Projektarbeit

FPGA-Implementierung und serielle Befehle-basierte Steuerung einer Roboter-Kontrolleinheit

Gutachter: Prof. Dr. Günter Kemnitz
Betreuung: Dipl.-Ing Hossam Addeen Ramadam

Chen Liang
Pu Wang
Technische Universität Clausthal
03.Mai.2010

Inhaltsverzeichnis

1 Zielstellung

Humanoide Roboter vom Typ Robonova werden in diese Arbeit zu interaktiv steuerbaren Agenten weiter entwickelt. Dieser Roboter besteht aus 16 Servomotoren und ein Mikroprozess, der die Daten bearbeiten und an den Motoren unter entsprechenden Protokoll schicken kann.

Unsere Arbeit ist, um die Mikroprozessorsteuerung des Roboters durch Xilinx Spartan III, auch FPGA genannt, zu ersetzen, genauer zu sagen, dass der Benutzer die Befehlen durch das Interface von Matlab eingibt, die über Schnittstellenkommandos im FPGA umgesetzt, um synchroner Bewegungen mehrerer Gelenke von Arm zu steuern.

Die von Benutzer im Matlab eingegebene Befehlen werden von Matlab bearbeitet und in Zwischencode übersetzt , anschließend wird sie zum FPGA schickt.

Wenn FPGA die Zwischencode empfangen hat, wird sie bearbeitet und in einem Speicher gespeichert, dann wartet er auf einem Start Signal von Benutzer. Das nennt man „Befehle Unterladen“. Sobald FPGA alle Zwischencode gespeichert, d.h. vorbereitet, wird Benutzer durch einem Signal informiert, dann kann Benutzer Start Signal bestätigen.

FPGA wird Start Signal empfangen und alle bearbeitete Zwischencode an Servomotor schicken. Dann können die Bewegungen des Roboters durchgeführt werden.

Um dies ganze Idee zu testen haben wir zuerst einen Roboter Arm, der aus drei Motoren bestehen, bekommen. Wenn einige Grundfunktion realisiert sind, sind wir in Roboter eingestiegen.

2 Grundlagen der Arbeit

In diesem Abschnitt werden alle Dokumentationen beschrieben, welche Anteile wir uns im Rahmen dieser Arbeit gestellt haben.

2.1 Robonova-I, Servomotor

2.1.1 Robonova-I

Der große Menschheitstraum ist kurz davor, in Erfüllung zu gehen: Nie wider aufstehen! Ganz gleich, ob die Fernbedienung außer Reichweite liegt oder das Bier im Kühlschrank gerade viel zu weit weg ist, der HiTec Robonova-1 liest seinem Besitzer jeden Wunsch von den Lippen ab... ...

ROBONOVA-I ist ein“harter Kerl“. Spezialbehandeltes und eloxiertes Aluminium gibt den Verbindungselementen die notwendige Festigkeit und ROBONOVA-I das hochwertige Erscheinungsbild. Die Kunststoffelemente bestehen aus Material höchster Güte, die notwendige Robustheit im täglichen Einsatz ist somit sichergestellt. ROBONOVA-I muss keinem ,,Fight“ aus dem Weg gehen, Konzept und Auslegung basieren auf den in Robotikwettbewerben gesammelten mehrjährigen Erfahrungen der Entwickler.

Die Bauelemente von ROBONOVA-I bestehen aus:

- Roboter Robonova 1 mit 16 montierten HSR-8498HB Digital-Robotservos
- montiertes Elektronikboard MR-C3024 mit ATMega128L CPU
- Schnellladegerät für 230V
- Akkupack NiMH 6,0 V (1000 mAh)
- RS 232 Schnittstellenkabel zwischen PC und Roboter
- Software ROBOBASIC, ROBOSCRIPT
- Robo-Remocon Infrarot-Sensor mit Fernbedienung

Stromversorgung : Betrieben wird der ROBONOVA-I aus dem mitgelieferten 6 Volt/ 1000 mAh NiMH Akku, der auch im laufenden Betrieb durch ein geeignetes Ladegerät gepuffert werden kann. Mit einer Akkuladung kann ROBONOVA-I ca. 60 Minuten betrieben werden, die tatsächliche Betriebszeit variiert mit der Komplexität der Bewegungsabläufe.

Das Control Board:Gehirn und Herzstück ist der bekannte ATMEL ATMega128L Prozessor, dessen interner Speicherbereich
um ein 64k*8 EEPROM als Speicher für Robo-Script und Robo-Basic-Programme erweitert wurde.

2.1.2 Servomotor

Die „Muskeln“ von ROBONOVA-I sind Servos, die bei HiTEC, einem weltweit führenden Servohersteller, speziell für die Robotik entwickelt wurden.

Sie bieten die bekannten Stärken der HiTEC-Servos enthält:
• Stellkraft
• Stellgeschwindigkeit
• Stellgenauigkeit
• Langlebigkeit
und wurden um die speziellen Anforderungen der Robotik erweitert:
• 180 Grad Drehwinkel
• innovative Gehäusebauform
• Motion Feedback Funktion
• Datenrückmeldefunktion: Strom, Spannung

Roboter Robonova 1 mit 16 montierten HSR-8498HB Digital-Robotservos.HiTEC-Robotics Digitalservo HSR-8498HB für Robotikanwendungen, drehbares Lager am Gehäuseboden, doppelseitige Montage, Digital-MOS-FET-Verstärker, Variationsmöglichkeiten für den Einbau, 2 Kugellager, Kabelauslass links/rechts, Karbonite-Spezialgetriebe.

Der Getriebesatz von dieser Digitalservo enthält alle Zahnräder, die Achsen und einen Mitnehmer für das Potentiometer.Die sind sehr robust, aber nicht unzerstörbar. Sollte durch einen schweren Sturz oder Gewalt mal ein Servo ausfallen, kann mit ein paar Handgriffen das Getriebe ausgetauscht werden.

2.2 Befehlsreferenz Robobasic

Robo-Basic ist ein für die Programmierung des ROBONOVA-I optimierter Basic-Dialekt,Mit dieser eigenständigen Entwicklungsumgebung mit Editor und Compiler kann man das Verhalten seines Roboters in einem eigenen für ROBONOVA-1 programmiern. Zusätzlich zu den Grundbefehlen in BASIC gibt es Befehle für Synchrone Servobewegungen, Servo Point to Point Bewegungen und Servo Motion Feedback. Die Kommunikation zwischen PC und Roboter erfolgt mittels der RS 232- Schnittstelle.

RoboBASIC ist eine spezielle Programmiersprache zur Steuerung von Robotern. Die stellt eine Erweiterung der allgemeinen Grundprogrammiersprache mit Befehlen zur Steuerung von Robotern dar. ES enthält natürlich allen Befehlen über:
-Befehle, die der Deklaration/Definition dienen
-Ablaufsteuerbefehle
-Digitale Signaleingabe und -ausgabebefehle
-Befehle für den Speicher
-Befehle für das LCD
-Auf den Operand bezogene Operationen
-Parameter, welche die Motorgruppe zuordnen
-Befehle zur Klangkontrolle
-Befehle zur externen Kommunikation
-Verarbeitungsbefehle … …

Weil die Grammatik von roboBASIC auf allgemeinem BASIC basiert, ist sind die meisten Strukturen von roboBASIC ähnlich wie in der Grundsprache.Der Zeichensatz von roboBASIC ist aus englischen Buchstaben (A-Z, a-z), Nummern (0-9) und speziellen Symbolen zusammengesetzt. Formeln können aus Werten zusammengesetzt werden, welche von integrierten Invariablen, Variablen oder Konstruktionen aus allen anderen Nutzungsoperatoren berechnet werden.

