Technische Universität Chemnitz
Straße der Nationen 62
09111 Chemnitz

Automatische Adaption von Hardware-Acceleratoren für Verhaltenssimulation

Diplomarbeit

Fakultät für Informatik
Studiengang Informatik
Professur Technische Informatik
Vertiefungsrichtung Eingebettete Systeme

von
Marcel Flade
Dorfstraße 81
09526 Pfaffroda-Hallbach

vorgelegt bei
Prof. Dr. habil. Wolfram Hardt

im
September 2004

Dank und Erklärung

Diese Arbeit entstand an der Professur Technische Informatik der Technischen Universität Chemnitz. Ich möchte hiermit besonders Herrn Prof. Dr. habil. Wolfram Hardt für das interessante Thema und die engagierte Betreuung während der Anfertigung dieser Arbeit danken. Auch für die Bereitstellung von Rechnern, sowie Hard- und Software durch die Professur und die Technische Universität Chemnitz spreche ich an dieser Stelle meinen Dank aus. Ebenfalls sei mir gestattet, den Mitarbeitern der Professur, speziell Markus Scheithauer, für die Diskussion fachlicher Fragen und hilfreichen Anregungen zu danken. Nicht zuletzt gilt mein Dank Kathrin Kasecker für das intensive Korrekturlesen dieser Arbeit und meiner Familie für ihre Unterstützung.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Chemnitz, im September 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Die Entwicklung eines digitalen technischen Systems unterliegt heutzutage hohen Anforderungen hinsichtlich Entwicklungszeit und Entwicklungskosten. Der Gewinn, den ein Produkt erzielt, fällt umso größer aus, je schneller es auf dem Markt verfügbar ist. Das Ziel des Entwicklungsprozesses liegt demzufolge darin, ein Produkt schnell und mit möglichst geringen Entwicklungskosten auf den Markt zu bringen.

Ein Schritt in diese Richtung stellt ein modulares Design dar. Es folgt dem Grundsatz, dass ein System aus verschiedenen Modulen aufgebaut ist (vgl. Abbildung 1.1). Die Entwicklung und der Test der einzelnen Module kann unabhängig voneinander erfolgen. Somit kann der Entwicklungsaufwand auf mehrere Personen oder Teams verteilt werden und die Entwicklungszeit reduziert sich durch die Parallelisierung der Entwurfsarbeit. Das Gesamtsystem ergibt sich aus der Zusammenführung und Kopplung der einzelnen Module.

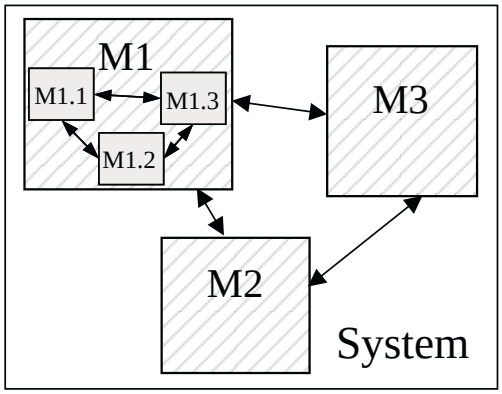


Abbildung 1.1: Schema eines modularen Designs

Einen weiteren Vorteil dieses Konzeptes stellt die Wiederverwendung (Reuse) von bereits entwickelten Modulen in neuen Systemen dar. Diese Intellectual Properties (IPs, vgl. Abschnitt 2.2) verkürzen den Entwicklungsaufwand für neue Systeme erheblich, da diese Module nicht neu entwickelt werden müssen. Im Idealfall können sie ohne Änderungen in das Design eingebunden werden.

Ein aktuelles Problem mit dem die Entwickler konfrontiert sind, bildet eine Lücke im Entwurfsprozess. Abbildung 1.2 stellt diesen Sachverhalt dar. Die

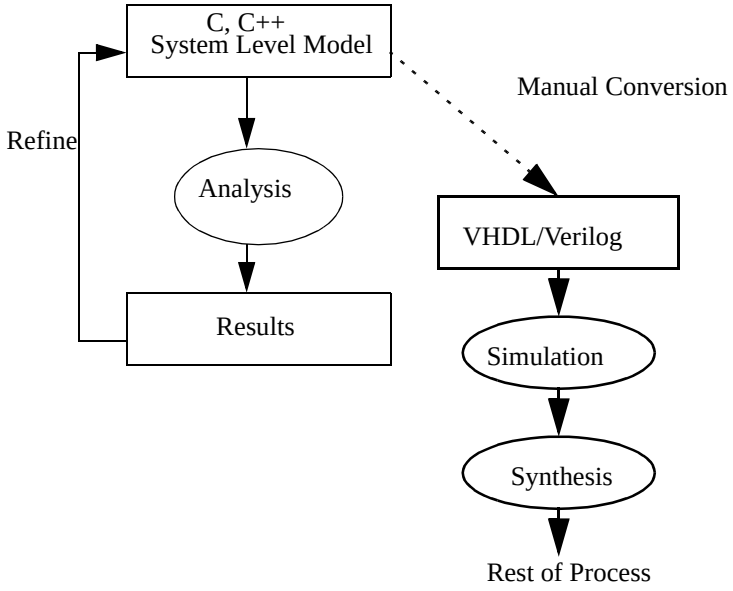


Abbildung 1.2: Aktuelle Methode des Hardwareentwurfsprozesses, Quelle [Opeb]

Entwicklung eines Systems beginnt mit einer Spezifikation auf Systemebene. Die dafür verwendete Sprache ist meist C oder C++. Die Spezifikation wird schrittweise analysiert und verfeinert. Dabei durchläuft die Systementwicklung verschiedene Abstraktionsebenen des Entwurfsprozesses (vgl. Abbildung 1.3 und 1.4). Das Ergebnis eines erfolgreichen Entwurfsprozesses besteht in der Implementierung des Systems, die wie in der Spezifikation festgelegt arbeitet.

Ab einem Verfeinerungsgrad, der einer Verhaltensbeschreibung entspricht, reichen die bestehenden Beschreibungsmöglichkeiten von C/C++ nicht mehr aus und es muss auf eine Hardwarebeschreibungssprache, wie zum Beispiel VHDL oder Verilog, übergegangen werden. Dieser Schritt stellt beim aktuellen Entwurfsprozess noch ein Hindernis dar, da er vom Entwickler von Hand durchgeführt werden muss. Dieser Prozess beansprucht viel Zeit und ist fehleranfällig. Nach der Konvertierung muss das Design ausgiebigen Tests unterzogen werden, um sicherzustellen, dass die Konvertierung mit der Spezifikation übereinstimmt. Dadurch verlängert sich die Entwicklungszeit und teure Resourcen werden gebunden.

Wünschenswert ist nun ein Entwurfssystem, das es erlaubt, den Entwurf von Hardware über alle Entwurfsebenen modular und in einer einheitlichen

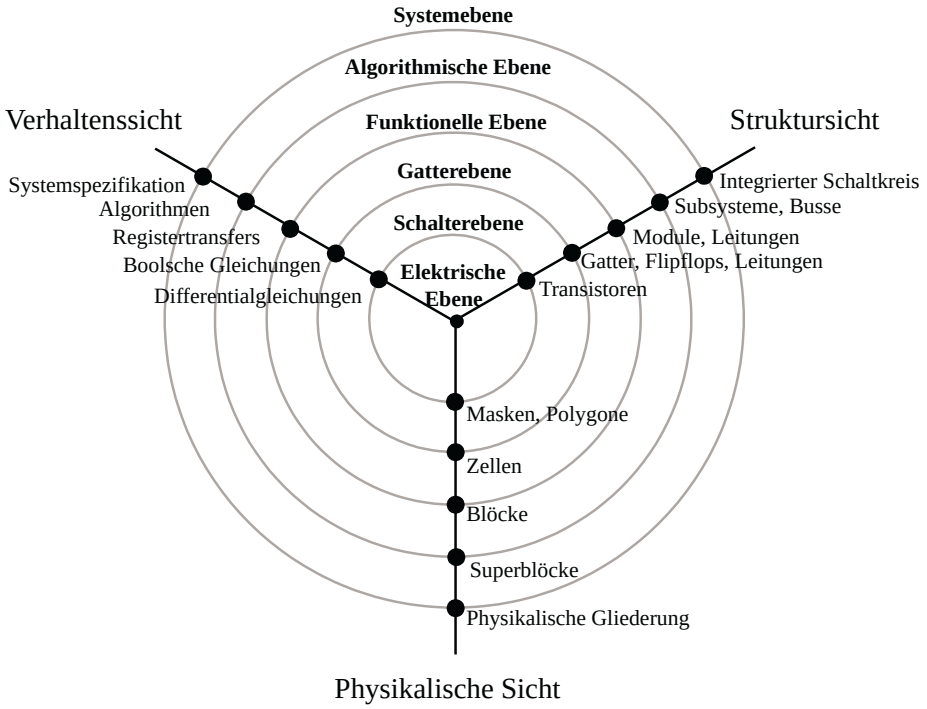


Abbildung 1.3: Das Y-Diagramm nach Gajski. Es zeigt die verschiedenen Sichten und Abstraktionsebenen beim Hardwareentwurf.

Beschreibungssprache durchzuführen. Für den zunehmend wichtiger werdenden Entwurf von eingebetteten Systemen sollte das Entwurfssystem zudem die Unterstützung der dafür benötigten Entwurfsdimensionen (vgl. Abbildung 1.5) gewährleisten. Außerdem sollte es die Verwendung von Intellectual Properties auf jeder Verfeinerungsebene für den Entwurf und den Test des Systems unterstützen.

Um diese Anforderungen zu erfüllen, wurde SystemC (Abschnitt 2.1) entwickelt. SystemC unterstützt den modularen Entwurfsprozess von der Systemspezifikation bis zur Register-Transfer-Ebene. Außerdem erlaubt es Hardware/Software-Codesign, was für die Entwicklung von HW/SW-Systemen vorteilhaft ist. Die Einbindung von sprachfremden Intellectual Properties ist als Netzliste möglich. Eine Netzliste ist eine synthetisierte Register-Transfer-Beschreibung. In höheren Abstraktionsebenen können diese Intellectual Properties nicht in ihrer Originalform verwendet werden. Zur Simulation eines Systems in diesen Abstraktionsebenen muss ein Modell der IP erstellt werden. Dieser Prozess ist fehleranfällig und kostet Zeit und Geld, was den Erfolg des Produktes weniger gut ausfallen lassen kann.

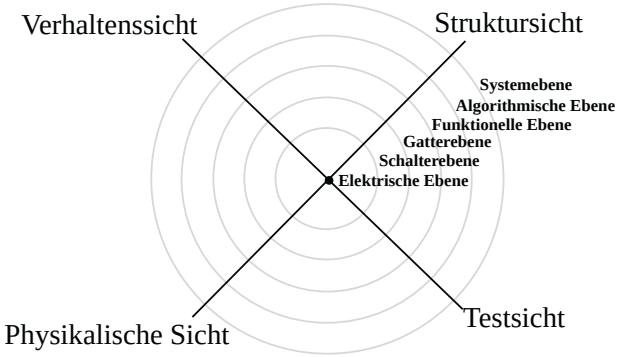


Abbildung 1.4: Das X-Diagramm nach Rammig. Gegenüber dem Y-Diagramm nach Gajski bringt es die Testsicht zusätzlich in jede Entwurfsebene mit ein.

Die Umgehung der Erstellung eines Modells der Intellectual Property könnte den Entwurfsprozess weiter beschleunigen. Jedoch muss dazu eine Lösung gefunden werden, um die originalen IPs bereits in die Simulation höherer Abstraktionsebenen einzubinden. Die Entwicklung einer solchen Lösung ist Ziel der vorliegenden Arbeit.

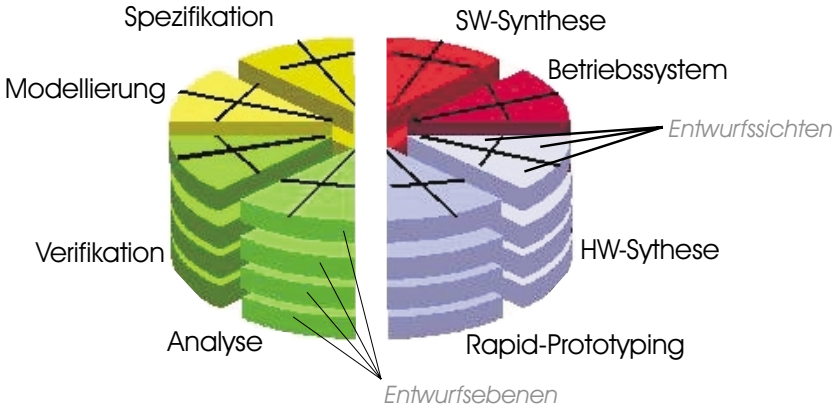


Abbildung 1.5: Die P-Chart Entwurfsstrukturierung nach [Har02] für den Entwurf von HW/SW-Systemen

1.1 Aufgabenstellung

Der Entwurf von Hardware-Software-Systemen wurde bisher dadurch erschwert, dass eine Lücke im Entwurfsprozess zwischen der System-Level-Beschreibung und der Hardwarebeschreibung existierte. Um diese Lücke zu schließen, wurde SystemC entwickelt.

Jedoch ist ein Nachteil dieser Systementwurfsprache, dass man bisher entwickelte Komponenten in VHDL oder Verilog, so genannte Intellectual Properties, erst auf Netzlistenebene in SystemC einbinden kann. Für die Simulation von höheren Abstraktionsebenen muss erst ein SystemC Modell der Komponente erstellt werden. Das bedeutet zusätzliche Entwicklungsarbeit und damit höhere Entwurfskosten.

In dieser Arbeit soll ein Co-Simulationsansatz auf Systemebene untersucht werden. Dabei soll die bereits synthesefähige VHDL-Komponente auf einem FPGA abgearbeitet werden und mit der Simulation der restlichen SystemC-Komponenten gekoppelt werden. Der Entwurfsaufwand reduziert sich und das Verhalten des Systems mit der realen Intellectual Property kann untersucht werden.

Zur einfacheren und schnelleren Adaptierung des FPGA an die SystemC-Simulation, soll ein automatisiertes Verfahren entwickelt werden. Der Vorteil eines automatisierten Verfahrens besteht darin, Zeit und Kosten bei der Vorbereitung der Simulation zu sparen.

1.2 Gliederung der Arbeit

Das Thema dieser Arbeit ist die **Automatische Adaption**[1] **von Hardware-Acceleratoren für Verhaltenssimulation**. Das erste Kapitel enthält eine kurze Einführung in die Thematik und die Aufgabenstellung. Kapitel 2 beschreibt den aktuellen Stand von Techniken, auf die diese Arbeit aufbaut oder die mit dem Thema verwandt sind. Mit der Methode der Adaptierung von Hardwareakzeleratoren an die SystemC Verhaltenssimulation beschäftigt sich Kapitel 3. Ein allgemeines Lösungskonzept wird darin vorgestellt und darauf

[1] Für den englischen Begriff Adaption wird im weiteren Verlauf dieser Arbeit der deutsche Begriff Adaptierung verwendet.

aufbauend eine technische Umsetzung geboten. Kapitel 4 beschreibt die Automatisierung des vorgestellten Lösungskonzeptes durch ein, im Rahmen dieser Arbeit entwickeltes, Programm. Der Nachweis über die korrekte Funktion der erarbeiteten Lösungen wird in Kapitel 5 an einem Beispiel durchgeführt. Das 6. Kapitel fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf Zukunftsperspektiven und Möglichkeiten der Weiterentwicklung.

2 Stand der Technik

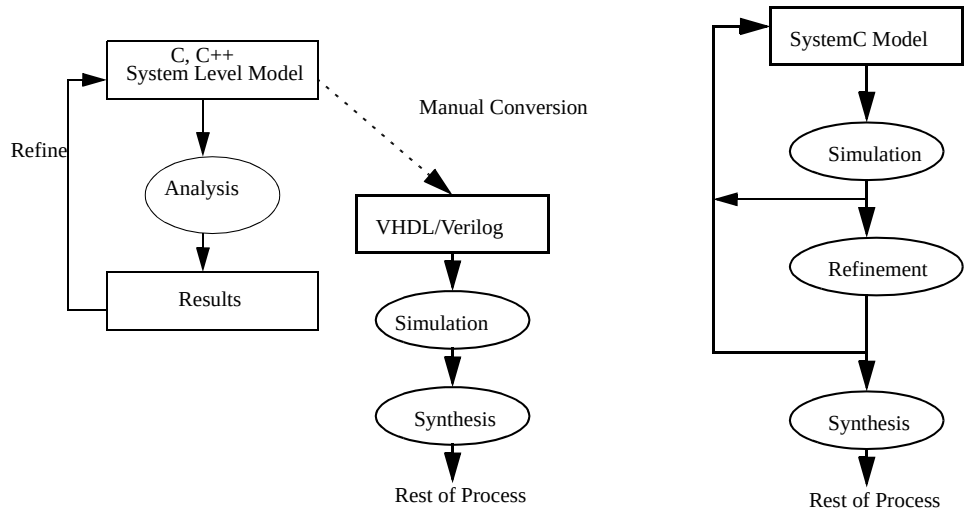
Dieses Kapitel beschäftigt sich mit dem aktuellen Stand der Techniken, auf die diese Arbeit aufbaut. Zunächst erfolgt eine Beschreibung der noch jungen Systementwurfssprache SystemC. Der zweite Teil des Abschnittes definiert den Begriff der Intellectual Properties und umreißt deren Bedeutung. Anschließend wird auf die Rolle der Simulation bei der Hardwareentwicklung eingegangen und einige Beispiele für Simulationswerkzeuge, darunter auch Hardwareakzeleratoren vorgestellt. Das Ende des Kapitels faßt die hier gewonnen Erkenntnisse zusammen und arbeitet noch einmal kurz die Vor- und Nachteile aktueller Techniken heraus.

2.1 SystemC

SystemC™[2] ist eine standardisierte Systementwurfs- und Verifikationssprache. Sie wurde im September 1999 eingeführt und durch die OSCI (Open SystemC™ Initative) standardisiert. Die OSCI ist ein Konsortium größerer EDA-Firmen und IP-Providern. Mitglieder dieses Konsortiums sind zum Beispiel ARM Ltd., Cadence Design Systems Inc., Synopsys Inc., Motorola, Panasonic und andere. Mehr Informationen zu den Mitgliedern des Konsortiums befinden sich auf der Homepage von SystemC [Opea].

[2] SystemC ist ein eingetragenes Warenzeichen der Open SystemC™Initiative.

Abbildung 2.1: Logo von SystemC™, Quelle [Opea]



Gegenwärtige Designmethodik SystemC Designmethodik

Abbildung 2.2: Gegenwärtige und SystemC Designmethodik, Quelle [Opeb]

SystemC ist eine C++ Klassenbibliothek und nutzt somit die Möglichkeit, C++ durch Klassenbibliotheken zu erweitern ohne neue syntaktische Konstrukte hinzuzufügen. Der Entwickler kann somit die ihm vertraute Sprache C++ und die damit verbundenen Entwicklungswerkzeuge weiterhin verwenden. SystemC wird über die Open Source Lizenz [VA] vertrieben. Die Bibliothek kann somit kostenlos genutzt werden. Der Nutzer enthält mit der Bibliothek ein kostenloses Entwurfs- und Simulationspaket. Es enthält die notwendigen Konstrukte um Systemarchitekturen, inklusive Hardwaretiming, Nebenläufigkeit und reaktivem Verhalten, zu modellieren, die nicht in Standard C++ enthalten sind. Außerdem verfügt die Bibliothek bereits über einen Simulator mit dem die entworfenen Designs getestet werden können. Die Beschreibung des Simulators folgt in Abschnitt 2.3.1.

Mit SystemC kann ein Entwickler effektive zyklengenaue Modelle von Softwarealgorithmen, Hardwarearchitekturen und Schnittstellen eines System-on-Chip- (SoC) und System-Level-Designs beschreiben. Außerdem bietet es die Möglichkeit eine ausführbare Spezifikation zu erstellen. Auch Hardware/Software-Codesign, welches in heutigen Systemen eine immer größere Bedeutung erlangt, wird unterstützt.

SystemC unterstützt den Systementwurf von der Spezifikation bis zur Register-Transfer-Ebene. Bei bisher verwendeten Beschreibungsmöglichkeiten bestand

eine Lücke im Entwurfsprozess. Zum Beispiel stehen in C/C++ keine geeigneten Beschreibungsmittel für Hardware zur Verfügung, da Konzepte, wie Nebenläufigkeit und ein Zeitmodell, fehlen. Deshalb war bisher eine manuelle Konvertierung von einem C/C++ Verhaltensmodell in eine, durch eine Hardwarebeschreibungssprache modellierte, Regsiter-Transfer-Beschreibung notwendig. Abbildung 2.2 stellt die bisher und auch gegenwärtig verwendete Designmethode und die Designmethodik von SystemC gegenüber.

Als junges Entwurfssystem gibt es allerdings noch Einschränkungen. So ist beim Softwaredesign die Einbindung von Echtzeitbetriebssystemen (RTOS) noch nicht möglich. Eine Verbesserung soll Version 3 von SystemC bringen.

Als Werkzeuge zur Synthese kommt der CoCentric® SystemC™ Compiler der Firma Synopsys® [Syn] zum Einsatz. Er erlaubt sowohl eine Synthese von der Register-Transfer-Ebene als auch von der Verhaltensebene. Der Compiler ist jedoch nicht in der kostenfreien Klassenbibliothek enthalten und muss gesondert gekauft werden. Mehr Informationen zum CoCentric® SystemC™ Compiler sind in [Syn03] zu finden.

2.2 Intellectual Properties

Gorden Moore formulierte 1965 das nach ihm benannte Mooresche Gesetz. Es besagt, dass sich die Anzahl der Transistoren auf einer gegebenen Fläche Silizium etwa alle 18 Monate verdoppelt. Abbildung 2.3 stellt diesen Sachverhalt grafisch dar. Dieses Gesetz hat bis heute noch Gültigkeit und wird nach Meinung von Experten auch noch 10 Jahre seine Gültigkeit behalten.

Für die Hardwareentwickler ergibt sich die Konsequenz, dass auf der gleichen Fläche Silizium eine immer größere Funktionalität untergebracht werden kann. Die Folge ist ein System-on-Chip-Design, bei dem versucht wird, ein digitales System auf einem einzigen Chip zu implementieren.

Der Vorteil einer System-On-Chip-Lösung ist eine größere Zuverlässigkeit gegenüber Systemen aus mehreren Chips. Schwachstellen liegen bei diesen Systemen vor allem in der physischen Verdrahtung der Chips untereinander.

Der Nachteil eines System-on-Chip ist eine größere Spezialisierung der Funktion. Mit einer größeren Fülle an Funktionen sinkt das Anwendungsfeld der Chips. Doch je kleiner das Anwendungsfeld, desto weniger Stückzahlen werden vom Chip gebraucht. Jedoch müssen sich mit dem Verkauf des Chips auch

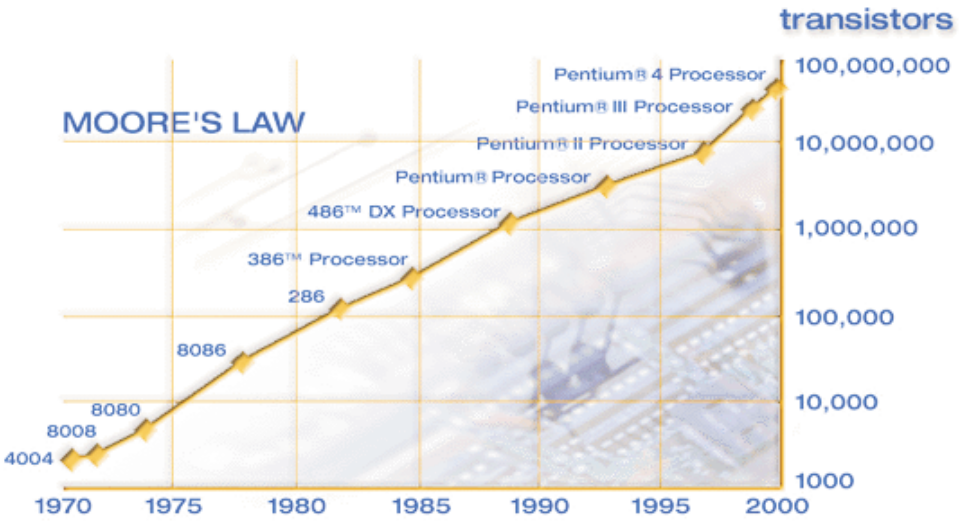


Abbildung 2.3: Darstellung von Moores Gesetz, Quelle [Int]

die Entwurfskosten amortisieren. Damit sich der Entwurf von SoC-Lösungen auch weiterhin lohnt, müssen die Entwicklungskosten und die Time-to-Market[3] sinken.

Strategien zur Senkung der Entwicklungskosten und der Time-to-market stellen zum Einen eine weitere Entwurfsautomatisierung und zum Anderen die Wiederverwendung bereits entwickelter Komponenten dar. Die Wiederverwendung von Komponenten basiert auf der Nutzung von Intellectual Properties.

Intellectual Properties (dt. Geistiges Eigentum) ist nach der allgemeinen Definition jedes Produkt des menschlichen Intellekts, das einzigartig, neuartig und nicht offensichtlich ist. Dazu gehören literarische, künstlerische und wissenschaftliche Arbeiten, Leistungen von Künstlern, Erfindungen auf allen Gebieten der menschlichen Erkenntnis, wissenschaftliche Entdeckungen, industrielle Designs, eingetragene Waren- und Dienstleistungsmarken, Handelsnamen und Kennzeichnungen sowie alle weiteren Rechte aus intellektueller Tätigkeit in den Gebieten der Industrie, Wissenschaft, Literatur oder Kunst. [Wor, Mar02, Mic04]

[3]Time-to-market gibt die Zeit an, die von der Idee bis zur Marktreife eines Produktes vergeht.

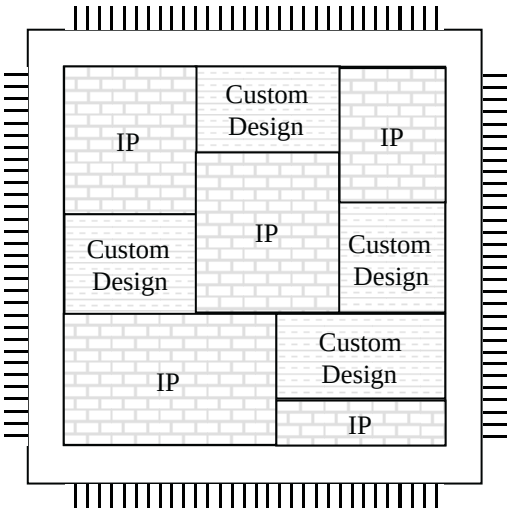


Abbildung 2.4: Aufbau eines System-on-Chip aus IPs und selbstentwickelten Komponenten

Eine Intellectual Property (IP) im Sinne der Hardwareentwicklung stellt eine Hardwarekomponente dar, die bereits entwickelt wurde. Diese Komponenten werden in neue Designs neben selbst entwickelten Komponenten eingebunden (Abbildung 2.4), im Idealfall ohne daran Änderungen vornehmen zu müssen. Dadurch entfällt die Entwicklung dieser Teilsysteme im Entwurfsprozess und Entwicklungszeit wird eingespart.

Die IP kann in verschiedenen Beschreibungen vorliegen [Har04]. Man unterscheidet zwischen Hard-IPs und Soft-IPs. Hard-IPs besitzen ein vorgegebenes Layout und Timing. Sie haben fixe Schnittstellen. Die Optimierung der IPs ist in Bezug auf Geschwindigkeit, Siliziumfläche und Leistungsaufnahme getroffen. Sie sind leicht zu verifizieren, außerdem halbleiterhersteller- und technologieabhängig. Jedoch besitzen sie den Nachteil, dass sie unflexibel sind.

Soft-IPs basieren auf einer synthetisierbaren Beschreibung ausgehend von einer Hardwarebeschreibungssprache wie zum Beispiel VHDL oder Verilog. Eine Soft-IP ist vorcompiliert als Netzliste oder als Quellcode verfügbar. Sie sind halbleiterhersteller- und technologieunabhängig konzipiert. Soft-IPs sind flexibel und zum Teil parametrisierbar.

Das Konzept der IPs wird als wichtiger Schritt angesehen, um die Entwicklungszeiten von Hardware zu verringern. Vor allem für die System-on-Chip-Entwicklung stellt es ein Schlüsselkonzept dar.

Jedoch gibt es bei der IP-Nutzung noch einige Hindernisse. Zum Beispiel stellt sich die Frage nach einem einheitlichen Qualitätkriterium für IPs, um bei der Auswahl von IPs einheitliche Vergleichskriterien zu haben. Zu den Qualitätsmerkmalen einer IP zählen unter anderem der Umfang der Dokumentation, Skripte zur Logiksynthese, Applikationsbeispiele, Simulationsergebnisse, Testdaten und Verifikationsergebnisse.

Um bestehende Hindernisse bei der Nutzung und dem Handel von IPs zu bewältigen, wurde das Projekt IPQ [ipq] ins Leben gerufen. Die Zielsetzung des Projektes liegt darin, eine entscheidende Verbesserungen für die Qualitätssicherung bei der Anwendung und Entwicklung von IP-Modulen zu erzielen. Dazu gehören Spezifikationsmethoden, eine intelligente IP-Suche, Eingangschecks von IPs, Verfahren zur IP-Anpassung und Beiträge zur Standardisierung.

Das dabei auftretende Problem lag im Nichtvorhandensein eines einheitlichen Beschreibungsstandards über Firmengrenzen hinweg. Jede Firma benutzt ihre eigenen internen Standard und verzichtet auch ungern auf diese, da eine Umstellung mit hohen Kosten verbunden ist. Dadurch wird der IP-Handel erschwert, denn die IPs müssen nach dem Kauf manuell an den firmeninternen Standard angepasst werden.

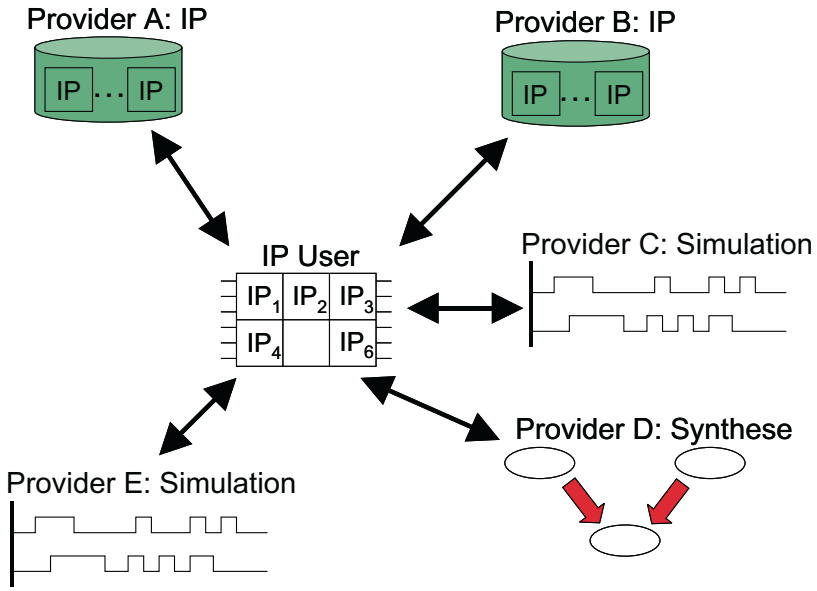


Abbildung 2.5: IPQ soll eine standardisierte Kommunikation von IP-User und verschiedenen IP-Serviceprovidern erlauben. Quelle [VH04]

Zur Lösung des Problems musste ein standardisiertes und von den Firmen akzeptiertes Austauschformat gefunden werden. Dieses Format musste außerdem die Möglichkeit bieten, eine firmeninterne Beschreibungen in das standardisierte Format zu konvertieren, aber auch den Weg vom standardisierten in firmeninterne Formate gewährleisten. Als Lösung wurde das IPQ-Format entwickelt. Es basiert auf XML-Schema. Außerdem fand die Entwicklung zahlreicher Tools statt, die zum Beispiel das Suchen einer IP vereinfachen. Weitere Informationen zu diesem Thema bieten [ipq], [VLKH04], [VH04] und [VLH+03].

Mit Hilfe des IPQ-Formates und der dazu entwickelten Tools wird eine standardisierte Kommunikationsmöglichkeit von IP-Usern und IP-Serviceprovidern (vgl. Abbildung 2.5) geboten. Das System ist bereits einsatzfähig. Jedoch scheitert sein Einsatz durch teilweise ungeklärte rechtlichen Fragen zum IP-Handel und zur IP-Nutzung. Doch auch für dieses Problem wird sich eine Lösung finden und eine intensivere Nutzung von IPs beim Systementwurf wird Einzug halten. Damit lassen sich dann die Entwicklungszeit und -kosten weiter senken.

2.3 Simulation

Der Begriff Simulation wurde vom Verein Deutscher Ingenieure (VDI) in der Richtlinie 3633 [Ver96] wie folgt definiert: „Simulation ist ein Verfahren zur Nachbildung eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.
Im weiteren Sinne wird unter Simulation das Vorbereiten, Durchführen und Auswerten gezielter Experimente mit einem Simulationsmodell verstanden.
Mit Hilfe der Simulation kann das zeitliche Ablaufverhalten komplexer Systeme untersucht werden.“

Die Simulation ist ein wichtiges Werkzeug beim Hardwareentwurf, um ein System während des Entwurfsprozesses zu testen und Fehler frühzeitig zu lokalisieren und zu beseitigen. Die Durchführung von Simulationen soll den Nachweis erbringen, dass der verfeinerte Entwurf noch mit der Spezifikation des Systems übereinstimmt.

Den Ausgangspunkt einer Hardwaresimulation bildet eine Beschreibung in einer Hardwarebeschreibungssprache (HDL, engl. Hardware Description Language). Die wichtigsten Sprachen sind VHDL, Verilog und SystemC. Auf

Entwurfsebene	**Aufwand (relativ)**
Verhalten	1
Register-Transfer	10
Gatter (Logik)	100
Schalterebene	1 000
Elektrische Ebene	10 000
Geometrie	100 000 - 1 000 000

Tabelle 2.1: Simulationsaufwand von verschiedenen Entwurfsebenen, Quelle [Mül02]

Basis dieser Beschreibung wird von einem Simulationswerkzeug ein Simulationsmodell erstellt. Ein Simulationsmodell kann unterschiedliche Abstraktionsebenen eines Systems nachbilden. Dabei erhöht sich der Detailgrad des Simulationsmodells mit steigender Nähe zur Implementierung.

Mit steigendem Detailgrad nähert sich die Genauigkeit des Simulationsmodells stark dem realen System an. Durch die Berücksichtigung von Verzögerungszeiten von Gattern und Leitungen, sowie dem Übergang zu einer vierwertigen Logik, entsprechen die Ergebnisse der Simulation den real zu erwarteten Daten des zu implementierenden Systems.

Der hohe Detailgrad der Simulation wirkt sich jedoch stark auf die Simulationsgeschwindigkeit aus. Durch eine steigende Anzahl zu berücksichtigender Parameter in einem Simulationsschritt, steigt der Simulationsaufwand stark an und die Simulationsgeschwindigkeit verringert sich. Tabelle 2.1 stellt diesen Sachverhalt dar. Dauert eine Simulation auf der Verhaltensebene nur ein paar Minuten, kann die gleiche Simulation auf Gatterebene durchaus mehrere Stunden beanspruchen. Deshalb sollte bei der Vorbereitung einer Simulation darauf geachtet werden, welche Genauigkeit die Simulationsdaten zur Verifizierung der aktuellen Entwurfsebene besitzen müssen und dementsprechend das Simulationsmodell auszuwählen.

Es existieren eine Vielzahl von Simulatoren und Simulationsmethoden, die die Simulation unterschiedlicher Abstraktionsebenen beherrschen. Einige Simulatinsmethoden versuchen zudem, dem Zeitverlust, der durch genauere Modelle entsteht, mit verschiedenen Ansätzen entgegenzuwirken. Eine kleine Auswahl von Simulatoren und Simulationsmethoden sollen die folgenden Abschnitte vorstellen.

2.3.1 Simulation mit SystemC

SystemC ist eine C++ Klassenbibliothek in der auch ein Simulator vorhanden ist. Eine ausführliche Beschreibung von SystemC enthält Abschnitt 2.1.

Der Ablauf der Simulation ist in Abbildung 2.6 dargestellt. Den Ausgangspunkt stellt das in SystemC beschriebene Modell eines Systems dar. Die einzelnen Module des Modells und der Testbench werden von einem C++-Compiler übersetzt und mit dem Simulationskern aus der Bibliothek zusammengelinkt. Das Ergebnis ist ein ausführbares Programm, welches die Simulationsergebnisse erzeugt.

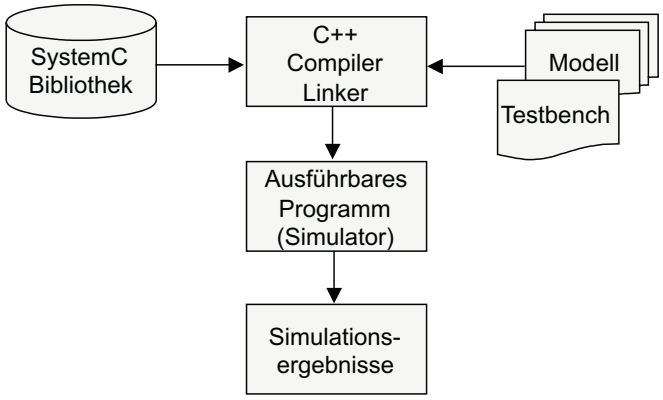


Abbildung 2.6: Simulation eines Modells mit SystemC

Die SystemC-Simulation ist zyklusbasiert. Die Aktualisierung von Prozessen und Signalen erfolgt mit dem Übergang des Taktsignals. Die Folge ist, dass die Genauigkeit der Simulation auf eine Taktperiode beschränkt ist. Die SystemC-Bibliothek enthält zur Simulationssteuerung einen zyklusbasierten Scheduler. Er behandelt alle Ereignisse von Signalen und führt die Prozesse aus, deren Eingangssignale sich ändern. Die manuelle Simulationskontrolle ist beschränkt auf das Starten, Stoppen und schrittweises Ausführen der Simulation durch SystemC-Funktionen.

Die Ausgabe der Simulationsergebnisse kann in Tracefiles erfolgen. Dazu bietet die Bibliothek spezielle Funktionen, die es auch erlauben, die Tracefiles in verschiedenen Formaten zu generieren. Formate die unterstützt werden sind das Integrated Signal Data Base (ISDB) Format, Waveform Intermediate Format (WIF) und das Value Change Dump (VCD) Format. Eine andere Möglichkeit die Simulationsdaten auszugeben, besteht in der Nutzung von vorhandenen C++-Ausgabefunktionen wie *cout*.

Weitere Informationen zu SystemC und der Simulation damit finden sich im Benutzerhandbuch [Opeb] sowie in der funktionalen Spezifikation [Ope02].

2.3.2 Hardwaresimulation mit ModelSim

ModelSim® ist eine HDL-Simulationssoftware, die von der Firma Model Technology [Mod] entwickelt wurde. Model Technology ist eine betriebseigene Tochtergesellschaft von Mentor Graphics [Mena]. ModelSim unterstützt die Simulation von VHDL, Verilog und SystemC. Dabei können Simulationsmodelle von der Verhaltenssimulation über Netzlistenmodelle bis hin zur platzierten und trassierten Schaltung mit einem genauen Zeitmodell simuliert werden. Den Simulator gibt es für alle gängigen Betriebssysteme.

Abbildung 2.7: Logo von ModelSim®, Quelle [Mod]

Zum Simulator gehört ein Projektmanager zur Kontrolle der Simulation. Zusätzlich bietet er eine integrierte Debugumgebung, die es erlaubt, auf alle Signale des Designs zuzugreifen. Der Verlauf der einzelnen Signale wird mit einem Waveformbetrachter visualisiert und kann dadurch genau verfolgt werden. Ein Wizard und Templates erlauben dem Benutzer, Testbenches schnell zu erstellen. ModelSim benutzt als Simulationssprache eine Tcl basierte Skriptsprache. Dadurch können an den Simulator leicht andere Programme angebunden werden.

