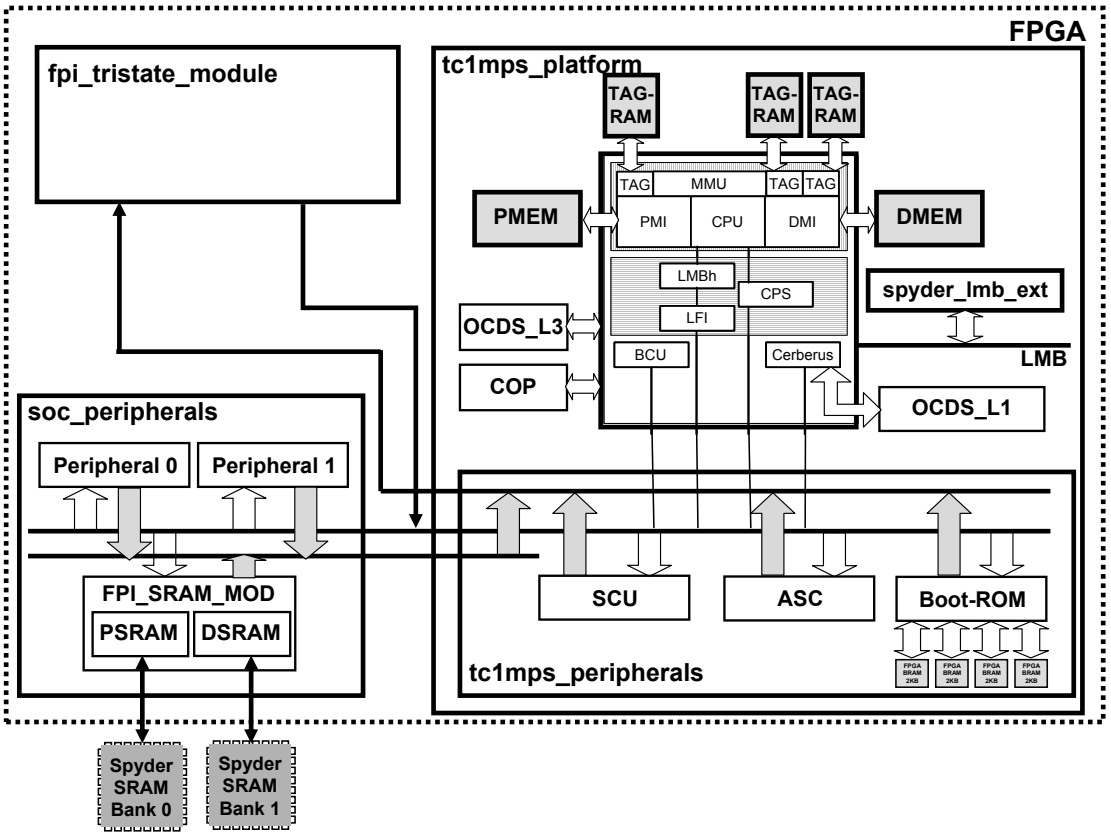


Abbildung 6.21: Implementierung eines TriCore1-basierten SoC mit zwei getrennten Submodulen für Peripheriekomponenten.

Im Submodul `fpi_tristate_module` wird der FPI-Bus an Tristate-E/A-Anschlüsse des FPGAs geführt. Dies hat zwei Vorteile. Zum Einen hat der FPI-Bus mit Hilfe dieser Konstruktion nach der Synthese eine definierte Länge und das zeitli-

che Verhalten des Busses ist stabiler. Zum Anderen lassen sich die FPI-Bus-Signale gut mit einem externen Logikanalysator überwachen. Darüber hinaus kann an dieser Schnittstelle dann auch eine Erweiterungsplattform mit zusätzlichen Hardware-Komponenten angeschlossen werden.

6.4.3 Plattformbasierter Entwurf mit Erweiterungsplattformen

Die im vorangegangenen Abschnitt aufgezeigte Lösung hat für die Entwicklung neuer Hardware-Komponenten allerdings einige Nachteile. Zum Einen sind durch die Integration der TC1MP-S-Plattform schon viele Ressourcen des Prototyping-Boards verbraucht. Aus Abbildung 6.18 auf Seite 114 geht hervor, dass ein TriCore1 mit 16KB Programm- und Datencache ca. 90% der Logikressourcen des größten zur Zeit verfügbaren FPGAs von Xilinx belegt. Zum Anderen entstehen sehr hohe Synthesezeiten für das Gesamtsystem, was bei einer Trennung von TriCore1-Plattform und der neuen Hardware-Komponente nicht der Fall ist. Da gerade diese aber noch Entwurfsfehler enthalten kann, sind sehr viele Syntheseläufe notwendig. Darüber hinaus ergeben sich auch strategische Probleme für den IP-Anbieter. Dieser muss bei einer Einplatinenlösung den Quelltext seiner IP für die Synthese zur Verfügung stellen.

In dieser Arbeit werden aus diesem Grund Erweiterungsplattformen vorgeschlagen, die die neue Hardware-Komponente aufnehmen und an das Prototyping-System angeschlossen werden. Die Topologie der Erweiterung des Prototyping-Systems kann dabei unterschiedlich sein. Manche Systeme lassen sich als Matrix aus mehreren Boards aufbauen [29]. Das Spyder-System kann dagegen über eine Backplane an den Erweiterungssteckern der Boards erweitert werden, die eine Busstruktur besitzt.

Bei der Kombination mehrerer Boards können unterschiedliche Probleme auftreten. Zunächst muss eine Partitionierung des Entwurfs auf die unterschiedlichen FPGAs erfolgen. Diese kann unter Umständen nur mit Werkzeugunterstützung bewältigt werden. Darüber hinaus kann bei sehr komplexen Entwürfen und aufgrund von Limitierungen der Anschlüsse des FPGAs und der Steckverbinder die Anzahl der Verbindungsleitungen zwischen den Boards nicht ausreichen.

Große Logikemulatoren umgehen diese Probleme, indem sie aufwändige Software-Werkzeuge für die Partitionierung und komplexe Verbindungsnetzwerke für die Verbindung der Platinen untereinander anbieten [60]. Die Infrastruktur für das Prototyping wird dadurch sehr teuer und kann eine Hürde für den Einstieg eines Kunden in eine neue Technologie darstellen, da er die Geräte bzw. Programme erst beschaffen muss.

Mit der vorliegenden Arbeit wird eine Alternative aufgezeigt, die es einem IP-Anbieter ermöglicht, einem Kunden eine kostengünstige Versuchsplattform für seine Produkte zur Verfügung zu stellen. Um Kosten einzusparen und dennoch eine effizi-

ente Entwicklungsumgebung für das Prototyping eines SoC zur Verfügung zu haben, müssen die Ressourcen des Boards optimal ausgenützt und der Entwurf funktionell partitioniert werden.

Dies geht dann besonders effizient, wenn die Verbindungstopologie des Rapid-Prototyping-Systems mittels eines Busses implementiert ist. Mit dieser Architektur kann der SoC-Bus des Entwurfs auf die als Bus realisierte Verbindungsstruktur des Rapid-Prototyping-Systems abgebildet werden. In Abbildung 6.18 wurde bereits gezeigt, dass eine TriCore1-basierte Plattform bestehend aus dem TriCore1-Kern, einer BCU, dem Software-Debugging-System und einem Speichersubsystem auf einem Spyder-Board nachgebildet werden kann. Dieses System kann erweitert werden, indem der FPI-Bus an den Erweiterungssteckern des Boards herausgeführt wird. Dazu muss eine Abbildung des FPI-Busses auf die Backplane des Spyder-Systems erfolgen.

Für die Abbildung des Busses auf die Backplane des Spyder-Systems stehen dem Entwickler drei unterschiedliche Möglichkeiten zur Auswahl. Bei den hier vorgestellten Lösungsmöglichkeiten wurde darauf geachtet, dass sie skalierbar sind und mit Boardmitteln des Prototyping-Systems umgesetzt werden können.

Die einfachste Lösung stellt eine direkte Abbildung der Leitungen des SoC-Busses auf die Anschlüsse des FPGAs und die Leitungen der Backplane dar. Da diese Lösung für sehr komplexe Bussysteme oft nicht realisierbar sein wird, gibt es zum Einen die Möglichkeit, die Verbindungen der Backplane im Multiplexerbetrieb und zum Anderen im bidirektionalen Betrieb zu verwenden. Bei beiden Lösungen müssen aber nicht, wie bei komplexeren Rapid-Prototyping-Systemen, spezielle Verbindungsplatinen verwendet werden, sondern es werden spezielle Eigenschaften der FPGAs des Spyder-Systems ausgenutzt.

Direkte Abbildung des Busses

Bei einer direkten Abbildung des Busses auf die Backplane des Spyder-Systems wird jeweils eine Signalleitung in unidirektionaler Richtung für die Implementierung einer Signalleitung des FPI-Busses verwendet. Wird die in Abschnitt 6.4.1 vorgestellte Realisierung des FPI-Busses verwendet, müssen an einer zentralen Stelle alle FPI-Bus-Signale zusammengeführt und mit einem *UND*-Gatter verknüpft werden. Damit entsteht die Situation, dass ein Board im Master-Modus und ein Board im Slave-Modus betrieben werden muss. Die daraus resultierende Architektur des Systems ist in Abbildung 6.22 dargestellt.

Auf dem Erweiterungsboard sind zwei Peripheriemodule P1 und P2 realisiert. Die Ausgangssignale beider Module werden zunächst lokal durch ein *UND*-Gatter geführt und müssen danach noch mit den Ausgangssignalen des TriCore1-Systems

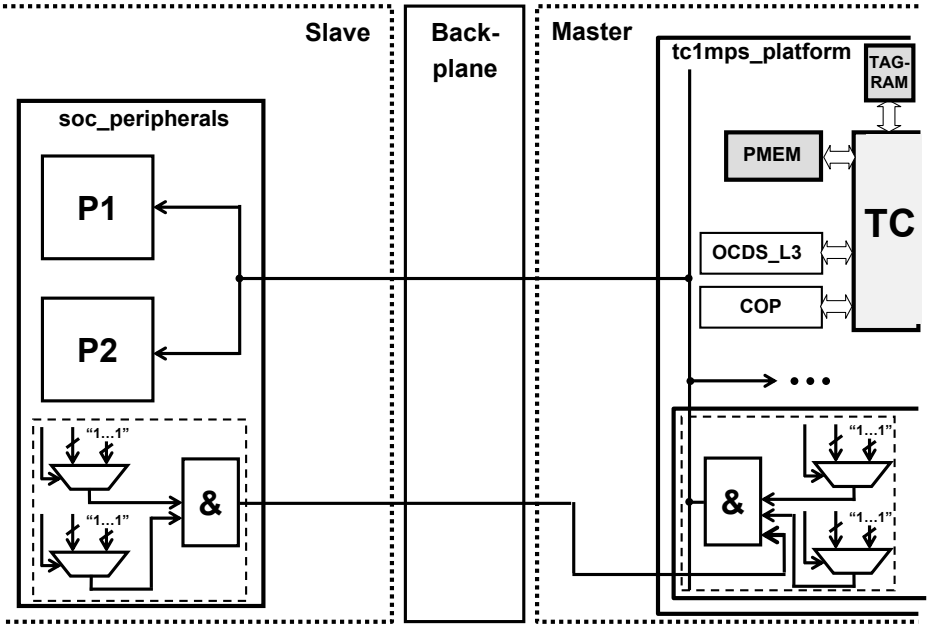


Abbildung 6.22: Implementierung eines verteilten SoC-Entwurfs mit einer TriCore1-Plattform und unidirektionalen Verbindungen auf der Backplane.

kombiniert werden. Danach stehen sie als FPI-Bus allen Modulen an den Eingängen zur Verfügung.

Je nach Art der Peripheriemodule ist aus der Sicht der Erweiterungsplattform eine unterschiedliche Anzahl an Signalleitungen auf der Backplane erforderlich. In Abbildung 6.23 ist eine Gegenüberstellung der für unterschiedliche Peripheriemodule erforderlichen Anzahl Signalleitungen zu finden. Da Erweiterungsstecker eins 85 und Erweiterungsstecker zwei 86 freie Anschlüsse bietet, ergibt sich insgesamt eine Limitierung von 171 Signalen auf der Backplane. Die Anzahl der Signalleitungen von Erweiterungsstecker eins ist um eine Leitung kleiner, da eine Leitung für die Übertragung des systemweiten Taktes verwendet wird.

Wie aus der Abbildung ersichtlich ist, werden bei einer Implementierung einer Hardware-Komponente mit Master/Slave-Schnittstelle auf der Erweiterungsplattform über 95% der Ressourcen der Backplane aufgebraucht. Zusätzlich treten Probleme auf, wenn noch Signale für die Ansteuerung von Peripheriekomponenten benötigt werden. Alleine für den Anschluß eines Software-Debuggers für den TriCore1 werden neun Anschlüsse an den Erweiterungssteckern benötigt. Eine serielle Schnittstelle braucht mindestens zwei weitere Signalleitungen für Sende- und Empfangseinheit. Das Herausführen von internen Signalen für das Hardware-Debugging ist ebenfalls nur noch sehr eingeschränkt möglich.

Darüber hinaus werden für Peripheriemodule häufig Unterbrechungssignale benö-

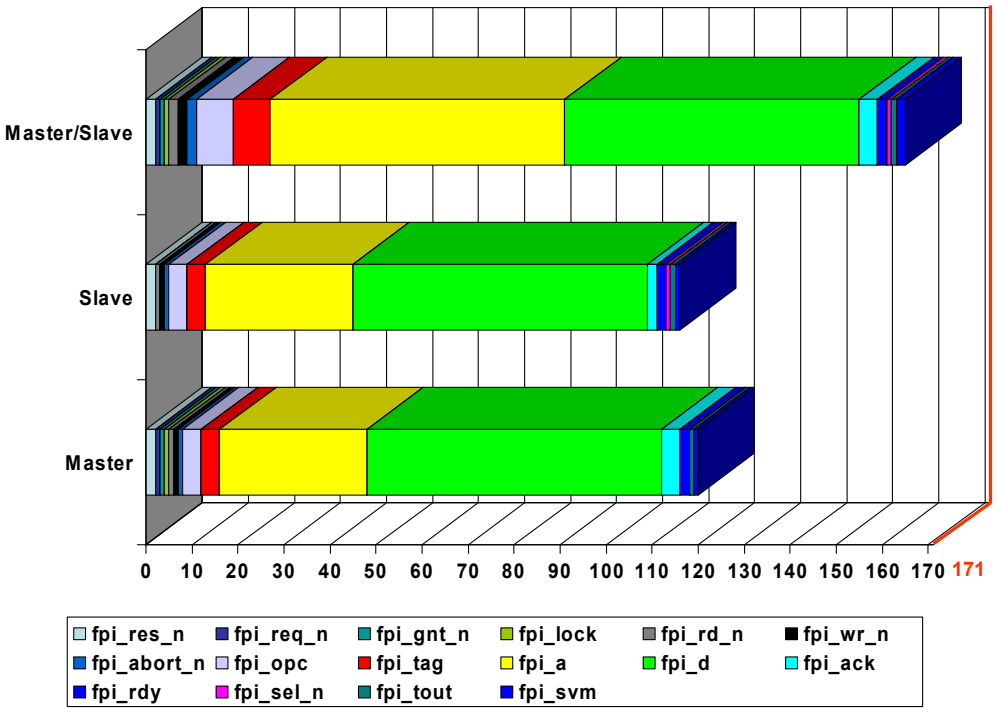


Abbildung 6.23: Verbrauch von Anschlussressourcen bei einer direkten Abbildung des FPI-Busses auf die Backplane.

tigt, für deren Implementierung nochmals mindestens sieben Leitungen erforderlich sind. Bei einem Peripheriemodul, welches eine Master/Slave-Schnittstelle besitzt, würden so insgesamt 179 Signale benötigt, was die Kapazität von 171 Signalleitungen der Backplane übersteigt. Gleiches gilt für zwei Module, bei denen eines mit einer Master- und das andere mit einer Slaveschnittstelle ausgestattet ist.

Verwendung einer Multiplexer-Lösung

Da eine direkte Abbildung des FPI-Busses auf die Backplane des Spyder-Systems mit wesentlichen Nachteilen verbunden ist, wurden zwei weitere Lösungsmöglichkeiten untersucht, die die Anzahl der benötigten Signalleitungen drastisch reduzieren. Während der Arbeit wurde bei der Entwicklung alternativer Buskonzepte vor allem auf die Realisierung kostengünstiger und einfacher Lösungen geachtet, die keine speziellen hardware- und software-technischen Anforderungen an das Prototyping-System stellen.

Das Spyder-System kann als ideale Grundlage hierfür angesehen werden. Große Hardware-Emulations-Systeme umgehen das Problem der Limitierung der Signalleitungen dadurch, dass diese im Multiplexerbetrieb betrieben werden. Dies kann zum Einen durch spezielle Verbindungs-Hardware und zum Anderen durch die Imple-

mentierung einer Multiplexerschaltung im FPGA geschehen. Die erste Lösung macht die Hardware teurer und die zweite Lösung verbraucht Ressourcen des FPGAs.

Die hier untersuchte Implementierung nutzt für den FPI-Bus spezielle E/A-Anschlüsse der FPGAs, die mit doppelter Datenrate betrieben werden können. Die Schnittstelle zur Spyder-Backplane wird mit Hilfe eines Submoduls realisiert, in dem die E/A-Anschlüsse mit doppelter Datenrate (*DDR-E/A*) instanziiert werden. Die Architektur des Entwurfs ist in Abbildung 6.24 dargestellt.

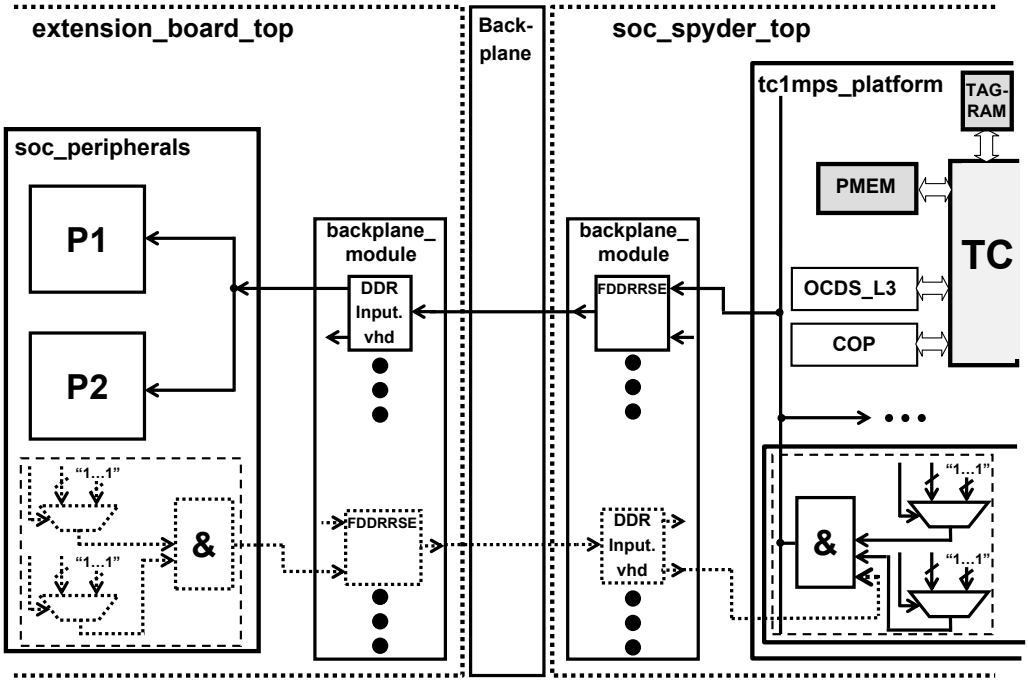


Abbildung 6.24: Abbildung des FPI-Busses auf DDR-E/A-Anschlüsse des FPGAs und die Spyder-Backplane.

Innerhalb des Submoduls `backplane_module` werden zwei Komponenten instanziiert, die für das Multiplexing der Daten auf der Backplane des Spyder-Systems verantwortlich sind. Die Komponente `FDDRRSE` ist eine vorgefertigte Komponente für Xilinx VirtexII-FPGAs, die aus zwei Eingängen einen Ausgang macht. Sie verwendet dazu zwei Takteingänge, die um $180°$ phasenverschoben sein müssen. Der Ausgang wird jeweils bei steigender Taktflanke auf ein Eingangssignal geschaltet.