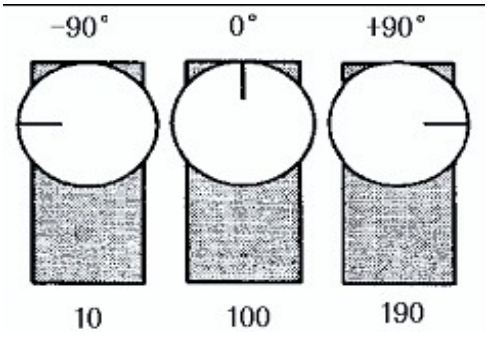
Ein Operator führt Rechenoperationen oder logische Vorgänge für einen gegebenen Wert aus. Ein arithmetischer Operator ist ein Symbol, das eine Verarbeitung ausführt. Wie in der allgemeinen BASIC-Sprache können Addition (+), Subtraktion (-), Multiplikation (*), Division (/) und Restebildung (% oder MOD) in roboBASIC genutzt werden. Ein relationaler Operator wird verwendet, um zwei Werte zu vergleichen. Ausgabe ist "TRUE" oder "FALSE". Ein Logikoperator wird für das Vergleichen von gemeinsamen Bedingungen verwendet. Das Ergebnis des Vergleichs wird mit „TRUE“ oder „FALSE“ beschrieben.

Diese Ausgabe wird für das Kontrollieren des Ablaufs des Programms in Form einer IF-Abfrage verwendet. Ein Logikoperator wird für das Vergleichen von gemeinsamen Bedingungen verwendet. Das Ergebnis des Vergleichs wird mit „TRUE“ oder „FALSE“ beschrieben. Diese Ausgabe wird für das Kontrollieren des Ablaufs des Programms in Form einer IF-Abfrage verwendet. Weil roboBASIC zur Steuerung von Hardware gestaltet ist, unterstützt roboBASIC keine Variablen oder Konstanten, die mit Zeichenfolgen verbunden sind, wie sie im Allgemeinen in BASIC verwendet werden.

Weil roboBASIC zur Hardwarekontrolle entwickelt wurde ist die Nutzung von Hexadezimalzahlen oder anderen Ausdrücken sinnvoller als die Nutzung von dezimalen Nummerntypausdrücken. In roboBASIC können Binärzahl (Bin), Oktonärzahl (Okt), Dezimalzahl (Dez), Hexadezimalnummer (Hexadezimalzahl) genutzt werden. In roboBASIC können Variablen als Bit-Einheiten gehandhabt werden. Um Variablen als eine Bit-Einheit zu nutzen wird der zeigende Bit-Operator „.“ Genutzt. Bei der Benutzung des Bit-Operators sind Bits 0~6 (Byte-Variable) und Bits 0~16 (ganzzahlige Variable) nutzbar. Allerdings sind nur Zahlen oder Konstanten mit diesem Operator nutzbar.

In unsere Arbeit brauchen wir besondere die Roboter-Kontrolleinheit Servomotoren und Gleichstrommotoren zu steuern.Bei Gleichstrommotoren kann die Kontrolleinheit Geschwindigkeit, Richtung und das Anhalten des Motors durch die Benutzung von digitalen Ein- und Ausgabebefehlen steuern.

Der Arbeitsbereich eines Servomotors reicht von -90° bis +90°. Bei der Benutzung von Schrittmotoren in roboBASIC werden Gradzahlen in Form von Numeralen zwischen 10 und 190 genutzt, weil die Kontrolleinheit keine negativen Werte verarbeitet.



Die allen Befelen, welchen wir brauchen die Servomotoren zu stueren,kann man die Befehlsstruktur und Erklärung des Befehls im Dokument „Robobasic Befehls-Bedienungsanleitung“ finden und nutzen.

2.3 Matlab ,Entwicklungsumgebung

In unsere Arbeit haben wir die Entwicklungsumgebung MATLAB als die Benuzter Schnittstelle ausgewälht. MATLAB ist ein interaktives System, das sich für wissenschaftliche numerische Berechnungen und zur Visualisierung in fast allen Bereichen technischer und naturwissenschaftlicher Disziplinen gewinnbringend einsetzen läßt.

MATLAB ist eine besondere Sprache zur Programmierung technisch-wissenschaftlicher Probleme, die es weitgehend erlaubt, diese und die zugehörigen Lösungen in der vertrauten mathematischen Notation auszudrücken. Dieser Entwicklungsumgebung ist Matrix-orientiert und erlaubt es, Probleme der Linearen Algebra in kompakter Form zu formulieren und zu lösen. Wie der Name MATLAB schon vermuten läßt, sind die Objekte, mit denen MATLAB arbeitet, Matrizen. Skalare werden folglich als Matrizen der Dimension 1 x 1 behandelt, Vektoren als Matrizen der Dimension 1 x *n* bzw. *m x* 1.

Matlab wurde Ende der 1970er Jahre von Cleve Moler an der Universität New Mexico entwickelt, um den Studenten die Fortan -Bibliotheken LINPACK und EISPACK für Lineare Algebra von einer Kommandozeile aus ohne Programmier-Kenntnisse in Fortran zugänglich zu machen. Zusammen mit Jack Little Bangert gründete Moler 1984 *The MathWorks* und machte Matlab zu einem kommerziellen Produkt, das zusammen mit einer ersten Funktions-Sammlung, der *Control System Toolbox*, vor allem in der Regelungstechnik viele Anwender fand. Die akademische Bindung ist in der Entwicklung und im Vertrieb von relativ preisgünstigen Studenten-Versionen bis heute erhalten geblieben und war möglicherweise auch die Grundlage für den Erfolg der Software neben anderen numerischen Plattformen wie MatrixX .

Matlab dient im Gegensatz zu Computeralgebrasystemennicht der symbolischen, sondern primär der numerischen (zahlenmäßigen) Lösung von Problemen. Die Software wird in Industrie und an Hochschulen vor allem für numerische Simulation sowie Datenerfassung, Datenanalyse und -auswertung eingesetzt.

Matlab ist auch die Basis für Simulink, ein anderes Produkt des Unternehmens The MathWorks, das zur zeitgesteuerten Simulation dient, und Stateflow , die ereignisorientierte Simulation benutzt wird, sowie für zahlreiche anwendungs- und domänenspezifische Erweiterungen.

Programmiert wird unter Matlab in einer proprietären ProgrammierSprache , die auf der jeweiligen Computer interpretiert wird. Kleinere Programme können als so genannte Skripts unktionen zu atomaren Einheiten verpackt werden, was das Erstellen von anwendungsorientierten Werkzeugkisten erlaubt.

Viele solcher Pakete sind auch kommerziell erhältlich. Durch die vereinfachte, mathematisch orientierte Syntax der Matlab-Skriptsprache und die umfangreichen

Funktionsbibliotheken für zum Beispiel Statistik, , Signal- und Bildverarbeitung ist die Erstellung entsprechender Programme wesentlich einfacher möglich als z. B. unter C. Ein Beispiel ist die Symbolic Toolbox zur Nutzung symbolischer Ausdrücke im Gegensatz zu mit Zahlen belegten Variablen. Ferner gibt es Schnittstellen, um C-Code einzubinden, sowie einen Übersetzer, mit dem aus einem Skript unabhängig von Matlab lauffähiger C-Code erstellt werden kann. Damit können mathematisch aufwendige Module für C-Projekte in der Matlab-Umgebung entwickelt und getestet werden.

MATLAB bietet aus der Objektorientieren Programmierungdie Konzepte von Klassen, Vererbung,Pakete,Pass-by-Value Aufrufen und Pass-by-Reference Aufrufen.