Die Simulation erfolgt auf die folgende Weise. Die Hardwarebeschreibung wird zunächst plattformunabhängig übersetzt. Anschließend erzeugt der Compiler daraus einen maschinenspezifischen Code für die ausführende Maschine, der zur Laufzeit optimiert wird. Die Ausführung des Codes erzeugt die Simulationsdaten des Modells. ModelSim zeichnet die Ausgaben des Codes auf und stellt sie zur Auswertung dem Nutzer zur Verfügung.

Von ModelSim existieren verschiedene Versionen. Die meisten Features stellt die LE Version zur Verfügung. Mehr Informationen zum Simulator bietet die Homepage von Model Technology [Mod] und das Handbuch zu ModelSim [Mod03].

2.3.3 HW/SW-Cosimulation

Hardware/Software-Codesign ist ein Schlüsselkonzept, um die Entwicklung moderner Hardware/Software-Systeme zu vereinfachen und zu beschleunigen. Anstatt Hardware und Software getrennt zu entwickeln und beide Bereiche im Anschluss mühevoll zusammenzufügen, erlaubt Hardware/Software-Codesign die Entwicklung beider Bereiche zu parallelisieren. Die Cosimulation von Hardware und Software liefert dabei ein wichtiges Hilfsmittel, um das Design zu testen und zu verifizieren.

Die HW/SW-Cosimulation verfolgt zwei konkurrierende Ziele. Das erste Ziel besteht darin, die Simulation mit der größtmöglichen Geschwindigkeit zu betreiben. Dem entgegen steht ein hoher Detailgrad der Simulationsergebnisse. Zum Erreichen einer hohen Genauigkeit der Simulationsergebnisse werden Modelle mit einem hohen Detailgrad verwendet. Aber je mehr Details simuliert werden, umso geringer ist die Simulationsgeschwindigkeit. Aus diesem Grund existieren verschiedene Ansätze zur HW/SW-Cosimulation, die versuchen, die Simulationsgeschwindigkeit zu erhöhen und dabei möglichst genaue Ergebnisse zu erzielen.

Der ursprüngliche Ansatz besteht darin einen **Hardwaresimulator** zu nutzen. Die Simulation findet unter Nutzung einer einzigen Simulationsumgebung statt, in der sowohl Hardware als auch die darauf ausgeführte Software simuliert wird. Dieser Ansatz bietet eine geringe Simulationsgeschwindigkeit, da das simulierte Hardwaremodell einen hohen Detailgrad besitzt. Der Vorteil besteht darin, dass keine Schnittstellen zu anderen Systemen notwendig sind.

Zur Beschleunigung der Simulation kann ein **Bus-Modell** verwendet werden (Abbildung 2.8). Ein Bus-Modell ist eine Hardwarebeschreibung, die nur ein ereignisdiskretes Modell des Bus-Interfaces des Prozessors simuliert. Die

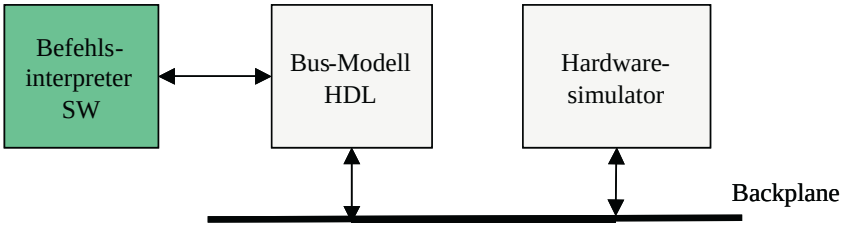


Abbildung 2.8: HW/SW-Cosimulation mit dem Bus-Modell

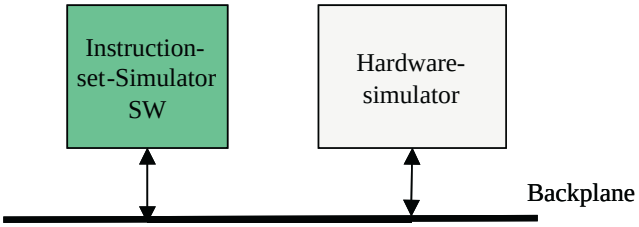


Abbildung 2.9: HW/SW-Cosimulation mit einem Instruction-set-Simulator

Funktionalität des Prozessors, dass heißt die Applikationssoftware, führt ein Befehlsinterpreter aus. Er verfügt über Information zu den benötigten Taktzyklen einer Sequenz von Befehlen zwischen zwei I/O-Operationen auf dem Bus. Dieses Modell ist nützlich, um die low-level Interaktion der Kommunikation auf dem Prozessorbus zu simulieren. Jedoch ist es unter Umständen nicht einfach, ein genaues Busmodell eines Prozessors zu erstellen.

Einen anderen Ansatz verfolgen **Instruction-set Simulatoren (ISS)**. Ihn stellt Abbildung 2.9 dar. Ein ISS bildet ein Modell der Befehlssatzarchitektur eines speziellen Prozessors nach. Das Modell enthält alle Details eines ereignisdiskreten Prozessormodells, welches die volle Funktionalität des Prozessors bereitstellt. Dieser Cosimulationsansatz verbindet einen Hardwaresimulator, auf dem die Hardwarekomponenten simuliert werden und den ISS, der die Ausführung der Software übernimmt.

Die **Heterogene Cosimulation** verfolgt die Kopplung von Hardware- und Software-Entwicklungstools. Dies erlaubt die schnelle Ausführung von Software-Anwendungscode auf einer simulierten Hardware. Die Cosimulation verschiedener Hardwarekomponenten und der Software läuft in einem heterogenen Netzwerk aus PC's oder Workstations.

Der Ansatz des **Hardware-Modells**, wie ihn Abbildung 2.10 zeigt, kann

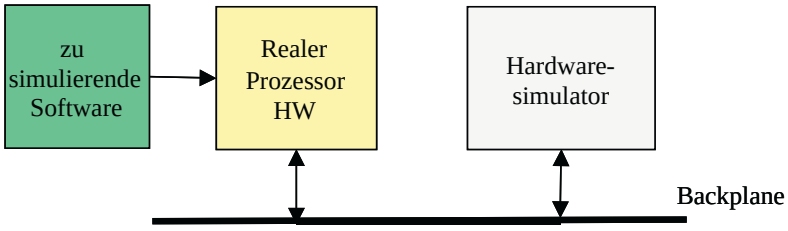


Abbildung 2.10: HW/SW-Cosimulation mit dem Hardware-Modell

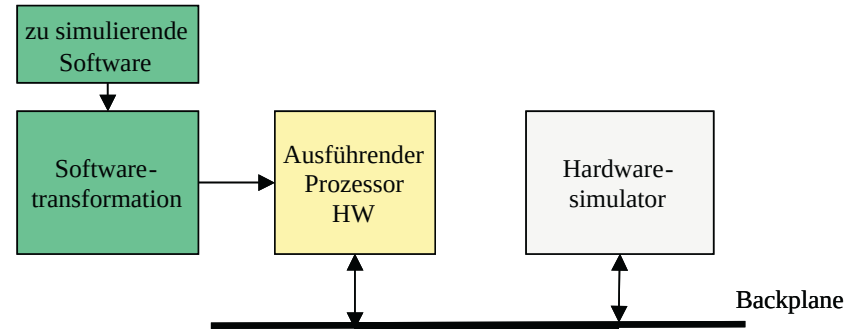


Abbildung 2.11: HW/SW-Cosimulation mit dem compilierten Modell

benutzt werden, falls der Zielprozessor bereits existiert. Zur Ausführung der Software findet der Zielprozessor statt eines Simulators Verwendung. Damit ist eine Echtzeitausführung der Software möglich. Die Simulation der Hardwarekomponenten findet durch einen Hardwaresimulator statt, der mit dem Prozessor gekoppelt ist.

Beim **compilierten Modell** kommt der Prozessor des ausführenden Rechners zum Einsatz (Abbildung 2.11). Die zu simulierende Software wird in Code dieses Prozessors transformiert und auf diesem ausgeführt. Eine Schnittstelle verbindet die Softwareausführung mit dem Hardwaresimulator, der die Ergebnisse der Hardwarekomponenten berechnet.

Die höchste Simulationsgeschwindigkeit erzielt unter allen Ansätzen die **Hardwareemulation**, welche Abbildung 2.12 veranschaulicht. Dieses Verfahren nutzt programmierbare Hardwarebausteine, wie zum Beispiel FPGAs. Auf Basis dieser Bausteine werden die entwickelten Hardwarekomponenten implementiert. Die Softwareausführung findet auf dieser Implementierung statt.

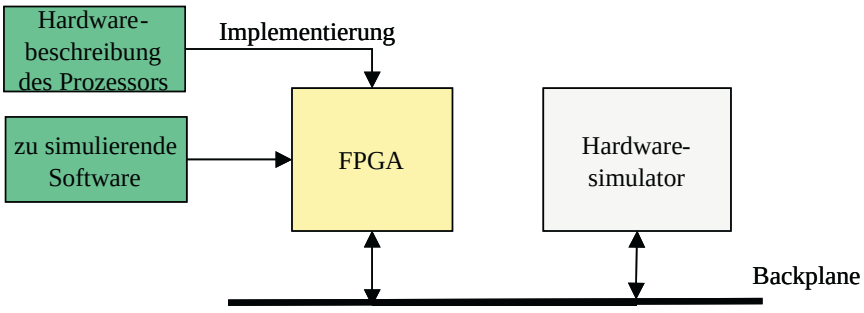


Abbildung 2.12: HW/SW-Cosimulation mit Hardwareemulation

Die Geschwindigkeit, die dieser Ansatz erreicht, kann bis zu einem Zehntel der realen Ausführungsgeschwindigkeit betragen.

Bei der Verwendung von mehreren Simulatoren spielt die Schnittstelle zwischen ihnen eine große Rolle. Die Schnittstelle stellt in den meisten Fällen den größten Engpass dar und ist somit ein wichtiger Faktor für die resultierende Simulationsgeschwindigkeit. Die Kopplung der Simulatoren kann auf verschiedene Arten erfolgen.

Die Simulatoren können parallel arbeiten und dabei durch einen Synchronisationsmechanismus für den Datenaustausch gekoppelt sein. Das hat zur Folge, dass ein schneller Simulator eventuell lange auf einen langsameren warten muss. Außerdem entsteht durch die Synchronisation ein hoher Kommunikationsoverhead für die Schnittstellen.

Ein anderer Kopplungsansatz besteht in einer Master-Slave-Simulation. Dabei übernimmt ein Simulator die Rolle des Masters und ruft andere Simulatoren auf, wenn er Ergebnisse von ihnen benötigt. Bei der Ausführung mehrere Simulatoren auf einem Prozessor bringt diese Variante Vorteile. Erstens wird jeweils nur ein Simulator ausgeführt, dem die gesamte Prozessorleistung zur Verfügung steht. Zusätzlich verringert sich der Kommunikationsoverhead, da nur wenige Synchronisationsdaten ausgetauscht werden müssen.

Die Hardware/Software-Cosimulation stellt ein wichtiges Werkzeug dar, um die Entwicklung für zukünftige Hardware/Software-Systeme zu beschleunigen und zu vereinfachen. Eine ausführlichere Beschreibung der Cosimulationsansätze bieten [Vra98] und [Har04].

2.3.4 Hardware-Akzeleratoren

Der Nachteil einer HDL-Simulationssoftware ist die stark sinkende Simulationsgeschwindigkeit mit steigendem Detailgrad und steigender Designgröße. Um diesen Nachteil der Simulation zu vermeiden, werden Hardwareakzeleratoren eingesetzt. Sie emulieren das System auf FPGAs oder simulieren es auf Spezialprozessoren. Dadurch steigert sich die Geschwindigkeit der Simulation zum Teil erheblich. Nach Angaben einiger Hersteller solcher Systeme liegt der Beschleunigungsfaktor zwischen 10 und 10 000.

Ein Hardwareakzeleratorsystem besteht allgemein aus einer Spezialhardware und einem Softwarepaket. Die Spezialhardware hat die Aufgabe, die Simulation oder Emulation durchzuführen und die Simulationsergebnisse zu liefern.

Das Softwarepaket ist dafür zuständig, den ausführbaren Code bzw. die Konfigurationsdaten für den Hardwareakzelerator aus einer HDL-Beschreibung zu compilieren bzw. zu generieren. Außerdem hat es die Aufgabe, die Kommunikation zwischen dem Hostrechner und dem HW-Akzelerator zu managen. Dies beinhaltet den compilierten Code in den HW-Akzelerator zu laden, Befehle zur Steuerung und zum Debuggen an den HW-Akzelerator zu senden und die Simulationsdaten zu empfangen und aufzuzeichnen.

2.3.4.1 Tharas - Hammer 100

Den Hardwareakzelerator Hammer® 100 stellt die Firma Tharas Systems [Thaa] her. Es ist ein Simulationssystem, das aus Spezialprozessoren besteht. Die Prozessoren berechnen das Simulationsmodell, das aus einer Hardwarebeschreibungssprache kompiliert wurde.

Das System besteht aus bis zu 8 Boards mit je 16 Spezialprozessoren je nach Ausbaustufe. Die Prozessoren sind in 0,18 Mikrometertechnologie mit 5 Metallisierungsebenen gefertigt. Das System ist in einem CompactPCI-Gehäuse untergebracht, wie in Abbildung 2.13 zu sehen ist.

Die Prozessoren sind speziell dafür gebaut, um Hardwarebeschreibungskonstrukte zu beschleunigen. Alle Prozessoren können miteinander kommunizieren und in jedem Befehlszyklus gemeinsam Daten benutzen. Das Gehäuse

Hammer® 100 System Prozessorboard des Hammer® 100

Abbildung 2.13: Hammer® 100 System von Tharas, Quelle [Thab]

nutzt eine geschützte Backplane, um eine hohe Prozessorkonnektivität zu liefern. Die Busarchitektur wurde speziell entworfen, um den hohen Kommunikationsanforderungen eines hoch parallelen Rechensystems gerecht zu werden.

Als Hardwarebeschreibungssprachen werden der IEEE Verilog 1364-2001 und der IEEE VHDL 1076-2002 Standard unterstützt. Der Compiler analysiert eine Hardwarebeschreibungssprache auf syntaktische und semantische Korrektheit. Aus der Beschreibung generiert er optimalen, parallelen Code für die Spezialprozessoren. Der Compiler übersetzt bis zu 50 Millionen RTL-Gatteräquivalente je Stunde.

Das Hammer 100 System unterstützt verschiedene Hardwarebeschreibungskonstrukte, wie zum Beispiel Latches, mehrere Takte, Gated Clocks und Modelle mit Signalstärken. Alle synthetisierbaren Verilog und VHDL Konstrukte können beschleunigt werden. Darüber hinaus ist es auch möglich, Testbenches zu beschleunigen, sowie Modelle mit Verzögerungszeiten auszuführen.

Zur Software gehört auch der Runtime Manager. Er erleichtert die Laufzeitkontrolle der Hardware und die Kommunikation zwischen dem Akzelerator und dem Host-Software-Simulator. Die Hammer 100 Software erlaubt eine nahtlose Integration von existierenden Verifikationsumgebungen. Somit kann eine verteilte Ausführung der Simulation auf dem Hammer 100 und einem Software-HDL-Simulator ebenfalls vollzogen werden.

Der Debugger kommuniziert mit dem Hardware Trace-Buffer, um während des Debuggen des Designs einen schnellen Datengewinn aus dem Akzelerator zu gewährleisten. Das Debugging wird in Hardware ausgeführt und ist dadurch sehr schnell. Auch schrittweises Debugging des Designs ist möglich. Die Aufzeichnung der Simulationsdaten kann in einem Tracefile erfolgen.

Nach Angaben von Tharas Systems [Thab] beschleunigt das Hammer 100 System die Simulation um den Faktor 10 bis 10000 gegenüber dem schnellsten Software-HDL-Simulator. Die Ausbaustufen des Hammer 100 sind für 2, 4, 8, 16 und 32 Millionen Gatteräquivalente ausgelegt. Bei einer Zusammenschaltung mehrerer Hammer 100 simuliert das System bis zu 64, 96 oder 128 Millionen Gatteräquivalente. Der Preis des Simulationssystems liegt zwischen 2,65 und 7,5 US-Cent pro Gatterequivalentkapazität.

Mehr Informationen zu diesem System bietet die Homepage von Tharas Systems [Thaa] und die Broschüre zum Hammer 100 System [Thab].

2.3.4.2 Mentor Graphics - VStation

Das VStation-System von Mentor Graphics [Mena] gliedert sich in zwei Teile. Die Hardwareseite bildet die VStation™PRO und das Softwarepaket heißt VStation™TBX. Das System stellt eine komplette Umgebung bereit, die es ermöglicht, komplexe Designs von 1,6 bis 120 Millionen Gatter zu verifizieren.

Die VStationPRO ist eine FPGA-basierte Emulationsumgebung, die sich aus mehreren Boards zusammensetzt und dadurch leicht skalierbar ist. Ein Board besitzt die Ressources, um entweder 1,67, 3,33 oder 6,67 Millionen Gatter zu emulieren. Die Emulationsgeschwindigkeit des Systems reicht von 500 KHz bis 2 Mhz. Nach außen stellt das System je nach Ausbaustufe zwischen 512 und 4608 I/O-Verbindungen bereit.

Der VStationPRO VirtualLogic™ Compiler erlaubt ein direktes Mapping des Designs von der Register-Transfer-Ebene auf die FPGAs der VStationPRO ohne zuvor eine Synthese auszuführen. Dadurch werden Signalnamen und die Hierarchie des Designs beibehalten. Eine Simulatorähnliche Debugumgebung ermöglicht es, alle Signale des Designs zu überwachen. Auch Quellcodedebugging mit Haltepunkten und das Anzeigen von Waveforms wird unterstützt.

Die Software VStationTBX (TestBench-XPress) liefert zusätzlich noch eine Verifikationsumgebung für die Simulation und Emulation. Es werden die Sprachen SystemC und SystemVerilog unterstützt. VHDL kann nur als RT-Beschreibung verarbeitet werden. VStationTBX erlaubt eine schnelle Kommunikation zwischen der VStationPRO, einem Simulator und den Verifikationsprogrammen auf dem Hostrechner.

Abbildung 2.14: VStation™Pro System von Mentor Graphics, Quelle [Men03]

Weiterführende Informationen zum VStation-System finden sich auf der Homepage der VStation [Menb]. Produktbeschreibungen zur VStationPRO liefern [Menc] und [Men03]. Mehr Informationen über den VStationTBX enthalten [Mend] und [Men04].

2.3.4.3 Aptix - System Explorer™

Der Aptix® System Explorer™ (Abbildung 2.15) ist ein System für Emulation, Rapid Prototyping und HW/SW-Cosimulation. Es erlaubt die Emulation und das Debugging eines System-on-Chip Designs.

Die Grundlage des Systems besteht aus der Field Programmable Circuit Board® Architektur. Diese Architektur kombiniert eine Prototypingfläche mit einer programmierbaren Verbindungsstruktur, den FPIC®s (Field Programmable Interconnect Component®). FPICs sind elektronisch konfigurierbare bi-direktionale Verbindungsstrukturen und stellen die Verbindungen zwischen den Prototypingflächen her. Die Prototypingflächen können mit FPGAs und Systemkomponenten wie DSPs, RAM oder Prozessoren bestückt werden. Dadurch bildet der System Explorer eine rekonfigurierbare Prototypingplattform.

Zum System Explorer gehört auch ein Softwarepaket. Es besteht aus den Programmen Design Pilot™, Explorer 2000™ und Expeditor™. Der Design Pilot führt das logische Mapping durch und generiert die Netzlisten. Er bildet die entworfene Logik und Soft-IPs auf die Ziel-FPGAs ab. Außerdem führt er eine automatische hierarchische Blockgruppierung und Partitionierung des Entwurfs durch.

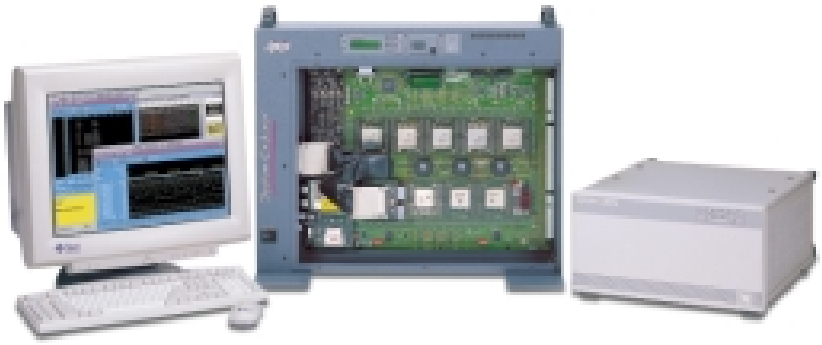


Abbildung 2.15: Aptix® System Explorer™, Quelle [Apt00]

Typ	**MP3CF**	**MP4CF**
Emulationskapazität in ASIC Gattern	2,5 Millionen	3 Millionen
Blockmemory	6 MBit	10 MBit
Prototypingfläche	1920 Pins	2880 Pins
max. Anzahl FPGAs	12	20
Anzahl Abgreifpunkte	1500	2000
Einsatzzweck	DSP-basierte Designs mit mäßigen Anforderungen an die Verbindungsstruktur zwischen den Komponenten	optimiert für Prototypen mit großen internen Bussen und hohen Anforderungen an die Verbindungsstruktur

Tabelle 2.2: Technische Daten des Aptix® System Explorer™

Der Explorer 2000™ führt das Mapping des partitionierten Designs auf die physischen Komponenten des Prototypingsystems aus. Er routet die einzelnen FPGAs und Komponenten untereinander durch die FPICs und konfiguriert anschließend den Prototypen. Nach der Konfiguration führt die Software eine Automatisierung des Hardware-Debugging durch. Der Explorer 2000 routed automatisch Probes[4] und konfiguriert den Agilent Logikanalysator zum Erfassen von Simulationsdaten.
Die Aufgabe des Expeditor™ besteht darin, eine Schnittstelle zu anderen Simulatoren und Testwerkzeugen herzustellen. Damit kann eine verteilte Simulation erfolgen.

Dem System Explorer liegt eine blockbasierte Verifikationsmethodik zu Grunde. Diese erlaubt die schnelle Erstellung von Prototypen, ein einfaches Debugging und eine rasche Implementierung von Designveränderungen. Der Prototyp wird Block für Block nach der Hierarchie des Designs aufgebaut. Da Logikänderungen meist auf eine FPGA begrenzt sind, ermöglicht das System nach einer Korrektur am Design, die einzelne Ersetzung der fehlerbehafteten FPGA und somit eine schnelle Änderung des Designs.

Im Entwurfsprozess kann mit Hilfe des System Explorers eine parallele Entwicklung von Software und Hardware erfolgen. Der Prototyp kann benutzt

[4]Probes sind Leitungen innerhalb der FPGA, die nach außen geführt werden, um die Signalwerte auf den Leitungen abgreifen zu können.

werden, um das System-on-Chip in seiner realen Systemumgebung zu emulieren. Es kann begonnen werden, Schnittstellen zu erproben und andere digitale oder analoge Subsysteme, während die SoC-Entwicklung noch voranschreitet.

Das frühe Vorhandensein eines Prototypen erlaubt es, dem Kunden schon früh im Entwurfsprozess eine Demonstration zu bieten und seine Resonanz in die weitere Entwicklung mit einfließen zu lassen.

Der System Explorer wird in zwei Ausführungen hergestellt. Die Bezeichnungen sind MP3CF und MP4CF. Tabelle 2.2 stellt die technischen Daten der beiden Systeme gegenüber. Weiterführende Informationen zum Aptix System Explorer sind auf der Homepage des Unternehmens [Apt] sowie in der Broschüre zum System Explorer [Apt00] enthalten.

2.4 Fazit

Die vorangehenden Punkte beschäftigten sich mit SystemC als neue Entwurfssprache, mit dem Nutzen von Intellectual Properties und der Bedeutung der Simulation im Hardwareentwurf. Dazu wurden einige Simulationswerkzeuge und -methoden vorgestellt.

SystemC bringt die Voraussetzungen mit, um den Entwurfsprozess von Hardwaresystemen und Hardware/Software-Systemen weiter zu beschleunigen. Ein Nachteil ist noch die mangelnde Unterstützung von sprachfremden IPs über der Register-Transfer-Ebene.

Intellectual Properties bilden zukünftig ein Schlüsselkonzept bei der Entwicklung von SoC-Designs. Durch ihren Einsatz können die Entwicklungszeit und -kosten weiter gesenkt werden. Es existieren bereits Lösungen zum einfachen Handel mit IPs. Jedoch behindern teilweise rechtliche Unklarheiten im Moment noch den IP-Handel. Aber in absehbarer Zukunft wird sich dieses Konzept durchsetzen.

Die Simulation ist ein wichtiges Werkzeug im Entwicklungsprozess. Es existieren verscheidene Simulationswerkzeuge, die verschiedene Ansätze verfolgen. Ein Ansatz besteht in einer reinen Simulationssoftware wie Modelsim und SystemC. Ansätze zur Cosimulation verfolgen das Ziel, verschiedene Simulatoren miteinander zu koppeln, um die Simulation von Hardware und Software parallel ausführen zu können. Hardware-Akzeleratoren sollen durch Spezialhardware die Simulation großer Designs beschleunigen. Doch sie bieten auch

die Möglichkeit der Kopplung mit Softwaresimulationslösungen. Die vorgestellten Hardware-Akzeleratoren erlauben teilweise die Nutzung von SystemC. Die Unterstützung beginnt jedoch erst auf Register-Transfer-Ebene. In höheren Abstraktionsebenen fehlt diese Unterstützung noch.

3 Die Methode der Adaptierung

Diese Kapitel beschreibt die Adaptierung eines Hardware-Akzelerators an die SystemC-Verhaltenssimulation. Der erste Teil beinhaltet die Herleitung des Hardware/Software-Interfaces[5] und die Analyse seiner Anforderungen. Eine Realisierung des HW/SW-Interfaces durch einen Interfaceblock wird im zweiten Abschnitt vorgestellt. Anschließend werden Details zur technischen Umsetzung dargestellt. Die Einsatzmöglichkeiten des HW/SW-Interfaces zeigt der letzte Abschnitt dieses Kapitels auf.

3.1 Das Hardware/Software-Interface

Das Ziel dieser Arbeit ist es, ein Verfahren zu entwicklen, welches die Simulation von VHDL-IPs in SystemC, auf einer höheren Abstraktionsebene als der RT-Ebene, erlaubt. Bisher ist die Einbindung dieser IPs für den Entwurf und somit auch für die Simulation erst auf Netzlistenebene möglich. In früheren Entwurfsstadien ist eine Simulation des Gesamtsystems mit diesen IPs jedoch schon sehr hilfreich, um Tests mit dem Systemverhalten durchführen zu können und frühzeitig Fehler zu entdecken.

Zur Simulation einer VHDL-IP mit SystemC existieren zwei grundsätzliche Möglichkeiten. Die erste Möglichkeit ist die Konvertierung der IP in ein SystemC-Modul (Abbildung 3.1). Die Konvertierung muss dabei von Hand erfolgen. Dieser Schritt verursacht zusätzliche Arbeit. Das erstellte SystemC-Modul muss gegen die VHDL-IP verifiziert werden, um sicherzustellen, dass ihr Verhalten übereinstimmt. Dieser Prozess bindet Ressourcen und kostet meist viel Zeit und Geld.

Einen Ansatz aus der Cosimulation vorfolgt die zweite Variante. Dabei findet die Simulation der VHDL-IP in einem VHDL-Simulator und die des SystemC-Modells in einem SystemC-Simulator statt. Die Kopplung der beiden Simulatoren realisiert eine Software/Software-Schnittstelle (Abbildung 3.2). Anforderungen, die die SW/SW-Schnittstelle erfüllen muss, sind der

[5] Für Interface wird in den folgenden Abschnitten der deutsche Begriff Schnittstelle verwendet

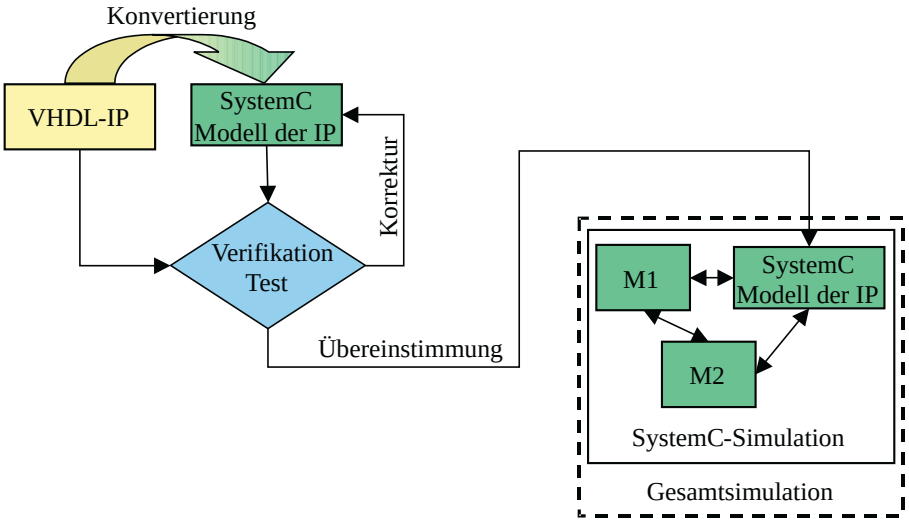


Abbildung 3.1: Adaptierung durch Konvertierung

Austausch von Simulationsdaten und Kontrollsignalen sowie die Synchronisation zwischen den Simulatoren, damit zum richtigen Zeitpunkt die korrekten Simulationsdaten vorhanden sind.

Allerdings kostet die Ausführung von zwei Simulatoren auf einem Rechner viel Rechenleistung. Die Folge ist eine geringere Simulationsgeschwindigkeit. Da kontinuierlich Daten zwischen den Simulatoren ausgetauscht werden müssen, spielt nicht nur die Ausführungszeit der beiden Simulatoren eine große Rolle sondern auch die Zeit, die die Kommunikation zwischen diesen beansprucht.
Der Vorteil dieser Variante liegt darin, dass die IP nicht von Hand konver-

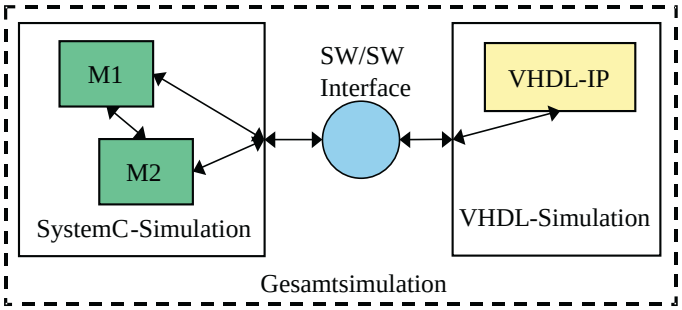


Abbildung 3.2: Adaptierung eines Simulators

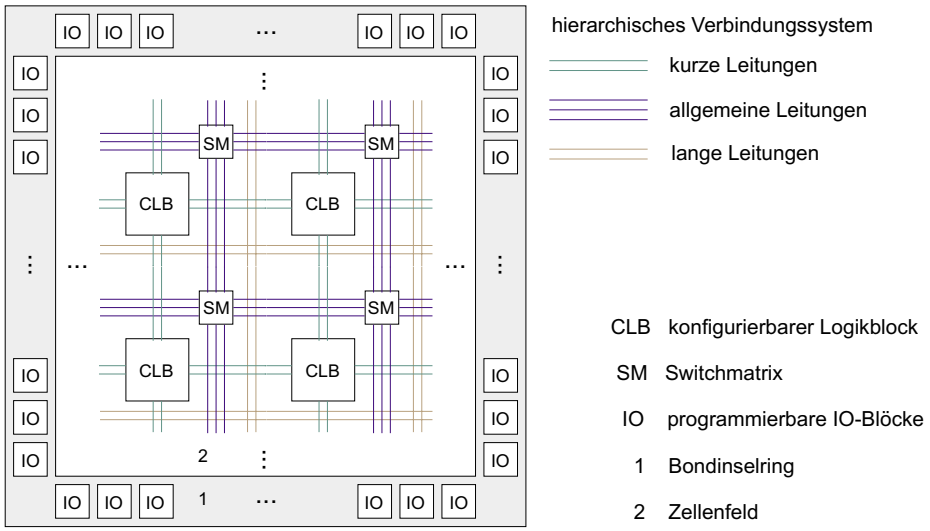


Abbildung 3.3: Schematischer Aufbau eines SRAM-basierten FPGA

tiert werden muss. Dieses Kriterium ist wichtig, weil dadurch die Zeit für Tests und Verifizierung eingespart wird. Um die Simulation zu beschleunigen, wird, zusätzlich zum Cosimulationsansatz, der Ansatz von Hardware-Akzeleratoren eingebracht. Das heißt, dass die IP in Hardware emuliert wird, was eine schnellere Ausführung der IP erlaubt, und mit dem SystemC-Simulator gekoppelt wird.

Der Einsatz von kommerziellen Hardware-Akzeleratorsystemen ist durch ihre hohen Kosten hier nicht praktikabel. Eine Lösung für dieses Problem wäre eine günstigere handelsübliche Hardware, die schnell konfiguriert werden kann. Mit dieser Hardware sollte der Anwender auch in der Lage sein, größere Schaltungen schnell zu realisieren.

Diese Voraussetzungen erfüllen SRAM-basierte Field Programable Gate Arrays (FPGA). FPGAs sind programmierbare Hardwarebausteine und gehören zur Gruppe der anwenderprogrammierbaren Schaltungen. Den grundsätzlichen Aufbau von FPGAs stellt Abbildung 3.3 dar. Sie bestehen im wesentlichen aus konfigurierbaren Logikblöcken, konfigurierbaren IO-Blöcken und einer konfigurierbaren hierarchischen Verbindungstruktur. Mit Hilfe von FPGAs können kombinatorische und sequentielle digitale Schaltungen realisiert werden. Mehr Informationen zu FPGAs bieten [Wan98] und die Homepages von FPGA-Herstellern [Alt, Atm, Lat, Xilc].

Um jedoch die FPGA nutzen zu können, ist die Synthesefähigkeit der IP

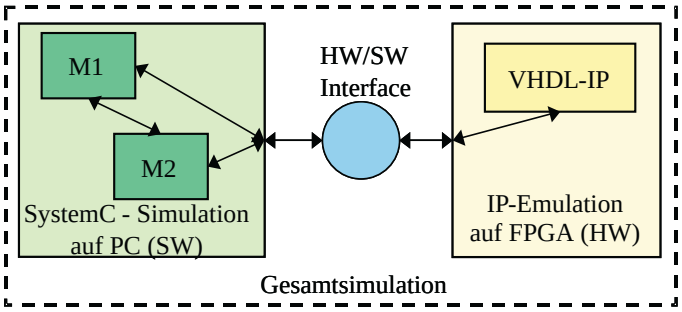


Abbildung 3.4: Allgemeines Hardware/Software-Interface

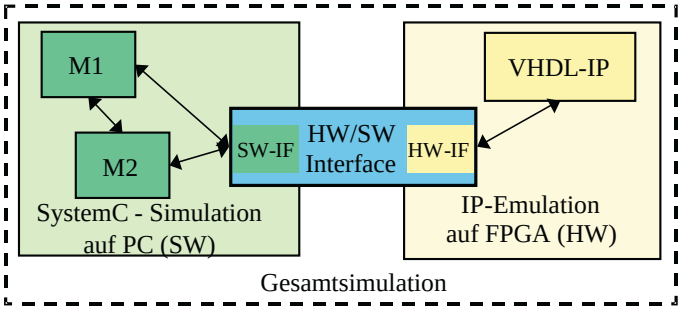


Abbildung 3.5: Genaueres Modell des Hardware/Software-Interface

unabdingbar. Durch diese Einschränkung entfällt die Verwendung von Hard-IPs (vgl. Abschnitt 2.2). Sie sind bereits synthetisiert und mit Layout und Timing versehen. Soft-IPs dagegen sind synthetisierbar.

Durch die Nutzung der FPGA muss im Folgenden auf der Hardwareseite von Emulation statt von Simulation gesprochen werden. Emulation ist das funktionelle Nachbilden eines Systems durch ein anderes. Die Implementierung der IP ist nun kein Modell mehr, wie für eine Simulation, sondern ist eine Nachbildung der realen digitalen Schaltung mit Hilfe eines FPGAs.

Eine **Hardware/Software-Schnittstelle** tritt nun an die Stelle der Software/Software-Schnittstelle (Abbildung 3.4), um die SystemC-Simulation mit der IP-Emulation zu koppeln. Doch wie soll die Kopplung erfolgen und welche Anforderungen werden damit an die Hardware/Software-Schnittstelle gestellt?

Die Kopplung der SystemC-Simulation erfolgt über eine SW/SW-Schnittstelle. Auf der Hardwareseite befindet sich eine HW/HW-Schnittstelle, um die Ver-

bindung mit der IP herzustellen. Abbildung 3.5 stellt diese genauere Betrachtung der HW/SW-Schnittstelle dar. Dies ermöglicht es, die eigentliche Grenze zwischen Hardware und Software für die SystemC-Simulation sowie für die IP-Emulation unsichtbar zu machen. Welchen Anforderungen die HW/SW-Schnittstelle genügen muss, um diese Art der Verbindung herzustellen, wird im folgenden Abschnitt analysiert.

3.1.1 Anforderungsanalyse für das HW/SW-Interface

Die Anforderungen an die HW/SW-Schnittstelle sind zunächst die gleichen wie bei einer SW/SW-Schnittstelle:

- Austausch von Simulationsdaten und Kontrollsignalen ermöglichen
- Synchronisation zwischen Simulation und Emulation gestatten
- Nutzung der IP ohne diese zu verändern

Zunächst soll die Anforderung des Datenaustausch näher betrachtet werden. Er geschiet über zwei Schnittstellen. Zum Einen über die Schnittstelle von IP und HW/SW-Schnittstelle. Diese HW/HW-Schnittstelle hängt allein von der IP ab. Da keine Änderungen an der IP vorgenommen werden sollen, bilden die Ein- und Ausgangssignale der IP die einzige Kommunikationsmöglichkeit. Das hat zur Folge, dass die HW/HW-Schnittstelle genau durch die Eingänge und Ausgänge der IP realisiert wird.

Die andere Schnittstelle befindet sich auf der Softwareseite. Dort muss die SystemC-Simulation mit der HW/SW-Schnittstelle verbunden werden. Prinzipiell ist es bei dieser SW/SW-Schnittstelle möglich, sie um Kontrollsignale zu erweitern. Das Minimum bilden aber auch hier die Ein- und Ausgänge der IP, da sonst Simulationsdaten verloren gehen würden. Der Nachteil bei einer Einführung zusätzlicher Signale liegt darin, dass der Simulationskern verändert werden müsste, um die Signale zu erzeugen. Eine bessere Variante, die den Simulationskern unbeeinflusst lässt, besteht darin, das HW/SW-Interface für den Simulationskern unsichtbar zu halten und nur die Ein- und Ausgänge der IP für die Simulation bereitzustellen. Daraus ergibt sich eine weitere Anforderung an die HW/SW-Schnittstelle. Sie muss intern Kontrollsignale erzeugen und auswerten können, um den Transport der Simulationsdaten sicherzustellen.

Ein weiteres Problem, welches sich aus dem Datentransport durch die HW/SW-Schnittstelle ergibt, ist die Überwindung der Grenze zwischen Hardware und Software. Dies kann durch Standardschnittstellen am PC erfolgen. Dabei ist es notwendig, dass diese Schnittstelle die Kommunikation vom PC zur FPGA und umgekehrt beherrscht, da die Simulationsdaten in beide Richtungen ausgetauscht werden müssen. Standardschnittstellen besitzen feste Parameter hinsichtlich der Daten, die sie in einem Kommunikationszyklus übermitteln können. Da unterschiedliche IPs unterschiedliche Schnittstellenbreiten besitzen, kann keine Aussage getroffen werden, ob die gesamten Simulationsdaten in einem Kommunikationszyklus übertragbar sind. Deshalb sollte die HW/SW-Schnittstelle in der Lage sein, die Simulationsdaten intern zu serialisieren, sie vollständig und ohne Verlust über die HW/SW-Grenze zu transportieren und auf der anderen Seite wieder richtig zu parallelisieren. Das heißt, innerhalb der HW/SW-Schnittstelle muss eine Datentransformation gestattet sein.

