Dokumentation

Elektrotechnik

Projektgruppe
Multiprocessor-in-the-Loop

Dipl.-Ing. (FH) Michael Mayer (Matr.-Nr.: 6342897)
Hanyi Li (Matr.-Nr.: 6288777)
Vadim Kataev (Matr.-Nr.: 6198126)

Betreuer:
Dipl.-Wirt.-Ing. Christian Liss

Paderborn
Oktober 2007

Inhaltsverzeichnis

1 AUFGABENSTELLUNG....3
2 GRUNDLAGEN ZU „HARDWARE-IN-THE-LOOP“ (HIL)....5
2.1 Definition von HIL....5
2.2 Vergleich verschiedener Testprozessbeschreibung....5
2.2.1 Übersicht....5
2.2.2 V-Modell....6
2.2.3 Rapid-Prototyping....6
2.2.4 Software-in-the-Loop (SIL)....7
2.2.5 Hardware-in-the-Loop (HIL)....7
2.3 Stand der Technik....8
2.3.1 Boeing Corporation: EASY5....9
2.3.2 dSpace....10
2.3.3 ETAS: ASCET....10
2.3.4 Wind River Systems: MATRIXX....11
2.3.5 Sonstige....11
3 UMSETZUNG DES PROJEKTS....14
3.1 Übersicht....14
3.2 ModelSim....15
3.3 HiLDE....18
3.4 Design Compiler for FPGA....20
3.5 Core-Generator....21
3.6 Xilinx ISE....22
3.7 RAPTOR2000....24
3.8 Foreign Language Interface (FLI)....24
4 ERGEBNISSE....26
4.1 Testprogramm....26
4.2 Simulation....26
4.3 Reale Prozessoren....26
4.4 Messung....27
4.5 Vergleich HIL zu ursprünglichem System....27
5 ZUSAMMENFASSUNG UND AUSBLICK....28

ABKÜRZUNGSVERZEICHNIS.....29
LITERATURVERZEICHNIS.....30
ABBILDUNGSVERZEICHNIS.....32

1 Aufgabenstellung

Der Arbeitsplan umfasst folgende Aufgabenbereiche:

- Information über die Ergebnisse der Projektgruppe des Wintersemesters (Schnittstellen-Modul) als auch über das aufzuteilende 4-Prozessor-System.
- Entscheidung über geeignete Simulations-Emulations-Schnittstelle im 4-Prozessor-System. Integration in das 4-Prozessor-System und Test des Schnittstellen-Moduls (VHDL).
- Austausch des simulierten Teils durch den entsprechenden Teil des SystemC-Modells und Implementierung eines einfachen Demonstrations-Programm in C
- Anfertigung und Ausarbeitung der Dokumentation

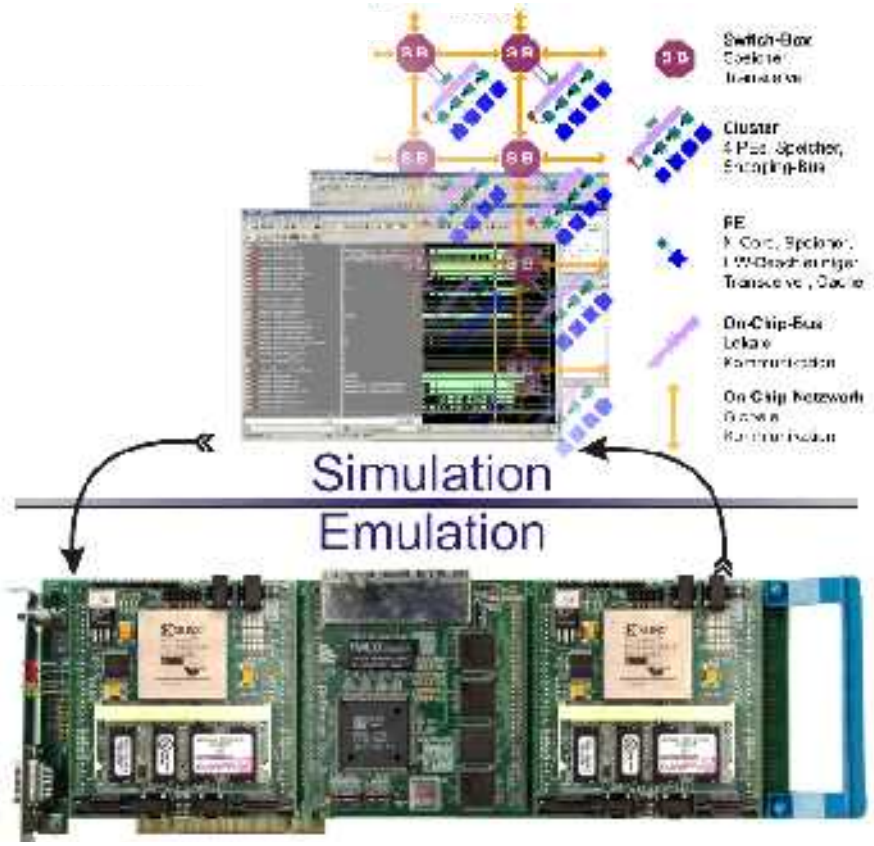

Abbildung 1.1: Aufgabenstellung

Das Ziel der Projektgruppe ist die Aufteilung eines Multi-Prozessors in einen simulierten und einen emulierten Teil.

2 Grundlagen zu „Hardware-in-the-Loop“ (HIL)

2.1 Definition von HIL

Eine Definition von Peter Wältermann aus seiner Dissertation „Der serielle Hybridantrieb – Vom rechnergestützten Entwurf bis zur Hardware-in-the-Loop-Realisierung“:

„Hardware-in-the-Loop-Simulation ist ein Verfahren im Produktentwicklungszyklus, bei dem eine oder mehrere reale Komponenten in Wechselwirkung[1] mit in Echtzeit simulierten Komponenten arbeiteten“. [7]

2.2 Vergleich verschiedener Testprozessbeschreibung

2.2.1 Übersicht

Abhängig vom jeweiligen Entwicklungsstand, kommt ein anderer Testprozess zur Anwendung, siehe dazu auch Abbildung 2.2: Simulation und Systemeinsatz. Der Software-in-the-Loop Test kommt zum Einsatz, wenn einzelne Softwaremodule ohne bestehende Hardware getestet werden sollen. Rapid-Prototyping bietet sich immer dann an, wenn schnell ein Prototyp oder Teilprototyp in der realen Umwelt zu entwickeln ist. Hardware-in-the-Loop wird mit realer Hardware und teilweise oder vollständiger simulierten Umwelt durchgeführt. Bei Hardware-in-the-Loop besteht die Möglichkeit auch nur Teilsysteme einer realen Hardware zu testet und den noch nicht entwickelten Teil zu simulieren. Die gewonnen Daten aus den verschiedenen Tests werden in die Messdaten der Umwelt und die Messdaten des Testmusters integriert.

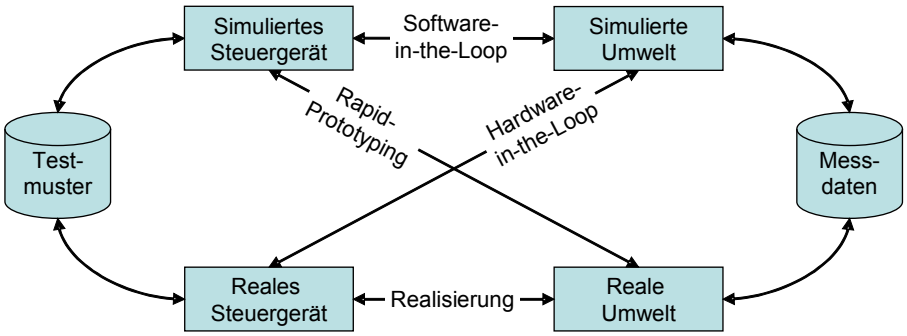

Abbildung 2.2: Simulation und Systemeinsatz [8]

Welches Testsystem zum Einsatz kommt ist abhängig vom jeweiligen Entwicklungsstand und der vorgegebenen Spezifikation. Bei neuen Projekten sind die Messdaten und Testmuster nur Schätzwerte, welche sich zum Beispiel auf die Werte von früheren ähnlichen Testaufbauten berufen, und werden erst im Laufe der Entwicklung durch Erweiterung mit realen Daten gefüllt. Dadurch werden die Tests nach jedem Durchlauf genauer und können spezifischer durchgeführt werden.

[1] Das bedeutet im allgemeinen im geschlossenen Regelkreis bzw. in der geschlossenen Simulationsschleife (Closed Loop)

2.2.2 V-Modell

Das V-Model wurde eingeführt um die komplexe Bereiche von der Entwicklung bis zur Produktion anschaulich beschreiben zu können, siehe dazu auch Abbildung 2.3: V-Modell. Meistens wird dieses Modell, während einer Entwicklung mehrmals durchlaufen, und wird deshalb auch als W-Modell bezeichnet.

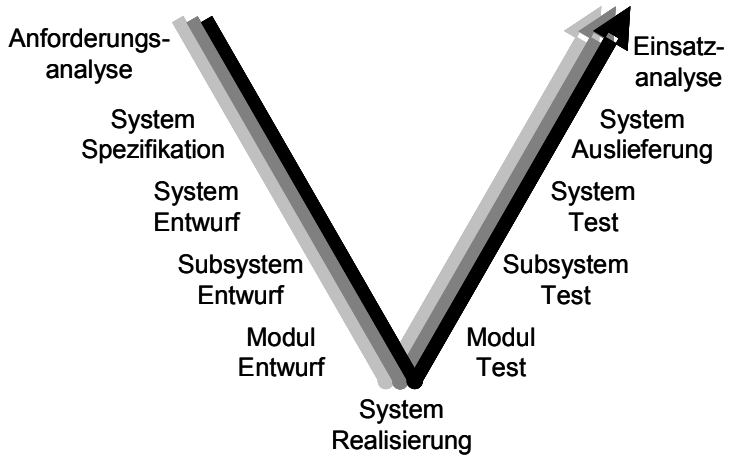

Abbildung 2.3: V-Modell [8]

2.2.3 Rapid-Prototyping

Beim Rapid-Prototyping werden verschiedene Verfahren beschrieben um in einer frühen Entwicklungsphase Prototypen zu generieren, welche in einer realen oder simulierten Umwelt testbar sind. Aus dem V-Model resultieren verschieden Rapid-Prototyping-Verfahren: Das konzeptorientierte Rapid-Prototyping, welches aus den Phasen Anforderungsanalyse, Systemspezifikation, System-Auslieferung und Einsatzanalyse besteht, das architekturorientierte Rapid-Prototyping, bestehend aus System-Entwurf, Subsystem-Entwurf, Subsystem-Test und System-Test und zuletzt des realisierungsorientierten Rapid-Prototyping, welches aus Modul Entwurf, System-Realisierung und Modul-Test besteht.

Konzeptorientiertes Rapid-Prototyping:

Beim konzeptorientierten Rapid-Prototyping wird eine Schaltung oder ein elektronisches System entwickelt, welches einem völlig realisierungsunabhängigen System gleichzusetzen ist. Das Verfahren dient dazu, ein allgemeines Konzept oder eine Funktionsweise zu bewerten, bevor die Weiterentwicklung durchgeführt wird. Die Funktionsweisen werden meist graphisch ermittelt und der Softwarecode für dieses System ist meist ein Modell-Code der in relativ kurzer Zeit auf das Test-System geladen werden kann. Für die weitere Entwicklung wird dieser automatisch generierte Code jedoch nicht weiter verwendet.

Beim konzeptorientierten Rapid-Prototyping werden die allgemeinen Funktionen bestimmt und bewertet. [8]

Architekturorientiertes Rapid-Prototyping:

Nach dem konzeptorientierten Rapid-Prototyping erfolgt das architekturorientierte Rapid-Prototyping. Bei diesem wird zum Beispiel in der Steuergeräte-Entwicklung als erstes der zu verwendete Microcontroller ausgewählt.

Im Vergleich zum konzeptorientierten Rapid-Prototyping sind beim architekturorientierten Rapid-Prototyping die ersten realen Funktionseinheiten entwickelt. Zu den ersten Funktionseinheiten gehören die Spannungsversorgung, und die Microcontroller-Eingangs- und Ausgangssignalverarbeitung. Es werden jedoch am Anfang nicht alle Funktionseinheiten im Bereich Eingangs- und Ausgangssignalverarbeitung realisiert. Die noch nicht realisierten oder nur zum Teil realisierten Teile werden simuliert. In dieser Entwicklungsphase erfolgt die Entscheidung über die Mikroprozessorfamilie. Damit können erste Annahmen über die Schnelligkeit gemacht werden. Zusätzlich lassen sich mögliche elektromagnetische Probleme des späteren Systems in dieser frühen Entwicklungsphase erkennen.

Durch den Prototyp, welches beim architekturorientierten Rapid-Prototyping entwickelt worden ist, lassen sich bereits grundsätzliche Aussagen über das System machen, ohne dass alle Untersysteme realisiert sind oder getestet sind. Der Entwurf der Systeme und Subsysteme beinhaltet das Hardwaredesign und Softwaredesign. [8]

Realisierungsorientiertes Rapid-Prototyping:

Das letzte Rapid-Prototyping Verfahren ist das realisierungsorientiere Rapid-Prototyping. Bei diesem steht ein Großteil des Systems bereits und es müssen einzelne Module nur noch ergänzt werden. In der Industrie werden diese Prototypen als Vorproduktserie bezeichnet. Diese Prototypen werden genutzt um die endgültige Leistungsfähigkeit des Systems beschreiben zu können. Die Software-Tests werden in dieser Phase als Black-Box-Tests durchgeführt, welche nicht mehr auf die einzelnen expliziten Funktionen eingeht, sondern auf das Verhalten des gesamten Systems.

Beim realisierungsorientierten Rapid-Prototyping werden die Prototypen bereits auf spezielle Anwendungsfälle getestet und der Code generiert. [8]

2.2.4 Software-in-the-Loop (SIL)

Software-in-the-Loop wird bei der Entwicklung immer dann eingesetzt wenn noch keine oder nicht funktionsfähige Hardware zur Verfügung steht. Bei diesem Testablauf können einzelne Module oder bereits die vollständige Software getestet werden. Weiterhin wird Software-in-the-Loop dann eingesetzt, wenn schnelle Tests notwendig sind, bei der die simulierte Umgebung um ein vielfaches schneller ist als die Reale. Durch den SIL-Test besteht zusätzlich die Möglichkeit Größen zu messen, die in einer späteren Entwicklungsphase nicht mehr oder nur noch schwer zugänglich sind.

