Hochschule für Wirtschaft und Recht Berlin (HWR)

Fachbereich Berufsakademie

Studienarbeit

Studiengang Informatik / IT07

Thema:	Erstellung einer Mikrokontrollerschaltung mit Firmware zur Steuerung von Servomotoren über USB/RS232, Treiberbibliothek für Windows und Linux und Demo-Applikationen
eingereicht von:	Christopher-Eyk Hrabia
Matrikelnummer:	613360
eingereicht am:	Berlin, den 18. Januar 2010
Wörteranzahl:	5437
Verantwortlicher Betreuer:	Herr Prof. Dr. Holznagel

Verfasser

Verantwortlicher Betreuer

Ziel dieser Studienarbeit ist es einen Roboterarm mit fünf Freiheitsgraden, welcher an der BA-Berlin entwickelt wurde, mit einem Servo-Steuerungs-Modul von einem PC aus steuern zu können.
Der folgende Bericht beschreibt die Erstellung eines Servo-Steuerungs-Moduls auf Basis eines AVR8 Mikrocontrollers und der zugehörigen Treiberbibliothek für Windows und Linux. Außerdem werden drei beispielhafte Anwendungsimplementierungen auf Basis der erstellten Treiberbibliothek vorgestellt. Ein besonderer Schwerpunkt in dieser Arbeit liegt im Umgang mit den Echtzeitanforderungen im Bereich der Firmwareimplementierung.

Inhaltsverzeichnis

Abbildungsverzeichnis I

Listings II

1. Einleitung 1

1.1. Aufgabenstellung 1
1.2. Ergänzende Ziele 2
1.3. Ausgangssituation 2

2. Konzept 4

2.1. Hardware und Firmware 4
2.2. Treiber 5
2.3. Applikationen 5

3. Hardware 6

3.1. Mikrocontoller 6
3.2. Schaltung 7
3.2.1. Pegelwandlung USB 8
3.2.2. Pegelwandlung RS232 8

4. Firmware 11

4.1. Kommunikationsbefehle 11
4.2. USB-Modul 12
4.3. UART-Modul 13
4.4. Servo-Modul 14

4.4.1. PWM-Signal 15

5. Treiber **20**

6. Applikationen **23**

6.1. Konsolen Applikation 23

6.2. Excel-VBA Applikation 23

6.3. Applikation mit grafischer Oberfläche 27

7. Fazit **29**

7.1. Probleme 29

7.2. Ausblick 30

A. Anhang **31**

A.1. Kopie der Internetquellen, erstellter Quellcode, und Bericht und Abbildungen in digitalisierter Form 31

Abbildungsverzeichnis

3.1. Schaltplan - BA Robot 9
3.2. Evaluation-Board - BA Robot 10

4.1. PWM-Signalgenierung mit Interrupts 1 17
4.2. PWM-Signalgenierung mit Interrupts 2 18

6.1. Konsolenapplikation 24
6.2. Grafische Excel-VBAApplikation 25
6.3. Applikation mit grafischer Oberfläche 27

Listings

5.1. Funktionen der Treiberbibliothek zur Steuerung der Servos 22

6.1. Deklarierung der Bibliotheksfunktionen in VBA 26

1. Einleitung

An der Berufsakdademie Berlin wurde vor einiger Zeit ein Roboterarm mit fünf Freiheitsgraden vom Dipl. Ing. Peter Erhardt zu Lehrzwecken konstruiert und mehrere Prototypen wurden produziert. Dieser Roboterarm besitzt fünf Modellbauservos, welche es ermöglichen, dass sich der Roboter bewegt. Es wurde jedoch keine Steuerungselektronik für diesen Roboterarm entwickelt. Im 3. Semester des Bachlorstudiengangs Informatik gab es in der Projektwoche „Embbeded Systems“ die Aufgabe, die grundlegende Ansteuerung mit einem ATmega128 auf einem CharonII von fünf Servomotoren zum Zwecke der Steuerung des Roboterarms zu entwickeln. Aus dieser Aufgabe, welche vom Autor bearbeitet wurde, entstand die Idee ein eigenständiges Hardwaremodul zu entwickeln, welches es ermöglichen sollte, den Roboterarm mit Hilfe eines PC zu steuern. Folgende Aufgabenstellung wurde dann aus dieser Idee entwickelt.