Nachdem die Anforderungen an den Datenaustausch herausgearbeitet sind, stellt sich nun das Problem der Synchronisation zwischen der SystemC-Simulation und der IP-Emulation. Die SystemC-Simulation ist zyklusbasiert (vgl. Abschnitt 2.3.1). Hier dient das Taktsignal als Auslösesignal für die Neuberechnung der Signale des im Test befindlichen Designs. Da keine Änderungen am Simulationskern vorgenommen werden sollen, findet dieses Auslösesignal auch bei der Emulation Verwendung. Das heißt, dass die IP über das Taktsignal der SystemC-Simulation gesteuert wird. Eine Master-Slave-Simulation ist die Folge, bei der die SystemC-Simulation den Master darstellt, da sie das steuernde Signal generiert. Die Emulation dagegen nimmt, durch die Reaktion auf das Mastersignal, die Rolle des Slave an.

Jedoch wirft die Synchronisation über das Taktsignal wiederum ein Problem auf. Es besteht darin, dass das Taktsignal durch die HW/SW-Schnittstelle und über die darin enthaltene HW/SW-Grenze transportiert werden muss. Es muss garantiert werden, dass alle Eingangssignale korrekt anliegen, bevor das Taktsignal angelegt wird. Durch eine reine Übertragung des Taktsignals ist dies nur schwer möglich.

Eine Lösung besteht darin, das Taktsignal auf der Softwareseite zu analysieren. Bei der aktiven Taktflanke wird dann zuerst die Übertragung der Simulationsdaten ausgeführt, dann ein Kontrollsignal zur Takterzeugung auf der Hardwareseite generiert. Die HW/SW-Schnittstelle generiert für die Emulation nun genau einen Takt und gibt eine Rückmeldung an den Softwareteil nach dessen Beendigung. Am Ende werden die Ausgangsdaten der Emulation zur Simulation übertragen, welche weiter fortschreiten kann. Abbildung

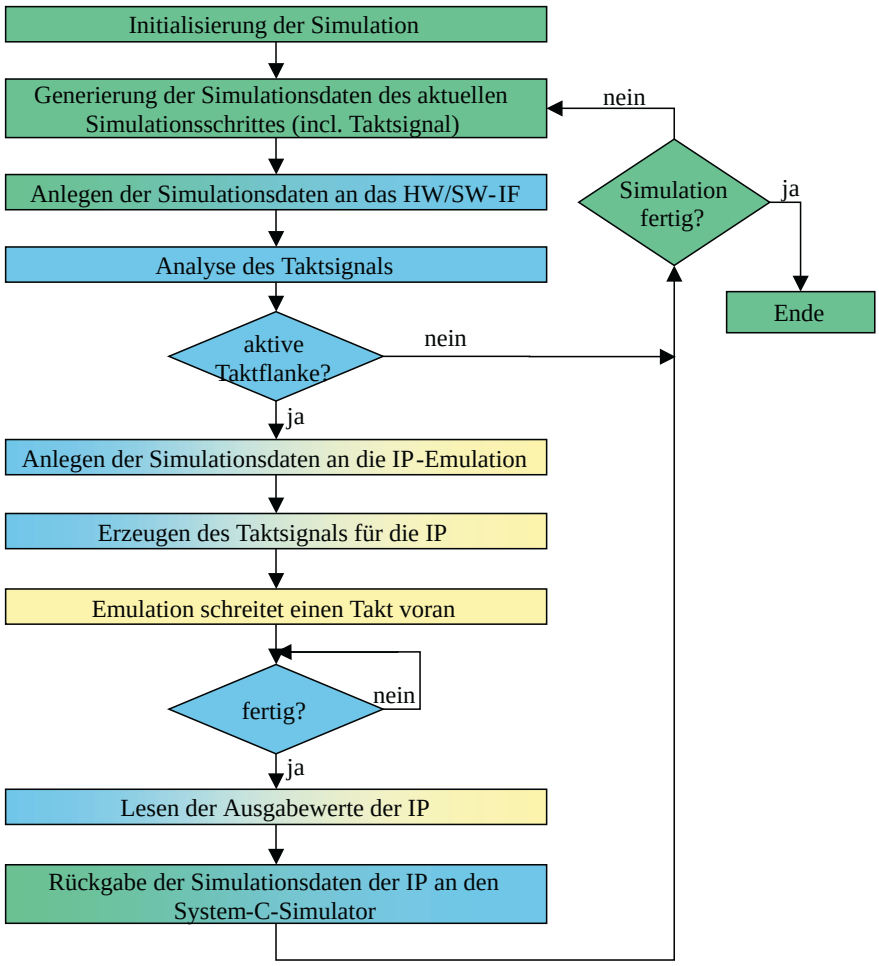


Abbildung 3.6: Ablauf der Synchronisation durch das HW/SW-Interface

3.6 stellt diese Art der Synchronisation im Rahmen der Simulation dar. Zur Umsetzung dieser Lösung benötigt die HW/SW-Schnittstelle einen Mechanismus, der es erlaubt, Eingangssignale zu analysieren und interne Kontrollsignale sowie Ausgangssignale zu generieren. Es muss also möglich sein, in der HW/SW-Schnittstelle eine Steuerung zu integrieren.

Die letze Anforderung stellt die Nutzung der IP dar, ohne sie zu verändern. Diese Anforderung wurde durch die vorangegangen Maßnahmen erfüllt. Eine Änderung der Schnittstelle oder der Funktionalität der IP fand nicht statt.

Zusammenfassend sollen hier noch einmal die Anforderungen an die HW/SW-Schnittstelle dargestellt werden. Die folgenden Anforderungen ergab die durchgeführte Analyse. Die HW/SW-Schnittstelle muss

1. zwei Komponenten kommunizieren lassen, ohne dass an den Komponenten Änderungen durchgeführt werden müssen.
2. den bidirektionalen Datentransport zwischen den Komponenten sicherstellen.
3. die HW/SW-Grenze intern überwinden.
4. die Möglichkeit bieten, Daten intern zu transformieren.
5. die Integration einer Steuerung erlauben.

3.2 Der Interfaceblock als HW/SW-Interface

Zur Realisierung der HW/SW-Schnittstelle benötigt man ein Schnittstellenkonstrukt, dass die im vorangegangen Abschnitt herausgestellten Anforderungen im Wesentlichen erfüllt. Falls das Konstrukt nicht allen Anforderungen genügt, sollte es so flexibel und erweiterbar sein, dass es sich auf die neuen Bedingungen leicht anpassen lässt. Ein Schnittstellenkonstrukt, dass einen Großteil der gestellten Anforderungen erfüllen kann, ist der Interfaceblock, der im folgenden Abschnitt kurz vorgestellt wird.

3.2.1 Das Modell des Interfaceblocks

Ein Interfaceblock (IFB) dient als Schnittstelle zwischen inkompatiblen Kommunikationskomponenten. Diese Kommunikationskomponenten können komplexe Strukturen (z.B. Bussysteme) oder funktionale Komponenten (z.B. Algorithmen) sein. Diese werden in der IFB-Terminologie entsprechend als Medium beziehungsweise Task bezeichnet. In einem Interfaceblock können Daten zwischen Sender und Empfänger transformiert werden. Dadurch ist ein Einsatz als Adapter zwischen physisch inkompatiblen bzw. semantisch inkompatiblen Kommunikationskomponenten möglich.

Der Interfaceblock ist modular aufgebaut. Er besteht, wie Bild 3.7 zeigt, aus einer Controlunit (CU), zwei Protokollhandlern (PH) und einem Sequenzhandler (SH). Jeder der Handler besitzt zusätzlich mindestens einen Modus, der die eigentliche Funktionalität implementiert.

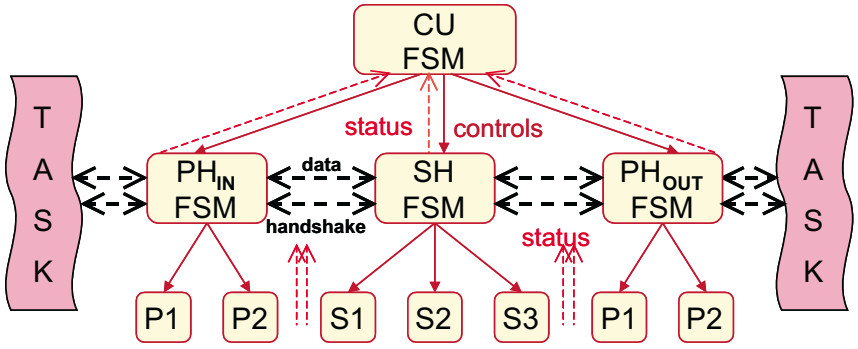


Abbildung 3.7: Makrostruktur des Interfaceblock

Die Controlunit übernimmt die Steuerungsaufgaben innerhalb des Interfaceblocks. Sie koordiniert und kontrolliert die Funktion der Protokollhandler und des Sequenzhandlers. Außerdem bestimmt sie den jeweils aktiven Modus eines Handlers. Sie steuert die Handler über Kontrollsignale und erhält deren aktuellen Zustand durch Statussignale.

Die Protokollhandler sind die externen Schnittstellen des Interfaceblocks. Die eigentliche Kommunikation mit den verbundenen Tasks und Medien erfolgt hier. Basierend auf seinen Modi verarbeitet der Protokollhandler die Protokolle der angeschlossenen Kommunikationskomponenten. Die Kommunikation beinhaltet sowohl das Senden als auch das Empfangen von Daten.

Das Bindeglied zwischen den Protokollhandlern stellt der Sequenzhandler dar. In ihm werden die eingehenden Nutzdaten konform zum ausgehenden Protokoll transformiert. Die Transformation folgt einer Abbildungsvorschrift, die die Sequenzhandlermodi implementieren. Die Transformation ist nicht nur auf strukturelle Änderungen der Daten beschränkt, sondern ermöglicht auch eine semantische Änderung.

Weitere Informationen zur Grundidee des Interfaceblocks, zur Beschreibungsform eines Interfaceblocks durch das Interface Synthese (IFS) Format, zur technischen Umsetzung und zum Stand der aktuellen Entwicklungen sind in [Ihm01], [HVI01], [IVH02a], [IVH02b], [IBJK+03], [IVH03], [Fic03], [Fla03] und [IH04] enthalten.

3.2.2 Analyse der Leistung des Interfaceblocks

Um den Interfaceblock als HW/SW-Schnittstelle einsetzen zu können, ist zunächst der Nachweis erforderlich, dass der Interfaceblock auch die gestellten Anforderungen (vgl. Abschnitt 3.1.1) theoretisch erfüllen kann.

Die erste Anforderung bestand darin, zwei Komponenten kommunizieren zu lassen, ohne an den Komponenten Änderungen durchzuführen. Der Interfaceblock erfüllt die Funktion eines Adapters. Ein Adapter ist nach [Wika] ein Gerät, dass zur Verbindung verschiedener mechanischer oder elektrischer Geräte dient, deren Anschlüsse wegen unterschiedlicher Formate oder Normen nicht zueinander passen. In dem vorliegenden Fall sind die zu verbindenden „Geräte“ die SystemC-Simulation und die IP-Emulation. Die Anschlüsse stellen die Schnittstellen dar, die in Software bzw. in Hardware vorliegen. Dadurch ist eine direkte Verbindung nicht möglich. Zur Verbindung wird die Eigenschaft des IFBs als Adapter genutzt. Somit ist die Verbindung jedenfalls theoretisch möglich.

Die nächste Anforderung umfasst den bidirektionalen Datentransport zwischen den Komponenten. Der IFB dient nach dem Modell als Schnittstelle zwischen Kommunikationskomponenten. Eine Schnittstelle (Interface) ist ein Teil eines Systems, das dem Austausch von Informationen, Energie oder Materie mit anderen Systemen dient [Wikb]. Nach der Definition beherrscht eine Schnittstelle den Austausch von Informationen, die im vorliegenden Fall die Simulationsdaten sind. Da der IFB eine Schnittstelle ist und Daten senden und empfangen kann, erfüllt er auch diese gestellte Anforderung.

Die Integration einer Steuerung in den IFB erfolgt durch die Controlunit. Sie erfüllt innerhalb des IFBs Steuerungs- und Kontrollaufgaben. Sie kontrolliert die Handler, indem sie Steuerungssignale an diese übermittelt und Statussignale der Handler auswertet. Somit ist sie der zentrale Punkt im IFB, von dem aus eine Steuerung arbeiten kann.

Weiterhin soll die HW/SW-Schnittstelle eine interne Datentransformation zulassen. Das kann in den Modi der Handler geschehen. Sie implementieren die Funktionalität zur Datenübertragung. Die Modi können also auch mit Algorithmen versehen sein, die Daten transformieren.

Die transformierten Daten müssen nun noch durch den IFB transportiert werden können. Innerhalb des IFBs findet Kommunikation zwischen der Controlunit und Protokollhandlern bzw. Sequenzhandler statt. Außerdem kommunizieren die Protokollhandler mit dem Sequenzhandler, um den Datentransport durch den IFB zu realisieren. Kommunikation, also der Austausch

von Daten, läuft nach einem Protokoll ab, damit der Sender und Empfänger sich gegenseitig verstehen. Da das Modell des IFBs das interne Kommunikationsprotokoll nicht festlegt, ist die Verwendung eines Protokolls möglich, das den Transport der transformierten Daten ermöglicht.

Zur Überwindung der HW/SW-Grenze innerhalb des IFB soll die freie Auswahl eines Protokolls dienen. Ein Protokoll wird über eine Schnittstelle übertragen. Die Benutzung einer Schnittstelle, die Hardware und Software physisch verbindet, bringt die Lösung dieses Problems. Der IFB ist aber bisher nur für reine Hardwareanwendungen konzipiert. Aus diesem Grund muss geprüft werden, ob sich das Modell dahingehend erweitern lässt, dass es die Einbindung einer physischen HW/SW-Schnittstelle gestattet.

3.2.3 Erweiterung des Interfaceblocks

Zur Feststellung der Erweiterbarkeit des IFBs wird der IFB schrittweise zwischen der SystemC-Simulation und der IP-Emulation aufgebaut. Die Ausgangssituation stellt die Verbindung eines Softwaretasks (SystemC-Simulation) mit einem Hardwaretask (IP-Emulation) dar. Die Verbindung soll mit Hilfe eines Interfaceblocks erfolgen. Abbildung 3.8 veranschaulicht die Ausgangssituation.

Zunächst richtet sich die Betrachtung auf den Hardwaretask, da dem Interfaceblock eine Hardwareanwendung zugrunde liegt. Die Verbindung zwischen dem Interfacebock und der Intellectual Property wird durch einen Protokollhandler bewerkstelligt (Abbildung 3.9). Er wird im Folgenden als $PH_{HW\text{-}out}$ bezeichnet, da er den hardwareseitigen Ausgang des Interfaceblock bildet.

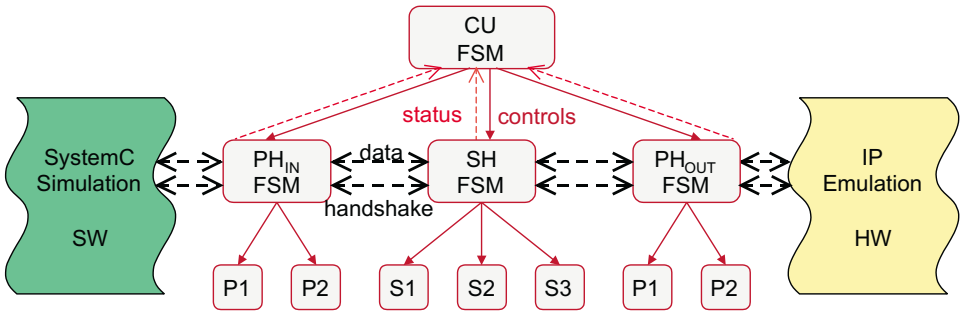


Abbildung 3.8: Verbindung eines SW-Task und eines HW-Task durch einen Interfaceblock

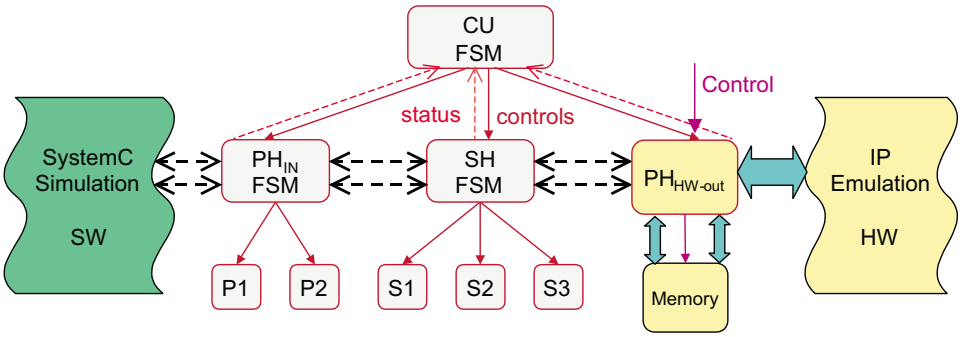


Abbildung 3.9: Die Hardwareschnittstelle des Interfaceblocks, der Protokollhandler$_{\text{HW-out}}$

Die Funktion, die die Schnittstelle $\text{PH}_{\text{HW-out}} \leftrightarrow$ IP-Emulation erfüllen muss, ist das Anlegen der Triggerdaten an die IP und das kontrollierte Abgreifen der Tracedaten von der IP während eines Emulationszyklus. Zur Sicherstellung der Datenkonsistenz, also dass die Daten nur aus dem gerade ablaufenden Emulationszyklus stammen, muss dieser Vorgang über Kontrollsignale gesteuert werden. Die Daten, die über die Schnittstelle transportiert werden, müssen bis zu ihrer Verarbeitung verfügbar sein. Aus diesem Grund erfolgt eine Zwischenspeicherung der Daten im IFB. Die Speicherung findet im Modus des $\text{PH}_{\text{HW-out}}$ statt, da die Modi der Handler diese Funktionalität bereitstellen können.

Die Verbindung von SystemC-Simulation und Interfaceblock realisiert wiederum ein Protokollhandler. Dieser Protkollhandler hat die Aufgabe, die Kommunikation mit der SystemC-Simulation auf Softwareseite sicherzustellen. Im Folgenden wird er als PH_{SW} bezeichnet (Abbildung 3.10). Neu im IFB-Modell ist, dass der Protokollhandler eine SW-Schnittstelle darstellt. An der Aufgabe der Schnittstelle ändert sich nichts. Die Funktionalität wird in Software jedoch durch Funktionen bestimmt, die Algorithmen implementieren und nicht durch kombinatorische Logik und endliche Automaten. Die Datenübergabe findet bei Funktionen über Parameter und nicht durch das Anlegen eines elektrischen Pegels statt. Das stellt aber kein Hinderniss dar. Eine Funktion kann ebenso Parameter auswerten und auch Rückgabewerte erzeugen. Außerdem erlaubt Software die Implementierung von umfangreichen Funktionen. Es spricht also nichts dagegen, dem Interfaceblock eine SW-Schnittstelle in Form eines Softwareprotokollhandlers zu geben. Die Modi liegen ebenfalls in Software und sind durch Funktionen implementiert, die vom PH_{SW} aufgerufen werden.

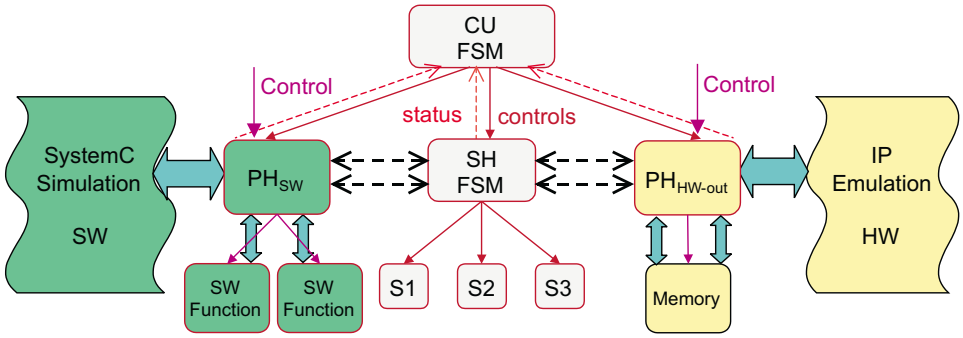


Abbildung 3.10: Die Softwareschnittstelle des Interafceblocks, der Protokollhandler$_{SW}$

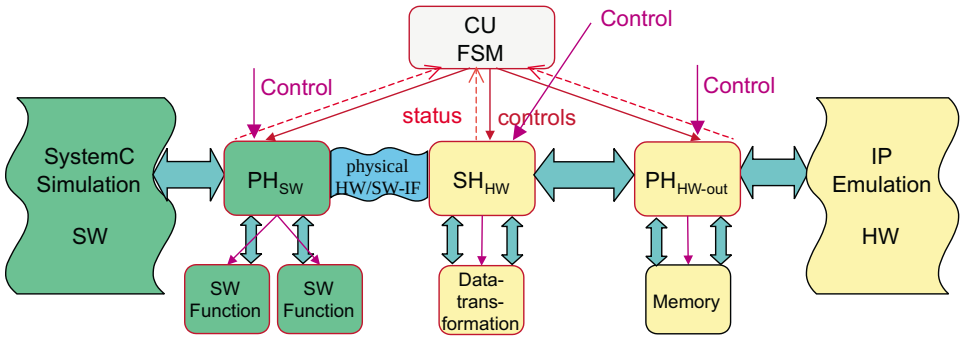


Abbildung 3.11: Die physische HW/SW-Schnittstelle im Interfaceblock

Nun stellt sich die Frage, an welche Stelle die physische HW/SW-Schnittstelle platziert wird. Es gibt dabei die Möglichkeiten, sie zwischen PH$_{SW}$ und SH oder SH und PH$_{HW-out}$ anzusiedeln. Um einen Großteil des IFB-Modells beizubehalten, wird die physische HW/SW-Schnittstelle als Schnittstelle von PH$_{SW}$ und SH genutzt (Abbildung 3.11). Da der Sequenzhandler in Hardware liegt, wird er weiterhin als SH$_{HW}$ bezeichnet. In Abschnitt 3.1.1 wurde bereits festgestellt, dass wahrscheinlich eine Transformation der Daten, die über die physische HW/SW-Schnittstelle transportiert werden, notwendig ist. Die Aufgabe des Sequenzhandlermodus besteht nun darin, die empfangenen und die zu sendenden Daten vom bzw. zum PH$_{SW}$ in ein protokollkonformes Format umzuwandeln.

Zur Kontrolle der Abläufe innerhalb des IFB kommt die Controlunit zum Einsatz. Sie hat die Aufgabe, die Handler zu steuern und zu überwachen. Bei der Verwendung des IFBs als HW/SW-Schnittstelle, muss der PH$_{HW-out}$,

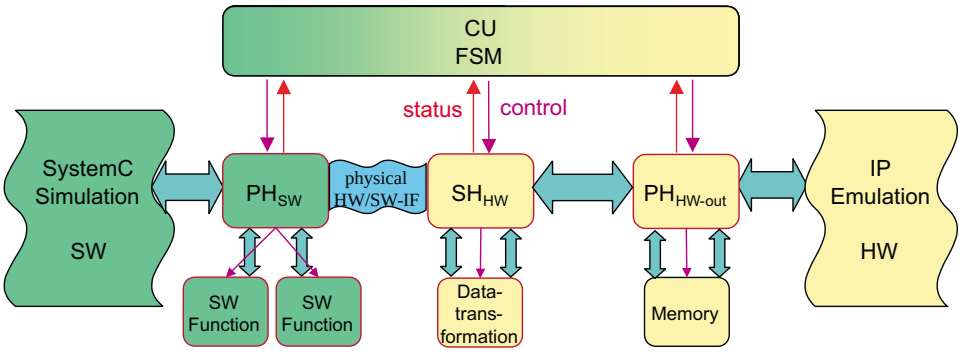


Abbildung 3.12: Controlunit zur Steuerung von Hardware und Software

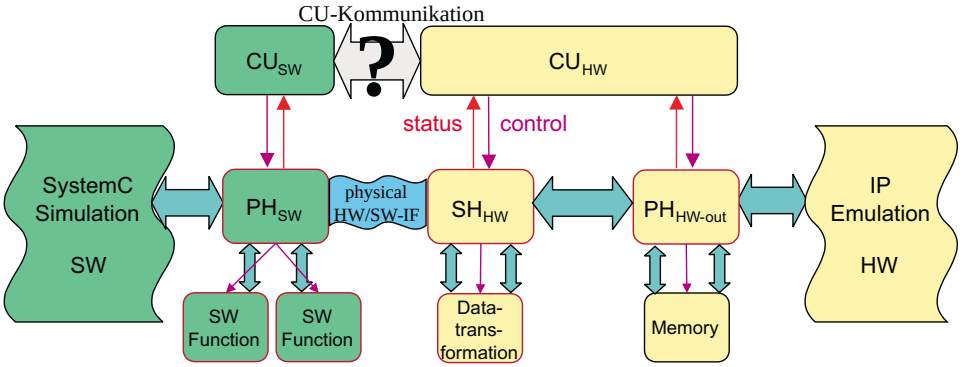


Abbildung 3.13: Teilung der Controlunit in Hardware- und Softwareteil

der SH_{HW} aber auch der PH_{SW} von der Controlunit gesteuert werden. Das erfordert, dass die Controlunit Software als auch Hardware ansteuern kann (Abbildung 3.12).

Die physische Trennung von Hardware- und Softwareseite des IFBs, gestaltet die Steuerung durch eine einzige Controlunit schwierig. Aus diesem Grund erfolgt eine Teilung der Controlunit in einen Softwareteil (CU_{SW}) und einen Hardwareteil (CU_{HW}), wie Abbildung 3.13 zeigt. Der Vorteil liegt nun darin, dass die jeweilige Controlunit ihren Teil direkt kontrollieren kann, ohne Steuersignale über die HW/SW-Grenze zu schicken und damit größere Verzögerungszeiten im Kontrollfluss zu verursachen.

Ein Problem, welches durch die Teilung auftritt, ist die Kommunikation zwischen den Controlunits. Zur geregelten Steuerung des IFBs und zur Synchronisation zwischen Hardware- und Softwareteil ist eine Kommunikation

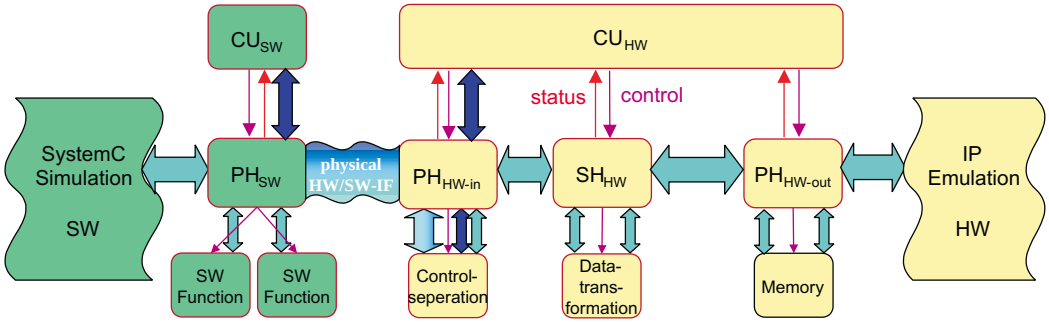


Abbildung 3.14: Controlunit in Software und Hardwareteil geteilt

zwischen CU_{SW} und CU_{HW} notwendig. Als Hinderniss ist dabei die HW/SW-Grenze zu überwinden. Dies kann auf zwei Arten erfolgen. Zum Einen über eine zweite physische HW/SW-Schnittstelle, die allein für den Transport und Austausch von Kontroll- und Statussignalen zwischen den Controlunits zuständig ist. Die Lösung bedeutet einen höhere Aufwand, in Bezug auf die Bereitstellung von Hardware. Außerdem muss eine zweite Schnittstelle in Software als auch in Hardware implementiert werden.

Die zweite Möglichkeit besteht darin, die Kommunikation über die physische HW/SW-Schnittstelle zu bewältigen, über die auch die Simulationsdaten ausgetauscht werden. Der Vorteil dieser Variante ist die Nutzung der vorhandenen physischen Schnittstelle ohne zusätzlichen Hardwareaufwand. Allerdings benötigt man jetzt ein zusätzliches Protokoll, dass über dem Protokoll der HW/SW-Schnittstelle angesiedelt ist, um eine Unterscheidung zwischen Simulationsdaten und Controlunitdaten zu ermöglichen. Dies bringt die Einführung eines weiteren Protokollhandlers auf der Hardwareseite des IFBs mit sich (Abbildung 3.14). Er wird im Folgenden als $PH_{HW\text{-}in}$ bezeichnet. Diese Maßnahme ist notwendig, da der Sequenzhandler laut Definition nur für die Änderung von Daten zuständig ist. Die Verarbeitung von Protokollen ist dem Protokollhandler vorbehalten.

Die CU-Kommunikation könnte also wie folgt ablaufen. Die CU_{SW} will ein Kontrollsignal an die CU_{HW} schicken. Dazu leitet sie das Signal an den PH_{SW} weiter, welcher es in das Protokoll zur Datenunterscheidung umwandelt und es mittels des normalen HW/SW-Protokoll an den Hardwareteil des IFBs sendet. Die Stelle im Hardwareteil, an dem die Daten empfangen werden, ist der $PH_{HW\text{-}in}$. Er prüft, ob die empfangenen Daten für die Simulation oder für die Controlunit bestimmt sind. Die Kontrollsignale werden an die CU_{HW} weitergeleitet. Sie kann die Signale auswerten und eine Antwort an die CU_{SW}

über den gleichen Weg zurücksenden.

Als Ergebnis liegt jetzt ein erweiterter Interfaceblock vor. Das heißt, dass das Modell des Interfaceblocks dahingehend erweiterbar ist, um damit auch eine HW/SW-Schnittstelle zu realisieren. Das Modell des Interfaceblocks ist also nicht auf reine Hardwareanwendungen begrenzt. Um dieses Ergebnis weiter zu untermauern, folgt den bisher theoretischen Betrachtungen, eine mögliche technischen Umsetzung dieses Modells am Beispiel des Simulationsinterfaceblocks.

3.3 Technische Umsetzung der Adaptierung durch einen Simulationsinterfaceblock

Dieser Abschnitt stellt eine Möglichkeit der technischen Umsetzung eines erweiterten Interfaceblocks dar. Da die Erweiterung speziell für die Kopplung von SystemC-Simulation und IP-Emulation zugeschnitten ist, wird der erweiterte Interfaceblock im weiteren Verlauf dieser Arbeit als **Simulationsinterfaceblock (SimIFB)** bezeichnet.

3.3.1 Die hardwareseitige Implementierungsplattform

Als Implementierungsplattform für den Hardwareteil und die Emulation der Intellectual Property kommt das Digilent 2E Development Board (Abbildung 3.15) zum Einsatz. Es ist eine Entwicklungsplatine, die mit einem Xilinx Spartan2E FPGA-Chip versehen ist. Die genaue Bezeichnung des FPGA-Chips lautet XC2S200E-PQ208. Mit dem FPGA können bis zu 200 000 Gatteräquivalente nachgebildet werden. Die Entwicklungsplatine verfügt außerdem über mehrere Extensionheader sowie über eine serielle, eine parallele und eine JTAG-Schnittstelle.

Mehr Informationen zur Entwicklungsplatine bietet das dazugehörige Handbuch [Dig02]. Erläuterungen zum Xilinx Spartan 2E FPGA enthält das Datenblatt [Xil03].

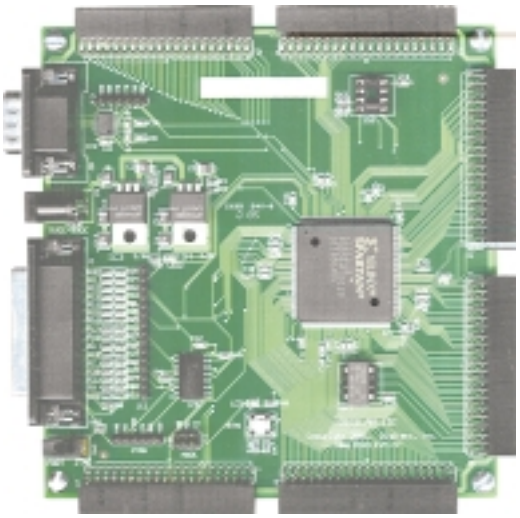


Abbildung 3.15: Digilent 2E Developmentboard mit Xilinx Spartan 2E FPGA, Quelle [Dig]

3.3.2 Umsetzung des HW/SW-Interfaces

Als physische Hardware/Software-Schnittstelle kommen unter Nutzung des Digilent 2E nur dessen Schnittstellen in Frage. Dass heißt, eine Verbindung ist über den seriellen, den parallelen, den JTAG-Port oder über den Extensionheader möglich.

In Abschnitt 3.1.1 wurde die Verwendung von Standardschnittstellen eines PCs angedacht. Ein PC stellt standardmäßig nur die serielle und die parallele Schnittstelle bereit. Somit entfällt die Verwendung von JTAG-Port und Extensionheader.

Die Wahl der Schnittstelle hängt nun von der Leistung der Schnittstelle ab. Tabelle 3.1 stellt die serielle und die parallele Schnittstelle einander gegenüber. Das wichtigste Leistungsmerkmal für die Verbindung der SystemC-Simulation und der IP-Emulation stellt die Übertragunsgeschwindigkeit dar. Deshalb findet die parallele Schnittstelle für die HW/SW-Schnittstelle Verwendung.

Die Kommunikation über die parallele Schnittstelle findet über das EPP-Protokoll (Enhanced Parallel Port) statt. Das EPP-Protokoll ist im IEEE-Standard 1284 definiert. Die Nutzung des EPP-Protokolls bringt einige Eigenschaften mit sich, die für die HW/SW-Schnittstelle und deren Umsetzung von Bedeutung sind.

Merkmal	**serielle Schnittstelle nach RS-232**	**parallele Schnittstelle nach IEEE 1284 (EPP)**
Übertragunsart	seriell asynchron	parallel synchron
Übertragungs-modi	senden empfangen	Daten senden Adressen senden Daten lesen Adressen lesen
Synchronisation	Signalflanke	Handshake
maximale Geschwindigkeit	115 200 Bit/s = 14 400 Byte/s	2 MByte/s
Anzahl der Datenbits pro Zyklus	8	8
Fehlererkennung	Parität	-

Tabelle 3.1: Vergleich von serieller Schnittstelle nach RS-232 und paralleler Schnittstelle nach IEEE 1284

Zum Einen ist das Protokoll ein Master-Slave-Protokoll. Dass heißt, die Kommunikation kann nur von einem Kommunikationspartner initiiert werden. Im Abschnitt 3.1.1 wurde festgestellt, dass es sich bei der vorliegenden Verbindung von SystemC-Simulation und IP-Emulation um eine Form der Master-Slave-Cosimulation handelt. Somit stellt diese Eigenschaft des Protokolls kein Hinderniss dar. Den Master der HW/SW-Schnittstelle bildet die Softwareseite, also der PC auf dem die SystemC-Simulation abläuft. Die Hardwareseite des Systems (Hardwareseite des IFBs und IP-Emulation) stellt den Slave dar. Somit existiert auch innerhalb des IFB eine Master-Slave-Beziehung zwischen den Controlunits. Doch dazu mehr im Abschnitt 3.3.7.

Eine weitere Eigenschaft des EPP-Protokolls, die für die Trennung von Datenfluss und Steuerfluss über die HW/SW-Schnittstelle von Bedeutung ist, bilden die vier verschiedenen Übertragungsmodi. Mit Hilfe des Protokolls ist der Master in der Lage, Daten und Adressen zu schreiben bzw. zu lesen. Die Modi werden durch unterschiedliche Formen des Handshakes realisiert. Zur Veranschaulichung stellt Abbildung 3.16 den Protokollautomaten des EPP-Protokoll auf der Slaveseite dar. Ausführlichere Informationen zur parallelen Schnittstele finden sich in [Axe97], [Pea04] und [War].

Die unterschiedlichen Übertragungsmodi bedeuten für die HW/SW-Schnitt-

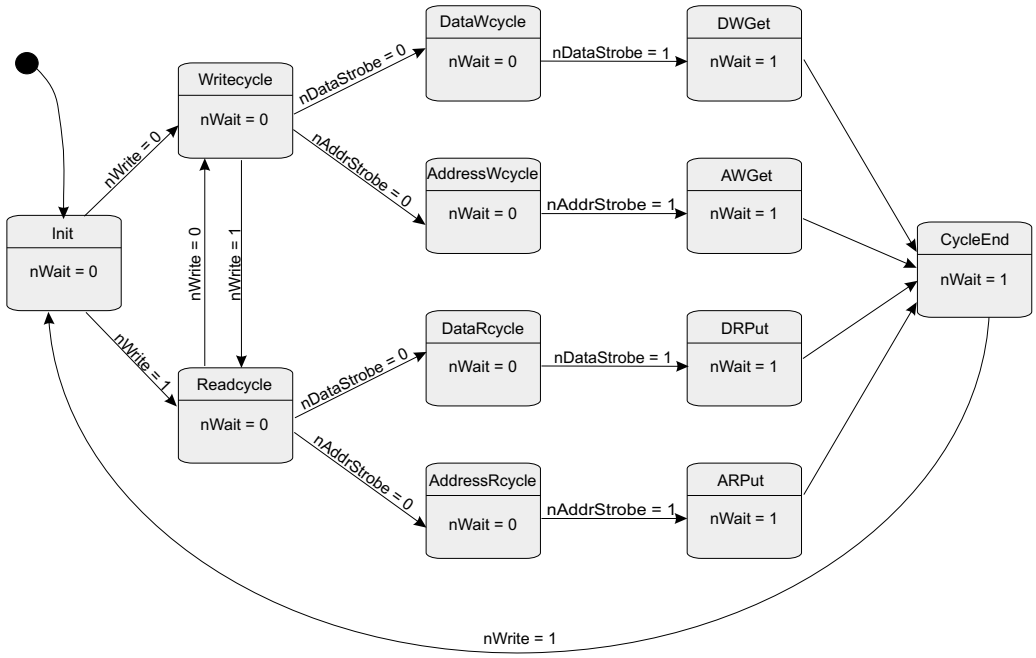


Abbildung 3.16: Protokollautomat des EPP-Protokoll

stelle, dass für die Kommunikation eine Adressbreite von 8 Bit zur Verfügung steht. Mit jeder Adresse können 8 Bit Daten adressiert werden. Mit 8 Bit lassen sich $2^8 = 256$ verschiedene Adressen darstellen. Da jeder Adresse 8 Bit Daten zugewiesen werden können, ergeben sich daraus $256 \cdot 8\,\text{Bit} = 2048\,\text{Bit}$ Daten, die adressiert geschrieben und 2048 Bit Daten, die adressiert gelesen werden können. Mit Hilfe dieses Mechanismus ist es möglich, die Daten mit einer Semantik zu versehen.

Auf dieser Basis setzt das Protokoll zur Datenunterscheidung auf. Es dient dazu, die Simulationsdaten von den Kontrollsignalen der Kommunikation zwischen den Controlunitteilen zu trennen. Allerdings wird hier nicht die gesamte Leistung der EPP-Schnittstelle genutzt. Adressen werden hier nicht doppelt belegt. Dass heißt, mit einer Adresse kann entweder geschrieben oder gelesen werden. Das Protokoll zur Datenunterscheidung beruht auf einer Teilung des Adressraumes. Die ersten 16 Adressen (00h[6] bis 0Fh) dienen zur Kommunikation der Controlunits. Diese haben somit $16 \cdot 8\,\text{Bit} = 128\,\text{Bit}$ für Kontrollsignale zur Verfügung. Für den Austausch von Simulationsdaten stehen die Adressen 10h bis FFh zur Verfügung. Die ersten Adressen ab

[6]Hexadezimale Zahlendarstellung

10h stehen den Eingängen der IP zur Verfügung und sind somit schreibbare Adressen. Im Anschluss daran folgen die lesbaren Adressen. Die erste lesbare Adresse (FRA, first readable address) berechnet sich aus der Formel $\text{FRA} = 10\text{h} + \lceil \frac{\text{Anzahl Eingangsbits der IP} - 1}{8} \rceil$. Somit ergibt sich für die maximale Anzahl von Ein- und Ausgängen der IP $(256 - 16) \cdot 8 = 1920$.