Um einen Eingang am FPGA mit doppelter Datenrate betreiben zu können, wurde die Komponente `DDR_Input.vhd` implementiert, deren Quelltext in Programmabschnitt 6.1 zu sehen ist. In der Komponente werden zwei Register erzeugt, die vom Synthesewerkzeug als DDR-E/A-Register der Anschlüsse des FPGAs erkannt und darauf abgebildet werden. Mit der positiven bzw. negativen Taktflanke des Takteingangs wird der Eingang des Flip-Flops, der einer Signalleitung auf der Backplane

entspricht, auf das jeweilige FPI-Bus-Signal weitergeschaltet. Da der FPI-Bus im Zustand `reset` aktive Signalleitungen benötigt, müssen die Ausgänge in den Flip-Flops entsprechend gesetzt werden (Zeilen 22 und 35).

```
1  entity DDR_Input is
2    port (
3      clk : in  std_logic;
4      rst : in  std_logic;
5      d   : in  std_logic;
6      q1  : out std_logic;
7      q2  : out std_logic
8    );
9  end DDR_Input;
10
11 architecture behavioral of DDR_Input is
12
13 begin -- behavioral
14
15 -- purpose: Describe input DDR register to be inferred
16 -- type   : combinational
17 -- inputs : clk, rst, d
18 -- outputs: q1
19 q1reg: process (clk, rst, d)
20 begin -- process q1reg
21   if (rst='0') then
22     q1 <= '1';                          -- FPI bus specific
23   elsif (clk'event and clk='0') then    -- clk event negedge
24     q1 <= d;
25   end if;
26 end process q1reg;
27
28 -- purpose: Describe input DDR register to be inferred
29 -- type   : combinational
30 -- inputs : clk, rst, d
31 -- outputs: q2
32 q2reg: process (clk, rst, d)
33 begin -- process q2reg
34   if (rst='0') then
35     q2 <= '1';                          -- FPI bus specific
36   elsif (clk'event and clk='1') then    -- clk event posedge
37     q2 <= d;
38   end if;
39 end process q2reg;
40
41 end behavioral;
```

Programm 6.1: Implementierung eines FPGA-E/A-Anschlusses mit doppelter Datenrate.

Für die beiden Plattformen wurde jeweils ein eigener *lokaler* FPI-Bus implementiert, der, wie bei der in 6.4.3 beschriebenen Lösung, auf dem Board mit der TC1MP-S-Plattform zu einem *globalen* FPI-Bus zusammengeführt wird. Der globale Bus wird dann wieder auf das Board mit den Peripheriemodulen zurückgeführt, wo er als Eingang dient. Durch die Kombination der Ausgänge der Peripheriemodule auf dem Erweiterungsboard auf eine gemeinsame FPI-Bus-Signalleitung ist es nicht notwendig, die Ausgänge einzeln über die Backplane zu leiten.

Die mit den DDR-E/A-Anschlüssen realisierte Erweiterung des Spyder-Systems hat allerdings einen entscheidenden Nachteil. Das Multiplexen der Signalleitungen wird mit Hilfe von Flip-Flops und im Falle der Komponente FDDRRSE mit einem um $180°$ phasenverschobenen Takt realsiert. Dies macht die Implementierung des korrekten zeitlichen Verhaltens des FPI-Busses sehr komplex, da neben dem Systemtakt noch zusätzliche schnellere Takte für die Steuerung des FDDRRSE und der Flip-Flops verwendet werden müssen.

Durch die Verwendung der DDR-E/A-Anschlüsse wurde dann zwar die Anzahl der benötigten Signalleitungen auf der Backplane drastisch reduziert, es werden aber mehr Ressourcen in Bezug auf die Anzahl der Taktdomänen verbraucht. Aus diesem Grund wird im Folgenden eine alternative Lösung für die Abbildung des FPI-Busses auf die Backplane vorgestellt, die nur eine minimale Anzahl zusätzlicher E/A-Anschlüsse des FPGAs gegenüber einer DDR-E/A-Lösung benötigt.

Verwendung einer Tristate-Lösung

Neben der Möglichkeit, die E/A-Anschlüsse des FPGAs mit doppelter Datenrate zu betreiben, gibt es noch die Möglichkeit, die Tristate-Treiber der E/A-Anschlüsse zu verwenden. Die Leitungen der Backplane müssen in diesem Fall bidirektional betrieben werden. Da es sich bei der in Abschnitt 6.4.1 vorgestellten Implementierung des FPI-Busses nicht um einen bidirektionalen Bus handelt, muss dieser entsprechend angepasst werden. Diese Anpassung wird in einem speziellen Hardware-Modul gekapselt, welches im FPGA implementiert ist und Logik der E/A-Anschlüsse des FPGAs verwendet.

Die bisher verwendete FPI-Bus-Logik muss hierfür pro Signal um ein weiteres *UND*-Gatter erweitert werden, welches die Steuerleitungen für die Multiplexer der Hardware-Komponenten zusammenfasst und den Tristate-Treiber Eingang T des E/A-Anschlusses des FPGAs bedient. Der Ausgang des Tristate-Treibers dient als Eingang für die IP-Module auf der Hostplattform und wird gleichzeitig auf die Erweiterungsstecker des Spyder-Boards geleitet, wo er mit den Eingängen der Hardware-Komponenten auf dem Erweiterungsboard verbunden werden kann.

Die Architektur des Entwurfs ist in Abbildung 6.25 für die Hostplattform, d.h.

für das Prototyping-Board welches den TriCore1-Kern realisiert, dargestellt. Für die Erweiterungsboards sieht die Schaltung entsprechend identisch aus.

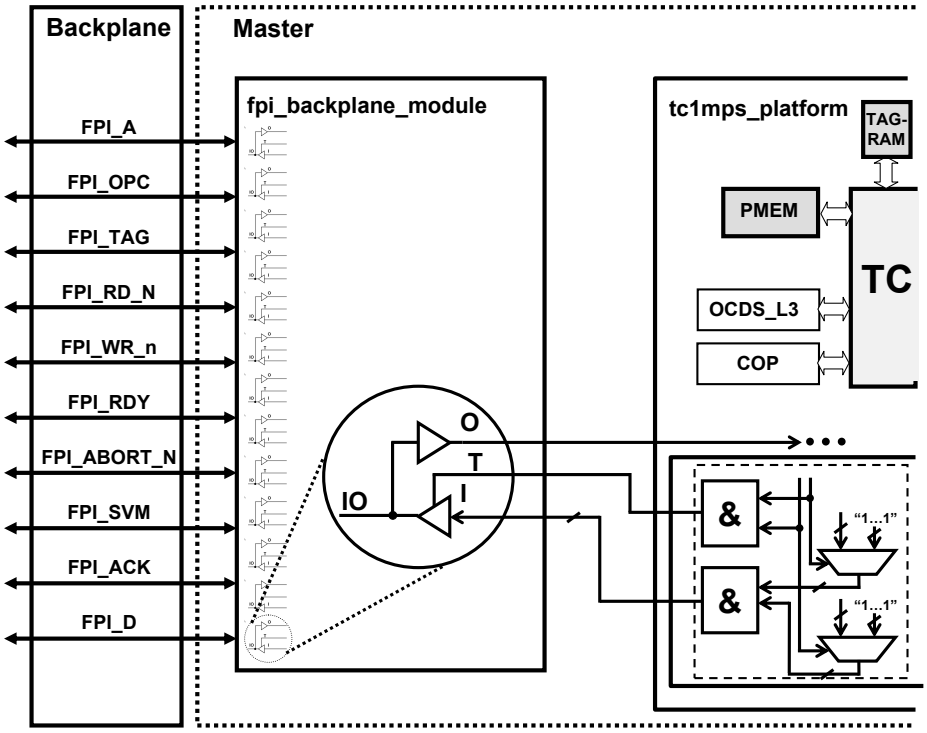


Abbildung 6.25: Abbildung des FPI-Busses auf Tristate-E/A-Anschlüsse des FPGAs und die Spyder-Backplane.

Bei dieser Lösung muss unter allen Umständen darauf geachtet werden, dass die Busse der beiden Boards nicht gegeneinander treiben. Die Arbitrierung des Busses wird vom Bus-Kontroller übernommen, der im Modul `tc1mps_platform` implementiert ist. Aus diesem Grund müssen für Hardware-Komponenten, die als Master auf dem Bus agieren wollen und auf der Erweiterungsplattform implementiert sind, zwei zusätzliche Signale für Busanforderung und -zusage auf der Backplane reserviert werden.

Der Zugriff für die Slaves wird über *Slave-Select*-Signale realisiert, die abhängig von dem Wert des Addressbusses sind. Da der Addressbus sowohl auf der Hostplattform als auch auf dem Erweiterungsboard verfügbar ist, können die Slave-Select-Signale auf den jeweiligen Boards erzeugt und an die Slaves weitergegeben werden.

Da der FPI-Bus zu jedem Zeitpunkt von einem Modul getrieben werden muss, entsteht das Problem, dass die Hostplattform wissen muss, ob auf dem Erweiterungsboard ein Master oder ein Slave aktiv ist. Ist dies der Fall, so darf die Hostplattform den FPI-Bus nicht mit vorgegebenen Werten treiben. Es ist daher eine zusätzliche Signalleitung auf der Backplane vom Erweiterungsboard zur Hostplattform notwendig,

die signalisiert, ob eine Hardware-Komponente auf dem Erweiterungsboard aktiv ist.

6.4.4 Verifikation der Hardware-Komponenten

Für die Verifikation der Hardware-Komponenten stehen bei dem vorgestellten System unterschiedliche Möglichkeiten zur Verfügung. Handelt es sich bei den zusätzlich integrierten Komponenten um Module, die einen Algorithmus in Hardware beschleunigen, also keine dedizierten E/A-Signale haben, so kann das Ergebnis der Berechnung direkt von einem Programm verifiziert werden, welches auf dem emulierten TriCore1-Mikrokontroller abläuft.

Sind die Ergebnisse nicht korrekt, können darüber hinaus interne Signale des Moduls entweder auf den Erweiterungssteckern des Spyder-Boards herausgeführt werden, oder es kann der FPGA-interne Logikanalysator der Firma Xilinx verwendet werden. Da dieser allerdings zusätzliche Ressourcen des FPGAs wie Block-RAMs und Taktdomänen benötigt, ist der Einsatz eines Erweiterungsboards für die Implementierung zusätzlicher Komponenten sinnvoll, da das Erweiterungsboard zusätzliche Ressourcen bereitstellt.

Module, die mit der Außenwelt kommunizieren, können durch die Verbindung mit externen Komponenten überprüft werden. An den Erweiterungssteckern kann beispielsweise eine physikalische Schaltung für eine RS232-Schnittstelle implementiert werden, die eine UART-Komponente mit einem PC verbindet. Dadurch kann eine serielle Kommunikation mit externen Geräten verifiziert werden. Die Stimulierung des Moduls wird in diesem Fall von Seiten des FPI-Busses durch Software getestet. Die Software kann später als Basis zur Implementierung der Treiber für die zu evaluierende Hardware-Komponente herangezogen werden. Die Eingabesignale werden in diesem Fall vom PC generiert.

Falls es sich bei dem zu implementierenden Protokoll allerdings um eine neue Form der Kommunikation handelt, also noch keine entsprechenden *Gegenstellen* existieren, gibt es zwei Möglichkeiten, diese neuen Module zu testen. Zum Einen kann die Testbench für die zu verifizierende Komponente gemeinsam mit dem Entwurf in das FPGA synthetisiert werden. Falls die Hardware-Testbench nicht synthetisierbar ist, kann diese prinzipiell in einem Simulator auf dem PC betrieben werden, und über den PCI-Bus mit dem Spyder-Board und dort mit der neuen Hardware-Komponente kommunizieren [104].

6.4.5 Zwischenergebnis

Mit der in Abschnitt 6.4.3 beschriebenen Tristate-Lösung des FPI-Busses konnte der gesamte FPI-Bus auf der Spyder-Backplane realisiert werden. In Abbildung 6.26 ist

die Reduzierung der benötigten Signalleitungen des FPI-Busses auf der Backplane durch die Verwendung der DDR-E/A- bzw. der Tristate-E/A-Anschlüsse des FPGAs gegenüber einer direkten Abbildung des FPI-Busses auf die Spyder-Backplane zu sehen.

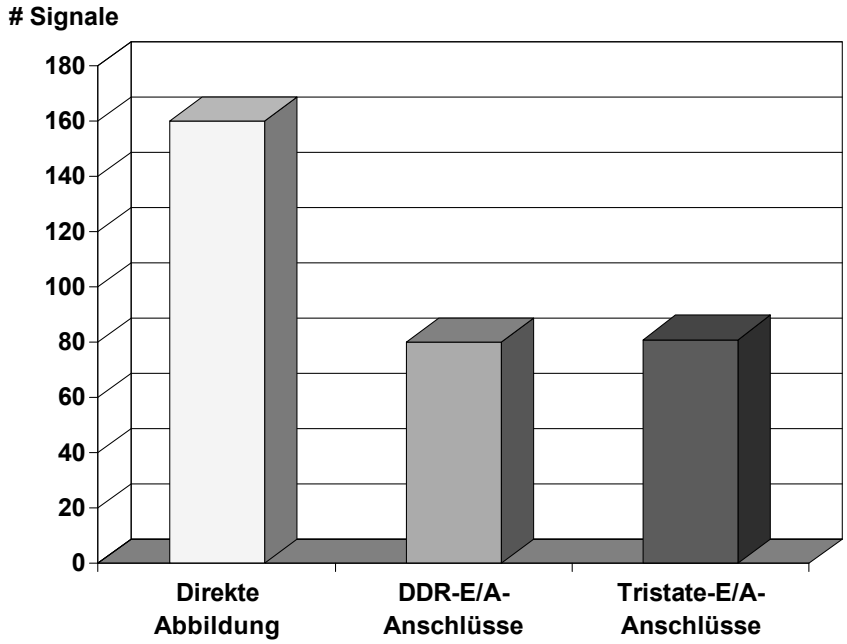


Abbildung 6.26: Ressourcenverbrauch unterschiedlicher FPI-Bus-Realisierungen auf der Backplane des Spyder-Systems.

Die Transformation des FPI-Busses in eine Tristate-Version benötigt drei Signalleitungen mehr als eine Lösung mit DDR-E/A-Anschlüssen. Da die Tristate-Lösung allerdings mit einem gemeinsamen Takt für das Gesamtsystem zu realisieren ist, ist diese Lösung der Variante mit den DDR-E/A-Anschlüssen vorzuziehen. Sie wird daher zur Anbindung von Erweiterungsboards und zum Prototyping neuer Hardware-Komponenten für den TriCore1-Mikrokontroller verwendet.

6.5 Zusammenfassung

In diesem Kapitel wurden die in Kapitel 5 entwickelten Konzepte für die schnelle und effiziente Entwicklung eingebetteter Hardware/Software-Systeme anhand eines konkreten Beispiels angewendet und deren Wirkungsweise gezeigt. Dazu wurde ein typischer Mikrokontroller-IP-Kern für SoCs auf dem kostengünstigen Rapid-Prototyping-System Spyder integriert. Es handelt sich um das Softmakro TC1MP-S, welches Teil der für SoC-Entwürfe bestimmten DesignWare-StarIP-Bibliothek von Synopsys ist.

Neben den technologischen Problemen bei der Konvertierung des ASIC-IP-Kerns auf das FPGA des Spyder-Systems konnte mit Hilfe der Partialemulation das Ressourcenproblem kostengünstiger Prototyping-Plattformen gelöst und ein minimales Mikrokontroller-System implementiert werden. Um eine transparente Software-Entwicklung zu unterstützen, wurde der volle Kern mit den für das Software-Debugging notwendigen Hardware-Debugging-Modulen auf einem Spyder-System der zweiten Generation implementiert und eine Cross-Entwicklungsumgebung an das System angebunden.

Diese Entwicklungsumgebung kann für die parallele Entwicklung neuer Hardware-Komponenten und der dazugehörigen hardware-nahen Software genutzt werden. Dazu werden Erweiterungsplattformen an das Prototyping-Systems angeschlossen, die zusätzliche Ressourcen bereitstellen, den Syntheseprozess verkürzen und darüber hinaus den Quelltext der Mikrokontroller-IP des Herstellers vor dem Kunden verbergen. Genau wie die Software kann die Hardware mit herkömmlichen Entwicklungsmethoden und Werkzeugen entwickelt und verifiziert werden.

7 Ergebnisse

In diesem Kapitel werden Ergebnisse präsentiert, die mit dem in dieser Arbeit vorgestellten neuen Konzept für die Entwicklung eingebetteter Hardware/Software-Systeme erzielt werden können. Als Grundlage dient der in Kapitel 6 vorgestellte Mikrokontroller-IP-Kern und das vorgestellte Prototyping-System.

Eines der Ziele dieser Arbeit ist, die Software-Entwicklung auf Hardware-Modellen zu beschleunigen. Da darüber hinaus für eine detaillierte Analyse des SoC ein architekturgenaues Hardware-Modell implementiert werden muss, wurde ein emulationsbasiertes Entwicklungssystem gewählt. Die Kosten für eine solche Entwicklungsumgebung sollten aber gegenüber herkömmlichen Hardware-Emulations-Systemen gesenkt werden. Abschnitt 7.1 vergleicht die in dieser Arbeit vorgestellte Methodik mit bisher in der Praxis üblichen Entwurfsverfahren für eingebettete Software im Hinblick auf diese Rahmenbedinungen.

In Abschnitt 7.2 werden Ergebnisse präsentiert, die sich auf die parallele Entwicklung komplexer Hardware-Komponenten und deren hardware-naher Software für SoCs beziehen. Für die Entwicklung von Hardware-Komponenten sind detaillierte Einblicke in die Hardware bis auf Signalebene notwendig, die denen der Simulation nahe kommen und für die Software-Entwicklung ist die Anbindung eines Software-Debuggers notwendig. Dabei sollen die Laufzeiten für die hardware-nahe Software-Entwicklung eines SoC allerdings beschleunigt und die Turn-around-Zeiten für das Hardware/Software-Debugging verkürzt werden.

7.1 Entwicklung eingebetteter Software auf Hardware-Modellen

In diesem Abschnitt werden Ergebnisse vorgestellt, die für die Software-Entwicklung mit einem emulationsbasierten Mikrokontroller-IP-Kern erzielt werden können. Um die Entwurfskosten für die Entwicklungsumgebung gegenüber herkömmlichen Hardware-Emulations-Systemen möglichst gering zu halten, musste der TriCore1-Kern auf einem Prototyping-System mit begrenzenten Ressourcen integriert werden. Syntheseergebnisse für diese Integration sind Bestandteil von Abschnitt 7.1.1. Abschnitt 7.1.2 enthält Vergleiche der Laufzeiten für Software-Benchmarks zwischen der neuen Methodik und Entwicklungswerkzeugen für den TriCore1-Mikrokontroller. Be-

trachtungen über die architekturgenaue Emulation des Prozessors und eine Zusammenfassung beschliessen diesen ersten Teil des Kapitels.

7.1.1 Implementierung einer TC1MP-S-Plattform

Innerhalb des plattformbasierten Entwurfs ist es wichtig, eine Basisplattform für bestimmte Anwendungsgebiete zu definieren. Beispielsweise wurde in Kapitel 3.1.4 ein Evaluierungsboard für die Implementierung von Telematik-Anwendungen im Automobilbereich vorgestellt. Diese Plattform enthält unter anderem einen im Fahrzeugbau üblichen CAN-Kontroller auf dem Chip.

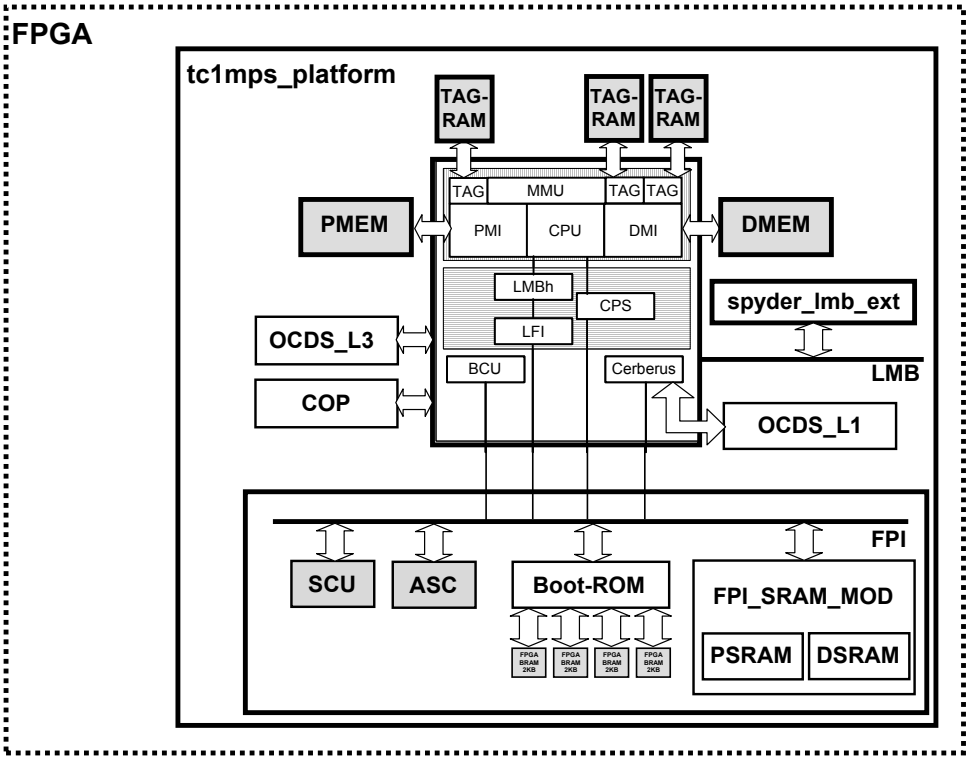


Abbildung 7.1: Implementierung einer minimalen TC1MP-S-Plattform.