Mit Software-in-the-Loop werden die verschiedenen Algorithmen bzw. der Code des späteren Systems getestet. [8]

2.2.5 Hardware-in-the-Loop (HIL)

Einer der ersten HIL-Simulatoren wurde in Zusammenarbeit zwischen der Volkswagen-Forschung und dem Institut IEE-Antriebstechnik der Universität Kassel entwickelt. Dieser testete das Motor Steuergerät für Zündung und Einspritzung, welches damals noch nicht in einem Steuergerät integriert waren.

Aufgrund der Komplexität und dem damit verbundenen Applikations- und Integrationsaufwand, werden Hardware-in-the-Loop Tests immer mehr im Bereich der Elektronikentwicklung bei Fahrzeugen eingesetzt. Während jeder Phase des Testens werden die Ergebnisse im V-Modell in das Konzept mit Integriert.

Ein weiterer Vorteil des HIL-Testens ist, dass die Testfälle reproduzierbar, im Bereich der Gefahrlosigkeit sowie der schnellen Umsetzung ablaufen. Die Gefahrlosigkeit ist ebenso für spätere rein reale Tests gegeben, weil bei den Aufbauten die worst-case Fälle bereits im Vorfeld getestet werden können. Somit ist die wesentliche Stärke der HIL Technologie im Vergleich zu konventionellen Prüfständen oder reinen Simulation darin begründet, dass es aus einer Kombination aus realen Bauteilen mit simulierten Umgebungsbedingungen zusammengesetzt ist. Es wird eine bestmögliche Abbildung der Bauteileumwelt mit geringem Aufwand gewährleistet. Die Prüftiefe kann mit dieser Art zu Testen deutlich erhöht werden ohne den Zeitaufwand um ein Vielfaches zu erhöhen. Der Versuchsablauf bleibt jedoch transparent und die Einstellungen von Betriebszuständen und die Vermeidung bzw. Erzeugung von Störeinflüssen bleibt erhalten. Das Ergebnis hängt wie bei der reinen Simulation direkt vom Detaillierungsgrad der Modelle ab. [7, 9]

Bei Steuergeräten zum Beispiel werden bei den ersten Tests die Sensoren und Aktoren nur simuliert und erst zu einem späteren Zeitpunkt durch reale ersetzt. Durch die exakte Beschreibung der Testumgebung und des Testablaufes, ist eine Reproduktion des Tests in jedem Bereich möglich und eine schnelle Eingrenzung von Fehlfaktoren möglich.

2.3 Stand der Technik

Überblick:

In dem Kapitel werden eine Auswahl von Produkten von verschiedenen Firmen im Bereich Hardware-in-the-Loop, sowie der des Rapid-Prototyping vorgestellt. Zu diesen Firmen gehören Boeing Corporation, dSpace, ETAS und Wind River Systems. Im Weiteren werden noch Produkte von General Electric Corp., UC Berkeley, der Universität-GH Kassel + Volkswagen Forschung, Mechatronik Laboratorium Paderborn beschrieben.

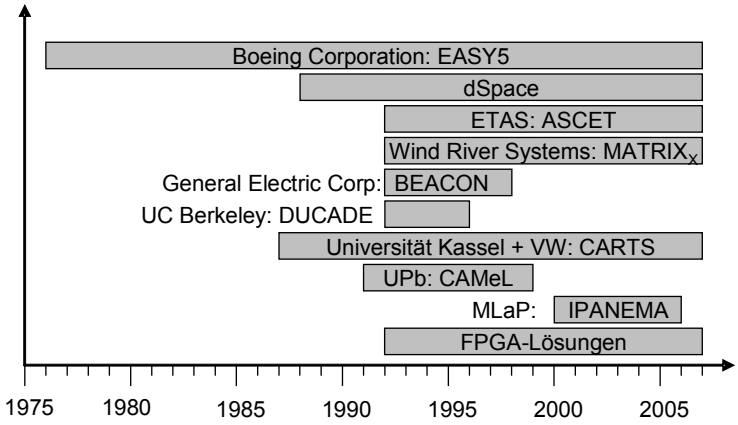

Abbildung 2.4: Zeitliche Übersicht

2.3.1 Boeing Corporation: EASY5

Engineering Analysis System Version 5 (EASY5) ist ein Softwarepaket zur Modellierung und Simulation dynamischer Systeme. Ursprünglich wurde dieses von der Boeing Corporation für den Einsatzbereich von zivilen Passagierflugzeugen entwickelt. Heute wird EASY5 eingesetzt bei der Planung bei Future Combat Systems Lead Systems Integrator (FCS LSI) in Zusammenarbeit mit der U.S. Army, in der Automobil Industrie und weiterhin Flugzeug Industrie. [12]

Entwickelt wurde EASY5 bereits 1976 als Werkzeugkopplung zwischen linearen und nichtlinearen Modellen. 1980 wurde das Programm von der Boeing Computer Service kommerziell vertrieben. Mit der neusten Version EASY5 (2005) können Modelle zu Matlab/Simulink, NASTRAN, ADAMS und Matrix Algebra Tool exportiert werden. Zusätzlich kann EASY5 neben der Windows Plattform auch auf der von Plattform Linux betrieben werden. [8]

Mit EAYS5 lassen sich eine große Anzahl von Systemen modellieren, dazu gehören unter anderem Mechanische, Elektrische, Luftfahrt/Astronautische, Hydraulische, Pneumatische, Thermische sowie kombinierte Systeme. Eines der Stärken von EASY5 ist, dass es ein großes Spektrum von Entwurfsmethoden und Analysemethoden liefert. Da es ursprünglich nicht für den Einsatz von Rapid-Prototyping zur Entwicklung elektronischer Steuergeräte bzw. elektronischer Systeme konzipiert war, sondern zur Beschleunigung der Offline Simulation erfüllt es keine harten Echtzeitbedingungen. [8]

Die zu simulierten Systeme werden modelliert und dargestellt mit Funktionsblöcken (Addierer, Dividierer, Signalgeneratoren, usw.) und mit vordefinierten Komponenten, welche verschiedene physikalische Elmente beschreiben, wie Pumpen, Generatoren, Motoren, usw. [13]

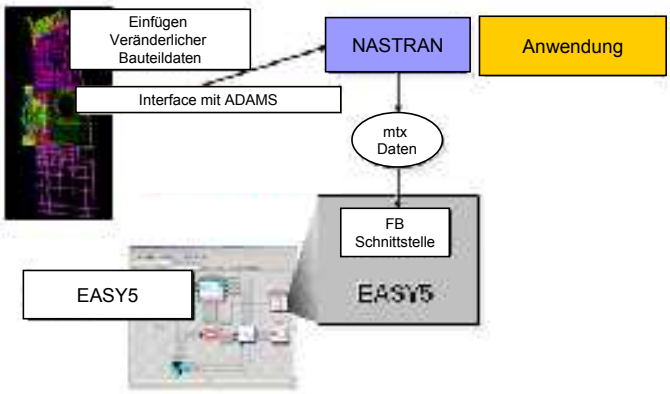

Abbildung 2.5: Reale Daten, NASTRAN und das EASY5 Programm [13]

EAYS5 (2005) ist ein Analyse Tool für nichtlineare Simulation, statische Analyse, designtes Regelungssystem und Daten Analyse. Das Einsatzspektrum von EASY5 vor allem in der Automobil Industrie, neben der Flugzeug Industrie, wird in den nächsten Jahren wahrscheinlich noch zunehmen. [8]

2.3.2 dSpace

dSpace ist einer der größten Vertreiber von Hardware-in-the-Loop und Rapid-Prototyping Systemen. Die dSpace GmbH wurde 1988 gegründet. Ihre Software Total Software Development Environment baut auf Matlab/Simulink und Realtime Workshop auf. Zusammen mit dem Zusatzprogramm ControlDesk können Steuerungen in Echtzeit erstellt werden. Mit dem Programm TargetLink für Microcontroller mit Integrierter Arithmetik lassen sich schnell effizient Codegenerierung durchführen. [7]

Die Software AutomationDesk ist ein generisches Testautomatisierungstool. Der Vorteil damit ist ein integriertes Testmanagement, ein leistungsfähiger Mechanismus zur Erstellung von eigenen Testbibliotheken, eine Ergebnisverwaltung sowie eine integrierte Reportgenerierung.

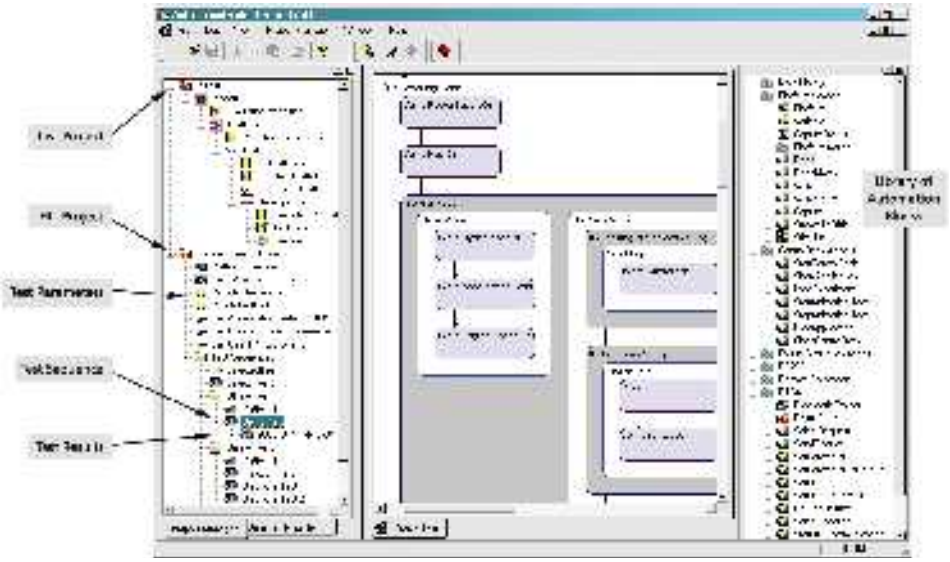

Abbildung 2.6: Integrierte Testumgebung Automation Desk

Neben der Automobil Industrie sind Robotik, Antriebstechnik, Bahntechnik, Luft- und Raumfahrt weitere Anwendungsgebiete. [14]

2.3.3 ETAS: ASCET

ETAS ist neben dSpace einer der führenden Hersteller von HIL-Systemen, für den Automobilen Bereich. Zur der Produktfamilie von Advanced Simulation and Control Engineering Tool (ASCET) gehören eine Vielzahl von Programmen, welche hauptsächlich bei der Funktions- und Softwareentwicklung von Embedded Systeme in der Kraftfahrzeug Elektronik eingesetzt werden. Es wird im Folgenden eine Auswahl beschrieben.[15]

Zu dieser Softwarefamilie gehört ASCET-SD (ASCET-Software-Development), welches ursprünglich durch eine Dissertation an der Universität Paderborn entstanden ist. Das Programm unterstützt die Schnittstelle zu MATLAB/Simulink, $MATRIX_X$. Damit besteht die Möglichkeit eine reine Software-in-the-Loop Lösung ohne Hardware zu realisieren. Zum Beispiel kann ein Motormodell, welches im Matlab/Simulink definiert worden ist, mit einer in ASCET generierten Motorsteuerung, in einem online Datenaustausch getestet werden.

Zur ASCET-Produktfamilie gehören zum Beispiel zusätzlich noch:

ASCET-MD (Modeling und Design), ASCET-RP (Rapid-Prototyping), usw. Mit ASCET-MD lassen sich Mithilfe von Blockdiagrammen und Zustandsautomaten auf physikalischer Ebene Modulkomponenten erstellen. Mit ASCET-RP lassen sich Rapid-Prototyping Anwendungen unter Echtzeitbedingungen durchführen. ASCET-RP unterstützt weitere Programme wie INCA. [14]

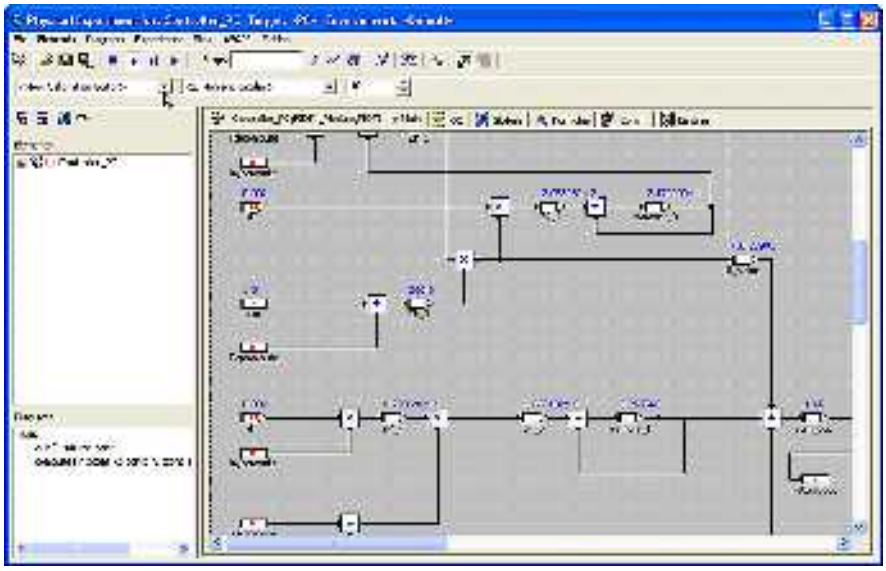

Abbildung 2.7: ETAS:ASCET

Die ASCET-Werkzeuge unterstützen Modellierung, Simulation, Rapid-Prototyping und Generierung von Embedded Software im Automobil und werden nicht nur im Automobilsektor sondern auch zum Beispiel in der Flugzeug Industrie eingesetzt. [7]

2.3.4 Wind River Systems: MATRIXX

Die Firma Wind River Systems entwickelte, neben $MATRIX_X$, das Werkzeug $MATRIX_X$ 00 und Tornado™, welches für den Einsatz von Hardware-in-the-Loop Simulation genutzt wird. Das Programm arbeitet unter Echtzeitbedingung und ermöglicht eine automatische Codegenerierung. Der Programmaufbau ist ähnlich wie bei MATLAB, und besteht aus verschiedenen Einzelmodulen und Toolboxen. Zu den wichtigsten Einzelmodulen gehört Xmath, welches zur mathematischen Analyse und Visualisierung, sowie SystemBuilt, für kontinuierliche und diskret formulierte Modelle. $MATRIX_X$ 00 ist zurzeit noch Hauptsächlich in der Automobil Industrie verbreitet, aber nicht ausschließlich dafür konzipiert. [15,16,17]

2.3.5 Sonstige

General Electric Corp: BEACON

BEACON wurde 1992 von der General Electric Corporation Entwickelt. BEACON ist eine Software zur Modellierung und Simulation von Steuerungs- und Regelungssystemen. Es wird ein automatisch ein ausführbarer Echtzeitcode erstellt. Die Software bietet aber keine Unterstützung zu anderen Programmen wie zum Beispiel $MATRIX_X$ oder STATEMATE an. [8]

