FAKULTÄT FÜR !NFORMATIK

Robot4Web

Ein vom Anwender programmierbares, auf dem Internet basierendes Steuerungs- und Demonstrationssytem für Roboter

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Informatik

eingereicht von

Sandor Biro
Matrikelnummer 9327589

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: o. Univ.-Prof. Dr. Dietmar Dietrich
Mitwirkung: Dipl.-Ing Herbert Nachtnebel

Wien, 28.10.2008 _____ _____
(Unterschrift Verfasser) (Unterschrift Betreuer)

Abstract

The Project Robot4Web is dedicated for didactical and demonstrative purposes. The goal of this diploma thesis is to create a robot which is attached to a FPGA board and may be controlled by it. Additionally, a predefined application programmed into the FPGA may be used to control the robot directly via a serial interface. The robot movements and its environment may be observed via a Webcam which is also attached to the FPGA. This allows it to use the robot for research in the area of artificial intelligence.

The robot is connected to a PC via the programming cable of the FPGA and directly over a serial link. The complete development system for programming the FPGA and therefore the robot is realized in a Web application, allowing it to use the system in an e-learning setup.

Kurzfassung

Robot4Web ist ein Projekt, erstellt für didaktische und demonstrative Zwecke. Ziel dieser Diplomarbeit ist: einen Roboter herzustellen, der in erster Linie über ein FPGA-Board programmierbar und steuerbar ist. Darüber hinaus besteht die Möglichkeit, den Roboter mit Hilfe einer in FPGA programmierte Anwendung, über eine serielle Schnittstelle direkt anzusteuern. Sämtliche Bewegungen des Roboters, aber auch seine Umgebung werden mit einer Webkamera erfasst, die ebenfalls mit dem FPGA verbunden ist. Durch diese Tatsache wird ermöglicht, dass in weiterer Folge Forschungsarbeiten auf dem Gebiet der künstlichen Intelligenz durchgeführt werden können.

Die Anbindung des Roboters an den PC wird über ein FPGA-Programmierkabel und über ein RS-232-Kabel stattfinden. Die gesamte Entwicklungsumgebung für die FPGA-Programmierung, und dadurch für den Roboter, ist mit Hilfe einer Webanwendung realisiert worden. Damit wird die Einsetzbarkeit des Systems im Bereich des E-Learning ebenfalls ermöglicht.

Danksagung

Ich möchte mich für das Verständnis, das mir meine Frau und mein Sohn entgegengebracht haben, mein Studium abschließen zu können, herzlich danken.

Ich möchte mich noch bei meinem Betreuer, Herrn DI Herbert Nachtnebel für die kompetente und immer bereite Unterstützung auch bedanken.

Abkürzungen

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Device
DCE	Data Communication Equipment
DCI	Digitally Controlled Impedance
DCM	Digital Clock Manager
DCR	Bus Device Control Register Bus
DTE	Data Terminal Equipment
EDIF	Electronic Design Interchange Format
FPGA	Field Programmable Gateway Array
FSM	Finite State Machine
IC	Integrated Circuit
LUT	Look UP Table
NCD	Native Circuit Description
NFS	Network File System
NGD	Native Generic Database
NTFS	New Technology File System
PCB	Printed Circuit Board
PWM	Pulsweitenmodulation
RTL	Register Transfer Layer
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver / Transmitter

UCF	User Constraints File
UUT	Unit Under Test
VHDL	Very High-Level Definition Language
VPN	Virtual Private Network

Inhaltsverzeichnis

Kapitel 1 **Einführung**

Das Internet ist heutzutage als eines der wichtigsten Kommunikationsmedien anzusehen, und die zur Verfügung stehende Bandbreite wächst stetig. Es existiert eine Reihe von Technologien zur Programmierung von Internetanwendungen und es werden vermehrt Systeme entwickelt, wo mit Hilfe von internetbasierten Technologien Teleoperatoren und Roboter angesteuert werden. Die Entwicklungsgeschichte der heutigen Industrieroboter führt bis zum Jahre 1940 zurück. Zunächst wurden Automaten entwickelt, die Sicherheit in nukleare Forschungslaboratorien verbessern sollten. Die späteren Weiterentwicklungen waren in der Lage, verschiedene, sich wiederholende Aufgaben durchzuführen. Die gegenwärtig produzierten Roboter sind mit besseren Sensoren und leistungsstarken Steuerungen ausgestattet, dadurch ist es möglich geworden, Roboter in der Medizin, im Halbleiterbau, Weltraumforschung und in anderen, mitunter höchste Präzision erforderlichen, Technologiebereichen einzusetzen.

Obwohl die Einsatzmöglichkeiten von Roboter kaum begrenzt sind, verwenden nur wenige Exemplare die Funktechnologie oder das Medium Internet für die Kommunikation mit einer externen Steuereinheit. Dies ist begründet durch einen großen Unterschied zwischen den Internettechnologien und modernen, eingebetteten Systemen, den sogenannten Embedded Systems. Während die eingebetteten Systeme in der Regel echtzeitfähig sind, fällt bei näherer Betrachtung der Internettechnologien auf, dass hier das Übertragungsprotokoll (TCP/IP, UDP, ...) und viele andere Komponente keine, oder nur eingeschränkte Echtzeitfähigkeiten besitzen.

Ziel dieser Diplomarbeit ist es, ein sechsachsiges Robotersystem für online Schulungen im Bereich FPGA (feldprogrammierbares Gatterarray), Design und Robotik zu entwerfen und zu realisieren. Das Robotersystem wird auf FPGA-Basis aufgebaut, und soll per Internet programmier-, sowie konfigurierbar sein.

Diese Plattform soll es Kursteilnehmern ermöglichen, Algorithmen sowie die Steuerungshardware des Roboters selbstständig zu Hause zu realisieren und mit Hilfe der vorbereiteten Entwicklungsumgebung zu prüfen. Für die Beschreibung der Steuerungshardware sollen Hardwarebeschreibungssprachen wie beispielsweise VHDL (Very High Speed Integrated Circuit Hardware Description Language) verwendet werden.

Das Robotersystem gliedert sich in die folgenden fünf, voneinander abhängigen, Komponenten: Den Webserver, den Job Scheduler, den Rechner für die Synthese und Simulation, das FPGA-Board und der Roboterarm.

Der bei diesem Projekt eingesetzte Webserver verwendet Microsofts Internet Information Services (IIS). Alle auf dem Webserver laufende Webanwendungen und Web Services werden mit ASP.NET in C# entwickelt. Die wichtigsten Aufgaben des Webservers im Robotersystem sind: Benutzerverwaltung, Darstellung des Prozesszustandes über das Internet, möglichst in „Echtzeit“, und das Bereitstellen von Mechanismen, damit der Benutzer transparent auf externe Programme zugreifen kann.

Der Job Scheduler wird eingesetzt, um die Kommunikation zwischen dem Webserver und den PC´s, auf denen die Simulation und die Synthese laufen, zu koordinieren. Der Job Scheduler löst die Steuerungs- und Verteilungsaufgaben des Systems, indem er auf die, durch externe Prozesse ausgelösten Ereignisse reagiert. Diese Ereignisse werden von ausgeführten Prozessen auf dem Webserver oder auf einem PC im Robotersystem erzeugt. Aufgaben des Job Schedulers sind: mehrere Client-PC`s an einen Webserver zu binden, das Starten von ausführbaren Dateien gemäß der jeweiligen Benutzeranfragen durchzuführen, und die Rückgabewerte bzw. die Meldungen an den Webserver weiterzuleiten. Über eine grafische Oberfläche kann der Betreuer die Benutzerzugriffe kontrollieren, oder aber die Kontrollmechanismen gänzlich dem Job Scheduler überlassen.

Damit keine Fehler entstehen die die Systemintegrität bedrohen könnten, werden, bevor noch das Benutzerprogramm an das FPGA-Board weitergeleitet wird, umfangreiche Tests an diesem vorgenommen. Neben den Tests werden die Simulation und die Hardwaresynthese ausgeführt. Für diese Zwecke werden weitere PC´s eingesetzt, die mit dem FPGA-Board direkt verbunden sind und den Job Scheduler ausführen. Diese PC´s müssen Windows 2000, Windows XP oder Windows Vista als Betriebssystem verwenden, und im lokalen Netzwerk Zugriff auf den Webserver haben.

Der Roboterarm wird mit Hilfe von Servomotoren realisiert, da diese höheren Drehmomente liefern, und mit einer höheren Drehzahl betrieben werden können. Um die Steuerung des Roboters auch ohne direkten Sichtkontakt zu ermöglichen, werden Webcams in das System integriert, die ein Bild des Roboterarms am Bildschirm anzeigen. Die Daten dieser Webcams sollten gleichzeitig auch dem FPGA-Board übergeben werden, damit in weiterer Folge Forschungsarbeiten auf dem Gebiet der Bildererkennung durchgeführt werden können.

Um den Roboterarm anzusteuern, verwendet das Robotersystem ein Xilinx Virtex2 Pro Entwicklungsboard.

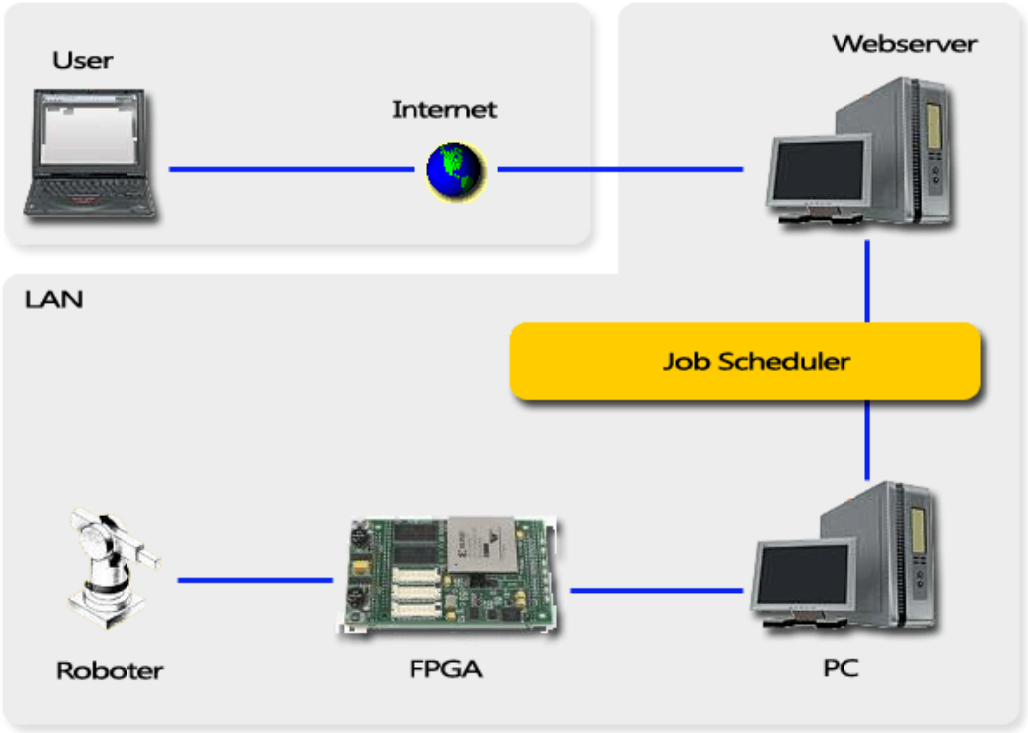

Abbildung 1: Die Robot4Web Topologie

1.1 Einsatz von Robotern und FPGAs in E-Learning

Roboter werden seit langem in der industriellen Produktion verwendet, und zwar meistens als Insellösung für spezielle Aufgabenbereiche. Durch die Weiterentwicklung der Internet- und Kommunikationstechnologien werden immer häufiger Roboter entwickelt, die universal einsetzbar sind; die nicht mehr vor Ort programmierbar sein müssen. Sie werden durch das Internet überwacht, und aus der Ferne programmiert. Neben dem industriellen Einsatz wurden Roboter und Robotersteuerungssysteme für das E-Learning entwickelt, um die Benutzung von teueren Hardware und Software möglichst leicht und effizient vermitteln zu können.

Das Technische Institut für Computertechnik auf der Technischen Universität Wien beschäftigt sich bereits seit längerer Zeit mit dem Design integrierter Schaltungen und deren Anwendung in E-Learning. Im Rahmen des Projekts „axis4web“ können Studenten über einen einfachen Webbrowser von zu Hause aus ein Digitaldesign auf das - am ICT befindlichen - FPGA-Board uploaden, die Taster und Schalter des Boards betätigen, und den Zustand der 7-Segment-Anzeigen beobachten.

Eine weitere typische Anwendung von Roboter in E-Learning ist das „eLearning-Programm Lektor zur Handhabungstechnik“. Lektor Handhabungstechnik ist für die berufliche Bildung in Schulen, Fachschulen, Fachhochschulen, bzw. in der Praxis konzipiert. Das Lernsoftware eignet sich besonders gut für die Ausbildung von Industriemechanikern und Mechatronikern, sowie für die Mitarbeiterschulung im Betrieb. Die einzelnen Module können zum selbstständigen Lernen in Einzelarbeit oder auch in Kleingruppen eingesetzt werden[23].

In dieser Diplomarbeit wird eine E-Learning Umgebung für die FPGA-Programmierung, und für die, mit dieser verbundenen Robotersteuerung geschaffen. Über diese Umgebung können zukünftige Studenten die in der Industrie etablierte Software für die FPGA-Programmierung kennenlernen und verwenden, und anschließend die Ergebnisse auf einen reellen Roboter umsetzen.

1.2 Abgrenzung

In dieser Diplomarbeit werden einige wichtige Aspekte der Robotersteuerung über das Internet wie Echtzeitigkeit, Zuverlässigkeit und Sicherheit nur marginal behandelt. Diese Bereiche betreffen nicht nur Hardware-Systeme, die online angesteuert werden, sondern alle zeitkritische Informationsübertragungen. Echtzeitfähigkeit, Zuverlässigkeit und Sicherheit sind die Grundlagen von neuen Internetgenerationen, und an der Weiterentwicklung der Methodik wird derzeit intensiv geforscht.

1.3 Gliederung der Arbeit

Obwohl „Robot4Web“ die Steuerung eines Roboterarms über FPGA als Ziel hat, berühren Entwicklung und Implementierung mehrere Fachgebiete. Besonders die Entwicklung des Roboterarms erforderte viel Zeit, um dessen mechanischen Eigenschaften zu planen und die passenden Baumaterialien zu finden. So wurden beim Bau des Roboters Alu- und Kupferröhre, Holzteile, Alu-Profile, Epoxidharz, Schaum- und Kunststoff verwendet.

Für das als Schutz für FPGAs und als Signalverstärker gedachte Board wurde ebenfalls aus einzelnen Bauelementen gebaut. Bei der Erstellung des Signalboards wurden Optokoppler, PNP-Transistoren, Widerstände, Kondensatoren, Steckleisten, Schalter und weitere Bauelemente verwendet, die anschließend auf ein Board verlötet wurden.

Der Aufbau dieser Arbeit ist wie folgt gegliedert:

Kapitel 2 behandelt die Grundlagen von VHDL und FPGA, und stellt ein Zusammenhang zwischen Software- und Hardwareimplementierung.

Im Kapitel 3 werden die entwickelten VHDL - Modelle präsentiert. Anschließend werden die Simulation und die Synthese der vorgestellten Modelle besprochen, dabei wird der Ablauf dieser Vorgänge erläutert und die verwendete Software beschrieben.

Im Kapitel 4 wird die Webschnittstelle dargestellt. Diese lässt sich in folgenden Bereichen unterordnen: Anwenderverwaltung, Simulation und Synthese, Streaming und Webschnittstelle für die Robotersteuerung.

Kapitel 5 behandelt den Entwurf, die technische Spezifikation und die Steuerung des Roboterarms. Dabei werden die im Roboterbau eingesetzten Motorentypen besprochen, und miteinander verglichen. Anschließend werden die verschiedenen Robotersteuerungen gegenübergestellt und behandelt.

Im Kapitel 6 wird nach einer kurzen Einführung in Reflexion- und Überspannschutz, das Signalbord vorgestellt. Anschließend wird das übertragene Steuersignal analysiert und die Messergebnisse präsentiert.

Kapitel 7 ist dem Job Scheduler gewidmet. Hier wird das Programm, das für Robot4Web konzipiert und realisiert wurde, vorgestellt. Dabei werden die verwendete Kommunikationstechnik und die Einsatzmöglichkeiten ausführlich beschrieben.

Kapitel 8 beschreibt die UART-Kommunikation. Speziell wird die Realisierung auf ein FPGA näher behandelt.

Eine Zusammenfassung der Arbeit wird im Kapitel 9 gegeben, ein Ausblick auf weitere Forschungsaspekte und zukünftige Anwendungsgebiete sind im Unterkapitel „Ausblick“ besprochen.

Im Anhang (Kapitel 10) sind dokumentierte Abbildungen und entwickelte Programmcodes untergebracht. Auf diese Elemente wird in den vorangegangenen Kapiteln Bezug genommen.

Kapitel 2 **Von VHDL zum FPGA**

In diesem Kapitel werden im Abschnitt 2.1 die Grundlagen und die verwendete Technologien der FPGA Boards beschrieben, auf denen Robot4Web aufbaut. Im Abschnitt 2.2 wird dieEntstehungsgeschichte von VHDL kurz zusammengefasst dargestellt, und letztendlich wird im Abschnitt 2.3 die Implementierung von VHDL Code im FPGA beschrieben.

2.1 **FPGA Grundlagen**

Der Begriff FPGA wird in dieser Diplomarbeit noch häufig verwendet, und bedarf eine kurze Zusammenfassung.

Die englische Abkürzung FPGA steht für Field Programmable Gateway Array und ist ein Programmierbarer Integrierter Schaltkreis.

Die Industrie hat durch ihre Entwicklung immer kompliziertere Steuerungen und Regelungseinheiten benötigt. An dieser Stelle hatte Ron Cline die Idee, einen beliebig konfigurierbaren Baustein zu realisieren, der dann von Ingenieuren vielseitig eingesetzt werden kann. So sind die SPLA, Simple Programmable Arrays entstanden. Mit den ersten programmierbaren Schaltkreisen hat man nur logische Schaltungen realisieren können.

Die erste FPGA-Architektur ist 1985 vom Herrn Freeman - ein Xilinx Mitbegründer - entwickelt worden. FPGA‘s unterscheidet sich von SPLA dadurch, dass anstelle einfacher logischer Gatter logische Blöcke (CLB) verwendet werden. Diese logischen Einheiten werden bei der Realisierung von komplexen logischen Funktionen eingesetzt. Eine wichtige Komponente der logischen Blöcke ist die Look-Up-Table (LUT). Abhängig von der Anzahl der LUT Eingänge, kann man alle binäre Funktionen mit n Variablen realisieren, wobei n die Anzahl der LUT-Eingänge ist. Die Abbildung 2 zeigt das vereinfachte FPGA Modell mit den konfigurierbaren logischen Blöcken. Die I/O Blöcke ermöglichen die Kommunikation zwischen den im FPGA-Schaltkreis angelegten dichten Verbindungskanälen und der Peripherie. Der Aufbau der logischen Einheiten in der FPGA erlaubt die Implementierung von sequenziellen und kombinatorischen Automaten.

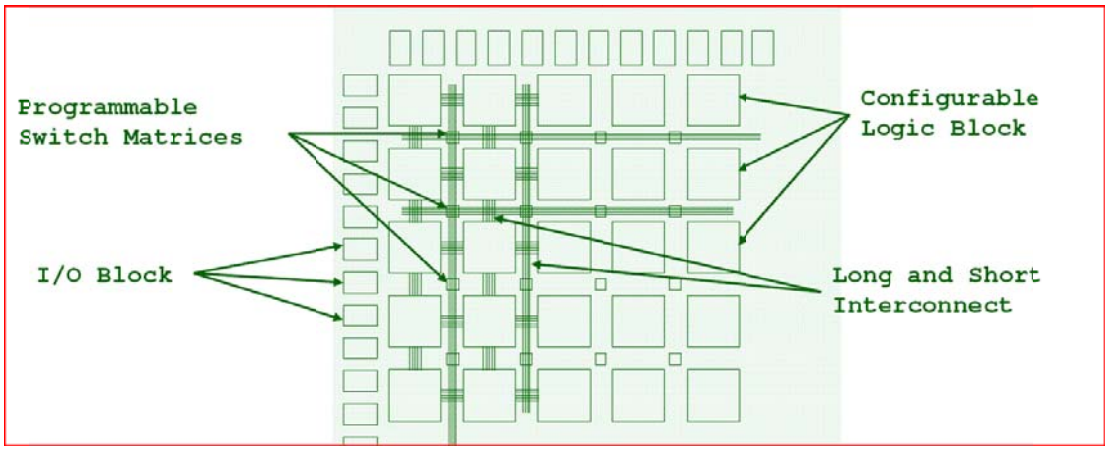

Abbildung 2: Vereinfachtes FPGA Modell

2.2 VHDL Grundlagen

Zielsetzung der VHDL Beschreibungssprache ist die Beschreibung von Verhalten und Funktion digitaler Schaltkreise. Weiterhin sollte ein vom Hersteller und Technologie unabhängiger, weltweiter Standard zur Beschreibung digitaler Hardware geschaffen werden. Mit Hilfe dieser Technologie ist das Planen, Simulieren und Synthetisieren von digitalen Schaltkreisen möglich geworden. Obwohl die VHDL Sprache sehr umfangreich ist, ist es trotzdem nicht möglich, alle in VHDL geschriebenen Codestücke in ein beliebiges FPGA zu implementieren. Die Implementierung wird durch die Synthetisierungssoftware, durch FPGA Eigenschaften oder einfach durch die gewünschte Geschwindigkeit, mit der die Applikation laufen soll, begrenzt.

Bereits in den 70-er Jahren ist die Notwendigkeit von Simulation und Synthese von Schaltkreisen erkannt worden. In den 80-er Jahren erscheinen die ersten VLSI (Very Large Scale Integration) integrierte Schaltkreise. Die integrierten Schaltungen beinhalten immer mehr Komponente (ROM, Register, etc.), Planung und Entwurf von neuen Schaltungen kommt mit der klassischen Methode nicht mehr aus. Alternative Wege werden gesucht.

VHDL erscheint im Jahre 1981, wird aber erst 1987 zum ersten Mal genormt und wird im weiteren etwa alle fünf Jahre überarbeitet. Beispielsweise ist 1993 der IEEE 1076 Standard überarbeitet und neu genormt worden.

Neben VHDL existieren noch andere Hardwarebeschreibungssprachen wie Verilog oder SystemC. Wegen der einfachen und leicht verständlichen Syntax von VHDL und wegen seiner weitreichenden Möglichkeiten, ist VHDL besonders im didaktischen Bereich sehr verbreitet. Eine Erweiterung von VHDL auf analoge Systeme ist mit VHDL-AMS definiert worden (VHDL analog / mixed signal).

2.3 Implementierung von VHDL-Code im FPGA

Der Planungsablauf von dem VHDL-Code bis zu einem in FPGA implementierbaren Schaltkreis beinhaltet drei Schritte: Spezifikation, Simulation und Synthese. Bei der Realisierung und in der Praxis werden diese Stritte noch weiter unterteilt und können einmal, oder auch zyklisch ausgeführt werden.

Spezifikation und System Design sind die ersten Schritte im VHDL Design Flow. Welche Aufgaben der geplante Schaltkreis zu lösen hat, wird gleich während der ersten Schritte bestimmt. Am Ende dieses Schrittes erhält man einen Schaltkreisentwurf, den man entweder aus logischen Bauelementen, ROMs und Registern realisieren kann, oder einfach mit Hilfe von VHDL beschreibt. Wenn man den Schaltkreis durch VHDL beschreibt, dann erhält man den zu implementierenden Schaltkreis nach der Hardwaresynthese. Der generelle Aufbau einer VHDL-Spezifikation beinhaltet die Schnittstellen-Beschreibung, Architektur eines Bausteins und Konfiguration.

Die Simulation dient grundsätzlich zur Verifizierung des funktionellen Verhaltens eines Designs. Es ist wichtig zu wissen, dass der einwandfrei simulierbare VHDL - Code nicht unbedingt in einem bestimmten FPGA implementierbar sein muss. Für die Simulation und Verifizierung von VHDL Code werden Softwaresimulatoren wie beispielsweise ModelSim von Mentor Graphics eingesetzt.

Zielsetzung der Hardwaresynthese ist die möglichst schnelle und kostengünstige Synthese elektronischer Hardware in einen FPGA zu realisieren. Die Hardwaresynthese wird von speziellen Synthesewerkzeugen erledigt, die meistens herstellerspezifisch sind. In diesem Schritt wird ein mit VHDL modelliertes Modell mit Hilfe einer Herstellerbibliothek in die FPGA befindlichen logischen Einheiten implementiert. Während der Hardwaresynthese werden die logischen Gatter zusammengeschaltet, die I/O Ports bestimmt und zu den Eingängen und Ausgängen des Schaltkreises geordnet. Letztendlich wird der so erstellte Code in den FPGA geladen. Grundsätzlich sind diese Schritte nicht mehr Plattformunabhängig, viele Faktoren können die Implementierung beeinträchtigen, wie die Anzahl der I/O-Böcke oder die verwendete Herstellungstechnologie. Die meisten plattformabhängigen Einstellungen werden von den verwendeten Syntheseprogrammen, basierend auf die Herstellerbibliothek, erledigt.

Für die erfolgreiche Realisierung von implementierbaren VHDL - Modellen ist die Beachtung des inneren Aufbaus der zugrunde liegenden FPGA absolut notwendig.

Kapitel 3 **Die Steuerungshardware im FPGA**

Für diese Diplomarbeit wurden zwei VHDL - Modelle entwickelt, um die Robotersteuerung zu demonstrieren, und als Plattform für zukünftige Applikationen zu dienen. In diesem Kapitel werden für Robotersteuerung entwickelte VHDL - Modelle vorgestellt und erläutert. Anschließend wird die Simulation und Synthese der vorgestellten Modelle und die verwendeten Entwicklungswerkzeuge beschrieben. Zum Schluss werden die Automatisierung und die Anbindung dieser Vorgänge über die Webinterface beschrieben.

3.1 **FPGA Plattformen**

Als Zielplattform für die Realisierung von digitalen Schaltungen wurde die Xilinx FPGA-Familie gewählt. Alle für dieses Projekt erstellten VHDL-Modelle sind auf Xilinx Virtex II-PRO und Xilinx Spartan 3A FPGAs getestet und implementiert worden.

Als Plattform für die verwendeten FPGAs dienen von unterschiedlichen Herstellern entwickelte Entwicklungsboards. Auf den Entwicklungsboards sind I/O-Anschlüsse, Konfigurationsspeicher, Erweiterungssockel, RAMs, AD/DA Wandler und weitere Erweiterungen angebracht. Die angebrachten Erweiterungen dienen der Speicherung von Konfigurationen, der Umwandlung physikalischer Größen, der Taktgenerierung oder sind mit den entsprechenden Kontrollern ausgestattete Schnittstellen.

Für den Xilinx Virtex-II Pro FPGA 2VP7-FG456 dient ein von der Firma Memec hergestellte Entwicklungsboard als Plattform. Der Spartan 3A FPGA ist auf einem einfacheren Entwicklungsboard von Digilent angebracht. Beide Plattformen besitzen für das Projekt Robot4Web wichtige I/O-Anschlüsse, unter anderem die seriellen Schnittstellen. Die verwendeten seriellen Schnittstellen und die Implementierung von synthetisierten UARTs wird in einem späteren Kapitel noch ausführlich beschrieben.

Zur Synthese und zum Konfigurieren beider FPGAs wird das windowsbasierte Project Navigator ISE 9.2 von Xilinx verwendet. Das Hochladen von Bitströmen geschieht über Programmierkabeln. Um die Virtex II-Pro FPGA zu konfigurieren, wird das Xilinx

Programmier- und Debuginterface Parallel Cable IV über die JTAG-Schnittstelle verwendet. Die Konfiguration von Spartan 3A FPGA gestaltet sich einfacher, da hier ein handelsübliches USB-Kabel verwendet wird.

Mit Hilfe von Jumpern, die sich auf den Entwicklungsboards befinden, können diese in verschiedene Programmiermodus versetzt werden. Es gibt insgesamt vier Programmiermodi: Boundary Scan Mode, Master-Serial Mode, Slave-Serial Mode und Select Map Mode, die abhängig vom Hardware Support der Boards unterschiedliche Konfigurationsgeschwindigkeiten aufweisen. Während der Entwicklung wird der Boundary Scan Modus bevorzugt.

3.2 Strukturelles Design

Um die Steuerung des Roboterarms über die bereitgestellte Webinterface zu ermöglichen, wurden zwei VHDL Modelle für die implizite und explizite Steuerung erstellt. Beide VHDL Modelle verwenden die gleiche Top-Level-Struktur, mit dem Unterschied, dass bei der expliziten Steuerung ein zusätzliches Signal für die UART Übertragung („rx“) gebraucht wird.

3.2.1 Einschränkungen im Modell

Da die Zielsetzung ein System ist, das von den Anwendern programmierbar ist, wurde das VHDL Modell so konzipiert, dass nur ein Teil der Module von den Anwendern veränderbar ist. Diese Einschränkung ist notwendig, da durch fehlerhafte Programmierung oder falsche Konfiguration Schäden entstehen können in der Anlage selbst, oder in der Umgebung.

Aus oben erwähnten Sicherheitsgründen, um die Systemintegrität zu bewahren, wird es den Anwendern nicht erlaubt, die Grundeinstellungen für die FPGA-Entwicklungsboards zu verändern. Diese Grundeinstellungen betreffen die Pin-Belegungen, und Takt- und Reset-Einstellungen des Entwicklungsboards.

Durch die Zuweisung von falschen Pin-Belegungen kann man den verwendeten FPGA stark beschädigen, oder sogar unbrauchbar machen. Die geringfügige Veränderung der Grundfrequenz in der Konfiguration hat eine große Auswirkung auf die Stellgeschwindigkeit und auf die Positionierung der Servomotoren. Eine gröbere Veränderung der Grundfrequenz von dem bestehenden Modell bewirkt, dass die generierten Steuersignale nicht mehr für die Servosteuerung geeignet sind. Ein Beispiel: bei Verwendung von 25MHz anstelle von 50MHz als Grundfrequenz in der FPGA werden die Pulsweitenmodulierte Steuersignale 2 bis 3 ms Lang sein. Die Servomotoren können aber Steuersignale nur im Bereich von 1ms bis 2ms Länge interpretieren.

Eine weitere Auswirkung der veränderten Grundfrequenz ist, dass die übertragene UART Signale von dem Baudgenerator erstellten Abtastsignale nicht richtig abgetastet werden. Die so entstandenen Steuersignale erzeugen dann unvorhersehbare Bewegungen des Roboterarms, da die gestellten Anforderungen nicht mehr erfüllt werden.

Das Top-Level Modell des, für die Robotersteuerung erstellten VHDL Modells, bildet der Plattform für die zukünftige Applikationen. Wegen der getroffenen Einschränkungen ist das Top-Level Modell von den Anwendern nicht veränderbar. Durch den modularen Aufbau von logischen Schaltungen in VHDL ist es leicht, die Schaltung so anzuordnen, dass die als empfindlich angesehene Komponente in einem Modul zusammengefasst wird. Um den Anwendern die Möglichkeit zu bieten, ihre logischen Schaltungen zu erstellen und in diese Umgebung zu integrieren, wurde das Modul „main_modul“ vorgesehen.

3.2.2 Entity

Auf der oberen Ebene (Top Level) werden Eingang- und Ausgangssignale des gesamten zu simulierenden Systems beschrieben. Auf der nachfolgenden Abbildung ist die äußere Sichtweise, eine Abstraktion von der eigentlichen physikalischen Schaltung, der Top-Level Modell von der expliziten und impliziten Steuerung dargestellt.

Die verwendeten TOP-Level Modelle haben zwei bzw. drei Signaleingänge und insgesamt sieben Signalausgänge. Diese Signale werden in der „TOP.UCF“ –Datei spezifiziert und den entsprechenden FPGA Pins zugewiesen.

Das Signal „clk_i“ ist die Takt Signalquelle mit 50MHz Taktfrequenz, und hat eine Periodendauer von 20ns. Per Definition ist das Taktsignal ein Rechtecksignal mit 50% HIGH und 50% LOW Anteil. Das „clk_i“ Signal wird zu jedem einzelnen Teilmodul zugeführt und ermöglicht, dass die einzelnen Module synchron arbeiten bzw. Schaltvorgänge und Zustandswechsel ausgelöst werden.

Das Signal „reset_n_i“ ist ebenfalls ein Eingangssignal. Er wird verwendet, um die gesamte Hardware-Logik asynchron in einen definierten Ausgangszustand zurückzusetzen. Entsprechend der Konfiguration, kann der Reset Signal HIGH oder LOW Initialwerte erhalten. In unserer „TOP.UCF“ –Datei wird „reset_n_i“ mit dem Zustand HIGH initialisiert.