Abbildung 7.1 zeigt die Architektur einer minimalen TC1MP-S-Plattform, die für die Entwicklung eingebetteter Hardware/Software-Systeme verwendet werden kann. Der Entwurf enthält neben dem TC1MP-S-Kern auch ein Submodul mit dem FPI-Bus und vier Peripheriemodulen. Neben der Anbindung eines Boot-ROMs und einem Speicherkontroller sind ein System-Steuerungs-Modul (*System Control Unit (SCU)*) und eine serielle Schnittstelle (*Asynchronous Synchronous Control (ASC)*) implementiert worden. Diese Konfiguration hat sich als sinnvolle Ausgangsplattform für die Erweiterung um zusätzliche Hardware-Komponenten erwiesen. Die SCU dient

dem GNU-Debugger GDB zur Identifikation des Chips und überwacht darüber hinaus das Gesamtsystem. Mit der ASC-Schnittstelle können Meldungen des Systems auf eine serielle Schnittstelle herausgegeben und Kommandos von einem PC entgegengenommen werden. Im Folgenden wird beschrieben, wie diese Plattform effizient auf dem Spyder-System integriert werden kann.

Implementierung der Speicherhierarchie des TC1MP-S

Der TC1MP-S hat eine Reihe unterschiedlich schneller und großer Speichermodule zur Verfügung. An dieser Stelle werden nur die Speicher für Programme und Daten betrachtet. Die Implementierung der TAG-RAMs für die Verwaltung der Caches wird weiter unten vorgestellt.

Für die Implementierung der Speicher des TC1MP-S stehen auf dem Spyder-System zum Einen Block-RAM- sowie verteilte Select-RAM-Ressourcen des FPGAs und zum Anderen zwei SRAM- und SDRAM-Chips des Prototyping-Boards selbst zur Verfügung. Wie in Abschnitt 6.3.2 gezeigt wurde, lassen sich sowohl die Block-RAMs als auch die Spyder SRAMs sehr gut für die Implementierung der Speicher einsetzen. Dies ist insbesondere deshalb interessant, da beide die Programmdaten *vor* der Ausführung aufnehmen und halten können. Dadurch ist keine Debugging-Umgebung notwendig, um Programme auf das Spyder-Board herunterzuladen. Gleiches gilt auch für die Implementierung eines vollständigen TC1MP-S-Systems mit integrierter Software-Debugger-Anbindung auf dem Spyder-Board.

In Abbildung 7.2 sind Syntheseergebnisse für die Implementierung eines TC1MP-S-Systems mit den beiden Peripheriekomponenten SCU und ASC aufgeführt. Mit den 96 Block-RAMs[1] des FPGAs konnten für den TriCore1 jeweils 32KB Programm- und Daten-Scratchpad-RAMs und jeweils 32KB Programm- und Daten-Speicher am FPI-Bus implementiert werden. Dabei kann aufgrund der Schnittstellen zum TC1MP-S nicht die volle Kapazität der Block-RAMs für die Scratchpad-RAMs genutzt werden. Insgesamt stellt ein Xilinx Virtex-II-8000-FPGA dem Entwickler 168 Block-RAM-Einheiten zur Verfügung. Neben dem auf insgesamt 64KB begrenzten Speicher für die Software-Entwicklung stellt vor allem diese Ressourcenbeschränkung ein Problem dar. Einerseits können so weniger Block-RAMs für die Implementierung interner Logikanalysatoren, wie beispielsweise ChipScope Pro, verwendet werden. Dies führt dazu, dass weniger Signale aufgezeichnet werden können. Andererseits stehen dadurch auch weniger Block-RAMs für die Implementierung weiterer Peripheriemodule für den TC1MP-S zur Verfügung.

Werden die Speicher des TC1MP-S dagegen mit den externen RAM-Bausteinen des Spyder-Systems implementiert, lassen sich deutlich FPGA-Ressourcen einspa-

[1] Mit einem Block-RAM können bis zu 2KB Daten gespeichert werden.

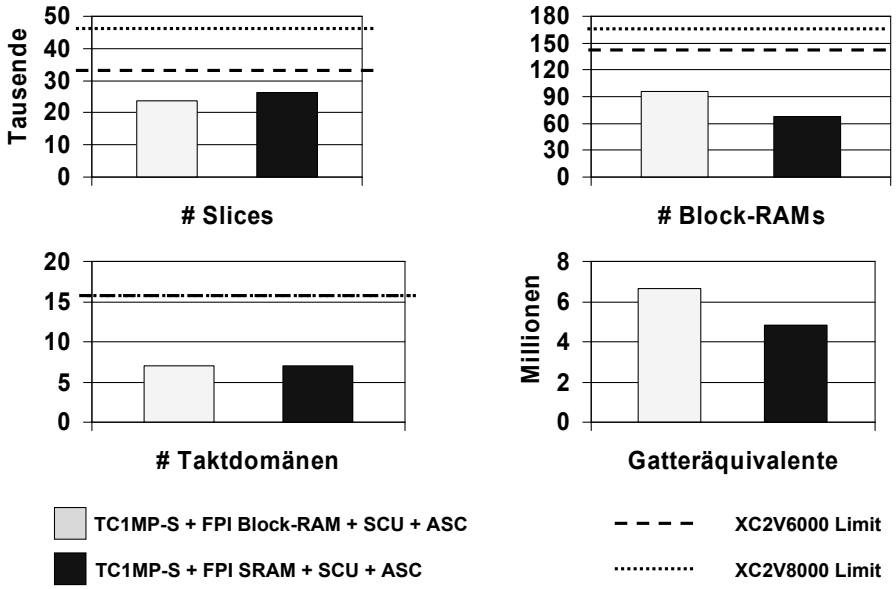


Abbildung 7.2: Syntheseergebnisse für die Implementierung der TC1MP-S-Speicherhierarchie in FPGA-Block-RAMs und Spyder SRAMs.

ren. Die 68 Block-RAMs des FPGAs werden hier für 64KB Scratchpad-RAM und 8KB Boot-ROM verwendet. Die Einsparung der FPGA-Ressourcen zeigt sich auch deutlich an der Anzahl der Gatteräquivalente, die das Werkzeug MAP für den Entwurf angibt. Hier wird die Implementierung des Speichers in den Block-RAMs mitberechnet, was insgesamt eine Anzahl von ca. 6,7 Millionen Gatteräquivalenten ergibt. Da die Speicher bei der SRAM-Lösung vom MAP-Werkzeug nicht gesehen werden, wird dort die Anzahl der Gatteräquivalente *nur* auf ca. 4,8 Millionen beziffert.

Implementierung der TAG-RAMs des TC1MP-S

Die Implementierung der Caches eines Prozessors stellt eine große Herausforderungen beim Rechnerentwurf dar. Um den Prozessor nicht zu verlangsamen müssen diese möglichst nahe am Kern liegen und schnell angesteuert werden können. Beim TC1MP-S sind die Caches deshalb als Teil der Scratchpad-RAMs implementiert. Für die Verwaltung der Caches werden von Infineon entsprechende HDL-Module mitgeliefert. In Tabelle 7.1 sind Syntheseergebnisse für die oben beschriebene TC1MP-S-Plattform mit 4 bzw. 8KB Programm- und Datencache dargestellt. Dabei wurden die TAG-RAMs mit den HDL-Modulen von Infineon implementiert. Die verwendeten Synthesewerkzeuge bilden diese Module auf die Logikressourcen (*Slices*) des FPGAs ab.

Wie aus den Ergebnissen ersichtlich ist, beansprucht eine Implementierung des

	TC1MP-S 4 KB P/D-Caches					TC1MP-S 8 KB P/D-Caches				
	Slices	E/A-Anschl.	Block-RAMs	Takt-dom.	Gatter-äquiv.	Slices	E/A-Anschl.	Block-RAMs	Takt-dom.	Gatter-äquiv.
Verwendet	46.590	233	68	5	5.162.014	51.245	233	68	5	5.348.448
Verfügbar	46.592	824	168	16	–	46.592	824	168	16	–
Auslastung	**99%**	28%	40%	31%	–	**109%**	28%	40%	31%	–

Tabelle 7.1: Syntheseergebnisse für die Implementierung des TC1MP-S mit 4KB und 8KB Programm- und Datencaches in den Slices des FPGAs.

TC1MP-S mit 4KB großen Programm- und Datencaches, bei der die TAG-RAMs mit Logikressourcen des FPGAs realisiert sind, nahezu das gesamte FPGA. Eine Implementierung mit 8KB Programm- und Datencache ist dagegen nicht mehr möglich. Da durch die Verwendung der Spyder SRAMs für die Realisierung von Programm- und Datenspeicher Block-RAM-Ressourcen eingespart wurden, können diese nun für die Partialemulation der TAG-RAMs verwendet werden. Der deutliche Unterschied der Ressourcenauslastung des Xilinx Virtex-II-8000-FPGAs durch diese Lösung ist in Abbildung 7.3 dargestellt.

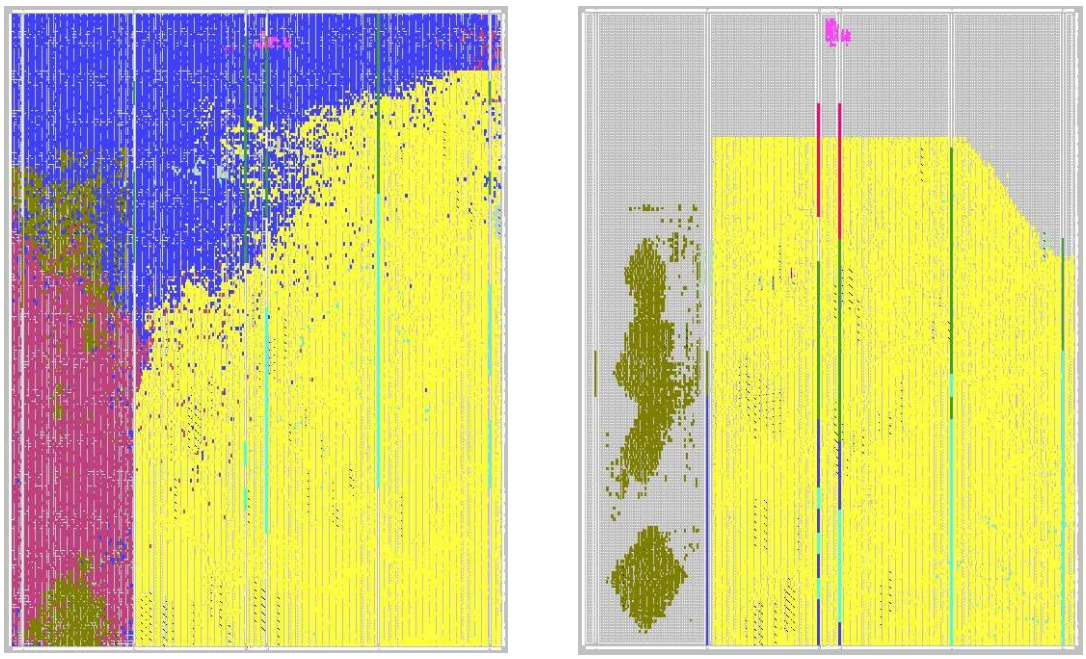


Abbildung 7.3: Platzierung der TAG-RAMs für 4KB P/D-Caches mit Logikressourcen (a) bzw. Block-RAMs (b).

Abbildung 7.3 (a) zeigt die Implementierung der TAG-RAMs in den Logikressourcen des FPGAs und Abbildung 7.3 (b) zeigt die Implementierung der TAG-RAMs in den Block-RAMs des FPGAs. Dort sind sie als rote (PTAG) und blaue (DTAGs) vertikale Streifen auf dem FPGA zu erkennen. In der Abbildung 7.3 (a) sind diese Module entsprechend über das FPGA verteilt. Der TriCore1-Kern selbst ist gelb dargestellt, das Peripheriemodul mit SCU und ASC braun. Die Scratchpad-RAMs schließlich, in denen auch die Programm- und Datencaches untergebracht sind, sind als dunkelgrüne (Programm) und petrolfarbene (Daten) vertikale Streifen in beiden Abbildungen zu erkennen, da diese auch in FPGA Block-RAMs implementiert wurden.

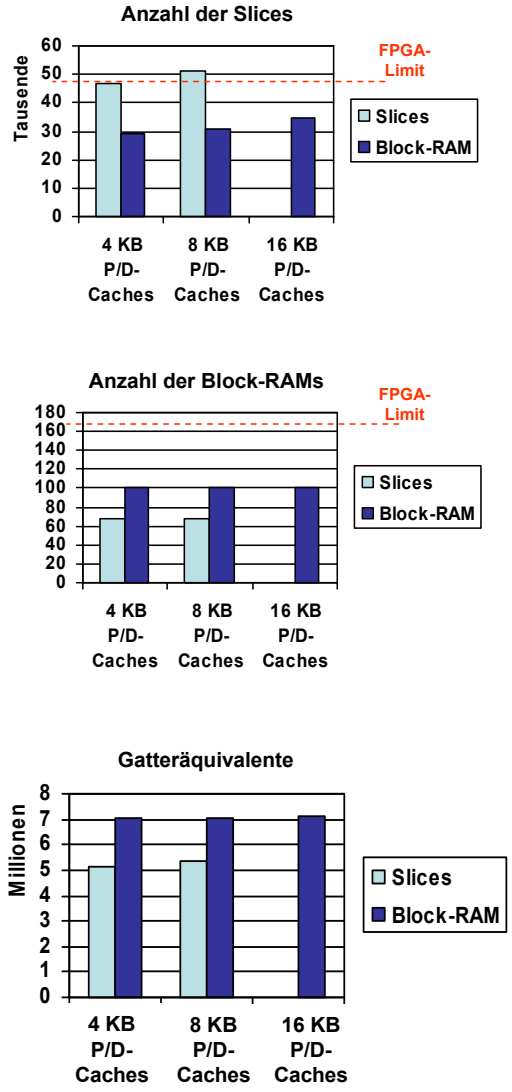


Abbildung 7.4: Vergleich der Ressourcenauslastung verschiedener TAG-RAM Implementierungen.

In Abbildung 7.4 ist eine Gegenüberstellung der Ressourcenauslastung für die unterschiedlichen Implementierungen der TAG-RAMs für die Cacheverwaltung dargestellt. Wie auch hier ersichtlich ist, findet eine TC1MP-S-Implementierung mit 8KB Programm- und Datencaches auf dem größten auf dem Spyder-System verfügbaren Virtex-II-8000-FPGA keinen Platz mehr. Der Ressourcenverbrauch der beiden zusätzlichen Peripheriemodule spielt dabei mit 601 Slices (SCU) und 840 Slices (ASC) keine gravierende Rolle. Durch die Nutzung der Block-RAMs lässt sich damit ein TC1MP-S-System mit bis zu 16KB Programm- und Datencaches realisieren. Die Anzahl der verwendeten Block-RAMs bleibt trotzt steigender Cachekapazität deshalb konstant, da aufgrund der Ansteuerung der TAGs nicht die gesamte Block-RAM-Komponente als TAG-RAM genutzt werden kann.

7.1.2 Laufzeitmessungen für eingebettete Software

Für die Software-Entwicklung in einer frühen Phase des Entwurfsprozesses können unterschiedliche Werkzeuge eingesetzt werden, die alle einen unterschiedlichen Abstraktionsgrad von der zugrundeliegenden Hardware haben. Bei bisherigen Entwicklungsmethoden lässt sich im Prinzip sagen, dass je detaillierter ein Modell der Hardware nachgebildet werden muss, umso größer werden die Laufzeiten. Die Software-Entwicklung mit einem HDL-Simulator ist deshalb für größere Software-Programme nicht mehr praktikabel. Dies verdeutlicht Tabelle 7.2, in der Laufzeiten für ein einfaches *Hello world!*-Programm angegeben sind. Das Programm wurde mit dem HDL-Simulator Modelsim [68] auf einem Intel Pentium 4 mit 2,8 GHz und 1GB Hauptspeicher simuliert. Die Emulation auf Spyder und das TriBoard sind jeweils mit 12 MHz getaktet.

	HDL-Simulator	Emulation auf Spyder	TriBoard TC1920A
Laufzeit: Hello world!	4 Min. 24 Sek.	1,3 ms	7.262 ms

Tabelle 7.2: Laufzeitvergleich für ein einfaches *Hello world!*-Programm.

In dieser Arbeit werden deshalb die Laufzeitergebnisse für Software auf dem emulierten TriCore1-Modell des Spyder-Systems mit einem Befehlssatzsimulator (BSS) und einem Evaluierungsboard verglichen. BSS werden vor allem für die Entwicklung hardware-unabhängiger Software verwendet und Evaluierungsboards sind oft auf eine bestimmte Anwendungsdomäne eingeschränkt.

Befehlssatzsimulator TSIM

Der TSIM [50] ist ein BSS für die TriCore-Mikrokontroller-Familie von Infineon, der in allen gängigen Software-Debuggern für den TriCore integriert ist. Der Simulator kann für die Performanzanalyse und hardware-unabhängige Software-Entwicklung verwendet werden. Er ist bezüglich Speicherhierarchie, Unterbrechungsmechanismus und der Abbildung von Peripheriemodulen in den Adressraum des Mikrokontrollers programmierbar.

Bei dem Simulator handelt es sich um ein befehlssatzgenaues (*instruction accurate*) Modell des TriCore, d.h. Effekte der Prozessorpipeline, Speicherschnittstellen- und Busprotokolle werden nicht beachtet. Der Simulator führt einen Befehl pro Takt des TriCore aus. Das algorithmische Modell kann die Unterbrechungsverwaltung des Kontrollers nachbilden. Die Register werden als Variablen und die Befehle als Funktionen, die auf den Variablen arbeiten, modelliert. Um die Simulation zu beschleunigen, werden die Befehle nur einmal dekodiert und dann in einem Puffer gespeichert

Die Genauigkeit des TSIM hängt stark von der untersuchten Anwendung ab. Die Zyklenzahl für die Programme ist dabei nur geschätzt. Eine Modellierung der Caches konnte bei den Messungen nicht nachgewiesen werden. Unterschiedliche Cacheersetzungsstrategien werden nicht modelliert. Allerdings wird die superskalare Architektur bei der Angabe der Zyklen berücksichtigt. Darüber hinaus werden keine Unterbrechungen der CPU durch Buszugriffe, Limitierungen der Ports des Registersatzes und Datenabhängigkeiten nachgebildet.

Evaluierungsboard TriBoard TC1920A

Für die verschiedenen Varianten des TriCore1-Mikrokontrollers gibt es Evaluierungsboards, die mit einem entsprechenden TriCore1-Mikrokontroller-Chip und den anwendungsspezifischen physikalischen Schnittstellen für die On-chip-Peripheriemodule ausgestattet sind. Stellvertretend soll an dieser Stelle das Evaluierungsboard TriBoard TC1920A [48] vorgestellt werden, welches für die Performanzvergleiche mit dem Spyder-System verwendet wurde.

Der TC1920 basiert auf der TriCore1.3-Architektur und ist speziell für Infotainment-Anwendungen im Automobil zugeschnitten. Dafür kombiniert er sowohl Peripheriemodule wie etwa CAN und J1850 aus dem Automobilbereich und Standardschnittstellen wie ADC, SSC/SPI, UART, IrDA, I^2S und I^2C auf einem Chip. Die Anwendungen reichen von Navigationssystemen, Internet-Radios bis hin zu Notrufsystemen in Autos.

Das TriBoard besitzt für die oben beschriebenen Peripheriemodule des Chips die entsprechenden physikalischen Schnittstellen auf dem Board. Diese werden durch zusätzliche Speicherchips wie etwa 4MB FlashRAM oder 128MB SDRAM ergänzt.

Für das Software-Debugging stehen Schnittstellen für die OCDS Level 1 und 2 zur Verfügung, durch die die Anbindung einer Cross-Entwicklungsumgebung für eingebettete Software realisiert werden kann.

EEMBC-Benchmarks zur Plattformbewertung

Für die Performanzanalyse der unterschiedlichen Plattformen werden eine Reihe von Benchmarks des *Embedded Microprocessor Benchmark Consortiums (EEMBC)* [31] verwendet. Das EEMBC-Konsortium wurde 1997 gegründet und hat es sich zur Aufgabe gemacht, sinnvolle Performanz-Benchmarks für Hardware und Software eingebetteter Systeme zu entwickeln. Durch die Mitarbeit nahmhafter Mikroprozessor-Hersteller, unter denen unter anderem ARM, IBM, TI und Infineon zu finden sind, wurden die EEMBC-Benchmarks zu einem de facto Standard für die Evaluierung von eingebetteten Prozessoren, Compilern und Java-Implementierungen.

Da die Benchmark-Suite kostenpflichtig ist, wurden sechs der Algorithmen für den TriCore1-Mikrokontroller neu implementiert. Als Benchmarks werden verwendet:

Schnelle Fourier Transformation (FFT): Die Fourier Transformation ist ein fundamentales Verfahren aus der Signalverarbeitung und überführt Signale aus der Zeit-Domäne in die Frequenz-Domäne. Die schnelle Fourier Transformation (*Fast Fourier Transformation (FFT)*) stellt ein vereinfachtes Verfahren der Fourier Transformation dar, welches nur auf diskreten Werten arbeitet. Um insbesondere für den BSS TSIM aussagekräftige Messwerte zu erhalten, operiert die FFT-Funktion auf einem Eingabevektor von 8 Werten und wird in einer Schleife 1000 Mal aufgerufen.

Diskrete Kosinus Transformation (DCT): Die DCT wird insbesondere im Bereich der Kompression von Bildinformationen verwendet. Sie ist beispielsweise in JPEG- und MPEG-Kompressionsverfahren zu finden. Bei der DCT wird ein Signal aus einem Wertebereich in einen Raum mit Kosinus-Frequenzen transformiert. Im vorliegenden Fall arbeitet die DCT auf einer 8×8-Matrix.