1.1. Aufgabenstellung

Aufgabe ist die Entwicklung und Umsetzung eines Konzepts zur Steuerung eines Roboterarms mit fünf Modellbausservos mit Hilfe eines AVR8-Mikrokontrollers und zugehöriger Schaltung. Ein Konzept für die Low-Level Treiberroutinen ist zu entwerfen und eine Kommunikationsschnittstelle USB/RS232 zum Host-PC zu entwickeln. Die Firmware des Hardwaremoduls soll in Assembler und oder C geschrieben sein und sich einfach auf weitere AVR8-Plattformen portieren lassen. Auf dem PC ist eine Treiberbibliothek zur Steuerung des Hardwaremoduls zu implementieren und dessen Funktionalität durch die Implementierung von Demo-Programmen nachzuweisen.

1.2. Ergänzende Ziele

Die Aufgaben, welchen vom Betreuer dieser Arbeit vorgegeben wurden, sind vom Autor um folgende eigene Ziele und Ideen erweitert worden. Es soll nicht nur eine der gegebenen Kommunikationsschnittstellen implementiert werden, sondern beide gleichzeitig.

Dieses Ziel wird angestrebt,weil es eine flexiblere Nutzung des Gerätes ermöglicht. USB ist die modernere Schnittstelle und im Gegensatz zu RS232 an jedem heutigen PC zu finden. Der Vorteil an der Implementierung von RS232 ist, dass dieser Kommunikationsweg auch sehr gut als Kommunikationsschnittstelle (Pegelwandlung könnte dann auch weggelassen werden) zu einer weiteren MCU verwendet werden kann. Dies wäre zum Beispiel sinnvoll, damit der Roboterarm auch ohne PC zu steuern wäre, wie zum Beispiel mit einer Fernbedienung.

Insbesondere die Firmware soll möglichst stark abstrahiert werden, sodass sich einfach über die Änderung weniger Präprozessorvariablen der Anwendungsbereich anpassen lässt. Damit zum Beispiel auch mehr oder weniger als fünf Servos angesteuert werden können. Auch sollen sich die Servos nicht nur in ihrer Winkelstellung manipulieren lassen, sondern auch in der Geschwindigkeit. Die Geschwindigkeit gibt an, wie schnell der Motor von seiner alten Winkelstellung in seine neue Winkelstellung fährt. Die Treiberbibliothek soll weitgehend unabhängig vom konfigurierten Kompilat der Firmware funktionieren. Zusätzlich soll eine einfache Begrenzung des Bewegungsraums eines Servos ermöglicht werden und diese auch dauerhaft im EEPROMgespeichert werden. Auch ist eine kostengünstige und möglichst einfache Hardwareumsetzung Ziel des Projektes.

1.3. Ausgangssituation

Als Ausgangssituation für das Projekt, welches den Codenamen „BA Robot“ erhalten hat, gab es nur den anzusteuernden Roboterarm und die erworbenen Grundlagen zum Thema Ansteuerung eines Servos mit Hilfe eines PWM-Signals aus dem 3. Semester. Der Roboterarm verfügt über fünf Modellbauservos, deren Steuerleitungen über dreipolige Kabel zugänglich sind. Zwei dieser Pole sind für die Stromversorgung des Motors und die dritte Leitung für die Übertragung des PWM-Signals. Das anliegende PWM-Signal gibt

an, in welcher Winkelstellung der Motor sich befindet. Der Motor erwartet ein PWM-Signal folgender Form: Es muss ca. 20ms ein Low-Pegel anliegen und dann muss ein High-Pegel, welcher zum Beispiel zwischen 0,8ms und 2,2ms liegt (Zeitbereiche können je nach Modell abweichen), folgen. Die Dauer des anliegenden High-Pegels gibts an, in welcher Winkelposition sich der Servo befinden soll.