Das „rx“ Signal wird für die serielle Datenübertragung verwendet. Die gesendeten Signale werden mit 9600 Baud übertragen. Bei jeder erfolgreichen Signalübertragung werden 8 Bits seriell übertragen. Da ein ganzer Befehlssatz aus 24 Bits besteht, werden die generierten Steuerbefehle in drei Schritte übertragen.

Aus der Abbildung 3 ist zu entnehmen, dass das verwendete Modell sieben Ausgangssignale hat. Durch die Ausgangssignale „engineX_o“ - wobei X die Nummer des korrespondierenden Robotergelenks ist - werden die Servomotoren gesteuert. Das Steuersignal ist ein Rechtecksignal, mit einer Periodendauer von 20 ms und eine Impulsdauer von 1 bis 2 ms Länge. Diese Signale werden von dem Signalgenerator-Modul erzeugt. Bei Signalgenerator-Modul handelt es sich um einen Modul, der wegen den getroffenen Einschränkungen durch die Anwender nicht veränderbar ist.

Abbildung 3: Top-Level Modell. a.) explizite Steuerung. b.) implizite Steuerung

Die äußere Sichtweise spiegelt sich in der Entity Deklaration wieder. Im nachfolgenden Codeausschnitt ist der Top Entity beschrieben. Alle hier beschriebenen Signalleitungen werden in der „TOP.UCF“ –Datei zu den I/O-Schnittstellen dem zu Grunde liegenden FPGA zugewiesen.

Neben jedem Ausgangssignal sind die anzusteuernden Robotergelenke als Kommentar dazugefügt.

```
entity top is
port(
      -- Eingangssignale
      clk_i     : in std_logic; -- 50MHz Taktsignal
      rx        : in std_logic; -- UART
      reset_n_i : in std_logic; -- asynchron Reset
      -- Ausgangssignale
      engine0_o : out std_logic; --Steuersignal, Sockel
      engine1_o : out std_logic; --Steuersignal, Neigung
      engine2_o : out std_logic; --Steuersignal, Oberarm
      engine3_o : out std_logic; --Steuersignal, Rotation A
      engine4_o : out std_logic; --Steuersignal, Handgelenk
      engine5_o : out std_logic; --Steuersignal, Rotation B
      engine6_o : out std_logic  --Steuersignal, Greifer

      );
end top;
```

Listing 1: Top Entity des R4W Modells.

3.2.3 Architektur

Die Beschreibung der Funktionalität eines Moduls wird in der Architektur(Architecture Body) zusammengefasst. Dies kann auf vielfältige Weise geschehen. Die VHDL Beschreibungssprache erlaubt die Programmierung einer Verhaltensbeschreibung, die Verwendung von Strukturbeschreibungen oder beides miteinander kombiniert.

Auf Abbildung 4 ist der strukturelle Aufbau des Top-Level Modells dargestellt. Aus dem Modell ist es einfach zu entnehmen, dass die Architektur des Top-Level Modells auf zwei grundlegende Modelle aufgebaut ist. Aus den verwendeten Modulen werden Instanzen gebildet, die dann durch deklarierte Signale miteinander verdrahtet werden. Auf dieser Ebene ist keine weitere Verhaltensbeschreibung notwendig. Die weiteren Hierarchieebenen bestehen jedoch vollständig oder nur teilweise aus Verhaltensbeschreibung.

Unter Berücksichtigung der getroffenen Einschränkungen wurde das gesamte VHDL Entwurf so konzipiert, dass alle Komponente, die von Anwendern erstellt wurden, in einem Modul zusammengefasst werden können. Dadurch wird erstens die Sicherheit erhöht, und zweitens die Übersichtlichkeit und Handhabung wesentlich erleichtert.

Um das Anwendermodul von dem Rest der Top-Level Architektur Komponenten leichter zu trennen, wurde für die Top-Level Architektur die Strukturbeschreibung gewählt.

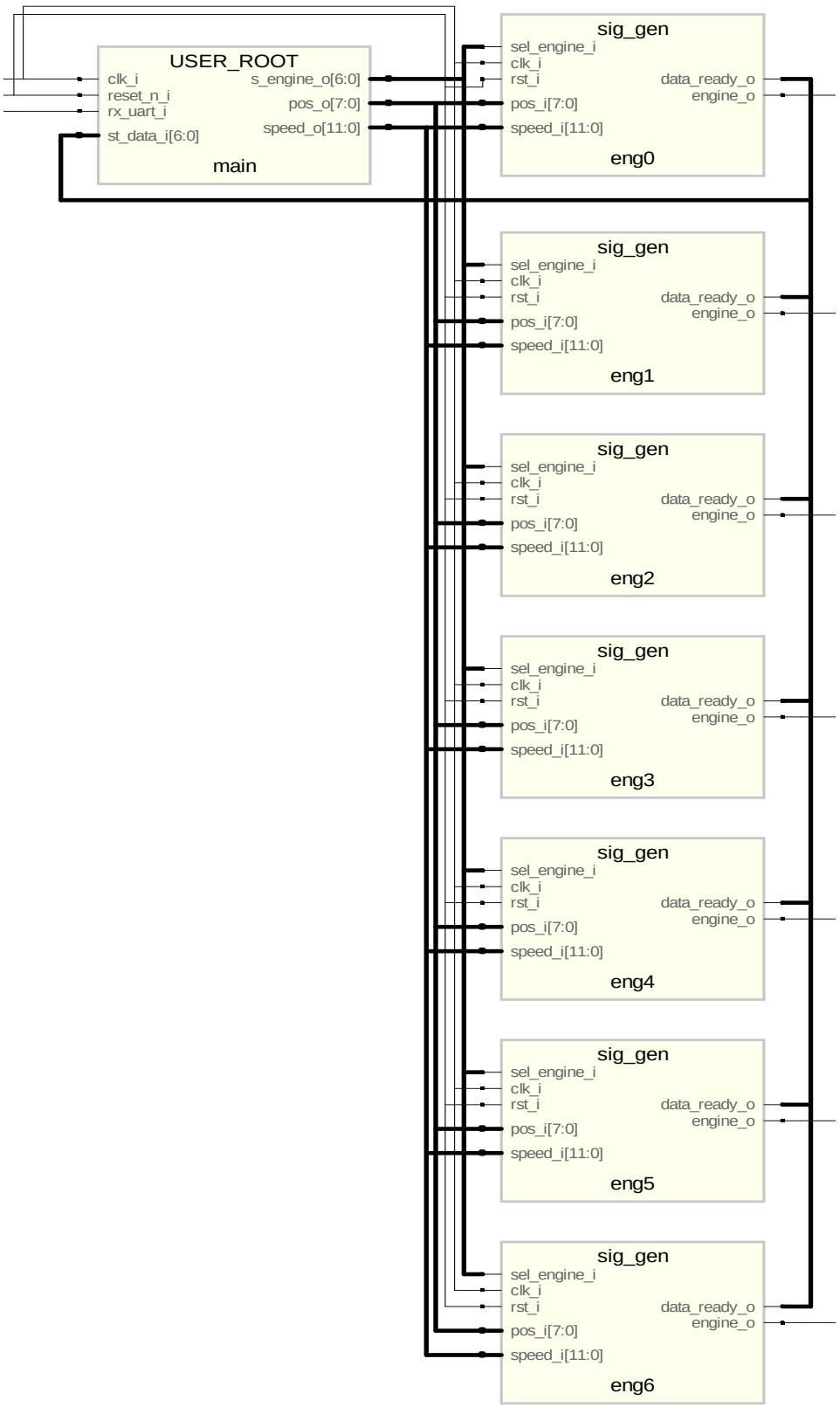

Abbildung 4: Innere Sicht des Top-Level Modells.

3.2.4 Das „User_Root“ Modell

Die VHDL-Entity des Modells ist vorgegeben, deswegen müssen alle Eingangs- und Ausgangssignale im Anwendermodul definiert sein. Die Architektur vom „User_Root“Modell kann vom Anwender selbst gestaltet werden, dabei können weitere Module oder das Verhalten beschrieben werden.

In die Entity des „User_Root“ Modells sind die Eingangssignale „clk_i“ und „reset_n_i“ unverändert von dem Top-Level Modell übernommen. Diese Signale steuern die globale Taktung und das asynchrone Zurücksetzen. Das „rx_uart_i“ Signal wird für die serielle Datenübertragung verwendet. Es besteht nur aus einer Signalleitung, und wird ebenfalls von der Top-Level Entity unverändert übernommen, ohne Verwendung lokaler Signale.

Von dem Signalgenerator Modul wird der Erhalt neuer Befehle mit einem Signal quittiert. Von allen sieben Signalgeneratoren werden die Signalleitungen in einem Signalbus zusammengefasst, und an das User_Root Modul weitergeleitet. Die Busbreite des „st_data_i“ Signalbuses entspricht der Anzahl der Signalgeneratoren, und somit der Anzahl der Robotergelenke. Um die Signale von dem Signalgenerator zu dem User_Root Modul zu verdrahten, werden in der Top-Level Architektur lokale Signalleitungen verwendet. Der Anwender hat in seinem Programm sicherzustellen, dass Signale nur dann übertragen werden dürfen, wenn von allen Instanzen des Signalgenerators die Bereitschaft des Datenempfangs signalisiert wurde.

Ausgangsseitig wird der „User_Root“ Modul mit dem Signalgenerator direkt verdrahtet. Über die Signalbusse „s_engine_o“, „pos_o“ und „speed_o“ werden die Signalgeneratoren ausgewählt, die neuen Positionen und die gewünschte relative Geschwindigkeit angegeben.

Das Signalbus „s_engine_o“ besteht aus sieben Signalleitungen, für jede Instanz des Signalgenerators eine Signalleitung. Mehrere Instanzen des Signalgenerators können durch gleichzeitige Setzung von Bits auf dem „s_engine_o“ Bus ausgewählt werden, und so die berechnete Kommandos simultan zu übertragen. Dieses Verhalten ist sehr nützlich, wenn der Roboterarm in einen sicheren Zustand gebracht werden soll, oder bei Aktivierung des Reset-Knopfes der Ausgangszustand angefahren werden soll.

Die Positionierung des Robotergelenks wird mit einer Auflösung von 8 Bit angesteuert. Das heißt, dass ca. 200 Grad (die maximale Verstellwinkel des Servomotors) in 256 Schritte aufgeteilt wird. Ein Schritt entspricht weniger als ein Grad Verstellwinkel und nähert sich der Grenze, die die verwendeten analogen Servomotoren erreichen können. Die Wert zur Berechnung der relativen Geschwindigkeit wird über das Signalbus „speed_o“, mit einer Busbreite von 12 Bit an den Signalgenerator gesendet.

Anwendungsbeispiel

Auf Abbildung 5 ist der strukturelle Aufbau des „User-Root“ Moduls dargestellt, wie es in der expliziten Steuerung eingesetzt wird. Da bei der seriellen Übertragung eine genaue Ab-

tastung der empfangenen Signale zusätzlich noch zu realisieren ist, wird ein Baudgenerator erstellt. Der Baudgenerator erstellt ein Taktsignal, das mit dem vereinbarten Übertragungstakt übereinstimmt. Bei einer Übertragungsrate von 9600 Baud ist die erstellte Abtastfrequenz 2604-mal kleiner, als die globale Taktfrequenz von 50MHz.

Die seriell gesendeten Signale sind weiterhin auf ihre Integrität zu prüfen. Die Überprüfung und die Zusammenfassung der Signalbits werden im „serin“ Modul behandelt, bevor die als parallele Signale an die „cntrl“ Module gesendet werden.

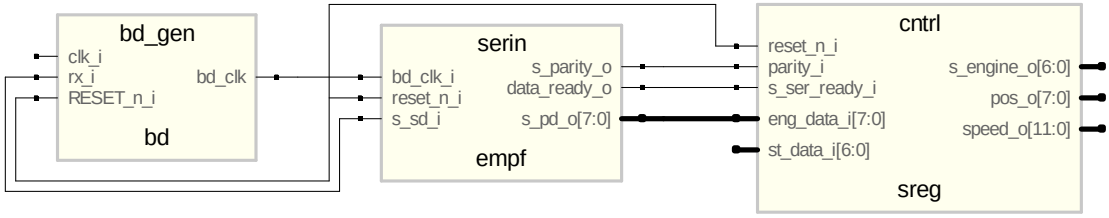

Abbildung 5: Innere Struktur des „User_Root“ Moduls für die serielle Steuerung.

Wegen der getroffenen Einschränkungen muss die Möglichkeit gegeben sein, das „User_Root“ Modul separat zu den vorhandenen Modulen zu laden. Dazu ist es notwendig, die Simulation- und Synthese-Ablauf so zu gestalten, dass dieses Modul zu den bestehenden Grundkomponenten dazugeladen und integriert wird. Das Laden von Anwendermodule wird vom Webinterface und vom Job Scheduler im „Hintergrund“ erledigt.

3.2.5 Signalgenerator Modell

Der Signalgenerator ist eines der grundlegenden Module, und bildet den wichtigsten Teil des VHDL Modells. Der Anwender hat keinen Zugriff auf dieses Modul, wodurch dies nicht verändert werden kann. Der Signalgenerator hat die Aufgabe, ein kontinuierliches Steuersignal, basierend auf den Eingangsparametern zu erstellen. Das generierte Steuersignal ist ein pulsweitenmoduliertes Signal mit einer Wiederholfrequenz von 50 Hz.

Das „User_Root“ Modell bildet mit jeder Instanz des Signalgenerators und mit den dazugehörigen Signalleitungen bzw. Signalbusse einen geschlossenen Regelkreis. Wodurch insgesamt sieben in sich geschlossene Regelkreise entstehen.

Die Ausgangsgrößen des Regelkreises sind die sieben Steuersignale, die Rückführung wird durch die Signalleitungen „st_data_i“ gebildet, der Regler selbst ist das „User_Root“ Modul, wobei die Signalgeneratorinstanzen als Prozesse angesehen werden. Die Führungsgrößen im Regelkreis sind in der Top-Level Entity definierte Eingangssignale.

Der Signalgenerator wird in der Top_Level Architektur instanziiert und verdrahtet. Vom „User_Root“ Modell wird über die Signalleitungen „s_engine_o“ jeder Instanz signalisiert, dass neue Eingangsdaten angelegt wurden. Die angelegten Signale müssen mindestens eine

Periode lang(20 ns) unverändert an den Signaleingängen des Signalgenerators liegen, um diese von jeder Signalgeneratorinstanz richtig interpretiert werden zu können.

Weitere Eingangssignalen sind „pos_i“ und „speed_i“. Beide Signale entsprechen den Ausgangssignalen vom „User_Root“ Modul, wo diese ausführlich auch beschrieben wurden. Durch die Möglichkeit, dass jeder Signalgeneratorinstanz Eingangsdaten zugewiesen werden können, und sie gezielt aktiviert werden können, erreicht man, dass jede Instanz voneinander unabhängig, verschiedene Steuersignale generieren kann. Die generierten Steuersignale werden über die „engine_o“ Signalleitungen ausgegeben, dann verstärkt an die Steuerleitung der Servomotoren angelegt.

Steuersignale erstellen

Im Gegensatz zu Schrittmotoren, sind die Servomotoren nicht darauf ausgelegt, mit unterschiedlicher Stellgeschwindigkeit angesteuert werden können. Es ist gelungen, das Zeitverhalten dieser Elektromotoren mit den generierten Steuersignalen so zu manipulieren, dass die Stellgeschwindigkeit und sogar die Beschleunigung der Servomotoren mit 8 Bit Auflösung zu regeln ist.

Da nur die gewünschte Sollposition angegeben werden kann, ist die Ansteuerung von Servomotoren viel einfacher als bei der Schrittmotoren. Aus der erhaltenen Sollposition und aus der aktuellen Position berechnen die Servo-Regler die Anfahrtsbeschleunigung und die Bremsphase. Je größer die Entfernung zwischen aktueller Position und Zielposition, desto höher ist die erreichte Geschwindigkeit nach der Beschleunigungsphase. Wenn man die Endposition nicht in einem Schritt sondern in vielen kleineren Schritten zurücklegt, dann sind die erreichten Geschwindigkeiten nach jeder Beschleunigung weitaus kleiner als bei einem Schritt.

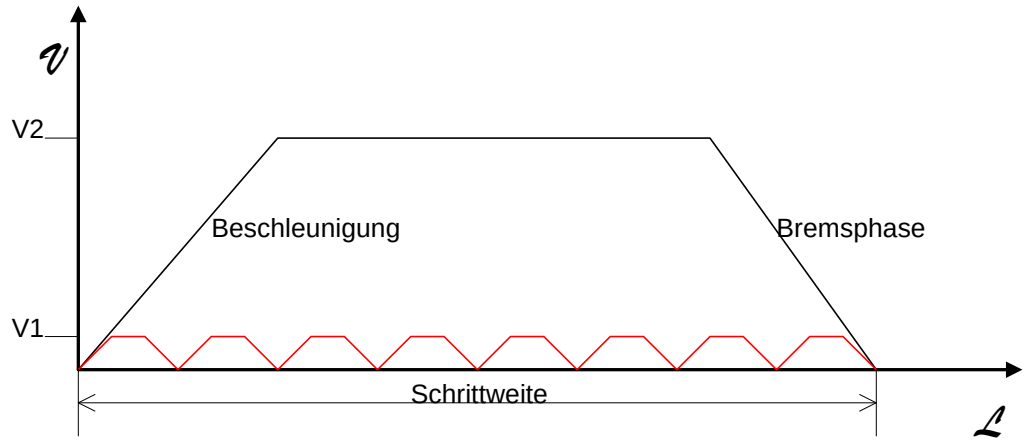

Abbildung 6: Servomotor Anfahrtsgeschwindigkeiten in Abhängigkeit der gewählten Schrittweiten.

Auf Abbildung 6 sind die erreichten Geschwindigkeiten in der Abhängigkeit der gewählten Schrittweiten dargestellt. Es ist leicht zu nachvollziehen, dass die gewählte Schrittweite die Anfahrtsgeschwindigkeit beeinflusst.

Auf Abbildung 6 der rot dargestellte Bewegungsablauf führt zu einer ruckartigen Bewegung des Roboterarms, da jeder Schritt auf der Darstellung erst nachdem der Servoarm zum Stillstand gekommen ist, gestartet wurde.

Um die Beschleunigung und die Bremsphase aus dem Bewegungsablauf zu eliminieren, müssen die Steuersignale so verschoben sein, dass bei gleich bleibender Schrittweite die Geschwindigkeiten konstant bleiben. Das wird dadurch erreicht, dass die Position der nachfolgenden Schritte bereits vor dem Eintritt in die Bremsphase versendet wird. Auf Abbildung 7 wird durch dicke Linie die tatsächliche Bewegung dargestellt. Die dünnen Linien zeigen den theoretischen Anfang jedes Schrittes. Tatsächlich sind die dünnen Geraden fast vertikal, da der Servomotor die maximale Anfahrtsgeschwindigkeit bei der gegebenen Schrittweite bereits erreicht hat.

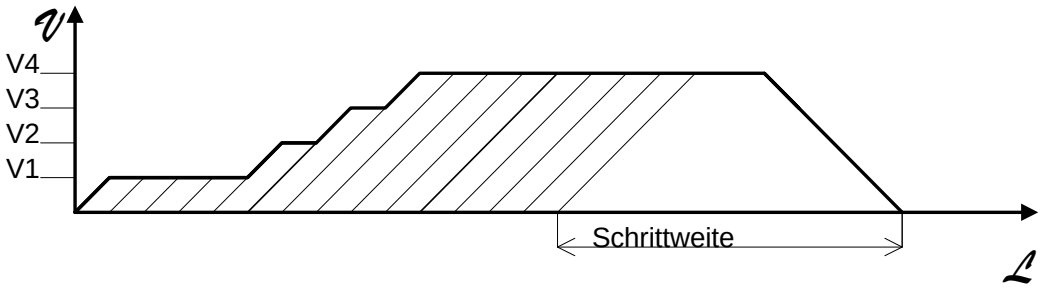

Abbildung 7: Sukzessive Erhöhung der Anfangsgeschwindigkeit von Servomotoren

Die Anfahrtsgeschwindigkeit lässt sich zweierlei regeln. Durch die Erhöhung der Schrittweite und durch die Änderung von den Abständen, in dem die neue Position gesendet wird, verändert sich auch die Geschwindigkeit, mit der von der Startposition in die Zielposition gelangt wird.

Die Berechnung der Geschwindigkeiten und Schrittweiten ist die Aufgabe vom Signalgenerator Modul. Um die aktuelle Position des Robotergelenks im VHDL Modul zu kennen, wird dieser Wert im Signalgenerator ständig aus den vorangegangenen Zielpositionen und aus internen Zuständen immer neu berechnet.

In der nachfolgenden Listing wird die Berechnung der Pulslänge in dem Signalgenerator zugrunde liegender VHDL –Code angegeben. Berechnet wird sie nach folgendem Modell:

Pulslänge = 1 + X / 256, wobei X die neue Position darstellt.

Weiterhin gilt: 1 ms < berechnete Pulslänge < 2 ms.

```
if v_COUNTER < 1000000  then
    PWM_LEN    := PWM_LEN +1;
    s_POSITION <= '1';
    if  PWM_LEN     >  v_NEW_POS then
        s_POSITION  <= '0';
    end if;
else
    v_COUNTER   := 0;
    PWM_LEN     := 0;
end if;
v_COUNTER := v_COUNTER +1;
```

Listing 2: Erzeugen von PWM Signale

Die oben angegebene Listing wird in jedem Takt einmal ausgeführt. Da die Taktlänge 20ns beträgt, entsprechen 1.000.000 Takte genau der Wiederholfrequenz von 50 Hz. Nach jedem Takt wird die Pulsweite „PWM_LEN“ um 20ns erhöht, dabei werden die durchlaufenen Takte in v_Counter Variable gezählt. Nachdem die Pulsweite die neue Position charakterisierende Länge erreicht hat, wird das Steuersignal „s_POSITION“ auf null gesetzt. Das Steuersignal wird bei jedem Takt verändert, und direkt an der Servosteuerung gesendet.

Die Geschwindigkeitsregelung (mit Hilfe obiger Berechnung) wird so ausgeführt, dass der Variablen „v_NEW_POSITION“ nicht die endgültige, an den Signalgenerator übermittelte Position übergeben wird, sondern nur ein Zwischenwert. Dadurch werden nach jeder Periodendauer des Steuersignals, leicht veränderte Steuersignale gesendet. Der Zielposition wird also nach jeder Periode, schrittweise angenähert.

3.3 Simulation

Die Simulation von digitalen Schaltungen ist eine wichtige Arbeitsphase, die vor der Synthetisierung der entwickelten Schaltung mehrmals durchgelaufen wird. Die Simulatoren unterstützen die Entwickler nicht nur bei der Fehlersuche, sondern helfen auch beim Verstehen von Schaltungen und Signalabläufen. Es gibt mehrere Hersteller von Simulationssoftware, aber das meist etablierte Simulationsprogramm ist ModelSim von Mentor Graphics. In diesem Projekt wird ModelSim als Simulationsprogramm für die VHDL Module eingesetzt. Da die Simulation hauptsächlich automatisiert ablaufen soll, wird ModelSim in Batch Modus ausgeführt.

3.3.1 Benchmarks

Um die entworfene digitale Schaltung zu testen, werden Testbenches verwendet. Sinnvollerweise wird der Testbench in der Sprache erstellt, in der die Schaltung beschrieben wurde. Der Testbench hat keine Bedeutung bei der Synthese von digitalen Schaltungen, er

wird ausschließlich beim Testen eingesetzt. Ein weiteres Merkmal der Testbench ist, das die Entity Deklaration lehr bleibt.

Die wichtigste Aufgabe eines Testbenches ist, die gewünschten Taktsignale, Reset-Signale und falls notwendig Eingangsdaten für die Simulation bereitzustellen. Für die Robotersteuerung sind es die seriellen Daten, die der Testbench mit der angegebenen Übertragungsrate bereitstellt.

Für die Simulation der von den Anwendern erstellten Programme werden Testbenches bereitgestellt, die dann heruntergeladen und modifiziert mit den anderen VHDL Dateien hochgeladen werden können.

3.3.2 Simulationsablauf

Die Simulation der von den Anwendern erstellten VHDL Modellen wird in enger Zusammenarbeit einerseits von der Webschnittstelle, andererseits von der ModelSim Simulationsprogramm durchgeführt. Da die Webschnittstelle als Interface dient, wo man für die Simulation notwendige Parametern eingibt und die zurückgegebene Ergebnisse betrachtet, müssen alle andere Programme, die in der Simulationsausführung beteiligt sind, entweder im Hintergrund ausgeführt, oder als Batch-Programm auf dem Server gestartet werden.

Bevor die Simulation gestartet werden kann, werden die nachfolgenden Parameter über die Webschnittstelle eingegeben, und in der „SIM_ATTRIBS“ Datei am Webserver gespeichert:

- TESTBENCH Name des Testbenches, die bei der Simulation eingesetzt wird
- SIM_TIME Simulationsdauer
- SIM_START_T Ab welchem Zeitpunkt der Simulationsverlauf dargestellt wird
- SIM_END_ Ende vom dargestellten Simulationsverlauf
- USERNAME Anwendername (Loginname)

Der „SIM_ATTRIBS“ Datei wird in das „Makefile“ inkludiert, bestimmend die Simulationsablauf und die gelieferte Ergebnisse.

Simulationsumgebung

Um die Simulation erfolgreich ausführen zu können, wird die UNIX-Umgebung für Windows, das Programm CYGWIN installiert. Mit Hilfe von Cygwin ist es möglich, die von ModelSim bereitgestellten Programme über ein Makefile aufzurufen. Der Simulationsworkflow wird mit Hilfe von Batch-Dateien und Makefiles gesteuert und ausgeführt, wobei der Jobscheduler bei nicht ausreichenden Zugriffsberechtigungen unterstützend eingesetzt wird.

Prozesse unter lokalen Zugriffsberechtigungen ausführen

Da die Internetuser nur beschränkte Ausführungs- bzw. Zugriffsrechte haben, ist es nicht möglich, mit der vorhandenen Benutzerrechten von einer Webseite aus die direkte Workflow-

Steuerung abzuwickeln. An dieser Stelle kommt der Jobscheduler zum Zug. In den Makefile, wo Prozesse mit erweiterten Benutzerrechten gestartet werden müssen, werden Batch Dateien ausgeführt, die dann Konfigurationsdateien erzeugen. Basierend auf die so erstellten oder modifizierten Dateien, führt der Jobscheduler mit lokalen Anwenderrechten die Simulationsschritte aus. Um den Workflow-Ablauf nicht zu unterbrechen, werden Watch-Dog Signalen verwendet, die nach einer Zeitüberschreitung der Simulationsausführung die Abarbeitung von Makefile fortsetzt.

```
./sim_vsim.bat $(TESTBENCH) > /cygdrive/c/inetpub/wwwroot/Robot4Web/reports/start_vsim.sh
waitfor /T 60 contvsim
```

Listing 3: Verwendung von Watch-Dog Signale in ein Makefile

In der oben dargestellten Listing wird die Datei „start_vsim.sh“ erstellt, und anschließend die Simulation vom Jobscheduler gestartet. Bei Erfolg werden „waitfor /SI contvsim“ vom Jobscheduler gestarteten Batch-Dateien ausgeführt, signalisierend für die wartenden Prozesse die Weiterführung.

Sobald alle vom Jobscheduler aufgerufenen Prozesse beendet sind, wird die Workflow-Ausführung mit eingeschränktem Zugriff weiter ausgeführt. Unter Windows Vista verwendet Cygwin das Konto „NetworkService“, um Befehle mit eingeschränktem Zugriff auszuführen.

ModelSim Befehle

ModelSim stellt eine große Anzahl von Befehlen bereit, die in Batch Modus ausgeführt werden können, wodurch eine Automatisierung von der Simulation möglich gemacht wird. In diesem Projekt wurden die nachfolgenden ModelSim Befehle eingesetzt: vsim, vlib, vcom, und vmake.

Mit „vlib“ können ModelSim Bibliotheken erstellt und anschließend kompiliert werden, dann werden sie für die nachfolgende Simulation eingebunden. In unserem automatisierten Simulationsablauf wird „vsim“ verwendet, um das „work“ Bibliothek zu erstellen.

Der Befehl „vcom“ kompiliert die gefundenen VHDL Dateien in die spezifizierte Bibliothek, oder in die standardmäßig eingestellte „work“ Bibliothek.

Da bei der Kompilierung von VHDL Codes die Abhängigkeiten zwischen den Komponenten häufig nicht berücksichtigt werden können, wird der Befehl „vmake“ aufgerufen, um ein „makefile“ zu erzeugen, in dem die Komponenten in (meistens) richtiger Reihenfolge abgearbeitet werden.

Mit dem Befehl „vsim“ und der anschließender Benchmark-Name wird die Simulation gestartet. Die bei der Ausführung aufgetretenen Fehlermeldungen werden in die Transcript-Datei abgespeichert, und später im Simulationsbericht über die Webschnittstelle aufgerufen.

Neben den vorher erwähnten ModelSim Befehle wird noch das „simulate.do“ Datei mit „vsim -do simulate.do“ aufgerufen. In diesem Fall werden alle in „simulate.do“ Datei befindliche Befähle hintereinander ausgeführt[8].

3.3.3 Ergebnisse der Simulation

Die Ergebnisse der Simulation werden über einer Berichtseite für die Anwender zugänglich gemacht. Nur authentifizierte Anwender können Zugang zu eigene Testberichte erlangen. Nach jeder Simulation werden die vorangegangenen Ergebnisse überschrieben.

Die simulierten Signalverläufe für die festgelegten Zeitrahmen werden in ein JPG und ein PS Datei abgespeichert. Im Testbericht ist noch das von ModelSim erstellte „Transcript“, das Simulationsreport und die „vsim.wlf“ Datei aufgelistet.

3.4 Hardware Synthese

Ziel der Hardware Synthese ist eine möglichst schnelle und kostengünstige Synthese, um digitale Schaltungen in eine FPGA zu realisieren. In der Praxis verwendete Synthesewerkzeuge ermöglichen die Erkennung von Fehlern im Design, die durch die Simulatoren unentdeckt bleiben, weil der Code syntaktisch korrekt ist. Aus diesem Grund empfiehlt es sich die Verwendung von Synthesewerkzeugen, und zwar bereits in einer früheren Entwurfsphase[12].

Zur Synthese von VHDL Modellen in Virtex II-Pro und in Spartan 3A des FPGAs wird das Windows basierte Synthesewerkzeug Synplify Pro von der Firma Synplicity verwendet. Dazu wurde auf das Betriebssystem Windows Vista das Programm Synplify Pro mit der Einbindung des Lizenzservers des Instituts ICT der Technischen Universität Wien installiert.

3.4.1 Syntheseablauf

Die Synthese von digitalen Schaltungen ist die automatische Umsetzung einer Darstellung der Schaltung auf einer Abstraktionsebene, in eine Darstellung auf die nächst tiefere Abstraktionsebene. Die Umsetzung wird in mehreren Schritten durchgeführt, die in zwei Syntheseebenen mit unterschiedlichen Technologieabhängigkeiten zusammengefasst werden können. In der technologieunabhängige „High-Level Synthese" wird die algorithmische Ebene auf die RT-Ebene übersetzt(engl.: RTL Translation). Die nachfolgende Logikoptimierung versucht nun die im ersten Schritt gewonnene technologieunabhängige Realisierung so umzuformen, dass eine optimale Umsetzung in die Zieltechnologie im Rahmen der Technologieabbildung möglich ist. Integraler Bestandteil der beiden letzten Schritte ist die Timing-Analyse. Nach der Technologieabbildung liegt eine optimierte Netzliste mit Bauelementen der Zieltechnologie vor, die dann platziert und verdrahtet werden kann[12].

3.4.2 Eingabedaten

Für die Synthese, ähnlich wie bei der Simulation, werden die VHDL Modelle geladen und kompiliert. Da die Synthese nur teilweise technologieunabhängig durchgeführt wird, sind weitere Informationen über das Zielsystem und die verwendete Technologie an das Synthesewerkzeug zu übergeben. Diese Informationen werden zur Steuerung der RTL-Synthese eingesetzt, die die RT-Ebene auf die technologieabhängige Gatter-Ebene übersetzt.

Die Übergabe von VHDL Dateien und herstellerspezifische Informationen an das Synthesewerkzeug Synplify Pro geschieht mit Hilfe einer Projekt-Datei. Die Projektdatei wird vor dem Synthesebeginn erstellt und besteht aus einem generierten und aus einem statischen Teil. Der generierte Teil beinhaltet die Pfade und die Dateinamen aller VHDL Dateien, die während der Synthese verwendet werden. Der statische Teil von der Projekt-Datei beinhaltet die Implementierungsoptionen über Hersteller, Technologie, Geschwindigkeit und Package, sowie Angaben zur Steuerung der Syntheseablauf. Da der letzte Teil der Projekt-Datei den Projektablauf und die verwendete Technologie beschreibt, werden diese Informationen nur von dem Systembetreuer verwaltet.