JPEG-Algorithmus: Beim JPEG-Algorithmus (*Joint Pictures Experts Group (JPEG)*) werden Bilder komprimiert, indem Frequenzanteile entfernt werden, die für das menschliche Auge nur schwer oder gar nicht wahrnehmbar sind. Durch das Entfernen von Frequenzanteilen gehen allerdings Informationen verloren, die später nicht wieder rekonstruiert werden können. Oft ist die Qualität der Komprimierung einstellbar. Für den Benchmark wurde eine Bild

mit acht Bildpunkten, von denen jeder Bildpunkt durch eine 8×8-Matrix beschrieben ist, mit dem JPEG-Algorithmus komprimiert.

LU-Zerlegung: Bei der LU-Zerlegung oder auch Dreieckszerlegung wird eine Matrix A in das Produkt einer linken unteren Dreiecksmatrix L (*lower*) und einer rechten oberen Dreiecksmatrix U (*upper*) so zerlegt, dass die Gleichung $P * A = L * U$ gilt. Die Matrix P bezeichnet die Permutationsmatrix, die in jeder Spalte und jeder Zeile genau eine Eins und sonst Nullen besitzt. Für die Messungen wurde die Dreieckszerlegung auf eine 3×3-Matrix angewendet.

Packet flow Der *Packet flow Benchmark* führt eine Untermenge der Paketweiterleitung eines Netzwerk-Routers durch. Mit ihm kann die potentielle Performanz eines Mikroprozessors gemessen werden, der in einem IP-Router-System zum Einsatz kommt. Üblicherweise überprüft ein Router den IP-Kopf im Rahmen des Weiterleitungsprozesses eines Paketes. Dazu wird zunächst der Kopf des *Link-Layers* einer eingehenden Nachricht entfernt, der IP-Kopf modifiziert, und der *Link-Layer*-Kopf wieder hinzugefügt. Im vorliegenden Benchmark ist der *Link-Layer*-Kopf allerdings bereits entfernt worden.

Viterbi-Dekoder: Ein Viterbi-Dekoder wird zur Dekodierung von Datensignalen eingesetzt, die über einen rauschbehafteten Kanal gesendet wurden, wie es beispielsweise bei GSM, WLAN-Netzen oder der Video- und Audioübertragung der Fall ist. Dazu werden die Daten vor der Übertragung zunächst z.B. mit einem *Reed-Solomon-Encoder* so kodiert, dass der Viterbi-Algorithmus die Daten mit einer Maximum-Likelihood-Dekodierung wieder rekonstruieren kann. Der Benchmark führt zunächst auf 32 Eingabewerten eine Kodierung durch, simuliert dann einen gestörten Übertragungskanal und dekodiert die Daten am Ende wieder. Dieser Prozess wird insgesamt 100 Mal durchlaufen.

Versuchsaufbau für die Messungen

Für die Laufzeitmessungen mit dem BSS TSIM wurde ein PC mit Intel Pentium 4 Prozessor mit 2,8 GHz und 1GB Hauptspeicher verwendet. Die Programme wurden mit dem GDB ausgeführt. Die Ausgabe des Simulators ist eine Datei, die die Laufzeit der Simulation auf eine Sekunde genau enthält. Darüber hinaus werden Zugriffsstatistiken auf die Adress- und Datenregister sowie eine Statistik über Anzahl

und Art der ausgeführten Befehle insgesamt und nach Befehlstypen ausgegeben. Es wird dann noch die geschätzte Laufzeit des Programms in Zyklen angegeben. Der Simulator liefert allerdings keine Statistiken über Cachezugriffe und Cachemisses.

Die Messungen mit dem TriBoard TC1920A und dem Spyder-System wurden mit einem Logikanalysator durchgeführt. Für die Ermittlung der Laufzeit mit dem Spyder-Board wurde das in Abschnitt 5.2.5 bereits erwähnte Trigger-Modul verwendet, welches jeweils am Anfang und Ende des zu vermessenden Algorithmus ein Signal auf der Spyder-Backplane ausgibt. Beim TriBoard wird als Trigger der Status einer LED verwendet, welcher sowohl optisch als auch an einem Stecker des Boards abgegriffen werden kann. Am Anfang einer Messung wird die LED an und am Ende der Messung ausgeschaltet. Beide Boards wurden mit einer Taktfrequenz von 12 MHz betrieben.

Ergebnisse der Laufzeitmessungen

In diesem Abschnitt werden Laufzeitmessungen für die vorgestellten EEMBC-Benchmarks und die verschiedenen TriCore1-Software-Entwicklungsplattformen, die zur Verfügung stehen, vorgestellt und bezüglich ihrer Effizienz und Architekturgenauigkeit bewertet. In Abbildung 7.5 sind zunächst einige statistische Daten der Anwendungen dargestellt, die der BSS TSIM in einer Protokolldatei ausgibt.

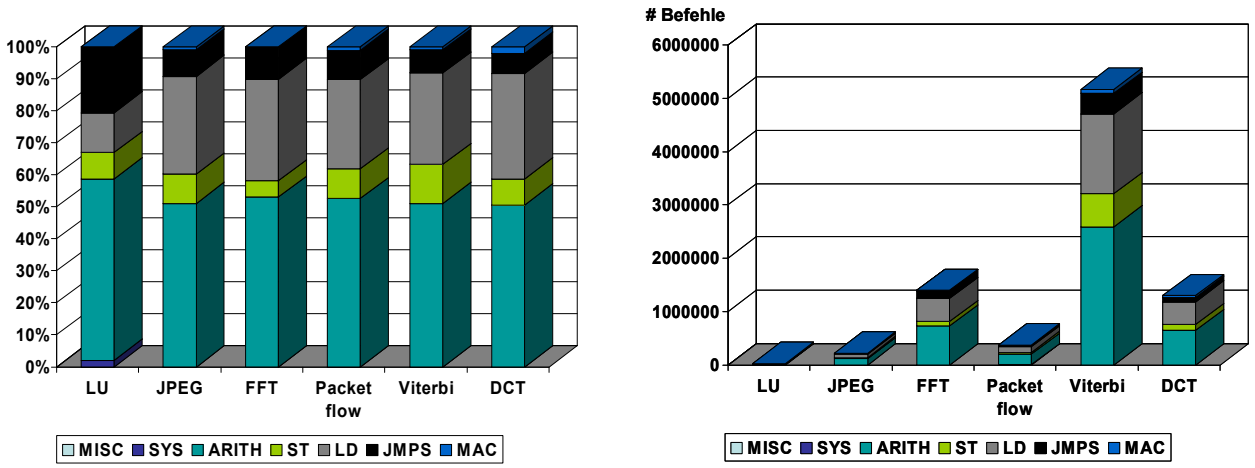


Abbildung 7.5: Befehlsanzahl und -verteilung für die untersuchten EEMBC-Benchmarks.

Die Befehle werden vom BSS in Gruppen eingeteilt. Bei der Matrizen-Zerlegung werden viele Sprungbefehle und arithmetische Operation ausgeführt, wohingegen bei Viterbi und DCT mehr Zugriffe auf den Speicher und weniger Sprünge und arith-

metische Operationen ausgeführt werden. Der Viterbi-Algorithmus stellt mit knapp über 5 Millionen Befehlen den größten Benchmark dar.

In den folgenden Abbildungen sollen zunächst Laufzeitvergleiche zwischen den unterschiedlichen Entwicklungsplattformen aufzeigt werden. Dazu werden unterschiedliche Speicherbereiche für die EEMBC-Programme verwendet, die sich durch unterschiedliche Zugriffszeiten sowie dadurch auszeichnen, dass sie in den Cache geladen werden können oder nicht. Eine entsprechende Einteilung des Speicherbereichs beim TriCore1 wurde in Abbildung 6.3 auf Seite 89 vorgestellt.

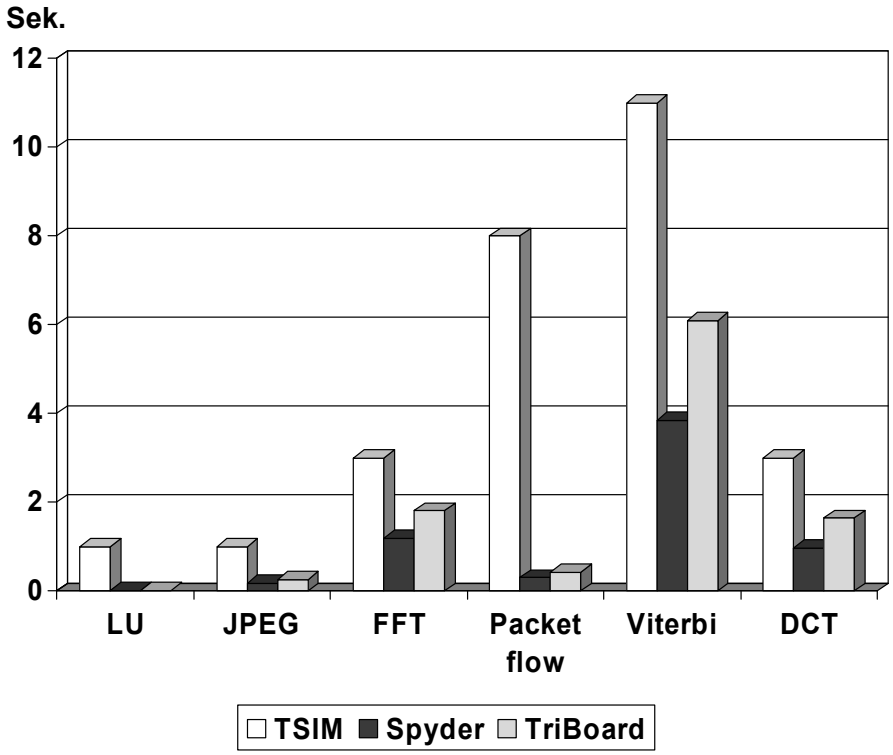


Abbildung 7.6: Laufzeitergebnisse für EEMBC-Benchmarks lokalisiert in Speicherbereichen, die nicht in den Cache geladen werden.

Abbildung 7.6 zeigt zunächst Laufzeitmessungen, bei denen Programm und Daten im Speicher ab Adresse `0xA0000000` abgelegt werden. Dieser Speicherbereich wird nicht in den Cache geladen. Auffallend ist der deutliche Unterschied in der Differenz der Laufzeiten zwischen BSS und Spyder bzw. TriBoard bei den beiden Algorithmen „Packet flow“ und „Viterbi“. Es müssen bei Viterbi zwar deutlich mehr Befehle als bei Packet flow ausgeführt werden, der BSS ist aber dennoch nicht wesentlich langsamer. Das bessere Abschneiden des BSS bei Viterbi ist darauf zurückzuführen, dass der TSIM Befehle nur einmal bei ihrem ersten Auftreten dekodiert und sie dann dekodiert im Speicher hält. Da beide Algorithmen nur wenig Sprünge

verwenden (siehe Abbildung 7.5), zeichnen sich beide durch eine hohe Lokalität des Programmes aus, wodurch der BSS im Vorteil ist.

Insgesamt fällt auf, dass sich eine deutliche Performanzsteigerung bis zu einem Faktor 65 (LU-Zerlegung) mit einer Hardware-Realisierung des TriCore1 gegenüber dem BSS erreichen lässt. Die schnelleren Ausführungszeiten des Spyder-Systems gegenüber dem TriBoard, trotz gleicher Taktfrequenz, rühren daher, dass beim Spyder-System weniger Wartezyklen beim Speicherzugriff eingefügt werden müssen.

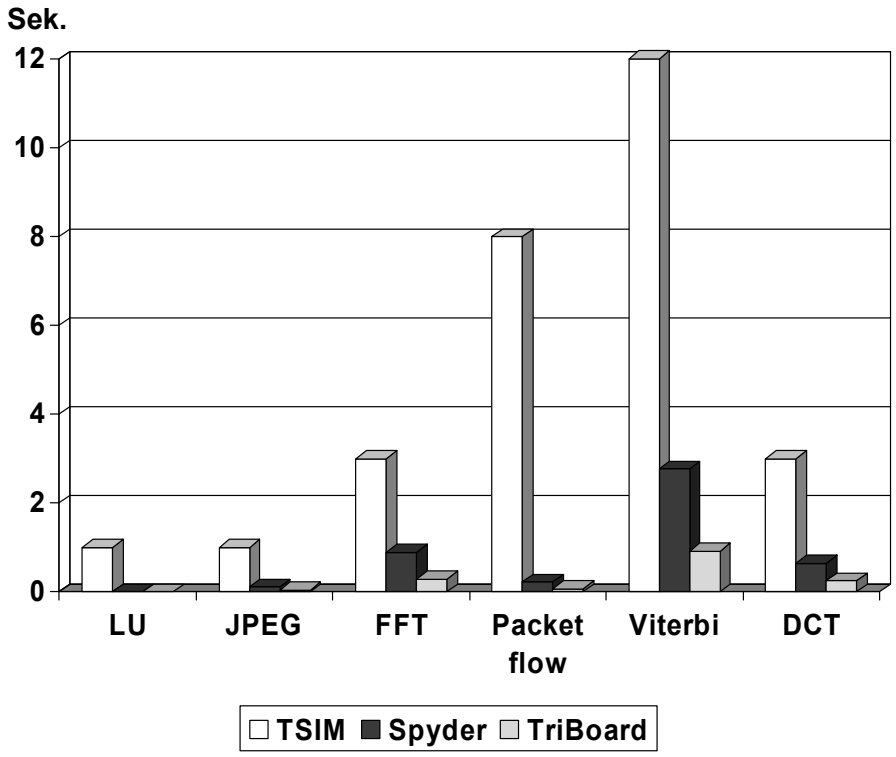


Abbildung 7.7: Laufzeitergebnisse für EEMBC-Benchmarks lokalisiert in Speicherbereichen, die in den Cache geladen werden.

Durch die Verwendung von Caches lassen sich noch deutlich bessere Performanzsteigerungen gegenüber dem BSS erzielen. In Abbildung 7.7 sind Vergleiche für die Ausführung der EEMBC-Anwendungen mit dem BSS und Spyder dargestellt, die ab dem Speicherbereich `0x80000000` für den Programmteil und `0x90000000` für den Datenteil lokalisiert sind. Da beim TriBoard an dieser Stelle kein physikalischer Speicher vorhanden ist, wurde dort für die Messungen Speicher ab der Adresse `0xC0000000` verwendet. Diese Speicherbereiche können alle in den Cache geladen werden. Während die Laufzeiten beim BSS konstant bleiben, konnten sowohl bei Spyder als auch TriBoard Perfomanzsteigerungen erzielt werden. Die schnelleren Ausführungszeiten des TriBoards gegenüber dem Spyder-System sind darauf

zurückzuführen, dass der Speicher ab der Adresse `0xC0000000` am LM-Bus des TriCore1 angeschlossen ist und sich dadurch die Zugriffszeiten reduzieren. Dies steht im Gegensatz zu den Messungen aus Abbildung 7.6.

In Abbildung 7.8 sind Laufzeitvergleiche für Programme dargestellt, die aus den beiden Scratchpad-RAMs des TriCore1 ausgeführt werden. Der Befehlscode wird ab der Adresse `0xD4000000` und die Daten ab der Adresse `0xD0000000` abgelegt. Die beiden EEMBC-Benchmarks JPEG und Packet flow konnten aufgrund ihrer Größe nicht ausgeführt werden, da der Code nicht vollständig in die Scratchpad-RAMs geladen werden konnte. Auch bei diesen Benchmarks sind signifikante Performanzsteigerungen des Spyder-Systems und des TriBoards gegenüber dem BSS zu verzeichnen.

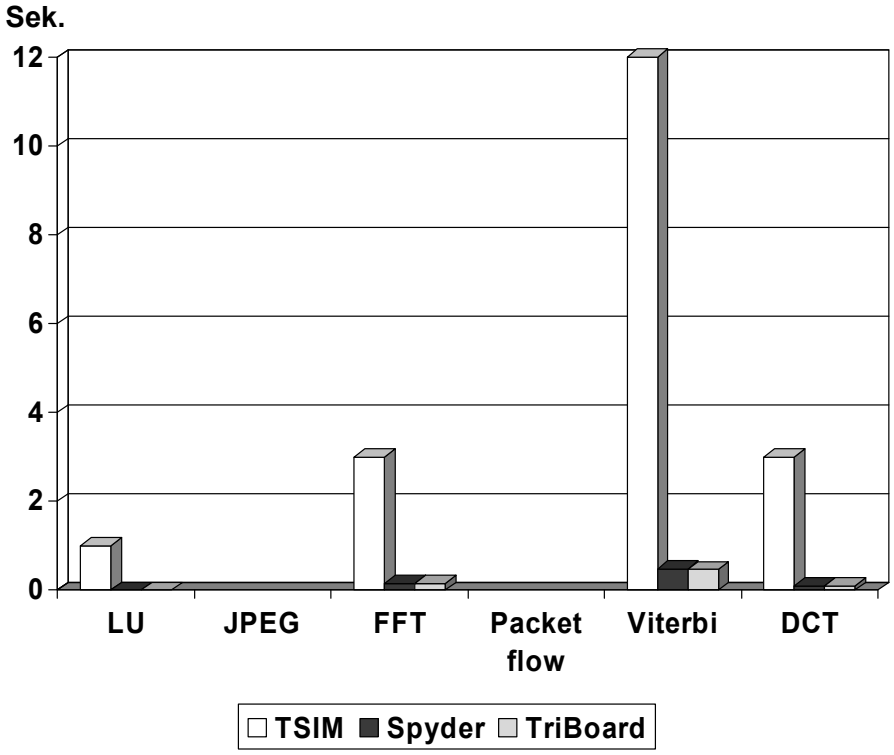


Abbildung 7.8: Laufzeitvergleich für EEMBC-Benchmarks lokalisiert in den Scratchpad-RAMs.

7.1.3 Architekturgenaue Emulation des TriCore1-Mikrokontroller-Kerns

Wie aus Abbildung 7.8 ersichtlich ist, sind die Laufzeiten von Spyder und TriBoard fast identisch. Dies ist darauf zurückzuführen, dass sowohl die Zugriffszeiten des TC1MP-S auf die internen Block-RAMs des Xilinx FPGAs als auch die Zugriffe des

TC1920 auf die internen Scratchpad-RAMs des Chips exakt in einem Takt erfolgen können. An dieser Stelle ist ein architekturgenaues Verhalten des Spyder-Systems zu beobachten.

Um dieses architekturgenaue Verhalten des Spyder-Systems generell zu bewerten, wurden die Ergebnisse der Laufzeitmessungen von Spyder und TriBoard den durch den BSS TSIM abgeschätzten Laufzeiten gegenübergestellt. Die Ergebnisse sind in Abbildung 7.9 dargestellt. Der Simulator TSIM gibt nach Simulationsende die Anzahl der Zyklen an, die der Prozessor für eine Anwendung benötigen würde. Diese wurden für eine Taktfrequenz von 12MHz in absolute Laufzeiten hochgerechnet und in Abbildung 7.9 für den BSS eingetragen.

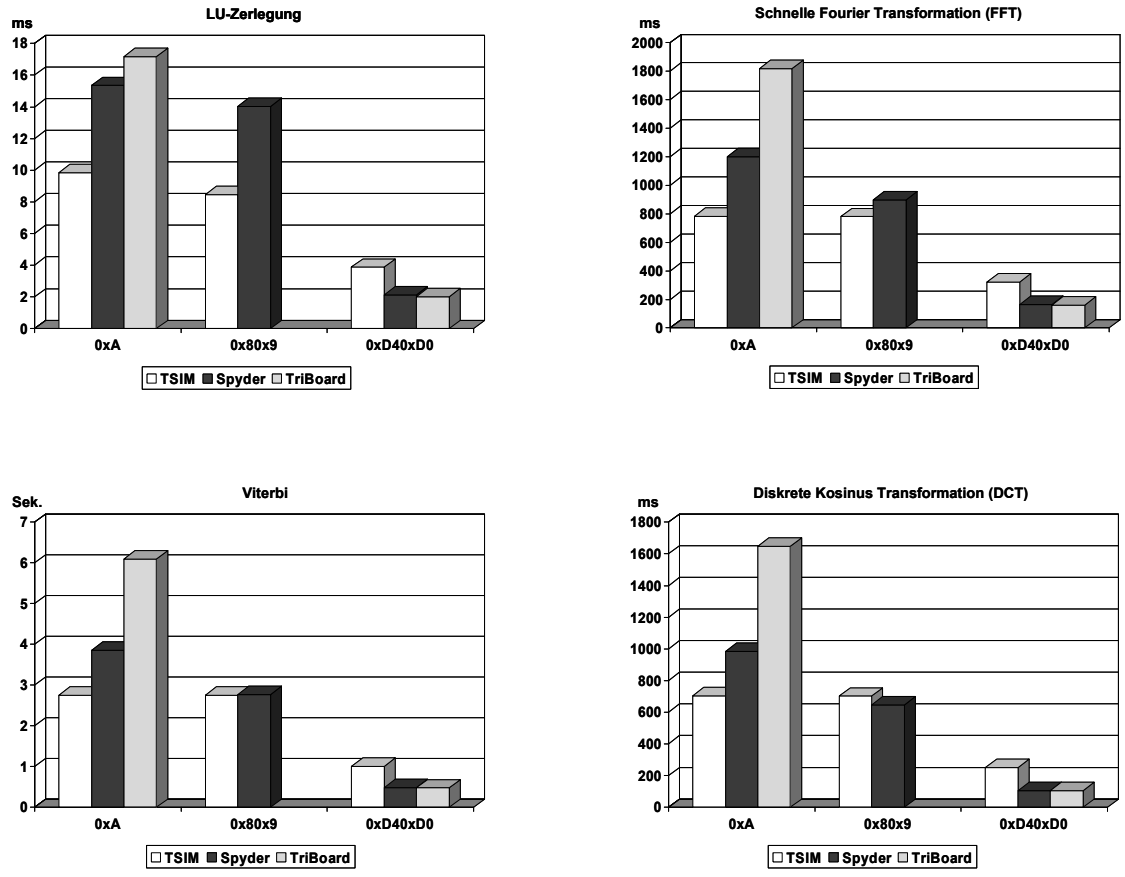


Abbildung 7.9: Vergleich der geschätzten Laufzeiten durch den BSS mit tatsächlich gemessenen Laufzeiten auf der Hardware.