2. Konzept

Das grundlegende Konzept des Projektes ergibt sich schon indirekt aus der Aufgabenstellung. Das Projekt ist in vier Teile zu gliedern: Hardware, Firmware, Treiber und Applikation. Da sich Hardware und Firmware sehr stark gegenseitig bedingen, werden sie im folgenden Konzept zusammen behandelt, jedoch erfolgt später die detaillierte Erläuterung in eigenen Abschnitten.

2.1. Hardware und Firmware

Durch das selbst definierte Ziel, die Umsetzung auf Hardwareebene möglichst einfach und kostengünstig zu gestalten, ist es notwendig, möglichst die gesamte Funktionalität in einen einzelnen, günstigen Mikrocontroller unterzubringen. Aus diesem Grund wurde für die Implementierung der USB-Schnittstelle nicht wie meist üblich auf eine Hardwarelösung zurückgegriffen, sondern auf eine reine Softwareimplementierung. Als Basis für diese Softwareimplementierung kam V-USB von Objective Development zum Einsatz. Eine nähere Erläuterung zur USB-Schnittstelle erfolgt in Abschnitt 4.2 auf Seite 12.

Die Grundlagen für die RS232-Schnittstelle ist durch einen oder mehrere Hardware-UART bei den meisten AVR8-Controllern gegeben. Die Generierung der PWM-Signale für die Servos kann theoretisch auch auf Hardwareebene oder Softwareebene erfolgen. Vorteil der Hardwarelösung ist, genau wie bei der USB-Schnittstelle, dass sie einfacher zu realisieren ist, aber auch weniger flexibel ist und auch höhere Kosten verursacht, da die Anzahl der vorhandenen Hardware-PWM bei den verschiedenen AVR8 sehr unterschiedlich ist. Außerdem wäre die gewünschte Portierbarkeit auf andere Plattformen auch nicht so gut gegeben. Aus diesem Grund erfolgt die Generierung des PWM-Signals auch softwareseitig.

Aus den genannten Anforderungen ergibt sich, dass eine Schaltung mit einem AVR8-Controller zu entwickeln ist, welche über Anschlüsse für USB und RS232 verfügt und welcher alle gewünschten Anforderungen softwareseitig implementiert, ausgenommen die UART-Kommunikation.

2.2. Treiber

Der entstehende Treiber soll, bedingt durch die Aufgabenstellung, gleichermaßen unter Linux und Windows einsatzfähig sein, daraus ergibt sich, dass die Implementierung so zu erfolgen hat, dass alleine durch erneutes kompilieren auf der Zielplattform der Treiber übernommen werden kann. Aus diesem Grund ist die Treiberbibliothek in C/C++ zu implementieren und mit einem auf beiden Systemen vorhanden Compiler zu kompilieren. Als Compiler kommt der GCC bzw. dessen Windowsportierung MinGW zum Einsatz. Da insbesondere für die USB und RS232 Kommunikation auf spezielle Systemaufrufe zurückgegriffen werden muss, ist es notwendig diese entsprechend der Zielplattformen geeeignet zu abstrahieren. Diese Abstraktion muss über Präprozessordirektiven erfolgen.

2.3. Applikationen

Die entstehenden Demo-Applikationen haben nur eine untergeordnete Priorität, weil sie nur dem Nachweis der Funktion und dem Aufzeigen der Möglichkeiten der erstellen Lösung dienen. Daher ist es sinnvoll, eine einfache Konsolenanwendung zu entwickeln, welche alle von der Treiberbibliothek zur Verfügung gestellten Befehle ausführen kann, da so eine Applikation schnell umzusetzen ist und auch schnell eine Möglichkeit zum Testen der eigenen Treiberbibliotek bietet. Abschließend ist es notwendig, eine testweise Implementierung der Treiberbibliothek im VBA-Umfeld zu überprüfen, da dies der angedachte spätere Anwendungsbereich der Bibliothek sein soll. Grund dafür ist, dass die entstehende Lösung von Maschinenbaustudenten zu Lehrzwecken eingesetzt werden soll und diese seit Jahren im VBA-Umfeld arbeiten.