3.4.3 Syntheseergebnisse

Das Ergebnis der Synthese ist eine Netzliste, die die Verbindung zwischen den FPGA Bausteinen, wie LUT, Flipflops, MUX, etc. beschreibt. Weil das Ergebnis der Synthese sich auf der Gatterebene befindet, können Signale und Ports auf der Gatterebene nur noch binäre Werte annehmen.

Die Netzliste, als Ergebnis von der Synplify durchgeführte Logiksynthese, wird in EDIF-Format(Electronic Design Interchange Format) abgespeichert, und in den weiteren Schritten von den in Xilinx ISE 9.2 enthaltenen Werkzeugen importiert. Der geloggte Syntheseablauf, und die von Synplify erstellten Berichte werden letztendlich auf die Berichtseite zusammengefasst, und den Erstellerbenutzer zugeordnet.

3.5 Bitstrom erstellen und hochladen

Zum Konfigurieren des FPGAs wird das Windows-basierte Entwicklungskit ISE 9.2 von Xilinx verwendet. Dazu wurde auf einem Windows-Betriebssystem die Entwicklungsumgebung installiert, und das vorhandene Workflow für die Simulation und Logiksynthese so erweitert, dass die Ergebnisse der vorangegangenen Schritte wiederverwendet werden können.

Bevor die FPGA Kofigurationsdatei(eng.: Bitstream) hochgeladen werden kann, werden von der Synthese erzeugten EDIF-Datei ausgehend noch weitere Zwischenschritte, wie das Übersetzen des Sourcecodes in eine NGD-Datenbank, Place and Route, usw. durchgeführt.

Zusätzlich sind noch das Programmierkabel, die FPGA Pin-Zuordnung und die Taktrate zu konfigurieren.

3.5.1 FPGA Pin-Zuweisung

Da die bisher entwickelte Modelle nur die FPGA interne Abläufe beschreiben, müssen die Schnittstellen definiert werden, wodurch der FPGA im Stande ist, die interne Schaltungen zu Takten, Signale zu empfangen und zu versenden. Die Definition dieser Schnittstelle ist vom Hersteller und von Modell abhängig, deswegen wird sie in einer vom Anwender konfigurierbare Datei zusammengefasst.

Eine UCF(User Constraints File) Datei ist eine vom Benutzer erstellte ASCII Datei, die Einschränkungen auf dem funktionellen Entwurf spezifiziert. Nachfolgend wird der TOP.UCF Dateiinhalt aufgelistet.

```
#------------------------------------------------
# Clock Period Constraints
#------------------------------------------------
NET "clk_i" PERIOD = 20 ns HIGH 50 %;
NET "clk_i" TNM_NET = "clk_i";
NET "clk_i"  LOC = "V12"  ;
NET "reset_n_i"  LOC = "V15" | PULLUP ;

NET "engine0_o"  LOC = "P17"  ;
NET "engine1_o"  LOC = "Y21"  ;
NET "engine2_o"  LOC = "K19"  ;
NET "engine3_o"  LOC = "U22"  ;
NET "engine4_o"  LOC = "R21"  ;
NET "engine5_o"  LOC = "V22"  ;
NET "engine6_o"  LOC = "U21"  ;

NET "rx"  LOC = "U9" | IOSTANDARD = LVTTL ;
```

Listing 4: Zuweisung von Virtex II-Pro I/O-Ports an das R4W Design und Taktbestimmung

In der oben dargestellten Datei wird die Zuordnung zwischen den Schnittstellen des VHDL-Modells (Ein- und Ausgänge der Top-Level-Entity) und den Schnittstellen des FPGA mit der Peripherie (Pins des FPGA) festgelegt. Die Zuweisung der Pins erfolgt mittels sogenannter „Pin Assignments“[15][17].

3.5.2 Von der Logiksynthese zum Bitstrom

Das Ergebnis der Logiksynthese wurde vom Synthesewerkzeug Synplicity erstellt; nun wird die Netzliste in die Xilinx ISE Umgebung importiert, um die nächsten Schritte Richtung physikalische Realisierung zu absolvieren. ISE hat ein eigenes Synthesewerkzeug(XST), die Synplicity aber bietet einige Vorteile, die für diese Arbeit unverzichtbar sind.

Die nachfolgenden Schritte sind in ein Makefile zusammengefasst, und werden mit Hilfe von „make“ automatisiert ausgeführt. Bei der Ausführung verwendete Befehle gehören zur ISE Entwicklungsumgebung, und werden in Batch Modus ausgeführt.

NGD-Datenbasis

Die NGD-Datenbasis ist eine logische Datenbasis, in der das erstellte VHDL Design als hierarchische Verschaltung von Xilinx-Komponenten-Makros dargestellt wird. Die Datenbasis wird mit Hilfe von „ngdbuild“ erstellt. Die Einstellungen in der UCF-Datei werden ebenfalls mit „ngdbuild“ in der NGD-Datenbasis eingetragen.

Technologie-Mapping

Die Abbildung der minimierten Schaltfunktionen und von der Xilinx-Komponenten-Makros auf die Ziel-FPGA ist ein sehr rechenintensiver Vorgang. Hier werden in der Netzliste definierte logische Schaltungen auf reale physikalische Schaltungen, die in der FPGA vorhanden sind, abgebildet. Dabei wird versucht Platz zu sparen, und um das Zeitverhalten der Schaltung zu verbessern, die miteinander verbundene LUTs und Flipflops so in CLBs zu verpacken, dass die Gesamtverdrahtungslänge minimal ausfällt. Die Technologie-Mapping wird mit Hilfe vom in ISE Entwicklungsumgebung enthaltenen „map“ Programm ausgeführt.

Place and Route

Bei Platzierung und Verdrahtung der Technologieelemente werden heuristische Algorithmen verwendet, um optimale Ergebnisse zu erzielen. Aus diesem Grund sind die Kosten für diesen Technologieschritt schwer abzuschätzen. „Place and Route“ wird in zwei sukzessive Schritte ausgeführt: Platzieren und Verdrahten. Beide Prozesse sind sehr rechenintensiv, und haben als Ziel eine optimale Platzierung zu finden, und anschließend so zu Verdrahten, dass möglichst alle (zeitliche)Randbedingungen erfüllt werden können. Die Ergebnisse vom „Place and Route“ haben einen großen Einfluss auf die Performance der Schaltung.

Das Programm „par“ führt die Platzierung und Verdrahtung der in der Netzliste definierten Komponente aus.

Bitstreams erstellen

Der Bitstrom wird mit Hilfe des ISE-BitGen-Programms aus NCD-Datei erzeugt. Eine NCD-Datei ist ein vollständig platziertes Design, eventuell mit Routing-Informationen ergänzt. Das Ergebnis ist eine FPGA-Konfigurationsdatei(Bit-Datei), die noch verschlüsselt und direkt in die FPGAs geladen werden kann[18].

3.5.3 Programmieren vom FPGA

Sind die Bitströme erzeugt, dann können diese Konfigurationsdateien mit Hilfe von Xilinx bereitgestelltes Tool iMPACT in der FPGAs im Boundary Scan Mode neu konfiguriert werden. Dabei werden im FPGA physikalische Verbindungen zwischen den einzelnen FPGA Komponenten entsprechend der Netzliste hergestellt. Die Verbindungen entstehen durch die programmierbaren Schalter, die die entsprechende Leitungssegmente zusammenschalten. In dieser Phase werden noch zusätzliche Optimierungen vorgenommen, um die entstehende Pfade zu minimieren, und gleichzeitig die Timing-Bedingungen zu erfüllen. Neben der Verdrahtung werden die LUTs mit den berechneten Werten oder mit Boole'schen Gleichungen initialisiert.

Die Konfiguration der FPGAs wird mit Hilfe vom Xilinx Programmier- und Debuginterface Parallel Kabel IV über die JTAG-Schnittstelle vorgenommen. Xilinx Spartan 3A verwendet ein USB-Kabel, um die Programmierung der FPGAs durchzuführen. JTAG ist die Bezeichnung für den von der Joint Test Action Group entwickelten Standard IEEE 1149.1.

3.6 FPGA Sicherheit

„Die Virtex-II-Pro-Bausteine besitzen eine integrierte Entschlüsselungs-Engine, die sich zur Sicherung des Konfigurations-Bitstroms und damit auch des FPGA aktivieren lässt. Den Bitstrom verschlüsselt man in der Xilinx-Software mit Hilfe eines Schlüsselsatzes, und das Virtex-II-Pro-Bauteil entschlüsselt den einlaufenden Bitstrom intern, anhand des gleichen Schlüsselsatzes.

Sobald das Design in den ISE-Tools einen Platzierungs- und Routinglauf hinter sich hat, wird der verschlüsselte Konfigurations-Bitstrom anhand des ISE-BitGen-Programms mit benutzerdefinierten Schlüsseln generiert. Die gleichen Schlüssel werden über den JTAG-Port mit Hilfe der ISE-iMPACT-Tools in das FPGA geladen.

Um den Schlüssel in den Baustein zu programmieren, muss dieser in den Key Access Modus versetzt werden, der das FPGA automatisch löscht – einschließlich aller alten Schlüssel und des aktuellen Bitstroms. Auch diese Maßnahme bietet Schutz gegenüber Angriffen.

Nachdem der Baustein mit den Schlüsseln programmiert ist, kann man ihn mit dem verschlüsselten Bitstrom konfigurieren. Sobald der verschlüsselte Bitstrom in das FPGA programmiert ist, lässt sich dieses weder durch unbeabsichtigte noch durch beabsichtigte Eingriffe oder Einbrüche rekonfigurieren, partiell konfigurieren oder auslesen. Jeder Versuch, ein Design zu stehlen, führt automatisch zur Löschung des gesamten FPGA (für Test- und Debugging-Zwecke lassen sich allerdings nicht-verschlüsselte Bitströme in ein FPGA programmieren, wo im vornhinein Schlüssel geladen wurden).

Die Schlüssel sind in einem speziellen Dreifach-DES-Block an einer Ecke des Bauteils angeordnet. Eine externe Batterie dient zur Speicherung der Schlüssel, wenn die Leiterplatte nicht unter Spannung steht. Jede normale Batterie mit einer Spannung zwischen 1 und 4 V lässt sich verwenden; das ermöglicht eine Lebensdauer von 15 Jahren.

Obwohl nicht-flüchtige PLD-Sicherheitsbits Lese-/Schreibzugriffe auf den Bauteil sperren, bietet die Dreifach-DES-Funktion von Virtex-II Pro mit Verschlüsselung echte Sicherheit. Und selbst wenn die Anordnung des Dreifach-DES-Blocks in den Virtex-II-Pro-FPGAs bekannt ist, lassen sich die Schlüssel ohne eine Löschung des Chips nicht abfangen.

Eine weitere Schutzbarriere gegen potenzielle Diebe sind neun Metall-Schichten. Weil bei der Speicherung der Sicherheitsbits keinerlei feste Verdrahtung beteiligt ist, sind sämtliche Versuche zum Scheitern verurteilt, die in den Virtex-II-Pro FPGAs verborgenen Sicherheitsbits per Wärmebildtechnik zu finden.“ [13]

Kapitel 4 **Die Webschnittstelle**

Das Projekt Robot4Web ist so geplant und ausgeführt, dass die Anwender über das Internet alle notwendigen Schritte für die Steuerung des Roboters bzw. die Simulation und Synthese von VHDL - Codes ausführen können. Die hier beschriebene Webschnittstelle erlaubt es, all diese Aufgaben über eine grafische Weboberfläche zu steuern.

Um diese umfangreichen Aufgaben erledigen zu können, ist noch zusätzlich eine auf Rollen basierende Anwenderverwaltung dazugefügt. Die Website basiert auf Microsoft.Net Technologie, und läuft auf ein Microsoft Internet Information Server(IIS 7) Webserver unter Windows Vista.

Dieses Kapitel behandelt den Entwurf und die Funktionsweise aufgabenspezifischer Schnittstellen.

4.1 **Zugriffskontrolle und Anwenderverwaltung**

Wenn bestimmte Personen keine Zugriffsberechtigung auf R4W Webseite haben, muss ihnen der Zugriff auf die entsprechende Schnittstelle verweigert werden. Um die richtigen Zugänge zu gewähren, werden die Nutzer in drei Gruppen geteilt, und die entsprechenden Rollen über die Rollenverwaltung zugeordnet. Demnach wird folgende Rollenteilung aufgestellt:

- Nutzer, die nur surfen dürfen, aber sonst keine Dienste nutzen sollen,
- Anwender, die Programme bedienen, und den Roboter steuern dürfen,
- Administratoren mit erweiterten Befugnissen.

Für die Nutzer sind keine Verwaltungsmaßnahmen zu treffen, da für diese Gruppe grundsätzlich kein Zugang in das sensible Bereich gewährt wird. Anwender müssen einen Eintrag in der Benutzerverwaltung haben, und sind in der Rolle „user“ zusammengefasst. Administratoren bekommen über die Benutzerverwaltung weitere Rechte zugewiesen, der verwendete Rollenname ist „admin“. Die erteilten Zugriffsberechtigungen sind in den WEB.CONFIG Dateien gespeichert. Da die Website in Verzeichnissen strukturiert ist, wird in jedem Webseitenverzeichnis ein WEB.CONFIG Datei erstellt, in dem die Zugriffsberechtigungen auf diese Verzeichnisse erteilt beziehungsweise verweigert werden.

Um die Anwender und Administratoren zu erkennen, um diesen Personen die zugewiesenen Ressourcen freizugeben, wird eine formularbasierte Authentifizierung verwendet. Für formularbasierte Authentifizierung werden die anwenderspezifische Daten wie: Anwendername, E-Mail, Passwort usw. in einer SQL-Express Datenbank erfasst. Für die erfolgreiche Identifizierung sind Passwort und Anwendername notwendig. Die Datenbankdatei mit Benutzerkonten und mit der Definition bestimmter Rechte ist in „App_Data“Verzeichnis auf dem Webserver abgelegt.

Die Zugriffskontrolle bietet weiterhin die Möglichkeit, verlorene Passwörter durch neu generierte zu ersetzen, oder bestehende Passwörter zu verändern. Die eingegebenen Anwenderdaten werden zusätzlich noch dazu benutzt, um für jeden Anwender ein Verzeichnis zu erstellen. In diese Verzeichnisse werden die hochgeladene VHDL - Programme und die vom System generierte Rückmeldungen gespeichert.

Durch die Webschnittstelle kann ein Administrator sehr einfach Anwendern Zugang gewähren beziehungsweise entziehen.

4.2 Streaming

Um die Roboterbewegungen zu verfolgen, werden in der Projektumgebung Webcams verwendet, die die erfassten Bewegungen des Roboterarms über das Internet bereitstellen.

Neben der verwendeten Webcams wird noch das Streamingprogramm G4 eingesetzt, um gespeicherte Videoaufnahmen für didaktische Zwecke wiederzugeben. Zusätzlich kann man das Programm G4 einsetzen, um Webcams ohne integrierte Webserver, „live Streaming“ zu ermöglichen. Streamingprogramme übertragen einen kontinuierlichen Fluss von Video- bzw. Audiodaten zwischen Webserver und Client. Im Project Robot4Web wird ein in MPEG Format codierte Datenübertragung verwendet.

Die eingesetzte Linksys WVC54GC Webcam bietet Webserver-Funktionalität, und ermöglicht die direkte Bildübertragung über das Internet. Die Kamera sendet einen Videostream kodiert mit MPEG 4 bei einer Auflösung von 320 x 240 Pixeln. Die Kamera kann auch auf Bewegungen im Raum reagieren, mit einer Aufzeichnung beginnen, und den Administrator oder die Anwender per E-Mail oder SMS informieren. Da die Linksys WVC54GC Webcam eine geringe Auflösung hat, eignen sich von der Kamera bereitgestellte Bilder nicht für weitere Bildverarbeitung. Die bereitgestellte Funktionalität wird für Dokumentierung von Roboteraktivitäten verwendet.

Das Programm G4 von der Firma „CT-Tech GbR“ ermöglicht das Versenden übers Internet bis zu vier Videoströmen gleichzeitig. Die wichtigste Eigenschaft des Programms ist aber, dass Videostream in hohe Auflösung gesendet werden können. Durch die eingesetzte Encoder-Technologie ermöglicht das Programm die live - Übertragung von Audio- und Videosignalen in DVD-Qualität. Um gleichzeitig aufgezeichnete Videodaten und Webcam-Ströme senden zu können, werden bis zu vier, von Port 80 abweichende Ports verwendet. Mit Hilfe

einer hochauflösenden Kamera, ermöglicht das Programm die gewonnenen Videoströmen über Musterkennungsprogramme zu bearbeiten und auszuwerten.

Um die gesendete Streaming nutzen zu können, ist auf der Empfängerseite ein Wiedergabeprogramm zu installieren, das entweder ein in Webbrowser integrierte Plug-in ist, oder auch ein eigenständiges Wiedergabeprogramm. Für die MPEG - Format ist die Windows Media Technologie-Plattform zu empfehlen.

4.3 Simulation und Synthese

Die Webschnittstelle, in Zusammenarbeit mit dem Job Scheduler, ermöglicht die Simulation und Synthese von bereitgestellten VHDL - Programmen. Dazu werden die von den Anwendern erstellten VHDL - Dateien auf dem Webserver geladen. Das Hochladen von Programmen oder die Datenübermittlung für die Robotersteuerung wird über die Webschnittstelle abgewickelt. Die Daten können mit einem Zippprogramm komprimiert, oder aber in nicht komprimierter Form übertragen werden. Das System erkennt automatisch die komprimierten Daten, und entpackt sie im anwenderspezifischen Verzeichnis.

Die Simulation und die Synthese von VHDL - Codes müssen nicht zwingend auf demselben Rechner ausgeführt werden, wo die Webserver-Applikationen laufen. Dieses zu entscheiden, ist die Aufgabe des Job Schedulers, mit dessen Hilfe Applikationen auf anderen Rechnern im selben Netzwerk ausgeführt werden können.

Um diese Interfaces benutzen zu können ist es erforderlich, dass die Anwender sich am Server authentifizieren. Bei dem Authentifizierungsvorgang werden die eingegebenen Daten mit den Anwender-Daten aus der Datenbank verglichen, und der Zugang wird mit den zugewiesenen Rechten gewährt. Die Daten für die Anwenderverwaltung kann man aus der Zentralen TU-Datenbank beziehen, oder es wird einfach überprüft, ob der User von einer TU-Wien - Webseite kommt, wo er schon im Vorfeld eine Authentifizierung erfolgreich ausgeführt hatte.

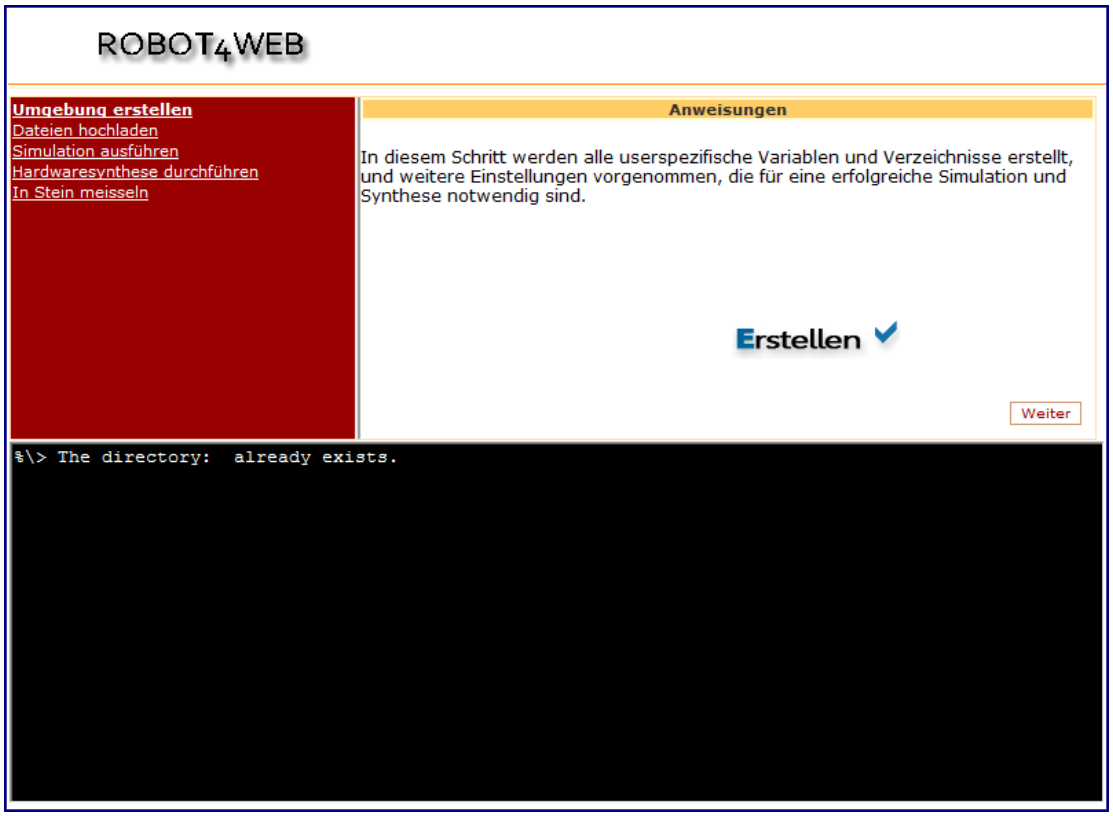

Abbildung 8: Webschnittstelle für Simulation und Synthese von VHDL - Dateien.

4.3.1 Seitenaufbau

Die für die Simulation und Synthese entworfene Webschnittstelle ist in drei Bereichen unterteilt, und besteht aus einem Navigationsteil, einem Bedienfeld und einem Ausgabefenster.

Die Navigationshilfe ermöglicht die Orientierung auf der grafischen Oberfläche, und die direkte Auswahl einzelner Zwischenschritte.

Das Bedienfeld beinhaltet zusätzliche Hinweise bzw. Anweisungen, die bei der Ausführung des gewählten Schrittes zu beachten sind. Für jede Ausführungsphase erscheint ein neues Bedienfeld mit dem, für diese Phase notwendigen Optionen. Mit Hilfe von Navigationsbuttons kann man Schritt für Schritt den gesamten Erstellungsprozess durchlaufen lassen, oder bestimmte Zwischenschritte wiederholen.

Das Ausgabefenster ist wie ein DOS-Shell designt, und soll dem Anwender das Gefühl vermitteln, dass die Programmausführung in einer Kommandozeilenebene stattfindet. Das Ausgabefenster bekommt alle Meldungen von den ausgeführten Prozessen, einschließlich solche Fehlermeldungen, die über den „standard error“-Stream geschickt wurden. Da die Programmausführungen über Kommandozeilenaufrufe abgewickelt werden, ist nicht schwer, die stderr Stream (also Standard Error) mit Hilfe von Pipelines umzuleiten und anschließend in das Ausgabefenster zu kopieren.

4.3.2 Zwischenschritte für Simulation und Synthese

Um Bitströme über die Webschnittstelle aus VHDL-Modellen zu erzeugen und anschließend in den FPGA zu laden, sind fünf sukzessive Schritte auszuführen. Jeder einzelner Schritt ist direkt aufrufbar, was die erneute Ausführung erfolgreich absolvierten vorangegangenen Vorgängen überflüssig macht.

Umgebung erstellen

In diesem Schritt werden alle Anwenderspezifische Variablen und Verzeichnisse erstellt, und weitere Einstellungen vorgenommen, die für eine erfolgreiche Simulation und Synthese notwendig sind. Die Ergebnisse dieses Schrittes werden umgehend im Ausgabefenster angezeigt. Die Erstellung der anwenderspezifischen Umgebung ist nur einmal auszuführen, und bleibt solange bestehen, bis der Administrator diese Daten löscht. Der Anwender wird jedenfalls darüber benachrichtigt, ob die Umgebung erstellt wird oder bereits existiert. Auf Abbildung 4 ist folgendes ersichtlich: der Anwender wird darüber informiert, dass das zu erstellende Verzeichnis bereist existiert.

Dateien hochladen

Über die, auf Abbildung 5 gezeigte Webschnittstelle kann man die Projektdateien hochladen. Bei dieser Übertragung handelt es sich nicht um FTP-, sondern HTTP-Datenübertragung; im Zuge dessen wird die maximale Upload-Größe auf 4MB eingeschränkt. Nach der erfolgreichen Datenübertragung wird serverseitig überprüft, ob die hochgeladene Dateien komprimiert sind. Falls es sich um eine ZIP-Datei handelt, wird diese anschließend ausgepackt. Bei der Dekomprimierung von ZIP-Dateien werden die, vom ZIP-Programm generierte Meldungen über dem Ausgabefenster zurückgegeben. Dazu ist es notwendig, dass das Programm zum Entpacken von komprimierten Dateien - in diesem Fall das UNZIP Programm - am Webserver vorinstalliert ist.

Im nachfolgenden Codeausschnitt wird die als Parameter übergebene Datei entpackt, dabei wird der Stream der Standardfehlerausgabe (stderr) so umgeleitet, dass gemeinsam mit der Standardausgabe geloggt werden kann:

```
@echo off
unzip -u %1 -d %2  2>&1
```

Simulation ausführen

Um eine VHDL - Modell zu simulieren, wird Modelsim verwendet. Modelsim wandelt mit Hilfe des VHDL - Compilers die VHDL Codes in eine für den Simulator verständliche Form um, dabei wird die Syntax des Codes überprüft, und nach vorhandenen Bibliotheken gesucht. Die gefundenen Bibliotheken werden zu Simulation von Modelsim geladen. Um die zu be-

trachtende Signale festzulegen, und um die Simulation ausführen zu können, muss der Anwender ein Testbench erstellen.

Für die Simulation über die Webschnittstelle werden Kommandozeilenaufrufe verwendet. Diese Kommandos sind ausführbare Programme, und lassen sich von einer Unix-Shell aus, ohne grafische Oberfläche von Modelsim, aufrufen. Die erzeugten Meldungen werden über dem Ausgabefenster dargestellt. Falls Dateien oder Grafiken erstellt werden, dann sind diese in dem Verzeichnis abgespeichert, der für den Anwender eingerichtet ist. Die Anwender haben dann später die Möglichkeit, die von Modelsim generierten Dateien herunterzuladen und zu analysieren.

Hardwaresynthese durchführen

Nachdem mit Hilfe der funktionalen Simulation die Korrektheit des FPGA-Designs überprüft ist, wird das Design synthetisiert. Als Synthesewerkzeug wird Synplify von Synplicity, oder ISE von Xilinx verwendet. Beide Programme lassen sich gleichzeitig einsetzen, da ISE-Projekte die Einbindung von Synplify als Synthesewerkzeug erlauben.

Durch das Auswählen vom Menüpunkt “Synthese Starten“ im Navigationsbereich wird die Simulation durch den Webinterface angestoßen. Tatsächlich wird eine Datei auf dem Webserver angelegt, die die Simulation beschreibt. Die Erzeugung dieser Datei veranlasst den Jobscheduler, die Hardwaresimulation auszuführen. Die endgültige FPGA-Implementierung wird jedoch noch nicht festgelegt. Das Ergebnis der Synthese ist die Netzliste, die für die Generierung der Bitströme weiter verwendet wird.

Wie bei den vorangegangenen Schritten, werden die erzeugte Meldungen im Ausgabenfenster wiedergeben.

Bitströme generieren

Nachdem die Simulation und die Hardwaresynthese erfolgreich abgeschlossen sind, werden die Bitströme generiert, und anschließend in den Virtex II Pro FPGA geladen. Die Generierung der Bitströme ist der letzte Schritt, den man absolvieren muss, um von VHDL – Modellen zum endgültigen Programm zu gelangen.

Ein FPGA wird konfiguriert, indem seine Architekturelemente mit den geeigneten Konfigurationsbits belegt werden. Die Gesamtheit aller Konfigurationsbits, gemeinsam mit den herstellerspezifischen Konfigurationsdaten, bezeichnet man als Bitstrom (Bitstream).

Der Bitstrom wird aus dem platzierten und verdrahteten Entwurf (noch in digitaler Form) durch spezielle Werkzeuge des FPGA - Herstellers generiert.

• Die wichtigsten Elemente eines Bitstroms sind die Konfigurationsbits, die u.a. zur Konfiguration der LUTs, der Schaltmatrizen und der Block RAMs dienen.

• Die Konfigurationsbits werden in Frames geteilt, die für die Konfiguration entsprechender Zonen des Konfigurationsspeichers auf dem FPGA verantwortlich sind.

Zum Beispiel: VirtexII-Pro XC2VP30 enthält 11.575.552 Konfigurationsbits, die in 1.756 Frames der Größe 6.592 geteilt sind.

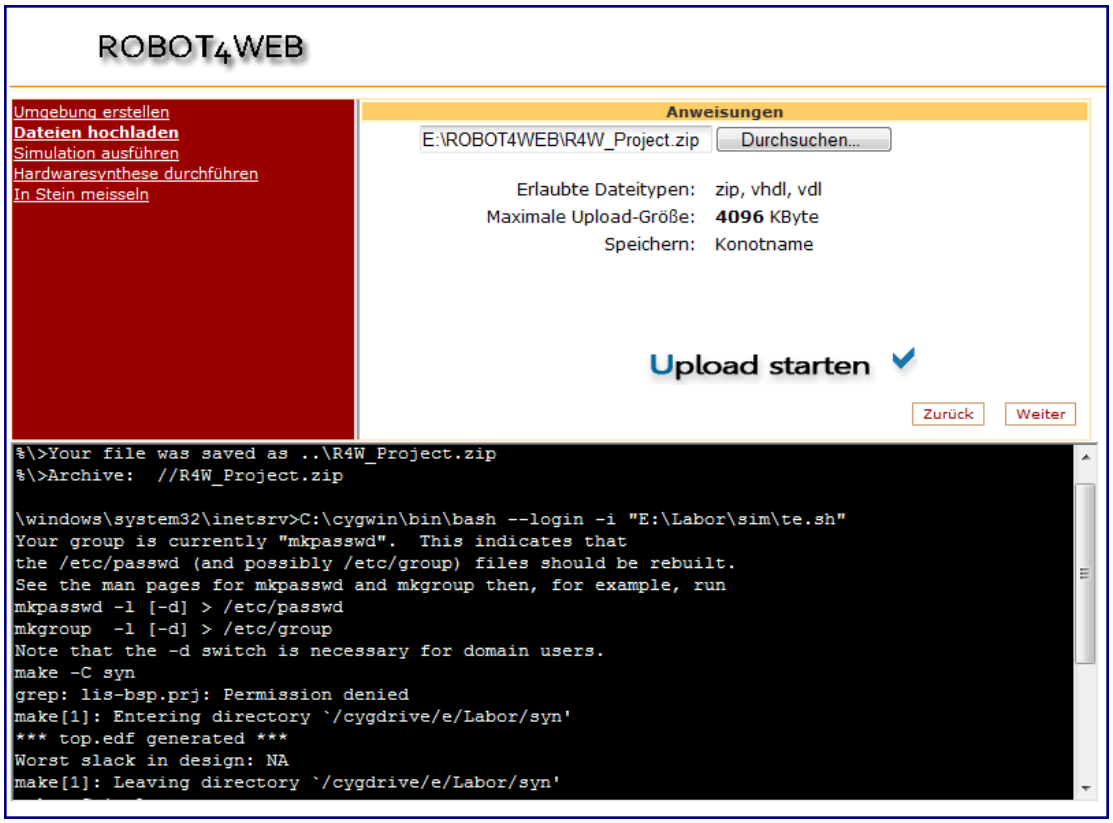

Abbildung 9: Hochladen von VHDL - Modulen und die anschließende Programmausführung

Da die VHDL Simulation und Synthese unter bestimmten Umständen länger brauchen, um Ergebnisse zu liefern, und um die Rückgabewerte nicht unnötig zu verzögern, werden moderne Rechner mit schnellerer Architektur für diese Aufgabe eingesetzt.

Mit Hilfe von zusätzlichen Programmen wie Job Scheduler, ist es möglich, das Robot4Web System dezentral; also auf unterschiedlichen Rechnern unter besserer Lastenaufteilung zu betreiben.

4.4 R4W Seitemap

Auf Abbildung 6 dargestellte Site Map bietet eine detaillierte Übersicht von der Robot4Web Website. Die hellgrau unterlegten Teile symbolisieren die Bereiche mit eingeschränktem Zugang.