Die Abschätzung der Ausführungszeiten für externen Speicher, der nicht in den Cache geladen wird, wird vom BSS schlecht abgeschätzt. Hier werden schnellere Laufzeiten angegeben, als die Zeiten, die tatsächlich benötigt werden. Der Grund dafür ist in den Zugriffskonflikten zu suchen, die auf dem Bus auftreten und die vom

BSS nicht berücksichtigt werden. Dass der BSS darüber hinaus die Auswirkungen der Caches nicht modelliert, zeigt sich daran, dass die Ausführungszeiten für Programme, die im Speicherbereich ab Adresse `0xA0000000` und `0x8/0x90000000` liegen, gleich angegeben werden. Betrachtet man die Messwerte für Programme aus dem Scratchpad-RAM, so zeigt sich, dass die Laufzeiten des Spyder-Systems und TriBoards sehr ähnlich sind. Dies ist ein Beweis dafür, dass die Scratchpad-RAMs des TriCore1 architekturgenau durch die Block-RAMs des Xilinx FPGAs nachgebildet werden können.

In Abbildung 7.10 sind Laufzeiten dargestellt, die mit unterschiedlichen Realisierungen der TAG-RAMs für die Cacheverwaltung erzielt wurden. Die TAG-RAMs können entweder mit Logikressourcen oder den FPGA Block-RAMs implementiert werden. Aufgrund der beschränkten Ressourcen beziehen sich die Messungen auf einen TriCore1 mit 4KB Programm- und Datencache. Wie die Abbildung 7.10 zeigt, sind die Laufzeiten für die Block-RAM-Variante und die Variante mit den Logikressourcen nahezu identisch. Auch die TAG-RAMs konnten somit mit den Block-RAMs architekturgenau nachgebildet werden.

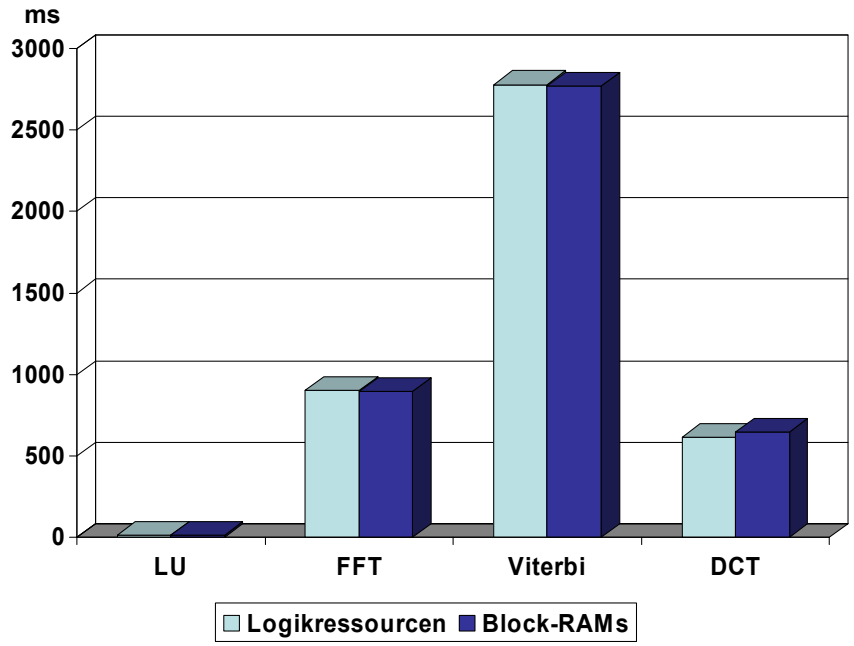


Abbildung 7.10: Laufzeitvergleiche unterschiedlicher Realsierungen der TAG-RAMs im Spyder-FPGA.

7.1.4 Zusammenfassung

In diesem Abschnitt wurden Ergebnisse präsentiert, die eine Beschleunigung der Software-Entwicklung auf einem architekturgenauen Hardware-Modell eines SoC mit der neuen Entwurfsmethodik belegen. Die Kosten für das Prototyping-System konnten dabei gegenüber Hardware-Emulations-Systemen drastisch reduziert werden. So werden die Kosten in [57] für solche Emulations-Systeme mit $0.55 bis $0.89 pro Gatter beziffert. Bei [60] betragen die Kosten für das Prototyping eines komplexen SoC $0.26 pro Gatter auf einem Rapid-Prototyping-System von Aptix [5]. Für das Spyder-System liegen die Kosten zwischen 0.002 und 0.03 Eurocent pro Gatter. Insgesamt ergibt sich mit dem in dieser Arbeit vorgestellten Konzept eine Kosteneinsparung für die Entwicklungsumgebung um ca. 80%.

Dies gelingt durch die Partialemulation von Teilen des SoC. Durch die Verwendung externer SRAM-Bausteine lassen sich erheblich Ressourcen im FPGA einsparen. So konnte die Anzahl der Block-RAMs, die für die Implementierung der Speicherhierarchie des TriCore1 verwendet werden, um 27 % reduziert werden. Gleichzeitig wurde die Speicherkapazität von 64KB auf 4MB erhöht.

Darüber hinaus werden durch die Implementierung der TAG-RAMs des Systems in den Block-RAMs der Xilinx FPGAs 37 % der Logikressourcen des FPGAs eingespart. Hierfür können unter anderem die Block-RAMs verwendet werden, die durch die Implementierung von Speicher in externem SRAM frei werden. Bei der Anbindung der Erweiterungsplattform für die Integration neuer Hardware-Komponenten können ebenfalls Ressourcen eingespart werden, indem die Tristate-E/A-Anschlüsse des FPGAs verwendet werden. Hier konnte die Anzahl der E/A-Anschlüsse, die an den Erweiterungssteckern des Prototyping-Boards benötigt werden, um 49 % reduziert werden.

Die vorgestellten Laufzeitmessungen mit den EEMBC-Benchmarks zeigen dabei die Beschleunigung der Software-Entwicklung mit der neuen Entwurfsmethode gegenüber einem BSS. Dort ist eine durchschnittliche Beschleunigung um einen Faktor 50 zu beobachten. Darüber hinaus bildet das Prototyping-System ein architekturgenaues Modell des TriCore1-Mikrokontrollers nach, wie an den Vergleichen mit dem TriBoard TC1920A gezeigt werden konnte.

7.2 Entwicklung eingebetteter Hardware/Software-Systeme

Das zweite Ziel dieser Arbeit besteht darin, durch eine parallele Entwicklung neuer Hardware-Komponenten und der dazugehörigen hardware-nahen Software auf der Basis einer bestehenden Hardware-Plattform den Entwurfsprozess für SoCs zu beschleunigen. In Abschnitt 7.1.2 wurde zwar deutlich, dass die Software-Entwicklung

mit einem Evaluierungsboard in etwa die gleichen Laufzeiten wie die Emulation auf dem Spyder-System aufweist, die hier aufgezeigte Entwicklungsmethodik hat aber gegenüber dem Evaluierungsboard den Vorteil, dass neue Hardware-Komponenten leicht integriert werden können. Dies stellt für die Entwicklung neuer SoCs einen wesentlichen Vorteil dar. In diesem Abschnitt wird anhand des TriCore1-Mikrokontrollers und drei Hardware-Komponenten gezeigt, wie diese Ziele mit der in dieser Arbeit beschriebenen Entwurfsmethodik verwirklicht werden können. Die untersuchten Hardware-Komponenten werden in Abschnitt 7.2.1 kurz vorgestellt. Zur Beschleunigung des Entwurfsprozesses sind zum Einen die Ausführungszeiten für die Software zu verkürzen und zum Anderen müssen die Synthesezeiten für das SoC gesenkt werden, um nach dem Auffinden und der Beseitigung eines Fehlers in der Hardware schnell wieder mit dem Debugging fortfahren zu können. Abschnitt 7.2.2 präsentiert Syntheseergebnisse für die Hardware-Komponenten und Abschnitt 7.2.3 zeigt das Debugging der Hardware und Software auf. In Abschnitt 7.2.4 folgen dann Laufzeitmessungen hardware-naher Software mit der hier vorgestellte Entwicklungsmethodik und bisher verwendeten Entwurfsmethoden. Abschnitt 7.2.5 fasst die wesentlichen Ergebnisse zusammen.

7.2.1 Überblick über die Hardware-Komponenten

Als Hardware-Komponenten kommmen zwei Peripheriebausteine des TriCore1 sowie ein rekonfigurierbarer Kommunikationskanal zum Einsatz. Bei den beiden TriCore1-Peripheriekomponenten handelt es sich um einen Systemzeitgeber (*System Timer Unit (STM)*) und eine synchrone serielle Schnittstelle (*Synchronous Serial Control (SSC)*), die z.B. Bestandteil des in Abschnitt 7.1.2 beschriebenen TC1920 sind. Der rekonfigurierbare Kommunikationskanal soll unter anderem in zukünftigen Produkten zusammen mit TriCore1-Mikrokontrollern eingesetzt werden.

Systemzeitgeber (STM)

Das STM-Modul des TriCore1 ist ein 56-Bit-Zähler, der mit der Taktfrequenz des Mikrokontrollers betrieben wird und sofort nach dem Einschalten der Betriebsspannung mit dem Zählen beginnt. Der STM kann nur durch das Ausschalten des Gerätes beeinflusst werden. Sonst ist der Zähler während des gesamten Produktlebenszykluses in Betrieb. Bei einer Taktfrequenz von $f_{hw\_clk} = 40MHz$ zählt der Zeitgeber $2^{56}/f_{hw\_clk} = 57.1$ Jahre. Das STM-Modul stellt eine Hardware-Komponente dar, die nicht mit der Außenwelt kommuniziert sondern nur vom TC1MP-S selbst angesprochen werden kann.

Synchrone serielle Schnittstelle (SSC)

Mit der synchronen seriellen Schnittstellenkomponente können bis zu sieben Slave-Geräte angesteuert werden. Die Schnittstelle unterstützt sowohl Master- als auch Slave-Funktionalität und kann im halb-duplex oder voll-duplex Modus betrieben werden. Der Unterschied zur asynchronen Kommunikation besteht darin, dass über eine zusätzliche Leitung ein Taktsignal vom Master an den Slave übertragen wird, mit welchem der Slave die Daten synchron übernehmen kann. Mit der SSC-Komponente können SPI-kompatible Geräte angesteuert werden.

Die Parameter für die Kommunikation sind sehr flexibel einstellbar. So lassen sich beispielsweise Datenbreite, Schieberichtung, Taktpolarität und Phase softwareseitig einstellen. Deshalb sind sehr komplexe Software-Treiber für ihren Betrieb notwendig, die ausführlich getestet werden müssen. Darüber hinaus wurde die SSC-Schnittstelle als Kommunikationspartner für den rekonfigurierbaren Kommunikationskanal verwendet, der im nächsten Abschnitt näher beschrieben wird.

Rekonfigurierbarer serieller Kommunikationskanal (RSK)

Der RSK ist eine von Infineon neu entwickelte serielle Kommunikationskomponente für deren Evaluierung das Spyder-System verwendet wurde. Dabei ist es mit dem RSK möglich, mehrere serielle Kommunikationsprotokolle parallel zu betreiben. Die Komponente besteht dafür aus mehreren Kanälen, deren Anzahl konfigurierbar ist und die unabhängig voneinander betrieben werden können. Die Kanäle wiederum können eines oder mehrere der Protokolle ASC, LIN, SSC/SPI, IIC oder IIS implementieren. Welcher Kanal welches Protokoll betreibt ist zur Laufzeit konfigurierbar. Der RSK hat dazu zwischen Protokoll-Modul und den Anschlüssen des Chips noch ein Kommunikationsnetz, welches je nach Protokoll die entsprechenden Anschlüsse treibt oder liest.

Abbildung 7.11 zeigt eine Konfiguration des RSK, die mit dem Spyder-System evaluiert wurde. Der RSK besteht dort aus zwei Kanälen, die die Protokolle ASC, SSC und IIC ausführen können. Die Komponente wurde zur Laufzeit so initialisiert, dass Kanal 0 das ASC-Protokoll und Kanal 1 das SSC-Protokoll ausführt. Aufgrund der extremen Konfigurierbarkeit der Hardware-Komponente sind ausgiebige Tests aller Funktionen des RSK unumgänglich, um Fehler im Entwurf ausschließen zu können, bevor die Komponente in einem SoC eingesetzt wird.

7.2.2 Syntheseergebnisse

In diesem Abschnitt werden Syntheseergebnisse für die Implementierung und Evaluierung der oben beschriebenen Hardware-Komponenten vorgestellt. Diese wur-

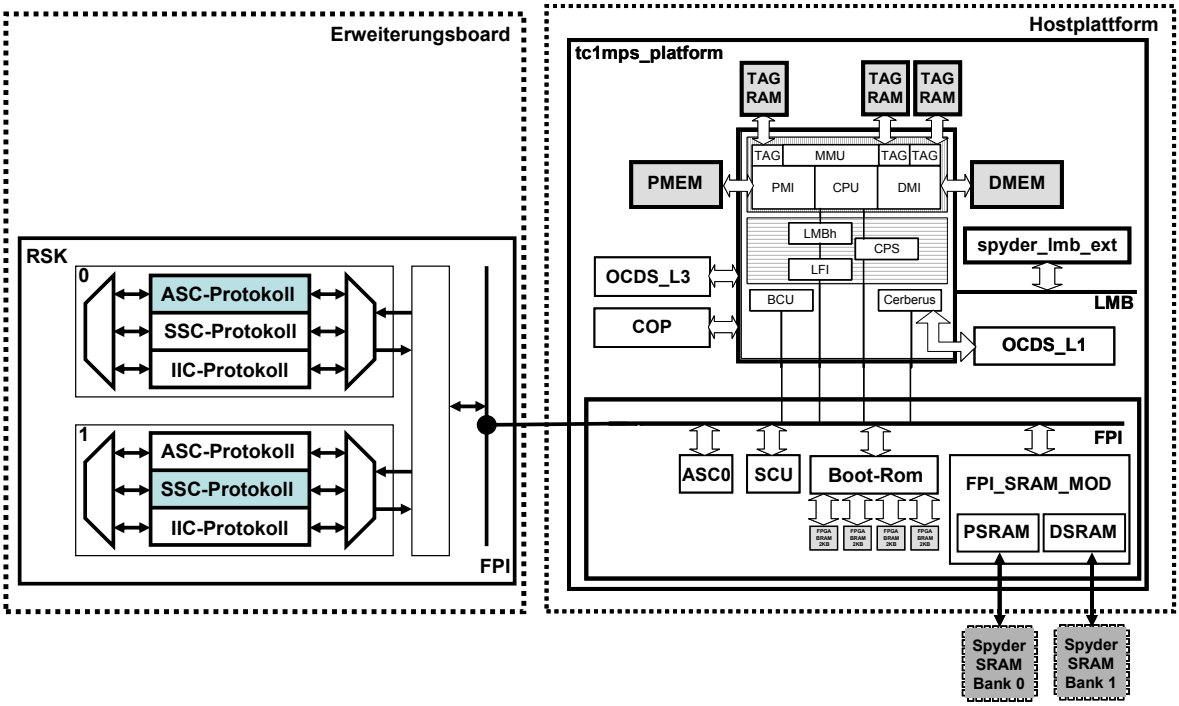


Abbildung 7.11: Implementierung der RSK-Komponente auf dem Prototyping-System.

den mit der in Abschnitt 7.1.1 beschriebenen TC1MP-S-Plattform kombiniert. Die Hardware-Komponenten wurden zum Einen zusammen mit dem TC1MP-S auf einem Spyder-Board und zum Anderen mit Hilfe einer Erweiterungsplattform implementiert.

Abbildung 7.12 zeigt Syntheseergebnisse für die Realisierung des SoCs auf einem Spyder-Board. Für die Evaluierung der Hardware-Komponenten wird eine TC1MP-S-Plattform mit 0KB Programm- und Datencache verwendet, um möglichst viele Logikressourcen für die Implementierung der zusätzlichen Komponenten zur Verfügung zu haben. Wie aus Abbildung 7.12 ersichtlich ist, können alle Hardware-Komponenten im Rahmen der Ressourcen des FPGAs implementiert werden. Die jeweils maximal zur Verfügung stehenden Ressourcen des FPGAs sind durch die Obergrenze der Diagramme für die Slices, Block-RAMs und Taktdomänen bestimmt. Da es sich bei der Anzahl der Gatteräquivalenten um eine Abschätzung für eine ASIC-Implementierung handelt, kann dort keine Obergrenze angegeben werden.

Bei der Komponente STM sind die primären Taktdomänen, deren Anzahl acht beträgt, aufgebraucht. Dadurch kann es zu Problemen beim Einsatz des internen Logikanalysators ChipScope Pro kommen. Dieser benötigt zur Steuerung der Trigger-Logik und zum Aufzeichnen des Taktes eine eigene Taktdomäne. Wird die Anzahl

der primären Taktdomänen jedoch überschritten muss der Entwickler den Entwurf manuell in Quadranten des FPGAs partitionieren, wie es bereits in Abschnitt 2.4 beschrieben wurde.

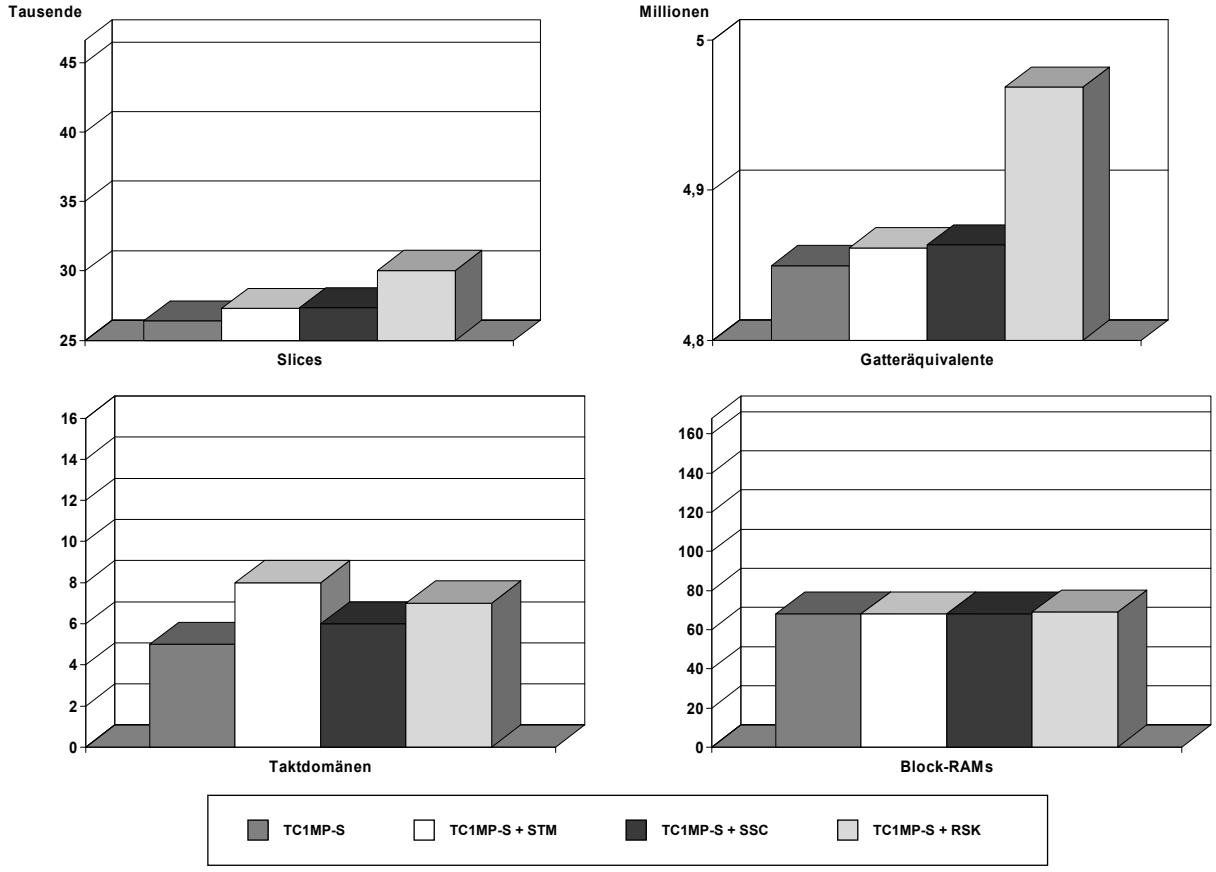


Abbildung 7.12: Auslastung des Spyder-FPGAs XC2V8000 für die Integration der drei Hardware-Komponenten zusammen mit der TC1MP-S-Plattform.