3. Hardware

In diesem Kapitel wird der verwendete Mikrocontroller und die verwendete Schaltung erläutert sowie auf die Problematik der Pegelwandlung für die verschiedenen Kommunikationsschnittstellen eingegangen.

3.1. Mikrocontoller

Als Mikrocontroller wurde zunächst ein ATmega8 gewählt, dieser 8Bit Mikrocontroller der Firma Atmel ist der günstigste und einfachste Controller der ATmega Serie und verfügt über 1KB integrierten RAM und 8KB Programmspeicher, sowie 512Byte EEPROM. Diese MCU wurde gewählt, da Sie über eine ausreichende Anzahl von Ausgängen, einen Hardware-UART und integrierten RAM verfügt. Der verhältnismäßig große, integrierte RAM ist das wichtigste Merkmal, denn dieser wird zwingend für die Softwareimplementierung der USB-Schnittstelle benötigt, ansonsten hätte auch ein noch einfacheres Modell benutzt werden können, zum Beispiel aus der ATtiny-Serie. Der ATmega8 wird für den Bau des Prototypen in DIL28-Form verwendet, da sich diese bequem von Hand auf einem Lochraster verlöten lässt. Der ATmega8 wurde zunächst mit 16MHZ externem Takt betrieben, es stellte sich jedoch heraus, dass bedingt durch die vielen Operationen, die die MCU, besonders bei der USB Kommunikation absolvieren muss, das Gerät stabiler funktioniert, wenn es mit 18MHZ getaktet wird. Ein weiterer Grund für die Taktung mit 18MHZ ist, dass nur dort die Möglichkeit besteht, eine CRC-Überprüfung durchzuführen (siehe dazu Abschnitt 4 auf Seite 11). Die 18MHZ sind zwar außerhalb der Spezifikation des ATmega8 (16MHZ maximal), jedoch ließen sich bei Tests keine Fehler feststellen. Falls diese Unsicherheit, zum Beispiel bei einer Produktion einer größeren Anzahl des Geräts vermieden werden sollte, könnte anstatt des ATmega8 einen ATmega88 verwendet

werden, welcher pinkompatibel ist und eine Taktung bis 20MHZ erlauben würde.

3.2. Schaltung

Die vorgestellte Schaltung (siehe Abbildung 3.1 auf Seite 9) stellt eine Schaltung eines Evaluation-Boards (siehe Abbildung 3.2 auf Seite 10) dar. Sie könnte für eine spätere Produktion noch um Bestandteile reduziert bzw. verändert oder ergänzt werden.

Der wichtigste Bestandteil ist die MCU selbst. Dieser ist als MEGA8-16 gekennzeichnet, an ihm wird direkt die gesamte Peripherie angeschlossen. Der Port B verfügt beim ATmega8 über 6 Ausgänge, diese werden jeweils für die Ansteuerung eines Servos verwendet, d.h. an jedem dieser Pins muss die gesamte Zeit ein PWM-Signal anliegen. Wie in Abschnitt 4 auf Seite 11 erläutert wird, wäre es mit der Firmware möglich auch 8 Servos anzusteuern. Dies wäre mit dem, in dieser Schaltung verwendeten, ATmega im DIL28 Gehäuse nicht möglich, da bei diesem kein Port mit 8 Ausgängen mehr frei ist. Um trotzdem 8 Servos anzusteuern, müsste die MCU gewechselt werden. Einige der Pins von Port B müssen auch für den ISP verwendet werden, daher sind diese auf einen 10-poligen Wannenstecker herausgezogen. Der 10-polige ISP -Anschluss ist nach Atmel-STK200-Standard belegt, es lassen sich also handelsübliche Programmer verwenden.