3.3.3 Der PH_{sw}

Die Aufgaben des Protokollhandlers der Softwareseite lauten wie folgt:

- Schnittstelle zur SystemC-Simulation bereitstellen
- Schnittstelle zum Parallelport des PCs implementieren
- Simulationsdaten senden
- Simulationsdaten lesen
- Kommunikation der CU_{SW} und CU_{HW} erlauben

Eine Anforderung an den Simulationsinterfaceblock besteht nach Abschnitt 3.1.1 darin, der SystemC-Simulation eine SW-Schnittstelle bereitzustellen,

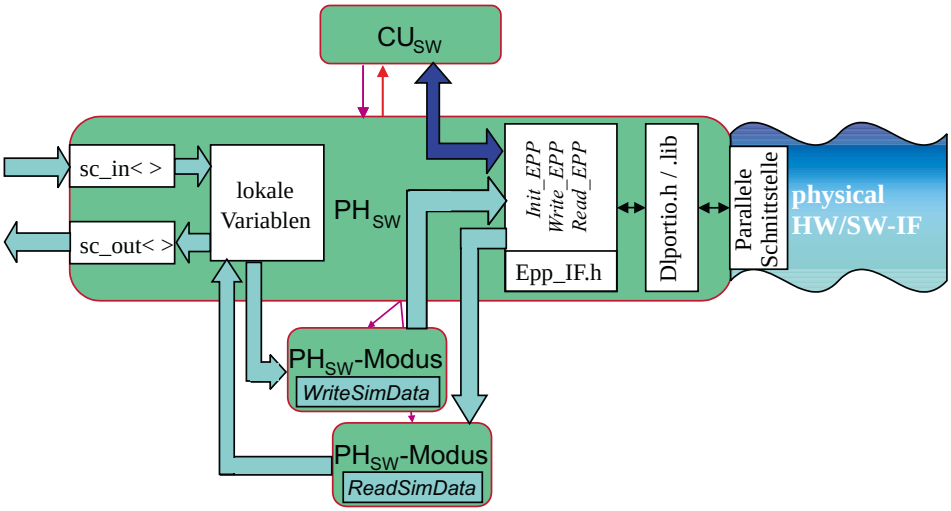


Abbildung 3.17: Detailierter Aufbau des PH_{SW}

Funktion	**Parameter**	**Aufgabe**
Init_EPP	→BADDR	Initialisierung der Schnittstelle
Write_EPP	→BADDR →Adresse →Datenbyte	Schreiben des Datenbyte auf die angegebene Adresse
Read_EPP	→BADDR →Adresse ←Datenbyte	Lesen des Datenbytes von der angegebenen Adresse

Tabelle 3.2: Grundfunktionen zum Zugriff auf die parallele PC-Schnittstelle (BADDR = Basisadresse des Parallelports über den die Kommunikation stattfinden soll; → Eingangsparameter, ← Rückgabeparameter)

bei der der Simulationskern nicht verändert werden muss. Aus diesem Grund wird der PH_{SW} als SystemC-Modul implementiert. Als Schnittstelle kommen die SystemC-Ports *sc_in<>* und *sc_out<>* zum Einsatz. Die Eingänge und Ausgänge der IP bestimmen die Breite und Anzahl der Ports. Lokale Variablen nehmen die Werte der Ports für die weitere Verarbeitung auf (Abbildung 3.17).

Der Zugriff auf den Parallelport des PCs wird über die Bibliothek *dlportio.lib* hergestellt. Diese Bibliothek wurde nicht in dieser Arbeit entwickelt, sondern als bereits fertige Lösung von der Website www.driverlinx.com [Scib] bezogen. Das Programm *port95nt.exe* [Scia] von der Website muss auf dem Rechner installiert sein, um die Bibliothek nutzen zu können.

Auf die Funktionen der Bibliothek bauen die drei Funktionen *Init_EPP, Write_EPP* und *Read_EPP* auf. Ihre Parameter und Aufgaben führt Tabelle 3.2 auf. Nur über diese Funktionen kann auf die parallele Schnittstelle zugegriffen werden. Diesen Weg nutzen alle Funktionen der Softwareseite des IFB, die auf die Schnittstelle zugreifen müssen. Das betrifft die PH_{SW}-Modi und die CU_{SW}. Der Zugriff über fest definierte Funktionen bringt den Vorteil, dass die Bibliothek *dlportio.lib* auch ausgetauscht werden kann und nur diese Funktionen angepasst werden müssen, falls eine andere Art des Schnittstellenzugriffs implementiert werden soll.

Den Transport der Simulationsdaten übernehmen die Modi des PH_{SW}. Den Modus zum Schreiben realisiert die Funktion *WriteSimData*. Den für das Lesen zuständigen Modus bildet die Funktion *ReadSimData*. Die Übergabeparameter sind bei beiden Funktionen gleich. Sie bestehen aus der Basisadres-

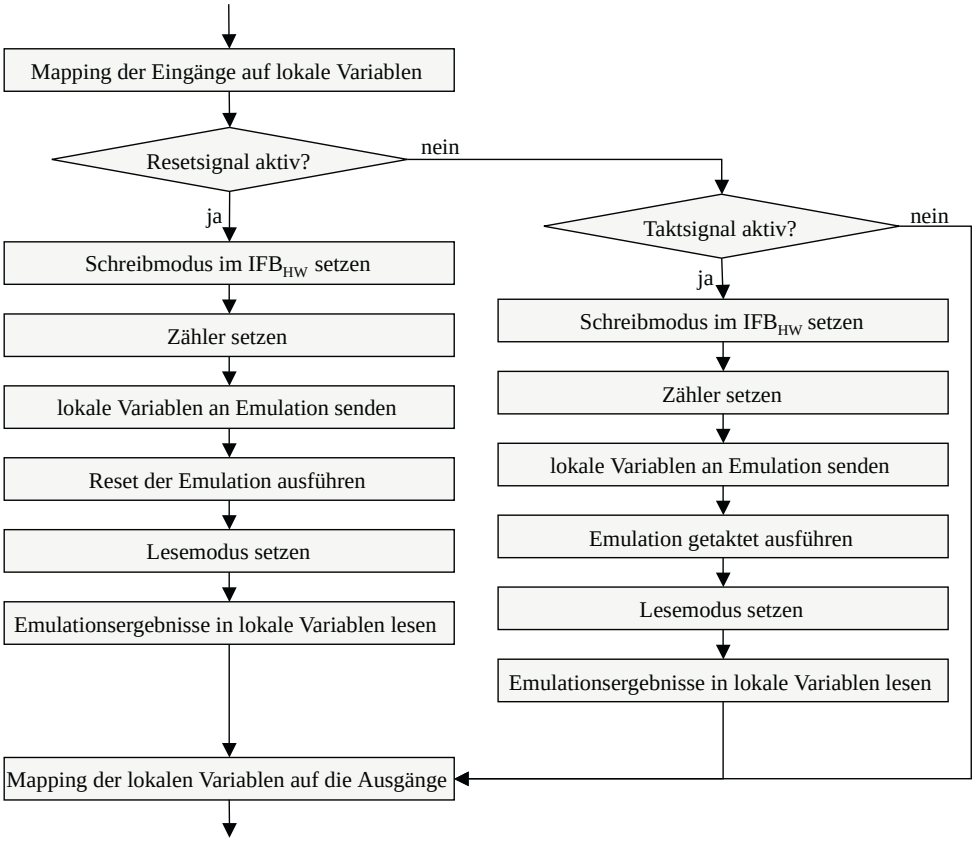


Abbildung 3.18: Ablaufplan zur Umsetzung des PH_{SW}

se des Parallelports, einem Zeiger auf ein Feld, dass die Simulationsdaten enthält bzw. aufnimmt und einem Wert für die Größe des Feldes in Byte.

Den Ablauf eines Funktionszyklus des PH_{SW} stellt Abbildung 3.18 dar. Am Anfang erfolgt das Übertragen der Eingänge auf lokale Variablen. Sie dienen als Zwischenspeicher, um die Daten auf die Breite der Eingänge der IP zu konvertieren. Anschließend wird geprüft, ob das asynchrone Resetsignal aktiv ist. Es besitzt Vorrang vor dem Taktsignal, da auch bei Registern das asynchrone Resetsignal die höhere Priorität hat. Ist das Signal aktiv, dann setzt die CU_{SW}-Funktion *SetWriteMode* die CU_{HW} in den Schreibmodus. Nun beginnt das Setzen der Zähler und das Übertragen der Simulationsdaten. Die Ausführung des asynchronen Reset für die IP-Emulation schließt sich an. Nachdem dieser Schritt beendet ist, setzt die Funktion *SetReadMode* die CU_{HW} in den Lesemodus. Das Auslesen der Emulationsergebnisse erfolgt in lokale Variablen, die zum Schluss auf die Ausgänge übertragen werden.

Ist das asynchrone Resetsignal nicht aktiv, so findet die Prüfung auf Aktivität des Taktsignals statt. Die einzelnen Schritte sind denen der Resetausführung gleich. Der einzige Unterschied besteht darin, dass nach dem Schreiben der Simulationsdaten kein Reset durchgeführt wird, sondern das Taktsignal, entsprechend der gesetzten Zählerwerte, an die IP-Emulation angelegt wird.

3.3.4 Der $PH_{HW\text{-}in}$

Der hardwareseitige Protokollhandler (Abbildung 3.19), der die Schnittstelle zur physischen HW/SW-Schnittstelle bildet, hat folgende Aufgaben:

- Kommunikation über die parallele Schnittstelle realisieren
- Trennung von Simulationsdaten und CU-Kommunikation

Für die Kommunikation über die parallele Schnittstelle implementiert der Modus des $PH_{HW\text{-}in}$ den Protokollautomaten für die Slave-Seite des EPP-Protokolls. Abbildung 3.16 stellt den Protokollautomaten dar. Es werden 4

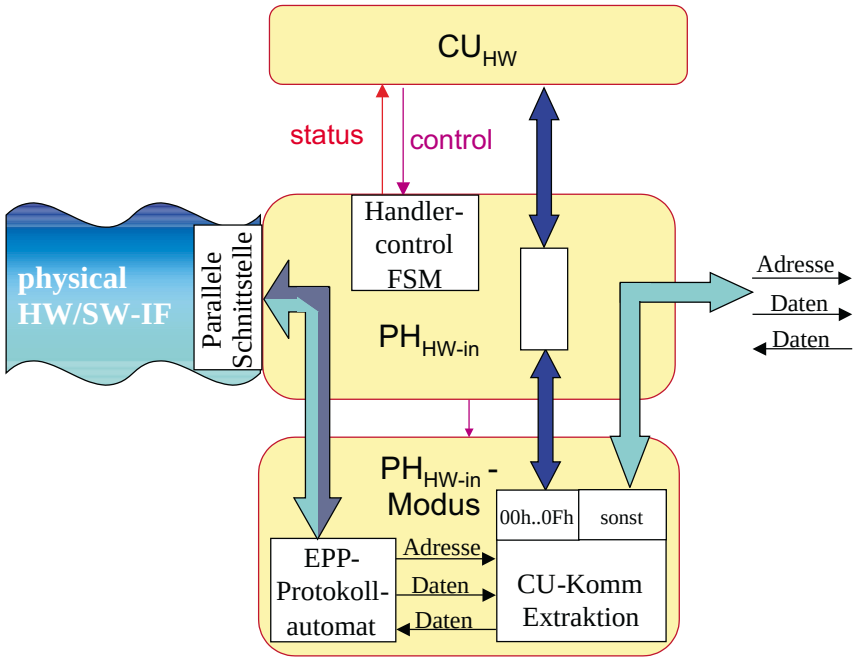


Abbildung 3.19: Detailierter Aufbau des $PH_{HW\text{-}in}$

EPP-Übertragungsarten durch den Protokollautomaten implementiert. Dazu zählen das byteweise Lesen von Adressen, Schreiben von Adressen, Lesen von Daten und Schreiben von Daten. Für die vorliegenden Aufgaben wird jedoch die Funktion des Lesens von Adressen nicht benötigt. Der Protokollautomat stellt in Registern die empfangene Adresse und die empfangenen Daten bereit. Ein weiteres Register speichert Daten zwischen, die von der Softwareseite abgerufen werden.

Die empfangene Adresse ist Grundlage für die zweite Aufgabe des $\mathrm{PH_{HW\text{-}in}}$, die Trennung von Simulationsdaten und CU-Kommunikation. Die CU-Kommunikation findet über die Adressen 00h bis 10h statt. Dieser Adressraum ist wiederum in einen schreibbaren (00h bis 07h) und einen lesbaren Teil (08h bis 10h) unterteilt. Es stehen also 8 Adressen zum Lesen und 8 zum Schreiben bereit. Die genaue Belegung und Funktion der Adressen ist im Abschnitt 3.3.7 beschrieben. Alle Adressen, die größer als 10h sind, werden unverändert an den $\mathrm{SH_{HW}}$ weitergegeben. Die Auswertung dieser Adressen ist nicht Aufgabe der $\mathrm{PH_{HW\text{-}in}}$.

Der endliche Automat Handlercontrol (Abbildung 3.19) dient allein zur internen Kontrolle des Handlers und zum Starten eines Modus. Den Datentransport durch den IFB beeinflusst er nicht.

3.3.5 Der $\mathrm{SH_{HW}}$

Die Aufgabe des Sequenzhandlers im Hardwareteil des Simulationsinterfaceblocks liegt in der korrekten Verteilung der Simulationsdaten. Durch die HW/SW-Schnittstelle liegen die Daten in Form von Tupel (Adresse, Datenbyte) vor. Anhand der Adresse muss nun zuerst eine Unterscheidung getroffen werden, ob es sich um eine Schreibadresse oder eine Leseadresse handelt.

$$\text{Schreibadressen:} \quad 10\text{h} \qquad \ldots\ 10\text{h} + \left\lceil \frac{n_i - 1}{8} \right\rceil - 1 \tag{3.1}$$

$$\text{Leseadressen:} \quad 10\text{h} + \left\lceil \frac{n_i - 1}{8} \right\rceil \ldots\ 10\text{h} + \left\lceil \frac{n_i - 1}{8} \right\rceil + \left\lceil \frac{n_o}{8} \right\rceil - 1 \tag{3.2}$$

n_i ... Anzahl Eingangsbits der emulierten IP mit Taktsignal
n_o ... Anzahl Ausgangsbits der emulierten IP

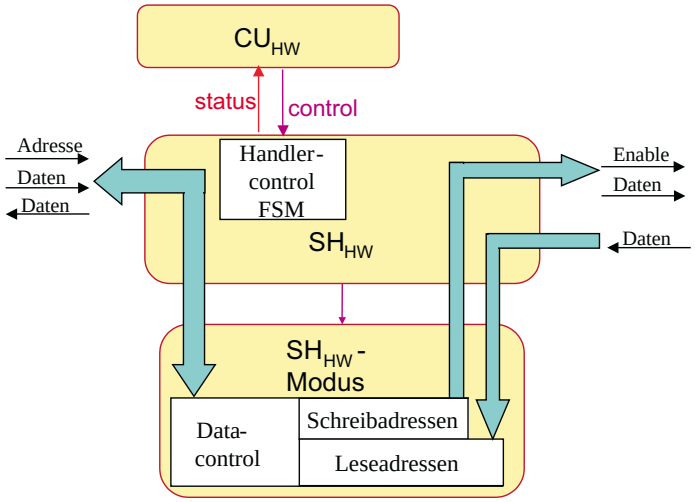


Abbildung 3.20: Detailierter Aufbau des SH_{HW}. Die Werte der Schreibadressen bilden den Bereich nach Formel 3.1, die Leseadressen nach Formel 3.2

Den Bereich der Schreibadressen gibt Formel 3.1 an. Er ist abhängig von der Anzahl der Eingangsleitungen n_i der emulierten IP. Da das Taktsignal gesondert behandelt wird, muss es von den Eingangsleitungen abgezogen werden. Die niedrigste Adresse, die überhaupt Daten transportieren kann, stellt die 10h dar.
Die Leseadressen schließen sich an den Bereich der Schreibadressen an. Sie sind zusätzlich abhängig von der Anzahl der Ausgangsleitungen der emulierten IP. Den Adressraum gibt Formel 3.2 an.

Nach der Dekodierung der Adresse werden die Daten zugeordnet. Für die Schreibadressen kommt dazu ein Demultiplexer zum Einsatz, da das Datenbyte je nach Adresse auf unterschiedliche Leitungen gelegt werden muss. Außerdem werden Enablesignale erzeugt, die zur Kommunikation mit dem $PH_{HW\text{-}out}$ dienen. Für die Abfrage von Leseadressen kommt ein Multiplexer zum Einsatz. Er legt die Daten vom $PH_{HW\text{-}out}$, die der aktuellen Adresse entsprechen, auf die Datenleitung zum $PH_{HW\text{-}in}$.

3.3.6 Der $PH_{HW\text{-}out}$

Die Aufgabe des Protokollhandlers der Hardwareseite, der die Schnittstelle zur IP-Emulation bildet, besteht darin, einen Speicher für die Simulationsdaten bereitzustellen. Zu speichern sind sowohl die anzulegenden Daten als

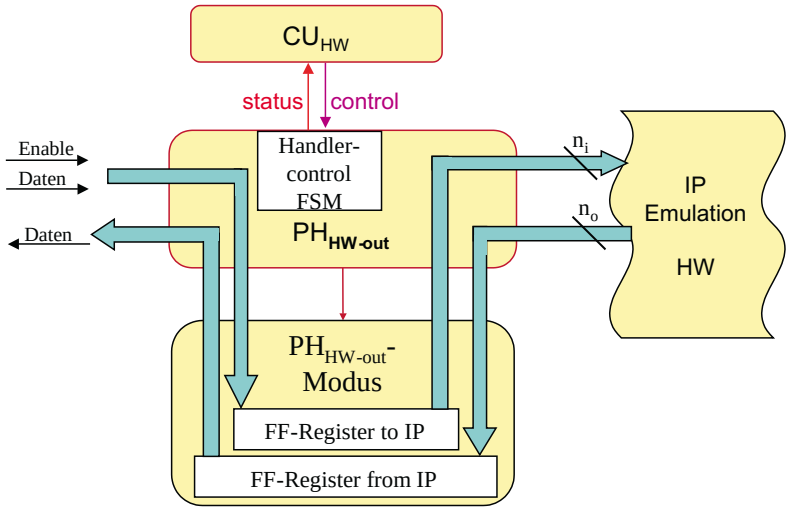


Abbildung 3.21: Detailierter Aufbau des $SH_{HW\text{-}out}$

auch die Emulationsergebnisse. Zusätzlich muss der $PH_{HW\text{-}out}$ noch einen Mechanismus enthalten, mit dem eine kontrolliert Auslösung des Taktsignals für die Emulation möglich ist.

Für jedes Eingangs- sowie Ausgangssignal der emulierten IP wird ein eigenständiger Speicher benötigt. Deshalb kommen Flipflops als Speicher zum Einsatz. Da die IP n_i Eingänge und n_o Ausgänge besitzt, muss der Modus des $PH_{HW\text{-}out}$ $n_i + n_o$ Flipflops zur Speicherung der Simulationsdaten bereitstellen.

Um eine Steuerung der Datenspeicherung zu erlauben, werden die Flipflops mit Enable-Signalen gesteuert. Die Flipflops mit den Eingangsdaten der IP werden durch den SH_{HW} gesteuert. Die Flipflops, die die Ausgangswerte der IP übernehmen, durch ein Kontrollsignal von der CU_{HW}, da diese die genaue Gültigkeit der Daten kennt.

Unter den n_i Flipflops für die Eingangssignale der IP befindet sich auch ein Flipflop für das Taktsignal. Es wird mittels eines Kontrollsignals von der CU_{HW} gesetzt oder gelöscht, da die CU_{HW} die Taktsteuerung implementiert.

3.3.7 Die Controlunit

Die Controlunit (Abbildung 3.22) besteht aus einem Hardware- (CU_{HW}) und einem Softwareteil (CU_{SW}), die beide zusammenarbeiten. Sie kommunizieren

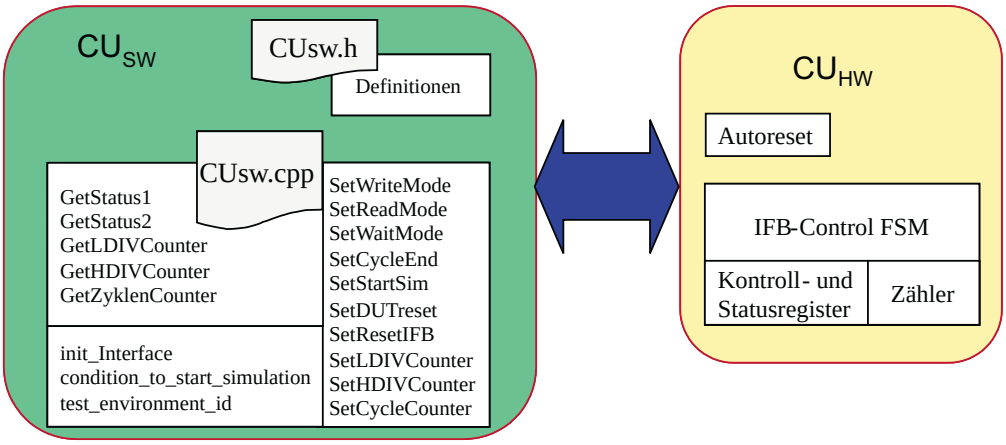


Abbildung 3.22: Detailierter Aufbau der Controlunits

miteinander über die HW/SW-Schnittstelle mit Hilfe von PH_{SW} und $PH_{HW\text{-}in}$. Die Grundlagen der Kommunikation wurden bereits in den Abschitten 3.3.3 und 3.3.4 erläutert.

In den Abschnitten 3.1.1 und 3.2.3 wurde herausgearbeitet, dass zwischen der SystemC-Simulation und der IP-Emulation eine Master-Slave-Beziehung besteht. Diese Beziehung wird auf den IFB übertragen, so dass der Softwareteil, also die CU_{SW}, die Rolle des Masters annimmt und der Hardwareteil, also die CU_{HW}, die Rolle des Slave.

Zunächst soll der Hardwareteil der Controlunit näher betrachtet werden. Er besteht aus einem Automaten zur Kontrolle des Interfaceblocks, der zusätzlich Kontrollregister, Statusregister und Zähler beinhaltet, und eine Komponente Autoreset. Über die Kontrollregister erhält der Automat Steuersignale von der CU_{SW}. Die Statusregister erlauben der CU_{SW}, den aktuellen Zustand des Hardwareteils abzufragen. Die Funktion der Kontroll- und Statusregister, die sie erfüllen, sowie die Adresse, über die sie angesprochen werden, führen Tabelle 3.3 und 3.4 auf.

Das Zustandsdiagramm der endlichen Automaten stellt Abbildung 3.23 dar. Aus Gründen der Übersichtlichkeit sind die Ausgaben der Zustände in Tabelle 3.5 aufgeführt. Zu Beginn führen die Zustände *Start_PHin*, *Start_SH* und *Start_PHout* das Starten der Handlerautomaten durch. Anschließend wartet der Automat im Zustand *Waiting* auf Signale der CU_{SW}.

Von diesem Zustand aus erreicht der Automat durch Setzen des Signals *cwrite* bzw. *cread* die Zustände *DWrite* bzw. *DRead*. In diesen Zuständen sollten

Adresse	**w/r**	**Funktion**
00 h	w	Neutrale Adresse
01 h	w	Kontrollregister
02 h	w	LDIV Zähler
03 h	w	HDIV Zähler
04 h	w	Zyklenzähler High-Byte
05 h	w	Zyklenzähler Low-Byte
06 h	w	keine
07 h	w	keine
08 h	r	Statusregister1
09 h	r	Statusregister2
0A h	r	LDIV Zählerwert
0B h	r	HDIV Zählerwert
0C h	r	Zyklenzählerwert High-Byte
0D h	r	Zyklenzählerwert Low-Byte
0E h	r	keine
0F h	r	Environment ID

Tabelle 3.3: Die Register der hardwareseitigen Controlunit. (w - schreibbar, r - lesbar)

die Simulationsdaten geschrieben bzw. gelesen werden. Dies ist nicht zwingend notwendig. Um jedoch ein versehentliches Starten der Emulation zu verhindern, sollten diese Funktionen in den Zuständen ausgeführt werden.

In den Zustand *NoAction* geht der Automat, falls das Startsignal gesetzt ist, aber noch kein Wert für die Anzahl der Simulationszyklen im Zyklenzähler eingetragen ist. Dies verhindert, dass ein Emulationszyklus durchlaufen wird ohne eine Angabe der Zyklenanzahl.

Der Zustand *DUTreset* führt einen asynchronen Reset der IP-Emulation durch. Da der asynchrone Reset dem Taktsignal übergeordnet und von ihm unabhängig ist, wird die Emulation dabei ohne Taktsignal ausgeführt. Die Speicherung der Ausgabedaten der Emulation erfolgt dabei wie in einem normalen Taktzyklus.

Beim Start der Emulation durch das Signal *cstart* wechselt der Automat in den Zustand *DPrepare* falls der Zyklenzähler einen Wert ungleich Null enthält. Der Zustand dient der Vorbereitung auf die Emulation. Es werden die Flipflops zur Aufnahme der Emulationsdaten im $PH_{HW\text{-}out}$-Modus freigegeben und die Zähler LDIV und HDIV geladen. Anschließend wechselt

Bit	7	6	5	4	3	2	1	0
Kontrollregister	creset	-	-	cdutreset	cnext	cstart	cwrite	cread
Statusregister1	phin_runs	sh_runs	phout_runs	snext	swork	sready	swrite	sread
Statusregister2	-	-	-	-	-	zero_cycles	zero_hdiv	zero_ldiv

Tabelle 3.4: Belegung der Kontroll- und Statusregister der CU_{HW}

der Automat in den Zustand *Clk_gen1*, in dem die Taktgenerierung beginnt. Der Zustand *Clk_gen1* erzeugt den Highpegel in positiver Logik während der Zustand *Clk_gen0* den Lowpegel generiert. Die Dauer der generierten Pegel bestimmten die Werte der Zähler HDIV und LDIV. Mit Hilfe der Zähler ist es möglich, die IP-Emulation wahlweise mit einem symmetrischen oder einem asymmetrischen Taktsignal zu betreiben. Die Frequenz, mit der die IP emuliert wird, berechnet sich nach der Formel $f_{IP} = \frac{f_{IFB}}{HDIV+LDIV+2}$. Die Herleitung der Formel beschreibt Anhang A. Um die gewünschte Anzahl an Taktperioden zu erzeugen, werden die Zustände *Clk_gen1* und *Clk_gen0* so oft nacheinander abgearbeitet, bis der Zyklenzähler den Wert 0 erreicht hat.

Nach Abschluss der Taktgenerierung wechselt der Automat in den Zustand *End_Cycle*. Dieser Zustand sperrt die Flipflops des $PH_{HW\text{-}out}$-Modus wieder, damit die aufgenommenen Emulationsdaten nicht verfälscht werden können. Der Automat verweilt in diesem Zustand, bis das Signal *cnext* oder *cread* gesetzt wird.

Das Signal mit der obersten Priorität ist *creset*. Es bringt den Automaten in den Zustand *DoReset*. Er löst ein Signal für die Komponente Autoreset aus. Das Signal veranlasst die Komponente, einen asynchrone Reset für die Hardwareseite des Interfaceblocks auszulösen. Somit ist es möglich, den Interfaceblock durch ein Signal der Softwareseite rückzusetzen.

Der Softwareteil der Controlunit setzt sich aus einem Definitionsteil und einem Funktionsteil zusammen. Die Definitionen werden in der Datei *CUsw.h*

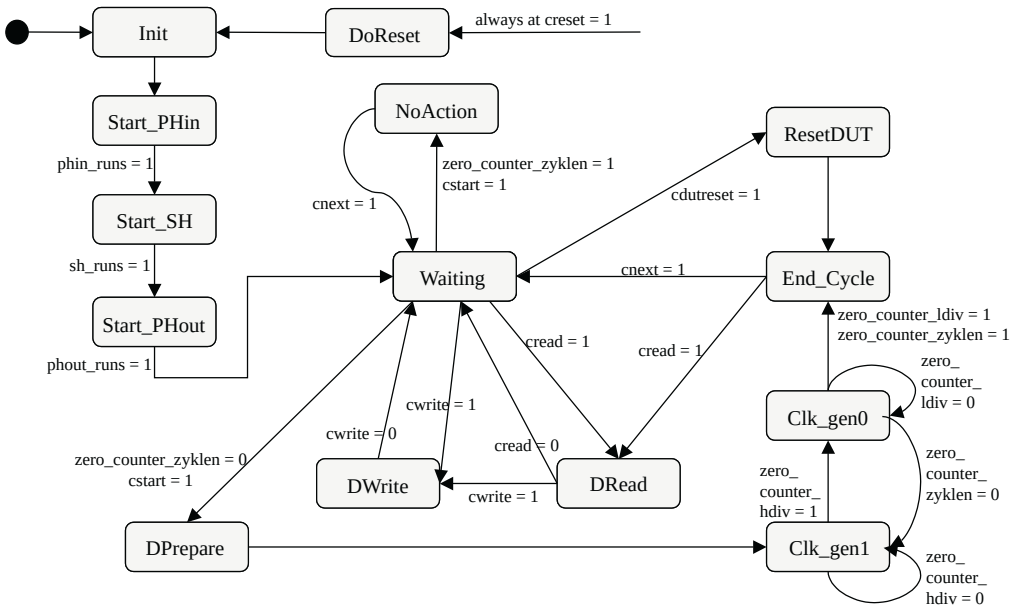


Abbildung 3.23: FSM der Controlunit. Die Ausgaben der Zustände sind in Tabelle 3.5 zu finden.

getroffen. Sie dienen zum Festlegen der Adressen der Kontroll- und Statusregister des Hardwareteils, sowie der Festlegung von vordefinierten Werten, mit denen zum Beispiel die Kontrollregister gesetzt und die Statusregister ausgewertet werden können. Diese Definitionen stehen der gesamten Softwareseite des Interfaceblocks zur Verfügung.

Die Funktionen der CU_{SW} sind in Tabelle 3.6 zusammengefasst. Sie teilen sich wie folgt in drei Gruppen:

- Funktionen zum Schreiben in die Kontrollregister
- Funktionen zum Abruf der Statusregister
- Hilfsfunktionen

Die Funktionen zum Schreiben in die Kontrollregister ermöglichen es, Funktionen im Hardwareteil des Interfaceblocks auszulösen. Der Automat kann in verschiedene Zustände versetzt werden, die Emulation gestartet oder der Interfaceblock zurückgesetzt werden. Außerdem erlauben die Funktionen das Setzen der Zähler HDIV, LDIV und das Taktzyklenzählers.

Signal \ **Zustand**	Init	Start_PHin	Start_SH	Start_PHout	Waiting	DWrite	DRead	DPrepare	Clk_gen1	Clk_gen0	NoAction	End_Cycle	DoReset	ResetDUT
phin_start	0	1	0	0	0	0	0	0	0	0	0	0	0	0
phin_stop	1	0	0	0	0	0	0	0	0	0	0	0	1	0
sh_start	0	0	1	0	0	0	0	0	0	0	0	0	0	0
sh_stop	1	0	0	0	0	0	0	0	0	0	0	0	1	0
phout_start	0	0	0	1	0	0	0	0	0	0	0	0	0	0
phout_stop	1	0	0	0	0	0	0	0	0	0	0	0	1	0
load_counter_zyklen	0	0	0	0	1	1	1	1	0	0	0	0	0	0
inclk0_counter_zyklen	0	0	0	0	0	0	0	0	0	1	0	0	0	0
inclk1_counter_zyklen	0	0	0	0	0	0	0	0	1	0	0	0	0	0
load_counter_hdiv	0	0	0	0	1	1	1	1	0	1	0	0	0	0
go_counter_hdiv	0	0	0	0	0	0	0	0	1	0	0	0	0	0
load_counter_ldiv	0	0	0	0	1	1	1	1	1	0	0	0	0	0
go_counter_ldiv	0	0	0	0	0	0	0	0	0	1	0	0	0	0
cstart_reset	0	0	0	0	0	0	0	0	1	1	1	0	0	0
sread	0	0	0	0	0	0	1	0	0	0	0	0	0	0
swrite	0	0	0	0	0	1	0	0	0	0	0	0	0	0
sready	0	0	0	0	0	0	0	0	0	0	0	0	0	0
swork	0	0	0	0	0	0	0	1	1	0	0	0	0	1
snext	0	0	0	0	0	0	0	0	0	0	1	1	0	0
ffreg_out_en	0	0	0	0	0	0	0	1	1	1	0	1	0	1
ffreg_in_en	0	0	0	0	0	0	0	0	1	1	0	1	0	1
setsimclock	0	0	0	0	0	0	0	0	1	0	0	0	0	0
activate_reset_loc	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Tabelle 3.5: Die Ausgaben des Clockcontrolautomaten in seinen Zuständen

Der Abruf von Statusregistern dient der CU_{SW} zur Information über den Zustand der CU_{HW}. Außerdem können die aktuellen Werte der Zähler abgerufen werden.

Die Hilfsfunktionen haben die Aufgabe, die Arbeit des Softwareteils des Interfaceblocks zu unterstützen. So sind Funktionen zur Initialisierung der parallelen Schnittstelle, zum Testen von Signalen auf eine bestimmte Bedingung vorhanden. Außerdem eine Funktion, die vor der Aufnahme der Kommunikation mit der Hardwareseite sicherstellen soll, dass sich der richtige Interfaceblock auf dem FPGA befindet.

Funktionsname	**Aufgabe**
SetWriteMode	Setzt cwrite-Bit in CU_{HW}
SetReadMode	Setzt cread-Bit in CU_{HW}
SetWaitMode	Setzt Kontrollregister auf 00h
SetCycleEnd	Setzt cnext-Bit in CU_{HW}
SetStartSim	Setzt cstart-Bit in CU_{HW}
SetDUTreset	Setzt Bit für asynchronen Reset der IP in CU_{HW}
SetResetIFB	Setzt creset-Bit in CU_{HW} und löst damit Reset des IFB aus
SetLDIVCounter	Setzt den Zählen LDIV in CU_{HW}
SetHDIVCounter	Setzt den Zähler HDIV in CU_{HW}
SetCycleCounter	Setzt den Zyklenzähler in CU_{HW}
GetStatus1	Liest erstes Statusbyte von CU_{HW}
GetStatus2	Liest zweites Statusbyte von CU_{HW}
GetLDIVCounter	Liest Wert des LDIV-Zähler
GetHDIVCounter	Liest Wert des HDIV-Zähler
GetZyklenCounter	Liest Wert des Zyklenzählers
condition_to_ start_simulation	Prüft, ob ein Signal einen bestimmten Wert hat
init_Interface	initialisiert die parallele Schnittstelle
test_environment_id	prüft, ob sich auf dem FPGA der richtige IFB befindet

Tabelle 3.6: Funktionen der CU_{SW}

3.4 Anwendungsmöglichkeiten

Der Haupteinsatzzweck dieses Hardware/Software-Interfaces ist die gekoppelte Simulation einer SystemC-Verhaltensbeschreibung und einer synthesefähigen VHDL-IP.

Eine weitere Anwendung liegt in der Simulation eines Prototypen auf einem FPGA unter Verwendung eines SystemC-Testbenches. Dies stellt den Haupteinsatzzweck ohne weitere SystemC-Module dar. Die einzigen Module sind dabei der Testbench, die Mainfunktion und das Koppelmodul des Interfaceblocks. Durch die Verwendung von SystemC können die Simulationsdaten des ausgeführten Designs in Tracefiles aufgezeichnet werden und erlauben so ein einfaches Hardwaredebugging.

Unter Benutzung eines FPGA, das dynamische Rekonfiguration erlaubt, können auch dynamisch rekonfigurierbare Designs emuliert werden. Diese Anwendung erlaubt den Test dieser Designs und wiederum die Aufzeichnung von Testdaten in einem Tracefile oder durch den Nutzer unter Verwendung von C/C++ Operationen.

Weiterhin ist vorstellbar, dass mit Hilfe eines Simulationsinterfaceblocks rechenzeitintensive Teile der Designbeschreibung auf ein FPGA ausgelagert werden. Die benötigte Rechenleistung nimmt ab und eine Beschleunigung der Simulation kann erreicht werden.

4 Der Simulationsinterfaceblock-Generator

Dieses Kapitel stellt das Programm **Simulationsinterfaceblock-Generator** vor. Seine Entwicklung fand im Rahmen dieser Arbeit statt.
Zunächst werden die Gründe für die Implementierung des Programms beschrieben. Der sich anschließende Abschnitt beschreibt die Merkmale und die Möglichkeiten des Programms. Danach wird die Arbeitsweise vorgestellt. Am Ende dieses Kapitels wird auf die Benutzung des Programms eingegangen und Verbesserungsmöglichkeiten aufgezeigt.

4.1 Motivation

Der Simulationsinterfaceblock verbindet eine IP-Emulation mit einer SystemC-Simulation. Die Parameter des Interfaceblocks hängen immer von den Eingängen und Ausgängen der emulierten IP ab. Aus diesem Grund muss der Simulationsinterfaceblock für jede IP neu erstellt werden.

Ein Ziel der Kopplung von Emulation und Simulation durch einen Interfaceblock liegt darin, Zeit bei der Simulationsvorbereitung einzusparen. Die Erstellung eines Simulationsinterfaceblocks kann aber auch viel Zeit in Anspruch nehmen. Die manuelle Erstellung des Simulationsinterfaceblocks erhöht zudem die Wahrscheinlichkeit, dass sich Fehler im Design einschleichen.

Um die menschlichen Fehler einzugrenzen und die Zeit für die manuelle Erstellung einzusparen, soll hier im Folgenden eine automatische Lösung vorgestellt werden. Die Lösung besteht im Programm Simulationsinterfaceblock-Generator und wurde im Rahmen dieser Arbeit entwickelt.

4.2 Arbeitssweise

Dieser Abschnitt stellt die Arbeitsweise des Simulationsinterfaceblock-Generators vor. Zunächst erfolgt die Vorstellung der Eingaben für das Programm

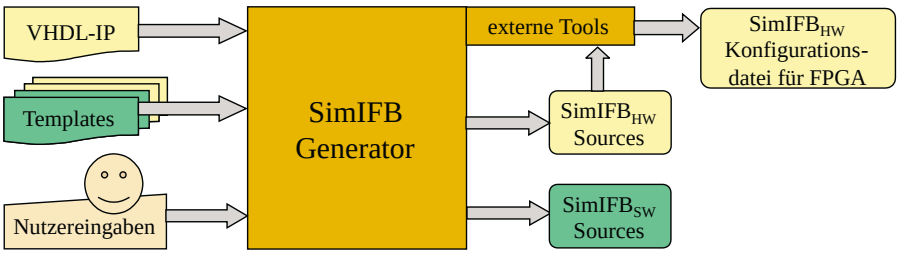


Abbildung 4.1: Eingaben und Ausgaben des SimIFB-Generators

und die gewünschten Ausgaben. Anschließend wird die interne Datenstruktur, auf der das Programm arbeitet, erklärt, bevor der Ablauf des Programms beschrieben wird.

4.2.1 Eingaben und Ausgaben

Für die Entwicklung eines Programmes sind als erstes die Eingaben und Ausgaben zu betrachten (Abbildung 4.1). Als Eingaben stehen dem Simulationsinterfaceblock-Generator die VHDL-IP, die emuliert werden soll und Templates zur Verfügung. Zusätzlich können vom Benutzer noch einige Parameter während des Programmlaufs festgelegt werden.