MATLAB kann Funktionen etwa in C oder Fortrfan aufrufen. Dazu wird eine sogenannte Wrapper-Funktion generiert, die die Übergabe von Parametern und Rückgabewerten steuert.
Bibliotheken in Java,ActiveX oder .NET können direkt aus MATLAB aufgerufen werden. Viele Bibliotheken in MATLAB, wie beispielsweise jene für die Anbindung von XML oder SQL, sind als Wrapper um Java oder ActiveX aufgebaut. Über den MATLAB Compiler und sogenannte Builder-Addons kann auch die umgekehrte Richtung genutzt werden und man aus JAVA oder .NET heraus Funktionen und Code in MATLAB aufrufen kann.
Als Alternative zur MuPAD-basierten *Symbolic Math Toolbox* (ebenfalls von MathWorks) kann MATLAB auch an Maple oder Mathematica angeschlossen werden.

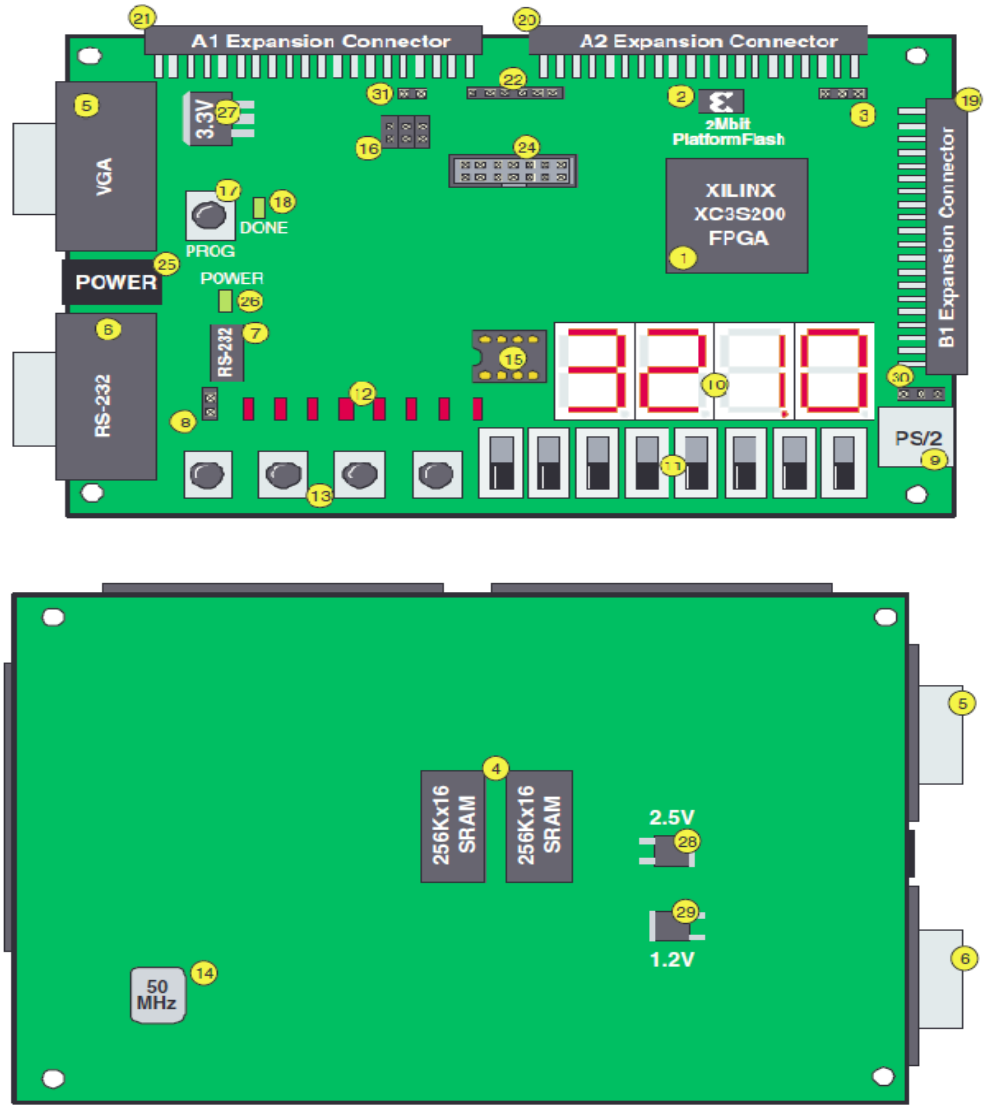
2.4 Xilinx Spartan III

Xilinx Spartan III Plattform ist die erste 90 nm FPGA der Welt und bietet mehrere Funktionen und Bandbreite, sie hat einen neuen Standard der programmierbaren Logik Industrie gebildet. Wegen zahlreichen Anwendungsbereiche und niedrigen Kosten hat diese Plattform sehr weit verbreitet.

Die Komponenten und Merkmale von Xilinx Spartan III:

1. 200,000-gate Xilinx Spartan III XC3S200 FPGA in a 256-ball thin Ball Grid Array Package
2. 2Mbit Xinlinx XCF02S Plattform Flash, in-system programmierbare Konfiguration PROM
3. 3-bit, 8-color VGA Display Port
4. 9-pin RS-232 serielle Port
5. 8 LEDs, 8 Switch-Schalter, 4 BTN-Taste
6. PS/2 Mouse/Keyboard
7. 50 Mhz Clock Frequenz

   .....



2.5 VHDL ,VHDL Sprache

VHDL ist die Abkrüzung von „Very High Speed Integrated Circuit Hardware Description Language“ , es ist eine Hardwarebeschreibungssprache. VHDL wurde in den frühen 1980er Jahren entwickelt und ist das Produkt von Normierungsbestrebungen eines Komitees, in dem die meisten größeren CAD-Anbieter und CAD-Nutzer, aber auch Vereinigungen wie die IEEE, vertreten waren.

Zusammenhang des VHDLs unter Hardware umfasst ein komplexes System, wie z.B. einen ganzen Computer, aber auch ein kleines einzelnes Element, wie z.B. ein logisches Grundgatter. Die Beschreibung eines logischen Grundgatters z.B. kann aber aus nur einer boolschen Gleichung bestehen; also einer kurzen und sehr konkreten unktionsbeschreibung. VHDL ist also eine Sprache, mit der man (digitale) Hardware im allgemeinen Sinne nachmodellieren kann vergleichbar mit einer Programmiersprache,

mit der es einfach möglich ist, komplizierte digitale Systeme zu beschreiben. Darüber hinaus gibt es sprachliche Erweiterungen in Form von VHDL-AMS, mit der auch analoge Systeme beschrieben werden können.

Bei VHDL arbeitet man nicht mit einzelnen elektronischen Bauteilen, sondern beschreibt das gewünschte Verhalten einer Schaltung auf einer höheren Abstraktionsebene. VHDL ermöglicht das schnelle Entwickeln großer und komplexer Schaltungen (z. B. Mikroprozessor mit über 20 Mio. Transistoren), die hohe Effizienz erfordern (zeitlich wie ökonomisch) und unterstützt den Entwickler bei allen Arbeiten. So kann ein System simuliert, verifiziert und schließlich eine Netzliste erstellt werden.

Aus der Netzliste können Masken für die Herstellung von MPGAs (mask programmable gate array) oder ähnlichen LSI (Large scale integration)-Chips produziert werden oder sie kann (nach Konvertierung in einen geeigneten Bitstream) direkt in ein FPGA (Field Programmable Gate Array) oder CPLD (Complex Programmable Logic Device) geladen werden.

3. Arbeitsreihenfolge

Ein Benutzer kann Robobasic Programmiersprache verwenden, um im „Benutzer_Schnittstelle“ von Matlab zu programmieren. Die verwendete Befehle sind: mov, speed. Dazu braucht man für Matlab noch zwei zusätzliche Teilprogramme: start und Endall, um serielle Schnittstelle zu öffnen, und um die von Benutzer eingegebene Parametern zu speichern, senden. Die Reihenfolge von der Befehlen „mov“ und „speed“ ist eigentlich uninteressant, Teilprogramm „Endall“ von Matlab wird diese Reihenfolge sortieren, denn diese Reihenfolge muss für VHDL-Programm geeigneten.