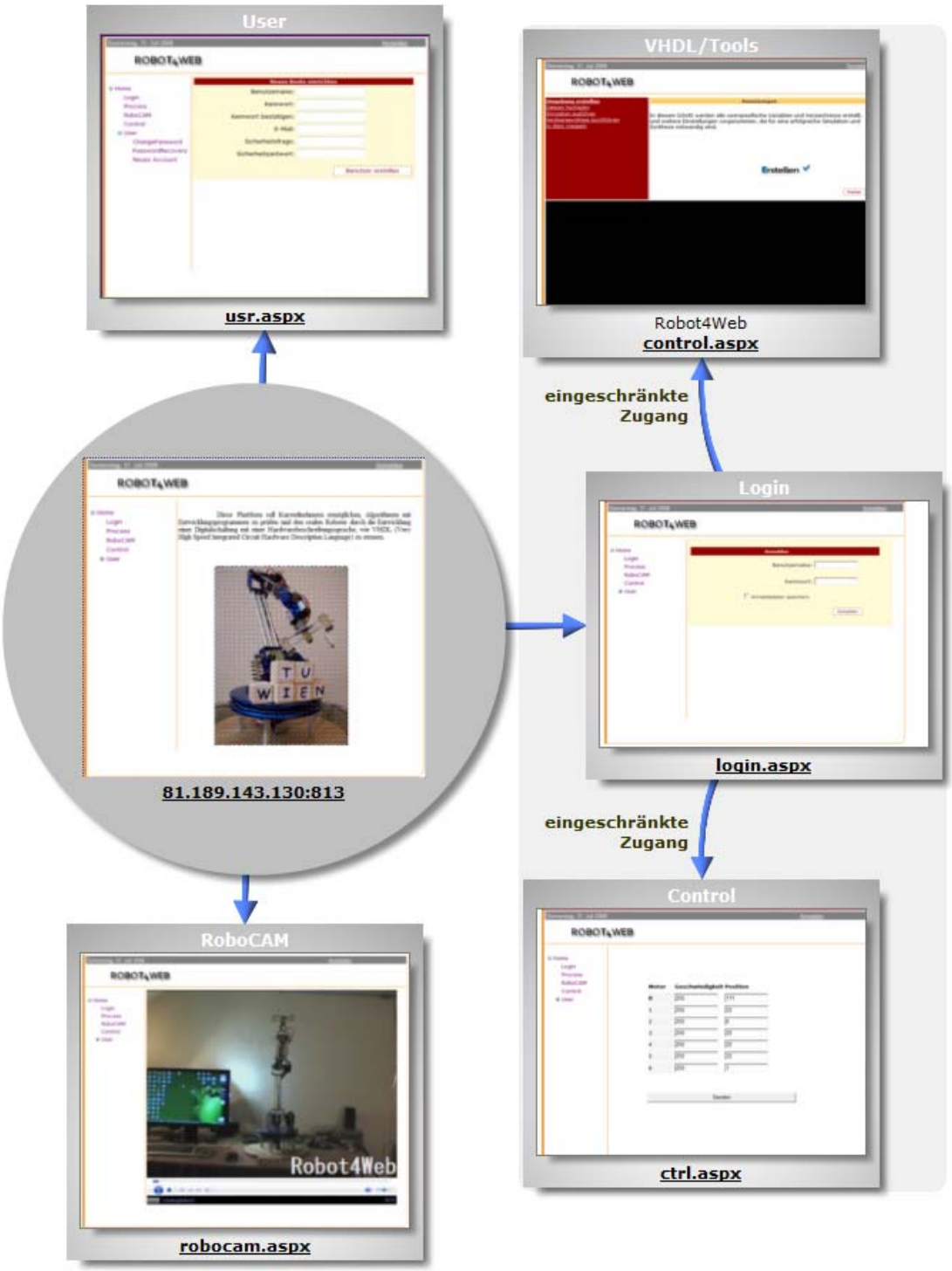

Abbildung 10: Robot4Web Webseitenaufbau

Kapitel 5 **Der Roboter R4W**

Der Begriff Roboter leitet sich vom tschechischen Wort „robota“ ab, was soviel wie Frontarbeit oder Sklavenarbeit bedeutet. Geprägt hat diesen Ausdruck der tschechische Bühnenautor Karel Capek 1921, durch sein Theaterstück Werstands Universal Robots. Bis Mitte des 20sten Jahrhunderts wurden Roboter als künstliche Wesen dargestellt, die intelligent agieren und sogar gewisse Rechte in der Gesellschaft haben. Die Roboterentwicklung ist von Science-Fiction-Schriftsteller geprägt, die sehr viel zu der Faszination Robotik beigetragen haben. Wenn man über die sozialwissenschaftliche Auswirkungen moderner Technologien spricht, darf man den Wissenschaftler und Autor Isaac Asimov nicht unerwähnt lassen. Isaac Asimov veröffentlichte 1942 die Drei Gesetze der Robotik:

1. Ein Roboter darf weder Menschen Schaden zufügen, noch durch Untätigkeit Beschädigungen zulassen.
2. Ein Roboter muss stets den menschlichen Befehlen nachkommen, sofern er dabei nicht die erste Grundregel verletzt.
3. Ein Roboter muss sich selbst vor Schaden bewahren, sofern dies nicht im Widerspruch zur ersten und zweiten Grundregel steht.

Thomas Ross entwickelte 1938 in Amerika den ersten wissenschaftlichen Roboter. Es handelte sich dabei um eine kleine Maschine, die wie eine Maus aus Versuche und Irrtümer lernend, den Weg aus einem Labyrinth herausfand. Die Roboterentwicklung geht heute in rasanten Schritten weiter, und die neuen Robotergenerationen werden von Jahr zu Jahr menschenähnlicher und auch erschwinglicher. In unserem Alltag finden wir heute Roboter fast in allen Aspekten unseres Lebens, und so werden wir heute Roboter in der Medizin, Personenpflege oder sogar bei Gartenarbeiten wiederfinden.

In diesem Kapitel wird die grundlegende Vorgehensweise und die Realisierung eines Roboterarms beschrieben. Dabei wird ausführlich diskutiert, welche mechanischen Voraussetzungen ein Roboterarm zu erfüllen hat. Im Kapitel 5.3 wird die Robotersteuerung beschrieben.

5.1 Entwurf

Bei dem Bau eines Roboters sind einige Fragen schon im Voraus bezüglich des Einsatzbereichs, der Funktionalität und der Sicherheit zu klären. Dabei muss zu Beginn erst einmal geklärt werden: was für einen Gegenstand soll der Greifer heben können. Wichtige Aspekte sind dabei Größe, Gewicht, Form und Festigkeit. Zielsetzung ist: einen Roboter für Laborarbeiten zu entwickeln, der über ein FPGA - Board steuerbar ist und kleine Gewichte, wie zum Beispiel Holzklötze greifen und positionieren kann.

Der Roboterarm wird dem menschlichen Arm nachempfunden, und benötigt sechs Gelenke und 9 Motoren. Für die Realisierung würden fünf Gelenke ausreichen, aber erst durch einen zusätzlichen Freiheitsgrad wird es möglich, den Testgegenstand aus unterschiedlichen Winkeln zu greifen. Wenn man den Roboterarm nach der Anzahl der Achsen einordnet, dann ist R4W ein: Sechs- Achs- Knickarm- Roboter.

Für den Roboterbau notwendige Materialien werden aus Baumärkten (Alu-Profilrahmen, Schrauben, etc.), Elektro- und Modellbaugeschäften bezogen.

5.1.1 Arbeitsbereich und Freiheitsgrad

Der erste Schritt besteht darin, einen Arbeitsraum festzulegen. Jedes Roboter-Armelement ist entweder eine rotatorische oder eine translatorische Achse. Die Art und die Anordnung der verwendeten Achsen bestimmen die Form des Arbeitsraumes. Die grundlegenden Gelenke sind: Rotationsgelenk, Torsionsgelenk, Revolvergelenk und Lineargelenk.

Alle Punkte des dreidimensionalen Raumes, die durch den Greifer des Roboters erreichbar sind, gehören zum Arbeitsraum des Roboters. Der Arbeitsraum bestimmt weiterhin die Grundkonfiguration des Roboters. Nach Art des Arbeitsraumes unterscheidet man zwischen Roboter mit:

1. Gelenkarm
2. kartesischen Koordinaten
3. Zylinderkoordinaten
4. Schwenkarm
5. Polarkoordinaten

Für diese Diplomarbeit wird ein Roboterarm in Kugelkoordinatenbauweise (Hohlkugel) mit Gelenkarm erstellt. Der Radius des Kugelarbeitsraumes ist annähernd 75 cm, das entspricht der ausgestreckten Roboterarmlänge. Der Roboterarm R4W, der sechs Freiheitsgrade hat, kann in seinem Arbeitsraum mit seinem Effektor jeden Punkt aus aller möglichen Orientierung erreichen. Die Voraussetzungen dafür sind, dass die Achsen nicht redundant sind, und die Achsschwenkwinkel ausreichend groß gewählt sind. Da bei diesem Roboter jede Achse sich um ca. 220 Grad drehen lässt, wird fast der ganze, theoretisch zur Verfügung stehende Arbeitsraum zur Gänze ausgenutzt.

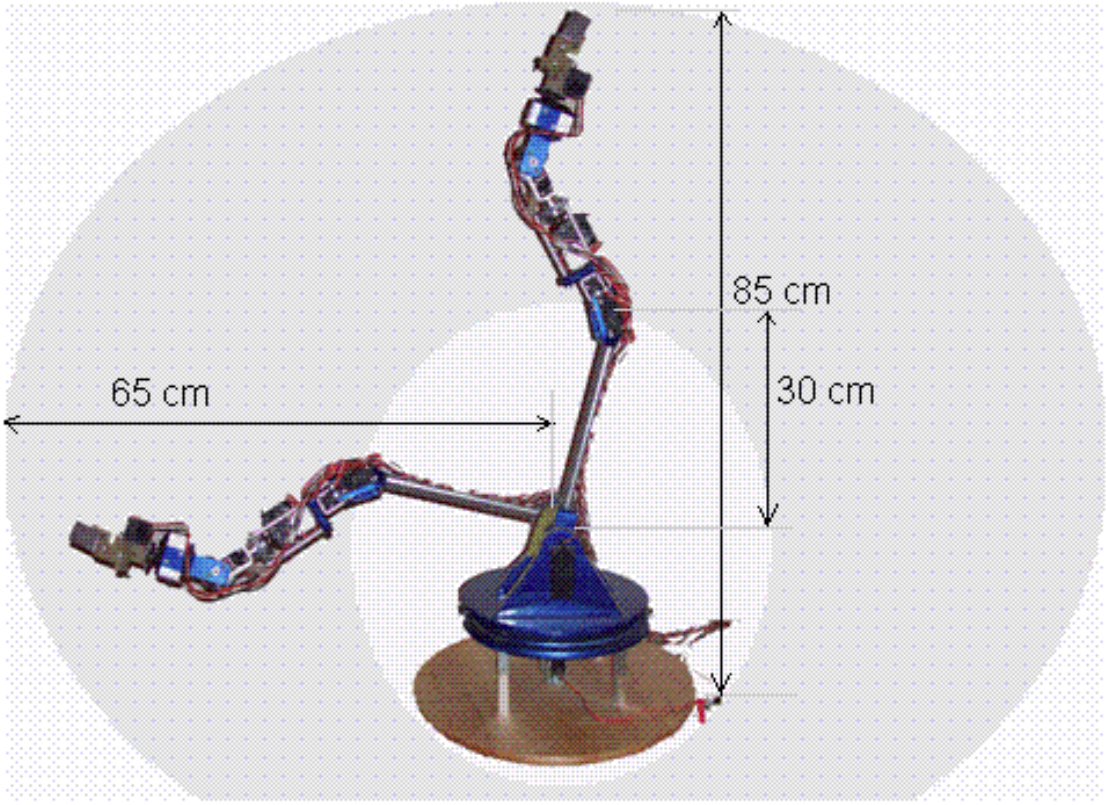

Abbildung 11: Arbeitsbereich des Roboterarms

Bei der Realisierung kommt es letztendlich darauf an, was der Roboter greifen können soll. Der Knickarm-Roboter ist sehr flexibel einsetzbar, und nach neuesten Trends werden sie immer mehr gegenüber Roboter mit anderen Bauarten bevorzugt.

5.1.2 Auswahl der Motoren

Die meist verbreitete Elektromotoren in der Robotik sind Servomotoren und Schrittmotoren. Um einen Roboterarm vernünftig bewegen zu können ist es wichtig, dass der Motor stark genug ist, und die Armkomponenten nicht überdimensioniert sind damit noch der Nutzlast ausreichend beschleunigt und gebremst werden kann.

Schrittmotoren

Schrittmotoren – aus dem Englischen abgeleitet, oft auch als Steppmotoren bezeichnet – verwenden eine einfache Schaltung, um die Motorentwicklung zu steuern. Schrittmotoren sind deutlich komplexer gebaut als Servomotoren und werden weit weniger in der Robotik eingesetzt als Servomotoren. Typische Anwendungsbereiche sind Matrixdrucker oder der Antrieb von Schreib-/Leseköpfe in Disketten- oder CD-/DVD-Laufwerken.

Schrittmotoren haben sowohl Vor- als auch Nachteile gegenüber Servos. Schrittmotoren sind in der Lage, ohne zusätzliche Sensoren Rückmeldung über die aktuelle Position zu geben. Der größte Nachteil der Schrittmotoren besteht darin, dass unter Belastung oder bei rascher Beschleunigung bzw. Bremsung, einzelne Schritte übersprungen werden können. Da diese Positionierungsfehler unter Belastung fortpflanzt, entsteht letztlich eine ungenaue Positionierung. [9]

Für den Roboterarm in dieser Diplomarbeit eignen sich die Schrittmotoren nicht, da teilweise Kräfte zu überwinden sind, die von Schrittmotoren ohne zusätzliche Getriebe nicht bewältigt werden können.

Servomotoren

Servomotoren (Kurzbezeichnung: "Servos") unterscheiden sich von den Schrittmotoren im Aufbau und in der Steuerung. Servos sind grundsätzlich robuster gebaut als die Schrittmotoren, und durch die Verwendung zusätzlicher, meist eingebauter Getriebe, haben sie ein größeres Drehmoment. Der Vorteil von Servomotoren gegenüber von Schrittmotoren besteht in der Genauigkeit, Kraft (sehr hohe Kraftübersetzung durch eingebaute Getriebe) und Dynamik (kurze Beschleunigungs- bzw. Bremsphase).

Ein Servomotor verwendet ein geschlossenes Regelungssystem, das die Geschwindigkeit oder die angefahrene Position kontrolliert. Das Anfahren und Beibehalten der vorgegebenen Position wird durch die Regler, bei ständiger Referenzierung der Sollwerte vorgenommen. Die aktuelle Position wird mit Hilfe eines an der Drehachse fixierten Potenziometers bestimmt.

Für den Roboterarm geeignete Servomotoren auszuwählen ist nicht ganz trivial, da viele Faktoren bei der Berechnung beachtet werden müssen. Es wird angenommen, dass durch Getriebe, Reibung usw. etwa 30% der Motorleistung verloren geht. Nach Abzug der geschätzten Verlustleistungen, wird das Drehmoment so gewählt, dass der Roboterarm recht flott seine maximale Endgeschwindigkeit erreicht. In der Planungsphase werden alle Berechnungen mit der vorgesehenen maximalen Nutzlast, die der Roboterarm in ausgestreckte, horizontale Lage halten bzw. bewegen kann, berechnet. Letztendlich bestimmen die zur Verfügung stehenden Motoren die endgültige Ausführung des Roboterarms.

Bei der Realisierung des Roboterarms R4W werden Servomotoren eingesetzt, die drei Anschlüsse (wie die meisten Servos) haben: VCC, PWM(pulsweitenmoduliertes Signal) und GND. Der Wertbereich für VCC liegt für alle hier eingesetzten Motoren bei 4.5V und 6.0V. Durch die Erhöhung der Versorgungsspannung von 4.5V auf 6.0V erhöhen sich die Winkelgeschwindigkeit und das Drehmoment um mehr als 10%.

Die wichtigste Größe für die Auswahl des Motors ist das Drehmoment. *Das Drehmoment ist das Vektorprodukt von Kraftarm und Kraft und wird in Newtonmeter gemessen ([SI] Nm).*

$$\vec{M} = \vec{r} \times \vec{F}$$

In der Robotertechnik werden häufig die Drehmomente in einer anderen Einheit: Kilogramm Zentimeter angegeben. 1Nm = 10.1972 kg-cm.

Da die Servomotoren in Halteposition ein anderes Drehmoment entwickeln als beim Anfahren einer Position, werden die wirkenden Kräfte in Haltekraft und Stellkraft differenziert.

Stellkraft

Als Stellkraft wird die größte Kraft bezeichnet, die auf den Servohebel im Abstand von 1cm zum Drehpunkt wirkt, und unter Belastung der Hebel noch in Bewegung halten kann.

Haltekraft

Haltekraft bezeichnet die Kraft, womit ein Servohebel im Abstand von 1cm zum Drehpunkt belastet werden kann, ohne den Hebel aus der neutralen Stellung zu bewegen. Diese Kraft ist deutlich geringer als die Stellkraft, und hat eine große Auswirkung auf das Roboterdesign. Diese Größe ist vorwiegend nur empirisch bestimmbar, da die meisten Hersteller die Haltekraft in ihrer Spezifikation erst gar nicht aufführen.

Weitere physikalische Größen, die bei dem Roboterentwurf nicht übergangen werden dürfen, sind: Stellzeit und Auflösung. Mit Hilfe der beiden Parametern lässt sich berechnen, wie schnell, und mit welcher Genauigkeit das Roboterarmglied bewegt werden kann.

Stellzeit

Diese Zeit gibt an, wie viel Zeit der Servomotor braucht, um seinen Hebel zum Beispiel um 60 Grad Stellwinkel zu bewegen. Meistens werden unterschiedliche Angaben bezüglich der Stellwinkel gemacht. So werden von einigen Herstellern 45, 60 oder 120 Grad Stellwinkel für die Stellzeit angegeben. Um die Unterschiede bei den verwendeten Motoren leichter zu erkennen, werden in der Tabelle 1 die Winkelgeschwindigkeiten aus den Herstellerangaben berechnet angegeben.

Servo-Auflösung

Gibt die maximale Anzahl der Schritte des Gesamtweges an. Berechnung der Auflösung:

$$Auflösung = \frac{Gesamtweg}{Schrittlänge}\ [Schritte] = \frac{max.Winkel}{Schrittwinkel}\ [Schritte].$$

Manchmal wird die Auflösung in Prozenten angegeben. Dieser Wert gibt an, wie viel Prozent des Gesamtweges als kleinster Stellweg realisierbar ist.

Sehr gute analoge Servos erreichen eine Auflösung von 200 bis 333 Schritte oder 0,5 bis 0,3 %. Digitale Servos können eine weitaus größere Auflösung erreichen. Spezielle Servomotoren erreichen eine Auflösung von 4096(12 bit) bis zu 65.536(16 bit) Schritte.

Bei allen Angaben, und beim Vergleichen sollte beachtet werden, dass die Elektromotoren mit unterschiedlicher Spannung betrieben werden können. Alle Servo-Daten werden für 4,8

oder für 6,0 VCC (wie bei den verwendeten Modellen) angegeben. Bei höherer Spannung arbeiten die Servos natürlich schneller und kräftiger.

Digitale Servomotoren

Digitale Servos arbeiten nach demselben Grundprinzip wie analoge Servos, mit dem Unterschied, dass die Steuerung von digitalen Servomotoren mit einem Mikroprozessor berechnet wird, anstatt von TTL-Logikbausteinen, wie das bei der analogen der Fall ist. Der eingesetzte Mikroprozessor erlaubt eine häufigere und genauere Ansteuerung als bei dem analogen Servo. Durch die häufigere Ansteuerung wird die Haltekraft ebenfalls erhöht.

In der Tabelle 1 sind alle Servomotoren einzeln aufgelistet, die bei der Realisierung des Roboters eingesetzt worden sind.

5.2 Pulsweitenmodulation

Von den verschiedenen Pulsmodulationsverfahren werden für die Steuerung von großen Lasten wie Elektromotoren, Pulsweitenmodulation, oder anders genannt, Pulsdauermodulation eingesetzt. Pulsdauermodulation gehört zu den wertkontinuierlichen Modulationsverfahren.

Bei der Pulsdauermodulation wird die Impulsdauer der Impulse eines Modulationsträgers durch die Abtastwerte gesteuert. Pulsamplitude und Pulsfrequenz bleiben dabei unverändert.

Weitere wertkontinuierliche Modulationsverfahren sind die Pulsphasenmodulation (PPM), die Pulsamplitudenmodulation(PAM), während der Pulscodemodulation(PCM) zu den wertdiskreten Pulsmodulationsverfahren gehört. [4]

Für die Servosteuerung des Roboterarms verwendetes pulsweitenmoduliertes Signal wird vom FPGA generiert, und über die IO-Blöcke an die Servomotoren gesendet. Diese PWM Signale haben eine Wiederholungsfrequenz von annähernd 50 Hz. Wegen der unterschiedlichen technologischen Realisierung gleicher VHDL - Signalgeneratoren, kann eine Abweichung von der 50 Hz - Marke auftreten. Die Implementierung gleicher VHDL-Modelle in Spartan 3A und in Virtex II-Pro ergibt leicht abweichende Impulsdauer und Wiederholfrequenz. Diese Abweichungen sind während des Betriebs unbedingt zu beachten und zu korrigieren, wenn man die FPGA-Plattform wechselt. Da durch den Einsatz verschiedener FPGAs die generierte Impulsdauer sich verändert, bei gleich bleibendem VHDL-Modell, müssen die maximale Auslenkpositionen in den VHDL-Code so gewählt werden, dass die Servomotoren nicht die vorgegebene maximale Auslenkung überschreiten können.

Die zeitliche Länge (Impulsdauer) des gesendeten Impulses entspricht dem Sollwert der Position. Ein Impuls von 1 ms bedeutet z. B. Auslenkung ganz nach links, ein Impuls von 1,5 ms ist die Mittelstellung des Servos, und 2 ms wäre Auslenkung ganz nach rechts. Die hier verwendeten Servos setzen in die Regelung, die direkt im Servomotor integriert ist, die

empfangenen Signale für die Positionierung des Roboters ein. Bei einigen Modellen wird mit Hilfe von Pulsweitenmodulation die Geschwindigkeit und Drehrichtung gesteuert.

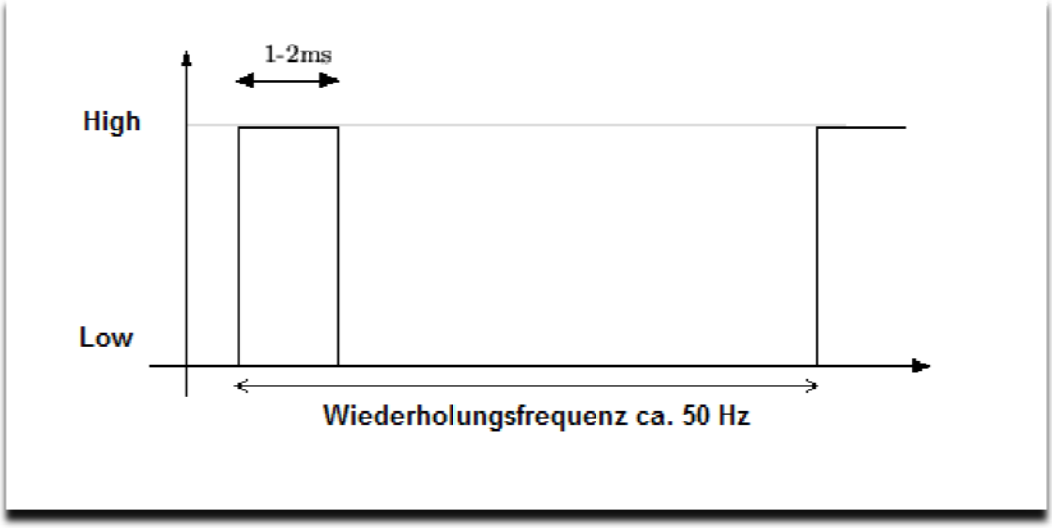

Abbildung 12: Verwendetes pulsweitenmoduliertes Signal

Die Vorgehensweise zum Erzeugen und Verstärken von PWM Steuersignale wird ausführlicher im Kapitel „Signalanalyse“ behandelt.

5.3 R4W, Technische Daten

In den nachfolgenden Tabellen werden die wichtigsten Daten zusammengefasst, die einen Roboter charakterisieren.

Achsenbewegung, Robot4Web

Achse *Nr. Bezeichnung*	*Arbeitsbereich*	*Drehmoment bei: 4.8V*	*Max. Geschwindigkeit bei:* *4,8 V*	*6 V*
0 Sockel	220°	130 Ncm	211°/s	240°/s
1 Neigung	200°	160 Ncm	211°/s	240°/s
2 Oberarm	200°	130 Ncm	211°/s	240°/s
3 Rotation A	220°	130 Ncm	211°/s	240°/s

4 Handgelenk	200°	130 Ncm	211°/s	240°/s
5 Rotation B	240°	130 Ncm	211°/s	240°/s
6 Greifer	80 mm	1,80 Ncm		

Tabelle 1: Technische Daten, Achsenbewegung

Achse *Nr. Bezeichnung*	*Motorentyp*	*Anzahl*	*Gewicht*
0 Sockel	Analog	1	49g
1 Neigung	Digital	2	146g
2 Oberarm	Analog	2	49g
3 Rotation A	Analog	1	49g
4 Handgelenk	Analog	1	49g
5 Rotation B	Analog	1	49g
6 Greifer	Analog	1	10g

Tabelle 2: Technische Daten, Motoren

Beschreibung	*Wert*
Gewicht	3,2 Kg
Anzahl der Achsen	7
Anzahl der Motoren	9
Motorentyp	Servomotor
Betriebsspannung	4,8 -6,0 V

Tabelle 3: Generelle Beschreibung

5.4 Steuerung

Im vorangegangenen Kapitel sind die Eigenschaften und die Verwendungsmöglichkeiten der in der Robotik eingesetzten Elektromotoren besprochen worden. In diesem Kapitel wird die Steuerung der Elektromotoren - insbesondere die von den Servomotoren - behandelt.

Moderne Robotersysteme werden immer komplexer und dadurch schwieriger zu entwerfen. Parallel damit steigt die Wahrscheinlichkeit von Fehlern. Die Steuerung von komplexen Robotern hat die wichtige Aufgabe, Steuersignale an den Roboter zu senden und aus Sensorsignalen den Roboterzustand zu erfassen bzw. Fehler in den Steuerungsmechanismen zu entdecken.

Da für Robot4Web ausschließlich Servomotoren eingesetzt wurden, kann man die Ansteuerung des Roboterarms sehr einfach gestalten. Bedingt durch den Einsatz von Servomotoren werden keine Zustandsinformationen von den eingesetzten Motoren an die Steuerung zurückgeführt. Eine mögliche Alternative für die Erfassung der Roboterarmzustände, wäre der Einsatz von Sensoren, wie kapazitive Näherungssensoren oder Potenziometer gewesen.

Für Robot4Web sind grundsätzlich zwei Ansteuerungsmöglichkeiten vorgesehen: Implizite und Explizite Steuerung. In der Echtzeit-Kommunikation wird noch die Ansteuerung mit harter Echtzeitanforderung behandelt, aber weil Echtzeitfähigkeit keine Anforderung an diesem Projekt ist, wird diese Ansteuerung hier nicht weiter behandelt.

5.4.1 Explizite Steuerung

Bei einer Expliziten Steuerung handelt es sich um eine Ereignisgesteuerte Führung des Roboters. Die Steuersignale zu den Servomotoren werden durch Ereignisse im Sender zu beliebigem Zeitpunkt ausgelöst.

Die explizite Steuerung findet bei dem Projekt Robot4Web zwischen einem Rechner (Sender), und mit diesem über ein RS-232 Kabel verbundene FPGA-Board als Empfänger statt. Da der Senderrechner direkt mit dem FPGA-Entwicklungsboard verbunden ist, kann der Steuerrechner andere Aufgaben, außer Steuersignale zu versenden, erledigen. Es ist sinnvoll, gerade den Steuerrechner mit Programmen auszustatten, die Synthese und Simulation auf die von dem User hochgeladene VHDL Codes erledigen. Als Steuerrechner wird in diesem Projekt ein IBM Laptop verwendet, ausgestattet mit einem RS-232 Kommunikations-Port und für die BIT-Stream - Erzeugung mit Xilinx ISE 9.2i Software.

Die senderseitige Kommunikationsaufgabe wird mit Hilfe von einem in C++ geschriebenen Programm ausgeführt. Das für diese Zwecke geschriebene Programm sendet und empfängt UART Signale mit 9600 Bit pro Sekunde. Das Programm ist so entworfen, dass es sowohl Steuerungsparameter als auch Dateien mit Kommandos übernimmt. Die Eingabe von Steuerungsparametern auf Kommandozeilenebene erlaubt dem Benutzer, einzelne Motoren

direkt zu steuern, ohne die Websteuerung zu benutzen. Diese Art von Steuerung wird zukünftig bei der Fehlersuche und bei den Tests eingesetzt. Mit Hilfe von in Dateien zusammengefassten Steuerbefehlen lassen sich komplexe Bewegungen ausführen. Bei diesen Befehlen handelt es sich nicht nur um vorgegebene Armbewegungen, sondern auch die zeitlichen Abläufe können damit koordiniert werden. Diese Möglichkeit besteht aber nur, wenn der Übergabeparameter eine Datei mit Steuerbefehlen ist.

Die Aufrufsyntax von RS232.exe :

- Bei Dateiübergabe:

  RS232.exe PORT: Filename

  z.B.: RS232.exe -p com1: –f c:\test_file.txt

- Bei Parameterübergabe:

  RS232.exe –p com1: –e 0..255 –v 0..255 –c 0..255

Das Programm RS232.EXE übermittelt an den UART Kontroller Motornummer, Geschwindigkeit und Sollposition, als achtstellige Binärzahlen. Dadurch ist es möglich, gleichzeitig mehrere Motoren mit der angegebenen Geschwindigkeit in eine angegebene Position zu fahren.

Weil das Programm nach jeder Befehlsausführung ca. 250 ms lang wartet, bis der nächste Befehl aufgerufen wird, kann man die Bewegung des Roboterarms in zeitlicher Abhängigkeit vom vorangegangenen Befehl steuern.

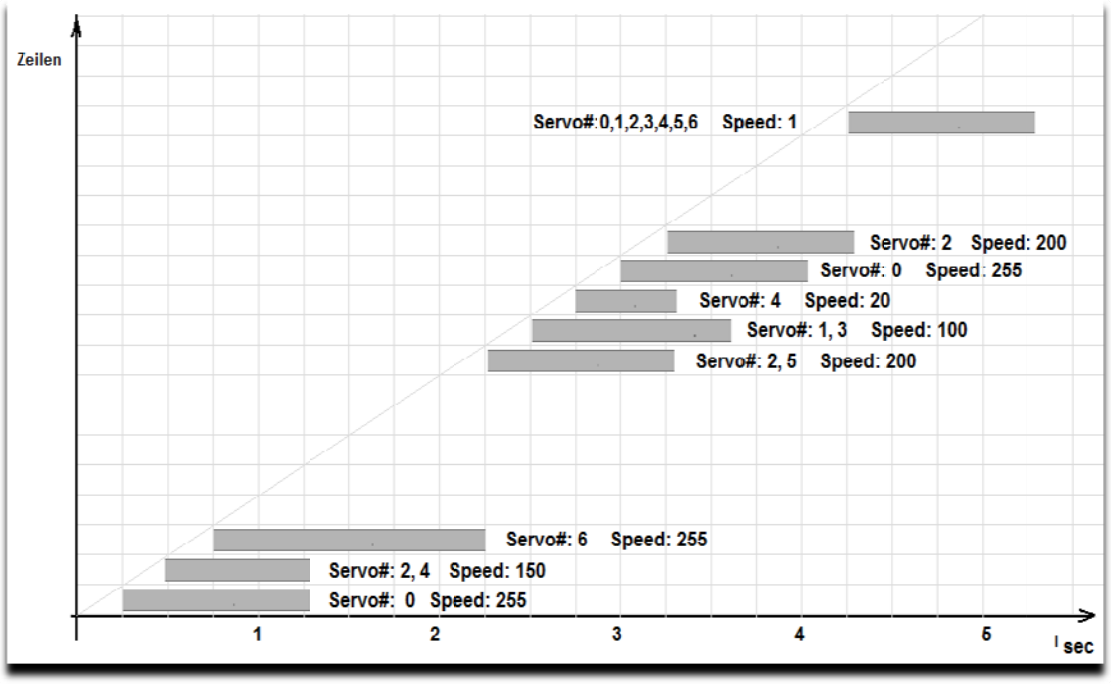

Abbildung 13: Zeitliche Abhängigkeit der Befehle

Auf Abbildung 6 ist der zeitliche Ablauf der Steuerbefehle dargestellt. Da die Steuerbefehle zeilenweise abgearbeitet werden, beginnt die Ausführung der nachfolgenden Befehle zeitversetzt. Aus der obigen Darstellung ist deutlich zu entnehmen, dass die Befehlsausführung unterschiedlich lang dauern kann. Wie viel Zeit ein Befehl bis zur vollständigen Ausführung braucht, ist abhängig von der angegeben Geschwindigkeit, und vom Weg zwischen Start- und Endposition der angesprochenen Achsen.