Die Programmstruktur von BEACON ist in drei Hierarchieebenen gegliedert. Die erste Ebene, der Graphische Blockdiagramm Editor wird für den Entwurf des Systems benötigt. Die zweite Ebene ist die Netzliste, welches die Daten in ein Zwischenformat speichert und alle vollständigen Diagramme beinhaltet. Die dritte und damit letzte Ebene bildet der Automatische Codegenerator. Diese transformiert aus dem graphischen Modell aus Blockdiagrammen einen simulations- oder echtzeitfähigen Code. [8]

Universität von Kalifornien: DUCADE

Domain Unified CAD Environment (DUCADE) entwickelt von der Universität von Kalifornien. DUCADE ist ein Verbindungsprogramm des mechanischen CAD-Werkzeuges ARIES und des elektronischen CAD-Werkzeugen RACAL/FINNSE. Mit dem Programm wird das System erst simuliert und anschließend direkt die Leiterplatte gefertigt. Im Jahr 1996 wurde die Entwicklung von DUCADE eingestellt. [8]

Universität-GH Kassel + Volkswagen Forschung: CARTS

CARTS wurde in Zusammenarbeit von der Universität Kassel und der Volkswagen Forschung entwickelt. CARTS ist ein Closed-Loop HIL-Simulator für den Einsatz von Motorsteuergeräten. Das verwendete Betriebssystem war bei den ersten Versionen OS9. Neue Versionen unterstützen auch Windows, aber mit reduziertem Spektrum. Die kommerzielle Vermarktung wurde von HP betrieben. [8]

Universität in Paderborn: CAMeL

Computer-Aided Mechatronic Laboratory (CAMeL) ist ein Entwicklungswerkzeug zur Modellierung und Simulation komplexer mechatronischer Systeme. CAMeL wurde im Jahr1991 an der Universität in Paderborn entwickelt. Das Programm ist in drei Ebenen gegliedert:

Objective-Dynamic-System Structure (ODSS): Modellierungsebene; Basiselemente, hierarchische Elemente und Koppelelemente

Objective-Dynamic System Language (ODSL): Systemrepräsentation in ODDS in mathematische Beschreibung überführen.

Dynamik System Code (DSC): verarbeitungsorientiertes System; Beschreibung in abstrakter, maschinenunabhängigen Weise

Mit CAMeL lassen sich nur Systeme realisieren, welche als Differentialgleichung aufgefasst werden können und nicht ereignisbasierte oder dem Object-Oriented Analysis and Design (OOAD) entstammen. [8,6]

Mechatronik Laboratorium Paderborn (MLaP): IPANEMA

Die Integration Platform for Networked Mechatronic Application (IPANEMA) Simulationsplattform wurde vom Fachbereich MLaP an der Universität Paderborn entwickelt. Das Programm lässt sich in vier verschiedenen Ebenen aufteilen:

Calculator: In diesem in hart Echtzeitbedingung laufenden Klasse, findet die Modellberechung des Systems dar.

Adapter: Bildet den Schnittpunkt zwischen den physikalischen Eingängen und Ausgängen, sowie der Simulation.

Assistent: Jedem Calculator-Prozess und Adapter ist ein Assistent zugeordnet, welcher die administrativen Aufgaben übernimmt.

Moderator: Stellt die Schnittstelle zwischen dem Interface des Benutzers und der der Assistenten dar. [7]

FPGA-basiertes Rapid-Prototyping:

Beim FPGA basierten Rapid-Prototyping werden Hardware- oder Logik-Emulatoren eingesetzt. Firmen, welche diese Emulatoren entwickelt haben, sind zum Beispiel Quickturn mit dem Logic Animator oder die Firma Zycad. Zycad vertreibt das Paradigm.

Ursprünglich wurden diese Emulatoren entwickelt zur prototypischen Realisierung von digitalen IC-Bausteinen. Eingesetzt werden diese Emulatoren bei leistungsfähigen Applikationen bei denen eine hohe Taktrate bis in den MHz Bereich gefordert wird. Die mögliche geringe Auslastung der einzelnen FPGAs (max. 10% Auslastung) und die lange Reprogrammierungszeit spricht jedoch gegen diese Lösung. Eingesetzt werden diese Systeme bei der Prozessorentwicklung. [8,4]

3 Umsetzung des Projekts

3.1 Übersicht

Die Aufgabe war die Aufteilung eines Multi-Prozessors in einen simulierten und emulierten Teil. Bisher waren vier Prozessoren an den Bus angeschlossen. Einer von diesen sollte nun simuliert werden. Die Schnittstelle zwischen dem Bus und der Foreign Language Interface (FLI)-Schnittstelle wurde bereits im WS06/07 erstellt. Der simulierte Prozessor wird mit Hilfe des Simulators ModelSim simuliert.

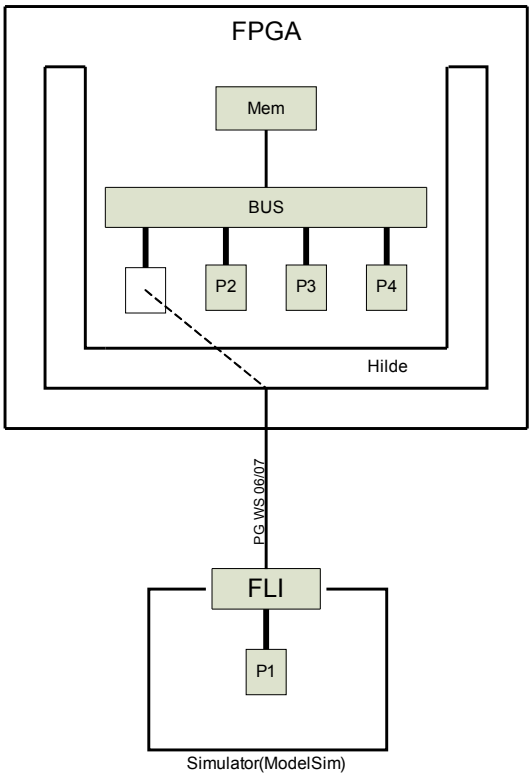

Abbildung 3.8: Ausgliederung eines Prozessors

RAPTOR2000:

Das Rapid-Prototyping System RAPTOR2000 wurde am Heinz-Nixdorf Institut vom Fachgebiet Schaltungstechnik entwickelt. Es dient zur prototypischen Implementierung mikroelektronischer Schaltungen. Es lassen sich Schaltungs- und Systementwürfe mit einer Komplexität von bis zu 60 Millionen Transistoren realisieren.

Projektplanung:

Als erstes wurde mit Hilfe von ModelSim, die Schnittstelle für den simulierten Prozessors geschaffen. Die Signale vom Bus wurden in das Programm zur Simulation mit eingebunden. Die entstanden Dateien wurden dann in einem Programm HiLDE übersetzt. Die Synthese wurde zuerst mit Synopsys DC-FPGA 2005 durchgeführt und in einer späteren Projektphase auf Xilinx ISE umgestellt, welches nach der Synthese für das Übersetzen, Platzieren und Verdrahten eingesetzt worden ist. Die Daten wurden mit Hilfe von RAPTOR2000GUI auf das RAPTOR-Bord geladen und mit der TestBench gestestet. Dieses Vorgehen ist dargestellt in der Abbildung 3.9: Übersicht über Projekt.

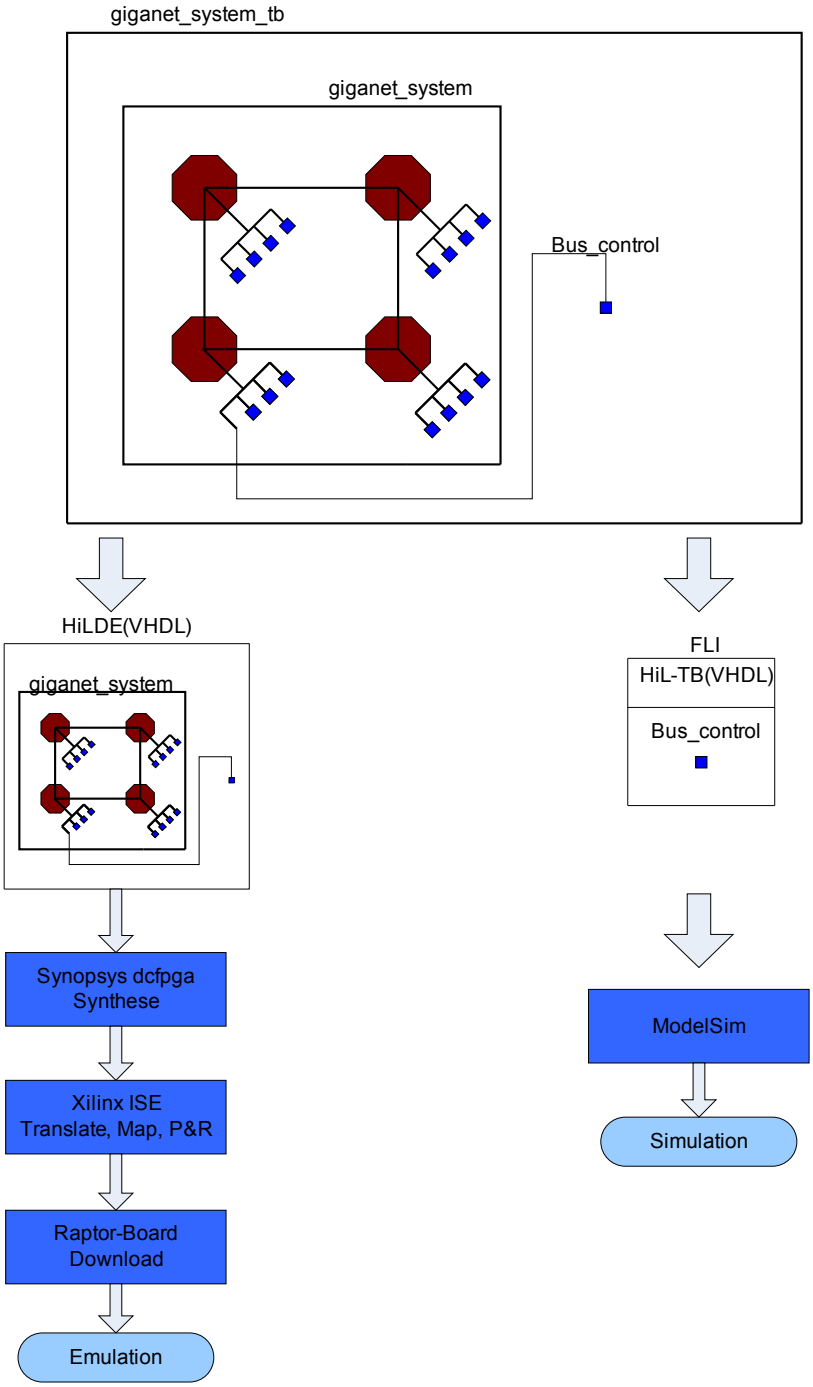

Abbildung 3.9: Übersicht über Projekt

3.2 ModelSim

Programmbeschreibung:

ModelSim wird von der Mentor Graphics Corporate vertrieben. Diese sind einer der Führer im Bereich der Electronic-Design-Automation (EDA). Es werden Lösungen für Hardware und Software angeboten.

ModelSim ist eine „mächtige“ Simulationsumgebung für HDLs. Diese erlaubt die taktsynchrone oder timinggenaue Simulation von digitalen Logikelementen. [10]

Hierarchische Struktur:

Das Cluster_System besteht aus verschiedenen Modulen. An der untersten ist die Verbindung mit dem physikalischen Bus über das bus_control gegeben. Auf der direkt höheren Ebene folgt das Modul ncore_wb, welches den Prozessor beschreibt. Darüber befindet sich die Ebene Pe_Cluster. Dieses beinhaltet die 4 Prozessoren. Das nächst höhere ist das Modul Cluster, mit pe_cluster, einer switch box und einem Speicherelement. Die oberste Ebene ist das cluster_system, welches aus 4 Clustern besteht.

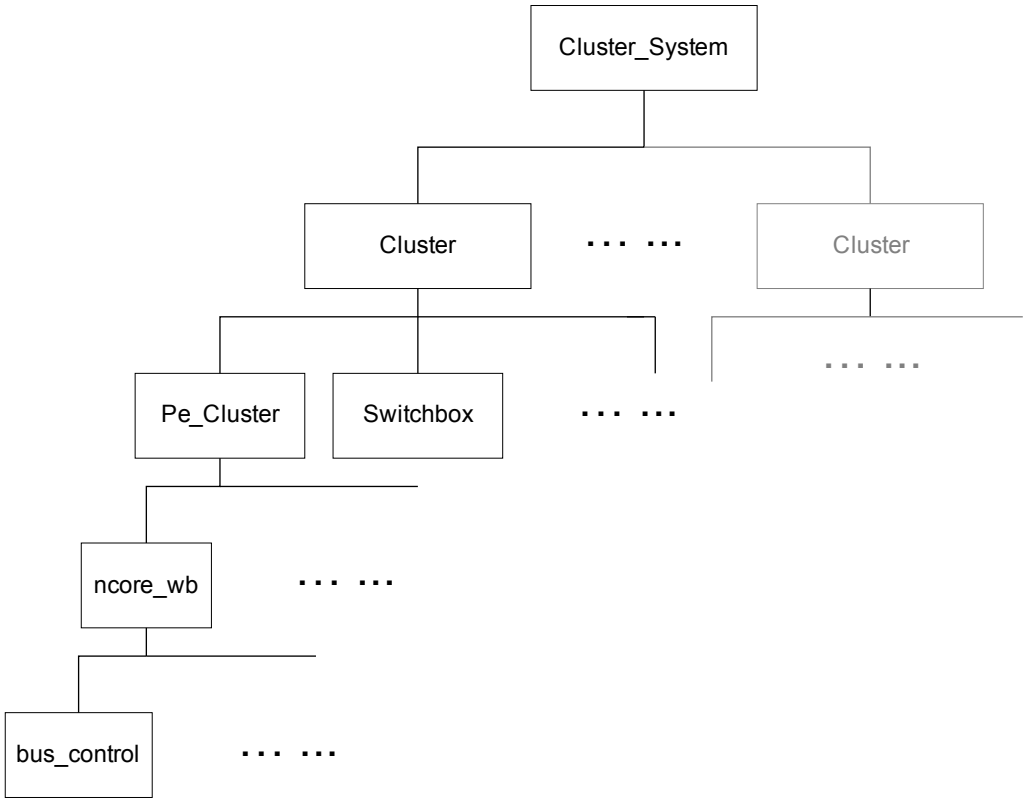

Abbildung 3.10: Hierarchische Struktur des Systems

Einen Prozessor ausgliedern:

Der erste Schritt, zur Ausgliederung eines Prozessors um diesen simulieren zu können, ist das Herausführen der Signale für diesen Mikroprozessor von der untersten Schicht der ncore_wb an die hierarchisch höchste Ebene des Programms, die des cluster_system. Die Signale welche herausgeführt wurden können der Abbildung 3.11: Anschlüsse/Signale des NCORE – Prozessors oder der Tabelle 3-1: Top Signale mit den Beschreibungen entnommen werden.