Der mit „1“ gekennzeichnete Bereich auf der Schaltung ermöglicht es das Gerät mit einer externen Stromversorgung von 7V-15V zu betreiben. Aus diesen 7V-15V wird dann mit Hilfe des LM7805 eine stabile 5V Gleichspannung generiert. Diese externe Stromversorgung ist bei der Benutzung der USB-Schnittstelle nicht notwendig, hier können die 5V direkt vom Host-PC über USB abgenommen werden. Aus diesem Grund verfügt die Evalution-Schaltung über einen Umschalter in Form eines Jumpers, mit dem gewählt werdem kann, wie das Gerät mit Spannung versorgt werden soll.

Die Bereiche „2“ und „3“ sind für die Kommunikationsschnittstellen USB und RS232 notwendig, sie beinhalten die entsprechenden Anschlussstecker und eine entsprechende Pegelwandlung. Sie werden in den Abschnitten 3.2.1 und 3.2.2 genauer erläutert. Links von der MCU befinden sich nur noch der externe Taktgeber, wofür ein Quarz mit 18MHZ verwendet wird und ein Resettaster sowie die Anschlüsse für die Spannungsversorgung der MCU.

Der ISP -Anschluss, der Bereich der externen Spannungsversorgung, der zugehörige Umschalter und der Resettaster könnten je nach Anwendugsbereich theoretisch auch weggelassen werden.

3.2.1. Pegelwandlung USB

Nach USB-Standard muss die USB-Kommunikation mit folgenden Spannungspegeln erfolgen: Der USB-Host sendet 0V für einen Low-Pegel und 3,3V für einen High-Pegel. Der USB-Host erwartet in Empfangsrichtung ein Signal mit einem High-Pegel von 2V-3,6V und einem Low-Pegel von 0V-0,8V. Auf Seite der MCU bereiten die Pegel, die die MCU erkennen muss, d.h. in seiner Empfangsrichtung keine Probleme, sie liegen im Spezifikationsbereich der TTL Logik. Die Pegel die die MCU senden muss sind hingegen schwieriger zu erreichen.

Eine Möglichkeit wäre eine Niedrigvoltvariante (3V Logik) des ATmega8 namens Atmega8L zu verwenden, jedoch wäre hier der Nachteil, dass die MCU maximal mit 12MHZ externem Takt betrieben werden könnte. Da aber für einen stabilen und fehlerarmen Betrieb des Geräts besser 18MHZ benötigt werden (siehe Abschnitt 3.1 auf Seite 6), kommt diese Möglichkeit nicht in Betracht.

Die zweite Möglichkeit und auch gewählte Möglichkeit ist es, die Pegel, welche an den USB-Signalleitungen „D+“ und „D-“ anliegen, zu wandeln. Um dies zu erreichen werden Zener-Dioden verwendet (in der Schaltung D2 und D3). Es werden 3,6V Low Power Dioden verwendet, welche zusammen mit dem 10KΩ Pullup-Widerstand einen Spannungspegel von ca. 3V ermöglichen.

3.2.2. Pegelwandlung RS232

Auch für die Kommunikation mit dem Host-PC über RS232 ist eine Konvertierung des Spannungspegels notwendig. Bei RS232 entsprechen -12V dem High-Pegel und 12V dem Low-Pegel von TTL. Um diese Konvertierung durchzuführen, wird der weit verbreitete Pegelkonverter MAX232 verwendet. Dieser IC-Baustein übernimmt zusammen mit einigen Kondensatoren (siehe Abbildung 3.1 auf der nächsten Seite Bereich „2“) die entsprechende Konvertierung des Signals.