Leere Zeilen oder Kommentarzeilen werden verwendet, um ein vielfaches der Verzögerungszeit (250 ms) verstreichen zu lassen, und damit dem Roboterarm Zeit zu verschaffen, dass er in eine Stellung fahren kann, von wo aus dann weitere Positionen angefahren werden können. Zum Beispiel, das Platzieren von einem Holzwürfel, das Anfahren von neuen Positionen oder Greifen von neuen Testgegenständen müssen solange ausgezögert werden, bis der Greifer wieder geöffnet ist, und damit die Positionierung des Gegenstandes abgeschlossen ist.

Als Empfänger bei der expliziten Steuerung, werden in der Xilinx Spartan 3A und Xilinx Virtex II Pro erstellte logische Schaltungen verwendet. Beider Schaltungen liegt dasselbe VHDL - Programm zugrunde, nur die Konfiguration und letztendlich die technologieabhängige Realisierung unterscheiden sich voneinander.

Auf beiden verwendeten FPGA - Entwicklungsboards befinden sich RS-232-Schnittstellen zum Steuern oder zum Datenaustausch mit externen Geräten. Diese Schnittstellen sind direkt mit den IO - Blöcken auf dem FPGA verdrahtet, und übermitteln das einkommende Signal direkt auf den FPGA. Da im FPGA keine UART - Kontroller implementiert ist, müssen die

anliegenden Signale von einem synthetisierten Modul interpretiert werden. Dazu wird ein in VHDL geschriebenes Programm verwendet, das die seriellen Daten bearbeitet und an weitere Module, wie der Signalgenerator weiterleitet.

Die wichtigsten Elemente des UART - Moduls sind das Baud-Generator und das Serin-Modul für die Extrahierung und Überprüfung von empfangenen Daten. Der als logische Schaltung realisierte Baud-Generator erzeugt die Abtastfrequenz für die empfangenen Signale. Die empfangenen Signalbits werden im Baud-Generator zusätzlich für die Synchronisation der generierten Abtastrate gebraucht. Für dieses Projekt wird eine Baudrate von 9600 sowohl für das Senden als auch für das Empfangen verwendet. Dieser Wert ist relativ niedrig, aber mehr als ausreichend für die Robotersteuerung.

Der Serin-Modul ist ein „finite state machine“ (FSM), der die empfangenen Signalbits in Abhängigkeit der zugrunde liegenden Spezifikation interpretiert. Dabei werden nicht nur die Start-, Stop-, Parity- oder Datenbits erkannt, sondern es wird die Integrität des empfangenen Datenbits anhand der mitgesendeten Paritybit überprüft.

Die nachfolgende Abbildung zeigt die RTL-Layer von einem, in FPGA eingebettete UART Kontroller. Die detaillierte Beschreibung zu der seriellen Kommunikation und deren Hardwaresynthese ist im Kapitel 8 zu finden.

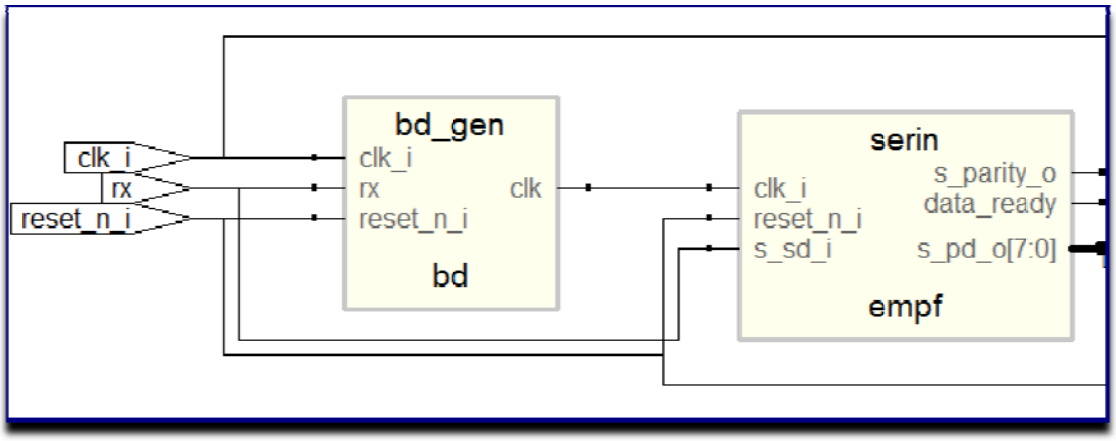

Abbildung 14: Empfangen von RS-232 Signale, RTL Schematic

5.4.2 Implizite Steuerung

Die Implizierte Steuerung ist viel einfacher zu realisieren als die explizite Steuerung, da die Kommunikationskette weniger Komponente durchlaufen muss. Bei der Impliziten Steuerung erzeugt eine im FPGA realisierte logische Schaltung die Steuersignale für den Roboterarm. Allerdings sind diese Signale von einem Programmschaltwerk zur Robotersteuerung erzeugt

worden, und lassen sich nur in den VHDL - Code modifizieren. Das aber erfordert eine neuerliche Hardwaresynthese. Der vielleicht größte Vorteil bei dieser Ausführung liegt darin, dass man Aufgaben in einer Endlosschleife ausführen kann, ähnlich wie der Einsatz von Roboter bei einer Laufbahnfertigung.

Die Aufgabe der impliziten Steuerung ist: übernehmen von VHDL – Programmen, die von den Anwendern über die Webschnittschtelle hochgeladen wurden, und nach erfolreicher Simulation und Synthese ausgeführt wurden.

Das Programmschaltwerk zur Robotersteuerung besteht aus mehreren, in VHDL beschriebenen Module: Programm Counter(PC), ROM, Decrementer, Register, Decoder und Signalgenerator. All diese Module verwenden das globale Taktsignal und Resetsignal, und sind miteinander durch innere Signale verbunden. Die eigentliche Ausgabe, die Steuersignale für den Roboterarm, werden nur von dem Signalgenerator erzeugt und zwar für jeden Servo ein Signal. Also das System besitzt die Eingänge CLK_i und RESET_i, sowie die sieben Ausgänge Engine_X_o, wobei X die Werte 0 bis 6 annimmt.

Die Aufgabe des Signalgenerators ist es aus den Eingangswerten, die den Motoren-Sollwert repräsentieren, ein kontinuierliches pulsweitenmoduliertes Signal zu generieren. Davon abhängig, welcher Wertbereich für die Sollzustände gewählt worden ist, wird die Auflösung der Steuersignale bestimmt. Wenn man nur vier Bit (vier Signalleitungen) für den Sollzustand gewählt hat, dann kann die ansteuerte Achse nur 16 unterschiedliche Positionen annehmen. Natürlich kann man durch die Erhöhung des Wertebereichs für die Sollzustandsdarstellung nicht unendlich vergrößern, ein Limit wird durch die Motorenauflösung gesetzt. Die hier verwendete Auflösung braucht acht Signalleitungen, also eine Schwenkung von ganz links nach ganz rechts kann auf 255 Zwischenschritte aufgeteilt werden. Das erzeugte Signal hat eine Wiederholungsfrequenz von 20ms. Die Impulslänge wird wie folgt berechnet:

Impulslänge $I = 1 + \frac{S}{A}$; wobei S der Sollwert und A die Auflösung ist, in ms gerechnet. Der Sollwert S ist so zu wählen, dass A größer / gleich S ist.

Das für Demonstrationszwecke entwickeltes Programmschaltwerk zur Robotersteuerung verwendet einheitlich eine Auflösung von 8 Bit. Da bei dem Roboterarm sowohl digitale als auch analoge Servos eingesetzt sind, wurde eine Auflösung gewählt, die für beide Motorarten noch gleich bleibt, obwohl die verwendeten digitalen Servos eine weit höhere Auflösung haben.

Auf der nachfolgenden Abbildung ist der Vollständig zusammengebaute Roboterarm zu sehen.

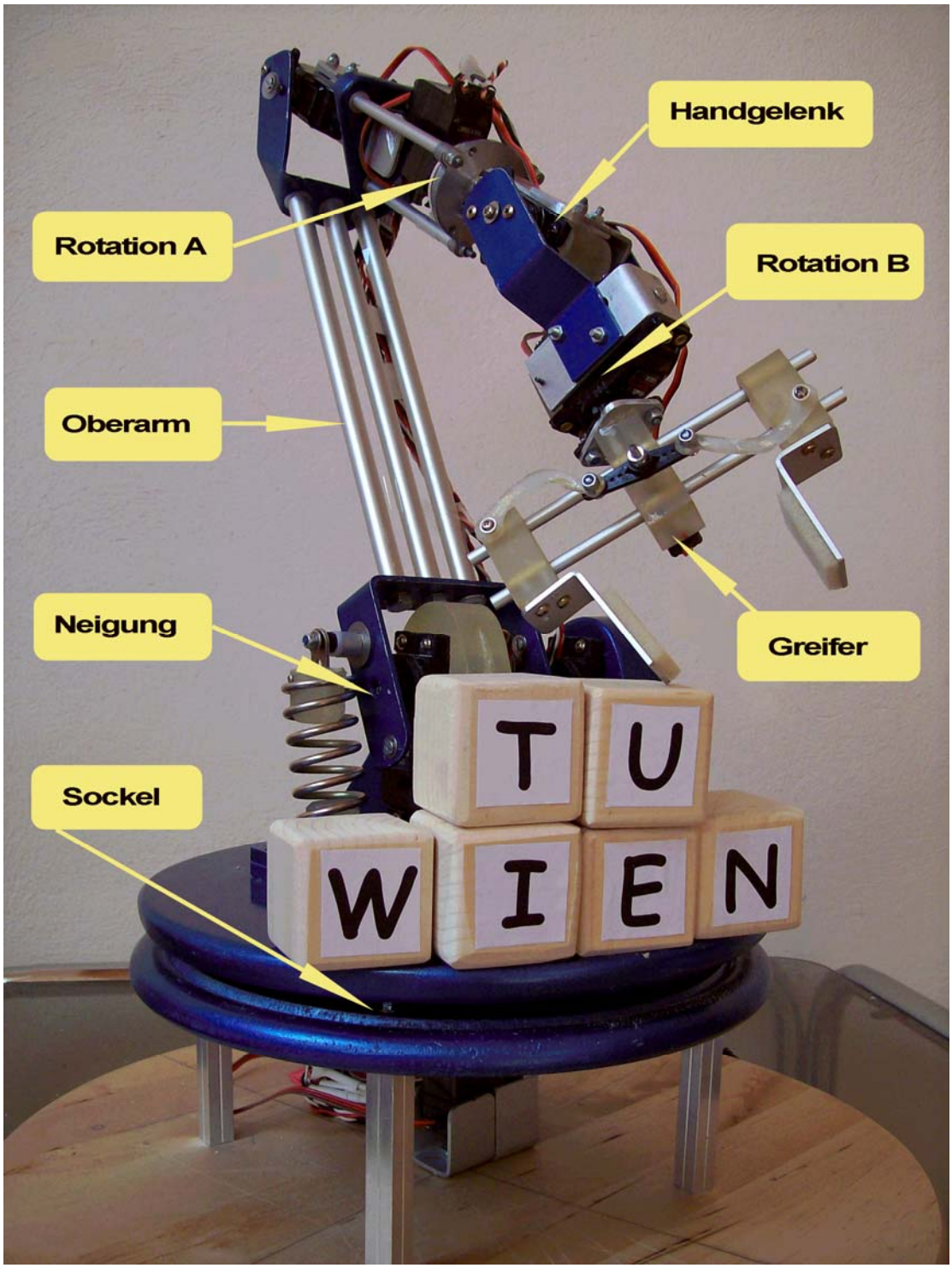

Abbildung 15: Roboterarm R4W

Kapitel 6 **Signalverstärkung und Schutz**

Während FPGAs größer und System-Taktgebergeschwindigkeiten schneller werden, wird der FPGA Board-Design und die Herstellung schwieriger. Mit immer schnellerer Taktrate wird die Erhaltung der Signalintegrität kritischer. Die FPGA-Board-Entwickler müssen darauf achten, und ständig überprüfen, dass die Signalleitungen richtig beendet werden, um Reflexionen oder andere Nebeneffekte zu vermeiden. Weiterhin müssen die Entwickler und Anwender darauf achten, dass auf dem FPGA Board existierende Schutzmechanismen durch unsachgemäße Anwendung oder Überlastung nicht überfordert werden.

Da im Roboterarm eingebaute Servomotoren einerseits eine höhere Strom- und Spannungsversorgung brauchen, als was die Xilinx FPGA-Board liefern kann, andererseits um das Board vor Überspannung und anderen Seiteneffekten zu schützen, ist noch ein zusätzliches Bauelement notwendig, das diese Aufgaben übernimmt.

In diesem Kapitel werden die Rolle und die Realisierung des Signalverstärkers beschrieben, der gleichzeitig als ein schützender Puffer zwischen den einzelnen Servomotoren und dem FPGA-Board geschaltet wird.

6.1 **Schutz**

Obwohl die modernen FPGA-Boards bereits Schutzmechanismen an die I/O Blöcke implementiert haben, ist dennoch wichtig und erforderlich, zusätzliche Maßnahmen zu ergreifen, um die teure Hardware zu schützen. Besonders wenn Elektromotoren eingesetzt werden, sind Schutzmaßnahmen unerlässlich. Bei Beschädigung oder durch Überlastung können die Elektromotoren hohe elektrische Spannung induzieren und die FPGA Boards beschädigen. Hinzukommt noch, dass die Elektromotoren während der Abbremsphase teilweise als Generatoren wirken und so unerwünschte Effekte verursachen können.

Beim Entwerfen von Schaltkreisen ist außerdem noch zu beachten, dass alle Signalleitungen richtig abgeschlossen sind, und keine Reflexionen entstehen können.

6.1.1 FPGA I/O Reflexionsschutz

Um eine Signalleitung zu beenden, die Signale mit hoher Übertragungsrate bearbeiten muss, werden traditionsgemäß Widerstände verwendet. Reflexionen am Leitungsende können auch auftreten, wenn Endpunktwiderstände weit vom Ende der Signalleitung sitzen. Um bessere Signalintegrität zu erreichen, entwickelte Xilinx eine neue Input/Output-Technologie für Virtex-II und Virtex-II Pro Familien, den digital gesteuerten Widerstand (Digitally Controlled Impedance). DCI justiert aktiv den Widerstand der I/O-Schnittstelle, um einem externen Bezugswiderstand zu entsprechen. Dabei wird eine konstante Impedanz gewährt, die Temperatur- und Spannungsunabhängig ist. Dadurch ist eine On-Chip-Terminierung der I/O-Pins möglich. Mit DCI sind die Endpunktwiderstände so nahe wie möglich zum Ausgangstreiber oder zum Eingangspuffer, und auf dieser Weise werden die Reflexionen beseitigt. Es gibt daher kein offenes Leitungsende, an dem etwas reflektieren könnte.

6.1.2 Überspannschutz

Um das FPGA-Board zu schützen, werden die verwendeten I/O Ports mit Hilfe von Optokoppler galvanisch (elektrisch) vom Rest der Robotersteuerung getrennt. Durch die Trennung bekommt man zwei Schaltkreise, die auf unterschiedliche Messpotenziale arbeiten. Die Signalübertragung wird nur in eine Richtung ausgeführt und zwar von der FPGA-Entwicklungsboard zu Steuerboard. Innerhalb der Optokoppler werden die vom FPGA generierte Signale als Lichtimpulse im infraroten Bereich an das Steuerboard übertragen.

Für diese Diplomarbeit verwendete Optokoppler besteht aus einer Infrarot-Leuchtdiode als Sender, und aus einem NPN-Fototransistor als Empfänger. Die nachfolgende Abbildung zeigt den schematischen Aufbau und Pinnbelegung dieser Optokoppler.

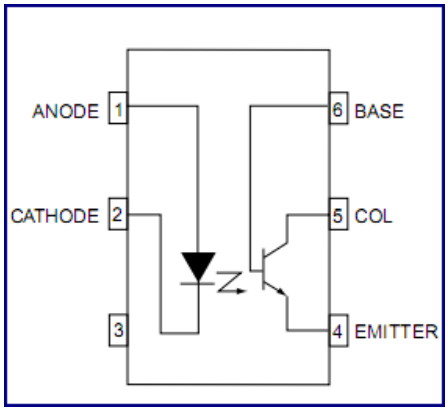

Abbildung 16: CNY 17-4 v0647K [Zeichnung aus dem Datenblatt]

Die Auswahl der richtigen Optokoppler ist bei der Signalübertragung ganz wichtig, besonders wenn steile Flanken -wie bei Rechtecksignale- vorkommen, ausgangsseitig dargestellt werden. Wegen der Infrarot-Leuchtdiode in der Optokoppler ist ein Vorwiderstand von 100 Ohm an der Anode vorzuschalten. Mit Hilfe der Optokoppler und einem nachgeschalteten Transistor erreicht man, dass FPGA-Boards mit verschiedener Ausgangsspannung verwendet werden können, ohne zusätzliche Regler zu implementieren. In diesem Projekt eingesetzte Virtex II Pro und Spartan 3A FPGAs haben 2.5 V bzw. 3.3 V Ausgangsspannung.

6.1.3 NOT-AUS Schalter

Der Roboterarm ist ein beweglicher mechanischer Teil, der in seiner Umgebung durch aufgetretene Fehlfunktion Schäden anrichten kann. Im Gefahrenfall kann die Stromversorgung für den Roboterarm durch ein NOT-AUS Schalter abgestellt werden, und ihn auf dieser Weise in einen sicheren Zustand gebracht werden.

Der Schalter ist am Signalboard angebracht, und wird im Notfall die Stromversorgung für den Signalboard und für den Roboterarm gleichzeitig abschalten. Durch seine hervorgehobene Stellung ist der NOT-AUS Schalter gut erreichbar positioniert und jederzeit leicht zu betätigen. Im eingeschalteten Zustand leuchtet ein rotes LED, (ist im Schalter eingebaut), signalisierend, dass die Stromversorgung eingeschaltet ist.

6.2 Signalboard

Der Signalboard hat eine zentrale Rolle bei der Robotersteuerung. Die Aufgabe des Signalboards ist es, die FPGA-generierten Signale zu verstärken, den FPGA galvanisch von dem Rest der Steuerung zu trennen, und die Versorgungsspannung zu regulieren.

Für die Realisierung des Signalbords sind die Bauelemente für diese speziellen Anforderungen entsprechend auszuwählen. Die gewählten Transistoren und Widerstände hatten bei Spitzenbelastung mehr als 6 A Ströme standhalten müssen. Andererseits ist es erforderlich, dass Signale bis zu 1MHz ohne signifikante Verzerrung dargestellt werden können.

6.2.1 Aufbau des Signalverstärkers

Der Signalverstärker (Impulsverstärker) verstärkt das Rechtecksignal. Dieses Bauelement wird benötigt, um die Aufrechterhaltung der Impulsqualität zu gewähren und um für die Servomotoren notwendige Signalpegel von mindestens 4,5V(TTL) zu erreichen. Der Signalverstärker wird aus dem Empfängertransistor des Optokopplers und aus einem zusätzlichen Transistor aufgebaut. Die so realisierte Transistorschaltung ist als Darlingtonschaltung bekannt.

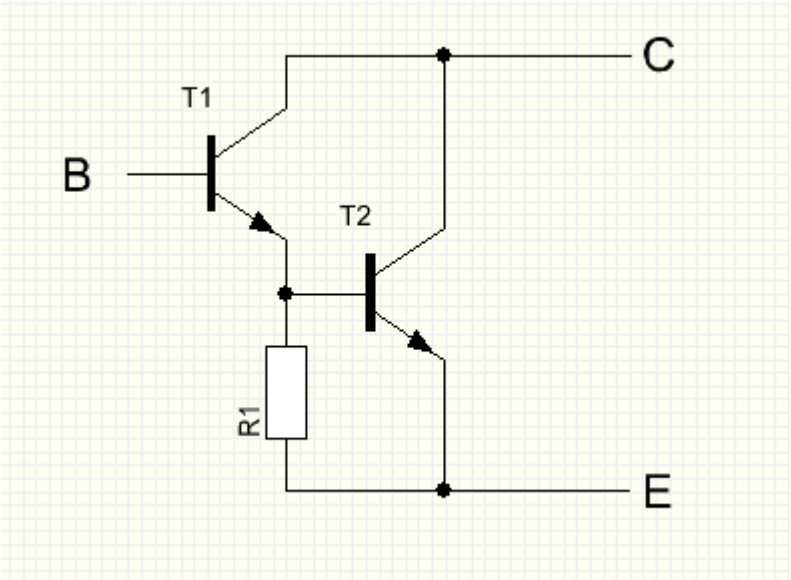

Abbildung 17: Darlingtonschaltung

Die Darlingtonschaltung ist eine Kettenschaltung aus zwei einzelnen Transistoren. Im Laststromkreis des ersten Transistors T_1 und im Arbeitsstromkreis des zweiten Transistors T_2 ist ein Widerstand R, der Einfluss auf die Stromverstärkung und das Schaltverhalten hat.

Durch die Darlingtonschaltung kann eine wesentlich höhere Stromverstärkung erreicht werden, als bei einem einzelnen Transistor. Die gesamte Verstärkung ist das Produkt einzelner Verstärkungen der Transistoren.

Formel zur Berechnung der Stromverstärkung:

$$\mathbf{B = B_{T1} * B_{T2}}$$

Vom Widerstand R (Abbildung 12) abhägig, kann man die Darlingtonschaltung für schnelle Schaltung oder für größere Verstärkung konfigurieren. Für die Schaltung in der Abbildung 13 ist wichtig, dass die Signale schnell geschaltet werden, deswegen werden Widerstände (1000 Ohm) verwendet.

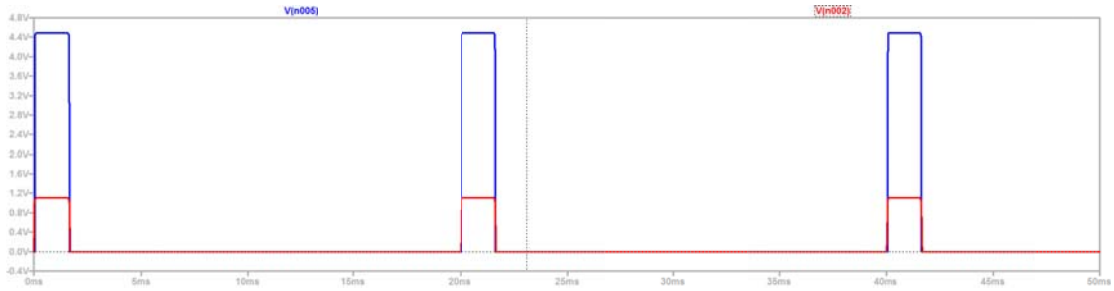

Abbildung 18: Signalverstärkung, Simulation mit Hilfe von LTSpice

Auf der oberen Abbildung werden die von FPGA erzeugten Rechtecksignale rot markiert. Die verstärkten Steuersignale, die direkt die Servomotoren des Roboterarms ansteuern, sind Blau markiert. Wenn man die Spannungen miteinander vergleicht, lässt sich eine annähernd vierfache Spannungsverstärkung feststellen.

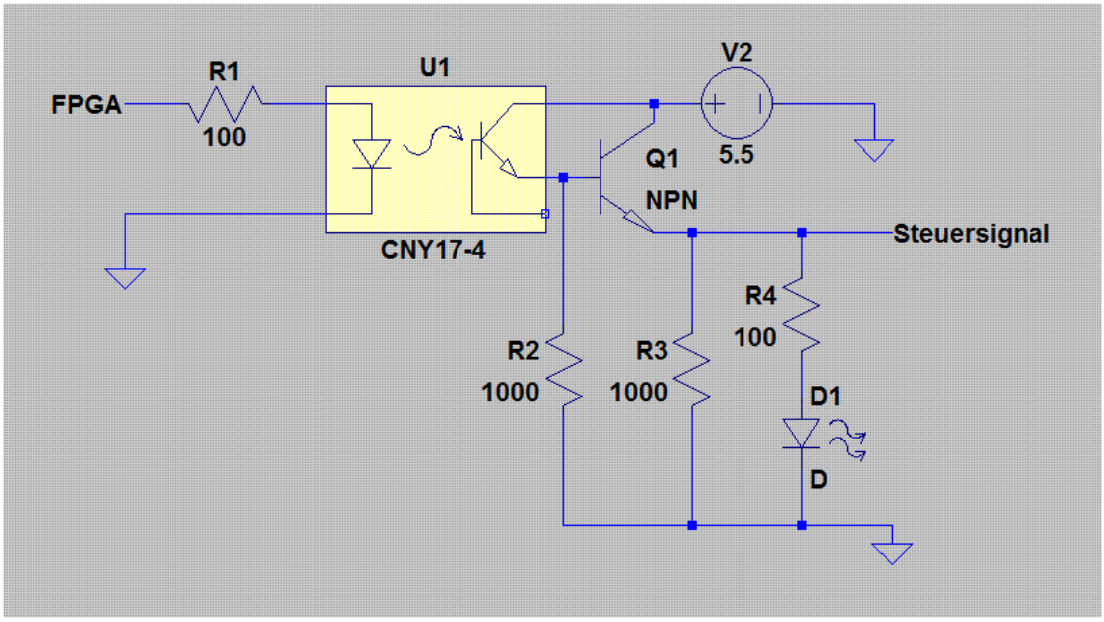

Abbildung 19: Signalverstärker

In der Abbildung 18 gezeigte Schaltung wird für jedes Signal bzw. für jedes Robotergelenk auf dem Signalboard erstellt. Insgesamt werden sieben Steuersignale verstärkt an die neun Servomotoren weitergeleitet.

In der Schaltung sind noch LEDs angebracht, die bei anlegen eines Signals aufleuchten. Diese LEDs dienen primär für die Fehlersuche. So kann man rasch feststellen, dass der FPGA Signale generiert, und an der Steckleiste die entsprechenden Steuersignale vorhanden sind.

6.2.2 Spannungsregler

Die Servomotoren des Roboterarms können mit unterschiedlicher Versorgungsspannung betrieben werden. Die eingesetzten Servomotoren arbeiten mit Versorgungsspannungen von 4.5 bis 6.0 VDC. Dabei ändern sich das Drehmoment und die Stellgeschwindigkeit der Servomotoren proportional zu der Versorgungsspannung.

Um den Roboter flexibler einsetzen zu können, wurde auf dem Signalboard ein Festspannungsregler gebaut. Mit dem Festspannungsregler kann man auf einfacher Weise die vom Roboterarm ausgeübte Kraft verkleinern oder erhöhen. Der Spannungsregler ist aus einem „starken“ MOSFET Leistungstransistor, einem Potenziometer und aus einem Kondensator aufgebaut.

Während der Roboterentwicklung wurde ein ATX-Netzteil verwendet, um den Roboter mit 5,5 VDC Versorgungsspannung und mit bis zu 6,0 A Strom zu versorgen. Zusätzlich erlaubt der Spannungsregler das Anschalten verschiedener Stromversorger. Durch das Potenziometer, das Teil des Festspannungsreglers ist, ist die gewünschte Ausgangsspannung für Stromquellen von 10 bis 24 VDC einstellbar.

6.2.3 Anschlüsse

Wie aus der Abbildung zu entnehmen ist, sind die Anschlüsse ihrer Funktionalität entsprechend in drei Reihen angeordnet. Um die Servomotoren und die FPGA-Entwicklungsboard anzuschließen, werden Stiftleisten verwendet.

Servoanschluss

Die Servomotoren sind an die drei Stiftleisten anzuschließen, die im mittleren Bereich des Signalboards angeordnet sind. Insgesamt sind sieben Anschlüsse vorhanden, die jeweils für je einen Servomotor vorgesehen sind. Der linke Stift ist für die Servosteuerung vorgesehen, an dieser Stelle wird die orangenfarbige Signalleitung des Servos angeschlossen. Der rechte Pin des Anschlusses ist der Grund, und der mittlere Pin liefert die Versorgungsspannung von 4,5 bis 6,0 VDC. Die meisten Servomotoren verwenden den gleichen Farbcode und die gleiche Anordnung für ihre Anschlüsse. Da die Stromversorgung für alle verwendeten Motoren die gleiche ist, sind die Versorgung-Pins parallel geschaltet.

Control Pins

Die Control Pins können sowohl für die Signalübertragung als auch für die Signalerfassung eingesetzt werden. Die Control Pins sind neben der SCSI Stiftleisten angebracht und mit den Signal-Pins direkt verbunden. So kann man während des Betriebes die Übertragungssignale von Virtex II Pro analysieren.

Wird für die Robotersteuerung der Spartan 3A Entwicklungsboard verwendet, werden die Signale über diese Control Pins übertragen. Die Control Pins sind paarweise angebracht, für jeden Servomotor/Signaleingang je ein Signal-Pin und ein Grund-Pin.

FPGA Signalanschluss für Virtex II Pro

Der Xilinx Virtex II Pro Entwicklungsboard wird über ein 50 Polige SCSI-Flachbandkabel an das Signalboard angeschlossen. Für die Signalausgabe wird der SystemACE MPU Anschluss verwendet.

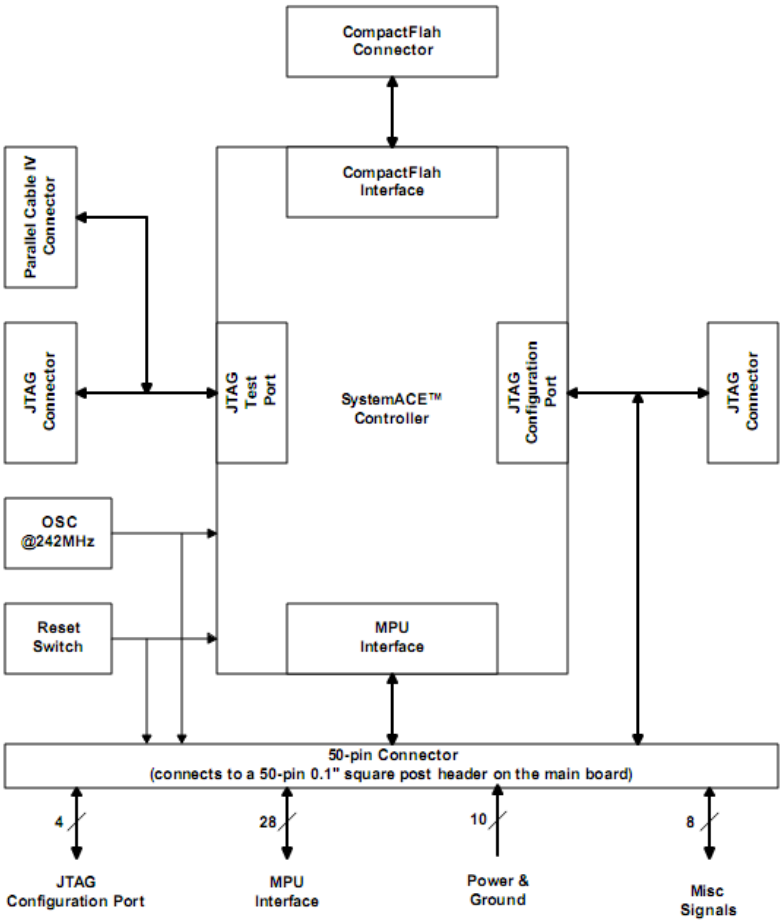

Abbildung 20: System ACE Anschluss

SystemACE (System Advanced Configuration Environment) ist ein flexibler multifunktionaler Anschluss, durch den alle Xilinx FPGA Boards konfigurierbar sind. Die Anschluss Pins können entweder einzeln angesteuert, oder externe Speichereinheiten wie Compact Flash oder Mikrodrives angeschlossen werden. Die Kommunikation über SystemACE wird durch einen Controller koordiniert. Die Abbildung 13 stellt die SystemACE Architektur dar.

6.3 Signalübertragung

Während der Entwurfsphase des Signalboards wurden Versuche an unterschiedlichen Entwürfen durchgeführt, um die Richtigkeit der Signale zu überprüfen und zu optimieren. Die erzeugten und übertragenen Signale wurden anschließend mit einem Speicheroszilloskop erfasst und analysiert.

So hat man nicht nur die Fehler in der Signalschaltung erkennen können, sondern wurden Fehler wie schlechte Lötstellen, die als kleine Resonatoren sich bemerkbar machten, ebenfalls entdeckt.

Um die Richtigkeit der Schaltung zu testen, bevor man die Komponente zusammenlötet, wurde das Programm „LT Spice“ von der Linear Technology Corporation für die Simulation von Schaltungen eingesetzt.

6.3.1 Optische Komponente

Optokoppler eignen sich nur sehr beschränkt für Signalübertragung. Bei Optokoppler ist die Übertragung von Signalen mit statischer, oder mit niedriger Frequenz unproblematisch. Bei höheren Frequenzen können Optokoppler wegen der Trägheit der optischen Komponenten, keine Signale mehr übertragen.