Signale/Variablen	Variablenart	Beschreibung
ncore_CLK_top	out std_logic	Takt
ncore_RESET_top	out std_logic	Reset
ncore_addr_top	in std_logic_vector (31 downto 0)	Adressleitung
ncore_data_bus_in_top	out std_logic_vector(31 downto 0)	Daten ein
ncore_data_bus_out_top	in std_logic_vector(31 downto 0)	Daten aus
ncore_r_w_top	in std_logic	lese/schreib zugriff
ncore_tsiz_top	in std_logic_vector(1 downto 0)	Speicherzugriffsbreite
ncore_tc_top	in std_logic_vector(2 downto 0)	Adressübersetzung
ncore_treq_top	in std_logic	Buszugriff
ncore_tbusy_top	in std_logic	Busbelastung
ncore_ta_top	out std_logic	erfolgreicher Buszugriff
ncore_tea_top	out std_logic	Zugriffsfehler

Tabelle 3-1: Top Signale

Durch das Herausführen von Eingängen und Ausgängen müssen bei den verschiedenen Ebenen noch Libraries mit eingebunden werden oder neue libraries erstellt werden. Diese sind im Anhang B mit den dazugehörigen Dateinamen zu finden.

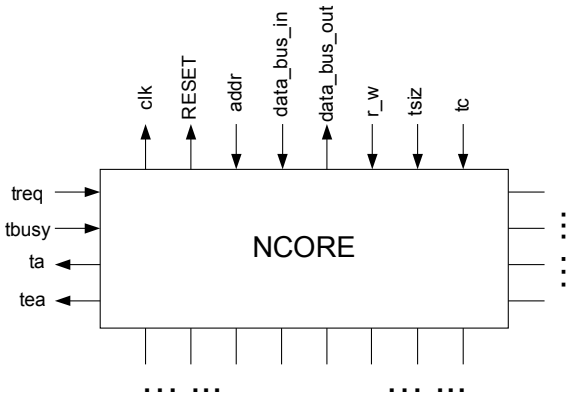

Abbildung 3.11: Anschlüsse/Signale des NCORE – Prozessors

Test Bench:

Zum Testen des bereits programmierten Teils wurde eine TestBench „giganet_system_tb“ erstellt. In dieser mussten die Signale zuerst deklariert, zugewiesen und später im Wave Fenster dargestellt.

Die verwendeten Signale sind gleich der Herausgeführten und stehen in der Tabelle 3-1: Top Signale.

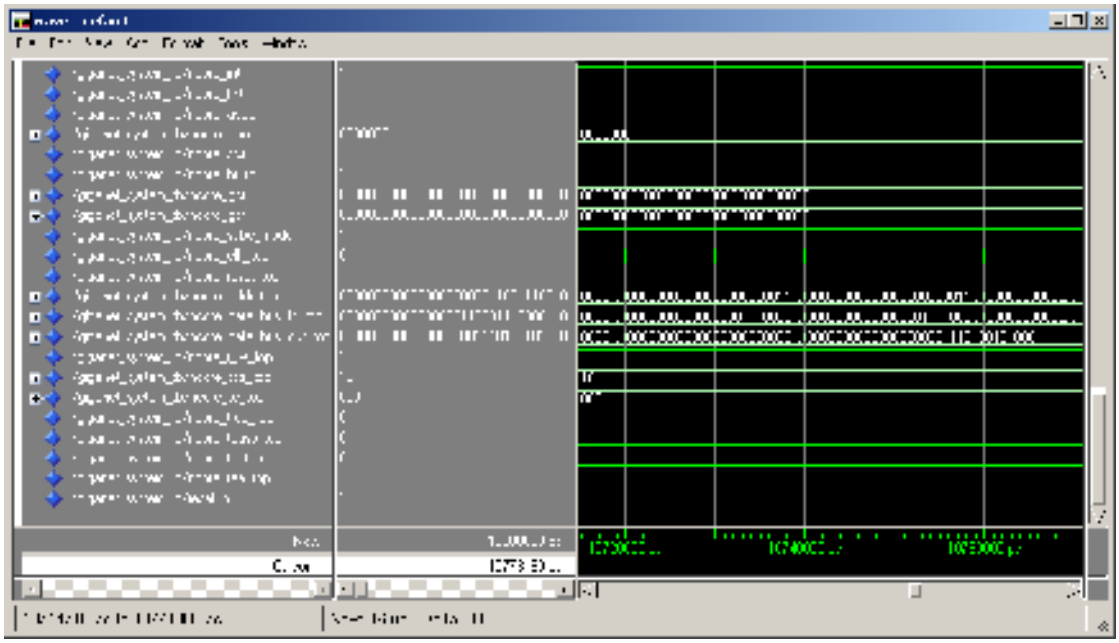

Abbildung 3.12: Wave Fenster von ModelSim

Im Abbildung 3.12: Wave Fenster von ModelSim sind ein Teil der Signale zu sehen, welche dargestellt werden. Die untersten Signale im Bild, sind die Top-Signale welche neu hinzugefügt worden sind.

3.3 HiLDE

Programmbeschreibung:

Hardware-in-the-Loop Design Environment (HiLDE) ist ein Programm, welches den Vorteil der Software Simulation mit dem des Rapid-Prototyping verbindet. Das Programm wurde an der Universität Paderborn im Fachbereich Schaltungstechnik entwickelt.

Programmablauf:

Folgende Änderungen mussten zuerst vorgenommen werden, damit HiLDE keinen Fehler erzeugt. Es dürfen keine Generics in der entity vorhanden sein. Es mussten somit die Variablen durch die entsprechenden Zahlenwerte ersetzt werden.

Zusätzlich musste die Library „virtex2_mem_lib“ durch den Befehl „--synopsys translate_off„ herausgenommen werden. Nach der Codezeile bei dem die Library genutzt wird, muss der Befehl „-synopsys translate_on“ erfolgen.

HiLDE kann keine „inout“ Befehle verarbeiten. Die Variablen, welche eine Input und Output besitzen, mussten in zwei Variablen aufgeteilt werden und später wieder zu einer zusammengeführt werden.

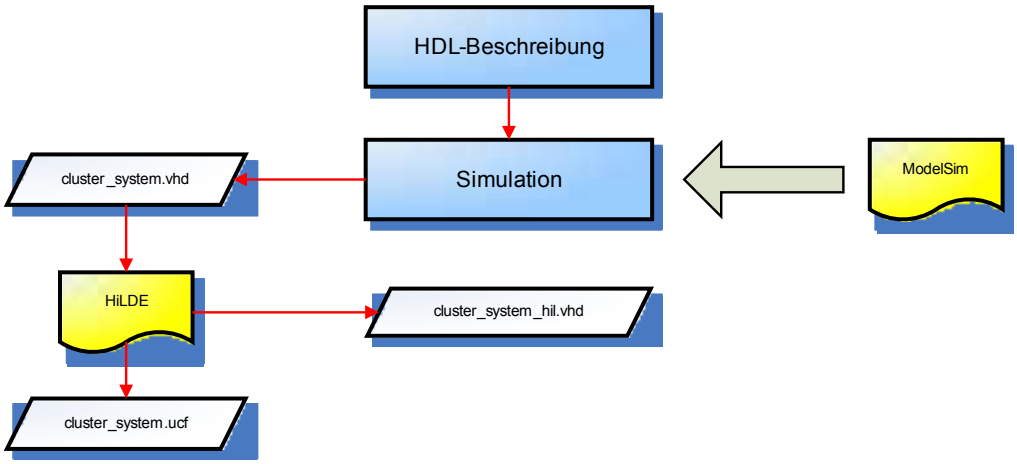

Abbildung 3.13: Ablauf

Als erstes erfolgt beim Arbeiten mit HiLDE, die Einstellung des Pfade und der Parameter. Zu den Parametern gehört das richtigen Datenformate und die Einstellung Optimize IO, die genaue Beschreibung wird im Anhang erläutert.

Danach wird von der HiLDE die benötigten „cluster_system_WRAP.vhd“, „cluster_system_.ucf“ Dateien und „cluster_system_Header.h“ erzeugt.

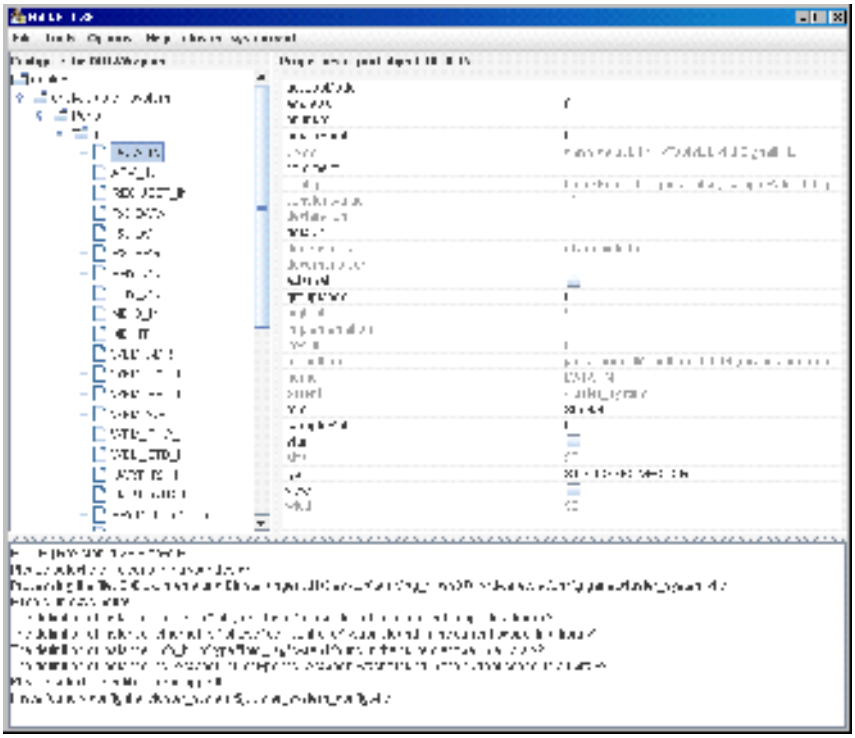

Abbildung 3.14: HiLDE: Ports

Die erzeugte Wrap-Datei muss noch Korrigiert werden. Als erstes muss der entity Name gleich der Datei Name sein. Dazu wurde die Datei cluster_system_wrap in cluster_system_hil umbenannt.

Die Bibliothek UniSim muss durch den Befehl Translate_off vor dem Synthetisieren ausgeschlossen werden.

Zusätzliche Bibliotheken müssen in die erstellte Datei cluster_system_hil eingebunden werden. Wichtig: Die Bibliothek raptor_lib muss hier erstmals eingebunden werden.

Einen weiteren Fehler machen Signale die den Wert Null zugewiesen bekommen. Zur Behebung dieses Fehlers muss ein Signal zero eingeführt werden, welches den Wert Null erhält. Die Signale, welchen den Wert Null zugewiesen bekommen, bekommen die neue definierte Variable zero zuweisen.

Die zwei „LINT_N“ Zuweisungen müssen noch herausgenommen werden. Diese werden nicht benutzt und führen bei der Synthetisierung sonst zu Fehlern.

Zusätzlich wurden noch Signale für die Switch Box, Physikalische Ports und zum Schreiben und Lesen auf das RAPTOR Bord eingebunden. Zum Schreiben und Lesen mit dem RAPTOR Bord müssen zusätzlich noch die entsprechenden Komponenten mit angelegt werden.

3.4 Design Compiler for FPGA

Programmbeschreibung:

Die Synthese wird mit Hilfe der Software Electronic-Design-Automation-(EDA)-Software von Synopsys® DCFPGA 2005 durchgeführt. Diese benötigt die Plattform Unix. Die Synthese ist ein Vorgang um aus einer abstrakten Beschreibung einer logischen Schaltung, welche in einer VHDL-Datei gespeichert ist, eine Netzliste der Schaltung zu erzeugen.

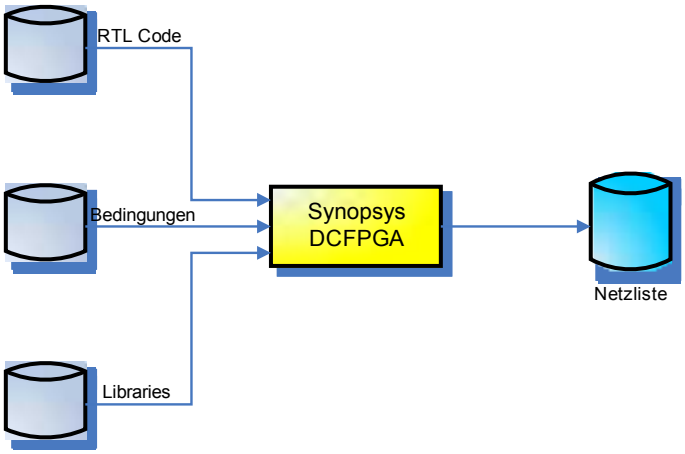

Abbildung 3.15: Beschreibung des Ablaufs bei Synopsys DCFPGA

DCFPGA wurde auf Basis der Synopsis-Design-Compiler-Technologie entwickelt und beinhaltet die Adaptive-Optimization™-Technologie und die Topographische Technologie.

Die Adaptive-Optimization-Technology aktiviert automatisch den Core-Synthese-Algorithmus, welcher die Parameter der Schaltungsgrößen, Schaltungstopologie und der

Timing-Constraints beinhaltet und dann dynamisch steuert, wie und in welcher Reihenfolge die Algorithmen angewandt werden sollen. Die mit DCFPGA entwickelten Schaltungen laufen im Durchschnitt 15% schneller als jene, die mit anderen FPGA-Synthese-Tools entwickelt wurden.

Die Topographische Technologie, versetzt den Entwickler in die Lage das Post-Layout-Timing und die Chipfläche schon während der RTL-Syntheseakkurat vorherzusagen. [21, 22]

Programmablauf:

Die Synthese wird wie oben beschrieben durchgeführt.