Eine manuelle Partitionierung des Entwurfs ist aber im Fall des TriCore1 schwierig, da dieser einen erheblichen Teil des zur Zeit größten verfügbaren FPGAs benötigt. In Abbildung 7.3(b) auf Seite 135 ist die Integration der TC1MP-S-Plattform dargestellt. Es ist deutlich sichtbar, dass der TC1MP-S-Kern (gelb) sich über alle vier Quadranten des FPGAs erstreckt und eine Aufteilung des Entwurfs in Quadranten deshalb eine Aufteilung des Mikrokontroller-Kerns selbst bedeuten würde.

Durch die Verwendung einer Erweiterungsplattform für das Spyder-System stehen dagegen acht weitere primäre Taktdomänen zur Verfügung. Abbildung 7.13 zeigt die Ressourcenauslastung für die drei Hardware-Komponenten auf einer Erweiterungsplattform, die mit einem Xilinx XC2V3000-FPGA ausgestattet ist. Für die Slices, Taktdomänen und Block-RAMs stellen die Obergrenzen der Diagramme die maxi-

mal verfügbaren Ressourcen dar. Für die Implementierung der Hardware-Komponente RSK sind auf der Erweiterungsplattform zwar immer noch acht Taktdomänen notwendig, da die Auslastung des FPGAs aber deutlich geringer als bei einer Ein-Board-Lösung ist, lässt sich der Entwurf hier deutlich besser auf die vier Quadranten des FPGAs aufteilen.

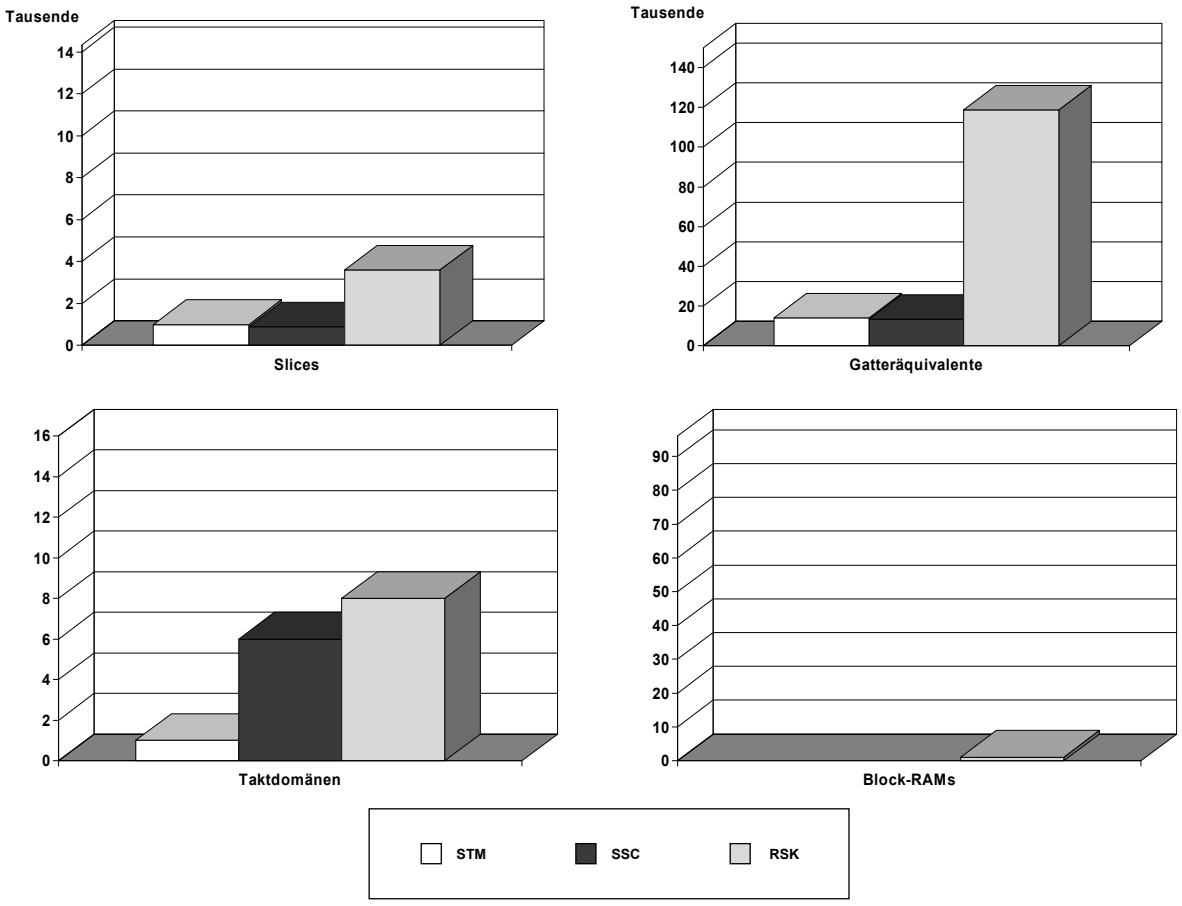


Abbildung 7.13: Auslastung des Spyder-Erweiterungsboards für die drei zusätzlichen Hardware-Komponenten.

Da es sich bei den Hardware-Komponenten in der Regel um Module handelt, die sich noch in der Entwicklung befinden, ist es darüber hinaus wichtig, dass nach Änderungen im HDL-Quelltext sehr zeitnah mit der Verifikation der geänderten Komponente fortgefahren werden kann. Dies bedeutet, dass die Zeiten für die Synthese so klein wie möglich sein müssen. Durch die Trennung der Implementierung der neuen Hardware-Komponenten von der TC1MP-S-Plattform lässt sich genau dieses Ziel erreichen.

Abbildung 7.14 stellt die Synthesezeiten für die Implementierung der Hardware-Komponenten zusammen mit der TC1MP-S-Plattform (EB) und mit Hilfe einer Er-

weiterungsplattform (EP) dar. Die Zeiten wurden auf einem Intel Pentium 4 mit 2,8 GHz und 1GB Hauptspeicher gemessen. Wie leicht zu erkennen ist, lassen sich die Hardware-Komponenten für die Erweiterungsplattformen erheblich schneller synthetisieren. So ist bei den hier behandelten Modulen eine Beschleunigung um den Faktor 42 bei SSC und Faktor 8 bei RSK möglich.

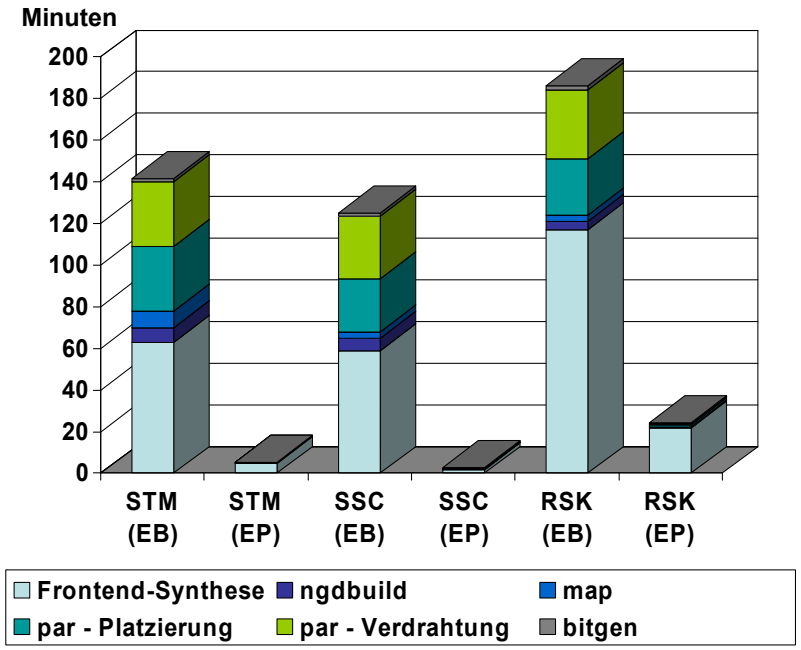


Abbildung 7.14: Vergleich der Synthesedauer zwischen Implementierungen mit (EP) und ohne (EB) Erweiterungsplattformen.

7.2.3 Debugging von Hardware und Software

Für die Hardware- und Software-Entwicklung spielt das Debugging des SoC eine wichtige Rolle. Hardware-seitig müssen Einblicke bis auf Signalebene möglich sein. Für die Software ist die Anbindung eines Software-Debuggers unumgänglich. Das Hardware-Debugging wird bei dem hier vorgestellten Konzept durch die Anbindung interner und externer Logikanalysatoren ermöglicht. Für die Integration eines kommerziellen Software-Debuggers wurden die entsprechenden Hardware-Debugging-Module der Debugging-Kette des TriCore1 auf dem Prototying-Board implementiert.

Das Debugging der Hardware und Software wird an dieser Stelle am Beispiel der SSC-Hardware-Komponente beschrieben. Abbildung 7.15 zeigt die Oberfläche eines Software-Debuggers, der mit einem TriCore1-Kern auf dem Spyder-System verbunden ist und ein Programm, welches die SSC-Schnittstelle zunächst initialisiert (`init()`) und dann eine fünf über die Datenleitung ausgibt (`transmit_int(5)`).

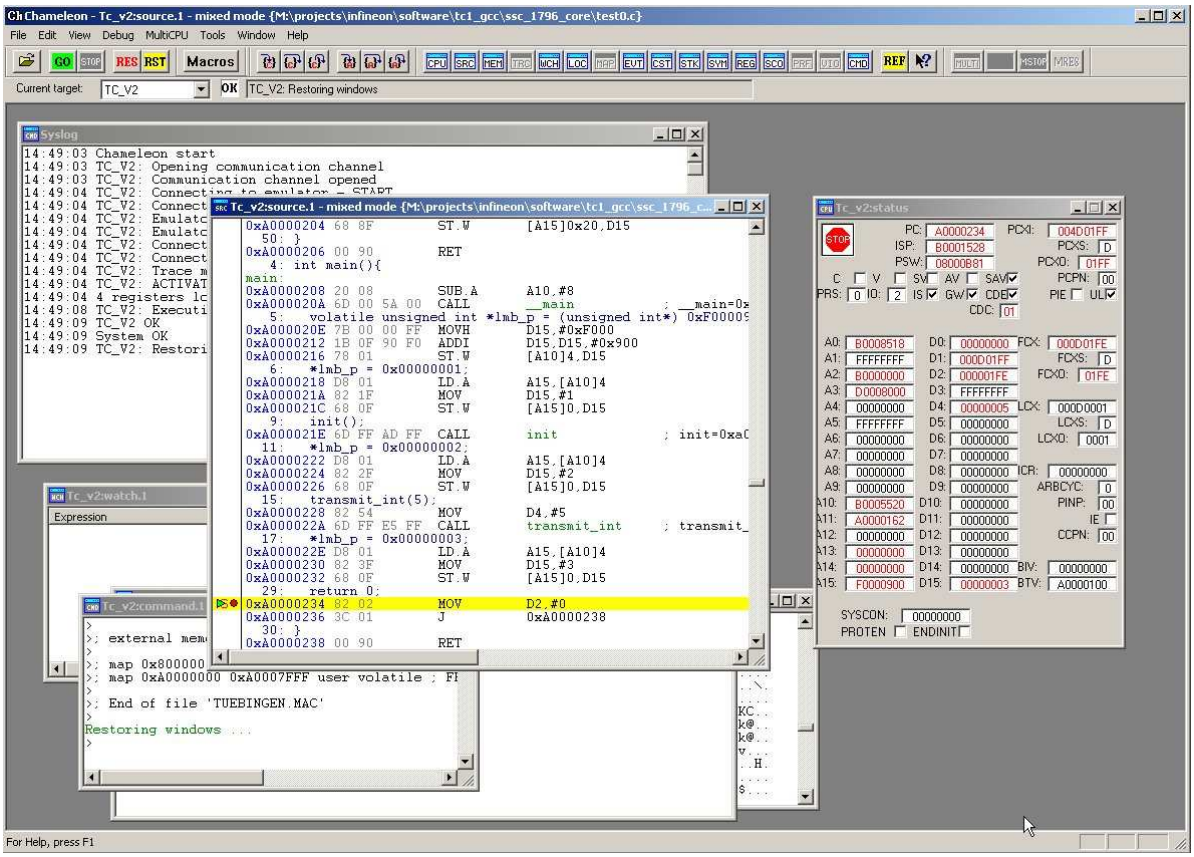


Abbildung 7.15: Ansicht eines Software-Debuggers für einen SSC-Initialisierungs- und Sendetest.

Das Programm wurde mit dem GNU C-Compiler für den TriCore1 übersetzt und dann mit dem Software-Debugger in die Spyder SRAMs geladen. In den Entwurf wurde ein Kern des internen Logikanalysators ChipScope Pro integriert und mit internen Signalen der SSC-Schnittstelle verbunden.

Abbildung 7.16 zeigt die Oberfläche der ChipScope Pro-Software nachdem die internen Signale der SSC-Schnittstelle ausgelesen wurden. Im oberen Teil des Fensters ist die Trigger-Bedingung für den Logikanalysator zu erkennen. Der Analysator triggert bei steigender (*rising (R)*) Flanke des Signals `ssc_en_o`. Im unteren Fenster sind resultierenden Signale zu sehen, die nach der Trigger-Bedingung aufgenommen wurden. Hier ist zu erkennen, dass, nachdem das Datum fünf in den Übertragungspuffer `ssctb` geschrieben wurde, der Schiebetakt `ssc_sh_clk_o` startet und die Daten auf der Signalleitung `ssc_ms_out_o` erscheinen.

Für die Integration des internen Logikanalysators ChipScope Pro stehen Programme zur Verfügung, die die Analysekerne automatisiert in die Netzliste des Entwurfs einfügen. Die Netzliste muss danach mit den Xilinx-Werkzeugen `ngdbuild`, `map`, `par` und `bitgen` neu synthetisiert werden. Auch an dieser Stelle sind die reduzier-

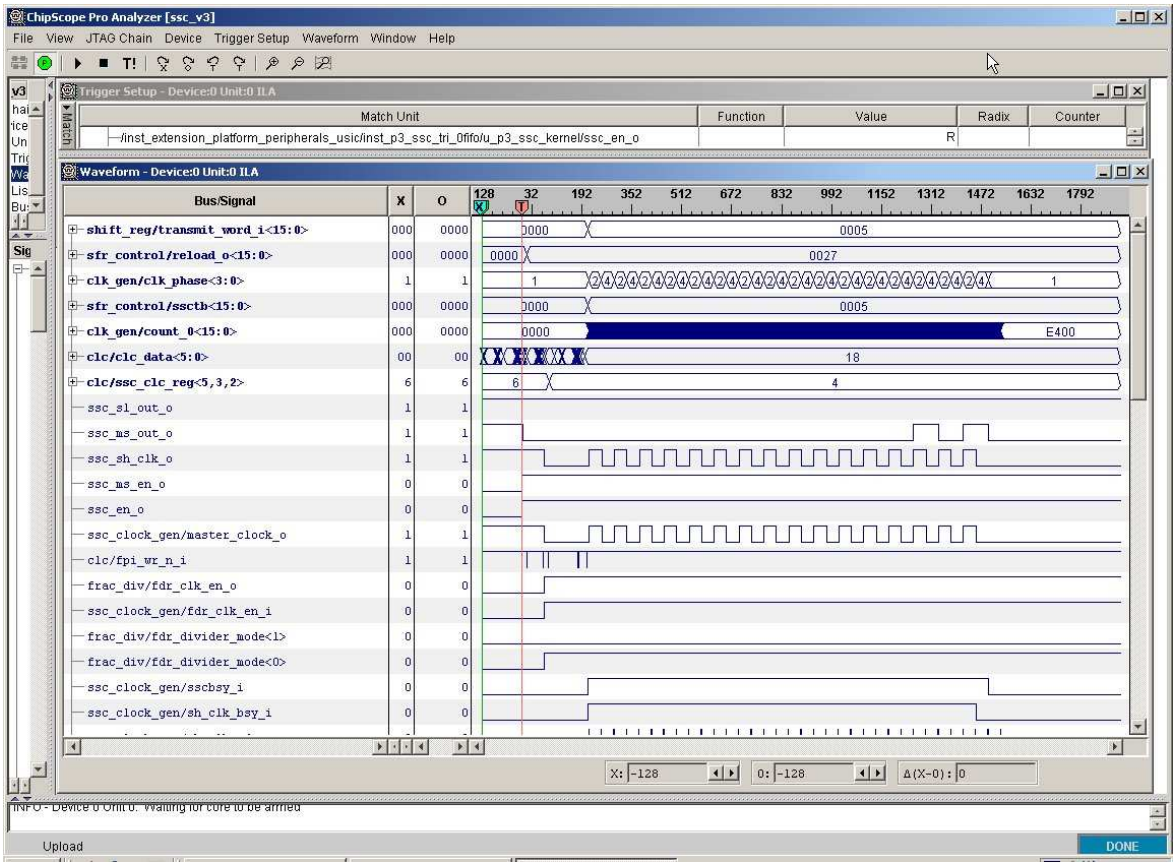


Abbildung 7.16: Interne Signale der SSC-Schnittstelle aufgezeichnet mit dem internen Logikanalysator ChipScope Pro.

ten Synthesezeiten wichtig, die mit Hilfe einer Erweiterungsplattform erzielt werden können.

Abbildung 7.17 zeigt externe Signale der Übertragungsleitungen der SSC-Schnittstelle, die auf der Backplane des Spyder-Systems mit einem externen Logikanalysator aufgenommen wurden. Diese stimmmen mit den mit ChipScope Pro aufgenommenen Signalen überein.

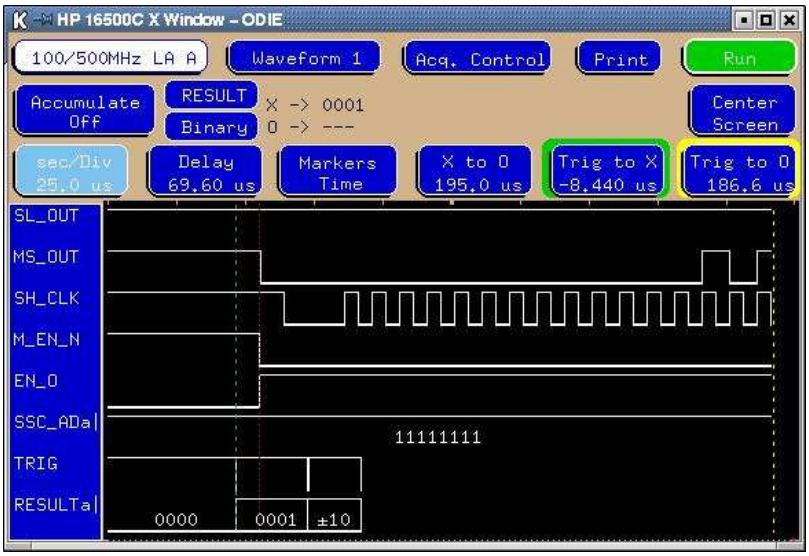


Abbildung 7.17: Ausgangssignale der SSC-Schnittstelle aufgenommen mit einem externen Logikanalysator.

7.2.4 Laufzeitmessungen

Da in dieser Arbeit die Entwicklung von Hardware-Komponenten adressiert wird, für die noch kein Chip vorliegt, gibt es – neben der Realisierung der Hardware-Komponente auf dem Spyder-System – bisher nur die Möglichkeit, die Hardware und Software für die neuen Komponenten anhand eines Simulationsmodells des TriCore1 zu entwickeln. Dementsprechend muss auch die hardware-nahe Software auf diesem Simulationsmodell ablaufen. Die in dieser Arbeit vorgestellte Entwurfsmethodik muss sich deshalb mit dieser Technologie vergleichen.

An dieser Stelle wird für die Laufzeitmessungen der Software ein HDL-Simulationsmodell verwendet, welches nach dem in Kapitel 5.3.2 beschriebenen Konzept arbeitet und für den TriCore1-Mikrokontroller sowie das Spyder-System implementiert wurde. Mit Hilfe einer HDL-Simulationsumgebung sind Einblicke bis auf Signalebene des SoC möglich. Allerdings kann das zeitliche Verhalten des Systems je nach Abstraktionsgrad der Simulation verfälscht werden. Für die Vergleichsmes-

sungen in diesem Abschnitt wurde ein Simulationsmodell des TriCore1 auf RTL-Ebene verwendet. Abbildung 7.18 zeigt eine Darstellung der Signale in einem HDL-Simulator für die oben beschriebene Ansteuerung der SSC-Schnittstelle. Auch hier sind unter anderem, wie in der Ausgabe des internen Logikanalysators ChipScope Pro aus Abbildung 7.16 die Signale `ssctb`, `ssc_sh_clk_o` und `ssc_ms_out_o` zu erkennen.