Wenn die Befehle von „Benutzer-Schnittstelle“ geordnet sind, werde die Parametern von „speed“ optimiert, damit es alle Motoren gleichzeitig anhalten lässt. Das ist in 3.3 geklärt

Am Ende wird Matlab alle Parametern von „Benutzer-Schnittstelle“ in einer Matrix speichern, in dieser Matrix enthält die Informationen für:

- Reihe,
- „speed“ oder „mov“,
- Gruppe Nummer,
- Winkeln für jeder Motor,
- Check_sum(wird in 3.5.3 geklärt).

Die letzte Aufgabe von Matlab ist um diese Matrix an FPGA zu schicken.

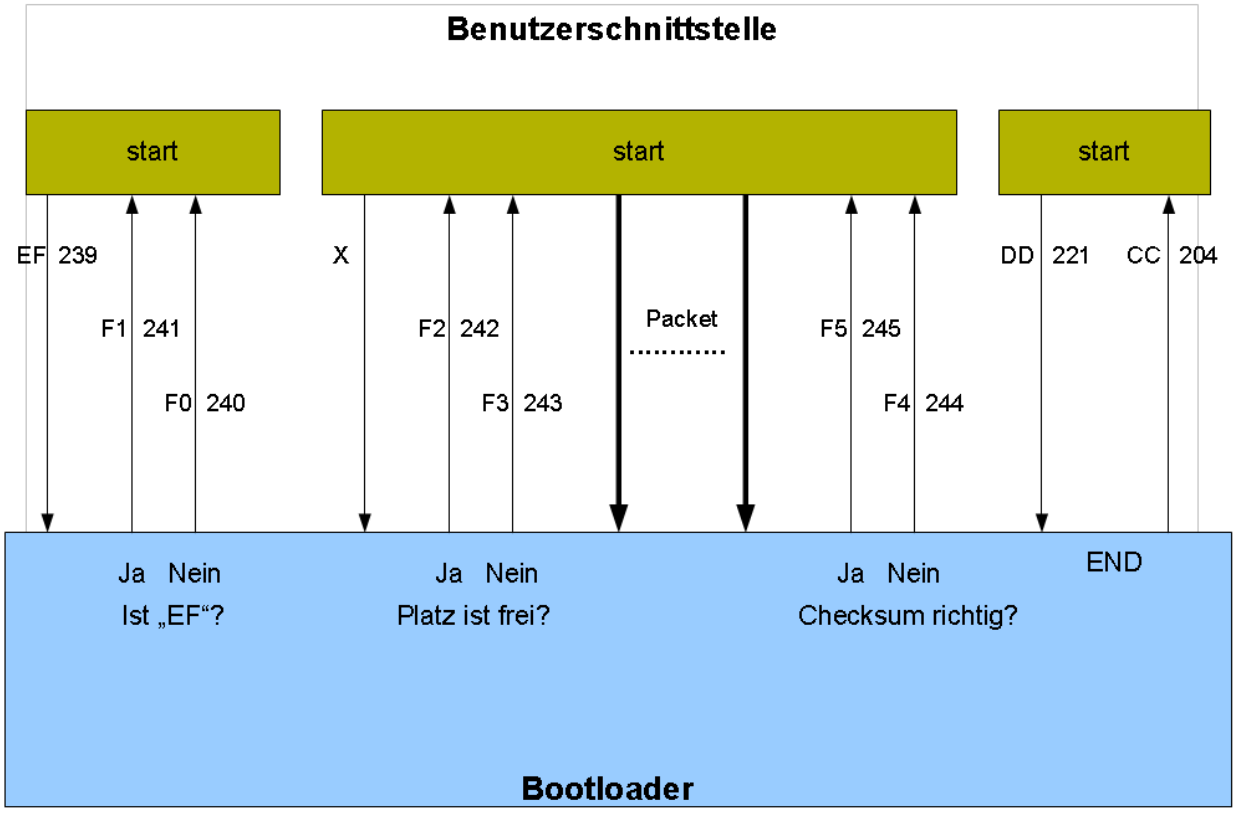
Wenn FPGA alle Parametern von dieser Matrix empfängt, werden die sofort in einem „3D-Ram“ gespeichert, während des Speichern muss VHDL-Programm überprüfen, ob die Zahlen richtig empfängt ist, wenn ein Fehler bei der Transformation auftritt, wird VHDL bei Matlab melden, dann schickt Matlab diese Zahl noch mal an FPGA.

Jetzt sind alle Parametern in diesem „3D-Ram“ richtig fertig gespeichert, dann wartet FPGA auf ein Start-Signal von dem Benutzer. Wenn der Benuzter „SWT“ oder „BTN“

bestätigt, wird VHDL jede Zahl von dem Speicher auslesen, bearbeiten, anschließende an Motoren senden.
Die Technik und Methoden, die diese Funktion realisieren können, werden in diesem kommenden Abschnitt geklärt.

3.1 Blockschaltbild des Systems

Die Programm besteht aus Beutzerschnittstelle und Bootloader. Wir haben zuerst ein Blockschaltbild aus unsere Idee konstruiert. Es folgt:



– „EF“: um zu fragen, ob Leitung schon offen ist;
– „F1“, „F0“: die Antwort von FPGA;
– „X“: um zu fragen, ob es genügende Platz vorhanden ist;
– „F2“, „F3“: die Antwort von FPGA;
– „F5“, „F4“: die Antwort von FPGA, um zu überprüfen, ob die Zahlen richtig gesendet, oder nicht;
– „DD“: eine Meldung von Matlab, um zu sagen, dass die alle Zahlen gesendet sind;
– „CC“: die Antwort von FPGA.