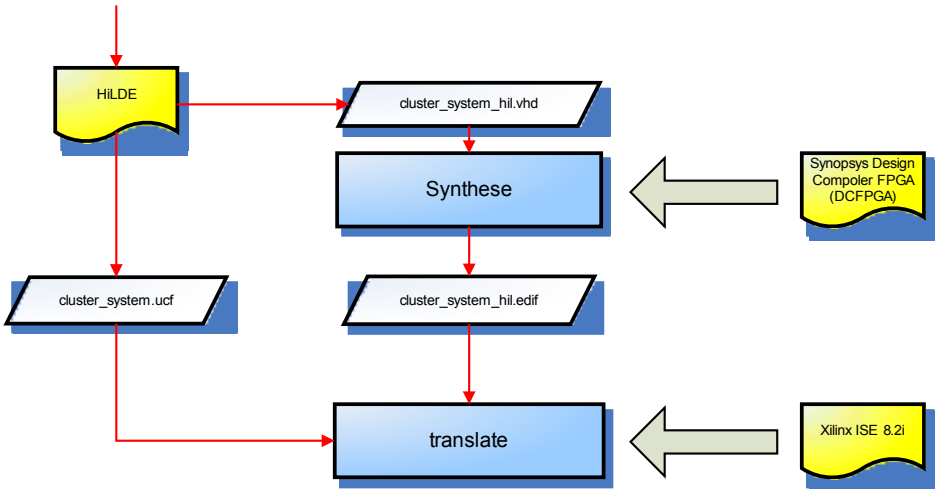

Abbildung 3.16: Ablauf der Synthese

Nach der Synthese werden die synthetisierten Daten in einer Electronic-Design-Interchange-Format (EDIF)- Datei gespeichert und können dann mit Xilinx ISE weiter verarbeitet werden.

In einer späteren Phase des Projekts erfolgte die Umstellung, dass jetzt mit Xilinx ISE das Projekt synthetisiert werden konnte. Die Synthese mit Xilinx ISE erfolgte schneller und war nicht mehr so fehleranfällig, wie die mit Synopsys. Deshalb und weil das Übersetzen, Plazieren und Verdrahten bereits mit Xilinx ISE durchgeführt worden ist, erfolgte die Umstellung.

3.5 Core-Generator

Programmbeschreibung:

Das CORE-Generator System ist ein Entwurfswerkzeug, das die parametrisierten Speicher liefert, die für Xilinx FPGAs optimiert werden. Es unterstützt eine Menge von gebrauchsfertigen Funktionen, zu diesen gehören einmal die einfachen arithmetischen Operatoren wie Addierer, Akkumulator, und Multiplizierer, sowie komplette System Bausteine wie Filtern, Wandlern, FIFOs und Speicher.

Programmablauf:

Bevor der Speicher erzeugt werden kann muss der C-Code mit welchem die Prozessoren laufen sollen durchkompiliert werden und ein Maschineller-Code erzeugt werden. Diese ASCII-Datei muss noch ein wenig geändert werden, damit diese vom

Xilinx CORE-Generator verwendet werden kann. Dafür wurde ein Linux-Skript geschrieben, welches am Ende des Kapitels erklärt wird.

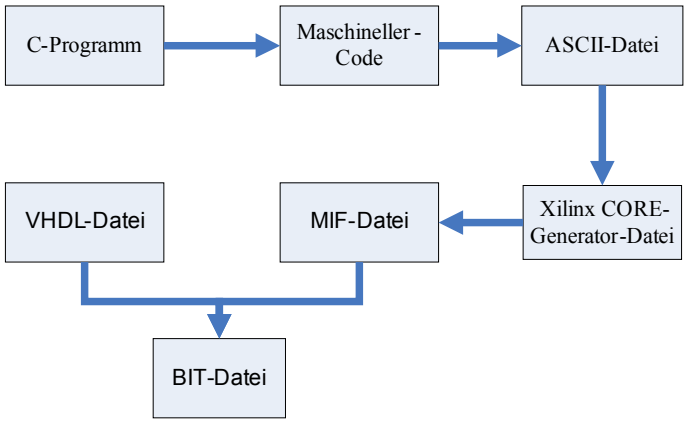

Abbildung 3.17: Übersicht über Speicher-Datei

Bei der Speichererstellung mit Xilinix CORE-Generator wird eine Vielzahl von Dateien erzeugt. Die benötigten Dateien sind:

bram_8k_8_8_01.mif

bram_8k_8_8_01.edn

bram_8k_8_8_01.vhd

Die verwaltete Speicher-Information werden in der Managed Information Format (MIF)-Datei beschrieben. In der EDIF Implementation Netlist (EDN)-Datei steht wie der Speicher implementier werden soll. Die VHD-Datei beschreibt die Architektur der Speicher-Datei. Der Inhalt eines bram_8k_8 ist vier Speicher-Dateien abgelegt. Die Verarbeitung der Speicherinhalte wurde in „memory_interface.vhd“ programmiert und dazu noch das Lesen aus dem Speicher in der Datei „k7b321825m.vhd“ angepasst.

Linux-Script:

Das Linux-Script vereinfacht die Umstellung der erzeugten Dateien ASCII-Datei in die von Xilinx CORE-Generator benötigte COE-Datei. Beim Ausführen des Linux-Skript werden zuerst alle Dateien, welche sich in dem angegebenen Ordner befinden, in eine COE-Datei umbenannt. Danach erfolgen zwei Zeilen indem das Format und der Anfang der Speicherwerte angezeigt werden. Zusätzlich werden noch die fehlenden Kommas am Zeilenende eingefügt und am Ende der Speicherwerte die Datei mit dem Strichpunkt abgeschlossen. Die Datei kann nun von Xilinx CORE-Generator geladen werden.

3.6 Xilinx ISE

Programmbeschreibung:

Xilinx ist einer der größten Entwickler und Hersteller von programmierbaren Logik ICs. Das Programm ISE ist eine front-to-back FPGA Designlösung für das Betriebssystem Linux und Windows. Es besteht aus Xilinx SmartCompile Technologie, einem CORE-Generator™ System sowie einem FPGA Editor.

Der große Vorteil der Synthese mit Xilinx ISE ist, dass es neben der Synthese immer auch eine Überprüfung der Daten stattfindet. So wird nach der Design Eingabe das

System auf die Funktionalität getestet. Es folgt die Synthese, mit nochmaliger Funktionsverifikation, bevor das Design implementiert wird. Dabei findet eine statische Zeitanalyse und Zeitsimulation statt. Im letzten Schritt werden die Daten auf die Xilinx Einheit geladen und es wird eine zyklische Überprüfung durchgeführt. [23]

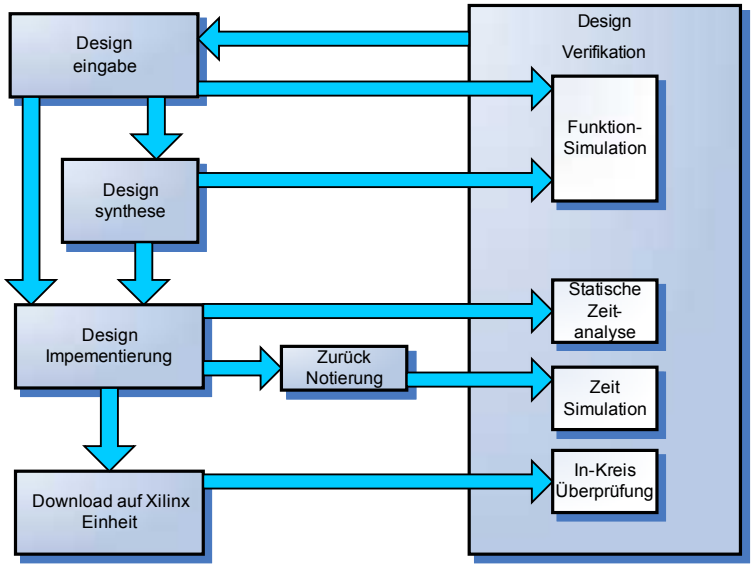

Abbildung 3.18: Verlauf mit ISE

Programmablauf:

Als erstes wurde ein neues Projekt erstellt mit der edif-Datei der Synthese, sowie der ucf-Datei, welche von HiLDE generiert worden ist. Zusätzlich müssen noch die Dateien mit den Informationen für die verschiedenen Speicher in das Projekt importiert werden.

Nach der Umstellung der Synthese von bisherigen DCFPGA zu Xilinx, wurde nicht mehr die edif-Datei benötigt, sondern es konnte direkt die vhd-Datei benutzt werden, um zuerst die Synthese und danach das Plazieren durchzuführen.

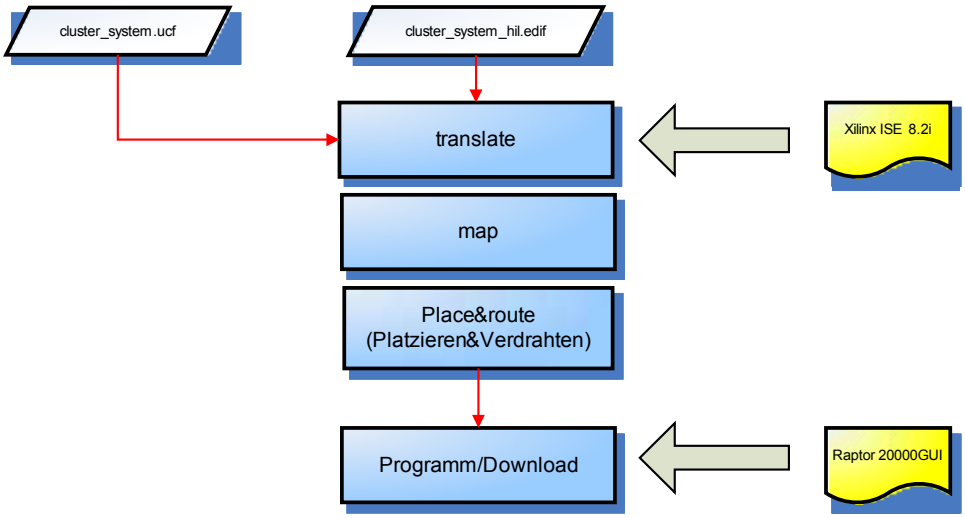

Abbildung 3.19: Ablauf Xilinx ISE

Dann wird mit der ISE das Projekt übersetzt. Beim Plazieren und Verdrahten, wird einmal die interne Verdrahtung und die Verdrahtung zu den Eingangs- und Ausgangspins durchgeführt, sowie die Extrahierung von realen Zeitparametern aus der Geometrie, siehe Abbildung 3.19: Ablauf Xilinx ISE.

3.7 RAPTOR2000

Programmbeschreibung:

Für die prototypische Implementierung mikroelektronischer Schaltungen hat das Fachgebiet Schaltungstechnik am Heinz Nixdorf Institut das Rapid-Prototyping-System RAPTOR2000 entwickelt.

Mit dem RAPTOR2000 System lassen sich Schaltungs- und Systementwürfe mit einer Komplexität von mehr als 100 Millionen Transistoren realisieren. Das System ist Modular aufgebaut, welche 6 Modulsteckplätze beinhaltet, mit jeweils 456 Signalleitungen. Jeder Modulsteckplatz besitzt frei konfigurierbare Signalleitungen zu benachbarten Modulen mit einer Bandbreite von mehr als 1GByte/s. Die FPGA-Module werden über den PCI-Bus konfiguriert. [3, 5]

Programmablauf:

Mit dem Programm RAPTOR2000GUI wird die fertig geroutete Datei, welche „cluster_system_hil.bit“ ist, auf den Multiprozessor geladen. Die Datei wird auf das Modul 3 des Multiprozessors gespeichert.

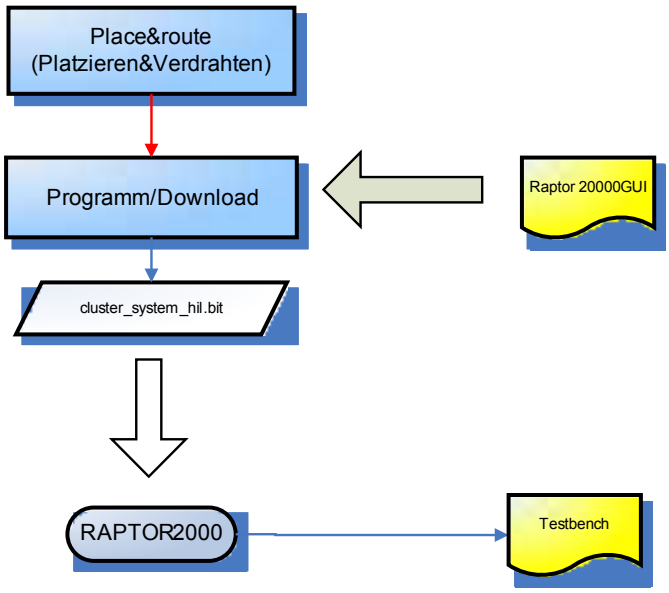

Abbildung 3.20: Daten auf RAPTOR2000 laden

3.8 Foreign Language Interface (FLI)

Der verwendete Simulator ModelSim wird mit einer Bibliothek Foreign Language Interface geliefert. Die FLI Bibliothek stellt eine Sammlung von C-Funktionen zur externen Steuerung dar, so dass es in anderen Programmiersprachen mit den in ModelSim erstellten Modellen zusammenzuarbeiten ermöglicht. Der Simulator ModelSim bietet eine bequeme Schnittstelle zwischen Simulationsumgebung und

Hardware-Treiber. In unserem Projekt wird die Foreign Language Interface für die Verbindung zwischen Raptor2000 Betriebssystem Treibern und der Testumgebung (ModelSim) benutzt und als ein so genannter ModelSim-Treiber implementiert wird. Der Treiber bearbeitet die Signale zwischen dem Raptor2000 und dem Simulator.

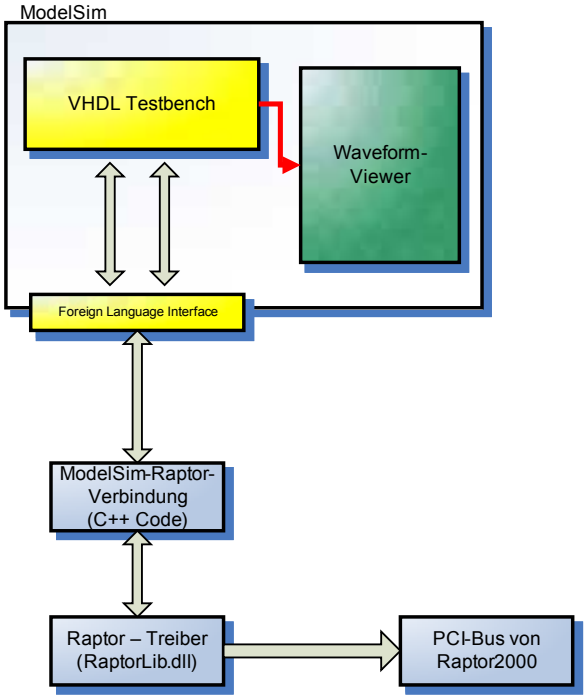

Abbildung 3.21: Aufgabe von FLI-Datei

4 Ergebnisse

4.1 Testprogramm

Das Testprogramm, welches in C geschrieben worden ist, testet alle vier Prozessoren in einem Cluster der Giganet Systeme. Die vier Prozessoren laden jeweils die Informationen aus dem ihnen zugewiesenen Register und schreiben in das nächst tieferem Register. Nur der Prozessor eins schreibt in das Register von Prozessor 4. Damit laufen die Daten einmal im Kreis. Die Register werden am Anfang des Programms zuerst mit Null initialisiert. Der Prozessor 1 addiert seinen Registerinhalt mit eins. Der zweite Prozessor multipliziert die Daten mit zwei, der dritte Prozessor addiert auf die Daten drei und der vierte Prozessor multipliziert mit fünf.