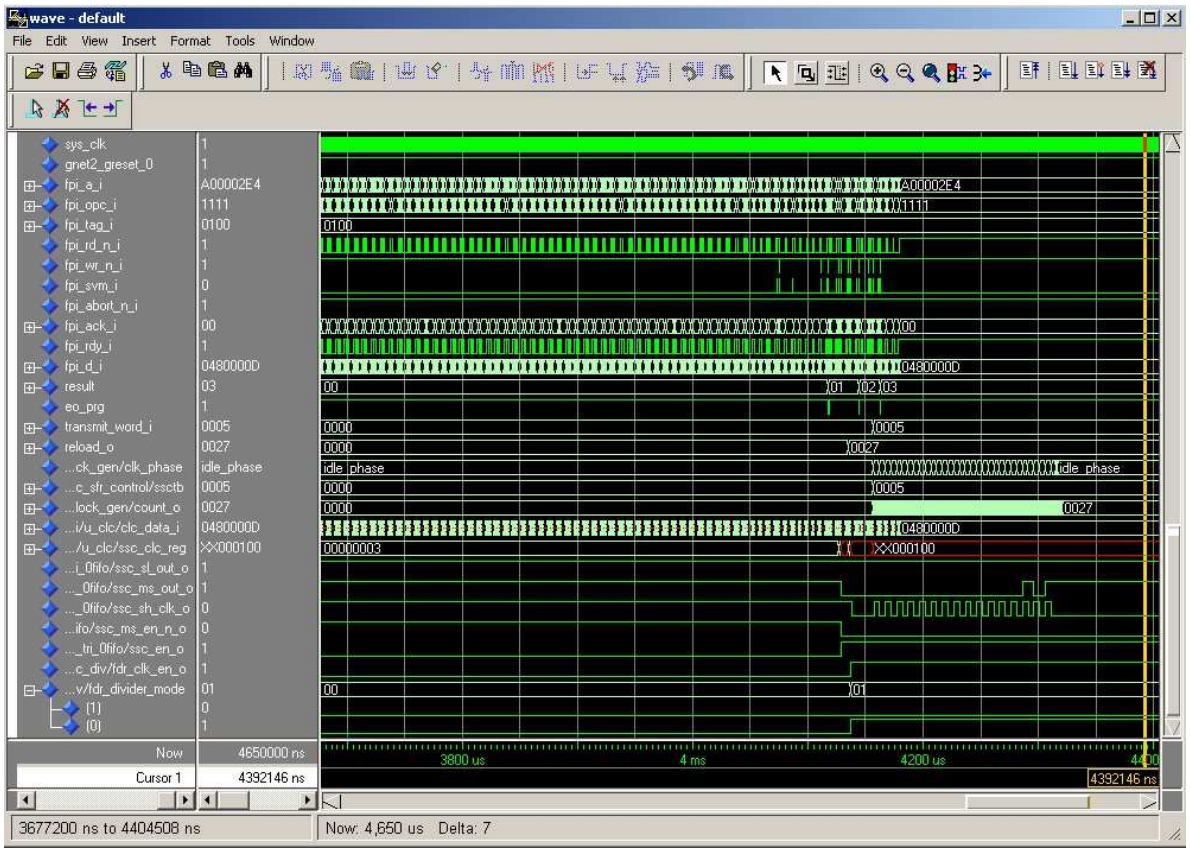


Abbildung 7.18: Software-Entwicklung mit einem HDL-Simulator.

Allerdings ist die Kombination mit einem Software-Debugger mit dieser Entwurfsmethode nicht möglich, wodurch das Debugging eingebetteter Hardware/Software-Systeme erheblich erschwert wird. Für das Software-Debugging kommt das in Kapitel 5.2.5 beschriebene Trigger-Modul zum Einsatz, welches Zeitpunkte im Entwurf markiert. Das Trigger-Modul wird im Programm durch Schreiben auf eine spezielle Adresse angesteuert. Programmabschnitt 7.1 zeigt die Ansteuerung des Trigger-Moduls für das SSC-Beispiel in den Zeilen 4, 6, und 8. Der Quelltext ist auch im Software-Debugger in Abbildung 7.15 dargestellt. In Abbildung 7.17 und 7.18 sind die Trigger-Signale als Zacken der Signale `TRIG` bzw. `eo_prg` zu erkennen. Die Programmabschnitte werden mit dem Signal `result` identifiziert.

```
1  int main(){
2     volatile unsigned int *lmb_p = (unsigned int*) 0xF0000900;
3
4     *lmb_p = 0x00000001;
5      init ();
6     *lmb_p = 0x00000002;
7      transmit_int (5);
8     *lmb_p = 0x00000003;
9
10    return 0;
11 }
```

Programm 7.1: Integration von Debugging-Marken für die HDL-Simulation.

Für die Laufzeitmessungen der Software wurden die Simulationszeiten eines HDL-Simulators auf einem Intel Pentium 4 mit 2,8 GHz und 1GB Hauptspeicher und die Emulationszeiten des SoC mit dem Prototyping-System Spyder, welches mit 8 MHz getaktet wurde, ermittelt. Die Messungen der Emulation wurden wie in Abschnitt 7.1.2 beschrieben durchgeführt.

Bevor der Entwickler mit dem eigentlichen Debugging der Anwendung beginnen kann, muss der Prozessor zunächst initialisiert werden. Dies wird in eingebetteten Systemen entweder von einer Assembler-Routine vor dem Einsprung in die Funktion `main()` oder von einem Betriebssystem durchgeführt. Für die Entwicklung mit dem HDL-Simulator heißt das, dass die Initialisierung bei jedem Neustart des Programms durchgeführt werden muss. Für die hier vorgestellten Messungen wurde kein Betriebssystem verwendet. Trotzdem dauert die Initialisierung des TriCore1 mit einem HDL-Simulator 78 Sekunden. Für den Software-Entwickler bedeutet dies, dass er mit dem eigentlichen Debugging der Anwendung durch Simulation erst nach 78 Sekunden beginnnen kann. Auf dem Spyder-System dauert die Initialisierung des Mikrokontrollers lediglich 141,9 ms.

In Tabelle 7.3 sind Laufzeitmessungen für ausgewählte Testprogramme der drei oben beschriebenen Hardware-Komponenten dargestellt. In den Laufzeiten für die Simulation und Emulation der Testprogramme sind die Zeiten für die Initialisierung des Mikrokontrollers mit enthalten, da diese bei den Testläufen immer ausgeführt werden muss.

Da die Komponente STM nicht initialisiert und konfiguriert werden muss, sondern zum Vermessen von Anwendungen verwendet wird, wurde der Algorithmus „Sieb des Eratosthenes“ für die ersten zehn Primzahlen mit dieser Komponente vermessen. Bei der SSC-Schnittstelle wurde ein Datum übertragen sowie ein Baudraten- und Stress-Test durchgeführt. Wie in Abbildung 7.11 auf Seite 150 bereits dargestellt, wurden für die RSK-Komponente die beiden Protokolle ASC und SSC getestet. Beim Baudraten-Test wird jeweils zunächst eine Baudrate eingestellt und dann ein Datum

übertragen. Beim Stress-Test werden mehrere Daten hintereinander übertragen. Bei der letzten Messung wurden beide Kanäle parallel einem Stress-Test unterzogen.

Modul	Programm	Simulation (Sek.)	Emulation (ms)	Beschleunigung (Faktor)
TC1MP-S	Initialisierung	78	141,9	550
STM	Primzahlberechnung	88	142,7	617
SSC	Datum senden	81	142,1	570
SSC	Baudraten-Test	229	152,4	1.503
SSC	Stress-Test	456	167,7	**2.719**
RSK (ASC)	Datum senden	99	142,5	695
RSK (ASC)	Schieberichtung	101	143,1	706
RSK (ASC)	Baudraten-Test	187	147,3	1.269
RSK (ASC)	Stress-Test	1.039	197,6	**5.259**
RSK (SSC)	Datum senden	93	142,5	653
RSK (SSC)	Baudraten-Test	141	144,8	974
RSK (SSC)	Stress-Test	977	196,1	**4.982**
RSK (ASC/SSC)	Stress-Test	3.752	360,5	**10.408**

Tabelle 7.3: Laufzeiten für die Entwicklung hardware-naher Software mit HDL-Simulator und FPGA-Emulation.

Die Messergebnisse aus Tabelle 7.3 zeigen, dass mit der in dieser Arbeit vorgestellten Entwurfsmethodik eine signifikante Beschleunigung der Software-Entwicklung erzielt werden kann. Insbesondere ist ersichtlich, dass bei den vier Stress-Tests für die Hardware-Komponenten SSC und RSK die Laufzeiten durchschnittlich um einen Faktor 5.800 schneller als bei der Simulation sind. Bei diesen Tests nimmt die Interaktion des Mikrokontrollers mit der Hardware-Komponente einen größeren Anteil des Programms ein. Bei den anderen Testprogrammen hat die Initialisierung des Mikrokontrollers einen Hauptanteil an der gemessenen Zeit. Aber auch dort ist noch eine durchschnittliche Beschleunigung um einen Faktor 800 zu beobachten.

7.2.5 Zusammenfassung

In diesem Abschnitt wurden Ergebnisse präsentiert, die eine Verbesserung des Entwurfsprozesses für eingebettete Hardware/Software-Systeme mit der in dieser Arbeit

vorgestellten Entwicklungsmethodik zeigen. Die Verbesserung ist sowohl qualitativ als auch quantitiv zu belegen.

Durch die parallele Entwicklung neuer Hardware-Komponenten und ihrer dazugehörigen Software für SoCs lässt sich der Entwicklungsprozess deutlich verkürzen. Dies spielt vor allem vor dem Hintergrund steigender Programmgrößen eine wichtige Rolle. Darüber hinaus wurde gezeigt, dass detaillierte Einblicke bis auf Signalebene der Hardware und Assemblerebene der Software möglich sind. Durch die frühe Evaluierung der Hardware und dazugehöriger Software können Fehler in der Hardware-Komponente und der Hardware/Software-Schnittstelle noch einfach behoben werden. Nach dem Tape-out des Chips ist dies nur durch eine neue Implementierung möglich, was Kosten in Millionenhöhe verursachen kann. In [55] werden die Kosten für die Produktion der Maske und einer Evaluierungskarte für zukünftige SoCs mit $5 Mio. beziffert.

Eine Verkürzung des Entwurfsprozesses mit der in dieser Arbeit vorgestellten FPGA-basierten Emulationsumgebung lässt sich aber auch quantitativ belegen. So konnten die Turn-around-Zeiten für die Hardware-Synthese durch die Verwendung von Erweiterungsplattformen für die untersuchten Hardware-Komponenten bis zu einem Faktor 42 beschleunigt werden. Darüber hinaus stehen dem Entwickler durch die Verwendung von Erweiterungsplattformen für die Implementierung der neuen Hardware-Komponenten zusätzliche Ressourcen zur Verfügung.

Beim Test der neuen Hardware-Komponenten und bei der Software-Entwicklung sind signifikante Geschwindigkeitsgewinne zu verzeichnen. So konnte alleine die Initialisierung des Mikrokontrollers um einen Faktor 550 gegenüber der HDL-Simulation beschleunigt werden. Dabei zeigt sich, dass die hier vorgestellte Entwicklungsmethodik vor allem für ausgiebige Stress-Tests und zum Auffinden von Fehlern geeignet ist, die erst nach geraumer Laufzeit auftreten. Beim Stress-Test der Komponente RSK für die beiden Protokolle ASC und SSC konnte ein Beschleunigungsfaktor von 10^4 erzielt werden. Gegenüber einer Software-Entwicklung mit einem TL-Modell des SoC auf Basis von SystemC ist mit der vorgestellten Emulationsumgebung bei 8MHz Taktfrequenz eine Performanzsteigerung um einen Faktor 10^2 zu erreichen [80].

8 Zusammenfassung

In dieser Arbeit wurde ein neues Konzept für die Entwicklung eingebetteter Hardware/Software-Systeme vorgestellt. Dabei wurden zwei Ziele verfolgt, die zusammen den Entwurfsprozess für SoCs verbessern. Zum Einen sollte die Entwicklung eingebetteter Software auf einem Hardware-Modell beschleunigt und zum Anderen die parallele Entwicklung von Hardware-Komponenten und hardware-naher Software für SoCs ermöglicht werden.

Aufgrund des steigenden Anteils der Software in eingebetteten Systemen, spielt deren Entwicklung eine wichtige Rolle. Um die Zeit bis zur Markteinführung kurz zu halten, muss möglichst früh mit der Software-Entwicklung begonnen werden. Bisherige Verfahren zur Entwicklung von SoCs zeichnen sich entweder durch hohe Kosten für das Entwicklungssystem, langsame Ausführungszeiten des Hardware/Software-Modells oder durch eine festgelegte Hardware-Architektur aus. Mit der vorgestellten neuen Entwurfsmethodik können die Kosten für das Entwicklungssystem dagegen um 80% reduziert, neue Hardware-Komponenten integriert und parallel dazu die Entwicklung hardware-naher Software bis zu einem Faktor 10^4 beschleunigt werden.

Die Verbesserung des Entwurfsprozesses wird durch die Verwendung eines FPGA-basierten Prototyping-Systems erreicht, auf dem die Hardware des SoC emuliert wird. Dadurch ist es möglich, eine vorgegebene SoC-Plattform um neu zu entwickelnde Hardware-Komponenten zu erweitern. Da aus Kostengründen auf die Verwendung von Hardware-Emulations-Systemen verzichtet wird, treten Probleme bei der Integration der Hardware des SoC auf dem Emulations-Board auf. Neben der Problematik des ASIC-IP-Prototypings auf FPGAs wurde das Problem der beschränkten Ressourcen der Prototyping-Umgebung identifiziert. Als Lösung wurde das Konzept der Partialemulation vorgestellt, bei dem die Ressourcen des *gesamten* Prototyping-Systems für die Integration des SoC ausgenutzt werden. Durch die Partialemulation konnten so ca. 30% der Ressourcen des Prototyping-Systems eingespart werden, wodurch das Prototyping noch größerer SoCs möglich ist.

Die Relevanz der in dieser Arbeit vorgestellten Konzepte wurde anhand eines komplexen Beispiels aus der Praxis aufgezeigt. Es wurde der TriCore1-Mikrokontroller der Firma Infineon, der Bestandteil der DesignWare-StarIP-Bibliothek von Synopsys ist und somit als typischer Vertreter eines Mikrokontroller-IP-Kerns für SoCs angesehen werden kann, auf dem kostengünstigen Rapid-Prototyping-System Spyder integriert. Durch die Anbindung einer Software-Entwicklungsumgebung wurde

das erste Ziel dieser Arbeit, nämlich die Beschleunigung der Software-Entwicklung auf einem architekturgenauen Modell der SoC-Hardware erreicht. Wie anhand von EEMBC-Benchmarks gezeigt wurde, lässt sich die Performanz gegenüber einem BSS im Schnitt um einen Faktor 50 steigern. Vergleiche mit einem Evaluierungsboard für den TriCore1 haben dabei gezeigt, dass das zeitliche Verhalten des Prozessors auf dem Spyder-System dem des TriCore1-Chip gleichzusetzen ist.

In einem zweiten Schritt wurde die neue Entwurfsmethodik für eingebettete Software ausgebaut, um die parallele Entwicklung neuer Hardware-Komponenten und deren hardware-naher Software gegenüber herkömmlichen Methoden zu verbessern. Eine Neuerung des eingeführten Konzepts ist die Verwendung kostengünstiger Erweiterungsplattformen für die Implementierung und Evaluierung der neu zu entwickelnden Hardware-Komponenten. Die Verwendung von Erweiterungsplattformen hat gegenüber der Implementierung des SoC auf einem Chip einige Vorteile. Zum Einen stehen dem Entwickler durch die Hinzunahme eines weiteren Boards mehr Ressourcen für die Integration seiner neuen Hardware-Komponente zur Verfügung. Zum Anderen ist die IP des Mikrokontroller-Herstellers geschützt, da diese auf der Hostplattform gekapselt ist. Dabei können im Gegensatz zu bisher verfügbaren Evaluierungsboards unterschiedliche Architekturvarianten des Mikrokontrollers evaluiert werden. Dies zeigt sich dadurch, dass mit dem Spyder-System unterschiedliche Speicherhierarchien und Cachegrößen für den TriCore1 durchgespielt werden können. Dies ist beispielsweise mit dem TriBoard TC1920A, bei dem der TriCore1 als fertiger Chip vorliegt, nicht möglich. Schlussendlich lassen sich die Turn-around-Zeiten beim Debugging der neuen Hardware-Komponenten, die noch Entwurfsfehler enthalten können und die beseitigt werden müssen, bis zu einem Faktor 40 reduzieren.

Die Effizienz des vorgestellten Konzepts wurde anhand von drei Hardware-Komponenten gezeigt, die mit einer TriCore1-Plattform kombiniert und für die Test-Treiber entwickelt wurden. Diese konnten bis auf Signalebene der Hardware und Assemblerebene der Software untersucht werden. Bei Stress-Tests der Hardware-Komponenten ergab sich dabei eine Performanzsteigerung gegenüber einer HDL-Simulation des Gesamtsystems bis zu einem Faktor 10^4.

A Abkürzungen

ADC	Analog Digital Converter
API	Application Programming Interface
ASC	Asynchronous Serial Control
ASIC	Application Specific Integrated Circuit
BSS	Befehlssatzsimulator
CAN	Controller Area Network
CLB	Configurable Logic Block
CPS	CPU Slave
CPU	Central Processing Unit
DCM	Digital Clock Manager
DDR	Double Data Rate
DHW	Dedizierte Hardware
DMI	Data Memory Interface
DRAM	Dynamic Random Access Memory
EAM	Ein-/Ausgabe Modul
EEMBC	Embedded Microprocessor Benchmark Consortiums
FPGA	Field Programmable Gate Array
FPI	Flexible Peripheral Interface
FPU	Floating Point Unit
HDL	Hardware Description Language
ICU	Interrupt Control Unit
IIC	Inter IC Bus
IIS	Infotainment Audio Bus
IP	Intelectual Property
IrDA	Infrared Data Association
JTAG	Joint Test Action Group
JVM	Java Virtual Machine
LFI	LMB to FPI Interface
LIN	Local Interconnect Network
LMB	Local Memory Bus
LMBh	Local Memory Bus Hub
LOC	Lines of code
LUT	Lookup Table

MMU	Memory Management Unit
PCXI	Previous Context Information
PMI	Programme Memory Interface
PSW	Programme Status Word
RAM	Random Access Memory
ROM	Read Only Memory
RSK	Rekonfigurierbarer Serieller Kommunikationskanal
RTL	Register Transfer Level
RTOS	Real Time Operating System
SCU	System Control Unit
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System-on-a-Chip
SPI	Serial Parallel Interface
SPR	Scratchpad RAM
SRAM	Static Random Access Memory
SRN	Service Request Node
SSC	Synchronous Serial Interface
SSRAM	Synchronous Static Random Access Memory
STM	System Timer Unit
TAG-RAM	Cacheverwaltung
TC1MP-S	TriCore1 Microprocessor Synthesizable
TLM	Transaction Level Modeling
UART	Universal Asynchronous Receiver Transmitter
UCF	User Constraint File
USB	Universal Serial Bus
VHDL	Very-high-speed-integrated-circuits Hardware Description Language
VLIW	Very long instruction word architecture

Literaturverzeichnis

[1] AGILENT TECHNOLOGIES: *Mictor probes*. Online im Internet: http://we.home.agilent.com/USeng/nav/-536893935.0/pc.html [Stand Mai 2005].

[2] AICAS REALTIME: *JamaicaVM*. Online im Internet: http://www.aicas.com [Stand Mai 2005].

[3] ALFKE, PETER: *Die FPGA-Evolution: Die Zukunft gehört anwenderprogrammierbaren Logik-Chips*. Elektronik Praxis, (1):67–71, Januar 2002.

[4] ALTERA: *Excalibur devices*. Online im Internet: http://www.altera.com/products/devices/arm/arm-index.html [Stand Mai 2005].

[5] APTIX: *System Explorer™*. Online im Internet: http://www.aptix.com [Stand Mai 2005].

[6] ARM: *Versatile platform baseboard for ARM926EJ-S*. Online im Internet: http://www.arm.com/products/DevTools/VPB926EJ-S.html [Stand Mai 2005].

[7] ASHENDEN, PETER J.: *The designer's guide to VHDL*. The Morgan Kaufmann series in systems on silicon. Morgan Kaufmann, San Francisco, 2. Auflage, 2002.

[8] ATMEL: *AVR STK500 user guide*. Atmel Corporation, 2000.

[9] BABB, JONATHAN, RUSSELL TESSIER, MATTHEW DAHL, SILVINA HANONO, DAVID HOKI und ANANT AGARWAL: *Logic emulation with virtual wires*. IEEE Transactions on CAD, 16(6):609–626, Juni 1997.

[10] BALZERT, HELMUT: *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, 1998.

[11] BARNHOLT, EDWARD W. (NED): *FPGA-zentrierte Verifikationsstrategie*. Mündlich, Euro DesignCon, München 2004.

[12] BLACKMORE, TIM: *Complete verification of TriCore2 by functional simulation.* Mündlich, Euro DesignCon, München 2004.

[13] BOEHM, BARRY: *Software engineering.* IEEE Transactions on Computers, 25:1226–1241, 1976.

[14] BOEHM, BARRY: *A spiral model of software development and enhancement.* IEEE Software Engineering Project Management, Seiten 513–527, 1987.

[15] BOLLELLA, GREG, JAMES GOSLING und BENJAMIN M. BROSGOL: *Real-time specification for Java*. Addison-Wesley Longman, Amsterdam, 2000.

[16] BOOCH, GRADY, JAMES RUMBAUGH und IVAR JACOBSON: *The unified modeling language user guide.* Addison-Wesley Object Technology Series, 1999.