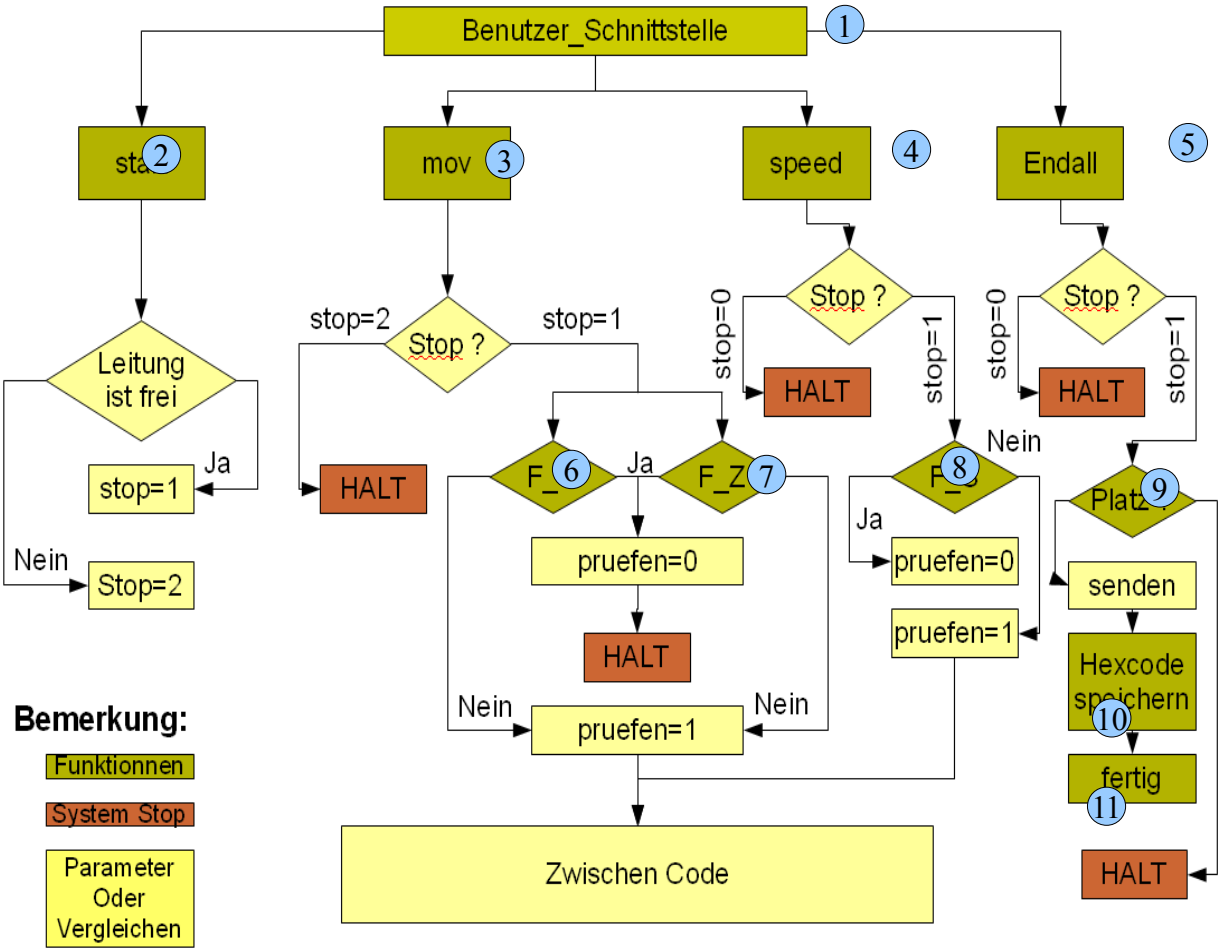
Die Kommunikation zwischen Benutzerschnittstelle und Bootloader wird hier durch Hexadezimalnummer als „Austausch“ oder „Übertragung“ von Informationen beschrieben. Alle HEX sind in diesem Zusammenhang eine zusammenfassende Bezeichnung für bestimmte Funktionsweise. Mit jedem Zahl sind die beiden Stellen ein Geben und Nehmen frühen. Zwischen die Hexadezimalzahlen ist ein Packet für Datenfluss.

3.2 Benutzerschnittstelle

Hier wird ein Benutzer in Matlab programmieren, aber statt Matlab Schlüßelwort von Matlab verwendet Benutzer Robobasic Programmiersprache, z.B. „speed“, „move“ usw.. Die Hauptfunktion von Matlab besteht aus zwei Teile, nämlich:

– Robobasic Programmiersprache in Matlab Programm zu übersetzen und anschließend Zwischencode zu erzeugen, die Robot-Arm steuert.
– Zwischencode an FPGA hochzuladen

Blockschaltbild für Matlab Programme:



1. Hier kann man Robobasic-Befehl eingeben, aber man muss darauf achten, dass man mit „Start“ anfangen und mit „Endall“ beenden.
2. In diesem Teilprogramm wird die Leitung zwischen Matlab und FPGA geöffnet und

anschließend geprüft, ob diese Leitung richtig funktioniert.

3. Es wird die von Benutzer eingegebene Parameter geprüft, wenn die Angabe in ihre entsprechenden Definitionsbereich liegt, dann kann es für Zwischencode erzeugt werden.
4. Analog für „speed“.
5. Wenn alle Befehle von dem Teilprogramm „Benutzer-Schnittstelle“ übersetzt und entsprechenden Zwischencode erzeugt werden, wird die Zwischencode in richtige Reihenfolge, die VHDL-Programm „Bootloader“ akzeptiert, geordnet. Diese geordnete Code wird an FPGA gesendet, anschließend wird diese Code in einem Datei gespeichert.
6. Dieser Teilprogramm wird für die Angabe von Benutzer überprüft. Analog für 7, 8.
9. Hier wird Matlab eine Nachfrage an FPGA gesendet, um zu fragen, ob „Bootloader“ noch genüngende Speicherplatz vorhanden ist.
10. Die geordnete Code wird in einem Datai gespeichert.

11. Am Ende wird Matlab Benutzer informiert, dass alle Befehlen erfolgreich an FPGA gesendet und gespeichert.

3.3 Bearbeiten der Zwischencode

In dem von Benutzer bearbeiteten Teilprogramm „Benutzerschnittstelle“ könnte 3 verschiedene Möglichkeiten passieren:

1. Es wird kein „speed“ eingegeben:

```
start;
mov(Parameter, Parameter,....);
mov(Parameter, Parameter,....);
mov(Parameter, Parameter,....);
… …
Endall;
```

2. „speed“ wird im Mittel eingegeben:

```
start:
mov(Parameter, Parameter,....);
speed(Parameter);
mov(Parameter, Parameter,....);
mov(Parameter, Parameter,....);
… …
Endall;
```

3. „speed“ am ganz Anfang:

start;
speed(Parameter);
mov(Parameter, Parameter,....);
mov(Parameter, Parameter,....);
mov(Parameter, Parameter,....);
… …
Endall;

Für 1. Möglichkeiten wird die Geschwindigkeit der allen 3 Motoren als Standard-Geschwindigkeit definiert, also „speed(5)“. Analog für alle Mov-Befehle, die keine Geschwindigkeitsdefinition haben, werden ihre Geschwindigkeit auch als Standard-Geschwindigkeit definiert, z.B. wie 2. Möglichkeit. Wenn die Geschwindigkeitsdefinition am ganz Anfang des Programm und später nicht mehr eintritt, wird die Geschwindigkeit aller Mov-Befehlen als dies „speed“ definiert.

Im Bootloader werden alle Parameter der Befehlen nur in diese Reihenfolge in einem Zwischenspeicher gespeichert:

„1“ „128“ „speed“ „Pruppe“ „Winkel-1“ „Winkel-2“ „Winkel-3“ „Winkel-4“ „Winkel-5“ „Winkel-6“ „Check“
„2“ „128“ „mov“ „Pruppe“ „Winkel-1“ „Winkel-2“ „Winkel-3“ „Winkel-4“ „Winkel-5“ „Winkel-6“ „Check“
„3“ „128“ „speed“ „Pruppe“ „Winkel-1“ „Winkel-2“ „Winkel-3“ „Winkel-4“ „Winkel-5“ „Winkel-6“ „Check“
„4“ „128“ „mov“ „Pruppe“ „Winkel-1“ „Winkel-2“ „Winkel-3“ „Winkel-4“ „Winkel-5“ „Winkel-6“ „Check“
„5“ „128“ „speed“ „Pruppe“ „Winkel-1“ „Winkel-2“ „Winkel-3“ „Winkel-4“ „Winkel-5“ „Winkel-6“ „Check“
... ...

D.h., von Matlab soll zuerst ein Befehl für „speed“ an FPGA gesendet werden, dann ein Befehl für „mov“, dann wieder ein Befehl für „speed“, usw.. Deswegen, wenn ein Benutzer egal in welchen Reihenfolge für „speed“ und „mov“ eingegeben hat, werden solche Befehlen in Teilprogramm „Endall“ von Matlab in Bootloader akzeptierte Reihenfolge geordnet, diese Vorsortieren wird die VHDL-Programm vereinfacht.