Nach dem Programmlauf sollen sowohl der Softwareteil also auch der Hardwareteil des Simulationsinterfaceblocks zur Verfügung stehen. Der Softwareteil besteht aus C/C++-Quelldateien, die in ein SystemC-Design eingebunden werden. Der Hardwareteil beinhaltet als erste Ausgabe die VHDL-Quelldateien des Simulationsinterfaceblock und die daran angebundene IP. Aus diesen Quelldateien erfolgt, unter Nutzung von externen Tools zur Synthese, Platzierung und Trassierung, die Generierung der Konfigurationsdatei für das FPGA.

4.2.2 Die internen Datenstrukturen

Ein Programm benötigt zur Speicherung und Verarbeitung von Daten eine Datenstruktur. Die Hauptdatenstruktur im Simulationsinterfaceblock-Generator ist die `ENTITY_STRUCTUR`. Sie nutzt die Datenstrukturen `PORT_LIST` und `GENERIC_LIST`. Zur Unterstützung bestimmter Operationen existieren

ENTITY_STRUCTUR	
Datentyp	**Variablenname**
char*	filename
char*	component_name
int	compname_position
GERNERIC_LIST*	generics
PORT_LIST*	ports
int	data_width_to_dut
int	hs_width_to_dut
int	data_width_from_dut
int	byte_to_dut
int	byte_from_dut
unsigned char	id
int	hdiv_counter
int	ldiv_counter
int	cycle_counter
int	epp_port_addr
int	fpga_prog_after_gen

Tabelle 4.1: Aufbau der Datenstruktur ENTITY_STRUCTUR

des Weiteren die Datenstrukturen `WORD_LIST` und `FILELIST`. Die genaue Beschreibung der Datenstrukturen erfolgt in den sich anschließenden Abschnitten.

4.2.2.1 Die Datenstruktur ENTITY_STRUCTUR

Die grundlegende Datenstruktur im Simulationsinterfaceblock-Generator ist die `ENTITY_STRUCTURE`. Sie speichert alle Daten, die für die Erstellung des Simulationsinterfaceblocks notwendig sind. Die Daten bestehen aus Analyseergebnissen der VHDL-IP und darauf aufbauender Berechnungen, sowie aus Parametern, die vom Nutzer zur Laufzeit des Programms festgelegt werden. Außerdem existieren einige Funktionen, die eine einfache Arbeit mit der Datenstruktur erlauben.

Zunächst soll die Funktion der Variablen der Datenstruktur, die Tabelle 4.1 aufführt, vorgestellt werden. Die Variable *filename* speichert den Namen der Quelldatei, die den Toplevel der VHDL-IP darstellt. Den Namen der VHDL-Entity dieser Datei enthält *component_name* und die Position innerhalb der Datei *compname_position*.

Funktion	**Aufgabe**
make_entitystructur	Erzeugt ein neues Element vom Typ `ENTITY_STRUCTUR`
init_entitystructur	Initialisiert die Datenstruktur vom Typ `ENTITY_STRUCTUR`
reserve_wordspace_ename	Reserviert Speicherplatz für die Variable *component_name*
reserve_wordspace_fname	Reserviert Speicherplatz für die Variable *filename*
check_entity_for_clock	Durchsucht die Ports der Entity nach einem Taktsignal und legt eines an, falls die Suche fehlschlägt.

Tabelle 4.2: Überblick zu den Funktionen zur Datenstruktur ENTITY_STRUCTUR

Die Zeiger *generics* und *ports* verweisen auf die Datenstrukturen `GENERIC_LIST` bzw. `PORT_LIST`. Sie dienen zum Erfassen der Generics und Ports der Entity. Ihre genaue Funktion beschreiben Abschnitt 4.2.2.3 bzw. Abschnitt 4.2.2.2.

Die Variablen *data_width_to_dut* und *data_width_from_dut* geben die benötigte Anzahl Signale an, um Daten zur und von der IP zu transportieren. *Byte_to_dut* und *byte_from_dut* stellen die Werte auf ein volles Byte gerundet dar. Die Variable *hs_width_to_dut* gibt die Breite der Handshakesignale an, die der SH_{HW} zum Zugriff auf den Speicher im $PH_{HW\text{-}out}$ benötigt.

In *id* wird eine 8 Bit lange Identifikationsnummer gespeichert. Ein Zufallsgenerator erzeugt sie. Sie dient zur Identifikation des Hardwareteils des Simulationsinterfaceblocks durch den Softwareteil um sicherzustellen, dass die richtigen Teilstücke miteinander arbeiten.

Die Variablen *hdiv_counter*, *ldiv_counter* und *cycle_counter* speichern die Werte, auf die während der späteren Simulation die Zähler des Hardwareteils eingestellt werden. Die Funktion der Zähler wurde bereits im Abschnitt 3.3.7 erläutert.

Epp_port_addr enthält die Adresse, über die die parallele Schnittstelle des ausführenden PCs angesprochen wird. Der Wert in *fpga_prog_after_gen* gibt an, ob der Simulationsinterfaceblock-Generator das FPGA nach der Generierung des Simulationsinterfaceblocks sofort konfigurieren soll.

Zur Datenstruktur `ENTITY_STRUCTUR` gehören neben den Variablen auch einige Funktionen, die Tabelle 4.2 aufführt. Sie dienen zur einfacheren Arbeit mit der Datenstruktur.

4.2.2.2 Die Datenstruktur PORT_LIST

Zur Speicherung der Struktur der durch den Simulationsinterfaceblock-Generator analysierten VHDL-Entity der IP, dient die Datenstruktur `PORT_LIST`. Sie wird durch eine doppelt verkettete lineare Liste gebildet. Tabelle 4.3 stellt die Variablen eines Elementes der Liste dar. Die Verkettung der einzelnen Elemente realisieren die Zeiger *prev* und *next*.

Eine VHDL-Entity besteht aus Ports, die die Schnittstelle der jeweiligen Komponente definieren. Für jeden Port existiert in der Liste ein Element. Die Variable *port_name* gibt den Namen des Ports an. *Port_position* speichert die Position des Ports in der VHDL-Datei. Die Richtung des Ports, also ob es sich um einen Eingang, Ausgang, einen bidirektionalen Port oder einen Buffer handelt, gibt *port_direction* an.

Ein Port besitzt eine Breite *port_width*. Die Grenzen des Ports bestimmen

PORT_LIST	
Datentyp	**Variablenname**
char*	port_name
int	port_position
unsigned char	port_direction
unsigned int	port_width
unsigned int	port_high
unsigned int	port_low
unsigned char	port_align
unsigned int	map_from
unsigned int	map_to
unsigned char	special
unsigned char	active
unsigned char	sign
struct port_list*	prev
struct port_list*	next

Tabelle 4.3: Aufbau der Datenstruktur PORT_LIST

Funktion	**Aufgabe**
make_port	Erzeugt ein neues Element vom Typ `PORT_LIST`
init_portlist	Initialisiert die Datenstruktur
insert_port	Fügt ein Element in eine bestehende Liste ein
reserve_wordspace	Reserviert Speicherplatz für die Variable *port_name*
sort_portlist_width	Sortiert die Liste absteigend nach der Breite der Ports *port_width*
sort_portlist_direction	Sortiert die Liste nach der Richtung der Ports *port_direction*

Tabelle 4.4: Überblick zu den Funktionen zur Datenstruktur PORT_LIST

port_high und *port_low*. Die Ausrichtung des Ports, dass heißt „to“ oder „downto“ falls er ein Vektor ist, speichert die Variable *port_align*.

Nach der Entityanalyse berechnet das Programm noch Werte zur Abbildung der Signale innerhalb des Interfaceblocks. Die Variablen *map_from* und *map_to* nehmen diese auf. Spezielle Attribute eines Ports, wie Takt- oder Resetsignal, legt die Variable *special* fest. In diesem Zusammenhang bestimmt *active* die aktive Taktflanke dieses Signals. Ob der Port im erstellten SystemC-Modul vorzeichenbehaftet ist oder kein Vorzeichen besitzt, beeinflusst die Varibale *sign*.

Neben den Variablen erleichtern einige Funktionen die Arbeit mit der Datenstruktur. Ihre Aufgaben stellt Tabelle 4.4 dar.

4.2.2.3 Die Datenstruktur GENERIC_LIST

Die Datenstruktur `GENERIC_LIST` wurde eingeführt, um generische Parameter in der VHDL-Entity zu unterstützen. Allerdings existiert bis jetzt nur die Definition der Datenstruktur. Tabelle 4.5 führt die enthaltenen Variablen auf. Eine doppelt verkettete lineare Liste realisert die Datenstruktur ähnlich wie die Liste der Ports (vgl. Abschnitt 4.2.2.2). Die Nutzung von VHDL-Generics ist im Simulationsinterfaceblock-Generator noch nicht implementiert. Aus diesem Grund wird an dieser Stelle auf eine ausführlichere Beschreibung verzichtet.

GENERIC_LIST	
Datentyp	**Variablenname**
char*	generic_name
int	generic_value
struct generic_list*	prev
struct generic_list*	next

Tabelle 4.5: Aufbau der Datenstruktur GENERIC_LIST

4.2.2.4 Die Datenstruktur WORD_LIST

Die Datenstruktur `WORD_LIST` hat die Aufgabe, das Parsen und Analysieren der VHDL-Entity der IP zu unterstützen. Dies geschieht, indem mit Hilfe der Datenstruktur die einzelnen syntaktischen Elemente der Entity wortweise gespeichert werden. Sie ist durch eine doppelt verkettete lineare Liste realisert, deren Elemente die Variablen aus Tabelle 4.6 enthalten.

Die Verkettung der Liste wird durch die Zeiger *prev* und *next* hergestellt. Einzelne Worte speichert die Variable *word*. Die Position des Wortes in der VHDL-Datei beinhaltet die Variable *position*.

Zur Datenstruktur gehören die Funktionen *make_word*, *insert_word* und *reserve_wordspace*, deren Aufgaben Tabelle 4.7 beschreibt.

WORD_LIST	
Datentyp	**Variablenname**
char*	word
int	position
struct word_list*	prev
struct word_list*	next

Tabelle 4.6: Aufbau der Datenstruktur WORD_LIST

Funktion	**Aufgabe**
make_word	Erzeugt ein neues Element vom Typ `WORD_LIST`
insert_word	Fügt ein Element in eine bestehende Liste ein
reserve_wordspace	Reserviert Speicherplatz für die Variable *word*

Tabelle 4.7: Überblick zu den Funktionen zur Datenstruktur WORD_LIST

FILELIST	
Datentyp	**Variablenname**
char*	filename
int	processed
struct filelist*	next

Tabelle 4.8: Aufbau der Datenstruktur FILELIST

Funktion	**Aufgabe**
filelist_new	Erzeugt ein neues Element vom Typ `FILELIST`
filelist_addfile	Fügt einen Dateinamen in die Liste ein, falls dieser noch nicht in der Liste vorhanden ist
filelist_free	Gibt den Speicherplatz für die Liste wieder frei

Tabelle 4.9: Überblick zu den Funktionen zur Datenstruktur FILELIST

4.2.2.5 Die Datenstruktur FILELIST

Die Aufgabe der Datenstruktur `FILELIST` besteht darin, eine Liste von Dateinamen zu speichern. Sie findet Verwendung beim Kopieren der IP und der darin deklarierten Komponenten in das neue Zielverzeichnis.

Die Datenstruktur ist eine einfach verkettete lineare Liste, bei der die Verkettung durch den Zeiger *next* erreicht wird. Außerdem kann sie in jedem Element der Liste einen Dateinamen (*filename*) und den Bearbeitungsstatus dieser Datei (*processed*) speichern. Tabelle 4.8 gibt den Aufbau der Datenstruktur wieder.

Die Funktionen zur Arbeit mit der Datenstruktur führt Tabelle 4.9 auf. Darin ist auch die Aufgabe der Funktionen dargestellt.

4.2.3 Der Programmablauf

Der erste Schritt, den der Simulationsinterfaceblock-Generator im Programmablauf durchführt, ist die Analyse der VHDL-IP. Bevor jedoch die Analyse durchgeführt werden kann, muss die Quelldatei aufbereitet werden. Der erste Arbeitsgang liest die Quelldatei der VHDL-IP in den Speicher. Anschließend entfernt die Funktion *vhdl_remove_comments* die Kommentare aus dem Quelltext. Die Formatierungszeichen, wie z.B. Zeilenumbrüche, Tabulatoren

und mehrfache Leerzeichen, entfernt die Funktion *string_remove_multispace*. Um ein einfacheres Parsen des Quelltextes zu ermöglichen, wandelt die Funktion *string_tolowercase* den Text in Kleinbuchstaben um. Am Ende der Vorbereitung sucht *vhdl_entity_borders* die Grenzen der Entity im Text. Es wird hierbei davon ausgegangen, dass der Quelltext nur eine VHDL-Entity beschreibt oder die erste Entity im Text das Toplevel der IP darstellt. Ihre Position bestimmt der Anfang des Schlüsselwortes *entity* und das Ende die Position des Schlüsselwortes *end*. Anhand dieser Grenzen löst die Funktion *string_extract* die Entity aus dem Text heraus, indem sie den Bereich an den Beginn des Textes schreibt und den Rest mit Leerzeichen auffüllt.

Sind alle vorbereitenden Arbeiten durchgeführt, startet durch die Funktion *vhdl_parse_entity* die Analyse der Entity. Sie beginnt mit einem syntaktischen Trennen des Textes mit Hilfe der Funktion *vhdl_seperate_syntax*. Das bedeutet, dass syntaktische Elemente im Text durch ein Leerzeichen voneinander getrennt werden. Zu den syntaktischen Elementen zählen zum Beispiel die Schlüsselwörter *entity*, *port* und *is*, aber auch Zeichenketten für Namen, sowie Begrenzungszeichen, wie Semikolon und Klammern. Nach der Trennung kommt die Datenstruktur `WORD_LIST` zum Einsatz, in der die syntaktischen Elemente wortweise gespeichert werden.

Auf dieser Datenstruktur erfolgt das eigentliche Parsen der Entity. Die Syntax, auf die der Parser aufbaut, stellt das Syntaxdiagramm in Abbildung 4.2 dar. Das Syntaxdiagramm gibt nicht die vollständige Syntax für eine VHDL-Entity wieder. Es ist nur auf die in diesem Zusammenhang benötigten syntaktischen Konstrukte reduziert. Andere mögliche Konstrukte wie Attribute werden durch den Parser ignoriert. Auch die Auswertung von Generics vernachlässigt der Parser, da ihre Verwendung im Simulationsinterfaceblock-Generator bis jetzt noch nicht implementiert ist.

Der Parser stellt zunächst den Namen der Entity fest und speichert ihn in der Datenstruktur `ENTITY_STRUCTURE`. Anschließend baut er eine Liste der gefundenen Ports und ihrer Parameter mit der Datenstruktur `PORT_LIST` auf. Wurden alle Ports analysiert, prüft die Funktion *vhdl_complete_entity* die Liste der Ports auf Vollständigkeit. Fehlen Informationen zu einem Port, so ergänzt diese die Funktion. Die Namen der Ports können durch die Umwandlung des Quelltextes in Kleinbuchstaben jedoch verfälscht sein. Um zu gewährleisten, dass die Groß- und Kleinschreibung der Portnamen erhalten bleibt, ersetzt die Funktion *vhdl_replace_names* nach dem Parsevorgang die Namen in der Portliste durch ihre Originale aus der Quelldatei. Nach dem Parsevorgang steht nun eine Liste der Ports der IP bereit, in der alle aus der Entity gewonnenen Informationen gespeichert sind.

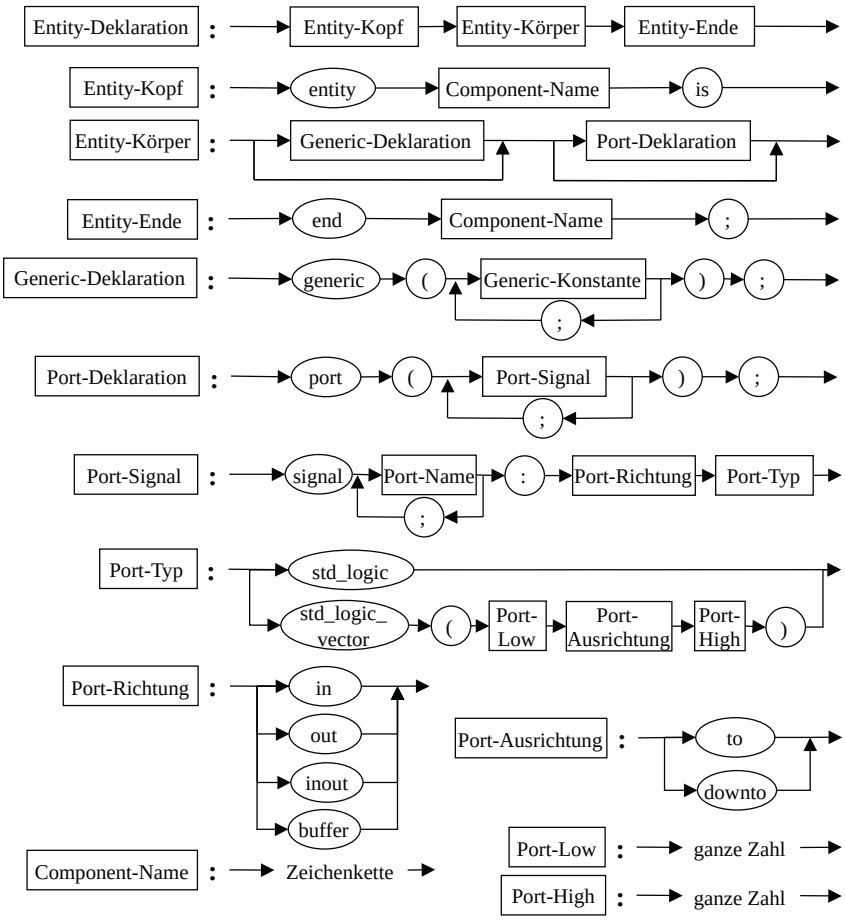


Abbildung 4.2: Syntaxdiagramm einer VHDL-Entity nach dem der Parser des Simulationsinterfaceblock-Generators arbeitet

An dieser Stelle benötigt der Simulationsinterfaceblock-Generator noch ein paar Informationen vom Benutzer. Er muss das Taktsignal sowie das asynchrone Resetsignal und deren aktive Signalflanken festlegen. Diese Informationen dienen später dem Simulationsinterfaceblock zur Auslösung des Transportes der Simulationsdaten. Um die Kompatibilität mit dem SystemC-Design zu gewährleisten, in das der Simulationsinterfaceblock eingebunden werden soll, kann der Benutzer festlegen, ob Vektoren vorzeichenbehaftet sind oder nicht. Der Wert des Zyklenzählers, die Adresse der parallelen Schnittstelle und ob die FPGA durch den Simulationsinterfaceblock-Generator konfiguriert werden soll, sind weitere vom Benutzer einzustellende Parameter.

Nach dem Sammeln aller benötigten Informationen erfolgt eine Sortierung

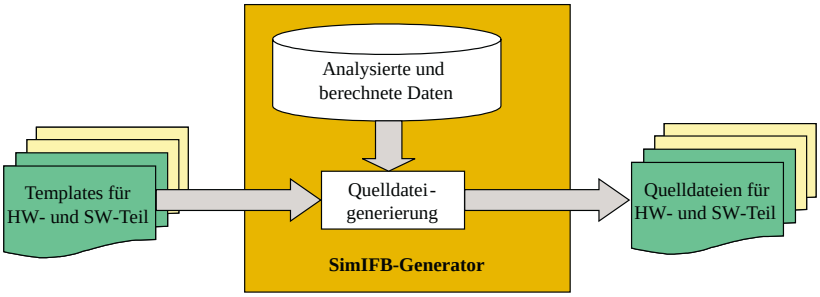


Abbildung 4.3: Erzeugung der Quelldateien des Simulationsinterfaceblocks aus Templates

der Portliste. Zuerst sortiert die Funktion *sort_portlist_width* die Ports absteigend nach ihrer Breite. Im Anschluss sortiert *sort_portlist_direction* sie nach ihrer Richtung. Die Sortierung dient zur Vorbereitung für das Signalmapping. Das Signalmapping bildet die Ports mit Hilfe der Funktion *map_entity_signals* auf interne Signale des Simulationsinterfaceblocks ab. Außerdem wird geprüft, ob die Abbildung zulässig ist, dass heißt, dass nicht mehr Signale durch den Simulationsinterfaceblock transportiert werden müssen, als die HW/SW-Schnittstelle zulässt. Für solch einen Fall bricht das Programm mit einer Fehlermeldung ab.

War das Mapping erfolgreich, startet die Generierungsphase der Quelltexte. Die Generierung erfolgt aus Templates, was schematisch Abbildung 4.3 darstellt. Diese Templates enthalten bereits große Stücken an Quellcode, da viele Teile für den Simulationsinterfaceblock immer gleich sind. Codestücke, die speziell angepasst werden müssen, ersetzt im Template eine Marke. Diese Marken werden während der Generierung durch einen aus den vorhandenen Daten berechneten Wert oder ein anhand dieser Daten generiertes Codestück ersetzt. Anschließend erfolgt die Speicherung des fertigen Quelltextes im Zielverzeichnis.

Die Generierungsphase besteht aus drei Teilabschnitten. Der erste Schritt startet durch Aufruf der Funktion *generate_HWpart*. Er erzeugt alle notwendigen VHDL-Quelldateien des Simulationsinterfaceblocks . Außerdem instanziiert er den Simulationsinterfaceblock und die VHDL-IP unter einem Toplevel, um diese später gemeinsam zu synthetisieren. Während der Bearbeitung einer Quelldatei wird diese auch in eine Projektdatei eingetragen, die alle Dateien zusammenfasst. Sie bildet die Informationsgrundlage für die spätere Synthese. Für die Synthetisierung müssen auch alle in der IP instanziierten Komponenten mit zu den VHDL-Quelldateien kopiert werden. Deshalb sucht

die Funktion *generate_dut* alle instanziierten Komponenten und sammelt sie mit Hilfe der Datenstruktur `FILELIST`. Anschließend kopiert sie die Funktion zu den anderen VHDL-Quellen und fügt sie zur Projektdatei hinzu. Die Voraussetzung, dass diese Funktion ihren Zweck erfüllt, besteht darin, dass sich die instanziierten Komponenten im gleichen Verzeichnis wie der Quelltext der IP befinden und ihr Dateiname mit dem Namen ihrer Komponentendeklaration plus der Dateierweiterung .vhd übereinstimmt.

Nach der Erzeugung der VHDL-Quellen steht die Generierung von Skripten an. Diese Skripte bauen auf externen Programmen auf. Sie dienen der Synthese (xst), der Übersetzung (ngdbuild), dem Mapping (map), der Platzierung und Trassierung (par), der Bitstromgenerierung (bitgen), der FPGA-Konfigurierung (impact) und der Timinganalyse (trce) der erzeugten VHDL-Quellen. Die externen Programme sind in der Xilinx Entwicklungsumgebung ISE [Xilb] enthalten. Da in dieser Arbeit nur Xilinx FPGAs zur Verfügung standen, beschränkt sich der Simulationsinterfaceblock-Generator auf die Nutzung der Xilinx-Tools. Eine ausführliche Beschreibung der Tools ist in [Xila] und [Xild] zu finden. Eine Erweiterung des Programms ist aber durchaus denkbar.
Nach der Generierung der Skripte erfolgt ihre Ausführung durch die Funktion *execute_hwscripts*. Am Ende der Skriptsphase analysiert die Funktion *analyze_timing* das Logfile des Timinganalyseskriptes. Es sucht nach der größten Verzögerungszeit und berechnet daraus die Werte für die LDIV- und HDIV-Zähler. Dies dient dazu, dass die Emulation mit dem richtigen Takt ausgeführt wird, damit die Emulationsergebnisse auch solang abgegriffen werden, bis an den Ausgängen der IP ein korrektes Ergebnis anliegt. Die LDIV- und HDIV-Zähler benötigt die dritte Generierungsphase, um alle Quelldateien des Softwareteils erstellen zu können.

Die dritte Generierungsphase erzeugt den Softwareteil, also die C/C++ Quell- und Headerdateien des Simulationsinterfaceblocks . Sie startet durch Aufruf der Funktion *generate_SWpart*. Der erste Schritt generiert das SystemC-Modul, welches die Schnittstelle des Simulationsinterfaceblocks zur SystemC-Simulation bereitstellt. Dieses Modul trägt den Namen der Entity der VHDL-IP. Die Headerdatei bildet die Schnittstelle, die Quelldatei realisiert die Funktionalität des PH_{SW}. Beide Dateien werden komplett durch den Simulationsinterfaceblock-Generator erstellt und nicht aus Templates. Alle anderen Dateien des Softwareteils generiert das Programm wiederum aus Templates.

Während des Programmlaufs schreibt der Simulationsinterfaceblock-Generator Informationen für den Benutzer zu den einzelnen Operationen sowie

aufgetretene Fehler in ein Logfile. Nach den Generierungsphasen ist das Programm abgeschlossen und die erzeugten Daten können verwendet werden. Mehr Informationen zur Verwendung der Daten bietet der Abschnitt 4.3.

4.3 Benutzung des Programms

Dieser Abschnitt erklärt die Benutzung des **Simulationsinterfaceblock-Generator**. Er gibt Hinweise zum Aufbau des Programms und zu den einzelnen Phasen, die während der Ausführung durchschritten werden. Außerdem enthält dieser Abschnitt Informationen zu den Nutzereingaben, die das Programm in den einzelnen Phasen erwartet. Am Ende wird ein Überblick über die vom Programm erzeugten Daten gegeben.

4.3.1 Programmstart und -aufbau

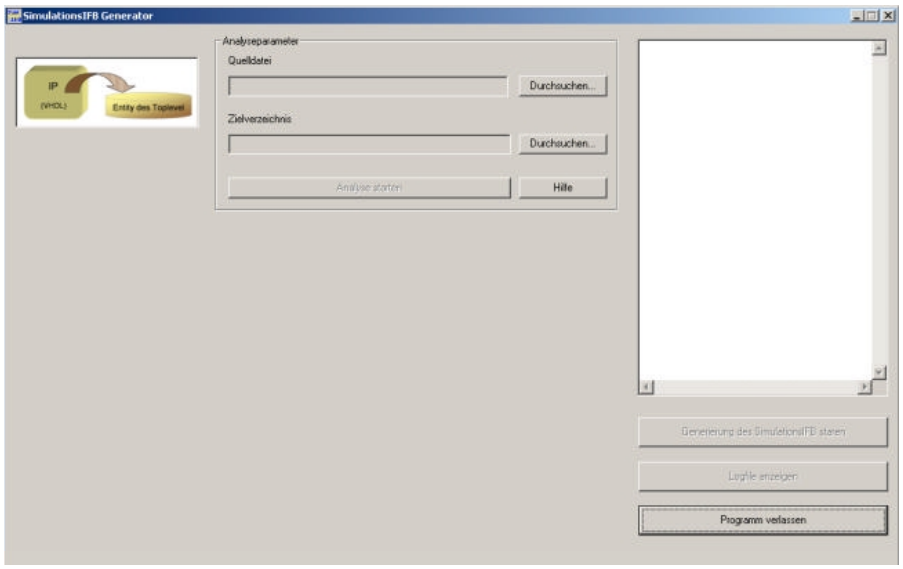


Abbildung 4.4: Ansicht des Simulationsinterfaceblock-Generators nach dem Programmstart

Das Programm Simulationsinterfaceblock-Generator startet durch den Aufruf der Datei ***simifb_generator.exe***. Es erscheint ein Fenster auf dem Bildschirm, das Abbildung 4.4 darstellt. Die linke Seite des Fensters zeigt durch

Grafiken die aktuelle Bearbeitungsphase an. In der Mitte des Fensters liegt der Bereich für die Eingaben des Benutzers. In den einzelnen Feldern des Nutzerbereichs existiert jeweils ein Hilfe-Button, um Informationen zu den entsprechenden Eingabefeldern zu erhalten. Der rechte Teil enthält ein Statusfeld, in dem Informationen vom Programm ausgegeben werden. Darunter befinden sich die Button zum Starten der Generierung, zum Anzeigen des Logfiles und zum Verlassen des Programms.

4.3.2 Die Analyse der Quelldatei

Die erste Eingabe, die der Nutzer durchzuführen hat, ist die Auswahl der VHDL-IP für die ein Simulationsinterfaceblock erstellt werden soll. Dies erfolgt im Bereich *Analyseparameter* unter dem Punkt Quelldatei. Mit einem Klick auf den oberen Durchsuchen...-Button öffnet sich ein Dialogfeld, in dem die Datei ausgewählt werden kann. Das Feld neben dem Button zeigt nach der Auswahl die Datei und ihr Verzeichnis an.

Die zweite Eingabe betrifft das Zielverzeichnis, in dem die generierten Dateien gespeichert werden sollen. Der entsprechende Dialog kann über den zweiten Durchsuchen...-Button erreicht werden. Den Pfad stellt das nebenstehende Textfeld nach der Auswahl dar.

Nach der Auswahl beider Parameter ist die Analysefunktion anwählbar (Abbildung 4.5). Durch Klicken des Button Analyse starten beginnt das Programm die VHDL-IP zu analysieren und ihre interne Datenstruktur aus der Entity der VHDL-IP aufzubauen.

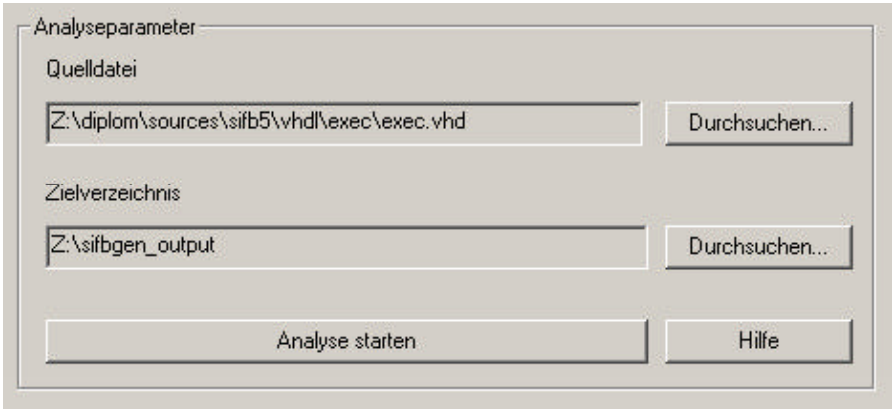


Abbildung 4.5: Ansicht des Feldes Analyseparameter

4.3.3 Festlegung von Signalparametern

Es werden zwei weitere Felder nach Beendigung der Analyse sichtbar. Zum Einen das Feld *Auswahl des Taktsignals* und zum Anderen das Feld *Auswahl des asynchronen Resetsignals*. Abbildung 4.6 zeigt das Programm in diesem Programmabschnitt.

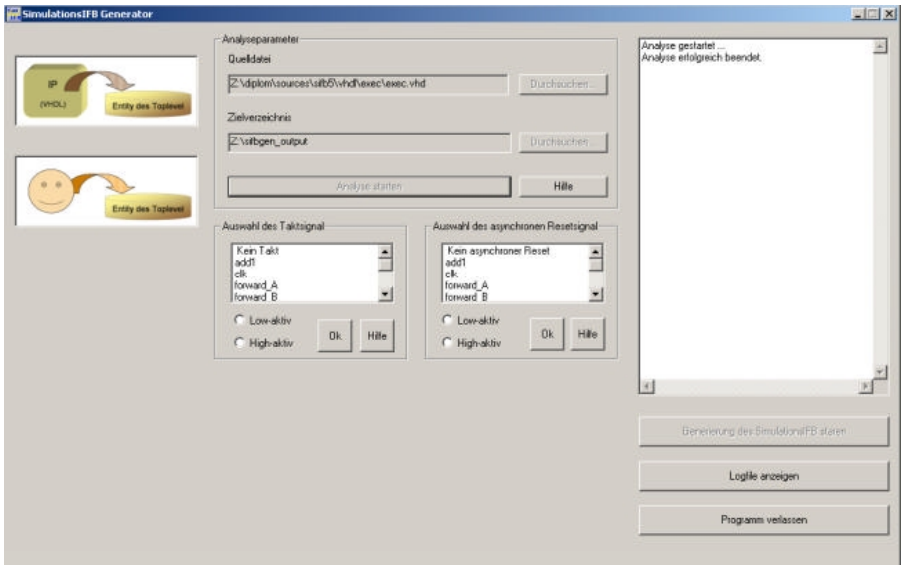


Abbildung 4.6: Der Simulationsinterfaceblock-Generator nach der Analysephase

Die in diesem Abschnitt erschienenen Felder dienen zur Auswahl des Taktsignals und des asynchronen Resetsignals aus den Eingängen der VHDL-IP. Auf Grund dieser Informationen erzeugt der Simulationsinterfaceblock-Generator die Sensitivitätsliste des SystemC-Moduls. Es existiert jeweils für Takt- und Resetsignal die Option, dass diese Signale in der VHDL-IP nicht vorhanden sind. Zur Darstellung dieses Sachverhaltes kann der Listeneintrag „Kein Takt“ bzw. „Kein asynchroner Reset“ ausgewählt werden. Ist ein Signal ausgewählt, so muss der Benutzer zusätzlich dessen aktive Taktflanke festgelegen. Ein Klick auf den jeweiligen Ok-Button beendet die Eingabe der Signalparameter und lässt keine Änderung an den Einstellungen mehr zu. Abbildung 4.7 stellt eine detailierte Ansicht der Felder zur Auswahl des Takt- und des asynchronen Resetsignals dar. Fehler bei der Eingabe zeigt das Statusfeld an.

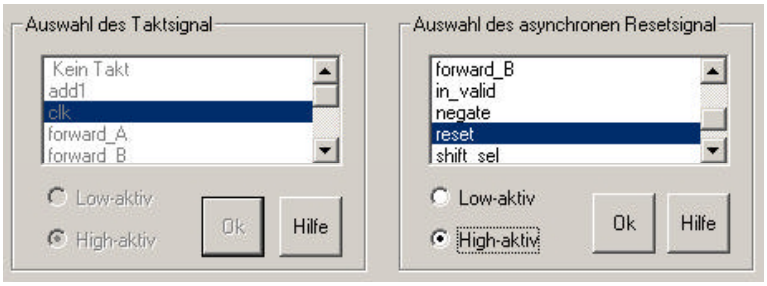


Abbildung 4.7: Ansicht der Felder zur Einstellung der Signalparameter

4.3.4 Weitere Parameter

Nun öffnet sich das letze Feld mit Benutzereingaben (Abbildung 4.8). In diesem Abschnitt können einige zusätzliche Parameter eingestellt werden. Auf der linken Seite des Feldes können semantische Einstellungen zu den Vektoren vorgenommen werden. Alle Signale, die als Vektor in der Entity der VHDL-IP eingetragen sind, können näher spezifiziert werden. Der Nutzer kann die Auswahl treffen, ob der Vektor mit einem Vorzeichen behaftet

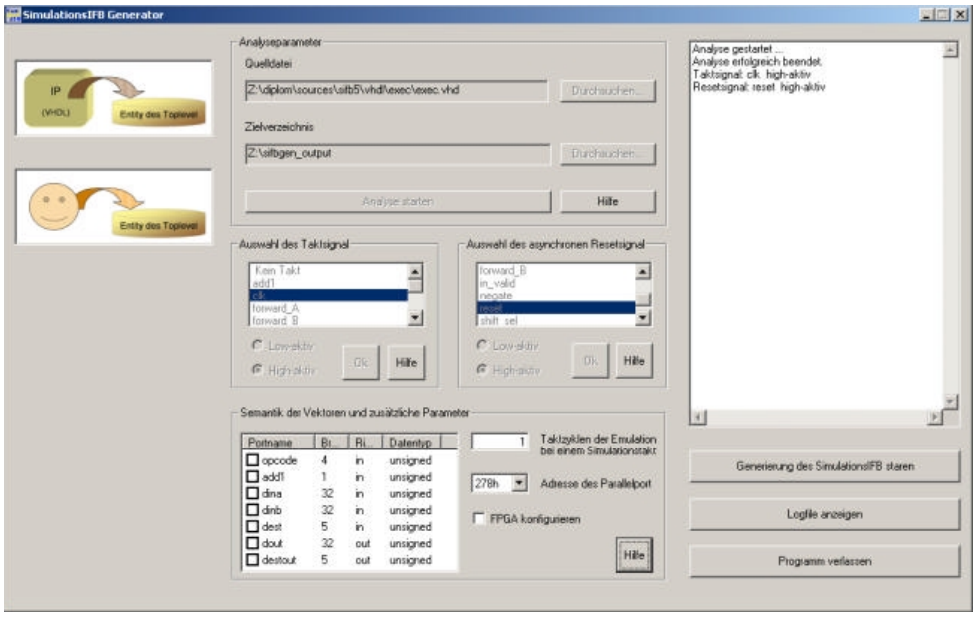


Abbildung 4.8: Der Simulationsinterfaceblock-Generator nach der Einstellung der Signalparameter

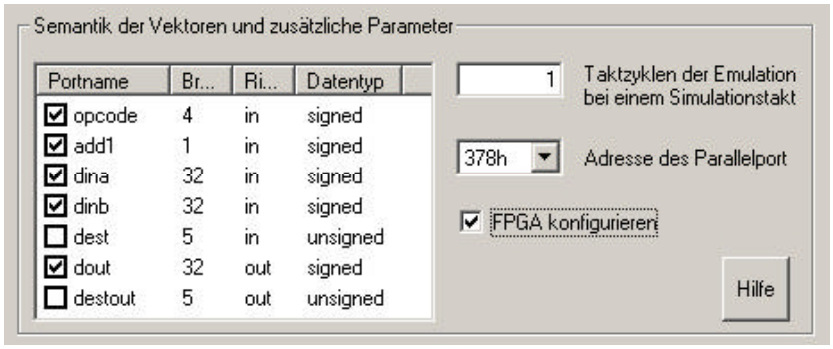


Abbildung 4.9: Detailierte Ansicht des Feldes zur Einstellung der Vektorsemantik und von zusätzlichen Parametern

sein soll oder nicht. Dies erleichtert die spätere Integration des generierten SystemC-Moduls in das SystemC-Design.

Rechts neben der Einstellung der Vektorsemantik befindet sich als erstes die Angabe der Taktzyklenanzahl, die die IP-Emulation in einem SystemC-Simulationstakt durchläuft. Voreingestellt ist der Wert 1. Der maximal mögliche Wert beträgt 65 536. Dieser Parameter kann dazu genutzt werden, um den Emulationstakt auf ein vielfaches des SystemC-Designtaktes einzustellen.

Im zweiten Parameterfeld ist die Auswahl der Adresse der parallelen Schnittstelle möglich. Zulässige Werte sind die Standardadressen 278h, 378h und 3BCh der Schnittstelle. Diese Adresse muss für den PC eingestellt werden, auf dem die SystemC-Simulation ablaufen wird. Es besteht jedoch die Möglichkeit, diesen Parameter nach der Generierung von Hand zu ändern. Die Erklärung folgt im Abschnitt 4.3.6.

Die Checkbox legt fest, ob die FPGA nach der Generierungsphase sofort konfiguriert werden soll, oder ob der Benutzer dies später selbst durchführen möchte. Um die spätere Konfigurierung der FPGA einfacher zu gestalten, wird das Skript zur Konfiguration *do_program.bat* immer erstellt und kann vom Benutzer dazu verwendet werden.

Eine Hilfestellung zu den Parametern kann über den Button Hilfe abgerufen werden. Abbildung 4.9 zeigt das Feld mit eingestellten Parametern.

4.3.5 Die Generierung

Der Button Generierung des SimulationsIFB starten löst die Erstellung des Simulationsinterfaceblocks aus. Als Erstes erfolgt die Generierung der Quelldateien des Hardwareteils (Abbildung 4.10). In diesem Schritt werden alle VHDL-Quelldateien erzeugt, die zur Synthese des Hardwareteils des Interfaceblocks und der Emulation der IP notwendig sind. Darunter fallen alle Dateien im Ausgabeverzeichnis *Hardware/Src* sowei die Projektdatei für die spätere Synthese.