Um Servomotoren mit einer Auflösung von 8 bit über ein Optokoppler ansteuern zu können, muss die gewählte Komponente mindestens 256 Signale in 1 ms übertragen können. Da die steigende und fallende Flanken bei einer höheren Übertragungsfrequenz steiler ausfallen, ist jedenfalls ein Optokoppler zu wählen, der eine Signalübertragungsrate von mindestens 1 MHz hat. Für dieses Projekt eingesetzter CNY 17-4 Optokoppler können Signale mit einer Frequenz von 1 MHz und höher übermitteln, und ist damit geeignet für die Übertragung von generierten Steuersignalen.

6.3.2 TTL-Schaltung

Mit Hilfe von Transistor-Kettenschaltungen lassen sich große Lastströme mit kleinen Strömen schalten. Bei fallenden Signalflanken muss man gewisse Eigenheiten berücksichtigen. Durch fehlende, oder zu groß proportionierte Emitterwiderstand, schalten die Transistoren langsam ab. Eine kurze Abschaltdauer wird nur durch einen kleinen Widerstand am Emitter erreicht. Doch dadurch verringert sich auch die Signalverstärkung.

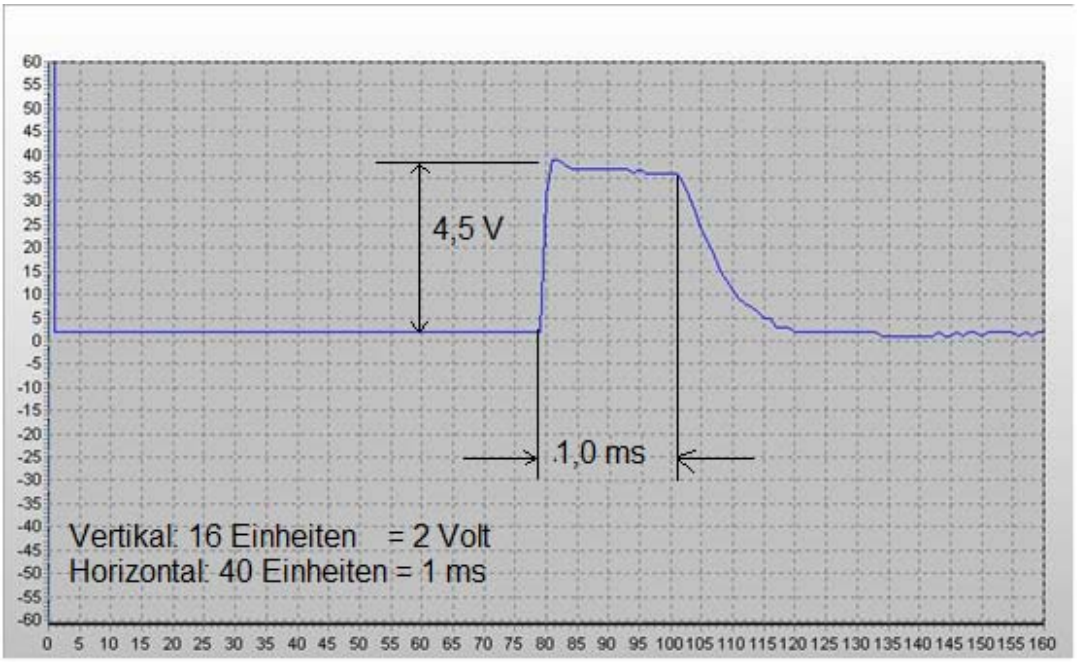

Abbildung 21: Übertragene Steuersignal ohne Emitterwiderstand.

In die auf Abbildung 13 dargestellte Schaltung wurde in der Entwicklungsphase des Signalboards der Widerstand R2 weggelassen. Bei de FPGA Signaleingang wurde ein 1,0 ms weites Rechtecksignal mit einer Wiederholfrequenz von 20 ms angelegt. Das entstandene Steuersignal wurde mit einem Speicheroszilloskop aufgezeichnet. Die Messkurve des Steuersignals ist auf Abbildung 17 dargestellt. Man sieht, dass die Signalamplitude die gewünschte Höhe von 4,5V erreicht hat, aber die abfallende Flanke sehr lange braucht, fast 1 ms, bis sie den Grundspannungspegel erreich hat.

Das erzeugte Steuersignal ist für die Ansteuerung von Servomotoren ungeeignet. Um geeignetes Steuersignal zu erzeugen, wurde der Emitterwiderstand R2 in die Schaltung eingefügt, um die absteigende Signalflanke rascher herunterzuziehen. Bei einer Widerstandgröße weniger als 500 Ohm, muss man mit starken Verstärkungseinbußen rechnen. Bei Widerständen über 10KOhm entstehen wiederum nur verzerrte Rechtecksignale, weil der Transistor erst dann abschaltet, wenn die Ladung der Basis über den Widerstand R2 abgeflossen ist.

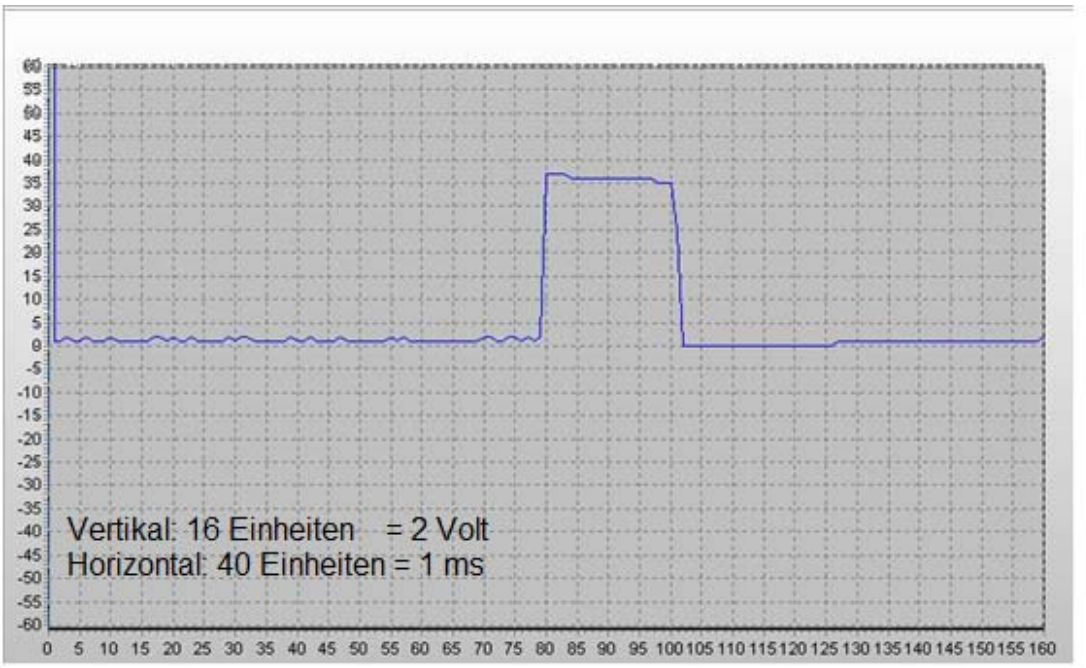

Abbildung 22: Steuersignal nach erweiterter Schaltung.

Die Rastereinheiten auf den Abbildungen 17 und 18 sind vom verwendeten Oszilloskop zugeordnet, und entsprechen folgender Einteilung:

- Vertikal: 16 Einheiten = 2 Volt
- Horizontal: 40 Einheiten = 1 ms

6.3.3 Mantelwellenfilter

Da entlang des Roboterarms die Signal- und Versorgungsleitungen für die Servomotoren verlegt sind, können einige Kabel mehr als ein Meter länge erreichen. Die langen Kabel wirken wie eine Antenne und verursachen Störungen, die sich entlang des Kabels fortpflanzen. Um Störungen an den Servoleitungen zu vermeiden, wurde ein Ringkern aus Ferrit unmittelbar nach dem Signalboard eingelegt. Zusätzlich wurden für die längeren Servoleitungen nur verdrillte Leitungen verwendet.

Mantelwellenfilter werden verwendet, um hochfrequente Störungen an den angeschlossenen Kabeln zu dämpfen. Dazu werden elektrisch passive Komponente wie Ferritkern oder Ferritringe verwendet. Ein Ringkern aus Ferrit bildet zusammen mit der herumgewickelten Servoleitung eine Spule, die durch den erzeugten frequenzabhängigen Blindwiderstand die Signalstörungen dämpft.

Kapitel 7 **Der Job Scheduler**

In diesem Kapitel wird eine konkrete Lösung vorgestellt, wie man in diesem Projekt konkurrierende Aufgaben zur Steuerung, Simulation und Synthese empfangen und abarbeiten kann. Der Job Scheduler wird vorgestellt, der all diese Aufgaben mit Hilfe einer Internetverbindung ortsunabhängig und quasi in „Echtzeit“ ausführen kann.

Der hier vorgestellte Job Scheduler ist für diese Diplomarbeit, speziell für die Robotersteuerung entworfen und realisiert.

7.1 **Aufgaben und Anforderungen**

Der Job Scheduler ist das Bindeglied zwischen den einzelnen Softwarekomponenten, und erfüllt im Projekt Robot4Web eine zentrale Steuerungsaufgabe. Bei der Aufgabenstellung wurde besonders die Anforderung eines Labors berücksichtigt, wo mehrere Personen gleichzeitig Simulationen und/oder Hardware-Synthese durchführen können. Weiterhin sollen mehrere Job Schedulers nebeneinander ausgeführt werden können, ohne dass sie sich gegenseitig stören. Der Vorteil vom Einsatz mehrerer Job Scheduler ist, dass man Benutzer und Aufgaben besser an die ausführenden Servern binden kann, und es ermöglicht die Verwendung mehrerer, für spezielle Aufgaben eingerichteter Rechnern. Job Starts erfolgen durch Verzeichnisüberwachung.

Grundsätzlich ist es möglich mehrere Job Schedulers so anzuordnen, dass die Ausgabe eines Job Schedulers gleichzeitig die Eingabe für den nächsten Job Scheduler wird. Jobs können in Job-Ketten organisiert werden, um die Reihenfolge der Abarbeitung sicherzustellen, und zur Skalierung in mehreren Instanzen ausgeführt werden.

7.1.1 Ereignisse in den NTFS Dateisystem

Um die anliegenden Aufgaben vom Job Scheduler zu erfassen, und ohne weitere Zeitverzögerung die entsprechende Ausführung der Programme auf dem vorgegebenen Rechner zu initialisieren, werden die Aufgaben zunächst als Ereignisse im zugrunde liegenden Dateisystem interpretiert.

Das NTFS (New Technology File System) ist das Dateisystem von Windows NT und seiner Nachfolger. Ab der Version NTFS 3.0 (Windows 2000) ist es möglich, Änderungen des Dateisystems in Win32 API bereitgestellten Mechanismen zu verfolgen.

Der wichtigste Bereich im Job Scheduler ist die Überwachung von Verzeichnissen bezüglich Änderungen. Bevor man mit der Realisierung des Überwachens von Verzeichnissen anfängt, ist es notwendig klarzustellen, welche Informationen man gerne erhalten möchte, und wie diese dann weiterverarbeitet werden sollen. Ein weiterer wichtiger Aspekt ist, welche Benutzerrechte der Anwender haben muss, um den gewünschten Vorgang ausführen zu dürfen.

Um die Meldungen des Filesystems zu erhalten, gibt es mehrere Möglichkeiten:

1. ReadDirectoryChangesW [5]

Nachteil: Bei großen Laufwerken oder bei Netzwerken kann sich die Leistung des Betriebssystems verringern. Ein solches Szenario wäre z. B. gegeben, wenn man gleichzeitig mit diversen Antivirenprogrammen Laufwerke überprüft. Ein weiterer Nachteil ist, dass die Änderung nur dann wahrgenommen werden kann, wenn das aufzurufende Programm läuft.

2. FindFirstChangeNotification, FindNextChangeNotification

Funktioniert ähnlich wie ReadDirectoryChangesW.

3. Change Journals

Ist die beste Methode um physische Laufwerke zu überwachen. Das NTFS Dateisystem unterstützt Change Journals ab der Windows 2000-er Version. In diesem Fall werden alle Änderungen eines Files oder Verzeichnisses in einer Datenbank aufgezeichnet. Bevor Change Journal verwendet werden kann, es ist erforderlich, dass er erzeugt und gestartet wird, da er nämlich nicht automatisch zur Verfügung steht. Weiters ist es nicht notwendig, ein Programm laufen zu lassen, um diese Änderungen aufzuzeichnen. Change Journals wird eingesetzt, um Dateien oder Filesysteme im Falle eines Computerfehlers, oder bei Laufwerksproblemen wiederherzustellen.

Der Nachteil von Change Journals ist, dass man damit nur unter Administratorberechtigungen arbeiten kann; es könnte zu Beeinträchtigung der Netzwerksicherheit führen, wenn man Anwendern diese Berechtigungen vergibt[1].

7.1.2 ReadDirectoryChangesW API

Um das Programm vielseitiger einsetzen zu können, wird im Job Scheduler ReadDirectoryChangesW API verwendet. Besonders interessante Einsatzmöglichkeiten ergeben sich während der Automatisierung von Robot4Web-Workflows.

Bei der Verwendung von ReadDirectoryChangesW ist es notwendig, zuerst einen „Handler“ des überwachten Verzeichnisses zu bekommen. Das wird mit der CreateFile-Funktion erreicht, wenn man sie mit dem Parameter „FILE_FLAG_BACKUP_SEMANTICS“ aufruft.

Für die asynchrone Anwendung wird zusätzlich das „FILE_FLAG_OVERLAPPED“ Attribut angegeben.

Der nachfolgende Codeausschnitt zeigt, wie man CreateFile in C++ aufgerufen wird, um für ReadDirectoryChangesW einen geeigneten Handler für die Verzeichnisüberwachung zu bekommen.

```
HANDLE hDir = CreateFile( strDirToWatch,
                          FILE_LIST_DIRECTORY,
                          FILE_SHARE_READ|FILE_SHARE_WRITE,
                          NULL,
                          OPEN_EXISTING,
                          FILE_FLAG_BACKUP_SEMANTICS|
                          FILE_FLAG_OVERLAPPED,
                          NULL
                        );
```

Bei dem Aufruf von ReadDirectoryChangesW wird angegeben, welches Verzeichnis überwacht werden soll, ob die Unterverzeichnisse ebenfalls einbezogen werden, und welche Ereignisse eine Reaktion auslösen soll.

```
BOOL bRes = ReadDirectoryChangesW(hD ir, szBuff,
                        sizeof(szBuff), TRUE,
                        FILE_NOTIFY_CHANGE_LAST_WRITE |
                        FILE_NOTIFY_CHANGE_DIR_NAME |
                        FILE_NOTIFY_CHANGE_FILE_NAME |
                        FILE_NOTIFY_CHANGE_SIZE,
                        &dwBytesWritten, NULL, NULL );
```

Die Einstellungen sind dann für jedes Verzeichnis bis auf die Laufwerksbuchstaben identisch, und können in der Konfiguration jederzeit während der Ausführung geändert werden.

.

7.1.3 Threadingmodelle

Die drei grundlegenden Threadingmodelle in der Win32-Umgebung lauten wie folgt: Single-, Apartment- und Free-Threading.

Single-Threading
Der einzelne Thread entspricht dem Anwendungsprozess. Ein Prozess kann als die Instanz einer Anwendung definiert werden, deren Eigentum der Speicherplatz dieser Anwendung ist. Die meisten Windows-Anwendungen sind Single-Thread-Anwendungen, und ein einzelner Thread übernimmt die gesamte Arbeit.

Apartment-Threading
Im Apartment-Threadingmodell operieren alle Threads innerhalb ihrer jeweiligen Unter-

bereiche des Hauptanwendungsspeichers. Dieses Modell ermöglicht die simultane, aber unabhängige Ausführung mehrerer Codeinstanzen.

Free-Threading
Free-Threading ist das Threadingmodell mit der größten Komplexität. Im Free-Threadingmodell können mehrere Threads gleichzeitig dieselben Methoden und Komponenten aufrufen. Anders als beim Apartment-Threading ist man dabei nicht auf separate Speicherbereiche beschränkt[2].

Der Job Scheduler verwendet das Free-Threadingmodell. Dieses Modell ermöglicht eine kurze Reaktionszeit des Programms auf die Änderungen im File-System und verbraucht nicht die ganze Prozessorleistung bei Belastung.

7.2 Funktionsweise und Abläufe

Der Job Scheduler ist eine Multithread-Anwendung, und kann mehrere, gleichzeitig aufgetretene Ereignisse behandeln. Eine neue Simulationsaufgabe oder ein neuer Positionierungsbefehl für den Roboterarm sind solche Ereignisse, die quasi parallel ablaufen können.

7.2.1 Die grafische Oberfläche

Über eine grafische Oberfläche kann man den Job Scheduler konfigurieren und die laufenden Aufgaben überwachen. Die grafische Oberfläche besteht aus drei Teilen: Konfigurationsteil, Überwachungsteil und Steuerungsteil.

Im Konfigurationsteil der grafischen Oberfläche werden die zu überwachenden Verzeichnisse eingegeben. Das Programm startet für jedes Verzeichnis, welches überwacht werden soll, einen Thread. Standardmäßig werden zwei Verzeichnisse konfiguriert. Das erste Verzeichnis ist für die vom Anwender hochgeladene Dateien vorgesehen und wird als „Event directory“ bezeichnet. Für die direkte Robotersteuerung wird der Dateipfad, wo die Steuerbefehle über den Webserver gespeichert werden, angegeben. Der Job Scheduler extrahiert aus dem angegebenen Dateipfad das zu überwachende Verzeichnis heraus und Speichert dieses im Register. Der Zweite Eintrag wird als Web-File benannt. Neben den Textboxen sind für die Funktionalitäten entsprechende Buttons angeordnet. Mit Hilfe von „Event directory“ Button kann man nur Verzeichnisse auswählen. Über das „Web-File“ wird dann der r4w Datei Ausgewählt und in die nebenstehende Textbox eingetragen.

In dem Überwachungsteil werden alle anstehenden Aufgaben untereinander in der eintreffenden Reihenfolge dargestellt. Das oberste Listenelement ist die älteste Eintragung, und wird gerade vom Job Scheduler ausgeführt. Sobald eine Aufgabe erfolgreich abgearbeitet wurde, wird sie aus der Liste der anstehenden Jobs entfernt. Ziel der Auflistung von nicht ausgeführten Aufgaben ist es: die Kommunikation auf mögliche Verbindungsfehler zu über-

wachen und Aktivitäten zu dokumentieren. Die dargestellte Liste entspricht inhaltlich dem Queue Inhalt.

Der dritte Bereich im Job Scheduler bilden die Steuerung- und Hilfe - Buttons. Der Job Scheduler ist nach dem Start sofort aktiv, ohne dass man etwas einstellen muss. Der OK Button erzwingt die Ereignisüberwachung mit den veränderten Einstellungen.

Das Programm terminiert bei der Aktivierung vom „Beenden“ Button, dabei werden alle Threads angehalten, und anschließend keine neuen Aufgaben mehr entgegengenommen. Die laufenden Ausführungen werden dabei nicht beeinträchtigt, da diese Stapelverarbeitungsprozesse sind, die vom Job Scheduler nur angestoßen wurden.

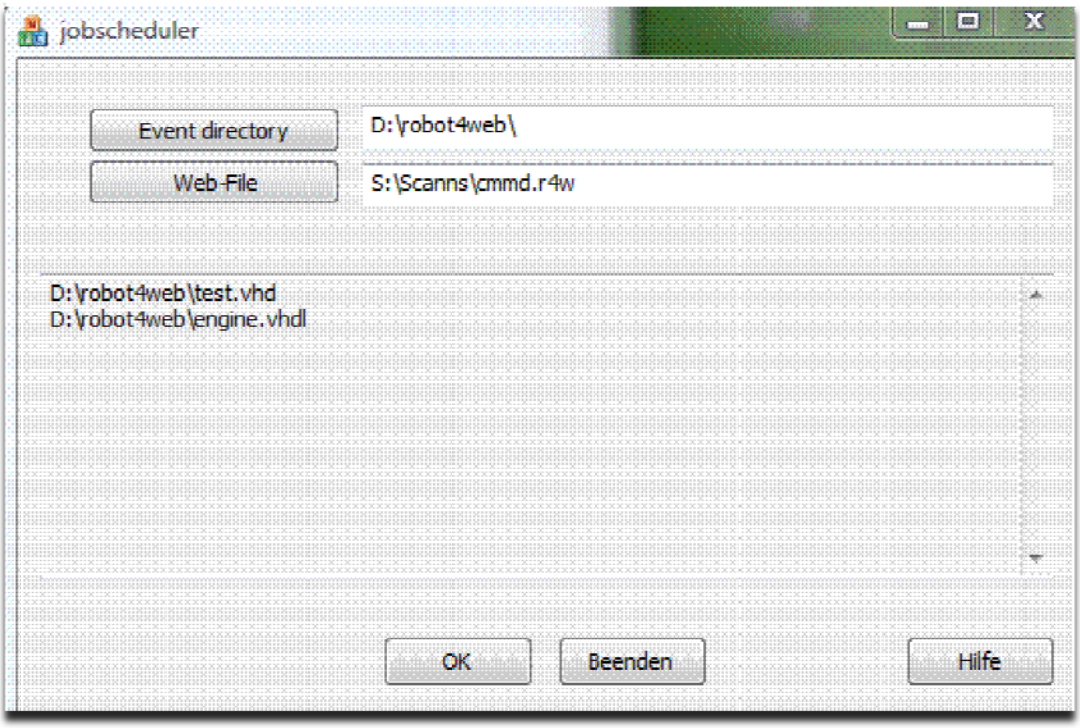

Abbildung 23: Die grafische Oberfläche vom Job Scheduler .

Die Hilfe und die Bedienungsanleitung zum Job Scheduler erscheinen nach Aktivierung des Hilfe Buttons. Diese Dokumente beinhalten einige wichtige Punkte, bezüglich Steuerung und Konfiguration von der Job Scheduler.

7.2.2 Konfiguration

Job Scheduler verwendet für die Speicherung von den Konfigurationsdaten das Register des Computers, wo der Job Scheduler läuft. Die aktuellen Informationen, die für die Verzeichnis-überwachung notwendig sind, werden in globalen Variablen abgespeichert. Da alle laufende Threads auf diese globale Variable zugreifen können, lässt sich die Pfadübergabe sehr einfach gestalten. Die globalen Variablen für die Pfadinformationen sind in die CjobschedulerApp Klasse als CString Type deklariert.

Konfiguration einlesen

Die Konfigurationsdaten werden beim Starten vom Job Scheduler in OnInitDialog() Funktion aus den Registern eingelesen und die globale Variablen Watch_Dir, Watch_Dir_Path und Web_File mit diesen Werten initialisiert.

CWinApp* pApp = AfxGetApp();

Von AfxGetApp() Funktion zurückgegebene Zeiger wird verwendet werden, um Anwendungsinformationen zugänglich zu machen. Der GetProfileString member Funktion wird eingesetzt, um einen String - Wert aus der Anwendungsregister zu lesen. Von GetProfileString verwendete Parameter geben an, in welcher Sektion der Eintrag befindet, den Eintragsnamen, (darf nicht NULL sein), und den voreingestellten Eintragswert. Die letzte Angabe ist die Vorgabe, falls der genannte Eintrag noch nicht existiert. In diesem Fall wird sie auf _T(" ") gesetzt, wie aus dem nachfolgender Listing zu erkennen ist.

theApp.Watch_Dir = pApp->GetProfileString(_T("Jobscheduler"), _T("Watch_Dir"), _T(" "));

theApp.Watch_Dir_Pat =pApp->GetProfileString(_T("Jobscheduler"), T("Watch_Dir_Path"), _T(" "));

theApp.Web_File = pApp->GetProfileString(_T("Jobscheduler"), _T("Web_FILE"),

_T(" "));

Konfiguration in die Registry schreiben

Die zu speichernden Konfigurationsdaten sind in den globalen Variablen hinterlegte Werte. Die Konfigurationsdaten werden im Register HKEY_CURRENT_USER unter Jobscheduler abgespeichert und in weiteren Schritten ausgelesen.

Zum Schreiben von Daten in die Registry, wird der WriteProfileString member Funktion verwendet. Der erste Parameter gibt den auszulesenden Registry-Sektor an, danach folgt der Name des Eintrags und letztendlich der Wert des Eintrags. Der Schlüssel, der angelegt werden soll, wird unter der Sektion „Software“ eingetragen.

```
// Konfiguration speichern
pApp->WriteProfileString(_T("Jobscheduler"), _T("Watch_Dir"),  theApp.Watch_Dir);
pApp->WriteProfileString(_T("Jobscheduler"), _T("Watch_Dir_Path"),  theApp.Watch_Dir);
pApp->WriteProfileString(_T("Jobscheduler"), _T("Web_FILE"), m_strDateiname);
```

In diesem Beispiel wird auf das Register eines Computers geschrieben, wo der Job Scheduler gestartet wurde.

7.2.3 Petri-Netz-Modell

Petri-Netze erlauben die Modellierung von parallelen Systemen, und ermöglichen dadurch die Analyse der Struktur und der dynamischen Abläufe. Ein Petri-Netz besteht aus vier Elementen: Stellen, Transitionen, Input-Funktion und Output-Funktion. Dabei stellt ein Petri-Netz einen endlichen, gerichteten Graph mit zwei Arten von Knoten - den Stellen und Transitionen –dar. [7]

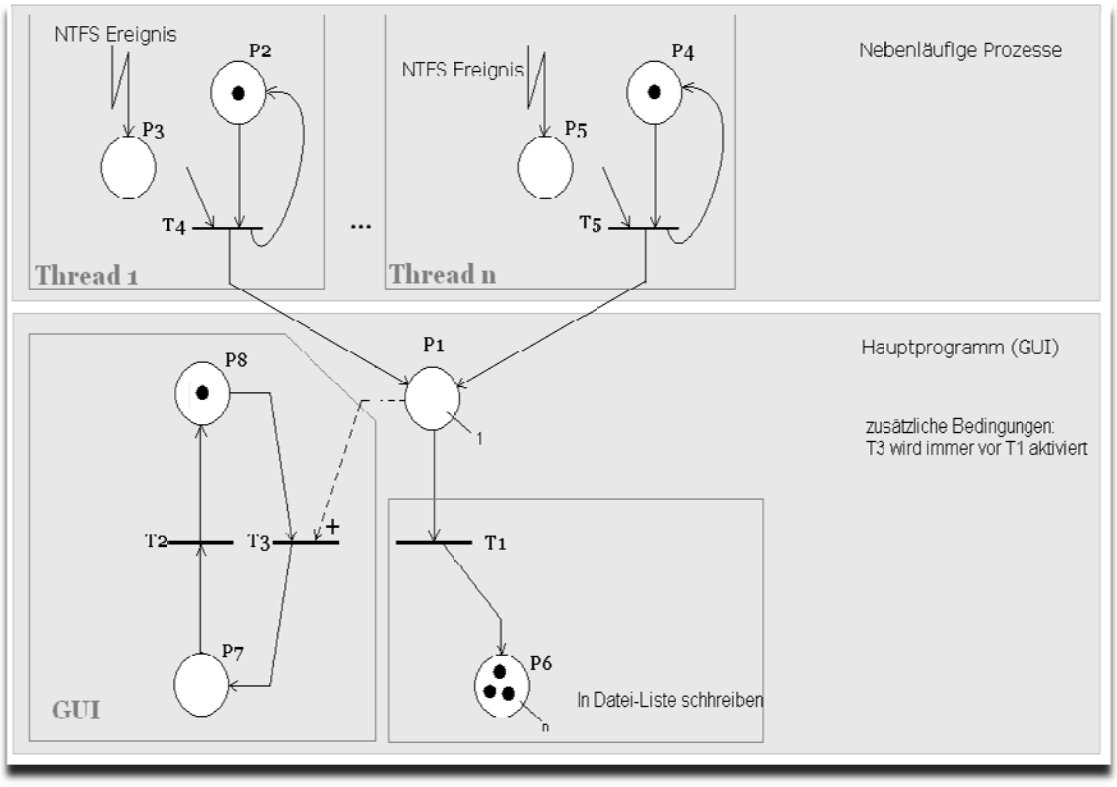

Abbildung 24: Petri-Netz-Modell vom Job Scheduler

Die auf Abbildung 3 dargestellten Transitionen[7] T4 und T5 erzeugen bei der Aktivierung ein Ereignis und signalisieren damit dem Hauptprogramm, dass die neuen Informationen in

die Liste einzutragen sind, und die Darstellung aufgefrischt werden soll. P2 und P4 sind Systemzustände[7] wo die Threads 1 bis n auf Ereignisse in dem Filesystem warten.

Die Systemzustände P7 und P8 bzw. die Transitionen T2 und T3 modellieren die grafische Oberfläche vom Job Scheduler. Für die Transition T3 wird die Schaltbedingung erfüllt und feuert, sobald der Systemzustand P1 eine Marke erhält, also wenn eine, von den nebenläufigen Prozessen erzeugte Meldung in Form von Postmessage eingetroffen ist. Die Stellenkapazität für P1 ist auf eine Markierung begrenzt, da die einkommenden Ereignisse nur sequenziell abgearbeitet werden können.

Um die Übersichtlichkeit zu bewahren, wurde in dem oben vorgestellten Petri-Netz-Modell nicht alle Komponente des Job Schedulers dargestellt. Das Abarbeiten in der Queue gespeicherten Informationen wurde aus dem obigen Modell weggelassen.

7.2.4 Realisierung

Es gibt eine Vielzahl von Konfigurationsmöglichkeiten, wie man den Job Scheduler im Projekt Robot4Web einsetzen kann. Von Einrechnerlösung bis auf mehrere Netzwerke verteilte Anwendung, kann man das Projekt Robot4Web so konfigurieren, dass man keine zusätzlichen Installationen ausführen muss.

Eine sehr übersichtliche Anwendung ergibt sich, wenn man den, zu überwachenden Verzeichnissen Laufwerksbuchstaben zuordnet. Welche Laufwerke oder Verzeichnisse gerade überwacht werden, kann man in der Konfiguration aus einer Liste der vorhandenen Laufwerken entnehmen. Bei den ausgewählten Laufwerken handelt es sich nicht nur um physische Laufwerke, es wird jede Art von Laufwerken überwacht, welche einen Laufwerksbuchstaben enthalten und welche vom Betriebssystem auch als solche behandelt werden. Weiterhin kann man zusätzliche Laufwerke definieren und überwachen lassen. Ein Neustart des Programms ist dabei nicht erforderlich, es wird automatisch ein neuer Thread für dieses Laufwerk gestartet, sobald Ereignisse auf dem neuen Laufwerk überwacht werden.

Bei der Veränderung der Konfiguration oder bei Änderung der auf Ereignisse überwachten Verzeichnisse, müssen die laufenden Prozesse terminiert, und bei der nächsten anstehenden Aufgabe die neue Einstellungen berücksichtigt werden. Dazu werden die laufenden Threads angehalten und Threads mit den neuen Einstellungen gestartet. In diesem Fall könnte es vorkommen, dass einige Änderungen (während die Threads angehalten sind) nicht verfolgt werden. Um diesen Informationsverlust zu vermeiden, wird der alte Thread möglichst sicher unter der Berücksichtigung der aktuellen Zustände beendet, dabei werden Mechanismen wie ExitThread nicht verwendet. ExitThread kann zu unerwünschten Nebeneffekten führen, da in diesem Fall der Thread beendet wird, bevor man Destruktoren oder „cleanup“ Funktionen ausführen kann.

Alle Überwachungsfunktionen werden in parallel laufenden Thread Prozesse ausgeführt, sonst würden sich diese gegenseitig blockieren. Der Job Scheduler besteht aus mehreren,

miteinander kommunizierenden Prozessen, die alle zu Beginn der Programmausführung parallel gestartet oder an einem späteren Zeitpunkt erzeugt werden. All diese Prozesse arbeiten dann gemeinsam an einem Problem und nutzen zum Datenaustausch Nachrichten, welche von einem Prozess zum anderen geschickt werden.

Bei einer Änderung der zu überwachenden Verzeichnisse durch den Anwender, wird zuerst ein neuer Thread für die neue Einstellungen gestartet, und gleichzeitig die globalen Variablen für die Beendigung des alten Threads gesetzt. Der alte Thread bleibt noch solange aktiv, bis die für diese Thread gesetzten globalen Variablen abgearbeitet sind, und der Thread damit terminieren kann.

Jedes Mal, wenn eine Änderung des Filesystems gemeldet wurde, werden die Datei- oder Verzeichnisnamen von den Dateien oder den Verzeichnissen, die die Änderungen hervorgerufen haben, mit dem aktuellen Zeitstempel in eine Queue abgespeichert. Bevor die neuen Elemente in Queue eingetragen werden, wird überprüft, ob gleichnamige Einträge bereits vorhanden sind. So wird sichergestellt, dass mehrere Änderungen von demselben Objekt nicht in der Queue aufgenommen werden können. Doppelte Einträge können in der Queue nicht genau zu einzelnen Ereignissen zugeordnet werden, deswegen müssen diese vermieden werden. Doppelte Einträge in der Queue könnten auf Datenänderungen im überwachten Verzeichnis während der Aufgabenabarbeitung, oder auch auf mehrmaliges Hochladen von Dateien während der Verarbeitung hinweisen.