Andrer Problem ist es, wie können wir die 3 Motoren gleichzeitig ihre Bewegungen beenden lassen. Zum Beispiel, ein Benutzer hat die Befehlen so gegeben:

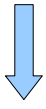
speed(5);
mov(30, 60, 90);

Das bedeutet, Motor 1 soll sich zu die Position 30 Grad bewegen, Motor 2 soll sich zu die Position 60 Grad bewegen, Motor 3 soll sich zu die Position 90 Grad bewegen, zwar mit die Geschwindigkeit „5“. Wenn alle 3 Motoren gleichzeitig ihre Positionen erreichen möchten, muss Motor 1 langsamer sein, um auf die andere zwei Motoren zu warten, und die andere zwei Motoren soll schneller sein, deswegen haben wir einen Algorithmus für „speed(Parameter)“ entwickelt:

speed-Parameter = $\frac{Mov.Parameter}{180\,Grad} * Speed.Parameter$,

also, vorheriger Beispiel verwenden wir noch mal:

speed(5);
mov(30, 60, 90);

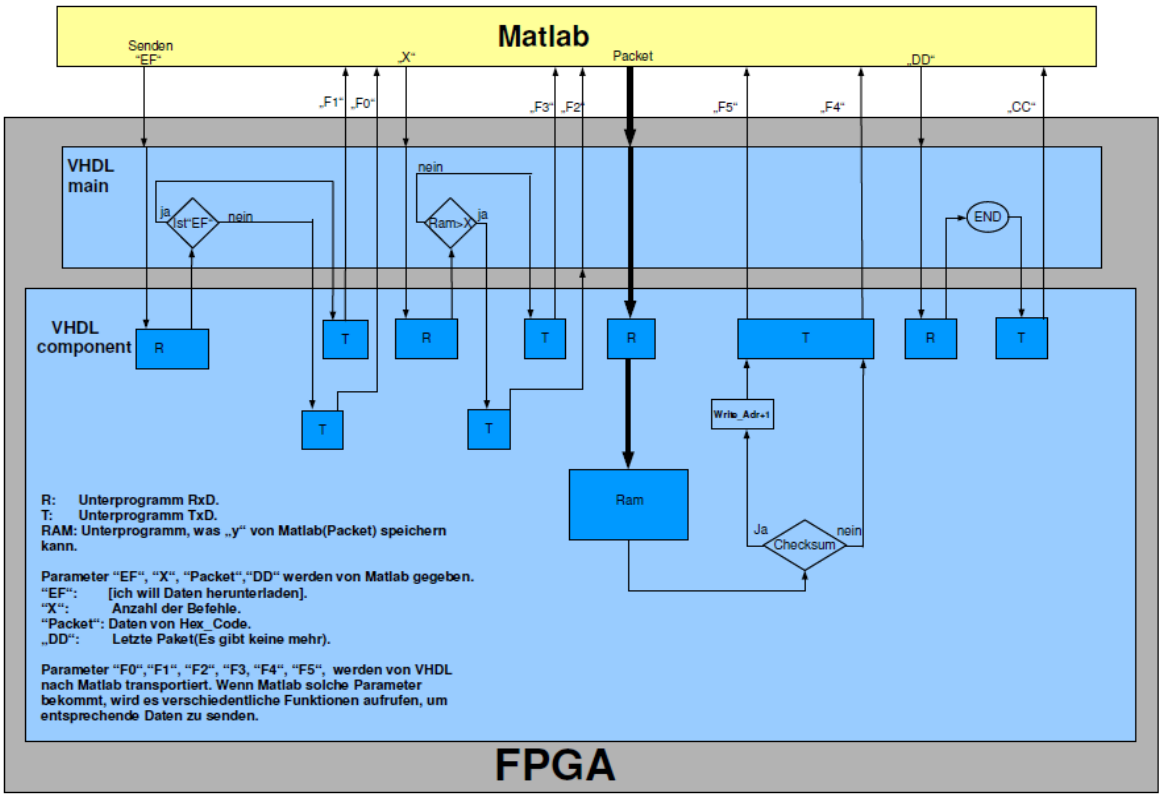


speed($\frac{30}{180}*5$, $\frac{60}{180}*5$, $\frac{90}{180}*5$);
mov(30, 60, 90);

Im idealen Fall werden 3 Motoren durch diese Umwandlung ihre Positionen gleichzeitig erreichen.

3.4 Bootloader

Bootloader ist mit VHDL beschriebt und an FPGA(Spartan 3) untergeladen. Mit Bootloader kann die FPGA die Mov- oder Speed-Befehl von Benutzerschnittstelle empfangen und im Ram Speichern, zum letzt die Befehle für Servos übersezten und senden. Die Funktionensweise kann man durch folgt Diagrmm weiter verstehen.



Bootloader ist durch die Serielle Schnittstelle (RS-232) von FPGA mit Benutzerschnittstelle von PC zu kommunizieren.RS232 benutzt für die Datenübertragung ein einfaches asynchrones serielles Verfahren.Seriell bedeutet, dass die einzelnen Bits des zu übertragenden Bytes nacheinander über eine einzige Datenleitung geschoben werden. Asynchron heisst, dass es keine Taktleitung gibt, die dem Datenempfänger genau sagt, wann das nächste Bit auf der Datenleitung liegt.
Es gibt insgesamt 4 Componenten im Main Programm:

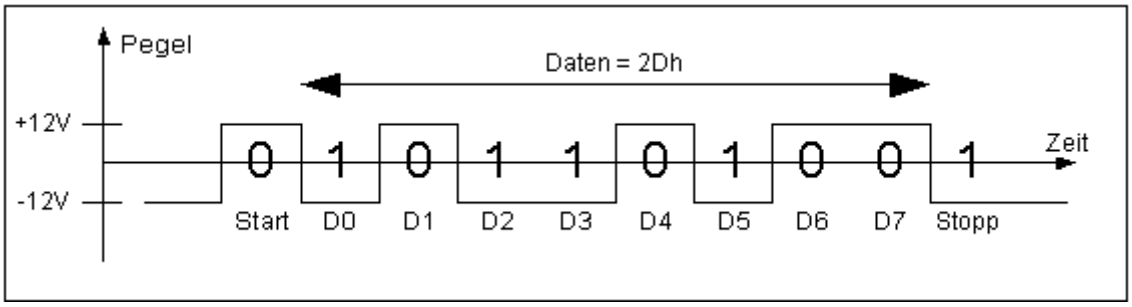
- TXD.vhd
- RXD.vhd
- Ram.vhd
- Senden .vhd

Jeder Componente hat seine bestimmte Funktionen realisiert. Danach wird die Funktionensweise in jeder Componente erklärt.

3.5 Befehle an Servomotor senden

3.5.1 TXD

Die Daten-Ausgangs-Leitung des Bootloader wird als **TXD out** bezeichnet. Sie wird mit der Daten-Eingangs-Leitung des Benutzerschnittstelle **TXD in** direkt verbunden. Die Datenleitung liegt im Ruhezustand auf -12V, also auf dem logischen Pegel '1'. Die Datenübertragung erfolgt byteweise. Bevor ein Byte übertragen werden kann, muß der Benutzerschnittstelle mit dem Bootloader synchronisiert werden. Das erfolgt durch das Senden eines dem Datenbyte vorgesetzten Startbits mit dem Wert '0'.



Am Begin des Startbits ändert sich der Pegel der Datenleitung von -12V auf +12V. Das ist für den Benutzerschnittstelle das Zeichen dafür, dass der Datentransfer beginnt. Der Bootloader legt nun in einem festen Zeitabstand die einzelnen Datenbits des Datenbytes auf die Datenleitung. Am Ende fügt er mindestens ein Stoppbit mit dem Wert 1 an. Damit ist die Datenleitung wieder im Ruhezustand. m oben dargestellten Bild wird die Bitfolge '10110100' übertragen, da die Übertragung mit dem LSB (also dem letzten Bit) voran erfolgt, ist die wirklich übertragene binäre Zahl B'00101101' oder im hexadezimalen Code 0x2D bzw. 2Dh.

Der Benutzerschnittstelle liest immer in der Mitte der Bits den Spannungspegel aus der Datenleitung, und empfängt so das Byte Bit für Bit.