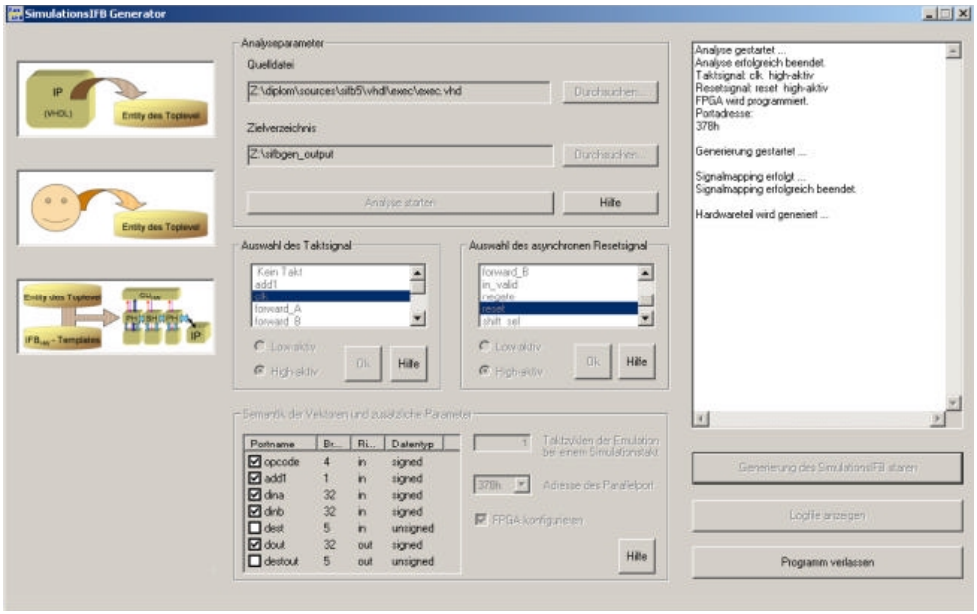


Abbildung 4.10: Der Simulationsinterfaceblock-Generator bei der Generierung des Hardwareteils

Anschließend erstellt das Programm die Skripte für die Synthese, das Übersetzen, das Mapping, die Platzierung und Trassierung, die Generierung des Bitstroms und die Konfiguration des FPGAs. Diese Phase stellt Abbildung 4.11 dar. Die Ausführung der Skripte schließt sich an die Generierung an. Zur erfolgreichen Ausführung der Skripte ist es notwendig, dass die Entwicklungsumgebung Xilinx ISE auf dem Rechner installiert ist, der den Simulationsinterfaceblock-Generator ausführt. Im derzeitigen Zustand unterstützt der Simulationsinterfaceblock-Generator zur Ausführung der Emulation nur das Digilab 2E Development Board [Dig02] mit der darauf befindlichen Xilinx Spartan2E FPGA. Zur Unterstützung anderer FPGAs und Synthesewerkzeuge kann der Simulationsinterfaceblock-Generator später erweitert werden.

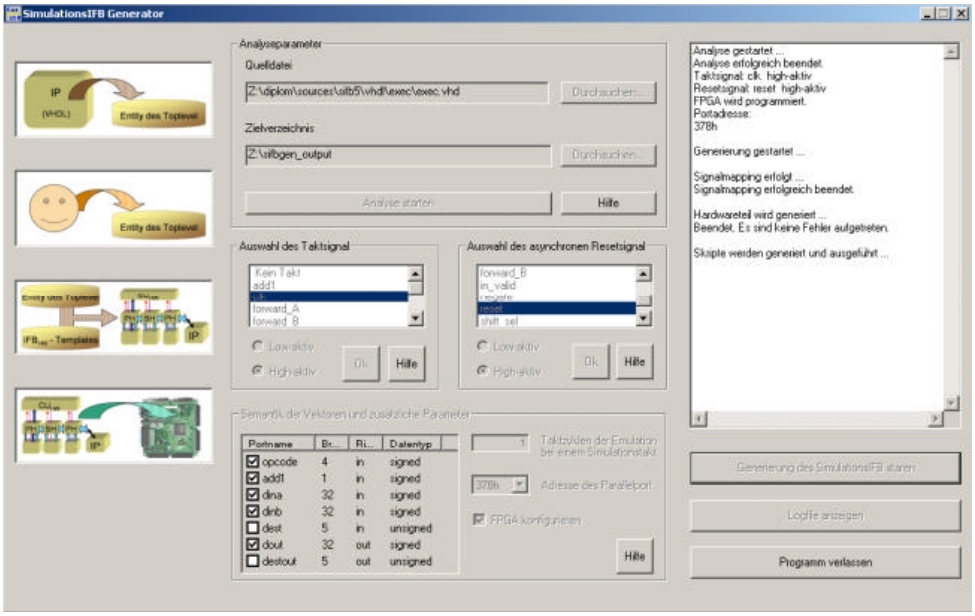


Abbildung 4.11: Der Simulationsinterfaceblock-Generator bei der Generierung der Skripte

Die Generierung des Softwareteils des Simulationsinterfaceblocks schließt sich an die Skriptausführung an (Abbildung 4.12). Sie baut teilweise auf die Synthese des Hardwareteils, speziell auf die Timinganalyse, auf. In dieser Phase erstellt der Simulationsinterfaceblock-Generator das SystemC-Modul aus den analysierten Daten der Entity der VHDL-IP und den Eingaben des Nutzers. Das SystemC-Modul trägt den Namen der Entity der VHDL-IP. Zu diesem Modul gehören noch weitere Bibliotheken, die aus Templates erstellt werden. In den Templates trägt der Simulationsinterfaceblock-Generator nur einige Parameter, wie zum Beispiel die Größe der Simulationsdaten, ein. Nach dem fehlerfreien Abschluss der Generierung ist der Programmlauf beendet.

Das Statusfenster zeigt an, ob Fehler bei den einzelnen Schritten aufgetreten sind. Dies kann der Fall sein, wenn das Programm die Templates nicht fand oder die Ausführung der Skripte fehlschlug. Im letzeren Fall muss der Benutzer sicherstellen, dass das Verzeichnis mit den Xilinxtools in der Pfadumgebung des Betriebssystems vorhanden und die Xilinx Umgebungsvariable gesetzt ist. Tritt trotzdem ein Fehler bei der Synthese auf, so sollte die VHDL-Datei der IP auf mögliche syntaktische Fehler überprüft werden. Darüber, in welchem Abschnitt der Synthese der Fehler auftrat, geben die Logdateien des Syntheseprozesses Auskunft.

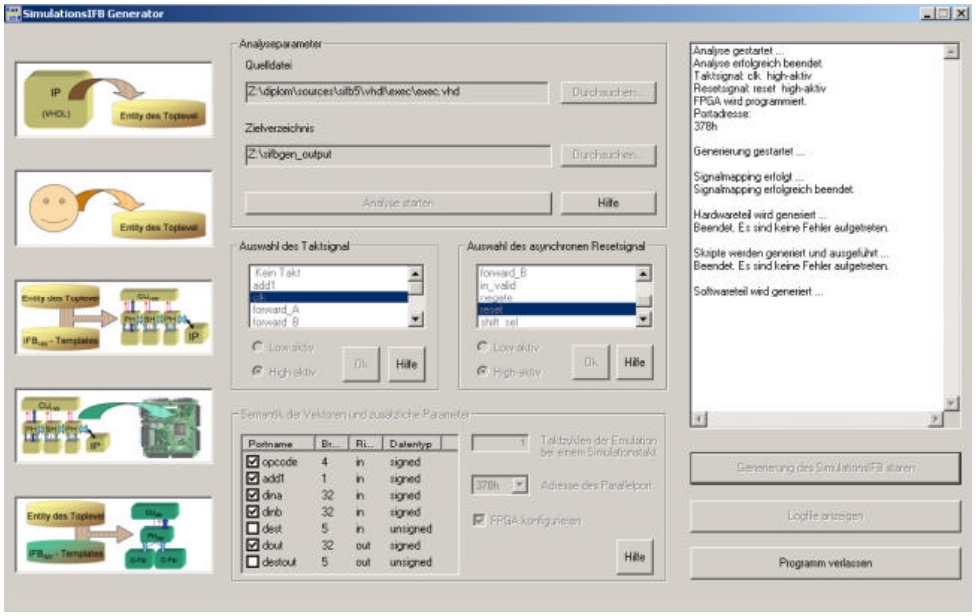


Abbildung 4.12: Der Simulationsinterfaceblock-Generator bei der Generierung des Softwareteils

4.3.6 Überblick über die generierten Dateien

Dieser Abschnitt soll einen Überblick über die generierten Daten des Simulationsinterfaceblock-Generators geben. Die Verzeichnisstrukur stellt Abbildung 4.13 dar. Im Zielverzeichnis befinden sich zunächst die Unterverzeichnisse *Hardware* und *Software* sowie zwei Dateien. Die Datei *simifb-gen.log* ist das Logfile des Simulationsinterfaceblock-Generators und enthält die Statusausgaben und Fehlermeldungen, die während des Programmlaufs auftraten. Es kann auch über den Button Logfile anzeigen aus dem Programm heraus eingesehen werden. Die ausführbare Datei *exec_scripts.bat* ermöglicht es, den Implementierungsvorgang des Hardwareteils zu wiederholen, da sie alle ausgeführten Skripte aufruft.

Im Verzeichnis *Hardware* befinden sich alle Dateien, die zur Erstellung des Hardwareteils des Simulationsinterfaceblocks notwendig sind. Die Aufgaben der darin enthaltenen Dateien führt Tabelle 4.10 auf. Neben den Dateien verfügt das Verzeichnis über Unterverzeichnisse. Sie dienen der geordneten Speicherung der Implementierungsdaten des Hardwareteils des Simulationsinterfaceblocks.

Abbildung 4.13: Überblick über die Verzeichnisstruktur der generierten Daten

Im Folgenden sind die Unterverzeichnisse und eine kurze Beschreibung der enthaltenen Daten aufgeführt:

Bit: Verzeichnis mit den Ausgaben der Bitfile-Generierung. Die Datei mit der Endung *bit* dient zur Konfiguration des FPGA.

Log: Dieses Verzeichnis enthält die Statusausgaben der ausgeführten Skripte. Dabei steht der erste Teil des Dateinamen für die ausgeführte Datei (z.B. xst.log für die Synthese mit xst.exe).

Map: Verzeichnis, dass die Daten aus dem Technologiemapping des Designs enthält.

Ngdbuild: In diesem Verzeichnis befinden sich die Dateien, die durch die Übersetzung des Syntheseergebnisses erstellt wurden.

Par: Die Ergebnisse der Platzierung und Trassierung enthält dieses Verzeichnis.

Src: Die VHDL-Quelldateien des Hardwareteils, die vom Simulationsinterfaceblock-Generator erstellt wurden, speichert dieses Verzeichnis.

Trce: Dieses Verzeichnis sammelt die Ausgabedaten der Timinganalyse.

Xst: Die, durch die Synthese erstellten Daten, fasst das Verzeichnis xst zusammen.

Dateiname	**Aufgabe**
sim_environment_hw.prj	Projektdatei für die Synthese
make.bat	Skript zur Ausführung aller Implementierungsskripte
do_xst.bat	Skript zum Starten der Synthese
xst_script	Hilfsdatei mit Parametern für die Synthese
do_ngdbuild.bat	Skript zum Ausführen des Übersetzungsvorgangs
do_map.bat	Skript für das Mapping des Desings
do_par.bat	Skript zum Ausführen der Platzierung und Trassierung
do_bitgen.bat	Skript zur Erstellung der Konfigurationsdatei für das FPGA
do_program.bat	Skript zum Starten der FPGA-Konfiguration durch das Programm Impact
program_script.cmd	Hilfsskript für Impact
do_trce.bat	Skript zum Durchführen der Timinganalyse

Tabelle 4.10: Überblick zu den Dateien im Verzeichnis Hardware

Das Verzeichnis *Software* enthält alle Quelldateien, die für die Einbindung des Simulationsinterfaceblocks in eine SystemC-Simulation nötig sind. Das SystemC-Modul, welches die Schnittstelle zur SystemC-Simulation bildet, trägt den Namen der Entity der VHDL-IP und die Dateiendung .h. Darin ist die Schnittstelle definiert. Die Funktionalität stellt die gleichnamige Datei mit der Endung .cpp bereit. Alle anderen Dateien im Verzeichnis *Software* enthalten Funktionen, die von diesen Dateien genutzt werden. Dabei ist besonders die Datei *CUsw.h* zu erwähnen. Sie definiert alle wichtigen Parameter für den Softwareteil des Simulationsinterfaceblocks. Dazu zählen die Adresse der parallelen Schnittstelle, die Größe der Simulationsdaten, sowie die Adressen der Kontroll- und Statusregister des Hardwareteils des Simulationsinterfaceblocks .

Die Einbindung der Dateien des Softwareteils in ein bestehendes Design sollte aus dem generierten Verzeichnis erfolgen. Dazu muss beim Aufruf des C/C++-Compilers das Softwareverzeichnis als zusätzlicher Pfad für Includedateien angegeben werden. Für den Linker ist als Bibliothek die Datei *dlportio.lib* sowie das Softwareverzeichnis als weiteres Bibliotheksverzeichnis mit anzugeben. Das hat den Vorteil, dass die Ausgaben des Simulationsinterfaceblock-Generators in einem Verzeichnis gesammelt bleiben. Dies verringert die Gefahr, dass ein Hardwareteil auf das FPGA geladen wird, der nicht zum benutzen Softwareteil gehört und dadurch falsche Simulationsergebnisse erzielt werden.

4.4 Verbesserungsmöglichkeiten und Fazit

Um die Einsatzmöglichkeiten des Simulationsinterfaceblock-Generators zu verbessern, sind einige Erweiterungen im Programm denkbar. In der gegenwärtigen Version verarbeitet der Simulationsinterfaceblock-Generator nur Ports von VHDL-Entities mit den Richtungen in oder out. Zur Unterstützung einer größeren Anzahl von VHDL-IPs könnte das Programm für die Verarbeitung von bidirektionalen Ports (Richtung inout) erweitert werden. Mit der Analysemöglichkeit von VHDL-Generics wird das Konzept von generischen Portbreiten umgesetzt. Damit ist auch eine Unterstützung von parametrisierbaren IPs denkbar.

Die Menge der unterstützen IPs kann weiter steigen, indem Mechanismen in das Programm eingebaut werden, die andere Hardwarebeschreibungssprachen, wie zum Beispiel Verilog, analysieren können. Somit wird eine Adaptierung von IPs in verschiedenen Hardwarebeschreibungssprachen erreicht und das Programm gewinnt an Flexibilität.

Die vorliegenden Version des Simulationsinterfaceblock-Generators schränkt die hardwareseitige Implementierungsplattform noch auf das Digilab 2E Developmentboard ein. Um diese Restriktion aufzuheben, muss das Programm für die Unterstützung anderer FPGA-Typen und -Boards erweitert werden. Damit einher geht die Nutzung von unterschiedlichen Entwicklungsumgebungen und Tools zur Synthese und Implementierung. Da andere FPGA-Boards auch andere phyische Schnittstellen besitzen können, ist zudem eine Unterstützung anderer physischer HW/SW-Schnittstellen zur Überwindung der HW/SW-Grenze denkbar.

Der Simulationsinterfaceblock-Generator wurde entwickelt, um eine VHDL-IP mittels eines Simulationsinterfaceblocks an eine SystemC-Simulation automatisch zu koppeln. Das Programm nimmt dem Nutzer die manuelle Adaptierung der IP-Emlation an die SystemC-Verhaltenssimulation ab, indem es automatisch eine HW/SW-Schnittstelle zwischen beiden Tasks erzeugt. Dadurch sinkt die Zahl der möglichen Fehler, die bei einer manuellen Adaptierung auftreten können und der Zeitaufwand für die Adaptierung verringert sich stark. Das Programm implementiert die Grundfunktionalität der Adaptierung und zeigt, dass eine automatische Lösung praktikabel ist.

5 Demonstrator

Diese Kapitel beschäftigt sich mit dem Demonstrator zu den in dieser Arbeit gewonnenen Erkenntnissen. Im ersten Teil dieses Kapitels wird der allgemeine Nutzen eines Demonstrators herausgestellt. Der zweite Abschnitt beschäftigt sich mit dem Aufbau und der Funktionsweise des verwendeten Demonstrators. Anschließend erfolgt die Vorstellung der Ergebnisse, die mit seiner Hilfe gewonnen wurden. Es folgen einige Betrachtungen zur Simulationszeit des originalen SystemC-Designs und des modifizierten Designs mit der HW/SW-Schnittstelle. Das Ende des Kapitels fasst die Ergebnisse zusammen und zieht daraus Schlussfolgerungen.

5.1 Bedeutung eines Demonstrators

Ein Demonstrator dient dazu, die erarbeiteten theoretischen Konzepte im praktischen Einsatz zu erproben und zu validieren. Mit seiner Hilfe können Schwachstellen und Fehler im theoretischen Modell aufgedeckt und korrigiert werden. Außerdem kann die Praxisrelevanz der Konzepte gezeigt werden. Liegt nur ein theoretisches Modell vor, so besteht die Möglichkeit, dass es sich nicht oder nur schwer in die Praxis umsetzen lässt. Unter Umständen bildet dies eine Hürde für einen praktischen Einsatz des Modells. Auf der anderen Seite kann eine rein praktische Implementierung eventuell nicht allgemein modelliert werden und bleibt damit auf eine spezielle Anwendung beschränkt.

5.2 Aufbau und Funktionsweise

Zur Demonstration des Konzeptes der HW/SW-Schnittstelle und des, im Rahmen dieser Arbeit, erstellten Programms Simulationsinterfaceblock-Generator sollte ein Beispiel gewählt werden, welches die praktische Relevanz der Ergebnisse dieser Arbeit wiederspiegelt. Aus diesem Grund soll nach Möglichkeit auf ein bereits bestehendes Design zurückgegriffen werden und keine reine Selbstentwicklung Verwendung finden.

Als Grundlage des Demonstrators dient ein SystemC-Design aus den Beispielen zur SystemC-Version 2.0.1. Diese Version ist auf der SystemC-Website

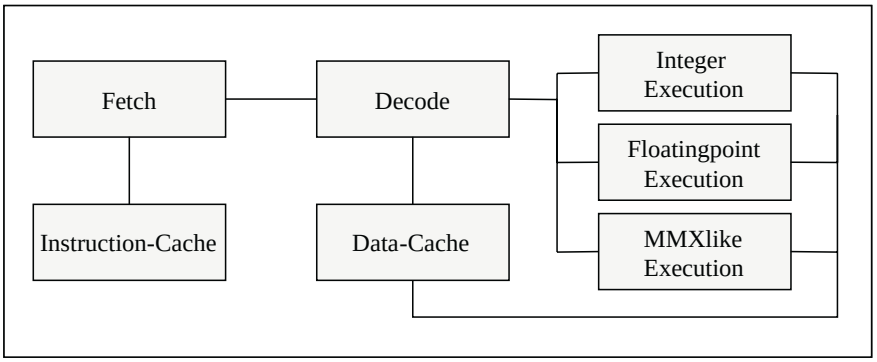


Abbildung 5.1: Aufbau des Designs RISC-CPU

[Opea] frei verfügbar. Das Beispiel beschreibt ein einfaches Design einer RISC-CPU in SystemC. Es wurde von Synopsys® [Syn] entwickelt, um die Fähigkeiten von SystemC bei der HW/SW-Partitionierung zu veranschaulichen. Außerdem kann es als Instruction-set Simulator eingesetzt werden. Das Design findet sich nach dem Entpacken des heruntergeladenen SystemC-Paketes im Verzeichnis */systemc-2.0.1/examples/systemc/risc_cpu*. Es beinhaltet zusätzlich Programme für die CPU zum Testen der Funktionalität. Ein Assembler ist außerdem beigefügt, um selbst Programme zu entwickeln.

Den Aufbau der beschriebenen RISC-CPU stellt Abbildung 5.1 dar. Das Design der RISC-CPU setzt sich zusammen aus einem Instructioncache, einem Datencache, einer Fetcheinheit, die die Instruktionen aus dem Instructioncache liest, einer Decodiereinheit zum Umwandeln der Instruktionen in CPU-Operationen und jeweils einer Ausführungseinheit für Ganzzahlen, Fließkommazahlen und MMX-Berechnungen. Im Folgenden wird dieses SystemC-Design der RISC-CPU als originale RISC-CPU bezeichnet.

Im vorliegenden Szenario (Abbildung 5.2) soll die Ausführungseinheit für ganze Zahlen durch eine VHDL-IP ersetzt werden, die die gleiche Funktionalität bereitstellt. Die anderen Teile des SystemC-Designs, das heißt die Beschreibung des Prozessors und das verwendete Testprogramm bleiben unverändert. Dieses Design wird im Folgenden als modifizierte RISC-CPU bezeichnet. Die VHDL-IP dient als Eingabe für das Programm Simulationsinterfaceblock-Generator. Er generiert daraus die Hardware- und die Softwareseite des Simulationsinterfaceblocks und die Emulation für diese VHDL-IP. Die genaue Funktionsweise der Generierung beschreibt Kapitel 4. Die Implementierung von Hardware-Simulationsinterfaceblock und IP-Emulation dient zur Konfiguration eines Digilab 2E Development Boards. Der Softwareteil

Opcode	Funktion	Verarbeitungsbreite
0000	-	-
0001	a + b + carry	32 Bit
0010	a - b - carry	32 Bit
0011	a + b	32 Bit
0100	a - b	32 Bit
0101	a * b	32 Bit
0110	a / b	16 Bit
0111	a nand b	32 Bit
1000	a and b	32 Bit
1001	a or b	32 Bit
1010	a xor b	32 Bit
1011	not a	32 Bit
1100	a << b	32 Bit
1101	a >> b	32 Bit
1110	a mod b	16 Bit
1111	-	-

Tabelle 5.1: Funktionen der entwickelten Integer-Executionunit

wird anstelle der Integer-Executionunit in das SystemC-Design eingebunden. Ein C++-Compiler erstellt daraus das Simulationsprogramm, welches ein PC, der über die parallele Schnittstelle mit dem FPGA-Board verbunden ist, ausführt.

Das Design der VHDL-IP wurde im Rahmen dieser Arbeit selbst entworfen. Es bildet alle Funktionen nach, die auch die SystemC-Beschreibung der Einheit enthält. Sie sind in Tabelle 5.1 aufgeführt. Nur der Dividierer unterliegt einer Einschränkung in der Verarbeitungsbreite. Dieser Kompromiss musste getroffen werden, da die verwendete Emulationsplattform, das Digilab 2E Developmentboard, nur über eine Xilinx Spartan2e FPGA (XC2S200E-PQ208) verfügt. Ihre Kapazität reicht nicht aus, um die Ausführungseinheit mit einem 32 Bit Paralleldividierer zu emulieren. Deshalb wurde die Verarbeitungsbreite der Division und der damit verbundenen Modulo-Brechnung auf 16 Bit gesenkt.

Diese Einschränkung stellt für den vorliegenden Demonstrator jedoch kein Hindernis dar, da im Testprogramm keine Division ausgeführt wird, die eine höhere Verarbeitungsbreite als 16 Bit beansprucht. Außerdem dient der Demonstrator dem Zweck, die Einsatzfähigkeit und korrekte Funktion des

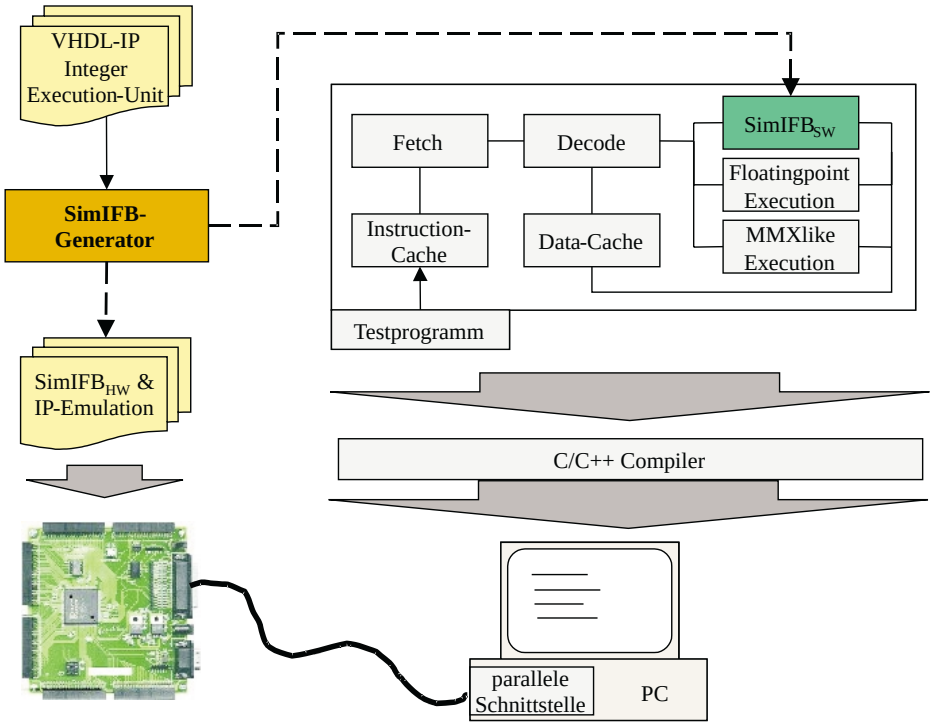


Abbildung 5.2: Aufbau des Demonstrators

Konzeptes des Simulationsinterfaceblocks und des Simulationsinterfaceblock-Generators nachzuweisen und nicht die der VHDL-IP. Im Fall des Simulationsinterfaceblocks bedeutet es den korrekten Austausch der Daten über die HW/SW-Schnittstelle und die erfolgreiche Kopplung der SystemC-Simulation mit der IP-Emulation. Für den Simulationsinterfaceblock-Generator heißt korrekte Funktion, dass der Simulationsinterfaceblock in Abhängigkeit von der VHDL-IP so erstellt wird, dass er seine Aufgabe erfolgreich erfüllen kann.

5.3 Nachweis der Funktion

Um den Nachweis zu erbringen, dass der Simulationsinterfaceblock seine Aufgabe erfüllt, sollen die Simulationsergebnisse der originalen und der modifizierten RISC-CPU miteinander verglichen werden. Dazu ist zunächst die Erstellung der Simulation der originalen RISC-CPU erforderlich. Dies geschieht einfach durch das Kompilieren und Linken der einzelnen SystemC-Module. Als Übersetzungs- und Linkprogramm kommt Microsoft Visual C++ 6.0

```
sc_in<bool>         reset;
sc_in<bool>         in_valid;
sc_in<int>          opcode;
sc_in<bool>         negate;
sc_in<int>          add1;
sc_in<bool>         shift_sel;
sc_in<signed int>   dina;
sc_in<signed int>   dinb;
sc_in<bool>         forward_A;
sc_in<bool>         forward_B;
sc_in<unsigned>     dest;
sc_out<bool>        C;
sc_out<bool>        V;
sc_out<bool>        Z;
sc_out<signed int>  dout;
sc_out<bool>        out_valid;
sc_out<unsigned>    destout;
sc_in_clk           CLK;
```

Abbildung 5.3: Schnittstellendefinition der originalen Integer-Executionunit. Der Datentyp bool gibt einfache Signale an. Int, signed int und unsigned geben Vektoren an.

zum Einsatz. Hinweise zur Nutzung von SystemC in dieser Entwicklungsumgebung beschreibt Anhang B.

Die Erstellung der modifizierten RISC-CPU läuft nach Abbildung 5.2 ab. Zunächst wird das Programm Simulationsinterfaceblock-Generator gestartet. Ihm dient als Eingabe die VHDL-IP der Integer-Executionunit. Das Programm analysiert sie. Anschließend erfolgt die Festlegung der Signalparameter. Das Taktsignal *clk* und das asynchrone Resetsignal *rst* sind dabei high-aktiv. Die Datentypen der Vektoren werden entsprechend den Datentypen der originalen Ausführungseinheit (Abbildung 5.3) festgelegt. Abbildung 5.4 zeigt die Einstellungen im Programm. Die Generierung des Simulationsinterfaceblocks für die VHDL-IP der Integer-Executionunit stellt den letzen Schritt der Programmausführung dar.

Der Hardwareteil des Simulationsinterfaceblocks und die Implementierung der VHDL-IP befinden sich nun auf dem FPGA-Board. Sie sind bereit für die Ausführung. Um die Simulation der originalen und der modifizierten RISC-CPU besser vergleichen zu können, wird im Softwareteil des Simulationsinterfaceblocks Quellcode, zur Ausgabe der Simulationsdaten auf dem Bild-

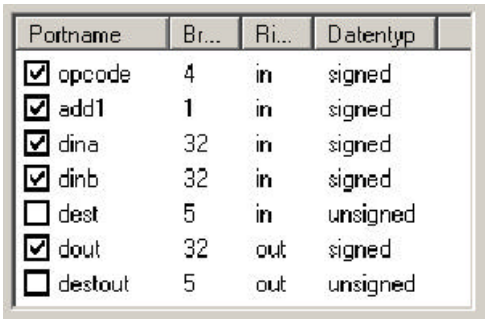


Abbildung 5.4: Festlegung der Datentypen im Simulationsinterfaceblock-Generator

schirm, manuell hinzugefügt. Dies erfolgt in der generierten Datei *exec.cpp*. Als Quelle dient hier die Datei *exec.cpp* der originalen RISC-CPU. Die anderen generierten Dateien bleiben unverändert.

Aus dem Design der originalen RISC-CPU wird die Integer-Executionunit entfernt und durch das generierte Modul ersetzt. Zur Übersetzung des Gesamtdesigns muss noch die Datei *dlportio.lib* zu den verwendeten Bibliotheken hinzugefügt werden. Außerdem ist als zusätzlicher Includepfad das Verzeichnis mit den generierten Quelldateien einzutragen. Nach der Durchführung dieser Schritte erfolgt das Übersetzen und Linken der modifizierten RISC-CPU. Auch hierbei kommt Microsoft Visual C++ 6.0 zum Einsatz.

Die Ausführung der Simulationen erzeugt deren Simulationsergebnisse, die auf dem Bildschirm ausgegeben werden. Die Ergebnisse sind in Anhang C auszugsweise gegenübergestellt. Ihre Gegenüberstellung zeigt, dass die Simulation der originalen und die der modifizierten RISC-CPU zu gleichen Simulationszeiten die gleichen Eingabe- und Ausgabedaten besitzen. Damit ist nachgewiesen, dass das Verhalten der beiden Designs übereinstimmt.

Nachdem sichergestellt ist, dass beide Designs über das gleiche Verhalten verfügen, soll der Nachweis erfolgen, dass der Simulationsinterfaceblock seine Aufgaben korrekt erfüllt. Seine Aufgaben lauten nach Abschnitt 3.1.1 wie folgt:

1. zwei Komponenten kommunizieren lassen, ohne dass an den Komponenten Änderungen durchgeführt werden müssen.

2. den bidirektionalen Datentransport zwischen den Komponenten sicherstellen.

3. die HW/SW-Grenze intern überwinden.
4. die Möglichkeit bieten, Daten intern zu transformieren.
5. die Integration einer Steuerung erlauben.

Am Demonstrator wurde gezeigt, dass die Simulationsdaten erfolgreich zwischen der SystemC-Simulation und der IP-Emulation in beide Richtungen ausgetauscht werden. Das zeigt, dass der Datentransport bidirektional über die HW/SW-Grenze möglich ist. Somit sind die Aufgaben 2 und 3 erfüllt. Da die Simulationsdaten für den Transport durch den Simulationsinterfaceblock intern umgewandelt werden und der Transport erfolgreich verläuft, ist auch die korrekte Erfüllung der 4. Aufgabe nachgewiesen. Die interne Steuerung, die Aufgabe 5 als Inhalt hat, arbeitet nach der Spezifikation, da sie den Datenaustausch steuert und dieser erfolgreich verläuft. Zum Nachweis der 1. Aufgabe müssen die kommunizierenden Tasks näher betrachtet werden. Das SystemC-Design bliebt, bis auf die Ersetzung der Integer-Executionunit, unverändert. Auch an der VHDL-Beschreibung der IP erfolgten keine Veränderungen. Da beide Komponenten durch den Simulationsinterfaceblock erfolgreich miteinander kommunizieren, gilt auch die 1. Aufgabe als erfüllt.

Den Simulationsinterfaceblock erzeugte der Simulationsinterfaceblock-Generator. Der Nachweis, dass der Simulationsinterfaceblock die ihm gestellten Aufgaben erfolgreich verrichtet, zeigt somit, dass er korrekt generiert wurde. Die Aufgabe, die die automatische Adaptierung der IP-Emulation auf dem FPGA mit der SystemC-Verhaltenssimulation durch einen Simulationsinterfaceblock beinhaltet, erfüllt der Simulationsinterfaceblock-Generator somit korrekt.

5.4 Betrachtungen zur Simulationszeit

Bei der Simulation spielt die benötigte Zeit eine große Rolle. Deshalb soll an dieser Stelle die Simulationszeit der originalen und der modifizierten RISC-CPU betrachtet werden.

Zu erwarten ist, dass die Zeit, welche die Simulation der modifizierten RISC-CPU beansprucht, größer ist, als die Simulationszeit der originalen RISC-CPU. Die Vermutung begründet sich darauf, dass bei der Simulation der modifizierten RISC-CPU zusätzlich Zeit für die Kommunikation zwischen

SystemC-Simulation und IP-Emulation, sowie für die interne Kommunikation des Simulationsinterfaceblocks aufgewendet werden muss.

Daraus folgt, dass die Geschwindigkeit der physischen HW/SW-Schnittstelle einen wesentlichen Faktor für die Simulationszeit darstellt. Die Untersuchung ihrer Geschwindigkeit stellt sich deshalb als erste Aufgabe (Abschnitt 5.4.1). Anschließend wird die Simulationszeit, unter Berücksichtigung der gewonnenen Erkenntnisse zur HW/SW-Schnittstelle, betrachtet.

5.4.1 Geschwindigkeit der parallelen Schnittstelle

Für die Messung der Übertragungsgeschwindigkeit der parallelen Schnittstelle mit dem EPP-Protokoll wurde im Rahmen dieser Arbeit eine Messeinrichtung entwickelt. Sie setzt sich aus dem Softwareprogramm **Speedtest_sw** und dem VHDL-Design für das Digilab 2E Developmentboard **Speedtest_hw** zusammen. Beide Teile sind auf der beiliegenden CD-ROM zu finden (siehe Anhang D).

Das Programm Speedtest_sw hat die Aufgabe, fortlaufend Daten über die parallele Schnittstelle zu schreiben oder zu lesen. Für den Programmaufruf müssen zwei Parameter angegeben werden. Zu Einem der Arbeitsmodus und zum Anderen die Adresse der parallelen Schnittstelle im PC. Das Programm besitzt sechs verschiedene Modi, die jeweils eine andere Form der Datenübertragung durch das EPP-Protokoll ausführen. Ihre Funktionen sind in Tabelle 5.2 aufgeführt. Die Modi 5 und 6 realisieren ein abwechselndes Schreiben

Modus	**Funktion**
m1	Datenbyte schreiben
m2	Adressbyte schreiben
m3	Datenbyte lesen
m4	Adressbyte lesen
m5	Adressbyte schreiben, Datenbyte schreiben
m6	Adressbyte schreiben, Datenbyte lesen

Tabelle 5.2: Die Modi der Software zur Messung der Übertragungsgeschwindigkeit der parallelen Schnittstelle im EPP-Modus

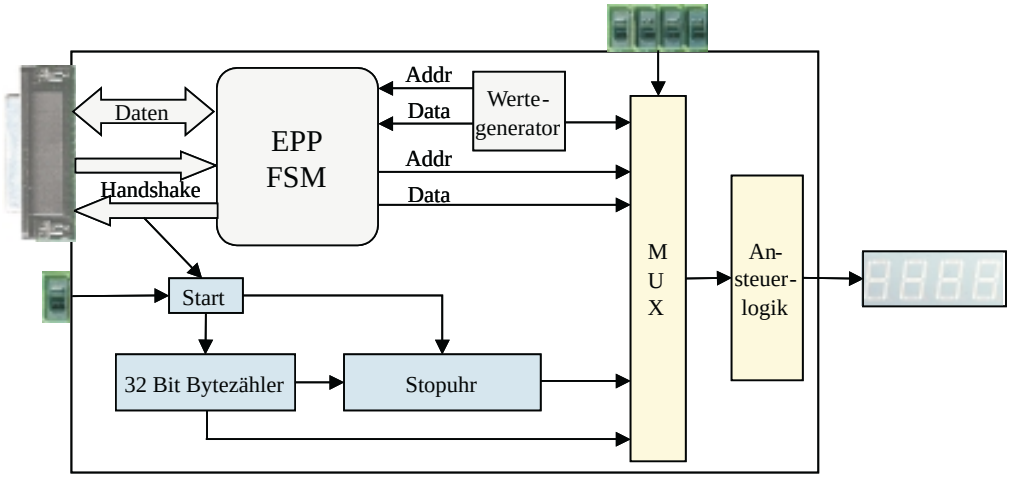


Abbildung 5.5: Aufbau der Hardware zur Geschwindigkeitsmessung

eines Adressbytes und Schreiben bzw. Lesen eines Datenbytes. Diese Formen der Datenübertragung sind für den Simulationsinterfaceblock besonders wichtig, da bei ihm jedem Datenbyte auch eine Adresse zugeordnet ist.

Den Aufbau des Hardwareteils zeigt Abbildung 5.5. Zur Messung der Übertragungsgeschwindigkeit des EPP-Protokolls dienen ein 32 Bit-Zähler und eine Stopuhr. Beide werden über ein gemeinsames Startsignal aktiviert. Der Bytezähler reagiert auf das Signal *nWait* des EPP-Protokolls. In jedem Übertragungszyklus ändert das Signal zweimal seinen Wert, einmal von high zu low und einmal umgekehrt. In jedem Übertragunszyklus wird dabei genau ein Byte übertragen. Die Stopuhr beginnt die Zeitmessung, sobald der Bytezähler einen Wert ungleich Null annimmt. Sie kann bis zu 1 000 Sekunden zählen. Ihre Genauigkeit hängt von der eingesetzten Taktfrequenz ab. Im vorliegenden Fall liegt eine Taktung mit 50 Mhz vor. Daraus ergibt sich eine maximale Genauigkeit von 20 ns.

Zur Ausgabe der Daten dient eine 7-Segmentanzeige. Die Auswahl der Daten übernimmt ein Multiplexer. Es können die ausgehenden sowie die eingehenden Adressen und Daten angezeigt werden. Jedoch fällt ihnen für die Messung keine Bedeutung zu. Die wichtigen Daten bilden die Anzahl der übertragenen Bytes und die dafür benötigte Zeit. Auch sie können auf der Anzeige ausgegeben werden. Das Ausgabeformat ist hexadezimal.

Die Messungen der Geschwindigkeit erfolgen für die Modi 5 und 6. Modus 5 schreibt abwechselnd ein Adressbyte und ein Datenbyte auf die parallele

Modus	übertragene Byte	s	ms	μs	ns	Byte/s	Durchschnittsgeschwindigkeit
m5	18 860 504	30	543	789	500	617 460	618 404 Byte/s
	18 892 543	30	530	303	600	618 813	
	18 976 578	30	660	073	340	618 940	
m6	19 450 840	30	323	718	040	641 440	641 881 Byte/s
	19 517 029	30	355	897	920	642 940	
	19 529 100	30	454	054	240	641 264	

Tabelle 5.3: Ermittelte Geschwindigkeiten der parallelen Schnittstelle

Schnittstelle. Er nutzt dazu die Funktion *Write_EPP*, die auch für die Datenübertragung im Simulationsinterfaceblock zum Einsatz kommt. Modus 6 schreibt ein Adressbyte und liest ein Datenbyte. Er verwendet die Funktion *Read_EPP*. Tabelle 5.3 stellt die Ergebnisse der Messungen dar.

Die Übertragungsgeschwindigkeit, die für die parallelen Schnittstelle mit dem EPP-Protokoll angegeben ist, beträgt nach [Pea04] zwischen 500 kByte/s und 2 MByte/s. Dies ergibt eine Zeit von 2 000 ns bis 500 ns für die Übertragung eines Bytes. In dieser Messung lag der Wert für reine Schreiboperationen bei durchschnittlich 618 404 Byte/s (1617 ns/Byte). Bei gemischten Schreib-Lese-Operationen erreichte die Schnittstelle eine Übertragungsgeschwindigkeit von durchschnittlich 641 881 Byte/s (1558 ns/Byte). Die Werte siedeln sich zwar im unteren Bereich der theoretisch angegebenen Geschwindigkeit an, liegen aber im spezifizierten Bereich.