Nach einem Queue-Eintrag im Thread-Prozess, werden die nachfolgenden Prozeduren aufgerufen, die für die Abarbeitung von den Queue-Einträgen zuständig sind. Um diese Prozesse zu aktivieren, sendet der Thread „Track Changes“ nach erfolgter Eintragung in die Queue den Message: WM_MYMESSAGE_QUEUE_READY aus. Diese ist wiederum im Message Map Bereich zu dem „`OnMessageCalculation`“ member Funktion zugeordnet.

Die nachfolgende Codesequenz zeigt die algorithmische Implementierung von der Nachricht WM_MYMESSAGE_QUEUE_READY:

…

```
ON_MESSAGE(WM_MYMESSAGE_QUEUE_READY,OnMessageCalculation)
```

….

```
LRESULT CjobschedulerDlg::OnMessageCalculation(WPARAM wParam, LPARAM
lParam)
{
	// Ausgabe neu einlesen.
	RenewDisplay();

	// Wird gerade keine Berechnung ausgeführt, dann wird ein neuer
Berechnungsthread gestartet.
	// Die globale Variable "Evaluating" stellt sicher, dass die
Aufgaben itarativ ausgeführt werden.
	if(!theApp.Evaluating)
	{
```

```
            theApp.Evaluating = true;
            DWORD dwThreadId;
            hthread =  CreateThread(NULL, 0,  &Evaluating, 0,  0,
&dwThreadId );  //Einen neuen Berechnungsthread starten.
      }
      return 0;
}
```

Wie man aus der obigen Codesequenz entnehmen konnte, wird in der Funktion OnMessageCalculation, bei bedarf ein neuer Tread-Prozess initialisiert und gestartet. Der Thread-Prozess: Evaluating wird dann die Queue-Einträge sukzessive abarbeiten. Dazu wird die zu bearbeitende Aufgabe, anhand von in Queue eingetragenen Dateinamen identifiziert, und in zwei Kategorien aufgeteilt. Zu jede Arbeitskategorie ist eine Batch-Datei zugeordnet, die im Ausführungsverzeichnis des Job Schedulers abgelegt werden. Die Batchdatei R4W_RS232.BAT ist für die direkte Robotersteuerung über die Webschnittstelle zuständig, die zweite Batchdatei R4W_SimSin.BAT für die Simulation und Hardwaresynthese. Während der Simulation und Synthese werden kontinuierlich Zwischenergebnisse und Meldungen über die Webschnittstelle an die Benutzer weitergeleitet.

Die tatsächliche Aufgabenausführung startet mit der Aufruf von Batchdateien.

7.2.5 Direkte Robotersteuerung über die RS-232 Schnittstelle

Die Aufgaben des Job Schedulers beinhalten neben der Aufgabenverteilung, wie Simulation oder Synthese, auch die direkte Ansteuerung des Roboterarms. Die Elektromotoren des Roboterarms kommunizieren mit dem Steuerungscomputer über einen vorgegebenen Port. Normalerweise wird der serielle Port COM1 verwendet. Da COM1 mit anderen Anwendungen belegt sein kann, ist es vorgesehen, dass man den Informationsaustausch über einen anderen seriellen Port ausführen kann. Für die Kommunikation vorgesehene Port-Nummer kann nur der Administrator eingeben oder verändern, diese Einschränkung ist für die Netzwerksicherheit unerlässlich. Der richtige COM-Port der seriellen Kommunikationsschnittstelle wird in der Batchdatei R4W_RS232.BAT bestimmt. Die Baudrate ist auf den Wert: 9600 festgesetzt und ist im Programm RS232.EXE hardkoded. Es ist notwendig einen festen Wert für die Baudrate anzugeben, weil sowohl beim Sender als auch bei dem Empfänger die gleiche Einstellungen verwendet werden müssen.

Für die direkte Robotersteuerung ist ein Webformular vorgesehen, über dem der Anwender für die entsprechenden Motoren, die neue Position und die Anfahrtsgeschwindigkeit bestimmen kann. Die gleichen Aufgaben lassen sich ebenfalls ausführen, wenn man das Programm RS232.EXE auf Kommandozeilenebene auf dem Rechner mit direkter Verbindung zu der FPGA-Board aufruft. Der Unterschied zwischen den beiden Methoden ist, dass bei einer Steuerung über die Webschnittstelle der Übergabeparameter für das Programm

RS232.EXE das File „CMMD.R4W“ ist, solange bei den Kommandozeilen Aufruf die Übergabeparametern die COM-Port, die relative Geschwindigkeit und die gewünschte Position sind.

Die nachfolgende Abbildung zeigt das für die direkte Ansteuerung des Roboterarms vorgesehene Webformular.

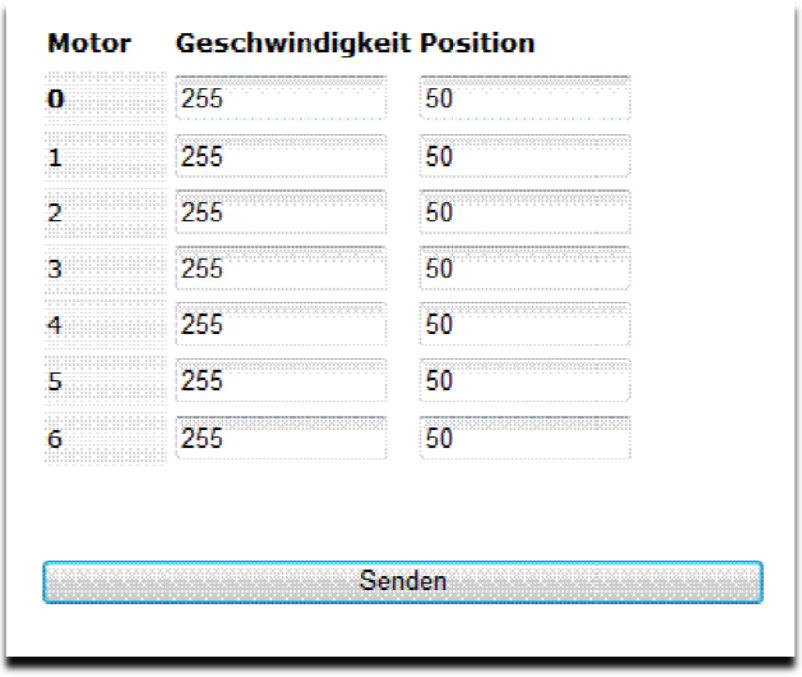

Abbildung 25: Formular für die direkte Robotersteuerung.

Bei der Positionierung des Roboterarms werden für jedes Servomotor die neue Stellposition und die relative Geschwindigkeit, mit dem die Motoren die neue Stellung anfahren werden, angegeben.

Mit dem Button „Senden“ werden die Werte aus den Formularfeldern in Datei „commd.r4w“ auf dem Webserver abgespeichert. Bei Speicherung bzw. Änderung der Datei wird durch das NTFS Dateisystem ein Ereignis erzeugt, was vom Job Scheduler sofort empfangen wird. Das Verzeichnis, wo die Datei CMMD.R4W abgespeichert ist, muss dem Job Scheduler bekannt sein, um auf jede Änderung der Datei reagieren zu können, und in der Queue der offenen Aufgaben aufzunehmen. Sobald Einträge in der Queue eingefügt wurden, startet eine neue Prozedur in einem weiteren Thread, und startet die „r4w_direct.bat“ BAT-Datei mit der „cmmd.r4w“ Datei als Übergabeparameter. Der nächste Queue Element wird erst nach Terminierung der BAT-Datei abgearbeitet. Die Benützung einer BAT-Datei ermöglicht die externe Definition der Abläufe, und erhöht die Flexibilität des Workflows.

CMMD.R4W Datei beinhaltet die Parameter, die über eine serielle Schnittstelle an der FPGA Board gesendet werden. Zeilen, die mit einem Leerzeichen beginnen, werden als Kommentarzeilen interpretiert. Da das Programm RS232.EXE nach jeder abgearbeiteten Zeile per Definition 100 ms Lang wartet, bis die nächste Zeile ausgeführt wird, kann man mit Hilfe der Kommentarzeilen die nachfolgenden Befehle zeitlich versetzt ausführen.

Bereits in die FPGA geladenes Programm beinhaltet ein Modul, das für die serielle Kommunikationsabwicklung verantwortlich ist. Dieses empfängt die gesendeten Parameter, und generiert daraus die Steuersignale für die entsprechenden Servo-Motoren.

Der nachfolgende Listing zeigt den typischen Inhalt von „CMMD.R4W“ Datei.

```
::Kommentar
1#255#100
2#255#100
4#255#100
8#255#255
16#255#50
32#255#150
64#155#255
```

Die „CMMD.R4W“ Datei wird so abgespeichert, dass für die serielle Kommunikation zuständiges Programm RS.EXE, die enthaltenen Befehle ohne Umwandlungen interpretieren kann. Unter anderem werden die Motorennummern in Form von 2er Potenz ($2^{\text{Motornummer}}$) angegeben. Jede Zeile beinhaltet die anzusprechende Motorkennung, die relative Geschwindigkeit und die anzufahrende Position. CMMD.R4W wird zeilenweise von dem Programm RS232.EXE abgearbeitet, dabei werden die Zeichen ‚#‘ als Trennzeichen interpretiert, und haben keine Bedeutung bei der Übertragung.

Die Abbildung 12 zeigt die schematischen Kommunikationswege bei der direkten Robotersteuerung. Der hellgraue Pfeil symbolisiert das Ereignis, was bei der Erstellung und Modifikation von „CMMD.R4W“ ausgelöst wird.

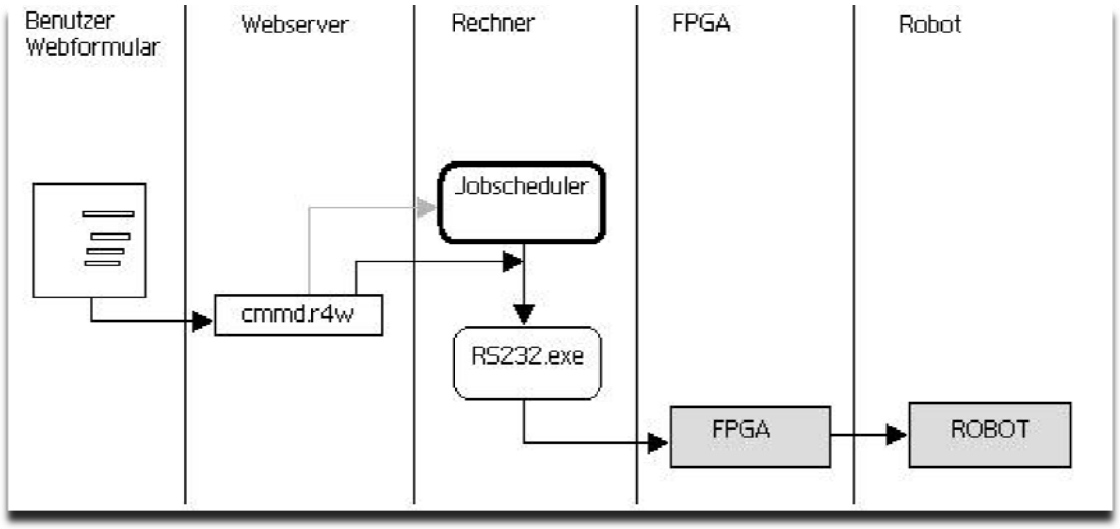

Abbildung 26: Kommunikationswege bei der direkten Robotersteuerung.

7.2.6 Aufgabensteuerung

Unter Aufgabensteuerung versteht man in diesem Zusammenhang die Erfassung und die Ausführung von anstehenden Simulations- bzw. Syntheseaufgaben und die anschließende kontinuierliche Rückgabe von Meldungen an die Benutzer.

Bei der Aufgabensteuerung werden die, in einem ZIP-File gepackte VHDL-Dateien von den Anwendern auf dem Webserver hochgeladen, anschließend simuliert und synthetisiert, dann die Ergebnisse an den Benutzern zurückgegeben. Abhängig davon, welche Aufgaben der Job Scheduler zu erledigen hat, verhält sich dieser unterschiedlich. Die verschiedenen Aufgaben werden anhand der eingetretenen Ereignisse in den überwachten Verzeichnissen erkannt.

Die Aufgaben des Job Schedulers bei der Aufgabensteuerung sind:

- Erkennen, dass Benutzer neue VHDL-Dateien auf dem Webserver hochgeladen haben.
- Erkennen ob es sich um Robotersteuerung oder um Aufgabensteuerung handelt.
- Simulation und Synthese ausführen.
- Die Rückgabemeldungen an den richtigen Benutzeraccount zurückgeben.
- Datenaustausch zwischen Webserver und Rechner koordinieren.

Die Abbildung 12 zeigt die Rolle des Job Schedulers bei Aufgabensteuerung. Die hellgrauen Pfeile zeigen die ausgelösten Ereignisse.

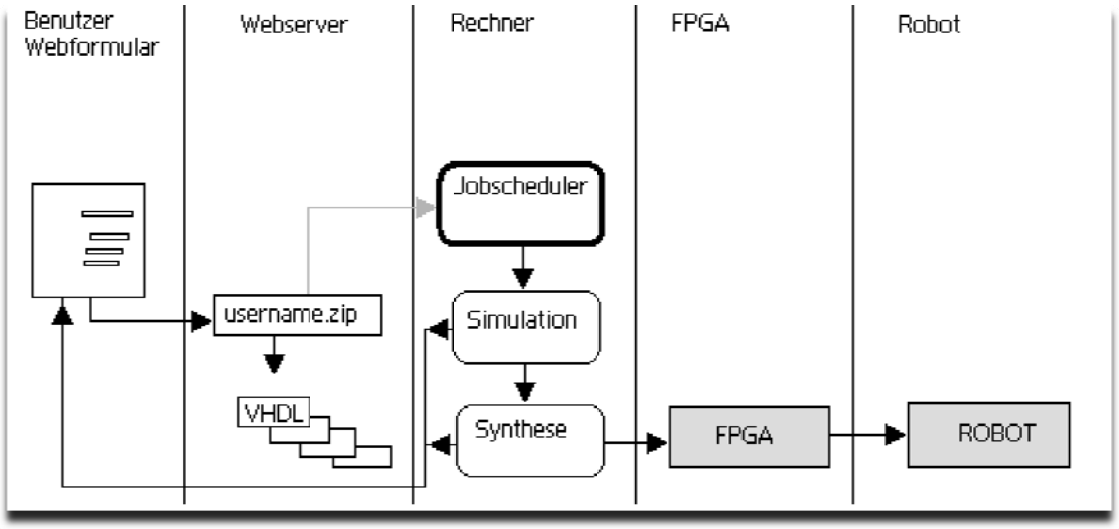

Abbildung 27: Job Scheduler mit Aufgabensteuerung

Bei der Aufgabensteuerung wird die Webschnittstelle „CONTROL.ASPX“ verwendet, um in Zusammenarbeit mit Job Scheduler, Teilaufgaben auszuführen. Solche Teilaufgaben, die sukzessive oder einzeln ausgeführt werden können, sind: Dateien hochladen und entpacken, Simulation starten, Synthese ausführen, Bitstreams in FPGA laden.

7.2.7 Job Scheduling über VPN

Für die Steuerungsaufgaben über das Internet kann man VPN(Virtuelle Private Network) Verbindung verwenden, und so quasi in Echtzeit auf die Steuerbefehle reagieren. Anders als bei den Internetverbindungen ohne VPN-Tunneltechnik, ist es möglich, über VPN-Verbindung NTFS-Ereignisse auf entfernten Rechnern zu verfolgen. Gegenüber anderen Tunnelarten zeichnet sich der VPN-Tunnel dadurch aus, dass er sämtliche Netzwerkpakete weiterleitet.

Da die meisten Webserver, wegen die sicherheitsrelevante Einschränkungen, nur in bestimmten Stellen Daten ablegen können wie z.B.: in die virtuelle Verzeichnissen, wird für die Datenübertragung über das VPN der Jobscheduler verwendet. Dabei wird vom Webserver erstellet „CMMD.R4W“ –Datei von ein zusätzlicher Instanz des Jobschedulers über das VPN in die gewünschte Verzeichnis des entfernten Rechners kopiert.

Die Verwendung von VPN in diesem Projekt erweitert die Einsatzmöglichkeiten von der Robotersteuerung entscheidend, und macht das System zu einer sicheren Internetanwendung. Ein weiterer Vorteil des VPN Einsatzes ist es, dass man für das lokale Netzwerk entwickelte Kommunikation ohne Änderungen über das Internet weiter verwenden kann.

Kapitel 8 **Die RS-232 Schnittstelle**

Die serielle Schnittstelle RS-232 wird zwischen zwei Geräten verwendet. Wegen des verwendeten einfachen Protokolls ist sie weniger störungsanfällig, und leicht zu realisieren. Um die Kommunikation zu ermöglichen, sind die meisten FPGA-Entwicklungsboards mit seriellen Schnittstellen ausgestattet. In dieser Diplomarbeit wird die serielle Schnittstelle für die Robotersteuerung eingesetzt. Dabei werden Positionierungsdaten über die serielle Schnittstelle vom Rechner auf die Entwicklungsboard übertragen.

RS232 benutzt für die Datenübertragung ein einfaches asynchrones serielles Verfahren. Seriell bedeutet, dass die einzelnen Bits des zu übertragenden Bytes nacheinander über einer einzigen Datenleitung geschoben werden. Asynchron heißt, dass es keine Taktleitung gibt, die dem Datenempfänger genau sagt, wann das nächste Bit auf der Datenleitung liegt. So ein Verfahren kann nur funktionieren, wenn Sender und Empfänger genau mit dem gleichen internen Takt arbeiten, und wenn dem Empfänger gesagt wird, wann das erste Bit genau anfängt (Synchronisation). [10]

8.1 **RS-232 auf dem Entwicklungsboard**

Bei der seriellen Übertragung wird eine hohe Störsicherheit durch die verwendeten physikalischen Pegel von +12V (für low) ist –12V (für high) gewährleistet. Da die FPGA Bausteine mit einer weit geringeren Spannung arbeiten, müssen für die Übertragung verwendete Signalpegel auf TTL Ebene reduziert werden. Diese Aufgabe wird von dem RS-232-Pegelkonverter übernommen. Auf nachfolgender Abbildung ist der von der Virtex II-Pro verwendete MAX3223 Pegelkonverter dargestellt.

Auf dem Virtex II-Pro Entwicklungsboard steht eine serielle Schnittstelle mit MAX3223 Pegelkonverter zur Verfügung. Ein zweiter Pegelkonverter wurde auf einer angefertigten Platine über einen Erweiterungssockel angebunden. Durch Setzen oder Entfernen von abgebildeten Jumpers(J31, J32), wird die Schnittstelle in DTE oder DCE Grundmodus betrieben. Hier wird der DCE Modus verwendet.

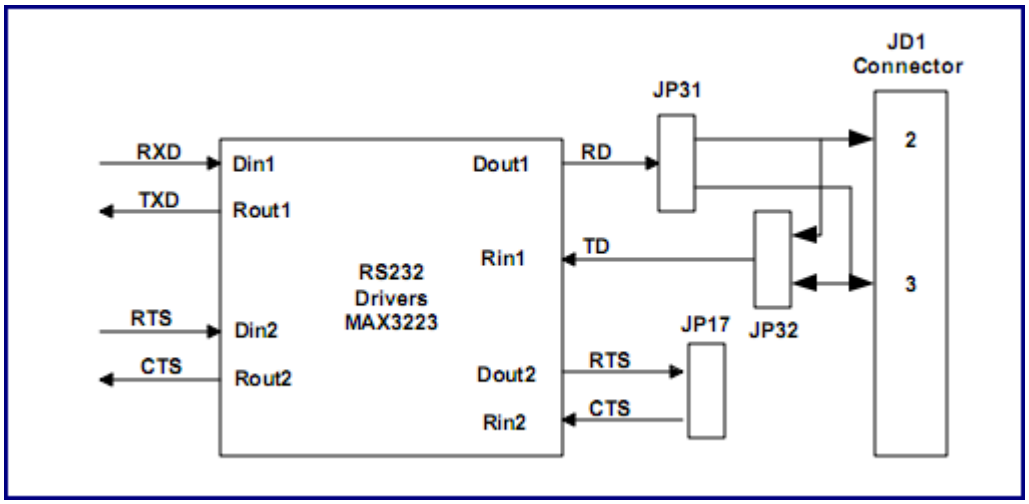

Abbildung 28: MAX3223 Pegelkonverter

Diese Treiberschaltkreise bewerkstelligen die Pegelkonvertierung der eingehenden Signale für die synthetisierten UART, von RS-232-Pegeln (–12 beziehungsweise +12 Volt) zu 0 beziehungsweise 3 Volt.

8.2 Realisierung in der FPGA

Für den Datenaustausch über eine serielle Schnittstelle zwischen FPGA und Peripheriegeräten, ist die nachträgliche Synthetisierung der UART Komponenten erforderlich. Es gibt unterschiedliche Möglichkeiten, einen RS-232 Kontroller in einer FPGA zu realisieren. Unabhängig davon, wie der UART Kontroller letztendlich realisiert wird, ist zusätzlich noch dafür zu sorgen, dass die Übertragungsfrequenz am Sender und Empfänger identisch ist. Davon Abhängig, wie die übertragenen Daten verwendet werden sollen, kann man die übertragenen Bit sofort auswerten, oder solange zwischenspeichern, bis alle Datenbits übertragen sind. Für die Zwischenspeicherung sind die Schieberegister und Automaten am besten geeignet.

Da in diesem Projekt die Integrität der übertragenen Daten, anhand der Parity Bits überprüft wird, werden die empfangenen Bits solange gespeichert, bis die gesendete Stopbits empfangen wurden, und die Gültigkeit der übertragenen Daten berechnet ist. Für die Überprüfung wird ein Automat, realisiert als ein VHDL Modell, verwendet. Der Automat erhält nach jedem Takt das gesendete Bit, danach wechselt sich sein Zustand, und zum Schluss berechnet er die Gültigkeit der empfangenen Daten. Ausgangsseitig werden die erhaltenen Bits auf ein 8Bit breite Bus angelegt, und gleichzeitig die Gültigkeit dieser Daten am Parity Ausgang signalisiert.

Weil ein komplettes Steuerbefehl aus drei Bytes besteht, werden drei Übertragungen benötigt, bevor die Daten weiterverarbeitet werden können. Um die erhaltenen Teile zu speichern, wird ein 16 Bit großes Schieberegister verwendet, in dem die ersten zwei Befehle hintereinander geschoben und gespeichert werden. Der dritte 8 Bit langer Teil wird nicht mehr im Schieberegister gespeichert, da die Daten für die Weiterverarbeitung am Ausgang des Kontrollermoduls anliegen.

8.2.1 RS-232 Taktgenerator

Die Realisierung des Taktgenerators für de periodische Abtastsignal bei asynchroner serieller Datenübertragung ist zwingend notwendig. Im Gegensatz zum synchronen seriellen Datentransfer, wird bei der asynchronen Variante auf die zusätzliche Taktleitung verzichtet bzw. sie wird nicht verwendet. In diesem Projekt wird für die Kommunikation zwischen Rechner und das FPGA-Entwicklungsboard die asynchrone Datenübertragung eingesetzt.

Es gibt unterschiedliche Verfahren, um die Empfangene binäre Signale abzutasten. Das hier verwendete Abtastverfahren verwendet die gleiche Abtastfrequenz, wie die seriellen Signale generiert werden.

Übertragungsgeschwindigkeit

Um die gesendeten Daten richtig empfangen zu können, wird vor der Realisierung der Kommunikationskanäle die Übertragungsgeschwindigkeit vereinbart. Die Übertragungsgeschwindigkeit wird in Baud angeben. Bei den Controlleranwendungen hat sich 9600 Baud Übertragungsgeschwindigkeit etabliert, und wird ebenfalls für die Datenübertragung zwischen Rechner und FPGA verwendet.

Berechnung der Abtastrate

Senderseitig(am Rechner) wird die serielle Datenübertragung über ein für diese Zwecke geschriebenes C/C++ Programm abgewickelt. Über die verwendete DCB Struktur wird die gewünschte Baudrate übergeben, die für eine serielle Kommunikation eingestellt wird. Die genaue Vorgansweise ist detailliert im Anhang vorgestellten Programm Code beschrieben.

In der FPGA realisierte VHDL Model ist ein Baudgenerator implementiert, die die vereinbarte Abtasttaktung erzeugt. In diesem Sonderfall entspricht die Abtastfrequenz einer Signalübertragungsfrequenz von 9600 Hz, bei einer Übertragungsrate von 9600 Baud, und bei der Verwendung von einer Signalleitung. Um dieses Nutzsignal abzutasten, wird ein periodisches Abtastsignal mit einer Periodendauer von 104166,(6) ns, und mit einer Impulslänge von 20 ns benötigt.

Um die Abtastsignal erzeugende digitale Schaltung in ein FPGA zu realisieren, muss die Anzahl der Grundtakte berechnet werden, nachdem eine Abtastung der empfangenen Signale

erfolgen kann. Aus den oben berechneten Werten und aus dem Wert der verwendeten Grundfrequenz ist es leicht, das Verhältnis zwischen Abtastfrequenz und Grundfrequenz aufzustellen. Da in diesem Projekt 50MHz Taktfrequenz für die Taktungder in FPGA realisierte digitale Schaltung verwendet werden, bekommt man die Abtast- und Grundfrequenz Verhältnis nach folgender Berechnung:

$$\frac{\text{Signallänge}}{\text{T Grundfrequenz}} = \frac{104166.(6)\text{ns}}{20\text{ ns}} = 5208.(3)$$

Auf der Basis der getroffenen Berechnungen ist es möglich, einen einfachen Baudgenerator in eine FPGA als digitale Schaltung zu implementieren. Für die Umsetzung wird ein Zähler verwendet, der nach jedem 5208.(3) gezählten Takt ein Abtastimpuls von 20 ns Länge (entsprechend der Grundfrequenz) erzeugt. Da man nicht ein drittel eines Taktes zählen kann, wird man einen einfachen Trick benutzen, um das Verhältnis zwischen dem generierten Abtasttakt und dem Grundtakt beim zählen aufrecht zu erhalten. So werden die ersten zwei Abtasttakte jeweils nach 5208 Grundtakten erzeugt, bei jedem dritten Abtasttakt wartet man einen zusätzlichen Grundtakt länger, bevor der Abtastimpuls generiert wird. Diese leichte Abtastverschiebung hat keine Auswirkung auf das Ergebnis, da nach jedem dritten Takt die Verschiebung kompensiert wird:

$$2 * 5208 + 5209 = 3 * 2508.(3)$$

8.2.2 Serielle Signalübertragung

Die Spezifikation der seriellen Schnittstelle erlaubt die Verwendung verschiedener Datenrahmen. Da alle möglichen Variationen in den Standards festgelegt sind, müssen bei beiden Geräten, die an der Kommunikation beteiligt sind, alle Parameter gleich eingestellt sein, damit eine erfolgreiche Kommunikation garantiert werden kann.

In diesem Projekt wird für die serielle Datenübertragung ein Signal mit einem Startbit, acht Datenbits(Nutzdaten), ein Parity-Bit und ein Stoppbit verwendet.

Synchronisation der Abtastfrequenz

Es gibt viele Gründe, warum von unterschiedlichen Geräten erzeugte Signale synchronisiert bzw. aufeinander abgestimmt werden müssen. Die physikalischen Eigenschaften der Übertragungsmedien, Temperaturschwankungen und die unterschiedliche Ausführung digitaler Schaltungen beeinflussen die Taktfrequenzen der Schaltungen. Auf nachfolgender Abbildung wird die Verschiebung des Abtastsignals gegenüber der Eingangssignal bei Verwendung gleicher Frequenzen, dargestellt. Es ist ersichtlich, dass ohne Synchronisation der beiden Signale früher oder später ein falscher Abtastwert bekommen wird.

Es gibt unterschiedliche Verfahren um diese Verschiebungsfehler zu korrigieren. Man kann zum Beispiel, die Eingangssignale mehrfach abtasten und anschließend aus den erhaltenen Werten eine 2-aus-3-Entscheidung oder eine andere Mehrheitsentscheidung zu treffen. Bei diesen Verfahren werden die Signale nicht aufeinander synchronisiert.

Bei dargestellten Abtastverfahren wird das Eingangssignal 1fach abgetastet, dabei verwenden sowohl Abtastsignal als auch das abzutastende Signal eine annähernd gleiche Taktung.

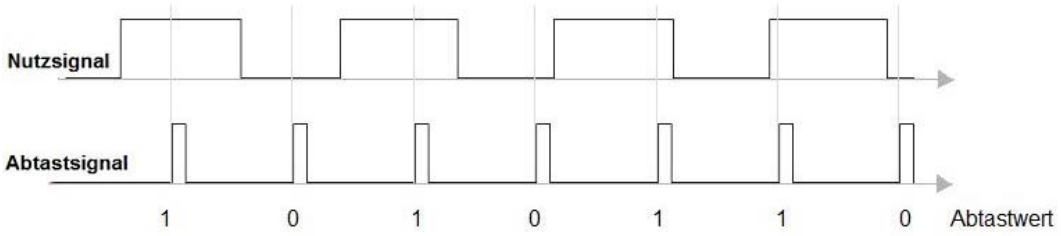

Abbildung 29: Verschiebung der Abtastsignale

Wenn keine Daten gesendet werden, liegt der Ruhepegel(„logischen 1“) an. Als erstes wird das Startbit(logisch „0“) gesendet, um den Empfänger mit dem Sender synchronisieren zu lassen. Um das Abtastsignal zu Positionieren, wird nach erhalt jeder „0“ Bits 2604 Takte gewartet, und anschließend den Abtasttakt generiert. Da 2604 Takte die halbe Periodendauer des abzutastenden Signals ist, wird der Abtastsignal genau in der Bit-Mitte des Eingangs-signals gesetzt.

Nach den empfangenen Startbits folgen acht Datenbits (Nutzdaten) mit den anschließenden Parity-Bit. Abgeschlossen wird die Übertragung mit einem Stoppbit logisch „1“. Die folgende Ruhezeit wird von dem Sender bestimmt, und ist im Programm mit 50 Milli-sekunden begrenzt.

Kapitel 9 **Schlussbetrachtung**

Die Roboter werden von Jahr zu Jahr immer vielseitiger eingesetzt. Es gibt immer wieder Veranstaltungen - die häufig von militärischen Einrichtungen unterstützt werden - wo die Roboter neuester Generation vorgestellt werden (z.B. autonome Roboter veranstalten ein Wettrennen). Nach dem Vorbild der DARPA Grand Challenge, bei der autonome Roboterfahrzeuge in der Mojavewüste ihr Können im Rahmen einer Leistungsschau demonstrieren, wurde kürzlich die "European Land-Robot Trial" (ELROB) am Truppenübungsplatz im unterfränkischen Hammelburg veranstaltet.

Neben der Roboterentwicklung, nimmt eine weitere Schlüsselstellung in dieser Diplomarbeit das E-Learning und Entwicklung digitaler Schaltungen mit Hilfe von über das Internet verbundener realer Hardware ein. Mit diesem Projekt wurde nebenbei gezeigt, dass E-Learning und Distant Education durchaus auch auf Lehrveranstaltungen mit Laborcharakter und praktische Übungen anwendbar sind.

Das Technische Institut für Computertechnik auf der Technischen Universität Wien beschäftigt sich bereits seit längerer Zeit mit dem Design integrierter Schaltungen, sowohl im Rahmen von Forschungsprojekten mit oftmaliger Beteiligung von Partnern aus der Industrie, als auch in der Lehre. Für die Lehrveranstaltung „ASIC-Entwicklung“, die Grundlagenwissen im Bereich des Designs digitaler Schaltungen vermitteln soll, wird in einem praktischen Laborteil auf die Produktpalette und Entwicklungstools der Firma Altera gesetzt. Studenten entwickeln dabei mit Hilfe der FPGA-Entwicklungsumgebung MAX+plus II das Design eines einfachen Mikrocontrollers, der ein Applikationsprogramm wie einen Taschenrechner oder Stoppuhr arbeitet. Im Rahmen des Projekts „axis4web“ können Studenten über einen einfachen Webbrowser von zu Hause aus ein Digitaldesign in das am ICT befindlichen FPGA-Board uploaden, die Taster und Schalter des Boards betätigen, und den Zustand der 7-Segment-Anzeigen beobachten. [11]

Das steigende Interesse hat gezeigt, dass das Thema Roboter sowie effiziente Steuerung aktueller ist denn je. Dass die industrielle Nutzung von Roboter von großer Bedeutung ist, ist mittlerweile unumstritten; die anfängliche Furcht, Roboter könnten Arbeitsplätze vernichten, ist ebenso unbegründet wie viele andere Vorurteile. Die Anbindung und Steuerung von FPGA- Prototypenboards über das Internet, bietet neue Möglichkeiten um das E-Learning zu erweitern und die Schlüsseltechnologien dabei leichter zugänglich zu machen.