4.2 Simulation

Zuerst wurden die Ergebnisse eine Simulation des Programms und der TestBench überprüft.

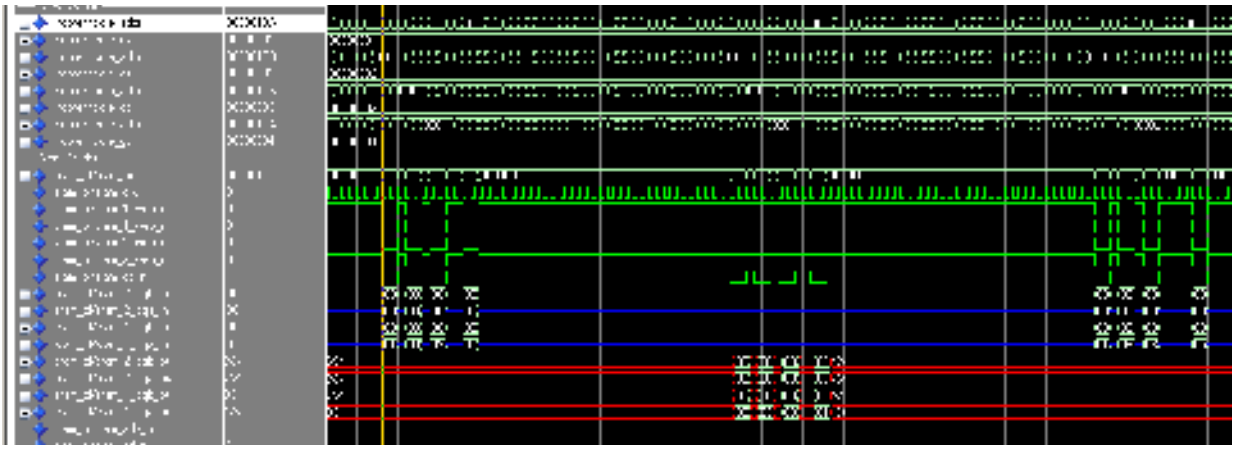

Abbildung 4.22: Simulations-Ergebnisse

Bei der Simulation erfolgten alle Lese- und Speichezugriffe, wie es im C-Programm programmiert worden ist, siehe dazu Abbildung 4.22: Simulations-Ergebnisse. In dieser Abbildung ist ein Durchlauf mit Lesen und Speichern der Daten der vier Prozessoren in die jeweiligen Register zu sehen.

4.3 Reale Prozessoren

Die Ergebnisse mit den realen Prozessoren lieferte das gleiche Ergebnis, wie die Simulation. Die Daten wurden von den jeweiligen Prozessoren aus den Registern gelesen, verarbeitet und in das richtige Register zurück geschrieben. Dies ist in der Abbildung 4.23: HIL-Ergebnisse dargestellt.

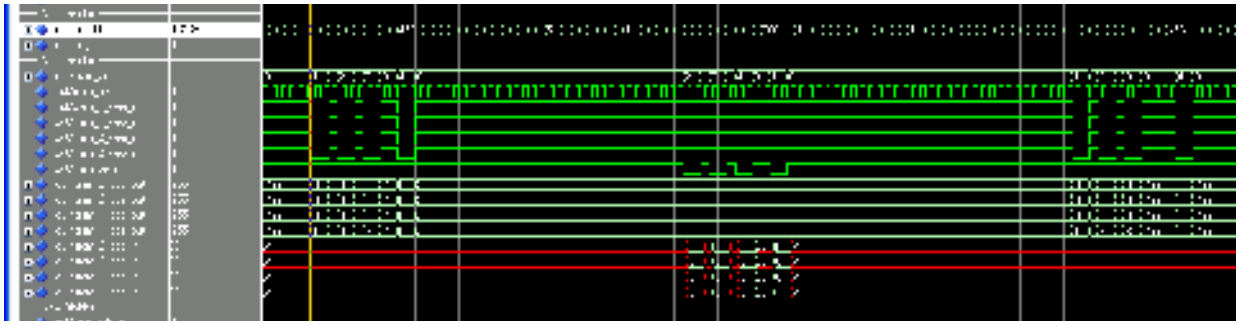

Abbildung 4.23: HIL-Ergebnisse

4.4 Messung

Die erste Messung wurde ohne SRAM und mit Takt und dem Reset direkt aus der TestBench durchgeführt und nicht durch das System zurückgeführt. Damit wurde eine Frequenz von 1,2272kHz ermittelt. Bei der zweiten Messung wurde der SRAM mit eingebunden und der Takt und das Reset wurden vom System geliefert. Die Messung ergab, dass das System mit einer Frequenz von 150,7719Hz lief. Die dritte Messung war Messung der Zeit bei reiner Simulation. Dabei wurde eine Frequenz von 3,739Hz gemessen.

Messung	Zeit	Takte	Frequenz
1	81,481668s	100 000	1,2272kHz
2	663,253280s	100 000	150,7719Hz
3	267,445388s	1 000	3,739Hz

4.5 Vergleich HIL zu ursprünglichem System

Der verwendete FPGA ist im HIL-System sowie im ursprünglichen System ein Virtex2 6000. Mit Xilinx ISE wurden folgende Daten für die verschiedenen Systeme ermittelt:

	MIL	Ursprüngliche
Anzahl genutzter Slices	21 881 (64%)	20 123 (59%)
Block RAMs	80	80
Gesamtanzahl äquivalente Gatter	5 583 572	5 554 861
Anzahl verbundener IOs	199 (24%)	285 (34%)
Max. Frequenz	21MHz	15MHz

5 Zusammenfassung und Ausblick

Die Projektgruppe Hardware-in-the-Loop hat einen Prozessor aus dem Cluster Verbund herausgeführt und diesen simuliert. Es mussten neben dem herausführen der Signale durch alle Hierarchie-Ebenen, noch weitere Änderungen wie diese zum Beispiel am Speicher vorgenommen werden. Das System lief somit mit einem simulierten und einen emulierten Teil. Die Ergebnisse ergaben, dass das System im Vergleich zur reinen Simulation um ein vielfaches schneller lief. Der Erwartete Geschwindigkeitsvorteil, welcher am Anfang des Projekts vermutet worden ist, wurde somit durch die Messungen bestätigt.

Der Speicher wurde um einige Parameter vereinfacht. Die Anpassung des alten Speichers an das aktuelle HIL-Projekt, wäre eine sinnvolle Erweiterung. Zusätzlich könnten Ergebnisse am Raptorbord an den vorhandenen Pins, mit Hilfe eines Demonstrators, oder mit LEDs zur Anzeige gebracht werden. Eine weitere mögliche Erweiterung ist das System um mehrere Cluster zu vergrößern. Damit vervielfältigt sich die Komplexität.

Das Programm HiLDE kann in Zukunft noch erweitert werden, damit zum Beispiel die Generics mit HiLDE benutzt werden können.

Abkürzungsverzeichnis

ASCET	Advanced Simulation and Control Engineering Tool
ASCET-	ASCET-Modeling und Design
ASCET- RP	ASCET-Rapid-Prototyping
ASCET-SD	ASCET-Software-Development
CAMeL	Computer-Aided Mechatronic Laboratory
DSC	Dynamik System Code
DUCADE	Domain Unified CAD Environment
EASY5	Engineering Analysis System Version 5
EDA	Electronic-Design-Automation-
EDIF	Electronic-Design- Interchange-Format
EDN	EDIF Implementation Netlist
FCS LSI	Future Combat Systems Lead Systems Integrator
FLI	Foreign Language Interface
HIL	Hardware-in-the-Loop
HiLDE	Hardware-in-the-Loop Design Environment
IPANEMA	Integration Platform for Networked Mechatronic Application
MIF	Managed Information Format
ODSL	Objective-Dynamic System Language
ODSS	Objective-Dynamic-System Structure
OOAD	Object-Oriented Analysis and Design
SIL	Software-in-the-Loop

Literaturverzeichnis

[1] D. Langen; Studie S-Core; Februar 2001; Universität Paderborn; Heinz Nixdorf Institut; Schaltungstechnik

[2] D. Langen; J. Niemann; M. Porrmann, H. Kalte, U. Rückert; Implementation of a RISC Processor Core for SoC Design – FPGA Prototype vs. ASIC Impementation; Universität Paderborn; Heinz Nixdorf Institut; Schaltungstechnik

[3] RAPTOR2000; Datenblatt DB-V2 Modul; Universität Paderborn; Heinz Nixdorf Institut; Schaltungstechnik

[4] C. Paiz, C. Pohl, M. Porrmann; Hardware-in-the-loop simulations for FPGA-based digital control design; Universität Paderborn; Heinz Nixdorf Institut; Schaltungstechnik

[5] RAPTOR2000; http://www.raptor2000.de/ ; Universität Paderborn; Heinz Nixdorf Institut; Schaltungstechnik

[6] P. Klingebiel, R. Diekmann, U. Lefarth, M. Fischer, J. Seuss; CAMeL/PVM: An Open, Distributed CAE Environment for Modelling an Simulation Mechtronic Systems; Universtiät Paderborn

[7] Peter Wältermann; Der serielle Hybridantrieb: Vom rechnergestützten Entwurf bis zur Hardware-in-the-Loop-Realisierung; Dissertattion; Universität-Gesamtschule Paderborn; 14. November 2000

[8] A. Burst; Rapid-Prototyping eingebetteter elektronischer Systeme auf Basis des CDIF-Datenaustauschformats; Dissertattion; 15. Februar 2000;Institut für Technik der Informationsverarbeitung Universität Karlsruhe

[9] B. Spitzer; Modellbasierter Hardware-in-the-Loop Test von eingebetteten elektronischen Systemen; Dissertattion; 18. Dezember 2001; Fakultät Elektrotechnik und Informationstechnik der Universität Fridericiana Karlsruhe

[10] M. Tahedl; Anleitung zu: VHDL-Simulation mit Modelsim; April 2003; Universität Ulm, Abteilung Allg. Elektrotechnik und Mikroelektronik

[11] A. Strey; Rechnerentwurf – Kapitel 7: Synthese; Skipt WS 2006/07; Universität Innsbruck, Institut für Informatik

[12] Boeing Integrated Defense Systems; Network & Space Systems Backgrounder; Februar 2007

[13] MSC Software; Easy5; http://www.mscsoftware.com/products/easy5.cfm?Q=396&Z=399

[14] P. Wältermann, H. Schütte, K. Dieckstall; Hardware-in-the-Loop-Test verteilter Kfz-Elektroniksysteme; dSpace GmbH

[15] ETAS; ASCET-SD – effiziente Software-Entwicklung trotz steigender Komplexität; ETAS Engineering Tools

[16] Wind River; Wind River Field Diagnostics

[17] Wind River; The MathWorks, Wind River Sign Definitive Agreement for The MathWorks to Distribute, Support Matrixx

[18] ATZelektronik: Automatisierter Closed-Loop-Testprozess für Steuergerätefunktionen; Januar 2007

[19] A. Schlosser, A. Bollig, D. Abel; Rapid Control Prototyping in der Lehre; GMA Kongress 2003, VDI-Berichte 1756

[20] J. O. Hamblen; Using Synthesis, Simulation, and Hardware Emulation to Prototype a Pipelined RISC Computer System; School of Electrical and Computer Engineering Georgia Institute of Technology

[21] Synopsys; Pressemitteilung; Synopsys präsentiert Design Compiler 2005; München 23. Mai 2005

[22] Synopsys Präsentiert Design Compiler 2005; Synopsys; München, 23. Mai 2005; „http://www.hbi.de/clients/Synopsys/pms05/23_05/DesignCompiler05.php“

[23] Xilinx ISE; http://www.xilinx.com/ise/logic_design_prod/webpack.htm

Abbildungsverzeichnis

ABBILDUNG 1.1: AUFGABENSTELLUNG.....3
ABBILDUNG 2.2: SIMULATION UND SYSTEMEINSATZ [8].....5
ABBILDUNG 2.3: V-MODELL [8].....6
ABBILDUNG 2.4: ZEITLICHE ÜBERSICHT.....8
ABBILDUNG 2.5: REALE DATEN, NASTRAN UND DAS EASY5 PROGRAMM [13].....9
ABBILDUNG 2.6: INTEGRIERTE TESTUMGEBUNG AUTOMATION DESK.....10
ABBILDUNG 2.7: ETAS:ASCET 11
ABBILDUNG 3.8: AUSGLIEDERUNG EINES PROZESSORS.....14
ABBILDUNG 3.9: ÜBERSICHT ÜBER PROJEKT.....15
ABBILDUNG 3.10: HIERARCHISCHE STRUKTUR DES SYSTEMS 16
ABBILDUNG 3.11: ANSCHLÜSSE/SIGNALE DES NCORE – PROZESSORS.....17
ABBILDUNG 3.12: WAVE FENSTER VON MODELSIM.....18
ABBILDUNG 3.13: ABLAUF.....19
ABBILDUNG 3.14: HILDE: PORTS.....19
ABBILDUNG 3.15: BESCHREIBUNG DES ABLAUFS BEI SYNOPSYS DCFPGA.....20
ABBILDUNG 3.16: ABLAUF DER SYNTHESE.....21
ABBILDUNG 3.17: ÜBERSICHT ÜBER SPEICHER-DATEI.....22
ABBILDUNG 3.18: VERLAUF MIT ISE.....23
ABBILDUNG 3.19: ABLAUF XILINX ISE.....23
ABBILDUNG 3.20: DATEN AUF RAPTOR2000 LADEN.....24
ABBILDUNG 3.21: AUFGABE VON FLI-DATEI.....25
ABBILDUNG 4.22: SIMULATIONS-ERGEBNISSE.....26
ABBILDUNG 4.23: HIL-ERGEBNISSE.....27
ABBILDUNG ANHANG.24: ÜBERSICHT ÜBER ABLAUF.....34
ABBILDUNG ANHANG.25: VHDL-DATEI LADEN.....40
ABBILDUNG ANHANG.26: AUSWAHL DER VHDL-DATEI.....40
ABBILDUNG ANHANG.27: DATEI AUSFÜHREN.....41

ABBILDUNG ANHANG.28: AUSWAHL DER RICHTIGEN DATEIFORMATS.....41
ABBILDUNG ANHANG.29: ANSICHT AUF PORTS.....42
ABBILDUNG ANHANG.30: XILINX 1.....43
ABBILDUNG ANHANG.31: XILINX 2.....43

Anhang A

In der folgenden Abbildung ist, der gesamte Ablauf von der VHDL-Beschreibung bis zum Laden des Programms auf das RAPTOR-Board dargestellt. Die genaue Beschreibung des Ablaufs ist im 3. Kapitel dargestellt.