[17] BRÖHL, ADOLF-PETER und WOLFGANG DRÖSCHEL: *Das V-Modell: Der Standard für die Software-Entwicklung.* Oldenburg-Verlag, 1993.

[18] BRÜGMANN, JOHANNES: *Entwurf und Implementierung einer Kommunikationskomponente für prozessorbasierte Systems-on-Chip*. Studienarbeit, Universität Tübingen, Technische Informatik, 2004.

[19] BYOUS, JOHN: *Java technology: the early years.* Online im Internet: http://java.sun.com/features/1998/05/birthday.html [Stand Mai 2005].

[20] CADENCE: *Cadence incisive verification platform.* Online im Internet: http://www.quickturn.com [Stand Mai 2005].

[21] CADENCE: *Quickturn ships record number of emulation gates worldwide with the MercuryPlus in-circuit emulation system, featuring custom FPGAs.* Press release, February 2001. Online im Internet: http://www.cadence.com/company/newsroom/press_releases/pr.aspx?xml=021901_QT [Stand Mai 2005].

[22] CHANG, HENRY, LARRY COOKE, MERRILL HUNT, GRANT MARTIN, ANDREW MCNELLY und LEE TODD: *Surviving the SoC revolution – a guide to platform-based design.* Kluwer Academic Publishers, 1999.

[23] CIFUENTES, CRISTINA und VISHY MALHOTRA: *Binary translation: static, dynamic, retargetable?* In: *Proceedings of the International Conference on Software Maintenance (ICSM)*, Seiten 340–349, 1996.

[24] CLARKE, PETER: *European Space Agency launches free Sparc-like core.* EE Times, März 2000.

[25] CLAUS, VOLKER und ANDREAS SCHWILL: *Duden Informatik.* Dudenverlag, 1993.

[26] DEY, SUJIT, DEBASHIS PANIGRAHI, LI CHEN, CLARK N. TAYLOR, KRISHNA SEKAR und PABLO SANCHEZ: *Using a soft core in a SoC design: experiences with picoJava.* IEEE Design and Test of Computers, Seiten 60–71, Juli-September 2000.

[27] DONLIN, ADAM, AXEL BRAUN und ADAM ROSE: *SystemC for the design and modeling of programmable systems.* In: BECKER, JÜRGEN, MARCO PLATZNER und SERGE VERNALDE (Herausgeber): *Field Programmable Logic and Application: 14th International Conference, FPL*, Seiten 811–820, Leuven, August 2004.

[28] DOUGLASS, BRUCE P.: *Real-time UML second edition – Developing efficient objects for embedded systems.* Addison-Wesley Object Technology Series, 2000.

[29] DREHER, WERNER, HANS-GEORG MARTIN und WOLFGANG ROSENSTIEL: *Das Weaver-II-Board als neue HW/SW-Codesign Plattform*. In: *2nd Workshop on System Design Automation (SDA'99)*, Rathen, April 1999.

[30] EDWARDS, KEITH W.: *core JINI*. The Sun Microsystems Press Java Series, 1999.

[31] EEMBC: *Embedded Microprocessor Benchmark Consortium*. Online im Internet: http://www.eembc.org [Stand Mai 2005].

[32] FLEXRAY™: *The communication system for advanced automotive control applications.* Online im Internet: http://www.flexray.com [Stand Mai 2005].

[33] GAJSKI, DANIEL D., RAINER DÖMER und JIANWEN ZHU: *IP-centric methodology and design with the SpecC language*. In: AHMED A. JERRAYA, JEAN P. MERMET (Herausgeber): *Proceedings of the NATO ASI on System Level Synthesis for Electronic Design*, Lucca, August 1998.

[34] GAJSKI, DANIEL D., NIKIL D. DUTT, ALLEN WU und STEVE LIN: *High-level-synthesis*. Kluwer Academic Puplishers, 1992.

[35] GAJSKI, DANIEL D., JIANWEN ZHU, RAINER DÖMER, ANDREAS GERSTLAUER und SHUQING ZHAO: *SpecC: Specification language and methodology.* Kluwer Academic Publishers, 2000.

[36] GOERING, RICHARD: *Synopsys, Quickturn team to link co-verification, emulation.* EE Times, Juni 1998.

[37] GRÖTKER, THORSTEN: *Modeling software with SystemC 3.0.* ESCUG-Treffen, Stresa, Italien, 2002.

[38] GRÖTKER, THORSTEN, STAN LIAO, GRANT MARTIN und STUART SWAN: *System design with SystemC.* Kluwer Academic Puplishers, 2002.

[39] GUPTA, AARTI: *Formal hardware verification methods: a survey*. Formal Methods in System Design, 1(2/3):151–238, 1992.

[40] HAQUE, FAISAL I., KHIZAR A. KHAN und JONATHAN MICHELSON: *The art of verification with VERA.* Verification Central, 2001.

[41] HAUCK, SCOTT: *The roles of FPGAs in reprogrammable systems*. Proceedings of the IEEE, 86(4):615–639, April 1998.

[42] HAUG, GUNTER: *Emulation synthetisierter Verhaltensbeschreibungen mit VLIW-Prozessoren.* Logos Verlag, Berlin, 2000.

[43] HEISE ONLINE: *Handy-Sparte von Siemens bleibt im Minus.* Online im Internet: http://www.heise.de/newsticker/meldung/53215 [Stand Mai 2005].

[44] IBM: *Evaluation kits for IBM PowerPC® processors.* Online im Internet: http://www-3.ibm.com/chips/products/powerpc/tools/600_ekit.html [Stand Mai 2005].

[45] IBM: *IBM workplace client technology, Micro Edition*. Online im Internet: http://www-306.ibm.com/software/wireless/wctme_fam/ [Stand Januar 2005].

[46] INFINEON TECHNOLOGIES AG: *IPEVAL2 prototyping platform.* Online im Internet: http://www.infineon.com/cgi/ecrm.dll/ecrm/scripts/sol_ov.jsp?oid=49449 [Stand Mai 2005].

[47] INFINEON TECHNOLOGIES AG: *TC1130 Starter Kit.* Online im Internet: http://www.infineon.com/cgi/ecrm.dll/ecrm/scripts/

`search/prom_page_dispatcher.jsp?oid=53393&search_string=TriBoard` [Stand Mai 2005].

[48] INFINEON TECHNOLOGIES AG: *TriBoard TC1920A hardware manual*. Infineon Technologies AG, München, Dezember 2001.

[49] INFINEON TECHNOLOGIES AG: *OCDS-JTAG software API guide (TriCore and C166) user guide*. Infineon Technologies AG, München, November 2002.

[50] INFINEON TECHNOLOGIES AG: *TriCore Instruction Set Simulator. TriCore 32-bit unified processor user guide v2.0.1*. Infineon Technologies AG, München, April 2002.

[51] INFINEON TECHNOLOGIES AG: *TriCore® 1.3 32-Bit unified processor core. Architecture overview handbook, V1.3.3*. Infineon Technologies AG, München, Juni 2002.

[52] INFINEON TECHNOLOGIES AG: *TC1MP-S soft macro user's manual*. Infineon Technologies AG, München, April 2003.

[53] JERRAYA, AHMED AMINE und SUNGJOO YOO: *Embedded software for SoC*. Kluwer Academic Publishers, 2003.

[54] KEATING, MICHAEL und PIERRE BRICAUD: *Reuse methodology manual – For System-on-a-Chip designs*. Kluwer Academic Publishers, 3. Auflage, 2002.

[55] KEUTZER, KURT, SHARAD MALIK, A. RICHARD NEWTON, JAN M. RABAEY und ALBERTO SANGIOVANNI-VINCENTELLI: *System-level design: Orthogonalization of concerns and platform-based design*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19(12):1523–1543, Dezember 2000.

[56] KHALILIAN, KAMBIZ, STEPHEN BRAIN, RICHARD TUCK und GLENN FARRALL: *Reusable verification infrastructure for a processor platform to deliver fast SoC development*. In: *Proceedings of the International Workshop on IP-Based Systems-on-Chip Design (IFIP 2002)*, Seiten 253–258, Grenoble, Oktober 2002.

[57] KOCH, ANDREAS: *A comprehensive prototyping platform for Hardware-Software Codesign.* In: *11th IEEE International Workshop on Rapid System Prototyping (RSP'2000)*, Seiten 78–83, Paris, Juni 2000.

[58] KOCH, GERNOT: *Interaktives Debugging algorithmischer Hardware-Verhaltensbeschreibungen mit Emulation.* Shaker-Verlag, Aachen, 1998.

[59] KROPF, THOMAS: *Introduction to formal hardware verification*. Springer-Verlag, 1999.

[60] KRUPNOVA, HELENA: *Mapping multi-million gate SoCs on FPGAs: industrial methodology and experience*. In: GIELEN, GEORGES und JOAN FIGUERAS (Herausgeber): *Design Automation and Test in Europe (DATE)*, Seiten 1236–1241, Paris, Februar 2004.

[61] KUHN, TOMMY: *Objektorientierter Hardware-Entwurf – Beschreibung, Simulation und Synthese.* Logos Verlag, Berlin, 2000.

[62] KUHN, TOMMY, TOBIAS OPPOLD, MARKUS WINTERHOLER, WOLFGANG ROSENSTIEL, MARC EDWARDS und YARON KASHAI: *A framework for object oriented hardware specification, verification, and synthesis.* In: *38th Design Automation Conference (DAC)*, Las Vegas, Juni 2001.

[63] KWON, YOUNG-SU, WOO-SEUNG YANG und CHONG-MIN KYUNG: *Signal scheduling driven partitioning for multiple FPGAs with time-multiplexed interconnection.* In: *Proceedings of the IFIP International Conference on Very Large Scale Integration (IFIP VLSI-SoC 2003)*, Darmstadt, Dezember 2003.

[64] LANGEN, DOMINIK, JÖRG-CHRISTIAN NIEMANN, MARIO PORRMANN, HEIKO KALTE und ULRICH RÜCKERT: *Implementation of a RISC processor core for SoC designs - FPGA prototype vs. ASIC implementation*. In: *Proceedings of the IEEE-Workshop: Heterogeneous reconfigurable Systems-on-Chip (SoC)*, Hamburg, 2002.

[65] LARA, GREG: *Virtex-4: Breakthrough performance at the lowest cost.* Xcell Journal, (51):33–37, Winter 2004.

[66] MARWEDEL, PETER: *Embedded System Design.* Kluwer Academic Publishers, September 2003.

[67] MARZOUGUI, MEHREZ, ADEL BAGANNE, MOHAMED ABID und RACHED TOURKI: *Co-simulation and communication synthesis for intellectual properties IPs based SoCs: approach and experimentation.* In: *Proceedings of the International Workshop on IP-Based Systems-on-Chip Design (IFIP 2002)*, Seiten 227–232, Grenoble, Oktober 2002.

[68] MENTOR GRAPHICS: *ModelSim*. Online im Internet: http://www.model.com [Stand Mai 2005].

[69] MENTOR GRAPHICS: *Seamless for hardware/software co-verification*. Online im Internet: http://www.mentor.com/seamless/ [Stand Mai 2005].

[70] MICROELECTRONICS DEVELOPMENT FOR EUROPEAN APPLICATIONS (MEDEA): *MEDEA Design Automation Roadmap*, 2003. Online im Internet: http://www.medea.org [Stand Mai 2005].

[71] MOUSSA, IMED, THIERRY GRELLIER und GIANG NGUYEN: *Exploring SW performance using SoC transaction-level modeling.* In: JERRAYA, AHMED AMINE, SUNGJOO YOO, DIEDEIK VERKEST und NORBERT WEHN (Herausgeber): *Embedded software for SoC*, Seiten 97–109, Niederlande, Februar 2003.

[72] NITSCH, CARSTEN, KARLHEINZ WEISS, THORSTEN STECKSTOR und WOLFGANG ROSENSTIEL: *Embedded system architecture design based on real-time emulation*. In: *11th IEEE International Workshop on Rapid System Prototyping (RSP'2000)*, Seiten 228–233, Paris, Juni 2000.

[73] OPENCORES.ORG: *Free open source IP cores and chip design.* Online im Internet: http://www.opencores.org [Stand Mai 2005].

[74] PRODESIGN CAD: *ASIC & SoC verification platforms.* Online im Internet: http://www.uchipit.com/ce/index.htm [Stand Mai 2005].

[75] RADETZKI, MARTIN, WOLFRAM PUTZKE-RÖMING und WOLFGANG NEBEL: *Objective VHDL: The object-oriented approach to hardware reuse.* In: *Advances in information technologies: the business challenge*, 1997.

[76] RECHENBERG, PETER und GUSTAV POMBERGER: *Informatik-Handbuch*. Hanser, 3. Auflage, 2002.

[77] ROSENSTIEL, WOLFGANG und RAUL CAMPOSANO: *Rechnergestützter Entwurf hochintegrierter MOS-Schaltungen.* Springer-Verlag, 1989.

[78] SANGIOVANNI-VINCENTELLI, ALBERTO und GRANT MARTIN: *Platform-based design and software design methodology for embedded sytems*. IEEE Design and Test of Computers, Seiten 23–33, November-Dezember 2001.

[79] SARMADI, SIAVASH BAYAT, SEYED GHASSEM MIREMADI, GHAZANFAR ASADI und ALI REZA EJLALI: *Fast prototyping with Co-operation of simulation and emulation.* In: GLESNER, MANFRED, PETER ZIPF und MICHEL RENOVELL (Herausgeber): *12th International Conference on Field-Programmable Logic and Applications (FPL 2002)*, Seiten 15–25, Montpellier, September 2002.

[80] SCHEPENS, COR: *Fast cycle-accurate SystemC SoC simulations.* ESCUG-Treffen, Euro DesignCon, München, 2004.

[81] SCHMITT, STEPHEN und WOLFGANG ROSENSTIEL: *Der Einsatz von Jini für die Realisierung durchgängiger Steuerungskonzepte in verteilten eingebetteten Systemen*. In: CAP, CLEMENS H. (Herausgeber): *Java-Informations-Tage (JIT'99)*, Seiten 223–232, Düsseldorf, September 1999.

[82] SCHMITT, STEPHEN und WOLFGANG ROSENSTIEL: *Realisierungsmöglichkeiten eingebetteter Systeme im Internet*. In: *9. E.I.S.-Workshop Entwurf Integrierter Schaltungen*, Seiten 85–91, Darmstadt, September 1999.

[83] SCHMITT, STEPHEN und WOLFGANG ROSENSTIEL: *Untersuchungen zur Implementierung von Java-Laufzeitumgebungen für eingebettete Systeme*. In: NET.OBJECTDAYS-FORUM (Herausgeber): *Tagungsband NET.OBJECT DAYS (NODE) 2000*, Seiten 171–180, Erfurt, Oktober 2000.

[84] SCHMITT, STEPHEN und WOLFGANG ROSENSTIEL: *Java in eingebetteten Systemen*. it+ti Informationstechnik und Technische Informatik, 43(3):142–150, Juni 2001.

[85] SCHMITT, STEPHEN und WOLFGANG ROSENSTIEL: *Konzeption einer adaptiven und dynamisch konfigurierbaren Laufzeitumgebung für echtzeitkritische eingebettete Systeme*. In: BODE, ARNDT und WOLFGANG KARL (Herausgeber): *Arbeitsplatzcomputer (APC) 2001 „Pervasive Ubiquitous Computing"*, Seiten 53–61, München, Oktober 2001.

[86] SCHMITT, STEPHEN und WOLFGANG ROSENSTIEL: *Fast and efficient development of hardware-dependent software for SoC designs*. In: SULLIVAN, BARRY (Herausgeber): *Euro DesignCon 2004*, München, Oktober 2004.

[87] SCHMITT, STEPHEN und WOLFGANG ROSENSTIEL: *Rapid prototyping of a microcontroller IP core under resource limitations*. In: STOFFEL, DOMINIK und WOLFGANG KUNZ (Herausgeber): *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, Seiten 214–223, Kaiserslautern, Februar 2004.

[88] SCHMITT, STEPHEN und WOLFGANG ROSENSTIEL: *Verification of a microcontroller IP core for System-on-Cip designs using low-cost prototyping environments*. In: LINDWER, MENNO, VASSILIOS GEROUSIS und JOAN FIGUERAS (Herausgeber): *Design, Automation and Test in Europe, Designers Forum, 2004*, Seiten 96–101, Paris, Februar 2004.

[89] SCHNERR, JÜRGEN, OLIVER BRINGMANN und WOLFGANG ROSENSTIEL: *Cycle accurate binary translation for simulation acceleration in rapid prototyping of SoCs.* In: WEHN, NORBERT und LUCA BENINI (Herausgeber): *Design, Automation and Test in Europe (DATE) 2005*, Seiten 792–797, München, März 2005.

[90] SCHNERR, JÜRGEN, GUNTER HAUG und WOLFGANG ROSENSTIEL: *Instruction set emulation for rapid prototyping of SoCs*. In: WEHN, NORBERT und DIEDERICK VERKEST (Herausgeber): *Design, Automation and Test in Europe (DATE) 2003*, Seiten 562–567, München, März 2003.

[91] SCHUBERT, THORSTEN, JÜRGEN HANISCH, JOACHIM GERLACH, JENS-E. APPELL und WOLFGANG NEBEL: *Evaluation of a refinement-driven SystemC™ -based design flow*. In: LINDWER, MENNO, VASSILIOS GEROUSIS und JOAN FIGUERAS (Herausgeber): *Design, Automation and Test in Europe, Designers Forum, 2004*, Seiten 262–267, Paris, Februar 2004.

[92] SEMICONDUCTOR INDUSTRY ASSOCIATION (SIA): *SIA roadmap for ASIC gate development*. Online im Internet: http://www.semichips.org/ [Stand Dezember 2001].

[93] SEMICONDUCTOR INDUSTRY ASSOCIATION(SIA): *International Technology Roadmap for Semiconductors (ITRS)*, 2001. Online im Internet: http://public.itrs.net [Stand Mai 2005].

[94] SPEAC: *Specification and algorithm/architecture-Co-Design for highly complex applications in automotive and communication*. Online im Internet: http://speac.fzi.de [Stand Mai 2005].

[95] STELZER, GERHARD: *Eine Debug-Sprache für alle Mikroprozessoren*. Elektronik Automotive, (1):63–64, Januar 2005.

[96] SUTHERLAND, STUART, SIMON DAVIDMANN und PETER FLAKE: *SystemVerilog for design - A guide to using SystemVerilog for hardware design and modeling*. Kluwer Academic Puplishers, 2004.

[97] SYNOPSYS: *DesignWare®Star IP Library*. Online im Internet: http://www.synopsys.com/products/designware/star_ip.html [Stand Mai 2005].

[98] SYNPLICITY: *Certify ASIC prototyping solution*. Online im Internet: http://www.synplicity.com/products/certify/index.html [Stand Mai 2005].

[99] TURNER, RAY: *System-level verification – A comparison of approaches*. In: *10th IEEE International Workshop on Rapid System Prototyping (RSP'99)*, Seiten 154–159, Clearwater, Juni 1999.

[100] VAANDRAGER, FRITS W.: *Lectures on embedded systems.* Nummer 1494 in *Lecture Notes in Computer Science*. Springer, 1998.

[101] VERISITY: *eCelerator: testbench acceleration*. Online im Internet: http://www.verisity.com/products/ecelerator.html [Stand Mai 2005].

[102] VERISOFT: *Verisoft at Munich*. Online im Internet: http://isabelle.in.tum.de/verisoft/ [Stand Mai 2005].

[103] VERSCHIEDENE: *SysML: System modeling language.* Online im Internet: http://www.sysml.org [Stand Mai 2005].

[104] WANG, YUHUI: *Eine Verifikationsmethode vom Softwaremodell bis zur FPGA-basierten Emulationsumgebung*. Studienarbeit, Universität Tübingen, Technische Informatik, Dezember 2003.

[105] WEISS, KARLHEINZ: *Architekturentwurf und Emulation eingebetteter Systeme*. Dissertation, Universität Tübingen, 1999.

[106] WEISS, KARLHEINZ: *Rekonfigurierbare Rechnerplattform „Car Infotainment"*. In: *Embedded World Conference 2003*, Nürnberg, Februar 2003.

[107] WOLBANK, THOMAS, JOHANNES REISINGER, PETER MACHEINER, JÜRGEN MACHL und MARKUS PLEIL: *Antriebssteuerung per FlexRay*. Elektronik Automotive, (6):34–37, Juni 2004.

[108] WOLF, WAYNE: *Computers as components – Principles of embedded computing design*. Morgan Kaufmann, 2001.

[109] X2E SPYDER SYSTEM DESIGN GROUP: *Spyder-System*. Online im Internet: http://www.x2e.de [Stand Mai 2005].

[110] XILINX: *ChipScope Pro*. Online im Internet: http://www.xilinx.com/ [Stand Mai 2005].

[111] XILINX: *MicroBlaze™ 32-bit soft processor core.* Online im Internet: http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?sSecondaryNavPick=Design+Tools&key=micro_blaze [Stand Mai 2005].

[112] XILINX: *PicoBlaze fully embedded 8-bit microcontroller.* Online im Internet: http://www.xilinx.com/products/design_resources/proc_central/grouping/picoblaze.htm [Stand Mai 2005].

[113] XILINX: *Virtex-II platform FPGA handbook*. Xilinx, 2000.

[114] XILINX: *ChipScope Pro software and cores user manual.* Xilinx, 2004.