Das funktioniert natürlich nur, wenn Bootloder und Benutzerschnittstelle mit der gleichen Geschwindigkeit die Bits schreiben und lesen. Deshalb ist es unbedingt erforderlich, dass die beide vorab auf eine gleiche Datenrate eingestellt werden. Eine Abweichung der Datenraten um mehr als 5% führt zu Lesefehlern, da der Benutzerschnittstelle dann bei den letzten Bits vor bzw nach dem Bit liest.

Die Datenrate wird in Baud (also in Bit pro Sekunde) angegeben. Dabei werden alle Bits (auch Start- und Stopp-Bit) gezählt, und Lücken zwischen den Bytetransfers ignoriert. Deshalb ist die Bautrate der reziproke Wert der Länge eines Bits. Als Datenraten sind

folgende Werte üblich:

Datenrate[baud]	300	2400	9600	19200	38400	57600	115200
Bitlänge	3,33ms	417µs	104µs	52,08µs	26,04µs	17,36µs	8,68µs

Wir haben hier die Baudrat als 19200 ausgewählt. Nach dem Stoppbit kann sofort wieder eine neue Übertragung mit einem Startbit beginnen. Man kann dem Empfänger aber auch etwas mehr Zeit geben, indem man 1,5 oder 2 Stoppbits sendet.

3.5.2 RXD

Falls der Benutzerschnittstelle seinerseits Daten zurück senden soll, wird dafür eine weitere Leitung benötigt. Die Daten-Ausgangs-Leitung des Benutzerschnittstelle wird als **RXD out** bezeichnet. Sie wird mit der Daten-Eingangs-Leitung des Bootloader **RXD in** direkt verbunden.

Wie sich erkennen von TXD lässt, sind die Datenleitungen invertiert.Jede Datenübertragung beginnt auch mit einem Startbit, welches dazu dient, Bootloader und Benutzerschnittstelle zu synchronisieren. Der Pegel der Datenleitung wechselt also von logisch 1, dem Ruhezustand, auf logisch 0.

Anschließend kann das Paritätsbit (Paritybit) folgen, das dazu dient, mögliche Übertragungsfehler zu entdecken. Man unterscheidet hierbei:

- Even-parität
- Odd-parität

Ist als Parität "even" gewählt, wird das Paritätsbit gesetzt, wenn die Anzahl der übertragenen Einsen in den Nutzdaten ungerade ist. Andernfalls ist es nicht gesetzt. Wählt man "odd" als Parität, dann wird das Paritätsbit gesetzt, wenn die Anzahl der übertragenen Einsen in den Nutzdaten gerade ist.

3.5.3 3D-RAM

Wir brauchen auch ein drei Diemensional Ram, um die Data von Benutzerschnittstelle zu spreichen.

XST stützt mehrdimensionale Array Artrn bis drei Maßen.Array können Signale, Konstanten oder VHDL Variablen sein. Sie können Anweisungen und arithmetische Betriebe mit Array tun.Sie können mehrdimensionale Array zu den Funktionen auch führen,und benutzen Sie sie in den instantiations.

Die Array muß in allen Maßen völlig begrenzt werden. Ein Beispiel wird unten gezeigt:

```
subtype WORD8 is STD_LOGIC_VECTOR (7 downto 0);
type TAB12 is array (11 downto 0) of WORD8;
type TAB03 is array (2 downto 0) of TAB12;
```

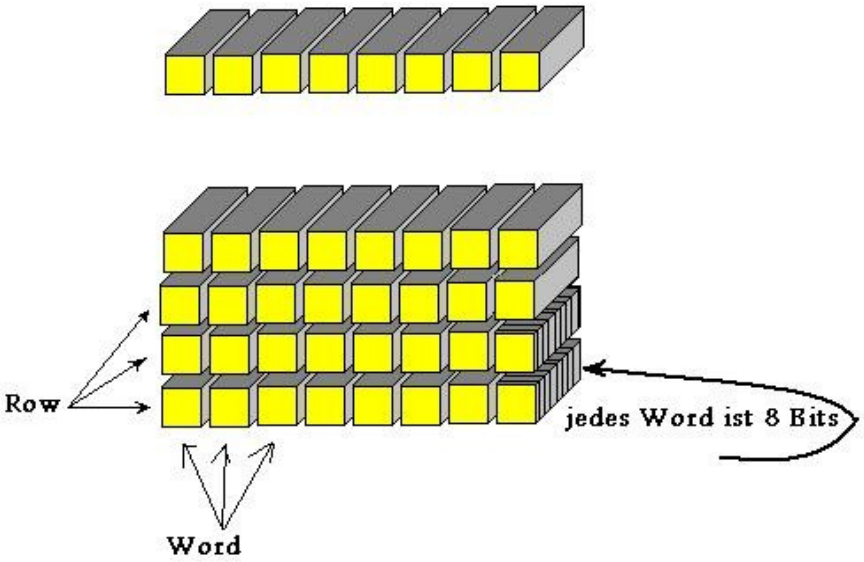
Sie können eine Array als Matrix, wie im folgenden Beispiel auch erklären:

subtype TAB13 is array (7 downto 0,4 downto 0)
of STD_LOGIC_VECTOR (8 downto 0);

In unsere drei dimensional Ram haben wir mit folgende Befehl definiert:

TYPE Row IS ARRAY (10 DOWNTO 0) OF std_logic_vector (7 DOWNTO 0);
TYPE mem_type IS ARRAY (12 DOWNTO 0) OF Row;
SIGNAL mem : mem_type;

Das bedeute, wir haben insgesamt 11 Zeilen und 13 Spalten und es besitzt 8 bits für jeder Zeile und Spalte. Wie folgt :



Die Länge von Row und Word kann man selbst definieren. Und mit Signal mem(Row) (Word) kann man irgendwelche data abholen und verarbeiten.

Im Ram wir haben auch die Funktion „checksum rechnen“ eingefügt. Es gibt zuerst ein checksum Wert in Benutzerschnittstelle gerechnet und in Ram gespeichert. Im Ram rechnet die checksum wert mit gleiche Algorithmus und vergleichen die beide. Wenn die beide checksum Wert gleiche ist, das Programm läuft weiter. Sonst die Bootloader sagt die Benutzerschnittstelle bescheid und die Benutzerschnittstelle sendet die Data wieder noch mal.

Es gibt zwei „Speed“ und „Drehwinkel“ Case im Ram. Mit die beide kann es die genaue geschwindigkeit und drehwinkel als Dezimalzahlen bestimmen. Und die Wert hängt von die Befhel aus Benutzerschnittstelle ab. Zuletzt sendet es die geeignete Datenbits von jede Dezimalzahl nacheinander zu Servos.

3.5.4 Senden an Motoren

Jetzt wird jede Zahl von dem Speicher ausgelesen und bearbeitet, denn man das Protokoll folgen muss, wenn man Motor steuern will. Die Befehl, die von Motor akzeptiert, soll so formuliert:

PC -> Servo

0x80	command	param1	param2	checksum	0x00	0x00

Servo -> PC (only by Get Params)

0x80	command	param1	param2	checksum	return1	return2

Jeder Motor kann sich von 0 Grad bis 180 Grad drehen, das entspricht die interne Position des Motors von 600 bis 2400, diese Dezimalzahl wird zu vier stelligen Hexadezimalzahl umwandelt, die erste zwei Stellen nennt man Parameter-1, letzte zwei Stellen nennt man Parameter-2. Im Speicher stehende Zahlen sind für Winkel oder Geschwindigkeit, z.B. 100 Grad entspricht die Motor-Position 1400, Parameter-1 ist 05, Parameter-2 ist 78.

Dazu braucht man noch „command“, F6 für „mov“ und E9 für „speed“.

„checksum“ ist so definiert:

$$\text{Checksum} = (256 - ((0x80 + \text{command} + \text{param1} + \text{param2}) \bmod 256)) \bmod 256$$

Ein Beispiel:
„70“ für Winkel ⟹ 0x80 F6 05 78 D
„70“ für Geschwindigkeit ⟹ 0x80 E9 00 50 47

Jede Hexadezimalzahl muss dann in VHDL wieder in Dezimalzahl umwandelt, dann in 8 stellige Binärzahl übersetzt, anschließende kann diese Binärzahl an Motor gesendet dürfen.