Übersicht:

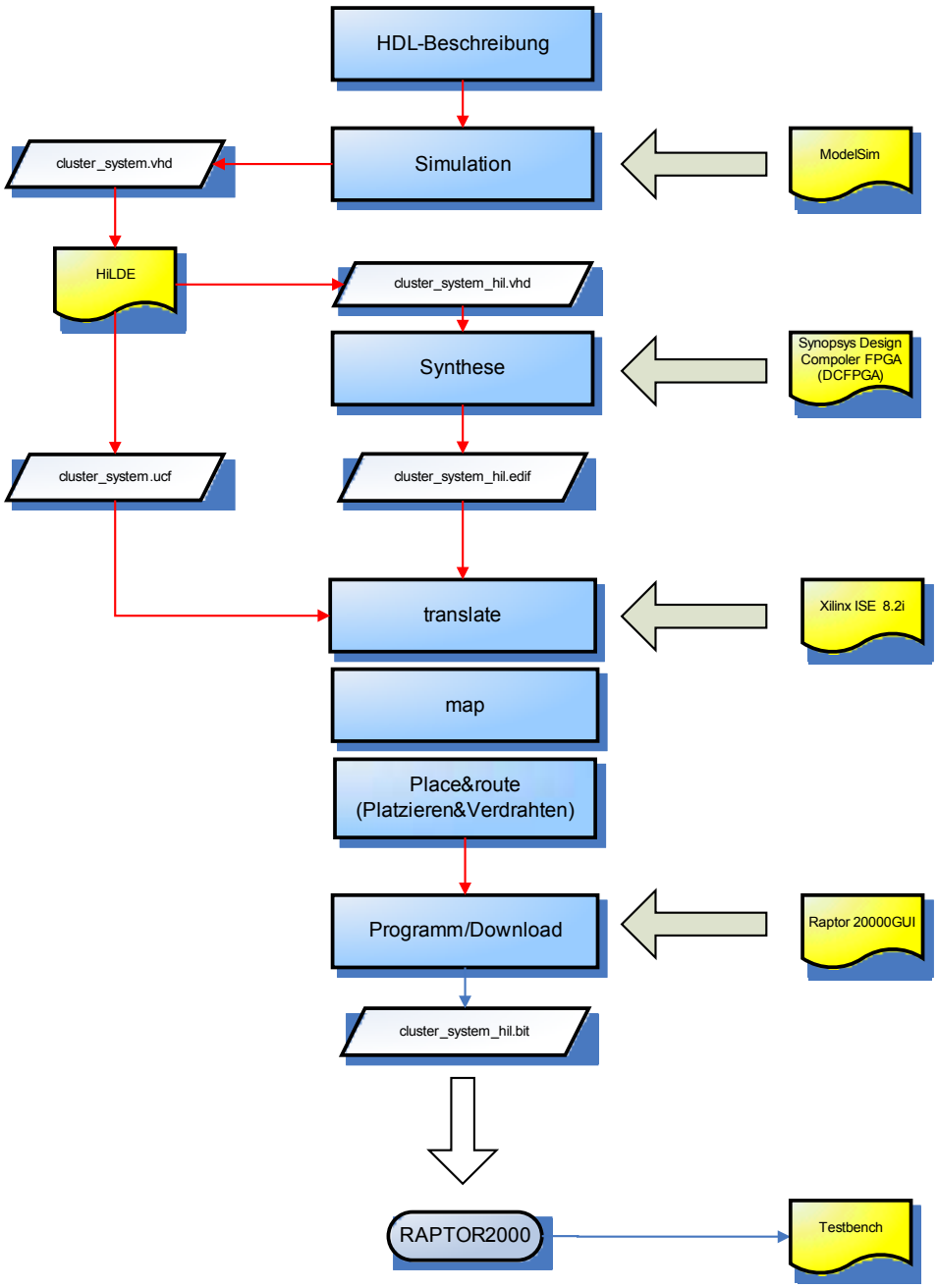

Abbildung Anhang.24: Übersicht über Ablauf

Übersicht über die Signale mit ihren Adressen und ihren Typen-Bezeichnungen, welche im FLI-Interface stehen.

Signal	in/out	Typ	Adresse
CLK	in	std_logic	0x00000020
RESET_N	in	std_logic	0x00000024
SRAM_ADR	out	std_logic_vector(20 downto 0)	0x00000028
SRAM_ADV_N	out	std_logic	0x0000002c
SRAM_ADSP_N	out	std_logic	0x00000030
SRAM_ADSC_N	out	std_logic	0x00000034
SRAM_1_GW_N	out	std_logic	0x00000060
SRAM_2_GW_N	out	std_logic	0x00000064
SRAM_1_BW_N	out	std_logic	0x00000068
SRAM_2_BW_N	out	std_logic	0x0000006c
SRAM_ZZ	out	std_logic	0x00000070
SRAM_LBO_N	out	std_logic;	0x00000074
SRAM_CS_1_N	out	std_logic	0x00000040
SRAM_CS_2	out	std_logic	0x00000044
SRAM_CS_2_N	out	std_logic	0x00000048
SRAM_1_WEa_N	out	std_logic	0x0000004c
SRAM_1_WEb_N	out	std_logic	0x00000050
SRAM_2_WEa_N	out	std_logic	0x00000054
SRAM_2_WEb_N	out	std_logic	0x00000058
SRAM_OE_N	out	std_logic	0x0000005c
SRAM_CLK	out	std_logic	0x00000038
SRAM_CLK_FB	in	std_logic	0x0000003c
SRAM_1_DQPa_in	in	std_logic	0x00000098

SRAM_1_DQPb_in	in	std_logic	0x000000a0
SRAM_2_DQPa_in	in	std_logic	0x000000a8
SRAM_2_DQPb_in	in	std_logic	0x000000b0
SRAM_1_DQPa_out	out	std_logic	0x0000009c
SRAM_1_DQPb_out	out	std_logic	0x000000a4
SRAM_2_DQPa_out	out	std_logic	0x000000ac
SRAM_2_DQPb_out	out	std_logic	0x000000b4
SRAM_1_DQa_out	out	std_logic_vector(7 downto 0);	0x0000007c
SRAM_2_DQa_out	out	std_logic_vector(7 downto 0)	0x00000084
SRAM_1_DQb_out	out	std_logic_vector(7 downto 0)	0x0000008c
SRAM_2_DQb_out	out	std_logic_vector(7 downto 0)	0x00000094
SRAM_1_DQa_in	in	std_logic_vector(7 downto 0)	0x00000078
SRAM_2_DQa_in	in	std_logic_vector(7 downto 0)	0x00000080
SRAM_1_DQb_in	in	std_logic_vector(7 downto 0)	0x00000088
SRAM_2_DQb_in	in	std_logic_vector(7 downto 0)	0x00000090
ncore_addr_top	in	std_logic_vector(31 downto 0)	0x000000c0
ncore_data_bus_out_top	in	std_logic_vector(31 downto 0)	0x000000c8
ncore_CLK_top	out	std_logic	0x000000b8
ncore_RESET_top	out	std_logic	0x000000bc
ncore_r_w_top	in	std_logic	0x000000cc
ncore_tsiz_top	in	std_logic_vector(1 downto 0)	0x000000d0
ncore_tc_top	in	std_logic_vector(2 downto 0)	0x000000d4
ncore_treq_top	in	std_logic	0x000000d8
ncore_tbusy_top	in	std_logic	0x000000dc
ncore_ta_top	out	std_logic	0x000000e0
ncore_tea_top	out	std_logic	0x000000e4

ncore_data_bus_in_top	out	std_logic_vector(31 downto 0)	0x000000c4

Anhang B

Im Folgenden ist ein Code-Ausschnitt aus ncore_wb zu sehen, welches Beispielhaft herausgenommen worden ist um die Einführung der neuen Top-Signale zu zeigen, welche für den simulierten Prozessors benötigt werden.

```
-------------------------------------------------------------------------------
-- ENTITY
-------------------------------------------------------------------------------
entity ncore_wb is
  generic (CPU_ID     : integer := 0;
              &
  port (NCORE_CLK  : in  std_logic;
              &
-- Definierung, zum herausfuehren der Signale
        ncore_CLK_top          : out  std_logic;
        ncore_RESET_top        : out  std_logic;
        ncore_addr_top         : in std_logic_vector(31 downto 0);
        ncore_data_bus_in_top  : out  std_logic_vector(31 downto 0);
        ncore_data_bus_out_top : in  std_logic_vector(31 downto 0);
        ncore_r_w_top          : in std_logic;
        ncore_tsiz_top         : in std_logic_vector(1 downto 0);
        ncore_tc_top           : in std_logic_vector(2 downto 0);
        ncore_treq_top         : in std_logic;
        ncore_tbusy_top        : in  std_logic;
        ncore_ta_top           : out  std_logic;
        ncore_tea_top          : out  std_logic;
        &);
end ncore_wb;
&
begin
-- N-CORE
&
 -- Zuweisung, zum herausfuehren der Signale
   gen_0: if ROW = 0 and COLUMN = 0 and CPU_ID = 1 generate
             ncore_CLK_top           <= NCORE_CLK;
             ncore_RESET_top         <= ncore_rst;
             ncore_addr              <= ncore_addr_top;
             ncore_data_bus_in_top   <= ncore_data_bus_in;
             ncore_data_bus_out      <= ncore_data_bus_out_top;
             ncore_r_w               <= ncore_r_w_top;
             ncore_tsiz              <= ncore_tsiz_top;
             ncore_tc                <= ncore_tc_top;
             ncore_treq              <= ncore_treq_top;
             ncore_tbusy             <= ncore_tbusy_top;
             ncore_ta_top            <= ncore_ta;
             ncore_tea_top           <= ncore_tea;
```

Die librarys welche in cluster_system_hil benötigt werden, sind in den folgenden Codezeilen zu entnehmen. Die library IEEE ist bereits in cluster_system vorhanden. Die library UniSim muss wie im Kapitel 3 beschrieben aus der Übersetzung herausgenommen werden. Die weiteren librarys wurden nach der Durchführung mit HILDE, wieder ergänzt.

```
cluster_system_hil:

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_misc.all;
--synopsys translate_off
library UniSim;
use UniSim.all;
--synopsys translate_on
library fpga_sb_lib;
library giganet_lib;
use giganet_lib.all;
use giganet_lib.giganet_pckg.all;
use fpga_sb_lib.switch_box_pckg.all;
library raptor_lib;
use raptor_lib.all;
```

C-Programm, welches die Prozessoren ausführen:

```
#define MY_ADDR 0x30000000
#define NO_OF_CPUS 4
void malloc_init()
{       void printu(unsigned long value)
        int main()
        unsigned long value;
        unsigned long write_target;
        unsidned int cpu_id = ncore_get_id();
        *((unsigned int*)(MY_ADDR + cpu_id*4)) = 0;
        write_target = cpu_id -1;
        if(write_target == 0) write_target = 4;
        while(cpu_id < 5)
        {       cpu_id = ncore_get_id();
                value = *((unsigned int*)(MY_ADDR + cpu_id*4));
                if(cpu_id == 1) value = value*5;
                if(cpu_id == 2) value = value +1;
                if(cpu_id == 3) value = value *2;
                if(cpu_id == 4) value = value +3;
                *((unsigned int*)(MA_ADDR + write_target *4)) = value;
        }
        return 1;
}
```

Anhang C

HiLDE:

HiLDE wird mit folgendem Pfad „…\dist>java –jar HILDE.jar“ gestartet. Danach wird die zu generierende Datei „cluster_system.vhd“ ausgewählt. Dazu wie im Bild gezeigt, den Befehl „OpenVHDL_Source“ anklicken.

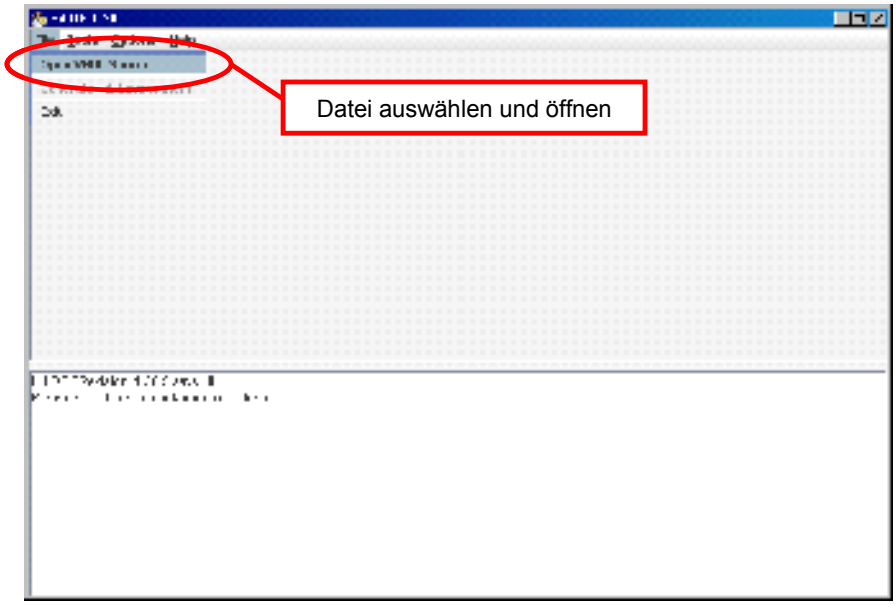

Abbildung Anhang.25: VHDL-Datei laden

Mit Doppelklick auf „cluster_system“ werden auf der rechten Seite „Selected Entities“ die möglichen auszuwählenden Dateien angezeigt.