9.1 Zusammenfassung

Ziel der vorliegenden Arbeit ist es, die Realisierung eines auf FPGA-Steuerung basierten Roboterarms für Laborübungszwecke umzusetzen. Um dieses Ziel zu erreichen, wurde das Projekt, wie in den vorangegangenen Kapiteln detailliert beschrieben, auf folgende Teilprojekte gegliedert:

1.) Realisierung von einer Webanwendung, um die Roboterbewegungen zu verfolgen, VHDL Simulation und Synthese durchzuführen, FPGA Programmieren und letztendlich die zurückgelieferte Ergebnisse zusammenfassend darzustellen.
2.) Entwurf und Realisierung von einem Signalverstärker-Board, um die vom FPGA erstellten Signale zu verstärken und gleichzeitig die teuere Hardware zu schützen.
3.) Um die übertragenen Informationen im FPGA zu speichern bzw. zu verarbeiten, sind VHDL Modelle erstellt worden. Bei dem Entwurf dieser Modelle wurden besonders auf die Sicherheit, sowie auf die Einsetzbarkeit in zukünftige Systeme geachtet.
4.) Für die Informationsübertragung von PC zum FPGA ist das Programm RS232.exe erstellt worden, mit dessen Hilfe Dateiinhalte oder auch einzelne Datensätze übertragen werden können.
5.) Um konkurrierende Aufgaben zur Steuerung, Simulation und Synthese in diesem Projekt abarbeiten zu können, wurde das Programm Job Scheduler erstellt. Der Job Scheduler wird all diese Aufgaben mit Hilfe einer Internetverbindung ortsunabhängig und quasi in „Echtzeit“ auf einen dafür vorgesehenen Rechner ausführen. Der hier vorgestellte Job Scheduler ist für diese Diplomarbeit, speziell für die Robotersteuerung entworfen und realisiert.
6.) Bau eines Sieben-Segment-Roboterarms, der von insgesamt neun Servomotoren angetrieben wird.

Mit dem Roboterarm, der in dieser Diplomarbeit erstellt wurde, und mit der dazu entwickelten Steuerung bzw. periphere Hardware lässt sich eine Vielzahl von Applikationen verwirklichen.

Das System ist jederzeit um neue Elemente erweiterbar. Durch die zusätzliche Integration von Kraft-Momenten- und Entfernungssensoren, kann der Roboter Fähigkeiten wie Fühlen und Tasten erlangen. Der Einsatz von Bildverarbeitungsprogrammen mit Mustererkennung kann eine Interaktion mit der Umgebung ermöglichen, wodurch der Roboter auch autonom interagieren kann.

Die Implementierung von VHDL Modellen wurde grundsätzlich auf zwei FPGA-Entwicklungsboards ausgeführt, und auf dieser Weise interessante Erkenntnisse über die Hardwaresynthese identischer VHDL Modelle auf unterschiedlichen Plattformen gewonnen. Sowohl Spartan 3A als auch Virtex II Pro sind auf Xilinx-Technologie basierende

FPGAs, und haben mehr als ausreichende Kapazität für den vorgesehenen Einsatzbereich. Die umgesetzten Beispielapplikationen haben weniger als 10% der zur Verfügung stehender FPGA Ressourcen von Virtex II Pro beeinsprucht.

Das Robot4Web Webinterface bietet eine übersichtliche Möglichkeit, VHDL-Codes über das Internet auf professionelle Plattformen ausführen zu können, und die generierte Rückgabe zu bewerten. Zusätzlich besteht die Möglichkeit, die Ausführung von synthetisierbaren Programmen direkt auf einem echten Roboter zu testen und zu betrachten.

Die, für diese Diplomarbeit entwickelte C/C++ Programme ermöglichen eine ereignisbasierte Kommunikation zwischen den Systemkomponenten. Diese Art von Kommunikation ist streng genommen keine Echtzeitkommunikation, aber trotzdem sehr effektiv und robust. Die Tatsache, dass keine zusätzlichen Installationen notwendig sind, um die Steuerungsdaten zu übertragen, zeigt mitunter die größte Stärke des Systems. Das bedeutet eine enorme Erleichterung bei zukünftigen Entwicklungen und Erweiterungen.

Der Signalboard ermöglicht eine sichere und saubere Signalübertragung. Infolge der galvanischen Entkopplung zweier, auf unterschiedlichem Messpotenzial laufender Schaltkreise, ist der verwendete FPGA-Entwicklungsboard bestens gegen Überspannung geschützt.

9.2 Ausblick

Die Entwicklung der FPGAs geht in die Richtung, immer dichtere Architekturen und gleichzeitig energiesparende Lösungen zu realisieren. Xilinx hat gerade eine neue FPGA-Architektur auf den Markt gebracht, die in dualem Energiemodus arbeitet. Mit Hilfe von „on-chip power management“ Technologie wird es möglich sein, während der Hardwaresynthese zukünftig die Energieversorgung der FPGA Komponenten zu optimieren. Durch den geringeren Energieverbrauch können nicht nur thermische Probleme besser in den Griff bekommen werden, sondern es wird bei Batteriebetrieb die Arbeitsdauer signifikant verlängert.

Weitere Verbesserungen erreicht Xilinx durch die verwendete ISE Software. Die neueste ISE-Version verspricht die zweifache Arbeitsgeschwindigkeit gegenüber der Vorgängerversion. Diese Analysiert die zu implementierende Module, und schaltet automatisch ungenutzte Taktleitungen, Signalleitungen oder logische Elemente ab. Die Verwendung von BUFGCE erlaubt die Ein- oder Abschaltung von Taktsignalen. BUFGMUX bietet die Möglichkeit, zwischen Takte zu wechseln. Beide Techniken vermeiden Glitches, und sparen dabei Energie.

Die neue Virtex®-5 FXT Familie von Xilinx FPGAs wurde hauptsächlich entwickelt, um die Integration der hoch entwickelten eingebetteten Systeme auf einer einzelnen, flexiblen Hochleistungsplattform zu ermöglichen.

Die Zukunft der FPGAs ist nicht klar definiert, laut FPGA-Hersteller sind mehrere Zukunftsszenarien denkbar. Laut Voraussagen, werden die zukünftigen FPGAs immer häufiger mit Mikroprozessor-Cores auf den Markt kommen, und bis 2010 etwa 30% der FPGAs ausmachen. Als treibende Kraft des FPGA-Markts werden sich Consumer-Elektronik und die Automobilbranche etablieren.

Anhang

Das Signalboard

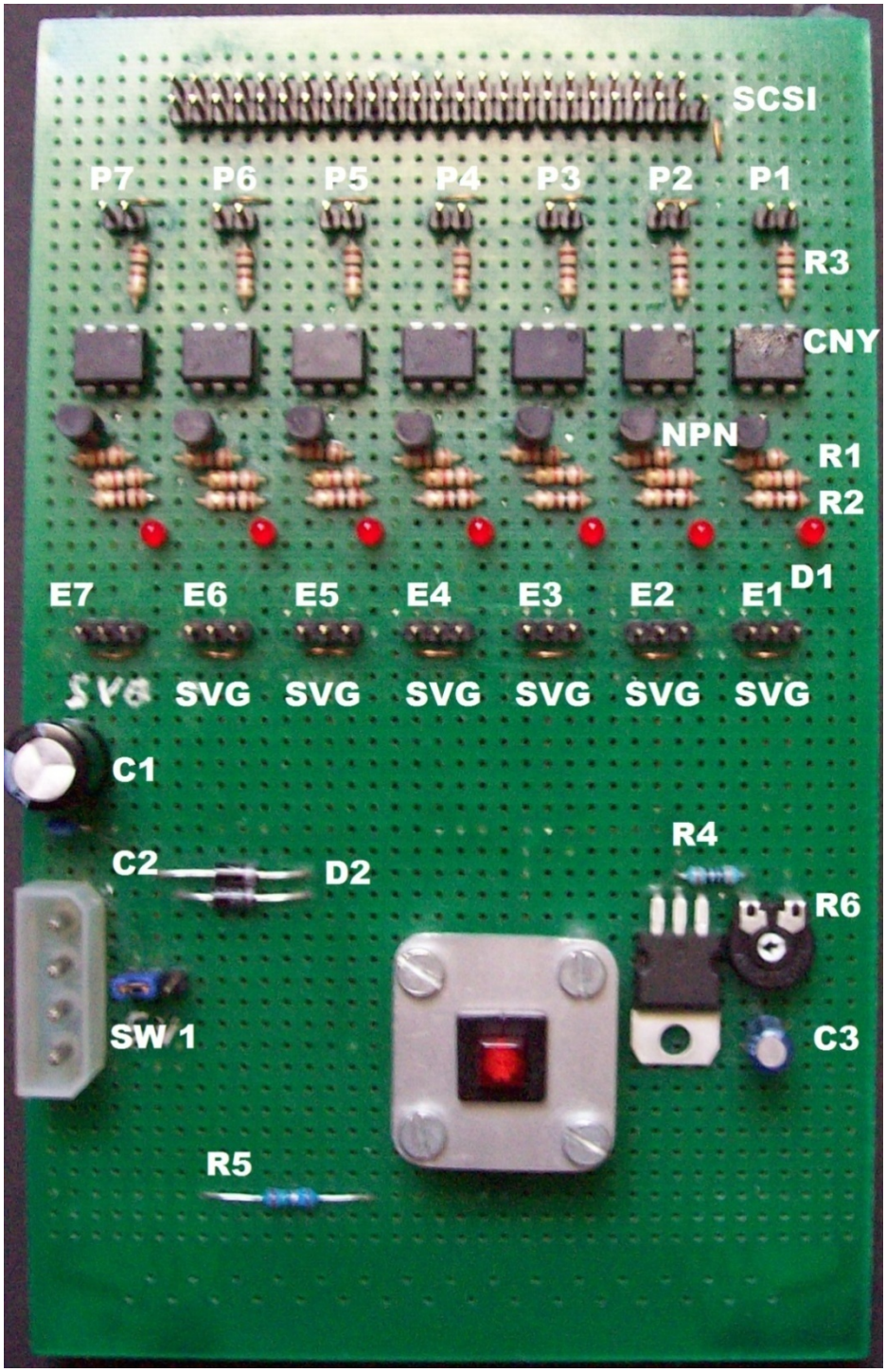

Abbildung 30: Signalverstärker

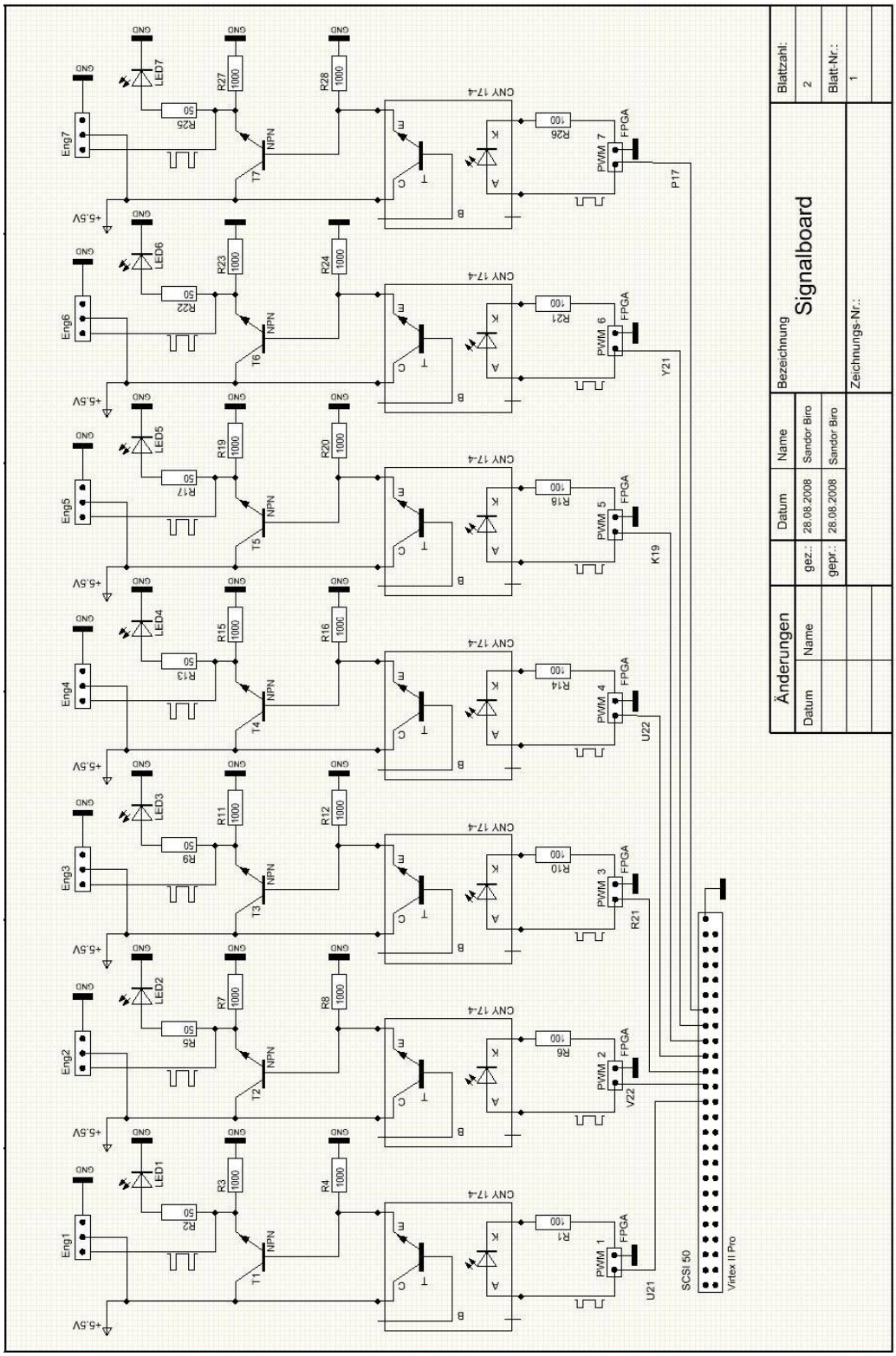

Abbildung 31: Signalverstärker, Stromlaufplan

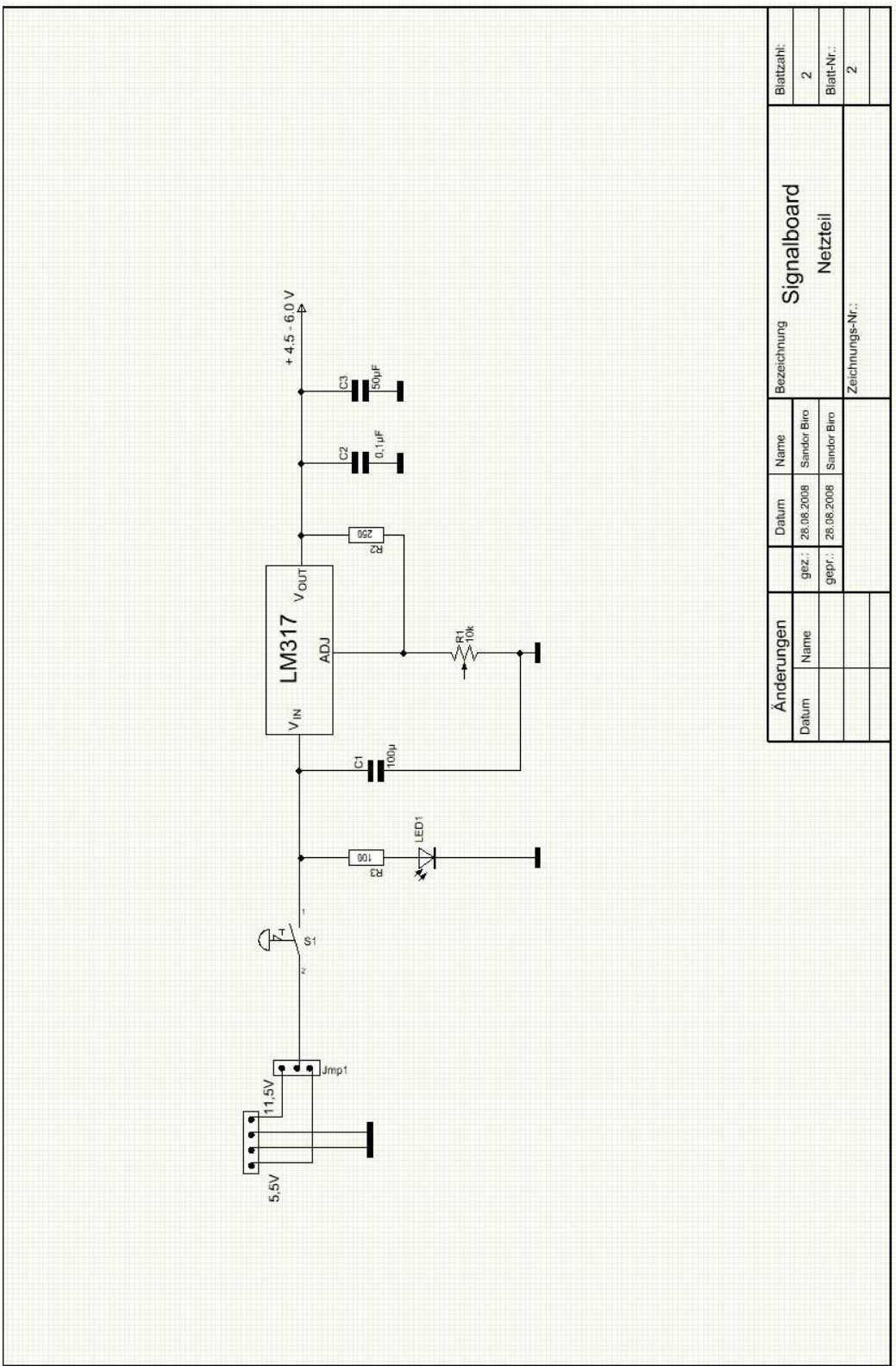

Abbildung 32:Stromlaufplan, Netzteil des Signalverstärkers

Virtex-II Pro Pin #	System ACE Signal Name	JP29 Pin #	JP29 Pin #	System ACE Signal Name	Virtex-II Pro Pin #
	3.3V	1	2	3.3V	
	TDO	3	4	GND	
	TMS	5	6	CLOCK	
	TDI	7	8	GND	
B1	PROGRAMn	9	10	TCK	
	GND	11	12	GND	
W21	OEn	13	14	INITn	W17
N18	MPA0	15	16	WEn	H18
R18	MPA2	17	18	MPA1	P18
	2.5V	19	20	MPA3	N19
R22	MPD00	21	22	2.5V	
J19	MPD02	23	24	MPD01	J20
U21	MPD04	25	26	MPD03	T22
V22	MPD06	27	28	MPD05	K20
R21	MPD08	29	30	MPD07	K18
U22	MPD10	31	32	MPD09	P21
K19	MPD12	33	34	MPD11	T21
Y21	MPD14	35	36	MPD13	V21
P17	MPA4	37	38	MPD15	W22
U18	MPA6	39	40	MPA5	T18
F22	IRQ	41	42	GND	
E13	RESETn	43	44	CEn	G22
Y18	DONE	45	46	BRDY	N21
W20	CCLK	47	48	BITSTREAM	V17
	GND	49	50	NC	

Tabelle 4: Signalbeschreibung und Pin-Belegung, System ACE Anschluss von VirtexII-PRO

In der Tabelle 4 grau unterlegten Signal-Pins werden für die Datenübertragung verwendet. Diese Pins werden über ein SCSI-Flachbandkabel mit dem Signalverstärker verbunden.

uart.c

```
/*-------------------------------------------
Projekt:    Robot4Web
Autor:      Sandor Biro

In diesem Modul sind die Funktionen definiert, die für
serielle Kommunikation eingesetzt werden.
Funktionen:
            OpenComPort,
            CloseComPort,
            SendCommand,
            WriteCom,
            Read_Commands,
            usage.

Aufruf in RS232.cpp: #include „uart.c“
-------------------------------------------*/

#include <windows.h>
#include <cstdio>
#include <conio.h>
#include <iostream>
#include <fstream>
#include <shlwapi.h>
#include <string>
#include "stdio.h"
#include <iomanip>
#include "atlbase.h"
#include "atlstr.h"
#include "comutil.h"

using namespace std;
#define COM_ERROR 1
#define COM_OK 0

HANDLE hCom;

// Öffnet ein serieller Kommunikationsport und stellt die
// Randeigenschaften für die Übertragung ein.
// Die Übergabeparameter ist die Portnummer.
// Datenübertragungsrate: 9600
// Parity :Odd
// Stopbits: 2
// Datenbits: 8
void OpenComPort(char* port)
{
      DCB dcb;
      COMMTIMEOUTS ct;

    size_t origsize = strlen(port) + 1;
    const size_t newsize = 100;
    size_t convertedChars = 0;
    wchar_t lpszMeldung[newsize];
```

```
    mbstowcs_s(&convertedChars, lpszMeldung, origsize, port,
_TRUNCATE);
    //Handler erstellen und an die globalen hCom Handler übergeben.
      hCom = CreateFile(lpszMeldung, GENERIC_READ | GENERIC_WRITE,
0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
      if(hCom==INVALID_HANDLE_VALUE) exit(COM_ERROR);

      //Initialisiert die Kommunikationsparameter.
      if(!SetupComm(hCom, 4096, 4096)) exit(COM_ERROR);

      if(!GetCommState(hCom, &dcb)) exit(COM_ERROR);
      dcb.BaudRate = 9600;
      ((DWORD*)(&dcb))[2] = 0x1001;  // Porteigenschaften setzen
für TXDI + no flow-control
      dcb.ByteSize = 8;
      dcb.Parity = ODDPARITY;
      dcb.StopBits = 2;

      if(!SetCommState(hCom, &dcb)) exit(COM_ERROR);

      // set the timeouts to 0
      ct.ReadIntervalTimeout = MAXDWORD;
      ct.ReadTotalTimeoutMultiplier = 0;
      ct.ReadTotalTimeoutConstant = 0;
      ct.WriteTotalTimeoutMultiplier = 0;
      ct.WriteTotalTimeoutConstant = 0;
      if(!SetCommTimeouts(hCom, &ct)) exit(COM_ERROR);
}

void CloseComPort()
{
      CloseHandle(hCom);
}

DWORD WriteCom(char* buf, int len)
{
      DWORD nSend;
      if(!WriteFile(hCom, buf, len, &nSend, NULL)) exit(COM_ERROR);

      return nSend;
}

void SendCommand(char b)
{
      WriteCom(&b, 1);
}

int ReadCom(char *buf, int len)
{
      DWORD nRec;
      if(!ReadFile(hCom, buf, len, &nRec, NULL)) exit(COM_ERROR);

      return (int)nRec;
}
```

```
 void usage( char* arg)
{
      printf("Falsches Argument!\n");
    printf("usage: %s -p Port -f Filename\n",arg);
      printf("or   : %s -p Port -e Engine -v Velocity -c Position
\n",arg);
}

// Öffnet die als Parameter übergebene Datei und liest
// Zeilenweise die Befähle aus. Anschließend werden die
// Steuerbefähle an PFGA-Entwicklungsboard gesendet.
// Randeigenschaften für die Übertragung ein.
// Parameter: Datei mit die Steuerbefähle
// Rückgabewert: Integer, signalisiert die Übertragungserfolg
int Read_Commands(char* sFileName)
{

  int zeile = 0;
  string line;
  ifstream myfile (sFileName);
  if (myfile.is_open())
  {
    while (! myfile.eof() )
    {
      getline (myfile,line);
        char delims[] = "#";
        char *result = NULL;
        zeile++;

        if (line[0]!= ':' && line[0]!= ' '&& line != "")
        {
              char* buffer = new char [line.size() +1];
              line.copy(buffer, string::npos);
              buffer[line.size()]=0;
              int t = 0;
              result = strtok( buffer, delims );
              while (result != NULL )
              {
                    t++;

                    WriteComChar( atoi(result));
                    Sleep(100);
                    result = strtok( NULL, delims );
              }
              if (t!=3)
              {
                    WriteComChar(0);
                    if(t== 1)
                          WriteComChar(0);
                    printf("RS-232: input/output error \n#  Input
sequence corrupt!  row: %d\n",zeile);
              }
        }
    }
    myfile.close();
```

```
  }
  else cout << "SEND DATA TO ROBOT:  \n Unable to open file!\n";

return COM_OK;
}
```

RS232.cpp

```
/*-------------------------------------------
Modulname:  rs232.cpp
Projekt:    Robot4Web
Autor:      Sandor Biro

 Das Programm rs232.exe ermöglicht die serielle Datenübertragung
 von einem Computer auf ein FPGA-Entwicklungsboard.
 In die Abhängigkeit, wie die Parameter übergeben sind,
 wird die Programmausführung beeinflusst.
 Die Parameter können in eine Datei zusammengefasst übergeben
werden
 oder einzeln für die Gelenknummer, ComPort, Geschwindigkeit und
Position.

 Programmstart:
      rs232.exe -p ComPort: -f Dateiname
      oder
      res232.exe -p ComPort -e Gelenknr. -v Geschwindigkeit -c
Position

-------------------------------------------*/
#include  <io.h>
#include  <stdio.h>
#include  <stdlib.h>

#include "stdafx.h"
#include "uart.c"


//
//
int  main(int argc, char*  argv[])
{

      int  i=0;
      int j=0;
      char* arg_list[5];
      for (int x=0; x<5;x++)
```

```
            arg_list[x] =NULL;

      if (argc < 3)
      {
            usage(argv[0]);
            return 1;
      }
      else
      {
            for (j=1; j<argc; j+=2)
            {
                  if ( _stricmp( argv[j] , "-f")==0) // Datei
                  {
                        arg_list[1] = argv[j+1];
                  }
                  else if ( _stricmp( argv[j] , "-p")==0) // ComPort
                  {
                        arg_list[0] = argv[j+1];
                  }
                  else if ( _stricmp( argv[j] , "-e")==0)
//Gelenknummer
                  {
                        arg_list[2] = argv[j+1];
                  }
                  else if ( _stricmp( argv[j] , "-v")==0) //
Geschwindigkeit
                  {
                        arg_list[3] = argv[j+1];
                  }
                  else if ( _stricmp( argv[j] , "-c")==0) //
Position
                  {
                        arg_list[4] = argv[j+1];
                  }
                  else
                  {
                        usage(argv[0]);
                        return 2;
                  }
            }

      }

      if (arg_list[1] != NULL && arg_list[0] != NULL){
            // Kommunikationsport für Datei-Eingabe wird geöfnet.
            OpenComPort(arg_list[0]);
            // Liest die Befehle zeilenweise aus der Datei und
sendet diese.
            int z =  Read_Commands(arg_list[1]);
      }
      else if (arg_list[2] != NULL && arg_list[2] != NULL &&
arg_list[2] != NULL && arg_list[0] != NULL)
      {
            // Kommunikationsport für die direkte Eingabe wird
geöfnet.
```

```
            OpenComPort(arg_list[0]);
            // Sende Gelenknummer
            SendCommand(atoi(arg_list[2]));     Sleep(10);
            // Sende Geschwindigkeit
            SendCommand(atoi(arg_list[3]));     Sleep(10);
            // Sende Position
            SendCommand(atoi(arg_list[4]));     Sleep(10);
      }
      else
      {
            usage(argv[0]);
            return 3;
      }

      CloseComPort();   //Kommunikationsport wird geschlossen
      return 0;
}
```

CD im Anhang

Inhaltsverzeichnis und Hilfe	/Start.PDF
Elektronische Version dieser Arbeit:	/ROBOT4WEB/R4W.PDF
Softwareprojekt, Job Scheduler:	/SOFTWARE/JOBSCHEDULER/
Softwareprojekt, Serielle Kommunikation:	/SOFTWARE/RS232/
Softwareprojekt, Webinterface :	/SOFTWARE/WEB/
VHDL Modelle	/VHDL/
Videoaufnahmen:	/VIDEO/

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Michael Dunn: “Guide to Writing Shell Extensions, A step-by-step tutorial on writing shell extensions“, 27 März 2000, Aktualisiert 14.03.2006.
http://www.codeproject.com/shell/shellextguideindex.asp

[2] MSDN © 2008 Microsoft Corporation: “Directory Management Functions (Microsoft)”, Februar, 2008.
http://msdn.microsoft.com/en-us/library/aa363950(VS.85).aspx

[3] Jeffrey Cooper Stein and Jeffrey Richter: “'Keeping an Eye on Your NTFS Drives: the Windows 2000 Change Journal“, Microsoft System Journal, Oktober 1999.
http://www.microsoft.com/msj/1099/journal2/journal2.aspx

[4] Gerhard-Helge Schildt: „Impulstechnik, Grundlagen und Anwendungen“, LYK Informationstechnik GMbH, August 2006, ISBN-10: 3-200-00791-5.

[5] Wes Jones: „CDirectoryChangeWatcher –ReadDirectoryChangeW all wrapped up“, Revision 2.0. The Code Project, 30 May 2002.
http://www.codeproject.com/KB/files/directorychangewatcher.aspx

[6] Mentor Graphics : „ModelSim User’s Manual“ Revision 040401, Version 6.1Beta 22.Februar.2005.
http://www.model.com

[7] Gerhard-Helge Schildt; Wolfgang Kastner.: „Prozessautomatisierung“, Springer Verlag,Wien, 1998, ISBN:3-211-82999-7.

[8] Mentor Graphics : „ModelSim Command Reference“ Revision: 040401 , Version 6.1Beta, 22 Februar, 2005.
http://www.model.com

[9] Erhard Henkes: „Roboter - Freund, Feind oder Sklave?“, abgefragt : 06.06.2008, http://www.henkessoft.de/Roboter/Roboter.htm.

[10] Sprut: “RS232 - Interface“, Webseite der Firma SPRUT Elektronik, Version: 20.05.2004, abgefragt: 06.06.2008.
http://www.sprut.de/electronic/interfaces/rs232/rs232.htm

[11] Peter P. Rössler: "ASICs im Internet - Das Projekt asix4web"; 2002.
https://www.ict.tuwien.ac.at/asicdesign/

[12] Frank Kesel, Ruben Bartholomä: „Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs“, Oldenbourg Wissenschaftsverlag, 2006, ISBN 3486575562, 9783486575569.

[13] Anil Telikepalli: „Sicherheit von flüchtigen und nicht-flüchtigen programmierbaren Bausteinen“, abgefragt: September 2008.
http://www.elektroniknet.de/home/bauelemente/fachwissen/uebersicht/aktive-bauelemente/programmierbare-logikasics/sind-fpga-designs-sicher/

[14] Xilinx, Inc.: “Spartan-3 Generation Configuration User Guide - Spartan™-3A, Spartan-3AN, Spartan-3A DSP, Spartan-3E, and Spartan-3 FPGA Families(with ISE 9.1i Design Examples)”, UG332, Version 1.2., 23 May, 2007.
http://www.xilinx.com/support/documentation/boards_and_kits/ug332.pdf

[15] Xilinx, Inc.: “ Virtex-II Pro™(P4/P7) Development Board User’s Guide”, DS-MANUAL-2VP4/7-FG456, Version 4.0, 5.März, 2003.
http://www.xilinx.com/support/documentation

[16] Xilinx, Inc.:” Virtex-II Pro and Virtex-II Pro X FPGA User Guide”, UG012, Version v4.1. 28 März 2007.
http://www.xilinx.com/support/documentation/boards_and_kits/ug012.pdf

[17] Xilinx, Inc.: "Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet", DS083, Version 4.6, 5.März, 2007.
http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf

[18] Xilinx, Inc.: "Xilinx ISE 9.2i Software Manuals and Help", 2007,
abgefragt: September 2008.
http://toolbox.xilinx.com/docsan/xilinx92/books/manuals.pdf

[19] Xilinx, Inc.: "Spartan-3A FPGA Starter Kit Board User Guide",
UG330 Version 1.3., Juni 21, 2007
http://www.xilinx.com/support/documentation/boards_and_kits/ug330.pdf

[20] Synplicity, Inc.: "Synplicity FPGA Synthesis", User Guide, Version Dezember 2005.
www.synplicity.com

[21] S. Hesse, G.J. Monkman, R. Steinmann, H. Schunk: „Robotergreifer", Hanser Verlag, Januar 2005, ISBN 3-446-22920-5

[22] Jeffrey Richter: "Microsoft .NET Framework-Programmierung in C#. Expertenwissen zur CLR und dem .NET Framework 2.0", 2.Auflage, Microsoft Press Deutschland, Juli 2006.

[23] Technik und Medien GmbH: „Das eLearning-Programm zur HANDHABUNGSTECHNIK",
http://www.bplusr.de/Produktblaetter/Handhabungstechnik_Produktinformation.pdf.