5.4.2 Vergleich der Simulationszeiten

Um eine vergleichende Messung durchzuführen, müssen die gleichen Ausgangsbedingungen für beide Simulationen geschaffen werden. Sie werden auf dem gleichen PC (Pentium4 mit 2,4 Ghz, 512 MB RAM, Betriebssystem Windows XP Professional) ausgeführt. Für beide Simulationen kommt der gleiche Testbench zum Einsatz. Außerdem wird in den Modulen *exec.cpp* der Code zur Bildschirmausgabe auskommentiert. Das generierte Modul der modifizierten RISC-CPU wird in jedem Simulationsschritt einmal vollständig abgearbeitet. Die Ausführung des Originalmodul hingegen findet Abschnittsweise mit Hilfe von *wait()*-Anweisungen statt. Aus diesem Grund gibt das generierte Modul mehr Daten auf dem Bildschirm aus. Da Ausgabeopera-

Simulation	Zeit	durchschnittliche Zeit
originale RISC-CPU	0,313 s 0,344 s 0,328 s	0,328 s
modifizierte RISC-CPU	0,391 s 0,359 s 0,406 s	0,385 s

Tabelle 5.4: Gemessene Simulationszeiten der originalen und der modifizierten RISC-CPU. Es wurden je 3 Messungen durchgeführt und der Mittelwert daraus gebildet.

tionen auf der Standardausgabe aber viel Zeit beanspruchen, können sie die Messergebnisse stark verfälschen.

Die Zeitmessung für beide Simulationen erfolgt durch das Programm *timedrun.exe*, das im Rahmen dieser Arbeit erstellt wurde. Es führt zwei Skripte aus. Als erstes wird das Skript *copy_simulation.bat* ausgeführt, welches alle für die Simulation benötigten Dateien in das aktuelle Verzeichnis kopiert. Für die Ausführung des zweiten Skriptes *run.bat* wird die Zeit gestoppt. Dieses Skript dient dazu, die Simulation aufzurufen und die Ausgaben in eine Datei umzuleiten. Die benötigte Zeit wird der Datei *time_taken_by_run.log* als letzter Eintrag angefügt.

Die Ergebnisse der Messung stellt Tabelle 5.4 dar. Die Simulation der originalen RISC-CPU benötigte eine durchschnittliche Zeit von 328 Millisekunden. Die Simulationszeit der modifizierten RISC-CPU liegt mit durchschnittlich 385 Millisekunden etwas darüber. Nun stellt sich die Frage, wodurch die um 57 Millisekunden größere Simulationszeit zu erklären ist.

Einen Unterschied zwischen der originalen und der modifizierten RISC-CPU stellt die Kommunikation über die HW/SW-Schnittstelle dar. Zunächst soll das Kommunikationsaufkommen und die dafür benötigte Zeit untersucht werden. Tabelle 5.5 enthält die Befehle, die in einem Simulationsschritt abgearbeitet werden. Befehle, die in Schleifen ausgeführt werden, sind mit Sternen gekennzeichnet. Die Anzahl der Schleifendurchläufe soll zunächst betrachtet werden, bevor wieder auf die Kommunikationszeit eingegangen wird.

Die erste Schleife dient zum sicheren Starten der IP-Emulation und besteht aus den Funktionen *SetStartSim* und *GetStatus1*. Sie wird in jedem Fall einmal durchlaufen. Im Normalfall endet die Schleife sofort nach dem ersten

Funktion	**Kategorie (write/read)**	**Anzahl Bytes (write/read)**
GetStatus1	read	1/3
SetWriteMode	write	2/0
SetLDIVCounter	write	2/0
SetHDIVCounter	write	2/0
SetCycleCounter	write	4/0
WriteSimData	write	20/0
SetWaitMode	write	2/0
SetStartSim*	write	2/0
GetStatus1*	read	1/3
GetStatus1**	read	1/3
SetCycleEnd	write	2/0
SetReadMode	write	2/0
ReadSimData	read	6/18
SetWaitMode	write	2/0

Tabelle 5.5: Das Datenaufkommen des generierten Moduls. Die Kategorie gibt an, ob es sich um eine Schreib- oder einen Lesefunktion handelt. Sterne hinter Funktionen bedeuten, dass sie in einer Schleife ausgeführt werden.

Durchlauf. Dies folgt aus der Übertragungszeit, die für ein Byte notwendig ist. Nach Abschnitt 5.4.1 beträgt die theoretisch kürzeste Zeit zur Übertragung eines Bytes über die parallele Schnittstelle 500 ns. Der Simulationsinterfaceblock ist mit 50 Mhz getaktet und besitzt demnach eine Taktperiode von 20 ns. Bevor die Funktion *GetStatus1* das Statusbyte liest, schreibt sie eine Adresse. Das heißt vom Ende der Funktion *SetStartSim* bis zum Lesen des Statusbyte vergehen mindestens 500 ns. Der Automat hat mindestens $500 : 20 = 25$ Taktperioden Zeit, um den Zustandswechsel auszuführen. Betrachtete man den Aufbau des Automaten in Abbildung 3.23 auf Seite 58 benötigt er einen Takt für den Zustandswechsel. Hinzu kommen jeweils ein Takt für die Wertübernahme durch die Kontrollregister und Statusregister. Dies zeigt, dass die Schleife unter normalen Bedingungen nur einmal Durchlaufen wird.

Zur Betrachtung der zweiten Schleife muss die Ausführungszeit eines IP-Emulationstaktes mit herangezogen werden. Die Frequenz der IP-Emulation berechnet sich nach Formel A.14 in Anhang A. Sie lautet

$$f_{IP} = \frac{f_{IFB}}{HDIV + LDIV + 2}$$

Die berechneten Werte des Simulationsinterfaceblock-Generators für HDIV und LDIV eingesetzt, ergibt sich die Frequenz der IP-Emulation von

$$f_{IP} = \frac{50\,\text{Mhz}}{6+6+2} = 3,571\,\text{Mhz}.$$

Damit beträgt die Zeit für eine Taktperiode 280 ns. Die Emulation startet sofort nach dem erfolgreichen Setzen des Startsignals durch *SetStartSim*. Anschließend wird die Funktion *GetStatus1* ausgeführt, die das Starten überprüft. Sie benötigt einen Schreibvorgang und drei Lesevorgänge. Drei Lesevorgänge wurden aus Gründen der Fehlerkorrektur eingeführt, da bei der Schnittstelle vereinzelt Lesefehler auftraten. Die drei empfangenen Bytes werden miteinander verglichen und das Byte zurüchgegeben, welches am häufigsten auftritt. Die Funktion besitzt damit eine Ausführungszeit von mindestens $4 \cdot 500\,\text{ns} = 2000\,\text{ns}$. Das bedeutet, dass die zweite Schleife startet, wenn die IP-Emulation schon abgeschlossen ist. Damit benötigt auch sie nur einen Durchlauf.

Nach der Untersuchung der Schleifendurchläufe soll wieder auf die Kommunikationszeit eingegangen werden. Schreibvorgänge im Simulationsinterfaceblock sind definiert als Schreiben eines Adressbytes und Schreiben eines Datenbytes. In jedem Simulationschritt müssen damit mindestens 36 Bytes übertragen werden. Ein Lesevorgang umfasst das Schreiben eines Adressbytes und Lesen von drei Datenbytes aus Gründen der Fehlerkorrektur. Die Lesevorgänge besitzen somit ein Kommunikationsaufkommen von 40 Bytes je Simulationsschritt. Die vorliegende Simulation besteht, nach den Simulationsausgaben in der Datei *risc_cpu.log* (siehe Anhang D), aus 278 Schritten inklusive der Initialisierung. Als Übertragungsgeschwindigkeit werden die gemessenen Werte aus Tabelle 5.3 herangezogen. Für die Kommunikationszeit ergibt sich

$$\frac{36\,\text{Byte}}{618404\,\text{Byte/s}} \cdot 278 + \frac{40\,\text{Byte}}{641881\,\text{Byte/s}} \cdot 278 = 0,0335\,\text{s} = 33,5\,\text{ms}$$

Damit lässt sich ein Teil der 57 Millisekunden Zeitunterschied zwischen der Simulation der originalen und der modifizierten RISC-CPU erklären.

Ein weitere Faktor, der die höhere Simulationszeit erklären kann, ist das Signalmapping in der Datei *exec.cpp*. In jedem Simulationsschritt müssen die Eingangssignale bitweise auf das Feld für die Simulationsdaten abgebildet werden. Eine Abbildung der empfangenen Daten von der IP-Emulation auf die lokalen Variablen findet in jedem Simulationsschritt statt. Dies kostet zusätzlich Zeit.

Im Vergleich zum generierten Modul hat das originale Modul sehr wenig Anweisungen auszuführen. Aus diesem Grund besitzt es einen Vorteil hinsichtlich der Simulationszeit. Es ist wahrscheinlich, dass der jetzige Geschwindigkeitsvorteil der Originaldesigns mit zunehmender Komplexität der SystemC-Beschreibung verloren gehen würde. In diesem Fall kann der Simulationsinterfaceblock auch zur Beschleunigung der SystemC-Simulation eingesetzt werden.

Bei der Durchführung der Messung ist nicht vorhersagbar, ob der Simulation über die gesamte Laufzeit die komplette CPU-Leistung zur Verfügung steht. Auf dem simulierenden PC werden weitere Prozesse des Betriebssystems ausgeführt. Falls das Betriebssystem während der Ausführung dem Simulator kurz den Prozessor entzieht, kommen Umschaltzeiten zwischen den Prozessen zur Simulationszeit hinzu. Damit erklären sich auch die Schwankungen bei den Messergebnissen der Simulationszeit.

Die Betrachtungen zeigen, dass die Simulationsgeschwindigkeit bei der Nutzung eines Simulationsinterfaceblocks stark von der Anzahl der ausgetauschten Daten zwischen SystemC-Simulation und IP-Emulation abhängt. Die HW/SW-Schnittstelle kann hier die Simulation ausbremsen. Das FPGA ist im vorliegenden Beispiel leistungsfähig genug, um nicht geschwindigkeitsmindernd zu wirken. Zur Erhöhung der Simulationsgeschwindigkeit ist der Einsatz einer physischen HW/SW-Schnittstelle denkbar, die eine höhere Übertragungsgeschwindigkeit als die parallele Schnittstelle erlaubt. In diesem Zusammenhang sollte dann auch die Leistungsfähigkeit der FPGA neu untersucht werden und gegebenenfalls ein schnelleres Modell eingesetzt werden, um ein Gleichgewicht zwischen Schnittstellengeschwindigkeit und FPGA-Leistung zu erreichen.

5.5 Schlussfolgerungen

Am vorgestellten Demonstrator wurde nachgewiesen, dass das Modell des Simulationsinterfaceblocks leistungsfähig genug ist, um die ihm gestellten Aufgaben korrekt zu erfüllen. Der Simulationsinterfaceblock wurde erfolgreich eingesetzt, um eine SystemC-Simulation mit einer IP-Emulation über eine HW/SW-Schnittstelle zu verbinden ohne dabei Änderungen an den Komponenten vorzunehmen.

In diesem Zusammenhang erfolgte außerdem der Nachweis, dass der Simulationsinterfaceblock-Generator erfolgreich arbeitet. Das Programm erstellte den

Simulationsinterfaceblock auf Basis einer Analyse der VHDL-IP und Nutzereingaben. Dieser automatisch generierte Simulationsinterfaceblock erfüllte die Aufgabe der Verbindung von SystemC-Simulation und IP-Emulation zur Zufriedenheit.

Der Einsatz eines bereits bestehenden SystemC-Designs als Teil des Demonstrators zeigt, dass die Leistung des Simulationsinterfaceblocks und des Simulationsinterfaceblock-Generators ausreicht, um auch im praktischen Einsatz Verwendung zu finden.

6 Zusammenfassung und Ausblick

Diese Kapitel bildet den Abschluss der Arbeit. Es fasst die Konzepte und Ergebnisse dieser Arbeit zusammen und erläutert daran die erfolgreiche Bearbeitung der Aufgabenstellung. Anschließend wird ein Ausblick auf Möglichkeiten der Weiterentwicklung geboten.

6.1 Zusammenfassung der Arbeit

Diese Arbeit untergliedert sich in zwei Aufgaben. Die erste Aufgabe bestand in der Untersuchung eines Co-Simulationsansatzes. Dieser Ansatz zielt darauf ab, eine synthesefähige VHDL-Komponente auf einem FPGA zu emulieren und die Emulation mit einer SystemC-Simulation zu koppeln. Die zweite Aufgabe ist eng mit der Ersten verknüpft. Dabei sollte aufbauend auf den Betrachtungen des Co-Simulationsansatzes ein Verfahren entwicklet werden, das eine automatische Adaptierung von SystemC-Simulation und IP-Emulation erlaubt.

Der erste Teil dieser Arbeit bot eine Einführung in die Problematik und stellte die Aufgabenstellung vor. Anschließend erfolgte die Betrachtung des aktuellen Standes der Techniken mit denen diese Arbeit verknüfpt ist. Daran schloss sich die Vorstellung von SystemC als Entwurfssystem und die Begutachtung seiner derzeitige Leistungsfähigkeit an. Der Begriff der Intellectual Properties wurde erklärt und das IPQ-Projekt kurz vorgestellt, mit dessen Hilfe bestehende Probleme bei der IP-Nutzung überwunden werden sollen. Zuletzt wurde auf den Begriff Simulation eingegangen und verschiedene Simulationsverfahren und -konzepte vorgestellt.

Das dritte Kapitel betrachtete den Co-Simulationsansatz. Dabei wurde herausgearbeitet, dass eine Hardware/Software-Schnittstelle benötigt wird, um die SystemC-Simulation mit der IP-Emulation zu koppeln. In diesem Zusammenhang erfolgte die Herausstellung der Aufgaben der HW/SW-Schnittstelle sowie der von ihr zu erbringenden Leistungen. Für eine mögliche Umsetzung der HW/SW-Schnittstelle fand das Modell des Interfaceblocks Verwendung.

Dazu wurde zunächst desses Leistungsfähigkeit analysiert, um sicherzustellen, dass es in der Lage ist, die Aufgaben zu erfüllen. Da das vorliegende Modell des Interfaceblocks eine Aufgabe nicht abdecken konnte, erfolgte eine Erweiterung das Konzept. Das Ergebnis bestand in einem erweiterten Interfaceblock, der die Hardware/Software-Grenze überwinden kann. Um den erweiterten Interfaceblock für die Co-Simulation einsetzen zu können, wurden Untersuchungen zur praktischen Umsetzung durchgeführt. Das Ergebnis bestand im Simulationsinterfaceblock, der eine Möglichkeit der praktischen Realisierung des erweiterten Interfaceblocks darstellt. Zum Abschluss des Kapitels wurden noch die verschiedenen Einsatzmöglichkeiten des Simulationsinterfaceblocks vorgestellt.

Das vierte Kapitel beschäftigte sich mit einer Lösung, die eine automatische Erstellung eines Simulationsinterfaceblocks erlaubt. Das Ergebnis bildet das Prorgamm Simulationsinterfaceblock-Generator. Es wurde detailiert auf die interne Arbeitsweise des Programms eingegangen, um eine gute Grundlage für Weiterentwicklungen des Programms zu bieten. Für die Anwendung des Programms zur Generierung von Simulationsinterfaceblöcken wurde ein Handbuch verfasst, um Benutzer einen schnellen Einstieg zu gewähren. Abschließend erfolgte die Vorstellung von Möglichkeiten der Weiterentwicklung zur Unterstützung eines breiteren Spektrums von Implementierungsplattformen und Intellectual Properties.

Der Demonstrator, über den das fünfte Kapitel handelt, diente dem Erproben und Validieren der Konzepte aus Kapitel drei und des Programm aus Kapitel vier. Es wurde der Aufbau und die Funktionsweise des Demonstrators vorgestellt. Anschließend wurde der Nachweis der korrekten Funktion durch die Ausführung des Demonstrators und dem Vergleich mit dem Ursprungsdesign erbracht. Da benötigte Zeit einer Simulation ein wichtiges Kriterium bei der Wahl der Simulationsart darstellt, schlossen sich dem Nachweis der Funktion Untersuchungen zur Simulationsgeschwindigkeit an. In diesem Zusammenhang wurde eine Messeinrichtung entwickelt, die die Geschwindigkeit der parallelen Schnittstelle ermittelte. Die Untersuchungen ergaben, dass die Simulationsgeschwindigkeit stark von der Geschwindigkeit der physischen HW/SW-Schnittstelle abhängt. Das Ergebnis des Demonstrators bestand darin, dass die erarbeiteten Konzepte dieser Arbeit ihre Aufgaben korrekt erfüllten und erfolgreich in der Praxis eingesetzt werden konnten. Auch das Prorgamm Simulationsinterfaceblock-Generator erfüllte seinen Zweck und erlaubt die automatische Adaptierung einer IP-Emulation und einer SystemC-Simulation. Anhand des Demonstrators wurde gezeigt, dass die gestellten Aufgaben dieser Arbeit erfolgreich gelöst wurden.

6.2 Ausblick für den erweiterten Interfaceblock

Abgeleitet vom Konzept des erweiterten Interfaceblocks wurde in dieser Arbeit die Implementierung eines Simulationsinterfaceblocks durchgeführt. Er dient zur Kopplung einer SystemC-Simulation auf einem PC und einer IP-Emulation auf einem FPGA. Im Bezug auf die Cosimulation könnten weiterer Arbeiten folgende Themen untersuchen:

- Unter Nutzung eines Simulationsinterfaceblocks könnte ein Tool für das Hardwaredebugging implementiert werden. Dies bietet die Möglichkeit, das Verhalten eines Hardwaredesigns als Implementierung auf einem FPGA zu untersuchen.
- Ein weiteres Feld für Untersuchungen liegt in der Emulation von dynamisch rekonfigurierbaren Designs. Dabei kann die Kopplung mit einer SystemC-Simulation erfolgen, um dynamisch rekonfigurierbare Teile eines größeren Designs zu untersuchen oder ein Hardwaredebugging des dynamisch rekonfigurierbaren Designs durchzuführen.

Um die Einsatzmöglichkeiten des allgemeinen Konzeptes des erweiterten Interfaceblocks zu erweitern, könnten Untersuchungen auf den folgenden Gebieten behilflich sein:

- Ein Gebiet von Untersuchung könnte der Einsatz als Schnittstelle zwischen Anwendungsprogrammen und Spezialhardware sein. Der Einsatz von Spezialhardware beschleunigt dabei bestimmte Funktionen, deren Ausführung in Software zu langsam ist, um eine Steigerung der Programmperformance zu erreichen.
- Weiterhin könnte das Konzept des erweiterten Interfaceblocks in das bestehende Interface Synthese (IFS)-Format aufgenommen werden und damit auch die Möglichkeit einer automatischen Generierung eines erweiterten Interfaceblock als HW/SW-Schnittstelle entstehen.

6.3 Ausblick für den Simulationsinterfaceblock-Generator

Die Entwicklung des Simulationsinterfaceblock-Generators im Rahmen dieser Arbeit fand speziell für die automatische Adaptierung von SystemC-Simulation und VHDL-IP-Emulation statt. Um ein größeres Einsatzfeld des Programms zu erlangen und damit die praktische Einsatzfähigkeit weiter zu steigern, sind einige Erweiterungen des Programms denkbar:

- In der gegenwärtigen Version verarbeitet der Simulationsinterfaceblock-Generator nur Ports von VHDL-Entities mit den Richtungen in oder out. Zur Unterstützung einer größeren Anzahl von VHDL-IPs könnte das Programm für die Verarbeitung von bidirektionalen Ports (Richtung inout) erweitert werden.
- Mit der Analysemöglichkeit von VHDL-Generics kann das Konzept von generischen Portbreiten umgesetzt werden. Damit ist auch eine Unterstützung von parametrisierbaren IPs denkbar.
- Die Erweiterung des Programms zur Unterstützung anderer Hardwarebeschreibungssprachen, wie zum Beispiel Verilog, würde die Einsatzbreite erheblich steigern.
- Eine Unterstützung unterschiedlicher FPGA-Typen und -Boards bringt eine größere Flexibilität des Programms hinsichtlich der Emulationsplattform mit sich. In diesem Zusammenhang steht die Nutzung von unterschiedlichen Entwicklungsumgebungen und Tools zur Synthese und Implementierung.
- Andere phyische Hardware/Software-Schnittstellen können zum Einen die Simulationsgeschwindigkeit erhöhen. Auf der anderen Seite erhöht sich dadurch ebenfalls die Flexibilität gegenüber unterschiedlichen Implementierungsplattformen.

A Herleitung der Formel zur Berechnung der Taktfrequenz der IP-Emulation

Zur Herleitung der Formel zur Berechnung der Taktfrequenz der IP-Emulation sind zunächst eine Variablendefinitionen durchzuführen. Die verwendeten Variablen definieren sich wie folgt:

f_{IFB}	...	Taktfrequenz des Interfaceblocks
T_{IFB}	...	Taktperiode des Interfaceblocks
f_{IP}	...	Taktfrequenz der IP-Emulation
T_{IP}	...	Taktperiode der IP-Emulation
t_{IP_H}	...	Zeit des Highpegels der Taktperiode der IP-Emulation
t_{IP_L}	...	Zeit des Lowpegels der Taktperiode der IP-Emulation
$HDIV$	...	Wert des Zählers HDIV
$LDIV$	...	Wert des Zählers LDIV
n_H	...	Anzahl der Durchläufe durch den Zustand Clk_gen1
n_L	...	Anzahl der Durchläufe durch den Zustand Clk_gen0

Der Interfaceblock ist mit der Taktfrquenz f_{IFB} getaktet. Die Taktperiode ergibt sich aus der Taktfrequenz nach der Formel

$$T = \frac{1}{f} \tag{A.1}$$

Somit ergibt sich für die Taktperiode des Interfaceblocks

$$T_{IFB} = \frac{1}{f_{IFB}} \tag{A.2}$$

Der endliche Automat in der $\mathrm{CU_{HW}}$ generiert den Highpegel als auch den Lowpegel des Taktsignals für die IP-Emulation mindestens einen Takt lang. Daraus folgt für die minimalen Pegelzeiten der Taktperiode der IP-Emulation

$$t_{IP_{H,min}} = T_{IFB} \qquad t_{IP_{L,min}} = T_{IFB} \tag{A.3}$$

Die Taktgenerierung erfolgt in den Zuständen *Clk_gen1* und *Clk_gen0* des Automaten der $\mathrm{CU_{HW}}$. Die Verweildauer in diesen Zuständen geben die Werte der Zähler HDIV und LDIV an. Aber jeder dieser Zustände wird mindestens einmal durchlaufen. Die Anzahl der Durchläufe dieser Zustände ergibt sich aus

$$n_H = HDIV + 1 \qquad n_L = LDIV + 1 \tag{A.4}$$

Die Anzahl der Durchläufe bestimmt, für welche Zeitspanne der jeweilige Pegel generiert wird. Die Zeitspanne beträgt für den jeweiligen Pegel aber mindestens $t_{IP_{H,min}}$ bzw. $t_{IP_{L,min}}$. Für die Gesamtzeit des High- bzw. Lowpegels ergibt sich

$$t_{IP_H} = n_H \cdot t_{IP_{H,min}} \qquad t_{IP_L} = n_L \cdot t_{IP_{L,min}} \tag{A.5}$$
$$t_{IP_H} = (HDIV + 1) \cdot t_{IP_{H,min}} \qquad t_{IP_L} = (LDIV + 1) \cdot t_{IP_{L,min}} \tag{A.6}$$

Aus den Formeln A.3 und A.6 bildet sich

$$t_{IP_H} = (HDIV + 1) \cdot T_{IFB} \qquad t_{IP_L} = (LDIV + 1) \cdot T_{IFB} \tag{A.7}$$

Die Dauer der Taktperiode für die IP-Emulation ergibt sich aus der Zeit des Highpegels und der Zeit des Lowpegels.

$$T_{IP} = T_{IP_H} + T_{IP_L} \tag{A.8}$$
mit A.7: $$T_{IP} = (HDIV + 1) \cdot T_{IFB} + (LDIV + 1) \cdot T_{IFB} \tag{A.9}$$
$$T_{IP} = (HDIV + 1 + LDIV + 1) \cdot T_{IFB} \tag{A.10}$$
$$T_{IP} = (HDIV + LDIV + 2) \cdot T_{IFB} \tag{A.11}$$
$$\tag{A.12}$$

Nach Formel A.1 werden die Taktperioden in Formel A.11 durch Frequenzen ersetzt

$$\frac{1}{f_{IP}} = (HDIV + LDIV + 2) \cdot \frac{1}{f_{IFB}} \tag{A.13}$$
$$f_{IP} = \frac{f_{IFB}}{HDIV + LDIV + 2} \tag{A.14}$$

B Hinweise zur Nutzung von SystemC in Mircrosoft Visual C++ 6.0

Um die SystemC Klassenbibliothek erfolgreich in der C/C++ Entwicklungsumgebung *Microsoft Visual C++ 6.0* einsetzen zu können, müssen einige Einstellungen vorgenommen werden. Sie betreffen den Compiler und den Linker und sind unter dem Menüpunkt `Project -> Settings...` vorzunehmen.

Die Einstellungen sind im Folgenden aufgeführt. Dabei ist als erstes die Registerkarte aufgeführt, unter dem sich die Einstellung findet. Es schließt sich der Unterpunkt und die jeweilige Parameter an. Der Wert, der für den Parameter einzutragen ist, steht darunter mit einer Bemerkung in Klammern.

```
C/C++ -> C++ Language -> Enable Run-Time Type Information (RTTI)
      aktiv (Haken eingetragen)

C/C++ -> Preprocessor -> Additional include directories
      ...\Systemc-2.0.1\src ( Pfad zu systemc.h)

Link -> General -> Object/library modules
      systemc.lib (hinzufügen zu vorhandenen Bibliotheken)

Link -> Input -> Additional library path
      ...\Systemc-2.0.1\msvc60\systemc\Debug
      (Pfad zur Datei systemc.lib)
```

Weitere Hinweise zur Arbeit mit SystemC enthält das Handbuch [Opeb].