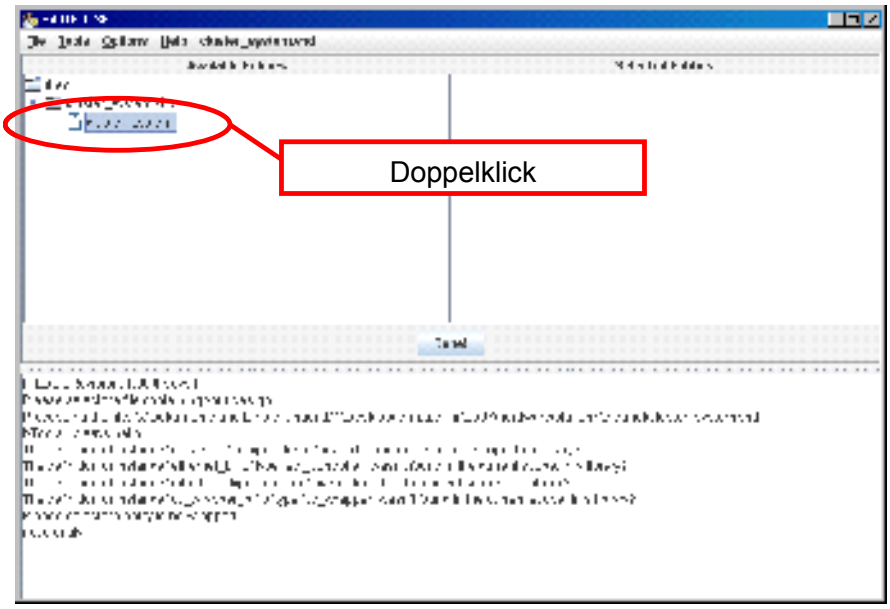

Abbildung Anhang.26: Auswahl der VHDL-Datei

Auf der rechten Seite „Selected Entities“ kann nun die Datei ausgewählt werden. In unserem Beispiel ist dies „cluster_system“ und mit dem „Done“-Befehl die Datei ausgeführt werden

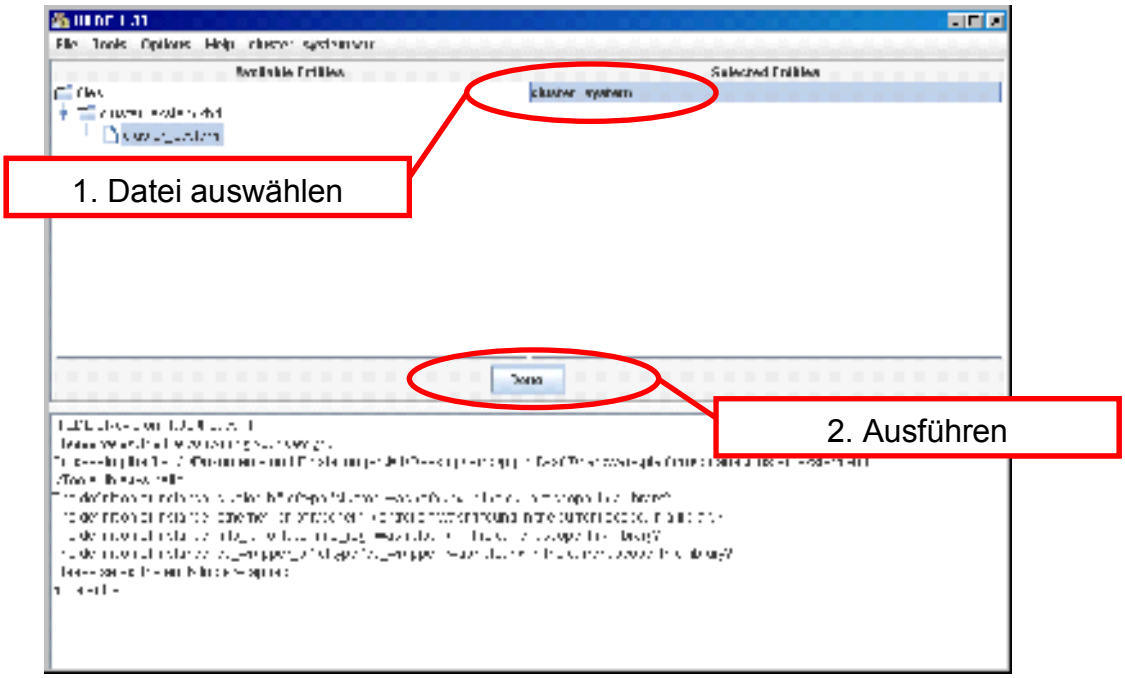

Abbildung Anhang.27: Datei ausführen

Unter Options muss jetzt noch das richtige Format eingestellt werden. Bei uns war dies das Format XC2V.

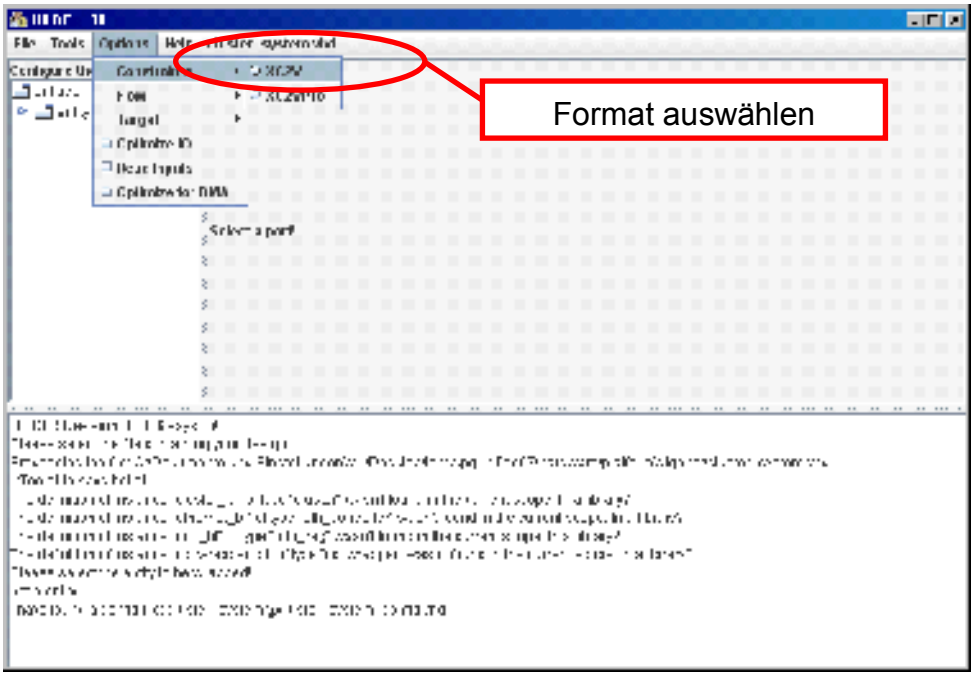

Abbildung Anhang.28: Auswahl der richtigen Dateiformats

Mit „Generate HIL Environment“, welcher in der Fensterleiste File zu finden ist. Werden die benötigten Dateien generiert. Bei unserem Projekt wurden die ersten beiden Dateien benötigt.

- „cluster_system_Wrap.vhd,
- „cluster_system.ucf“,
- „cluster_system@cluster_system_config.vhd“
- „cluster_system_SFunc.m“
- „cluster_system_Labels.m“

Danach können die verschiedenen Ports mit ihren Eigenschaften kontrolliert werden.

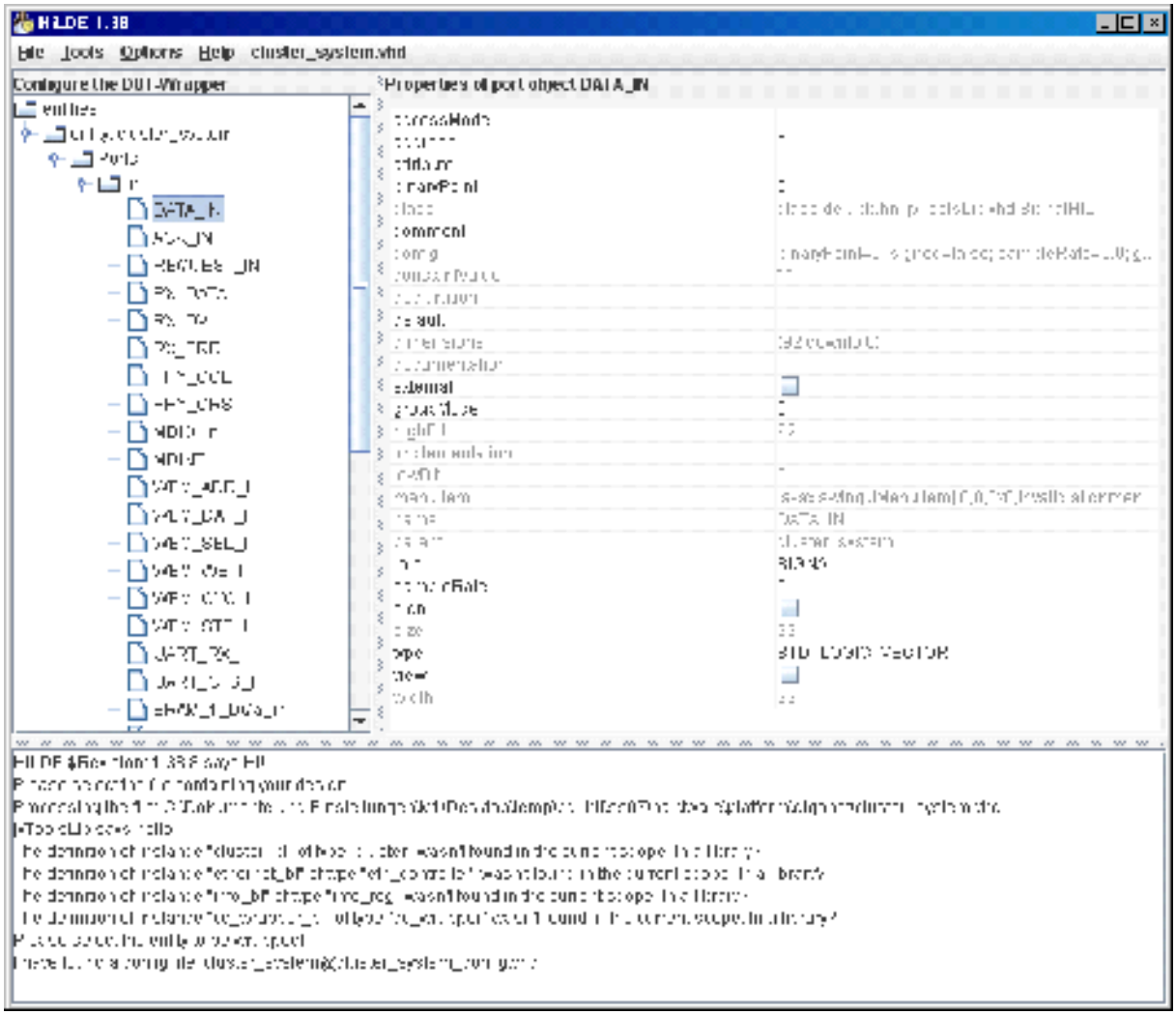

Abbildung Anhang.29: Ansicht auf Ports

Xilinx:

Bei der Xilinx ISE mussten die im Bild dargestellten Parameter eingestellt werden:

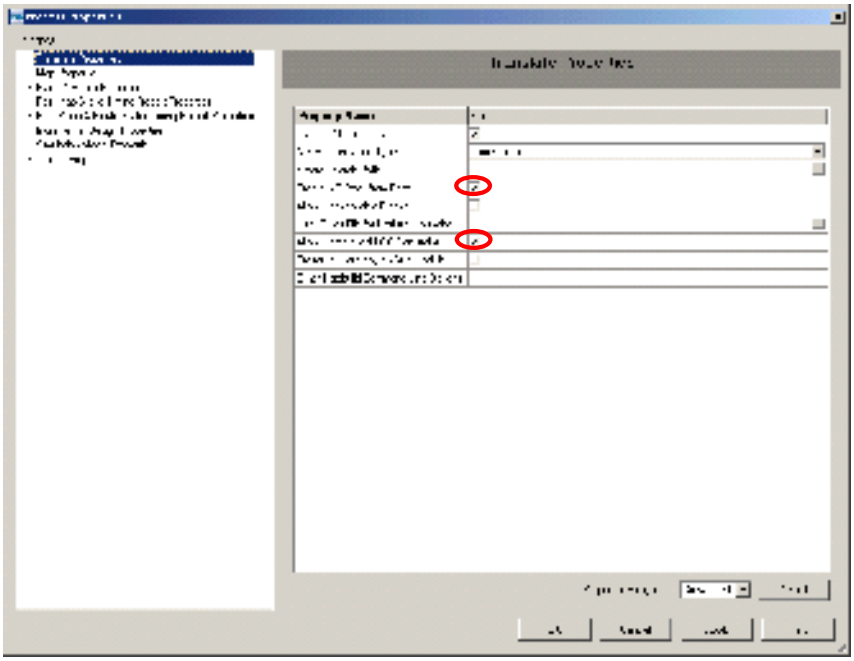

Abbildung Anhang.30: Xilinx 1

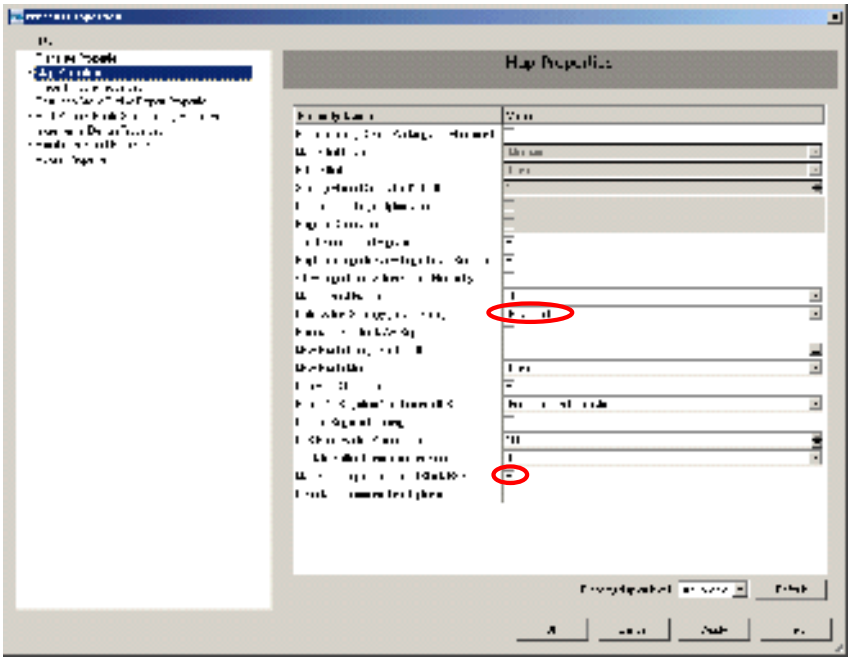

Abbildung Anhang.31: Xilinx 2