Die senden hat zuerst 8 bits von Ram empfangt und dann zu Servos senden.Weil wir 3 Servos zu testen genommen haben , dann es auch 3 senden componenten aufgebaut hat. Das bedeutet jeder send bedient ein Servo, um die definierte Geschwindigkeit und Dehrwinkel zu übertragen.

Das Installationssatzbrett des Starter-Spartan-3 hat drei 40-Pins Expansion Stecker, die A1, A2 und B1 beschriftet werden.Jedes Schnittstelle bietet etwas Fähigkeit an, das FPGA auf dem Installationssatz-Brett des Starter-zu programmieren Spartan-3.Zum Beispiel liefert Port A1 Zusatz logik, um die FPGA und der Plattform grelle JTAG Kette zu fahren. Ähnlich stellen A2 und B1 Anschlüsse für Haupt- oder Sklaveserienmodus Konfiguration zur Verfügung. Schließlich bietet PortB1 auch Meister oder Sklaven paralleler Konfiguration Modus an.

Jede 40-Pins Expansion überschrift, gezeigt in die Bild, Gebrauch 0.1 Zoll (Mil 100) BAD-Abstand.Pin 1 von jedem Stecker ist immer Boden.Ähnlich ist Pin 2 immer das +5V DC, das vom Schaltung Spg.Versorgungsteil ausgegeben wird.Pin 3 ist immer der Ausgang vom +3.3V DC Regler.Wir haben B1 als die Schnittstelle für Data übertrgung.

3.5.5 Sendetakt

Um die Zahlen von 3D-Ram an Motor senden, brauchen wir RS232 Protokoll. Die Übertragung einer Zahl erfolgt byteweise. Während einer Übertragungspause ist das Sendesignal TxD='1'. Jedes Datenpaket beginnt mit einem Startbit='0' gefolgt von den 8 Datenbits, optional einem paritätsbit und einem Stopbit='!'. Danach kann sich eine Pause(Übertragungsleitung gleich Eins) oder das Startbit der nächsten Übertragung anschließen. Das Paritätsbit berechnet sich aus der Modolo-2- oder EXOR-Summe der Daten bits und dient zur Fehlererkennung.

Der Basistakt CLK von Spartan-III hat eine Frequenz von 50MHZ, dieser Takt wird üblicherweise heruntergeteilt auf einem Takt mit der 16-fachen Baudrate und einem Tastverhältnis 1:1:

```
process(CLK)
        variable Teiler: integer range 0 to cTeiler-1:=0;
begin
        if CLK'event and CLK='1' then
                if Teiler=cTeiler-1 then
                        Teiler:=0;
                        Baudtakt_x16 <= not Baudtakt_x16;
                else
                        Teiler:=teiler+1;
                end if;
        end if;
end process;
```

Gebräuchliche Baudraten und Teilerkonstanten zeigt nachfolgende Tabelle:

Baudrate	cTeiler (Basistakt 50MHZ)
2,4 kBaude	**648**
4,8 kBaude	**324**
9,6 kBaude	**162**
19,2 kBaude	**81**

Der Takt Baudrate_X16 wird in einem weiteren Teiler (4-bit-Zähler) durch 16 geteilt und als Takt für den Sendeautomaten genutzt.

3.6 Funktionsweise

Laden Sie zuerst die VHDL Programm “Bootloader_Roboter_X” in FAGA unter. . Dann öffnen Sie alle M-File in “Matlab_benutzer” mit Matlab und die File “rs232” durchführen, um die serieschnittstelle RS232 zu öffen. Dann gehen Sie zu M-File “Benutzers_Schnittstelle”, hier kann man die gewunschte Geschwindigkeit und Mov-Winkel eingeben, zum Beispiel:

```
start;
mov('G6A', 100,  58, 135, 160, 100, 100);
speed('G6D', 80);
mov('G6D', 100,  58, 135, 160, 90, 100);
speed('G6B', 100);
mov('G6B', 100,  30,  80, 0, 0, 0);
mov('G6C', 100,  30,  80, 0, 0, 0);
Endall;
```

In Mov(... …) die erste Stell ist die Auswahl von Robonova Gelenken. Weil wir nur ein Robot Arm
zum testen genommen, hier kann man nur „G6A“ eingeben. Dann von zweite bis siebte sind die
Mov-Winkel.Aber für unsere Robot Arm brauchen wir die zweite bis vierte zum testen.

In speed(... …) die erste Stell ist auch für Robonova Gelenken auswahl,und danach ist die gewunschte Geschwindigkeit.

Dann führen Sie das Programm durch. Die Programm sendet dann die untere Wert in die Tabelle zu Bootloader senden durch RS232 und in 3D-Ram gespeichert.

Zwischencode, durch Übersetzen und Sortieren:

1	128	2	1	3	2	4	4	3	3	149
2	128	1	1	100	58	135	160	100	100	14
3	128	2	4	44	26	60	71	40	44	159
4	128	1	4	100	58	135	160	90	100	4
5	128	2	2	56	17	44	0	0	0	247
6	128	1	2	100	30	80	0	0	0	83
7	128	2	3	56	17	44	0	0	0	247
8	128	1	3	100	30	80	0	0	0	83

Wenn LD0 leuchtet, das bedeutet das Programm funktionert gut.Sie können weiter die Start Button SW0 einschalten. Die Robot Arm bewegt dann als die erst Geschwindigkeit und bis die erste Mov-Winkel dreht. Dann kann man die BTN3 weiter drücken, die Robot Arm nimmt die zweite Geschwindigkeit und Mov-Winkel und bewegt. Die Led addiert gleichzeitig 1,d.h LD2 leuchtet jetzt. Als diese Funktionsweise, Wenn man die nächste Befehle von Benutzerschnittstelle abholen will, dann drücken Sie weiter BTN3.

Wir haben noch eine spezial Funktion in File“automatische_Bewegung” für Bootloader aufgebaut. Mit das kann die Roboter Arm die Befehl in Benutzerschnittstelle kontinuierliche lesen. Wenn man das start Butten SW0 eingeschaltet, die Robote Arm bewegt automatisch von die erst bis letzte Mov-Winkel in Benutzerschnittstelle.

4 Zusammenfassung aller Programme

Matlab:
- Benutzerschnittstelle.m
- start.m
- Endall.m
- mov.m
- speed.m
- entfernen.m
- rs232.m
- platzfrei.m
- leitung_nachfragen.m
- fehler_gruppe.m
- fehler_zahl.m
- fertig.m
- speichern.m

VHDL:
- main.vhd
- main.ucf
- RxD.vhd
- senden1.vhd
- senden2.vhd
- senden3.vhd
- RAM1.vhd
- Txd_1.vhd

5 Fazit

Durch diese Arbeit haben wir gelernt, wie man Zahlen mit der Hilfe von Matlab an FPGA und von FPGA an Servomotor sendet. Synchronisation speilt hier eine sehr wichtige Rolle, obwohl für Zahlen senden, als auch für Speichern.

Am Ende hat unsere Arbeit das realisiert:
- Benutzer programmiert Robobasic in Matlab.
- Matlab übersetzt Robobasic.
- Matlab erzeut Zwischencode von den Parameter, die durch Benutzer eingegeben ist.
- Matlab sortiert die Zwischencode in eine spezielle Reihenfolge.
- Matlab sendet diese Code an FPGA.
- FPGA speichert die Code in einem 3D-Ram.
- Wenn Benutzer bestätigt, werden die Code von 3D-Ram an den Servomotoren gesendet.
- Servomotor bewegen sich, so wie die Befehle beschreiben.