C Gegenüberstellung der Simulationsdaten

Die Simulationsergebnisse der originalen und modifizierten RISC-CPU werden hier auszugsweise gegenübergestellt. Bei den Ausgaben der originalen RISC-CPU wurden die Ausgaben, die keine Simulationsdaten enthielten, entfernt, um eine bessere Übersicht zu gewährleisten. Zur direkten Gegenüberstellung sind die Simulationsausgaben der modifizierten RISC-CPU ebenfalls gekürzt aufgeführt. Die vollständigen Ausgabedaten finden sich auf der dieser Arbeit beiliegenden CD- ROM. Unter den Verzeichnisssen */Testdesigns/Risc_cpu_original/Simulation* sowie */Testdesigns/Risc_cpu_simifb/Simulation* existiert jeweils eine Datei *risc_cpu.log*, die die Daten enthält. Mehr zum Inhalt der Verzeichnisse bietet Anhang D, der das Inhaltsverzeichnis der CD-ROM beinhaltet.

```
          Ausgaben der                |           Ausgaben der
       originalen RISC-CPU            |   RISC-CPU mit SimulationsIFB
-------------------------------------|-----------------------------------
                                      |
** ALERT ** ID: initialize Archite   | ** ALERT ** ID: initialize Archite
                ctural Registers      |                 ctural Registers
** ALERT ** BIOS: initialize BIOS    | ** ALERT ** BIOS: initialize BIOS
** ALERT ** DCU: initialize Data     | ** ALERT ** DCU: initialize Data
                 Cache                |                  Cache
-----------------------              | -----------------------
IFU : mem=0xf000001                  | IFU : mem=0xf000001
IFU : pc= 1 at CSIM 5 ns             | IFU : pc= 1 at CSIM 5 ns
-------------------------------      | -------------------------------
ID: R0=R0(=0)                        | ID: R0=R0(=0)
  : at CSIM 7 ns                     |   : at CSIM 7 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= 0 B= 0                | ALU : op= 3 A= 0 B= 0
ALU : R= 0-> R0 at CSIM 9 ns         | ALU : R= 0-> R0 at CSIM 9 ns
------------------------             | ------------------------
IFU : mem=0xf000002                  | IFU : mem=0xf000002
IFU : pc= 2 at CSIM 12 ns            | IFU : pc= 2 at CSIM 12 ns
-------------------------------      | -------------------------------
ID: R0=R0(=0)                        | ID: R0=R0(=0)
  : at CSIM 14 ns                    |   : at CSIM 14 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= 0 B= 0                | ALU : op= 3 A= 0 B= 0
```

```
ALU : R= 0-> R0 at CSIM 16 ns        | ALU : R= 0-> R0 at CSIM 16 ns
-------------------------            | -------------------------
IFU : mem=0xf000003                  | IFU : mem=0xf000003
IFU : pc= 3 at CSIM 19 ns            | IFU : pc= 3 at CSIM 19 ns
-------------------------------      | -------------------------------
ID: R0=R0(=0)                        | ID: R0=R0(=0)
  : at CSIM 21 ns                    |   : at CSIM 21 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= 0 B= 0                | ALU : op= 3 A= 0 B= 0
ALU : R= 0-> R0 at CSIM 23 ns        | ALU : R= 0-> R0 at CSIM 23 ns
-------------------------            | -------------------------
IFU : mem=0xf000004                  | IFU : mem=0xf000004
IFU : pc= 4 at CSIM 26 ns            | IFU : pc= 4 at CSIM 26 ns
-------------------------------      | -------------------------------
ID: R0=R0(=0)                        | ID: R0=R0(=0)
  : at CSIM 28 ns                    |   : at CSIM 28 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= 0 B= 0                | ALU : op= 3 A= 0 B= 0
ALU : R= 0-> R0 at CSIM 30 ns        | ALU : R= 0-> R0 at CSIM 30 ns
-------------------------------      | -------------------------------
                                     |
ICU ALERT: ******AFTER RESET******   | ICU ALERT: ******AFTER RESET******
ICU ALERT: **********************    | ICU ALERT: **********************
-------------------------------      | -------------------------------
ID: R0=0x0(0) fr ALU at CSIM 31 ns   | ID: R0=0x0(0) fr ALU at CSIM 31 ns
-------------------------            | -------------------------
IFU : mem=0xf160000a                 | IFU : mem=0xf160000a
IFU : pc= 5 at CSIM 33 ns            | IFU : pc= 5 at CSIM 33 ns
-------------------------------      | -------------------------------
ID: R6=10 at CSIM 35 ns              | ID: R6=10 at CSIM 35 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= 10 B= 0               | ALU : op= 3 A= 10 B= 0
ALU : R= 10-> R6 at CSIM 37 ns       | ALU : R= 10-> R6 at CSIM 37 ns
-------------------------------      | -------------------------------
ID: R6=0xa(10) fr ALU at CSIM 38 ns  | ID: R6=0xa(10) fr ALU at CSIM 38 ns
-------------------------            | -------------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 6 at CSIM 40 ns            | IFU : pc= 6 at CSIM 40 ns
-------------------------------      | -------------------------------
                                     |
******************************       | ******************************
ID: REGISTERS DUMP at CSIM 42 ns     | ID: REGISTERS DUMP at CSIM 42 ns
******************************       | ******************************
REG :==============================  | REG :==============================
  R 0(00000000)    R 1(00000001)     |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)     |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
```

```
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
-------------------------            | -------------------------
IFU : mem=0xf160000a                 | IFU : mem=0xf160000a
IFU : pc= 7 at CSIM 47 ns            | IFU : pc= 7 at CSIM 47 ns
-------------------------------      | -------------------------------
ID: R6=10 at CSIM 49 ns              | ID: R6=10 at CSIM 49 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= 10 B= 0               | ALU : op= 3 A= 10 B= 0
ALU : R= 10-> R6 at CSIM 51 ns       | ALU : R= 10-> R6 at CSIM 51 ns
-------------------------------      | -------------------------------
ID: R6=0xa(10) fr ALU at CSIM 52 ns  | ID: R6=0xa(10) fr ALU at CSIM 52 ns
-------------------------            | -------------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 8 at CSIM 54 ns            | IFU : pc= 8 at CSIM 54 ns
-------------------------------      | -------------------------------
                                     |
******************************       | ******************************
ID: REGISTERS DUMP at CSIM 56 ns     | ID: REGISTERS DUMP at CSIM 56 ns
******************************       | ******************************
REG :============================    | REG :============================
  R 0(00000000)    R 1(00000001)     |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)     |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
```

```
                                  |
------------------------          | ------------------------
IFU : mem=0xf160000a              | IFU : mem=0xf160000a
IFU : pc= 9 at CSIM 61 ns         | IFU : pc= 9 at CSIM 61 ns
------------------------------    | ------------------------------
ID: R6=10 at CSIM 63 ns           | ID: R6=10 at CSIM 63 ns
------------------------------    | ------------------------------
ALU : op= 3 A= 10 B= 0            | ALU : op= 3 A= 10 B= 0
ALU : R= 10-> R6 at CSIM 65 ns    | ALU : R= 10-> R6 at CSIM 65 ns
------------------------------    | ------------------------------
ID: R6=0xa(10) fr ALU at CSIM 66 ns | ID: R6=0xa(10) fr ALU at CSIM 66 ns
------------------------          | ------------------------
IFU : mem=0x0                     | IFU : mem=0x0
IFU : pc= a at CSIM 68 ns         | IFU : pc= a at CSIM 68 ns
------------------------------    | ------------------------------
                                  |
******************************    | ******************************
ID: REGISTERS DUMP at CSIM 70 ns  | ID: REGISTERS DUMP at CSIM 70 ns
******************************    | ******************************
REG :============================ | REG :============================
  R 0(00000000)    R 1(00000001)  |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)  |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)  |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)  |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)  |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)  |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)  |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)  |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)  |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)  |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)  |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)  |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)  |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)  |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)  |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)  |   R30(00000030)    R31(00000031)
================================= | =================================
                                  |
------------------------          | ------------------------
IFU : mem=0xf160000a              | IFU : mem=0xf160000a
IFU : pc= b at CSIM 75 ns         | IFU : pc= b at CSIM 75 ns
------------------------------    | ------------------------------
ID: R6=10 at CSIM 77 ns           | ID: R6=10 at CSIM 77 ns
------------------------------    | ------------------------------
ALU : op= 3 A= 10 B= 0            | ALU : op= 3 A= 10 B= 0
ALU : R= 10-> R6 at CSIM 79 ns    | ALU : R= 10-> R6 at CSIM 79 ns
------------------------------    | ------------------------------
ID: R6=0xa(10) fr ALU at CSIM 80 ns | ID: R6=0xa(10) fr ALU at CSIM 80 ns
------------------------          | ------------------------
```

```
IFU : mem=0x0                          | IFU : mem=0x0
IFU : pc= c at CSIM 82 ns              | IFU : pc= c at CSIM 82 ns
-------------------------------       | -------------------------------
                                       |
*******************************        | *******************************
ID: REGISTERS DUMP at CSIM 84 ns       | ID: REGISTERS DUMP at CSIM 84 ns
*******************************        | *******************************
REG :============================      | REG :============================
  R 0(00000000)    R 1(00000001)       |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)       |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)       |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)       |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)       |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)       |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)       |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)       |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)       |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)       |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)       |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)       |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)       |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)       |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)       |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)       |   R30(00000030)    R31(00000031)
================================       | ================================
                                       |
------------------------               | ------------------------
IFU : mem=0xf160000a                   | IFU : mem=0xf160000a
IFU : pc= d at CSIM 89 ns              | IFU : pc= d at CSIM 89 ns
-------------------------------       | -------------------------------
ID: R6=10 at CSIM 91 ns                | ID: R6=10 at CSIM 91 ns
-------------------------------       | -------------------------------
ALU : op= 3 A= 10 B= 0                 | ALU : op= 3 A= 10 B= 0
ALU : R= 10-> R6 at CSIM 93 ns         | ALU : R= 10-> R6 at CSIM 93 ns
-------------------------------       | -------------------------------
ID: R6=0xa(10) fr ALU at CSIM 94 ns    | ID: R6=0xa(10) fr ALU at CSIM 94 ns
------------------------               | ------------------------
IFU : mem=0x0                          | IFU : mem=0x0
IFU : pc= e at CSIM 96 ns              | IFU : pc= e at CSIM 96 ns
-------------------------------       | -------------------------------
                                       |
*******************************        | *******************************
ID: REGISTERS DUMP at CSIM 98 ns       | ID: REGISTERS DUMP at CSIM 98 ns
*******************************        | *******************************
REG :============================      | REG :============================
  R 0(00000000)    R 1(00000001)       |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)       |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)       |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)       |   R 6(0000000a)    R 7(fcf0fdef)
```

```
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
------------------------             | ------------------------
IFU : mem=0xf160000a                 | IFU : mem=0xf160000a
IFU : pc= f at CSIM 103 ns           | IFU : pc= f at CSIM 103 ns
-------------------------------      | -------------------------------
ID: R6=10 at CSIM 105 ns             | ID: R6=10 at CSIM 105 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= 10 B= 0               | ALU : op= 3 A= 10 B= 0
ALU : R= 10-> R6 at CSIM 107 ns      | ALU : R= 10-> R6 at CSIM 107 ns
-------------------------------      | -------------------------------
ID: R6=0xa(10) fr ALU at CSIM 108 ns | ID: R6=0xa(10) fr ALU at CSIM 108 ns
------------------------             | ------------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 10 at CSIM 110 ns          | IFU : pc= 10 at CSIM 110 ns
-------------------------------      | -------------------------------
                                     |
*******************************      | *******************************
ID: REGISTERS DUMP at CSIM 112 ns    | ID: REGISTERS DUMP at CSIM 112 ns
*******************************      | *******************************
REG :================================| REG :================================
  R 0(00000000)    R 1(00000001)     |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)     |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
```

```
================================   | ================================
                                   |
------------------------           | ------------------------
IFU : mem=0xf1500005               | IFU : mem=0xf1500005
IFU : pc= 11 at CSIM 117 ns        | IFU : pc= 11 at CSIM 117 ns
-------------------------------    | -------------------------------
ID: R5=5 at CSIM 119 ns            | ID: R5=5 at CSIM 119 ns
-------------------------------    | -------------------------------
ALU : op= 3 A= 5 B= 0              | ALU : op= 3 A= 5 B= 0
ALU : R= 5-> R5 at CSIM 121 ns     | ALU : R= 5-> R5 at CSIM 121 ns
-------------------------------    | -------------------------------
ID: R5=0x5(5) fr ALU at CSIM 122 ns| ID: R5=0x5(5) fr ALU at CSIM 122 ns
------------------------           | ------------------------
IFU : mem=0x0                      | IFU : mem=0x0
IFU : pc= 12 at CSIM 124 ns        | IFU : pc= 12 at CSIM 124 ns
-------------------------------    | -------------------------------
                                   |
*******************************    | *******************************
ID: REGISTERS DUMP at CSIM 126 ns  | ID: REGISTERS DUMP at CSIM 126 ns
*******************************    | *******************************
REG :==============================| REG :==============================
  R 0(00000000)    R 1(00000001)   |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)   |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)   |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)   |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)   |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)   |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)   |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)   |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)   |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)   |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)   |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)   |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)   |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)   |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)   |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)   |   R30(00000030)    R31(00000031)
================================   | ================================
                                   |
------------------------           | ------------------------
IFU : mem=0xf550000                | IFU : mem=0xf550000
IFU : pc= 13 at CSIM 131 ns        | IFU : pc= 13 at CSIM 131 ns
-------------------------------    | -------------------------------
ID: R5=R5(=5)                      | ID: R5=R5(=5)
  : at CSIM 133 ns                 |   : at CSIM 133 ns
-------------------------------    | -------------------------------
ALU : op= 3 A= 5 B= 0              | ALU : op= 3 A= 5 B= 0
ALU : R= 5-> R5 at CSIM 135 ns     | ALU : R= 5-> R5 at CSIM 135 ns
-------------------------------    | -------------------------------
```

```
ID: R5=0x5(5) fr ALU at CSIM 136 ns  | ID: R5=0x5(5) fr ALU at CSIM 136 ns
-------------------------            | -------------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 14 at CSIM 138 ns          | IFU : pc= 14 at CSIM 138 ns
-------------------------------      | -------------------------------
                                     |
*******************************      | *******************************
ID: REGISTERS DUMP at CSIM 140 ns    | ID: REGISTERS DUMP at CSIM 140 ns
*******************************      | *******************************
REG :=============================   | REG :=============================
  R 0(00000000)    R 1(00000001)     |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)     |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
-------------------------            | -------------------------
IFU : mem=0xf550000                  | IFU : mem=0xf550000
IFU : pc= 15 at CSIM 145 ns          | IFU : pc= 15 at CSIM 145 ns
-------------------------------      | -------------------------------
ID: R5=R5(=5)                        | ID: R5=R5(=5)
  : at CSIM 147 ns                   |   : at CSIM 147 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= 5 B= 0                | ALU : op= 3 A= 5 B= 0
ALU : R= 5-> R5 at CSIM 149 ns       | ALU : R= 5-> R5 at CSIM 149 ns
-------------------------------      | -------------------------------
ID: R5=0x5(5) fr ALU at CSIM 150 ns  | ID: R5=0x5(5) fr ALU at CSIM 150 ns
-------------------------            | -------------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 16 at CSIM 152 ns          | IFU : pc= 16 at CSIM 152 ns
-------------------------------      | -------------------------------
                                     |
*******************************      | *******************************
ID: REGISTERS DUMP at CSIM 154 ns    | ID: REGISTERS DUMP at CSIM 154 ns
*******************************      | *******************************
REG :=============================   | REG :=============================
  R 0(00000000)    R 1(00000001)     |   R 0(00000000)    R 1(00000001)
```

```
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)     |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================  | ==================================
                                     |
------------------------            | ------------------------
IFU : mem=0xf550000                  | IFU : mem=0xf550000
IFU : pc= 17 at CSIM 159 ns          | IFU : pc= 17 at CSIM 159 ns
-------------------------------     | -------------------------------
ID: R5=R5(=5)                        | ID: R5=R5(=5)
  : at CSIM 161 ns                   |   : at CSIM 161 ns
-------------------------------     | -------------------------------
ALU : op= 3 A= 5 B= 0                | ALU : op= 3 A= 5 B= 0
ALU : R= 5-> R5 at CSIM 163 ns       | ALU : R= 5-> R5 at CSIM 163 ns
-------------------------------     | -------------------------------
ID: R5=0x5(5) fr ALU at CSIM 164 ns  | ID: R5=0x5(5) fr ALU at CSIM 164 ns
------------------------            | ------------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 18 at CSIM 166 ns          | IFU : pc= 18 at CSIM 166 ns
-------------------------------     | -------------------------------
                                     |
*******************************     | *******************************
ID: REGISTERS DUMP at CSIM 168 ns    | ID: REGISTERS DUMP at CSIM 168 ns
*******************************     | *******************************
REG :===============================  | REG :===============================
  R 0(00000000)    R 1(00000001)     |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)     |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
```

```
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
------------------------             | ------------------------
IFU : mem=0xf550000                  | IFU : mem=0xf550000
IFU : pc= 19 at CSIM 173 ns          | IFU : pc= 19 at CSIM 173 ns
-------------------------------      | -------------------------------
ID: R5=R5(=5)                        | ID: R5=R5(=5)
  : at CSIM 175 ns                   |   : at CSIM 175 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= 5 B= 0                | ALU : op= 3 A= 5 B= 0
ALU : R= 5-> R5 at CSIM 177 ns       | ALU : R= 5-> R5 at CSIM 177 ns
-------------------------------      | -------------------------------
ID: R5=0x5(5) fr ALU at CSIM 178 ns  | ID: R5=0x5(5) fr ALU at CSIM 178 ns
------------------------             | ------------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 1a at CSIM 180 ns          | IFU : pc= 1a at CSIM 180 ns
-------------------------------      | -------------------------------
                                     |
******************************       | ******************************
ID: REGISTERS DUMP at CSIM 182 ns    | ID: REGISTERS DUMP at CSIM 182 ns
******************************       | ******************************
REG :============================    | REG :============================
  R 0(00000000)    R 1(00000001)     |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)     |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
------------------------             | ------------------------
IFU : mem=0xf550000                  | IFU : mem=0xf550000
IFU : pc= 1b at CSIM 187 ns          | IFU : pc= 1b at CSIM 187 ns
-------------------------------      | -------------------------------
ID: R5=R5(=5)                        | ID: R5=R5(=5)
```

```
  : at CSIM 189 ns                  |   : at CSIM 189 ns
-------------------------------     | -------------------------------
ALU : op= 3 A= 5 B= 0               | ALU : op= 3 A= 5 B= 0
ALU : R= 5-> R5 at CSIM 191 ns      | ALU : R= 5-> R5 at CSIM 191 ns
-------------------------------     | -------------------------------
ID: R5=0x5(5) fr ALU at CSIM 192 ns | ID: R5=0x5(5) fr ALU at CSIM 192 ns
------------------------            | ------------------------
IFU : mem=0x0                       | IFU : mem=0x0
IFU : pc= 1c at CSIM 194 ns         | IFU : pc= 1c at CSIM 194 ns
-------------------------------     | -------------------------------
                                    |
*******************************     | *******************************
ID: REGISTERS DUMP at CSIM 196 ns   | ID: REGISTERS DUMP at CSIM 196 ns
*******************************     | *******************************
REG :===============================| REG :===============================
  R 0(00000000)    R 1(00000001)    |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)    |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000005)    |   R 4(00000004)    R 5(00000005)
  R 6(0000000a)    R 7(fcf0fdef)    |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)    |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)    |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)    |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)    |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)    |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)    |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)    |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)    |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)    |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)    |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)    |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)    |   R30(00000030)    R31(00000031)
================================    | ================================
                                    |
------------------------            | ------------------------
IFU : mem=0x2550001                 | IFU : mem=0x2550001
IFU : pc= 1d at CSIM 201 ns         | IFU : pc= 1d at CSIM 201 ns
-------------------------------     | -------------------------------
ID: R5= R5(=5)+1                    | ID: R5= R5(=5)+1
  : at CSIM 203 ns                  |   : at CSIM 203 ns
-------------------------------     | -------------------------------
ALU : op= 3 A= 5 B= 1               | ALU : op= 3 A= 5 B= 1
ALU : R= 6-> R5 at CSIM 205 ns      | ALU : R= 6-> R5 at CSIM 205 ns
-------------------------------     | -------------------------------
ID: R5=0x6(6) fr ALU at CSIM 206 ns | ID: R5=0x6(6) fr ALU at CSIM 206 ns
------------------------            | ------------------------
IFU : mem=0x0                       | IFU : mem=0x0
IFU : pc= 1e at CSIM 208 ns         | IFU : pc= 1e at CSIM 208 ns
-------------------------------     | -------------------------------
                                    |
```

```
*****************************      | *****************************
ID: REGISTERS DUMP at CSIM 210 ns  | ID: REGISTERS DUMP at CSIM 210 ns
*****************************      | *****************************
REG :============================  | REG :============================
  R 0(00000000)    R 1(00000001)   |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)   |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000006)   |   R 4(00000004)    R 5(00000006)
  R 6(0000000a)    R 7(fcf0fdef)   |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)   |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)   |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)   |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)   |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)   |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)   |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)   |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)   |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)   |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)   |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)   |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)   |   R30(00000030)    R31(00000031)
==================================  | ==================================
                                    |
-------------------------          | -------------------------
IFU : mem=0x1156fffa                | IFU : mem=0x1156fffa
IFU : pc= 1f at CSIM 215 ns         | IFU : pc= 1f at CSIM 215 ns
-------------------------------    | -------------------------------
ID: bne R5(=6), R6(=10), pc+=(-6).  | ID: bne R5(=6), R6(=10), pc+=(-6).
ID: at CSIM 217 ns                  | ID: at CSIM 217 ns
-------------------------------    | -------------------------------
ALU : op= 3 A= 0 B= 0               | ALU : op= 3 A= 0 B= 0
ALU : R= 0-> R0 at CSIM 219 ns      | ALU : R= 0-> R0 at CSIM 219 ns
-------------------------------    | -------------------------------
ID: R0=0x0(0) fr ALU at CSIM 220 ns | ID: R0=0x0(0) fr ALU at CSIM 220 ns
-------------------------          | -------------------------
IFU : mem=0x0                       | IFU : mem=0x0
IFU : pc= 20 at CSIM 222 ns         | IFU : pc= 20 at CSIM 222 ns
-------------------------          | -------------------------
IFU ALERT: **BRANCH**               | IFU ALERT: **BRANCH**
-------------------------------    | -------------------------------
                                    |
*****************************      | *****************************
ID: REGISTERS DUMP at CSIM 224 ns  | ID: REGISTERS DUMP at CSIM 224 ns
*****************************      | *****************************
REG :============================  | REG :============================
  R 0(00000000)    R 1(00000001)   |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)   |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000006)   |   R 4(00000004)    R 5(00000006)
  R 6(0000000a)    R 7(fcf0fdef)   |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)   |   R 8(00000008)    R 9(00000009)
```

```
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
-----------------------              | -----------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 20 at CSIM 229 ns          | IFU : pc= 20 at CSIM 229 ns
-----------------------              | -----------------------
                                     |
******************************       | ******************************
ID: REGISTERS DUMP at CSIM 231 ns    | ID: REGISTERS DUMP at CSIM 231 ns
******************************       | ******************************
REG :============================    | REG :============================
  R 0(00000000)    R 1(00000001)     |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000006)     |   R 4(00000004)    R 5(00000006)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
IFU ALERT: **BRANCH**                | IFU ALERT: **BRANCH**
-----------------------              | -----------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 20 at CSIM 237 ns          | IFU : pc= 20 at CSIM 237 ns
------------------------------       | ------------------------------
ID: clear branch at CSIM 238 ns      | ID: clear branch at CSIM 238 ns
------------------------------       | ------------------------------
                                     |
******************************       | ******************************
```

```
ID: REGISTERS DUMP at CSIM 239 ns    | ID: REGISTERS DUMP at CSIM 239 ns
******************************       | ******************************
REG :============================    | REG :============================
  R 0(00000000)    R 1(00000001)     |   R 0(00000000)    R 1(00000001)
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000006)     |   R 4(00000004)    R 5(00000006)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
------------------------             | ------------------------
IFU : mem=0x1123000                  | IFU : mem=0x1123000
IFU : pc= 21 at CSIM 244 ns          | IFU : pc= 21 at CSIM 244 ns
-------------------------------      | -------------------------------
ID: R1= R2(=-1048350)+R3(=-1)        | ID: R1= R2(=-1048350)+R3(=-1)
  : at CSIM 246 ns                   |   : at CSIM 246 ns
-------------------------------      | -------------------------------
ALU : op= 3 A= -1048350 B= -1        | ALU : op= 3 A= -1048350 B= -1
ALU : R= -1048351-> R1 at CSIM 248ns | ALU : R= -1048351-> R1 at CSIM 248ns
-------------------------------      | -------------------------------
ID: R1=0xfff000e1(-1048351) fr ALU   | ID: R1=0xfff000e1(-1048351) fr ALU
 at CSIM 249 ns                      |  at CSIM 249 ns
------------------------             | ------------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 22 at CSIM 251 ns          | IFU : pc= 22 at CSIM 251 ns
-------------------------------      | -------------------------------
                                     |
******************************       | ******************************
ID: REGISTERS DUMP at CSIM 253 ns    | ID: REGISTERS DUMP at CSIM 253 ns
******************************       | ******************************
REG :============================    | REG :============================
  R 0(00000000)    R 1(fff000e1)     |   R 0(00000000)    R 1(fff000e1)
  R 2(fff000e2)    R 3(ffffffff)     |   R 2(fff000e2)    R 3(ffffffff)
  R 4(00000004)    R 5(00000006)     |   R 4(00000004)    R 5(00000006)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
```

```
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
------------------------             | ------------------------
IFU : mem=0x1234000                  | IFU : mem=0x1234000
IFU : pc= 23 at CSIM 258 ns          | IFU : pc= 23 at CSIM 258 ns
------------------------------       | ------------------------------
ID: R2= R3(=-1)+R4(=4)               | ID: R2= R3(=-1)+R4(=4)
  : at CSIM 260 ns                   |   : at CSIM 260 ns
------------------------------       | ------------------------------
ALU : op= 3 A= -1 B= 4               | ALU : op= 3 A= -1 B= 4
ALU : R= 3-> R2 at CSIM 262 ns       | ALU : R= 3-> R2 at CSIM 262 ns
------------------------------       | ------------------------------
ID: R2=0x3(3) fr ALU at CSIM 263 ns  | ID: R2=0x3(3) fr ALU at CSIM 263 ns
------------------------             | ------------------------
IFU : mem=0x0                        | IFU : mem=0x0
IFU : pc= 24 at CSIM 265 ns          | IFU : pc= 24 at CSIM 265 ns
------------------------------       | ------------------------------
                                     |
******************************       | ******************************
ID: REGISTERS DUMP at CSIM 267 ns    | ID: REGISTERS DUMP at CSIM 267 ns
******************************       | ******************************
REG :============================    | REG :============================
  R 0(00000000)    R 1(fff000e1)     |   R 0(00000000)    R 1(fff000e1)
  R 2(00000003)    R 3(ffffffff)     |   R 2(00000003)    R 3(ffffffff)
  R 4(00000004)    R 5(00000006)     |   R 4(00000004)    R 5(00000006)
  R 6(0000000a)    R 7(fcf0fdef)     |   R 6(0000000a)    R 7(fcf0fdef)
  R 8(00000008)    R 9(00000009)     |   R 8(00000008)    R 9(00000009)
  R10(00000010)    R11(0000ff31)     |   R10(00000010)    R11(0000ff31)
  R12(0000ff12)    R13(00000013)     |   R12(0000ff12)    R13(00000013)
  R14(00000014)    R15(00000015)     |   R14(00000014)    R15(00000015)
  R16(00000016)    R17(00fe0117)     |   R16(00000016)    R17(00fe0117)
  R18(00fe0118)    R19(00fe0119)     |   R18(00fe0118)    R19(00fe0119)
  R20(00fe0220)    R21(00fe0321)     |   R20(00fe0220)    R21(00fe0321)
  R22(00fe0322)    R23(00ff0423)     |   R22(00fe0322)    R23(00ff0423)
  R24(00ff0524)    R25(00ff0625)     |   R24(00ff0524)    R25(00ff0625)
  R26(00ff0726)    R27(00ff0727)     |   R26(00ff0726)    R27(00ff0727)
  R28(00f70728)    R29(00000029)     |   R28(00f70728)    R29(00000029)
  R30(00000030)    R31(00000031)     |   R30(00000030)    R31(00000031)
==================================   | ==================================
                                     |
```

```
------------------------           | ------------------------
IFU : mem=0xffffffff               | IFU : mem=0xffffffff
IFU : pc= 25 at CSIM 272 ns        | IFU : pc= 25 at CSIM 272 ns
-------------------------------    | -------------------------------
ID: - SHUTDOWN - at CSIM 274 ns    | ID: - SHUTDOWN - at CSIM 274 ns
ID: - PLEASE WAIT ...... -         | ID: - PLEASE WAIT ...... -
-------------------------------    | -------------------------------
```

D Inhalt der Quellcode-CD

Zur Sammlung und Bereitstellung der Daten, die im Rahmen dieser Arbeit erstellt wurden, dient eine CD-ROM. Sie liegt dieser Arbeit bei. In diesem Abschnitt soll der Inhalt der CD-ROM kurz dargestellt werden, um einen besseren Überblick zu gewährleisten. Die Verzeichnisstruktur gliedert sich wie folgt:

```
[+]/
  |-[+]Doc
  |   |-[+]Bilder
  |
  |-[+]Programme
  |   |-[+]Epp-speedtest
  |   |   |-[+]Hardware
  |   |   |-[+]Software
  |   |
  |   |-[+]Simifb-generator
  |   |   |-[+]Templates
  |   |
  |   |-[+]Timedrun
  |
  |-[+]Sources
  |   |-[+]Epp-speedtest
  |   |   |-[+]Hardware
  |   |   |-[+]Software
  |   |
  |   |-[+]Simifb-generator
  |   |-[+]Timedrun
  |
  |-[+]Testdesigns
      |-[+]Counter_8bit
      |   |-[+]Sifbgen-output
      |   |-[+]Simulation
      |   |-[+]SystemC
      |   |-[+]Vhdl
      |
      |-[+]Risc_cpu_original
      |   |-[+]Simulation
      |   |-[+]SystemC
      |
      |-[+]Risc_cpu_simifb
          |-[+]Intexecunit
          |-[+]Sifbgen-output
          |-[+]Simulation
          |-[+]SystemC
```

Im Rootverzeichnis befinden sich die Dateien *cd-directories.txt* und *cd-content.txt*. *Cd-directories.txt* enthält die Verzeichnisstruktur der CD-ROM. In *cd-content.txt* sind alle Dateien der CD-ROM mit ihren Verzeichnissen aufgelistet.

D.1 Verzeichnis DOC

Das Verzeichnis *DOC* beinhaltet diese Arbeit als PDF-Dokument. Im Unterverzeichnis *BILDER* befinden sich die zu dieser Arbeit erstellten Bilder. Bilder, die aus anderen Dokumenten oder von Websiten stammen, sind im Text mit Quellenangaben versehen und nicht im Verzeichnis enthalten. Die Bilder liegen im EPS-Format vor.

D.2 Verzeichnis PROGRAMME

Im Verzeichnis *PROGRAMME* befinden sich die Programme, die im Rahmen dieser Arbeit entstanden. An dieser Stelle sind nur die zur Ausführung notwendigen Dateien vorzufinden. Die Quelltexte liegen im Verzeichnis *SOURCES*.

■ Programme/Epp-speedtest

Epp-speedtest dient zur Messung der Geschwindigkeit der parallelen Schnittstelle. Die Messeinrichtung besteht aus einer Hardwarekomponente im Unterverzeichnis *Hardware* und einer Softwarekomponente im Unterverzeichnis *Software*. Die Hardwarekomponente wird durch eine Konfigurationsdatei *speedtest.bit* realisiert. Sie dient zur Konfiguration der Xilinx FPGA Spartan XC2S200E-PQ208 auf dem Digilab 2E Developmentboard und sollte auch nur für diesen Typ verwendet werden. Die Softwarekomponente bildet das Programm *speedtext_sw.exe*. Die Datei *readme.txt* enthält eine Anleitung zur Nutzung der beiden Teile.

■ Programme/Simifb-generator

Simifb_generator.exe ist die Datei zum Starten des Simulationsinterfaceblock-Generators. Die Benutzung des Programms ist in Kapitel 4, speziell in Abschnitt 4.3, ausführlich dargestellt. Die Datei *readme.txt* enthält eine kurze

Anleitung zum Starten des Programms. Das Unterverzeichnis *TEMPLATES* wird im folgenden Abschnitt beschrieben. Welche Dateien es enthalten muss, gibt die Datei *template-content.txt* an.

■ Programme/Simifb-generator/Templates

Im *TEMPLATES*-Verzeichnis befinden sich die Templates, die zum Erstellen eines Simulationsinterfaceblocks notwendig sind. Wenn das Verzeichnis umbenannt oder verschoben werden muss, dann sind Änderungen im Quelltext und ein erneutes Kompilieren des Programms notwendig.

Folgende Dateien müssen in diesem Verzeichnis enthalten sein, damit der Simulationsinterfaceblock-Generator seine Arbeit korrekt erfüllen kann:

Softwareteil	Hardwareteil
CUsw.cpp.t	auto_reset.vhd.t
CUsw.h.t	cu.vhd.t
Dlportio.h.t	data_splitting_off.vhd.t
DLPORTIO.lib.t	handlercontrol.vhd.t
EppIFsw.h.t	ifb.vhd.t
EppIFsw.cpp.t	phandler_out.vhd.t
epp.vhd.t	phandler_in.vhd.t
PHsw-Mode-ReadSimData.h.t	phin_mode_01.vhd.t
PHsw-Mode-WriteSimData.h.t	phout_mode_01.vhd.t
PHsw-Mode-ReadSimData.cpp.t	shandler.vhd.t
PHsw-Mode-WriteSimData.cpp.t	sh_mode_01.vhd.t
StdAfx.h.t	simdata_ctrl.vhd.t
	sim_ctrl.vhd.t
	sim_environment_HW.vhd.t
	sim_environment_hw.ucf.t

■ Programme/Timedrun

Das Programm *timedrun.exe* dient zur Messung der Ausführungszeit eines Programms. Eine Anleitung zur Nutzung enthält *readme.txt*. In den Skripten *copy_simulation.bat* und *run.bat* können Befehle eingetragen werden. Das erste Skript fällt dabei nicht unter die Zeitmessung. Die Messung umfasst allein die Ausführung von *run.bat*.

D.3 Verzeichnis SOURCES

Das Verzeichnis *SOURCES* enthält die Quellcodes, die Projektdateien und die Ausgaben von Kompilation und Implementierung zu den Programmen im Verzeichnis *PROGRAMME*.

■ Sources/Epp-speedtest

Im Unterverzeichnis *SPEEDTEST_HW* befindet sich das Projekt, die VHDL-Quellcodes und die Implementierungsdaten zum Hardwareteil der Messeinrichtung. Sie wurden mit der Entwicklungsumgebung Xilinx ISE 5 erstellt.

Das Unterverzeichnis *SPEEDTEST_SW* enthält die Projektdatei, die C/C++-Quellcodes und die Kompilationsdaten zum Softwareteil der Messeinrichtung. Ihre Erstellung erfolgte mit Hilfe der C/C++-Entwicklungsumgebung Microsoft Visual C++ 6.

■ Sources/Simifb-generator

Diese Verzeichnis beinhaltet das Projekt für den Simulationsinterfaceblock-Generator . Außerdem liegen in diesem Verzeichnis die Quellcodes und die Kompilationsdaten zum Simulationsinterfaceblock-Generator . Das Projekt wurde mit der C/C++-Entwicklungsumgebung Microsoft Visual C++ 6 bearbeitet.

■ Sources/Timedrun

In diesem Verzeichnis befindet sich das Projekt zum Erstellen des Programms zur Messung der Simulationszeit. Es umfasst die Quellcodes und die kompilierten Daten. Erstellt wurde es mit Hilfe der C/C++-Entwicklungsumgebung Microsoft Visual C++ 6.

D.4 Verzeichnis TESTDESIGNS

Das Verzeichnis *TESTDESIGNS* enthält Projekte zum Testen der Funktionalität des Simulationsinterfaceblocks und des Simulationsinterfaceblock-Generators. Das Unterverzeichnis *RISC_CPU_SIMIFB* beinhaltet dabei den Demonstrator, der in Kapitel 5 vorgestellt wurde.

■ Testdesigns/Counter_8bit

Das Unterverzeichnis *SIFBGEN-OUTPUT* enthält die Ausgaben des Simulationsinterfaceblock-Generators zum VHDL-Design aus dem Unterverzeichnis *VHDL*.

In *SIMULATION* befinden sich die ausführbare Datei *sw_ifb_counter8bit.exe* zur Durchführung der Simulation und das Logfile *sw_ifb_counter8bit.log*, das die Ausgaben der Simulation enthält. Außerdem stehen zur Durchführung einer Zeitmessung die Dateien *timedrun.exe*, *copy_simulation.bat* und *run.bat* zur Verfügung. Die Datei *time_taken_by_run.log* enthält die dafür benötigte Zeit.

Im Unterverzeichnis *SYSTEMC* liegt das SystemC-Design, aus dem die Simulation erstellt wurde. Es enthält die Projektdateien, die Quellcodes und die kompilierten Daten. Das Projekt wurde mit der C/C++-Entwicklungsumgebung Microsoft Visual C++ 6 bearbeitet. Zur erneuten Übersetzung müssen die Einstellungen für die Pfade auf die aktuellen Gegebenheiten angepasst werden.

Die VHDL-Quellcodes, die als Eingabe für den Simulationsinterfaceblock-Generator dienten, befinden sich im Unterverzeichnis *VHDL*. Die Dateien *counter_8bit.vhd* und *shift_l16.vhd* beschreiben einen 8 Bit-Zähler und ein 16 Bit-Schieberegister.

■ Testdesigns/Risc_cpu_original

Das Unterverzeichnis *SIMULATION* beinhaltet die Dateien zur Simulation der originalen RISC-CPU. Es existieren zwei Versionen der Simulation. Zum Einen die Version, die für die Zeitmessung aus Abschnitt 5.4.2 Verwendung fand. Bei ihr wurden die Ausgaben des Moduls *exec.cpp* entfernt. Ihre Dateinamen besitzen den Zusatz „_timed“. Dazu gehören die folgenden Dateien:

***risc_cpu_timed.exe*:** ausführbare Datei zum Starten der Simulation

***risc_cpu_timed.log*:** Logfile mit den Ausgaben der Simulation

***time_taken_by_run_timed.log*:** Ergebnisse der Zeitmessung

Die zweite Version verfügt über die vollständigen Ausgaben. Folgende Dateien zählen zu dieser Version:

***risc_cpu.exe*:** ausführbare Datei zum Starten der Simulation

***risc_cpu.log*:** Logfile mit den Ausgaben der Simulation

Des Weiteren befinden sich im Verzeichnis *SIMULATION* die Dateien zur Zeitmessung (*timedrun.exe, run.bat, copy_simulation.bat*) sowie die Eingabedaten für den Testbench (*bios, dcache, icache, register*).

Im Unterverzeichnis *SYSTEMC* befinden sich die Quelltexte des originalen SystemC-Designs der RISC-CPU, wie es in den Beispielen zu SystemC vorhanden ist. Der einzige Zusatz ist die Projektdatei für Microsoft Visual C++ 6 und die Kompilation der Quelldateien. Zur erneuten Übersetzung müssen die Einstellungen für die Pfade auf die aktuellen Gegebenheiten angepasst werden.

■ Testdesigns/Risc_cpu_simifb

INTEXECUNIT enthält das Xilinx ISE 5-Projekt, in dem die VHDL-Beschreibung der Integer-Executionunit erstellt wurde. Zum Projekt gehören Quelltexte, Testbenches und Implementierungsdaten.

Das Unterverzeichnis *SIFBGEN-OUTPUT* enthält die Ausgaben des Simulationsinterfaceblock-Generators. Als Eingaben dienten die VHDL-Quelldateien aus dem Verzeichnis *INTEXECUNIT*.

Im Unterverzeichnis *SIMULATION* befinden sich die Dateien zur Simulation des modifizierten RISC-CPU-Design mit Simulationsinterfaceblock. Es existieren zwei Versionen der Simulation. Zum Einen die Version, die für die Zeitmessung aus Abschnitt 5.4.2 Verwendung fand. Bei ihr wurden die Ausgaben des generierten Moduls *exec.cpp* entfernt. Ihre Dateinamen besitzen den Zusatz „_timed“. Dazu gehören die folgenden Dateien:

***risc_cpu_timed.exe*:** ausführbare Datei zum Starten der Simulation

***risc_cpu_timed.log*:** Logfile mit den Ausgaben der Simulation

***time_taken_by_run_timed.log*:** Ergebnisse der Zeitmessung

Die zweite Version verfügt über die vollständigen Ausgaben. Folgende Dateien zählen zu dieser Version:

***risc_cpu.exe*:** ausführbare Datei zum Starten der Simulation

***risc_cpu.log*:** Logfile mit den Ausgaben der Simulation

Des Weiteren befinden sich im Verzeichnis *SIMULATION* die Dateien zur Zeitmessung (*timedrun.exe, run.bat, copy_simulation.bat*) sowie die Eingabedaten für den Testbench (*bios, dcache, icache, register*).

Das Unterverzeichnis *SYSTEMC* enthält das Projekt, die Quelltexte und die kompilierten Daten für die Erstellung der Simulation der modifizierten RISC-CPU. Das Projekt wurde mit Microsoft Visual C++ 6 bearbeitet. Zur erneuten Übersetzung müssen die Einstellungen für die Pfade auf die aktuellen Gegebenheiten angepasst werden.

Literatur

[AB96] Sorin A. Huss Klaus Waldschmidt Andreas Bleck, Michael Goedecke. *Praktikum des modernen VLSI-Entwurfs*. B. G. Teubner, Stuttgart, 1996.

[Alt] Altera Corporation. Altera: Leaders in FPGAs, CPLDs, and Structured ASICs. http://www.altera.com/. Homepage.

[Apt] Aptix® Corporation. www.aptix.com. Homepage.

[Apt00] Aptix® Corporation. *System Explorer™ - Reconfigurable System Prototyping for SoC Emulation*, 2000. Broschüre zum System Explorer™.

[Atm] Atmel Corporation. Atmel Corporation. http://www.atmel.com/. Homepage.

[Axe97] Jan Axelson. *Parallel Port Complete*. Lakeview Research, Madison, USA, 1997.

[Dem93] Klaus Dembowski. *Computerschnittstellen und Bussysteme*. Markt-und-Technik-Verlag, Haar bei München, 1993.

[Dig] Digilent Inc. http://www.digilentinc.com. Homepage.

[Dig02] Digilent Inc. *Digilab 2E Reference Manual*, 14. April 2002. www.digilentinc.com.

[Els94] Jürgen Elsing. *Schnittstellen-Handbuch: verständliche Erläuterung und Benutzung von Centronics, V24, IEC-Bus*. IWT Verlag GmbH, Vaterstetten bei München, 1994. 4. Auflage.

[Fic03] Oliver Fick. Verschlüsselung von Parametern komplexer Schnittstellen für eingebettete Systeme auf Basis von XML. Studienarbeit, Universität Paderborn, 2003.

[Fla03] Marcel Flade. FPGA-basierte Fail-safe-Schnittstellen für eingebettete Systeme. Studienarbeit, Technische Universität Chemnitz, 2003.

[GL94] Manfred Selz Gunther Lehmann, Bernhard Wunder. *Schaltungsdesign mit VHDL - Synthese, Simulation und Dokumentation digitaler Schaltungen*. Franzis-Verlag GmbH, Poing, 1994.

[Har02] Dr. Wolfram Hardt. *Integration von Verzögerungszeit-Invarianz in den Entwurf eingebetteter Systeme*. Shaker Verlag, Aachen, 2002.

[Har03] Prof. Dr. Wolfram Hardt. Hardware-Software Codesign. Vorlesung, Technische Universität Chemnitz, 2003.

[Har04] Prof. Dr. Wolfram Hardt. Hardware-Software Codesign II. Vorlesung, Technische Universität Chemnitz, 2003/2004.

[HVI01] Wolfram Hardt, Markus Visarius, and Stefan Ihmor. Rapid prototyping of real-time interfaces. In *Field Programmable Logic (FPL) - Poster Session*, Belfast, Northern Ireland, UK, October 2001.

[IBJK+03] Stefan Ihmor, Nilson Bastos Jr., Rafael Cardoso Klein, Markus Visarius, and Wolfram Hardt. Rapid Prototyping of Realtime Communication - A Case Study: Interacting Robots. June 2003.

[IH04] Stefan Ihmor and Wolfram Hardt. Runtime Reconfigurable Interfaces - The RTR-IFB Approach. 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 3, April 2004. Santa Fe, New Mexico, USA.

[Ihm01] Stefan Ihmor. Entwurf von Echtzeitschnittstellen am Beispiel interagierender Roboter. Diplomarbeit, Universität Paderborn, 2001.

[Int] Intel Coporation. Intel Research - Silicon - Moore's Law. http://www.intel.com/research/silicon/mooreslaw.htm. Homepage.

[ipq] Projekt IPQ - Home Page. https://www.ip-qualifikation.de.

[IVH02a] Stefan Ihmor, Markus Visarius, and Wolfram Hardt. A Consistent Design Methodology for Configurable HW/SW-Interfaces in Embedded Systems. Montreal, Canada, Aug. 2002.

[IVH02b] Stefan Ihmor, Markus Visarius, and Wolfram Hardt. A Design Methodology for Application-specific Real-Time Interfaces. In

Proc. of the International Conference on Computer Design, Freiburg, Germany, Sept. 2002.

[IVH03] Stefan Ihmor, Markus Visarius, and Wolfram Hardt. Modeling of Configurable HW/SW-Interfaces. pages 51 – 60, Feb. 2003.

[Lat] Lattice Semiconductor Corporation. FPGA, CPLD and SERDES Programmable Logic Devices by Lattice Semiconductor. http://www.vantis.com/. Homepage.

[Mar02] Norbet Schuhmann Martin Speitel. Erfahrungen mit Intellectual Property. *www.elektroniknet.de*, 2002.

[Mäd] Andreas Mäder. Vhdl kompakt. http://tech-www.informatik.uni-hamburg.de/vhdl/doc/kurzanleitung/vhdl.pdf.

[Mena] Mentor Graphics Corp. www.mentor.com. Homepage.

[Menb] Mentor Graphics Corp. High-Performance System Integration Verification: VStation. http://www.mentor.com/vstation/. Homepage.

[Menc] Mentor Graphics Corp. VStationPRO High-Performance System Verification. http://www.mentor.com/-vstation/vstation_pro.html. Homepage.

[Mend] Mentor Graphics Corp. VStationTBX - High-Performance Verification Accelerator. http://www.mentor.com/vstation/-vstation_tbx.html. Homepage.

[Men03] Mentor Graphics Corp. *VStationPRO High-Performance System Verification*, 2003. Broschüre zur VStationPro.

[Men04] Mentor Graphics Corp. *VStationTBX - Datasheet*, 2004. Broschüre zur VStationTBX.

[Mic04] Microsoft Corporation. Geistiges Eigentum - Definition. http://www.microsoft.com/germany/digital-mentality/definition.mspx, 2004. Homepage.

[Mül02] Prof. Dr. Dietmar Müller. Entwurfssysteme. Vorlesung, Technische Universität Chemnitz, 2002.

[Mül03] Prof. Dr. Dietmar Müller. ASIC Entwurf. Vorlesung, Technische Universität Chemnitz, 2003.

[Mod] Model Technology. www.model.com. Homepage.

[Mod03] Model Technology. *ModelSim® LE User's Manual*, Dezember 2003. Version 5.8a.

[Mon00a] Prof. Dr.-Ing. Dieter Monjau. Rechnerorganisation. Vorlesung, Technische Universität Chemnitz, 2000.

[Mon00b] Prof. Dr.-Ing. Dieter Monjau. VHDL - Einführung. Unterlagen zur Vorlesung Rechnerorganisation, Technische Universität Chemnitz, 2000.

[Nau99] Dr. Bernt Naumann. Digitaltechnik. Vorlesung, Technische Universität Chemnitz, 1999.

[Nau01] Dr. Bernt Naumann. Werkzeuge für den Systementwurf. Vorlesung, Technische Universität Chemnitz, 2001.

[Opea] Open SystemC Initiative. SystemC™ Homepage. http://www.systemc.org. Homepage.

[Opeb] Open SystemC Initiative. *SystemC™ User's Guide*. Version 2.0.

[Ope02] Open SystemC Initiative. *Functional Specification for SystemC™ 2.0*, April 2002. Version 2.0-Q.

[Pea04] Craig Peacock. Beyond Logic. http://www.beyondlogic.org/, Juli 2004. Webseite.

[Ram89] Fanz-J. Rammig. *Systematischer Entwurf digitaler Systeme*. B. G. Teubner, Stuttgart, 1989.

[Scia] Scientific Software Tools Inc. http://www.driverlinx.com/-DownLoad/DlPortIO.htm. Downloadseite von port95nt.exe.

[Scib] Scientific Software Tools Inc. SST - Data Acquisition Hardware/Software and Custom Applications. http://www.driverlinx.com. Homepage.

[Syn] Synopsys Corporate Marketing. Synopsys® Homepage. http://www.synopsys.com. Homepage.

[Syn03] Synopsys Inc. *CoCentric® SystemC™ Compiler - RTL User and Modeling Guide*, Juni 2003. Version U-2003.06, www.synopsys.com.

[Tei97] Dr.-Ing. Jürgen Teich. *Digitale Hardware/Software-Systeme - Synthese und Optimierung.* Springer-Verlag Berlin, Heidelberg, 1997.

[Thaa] Tharas Systems Inc. www.tharas.com. Homepage.

[Thab] Tharas Systems Inc. *Hammer® 100 Hardware Accelerator for Verilog, VHDL and Mixed language simulation.* Broschüre zum Hammer® 100 System.

[Thi94] Michael Thieser. *PC-Schnittstellen.* Franzis-Verlag GmbH, München, 1994.

[VA] VA Software Corp. SystemC™Open Source Lizenz. http://www.systemc.org/web/sitedocs/-open_source_licensing.html.

[Ver96] Verein Deutscher Ingenieure. *VDI 3633 - Simulation von Logistik-, Materialfluß- und Produktionssystemen - Begriffsdefinition*, November 1996.

[VH04] Markus Visarius and Wolfram Hardt. The IPQ Format – An Approach to Support IP based Design. In *Proc. of the GI/ITG/GMM-Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 106 – 115, Kaiserslautern, Germany, Feb. 2004.

[VLH+03] M. Visarius, J. Lessmann, W. Hardt, F. Kelso, and W. Thronicke. An XML Format based Integration Infrastructure for IP based Design. In *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI 2003)*, pages 119 – 124, São Paulo, Brazil, 08. - 10. September 2003. IEEE Computer Society.

[VLKH04] Markus Visarius, Johannes Lessmann, Frank Kelso, and Wolfram Hardt. Generic integration infrastructure for ip based design processes and tools with a unified xml format. *Integration, the VLSI journal*, 37(4):289 – 321, September 2004.

[Vra98] Hendrikus P.E. Vranken. *Design for test and debug in hardware/software systems.* Technische Universität Eindhoven, Eindhoven, 1998.

[Wan98] Markus Wannemacher. *Das FPGA-Kochbuch.* International Thomson Publishing GmbH, Bonn, 1998.

[War] Warp Nine Engineering. Warp Nine Engineering- The IEEE 1284 Experts. http://www.fapo.com/ieee1284.htm. Webseite.

[Wika] Wikimedia Foundation Inc. Wikipedia - Die freie Enzyklopädie. http://de.wikipedia.org/wiki/Adapter. Definition: Adapter.

[Wikb] Wikimedia Foundation Inc. Wikipedia - Die freie Enzyklopädie. http://de.wikipedia.org/wiki/Schnittstelle. Definition: Schnittstelle.

[Wor] World Intellectual Property Organization. WIPO - World Intellectual Property Organization. http://www.wipo.int/. Homepage.

[Xila] Xilinx Inc. Development System Reference Guide - ISE5. http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/-dev/dev.pdf.

[Xilb] Xilinx Inc. Xilinx: Design Tools Center. http://www.xilinx.com/-products/design_resources/design_tool/index.htm. Homepage.

[Xilc] Xilinx Inc. Xilinx: Programmable Logic Devices, FPGA & CPLD. http://www.xilinx.com. Homepage.

[Xild] Xilinx Inc. Xilinx Synthesis Technology (XST) User Guide. http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/xst/xst.pdf.

[Xil03] Xilinx Inc. *Spartan-IIE 1.8V FPGA Family: Complete Data Sheet*, 9. Juli 2003. Version 2.1, http://direct.xilinx.com/bvdocs/publications/ds077.pdf.

Abkürzungsverzeichnis

ASIC Application-Specific Integrated Circuit

CD Compact Disc

CPU Central Prozessing Unit

CU Controlunit

DSP Digitaler Signalprozessor

EDA Electronic Design Automation

EPP Enhanced Parallel Port

EPS Encapsulated PostScript

FPGA Field Programmable Gate Array

FPIC Field Programmable Interconnect Component

FSM Finite State Machine

HDL Hardware Description Language

HW Hardware

IEEE Institute of Electrical and Electronics Engineers

IF Interface

IFB Interfaceblock

IFS Interface Synthese

I/O Input-Output

IP Intellectual Property

ISS Instruction-set Simulator

MMX Multimedia Extension

OSCI Open SystemC Initiative

PDF Portable Document Format

PC Personal Computer

PH Protokollhandler

RAM Random Access Memory

RISC Reduced Instruction Set Computing

ROM Read Only Memory

RT Register-Transfer

RTL Register-Transfer-Level

RTOS Real-Time Operation System

SH Sequenzhandler

SimIFB Simulationsinterfaceblock

SoC System-on-Chip

SRAM Static Random Access Memory

SW Software

VDI Verein Deutscher Ingenieure

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

XML Extensible Markup Language