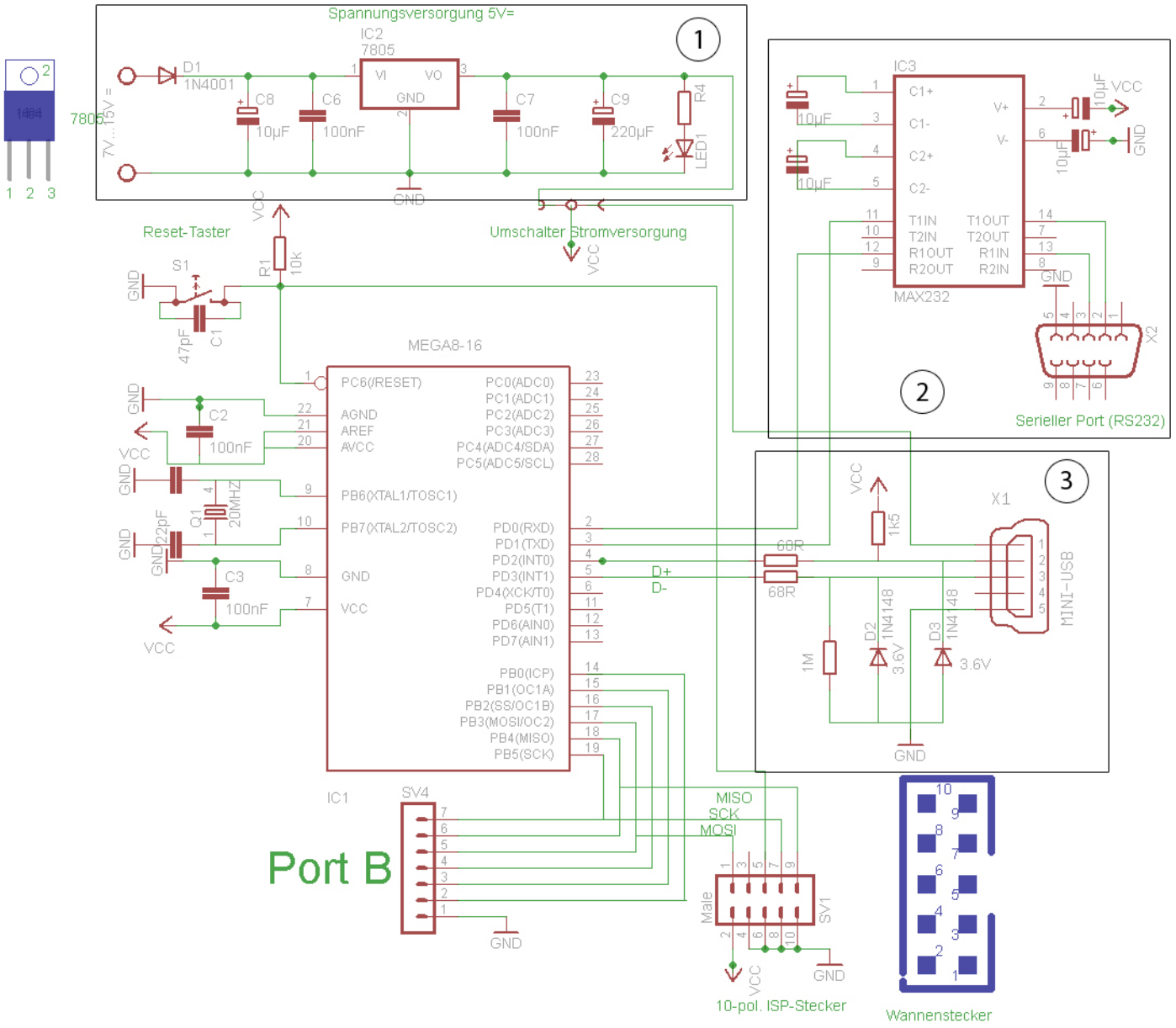

Abbildung 3.1.: Schaltplan - BA Robot

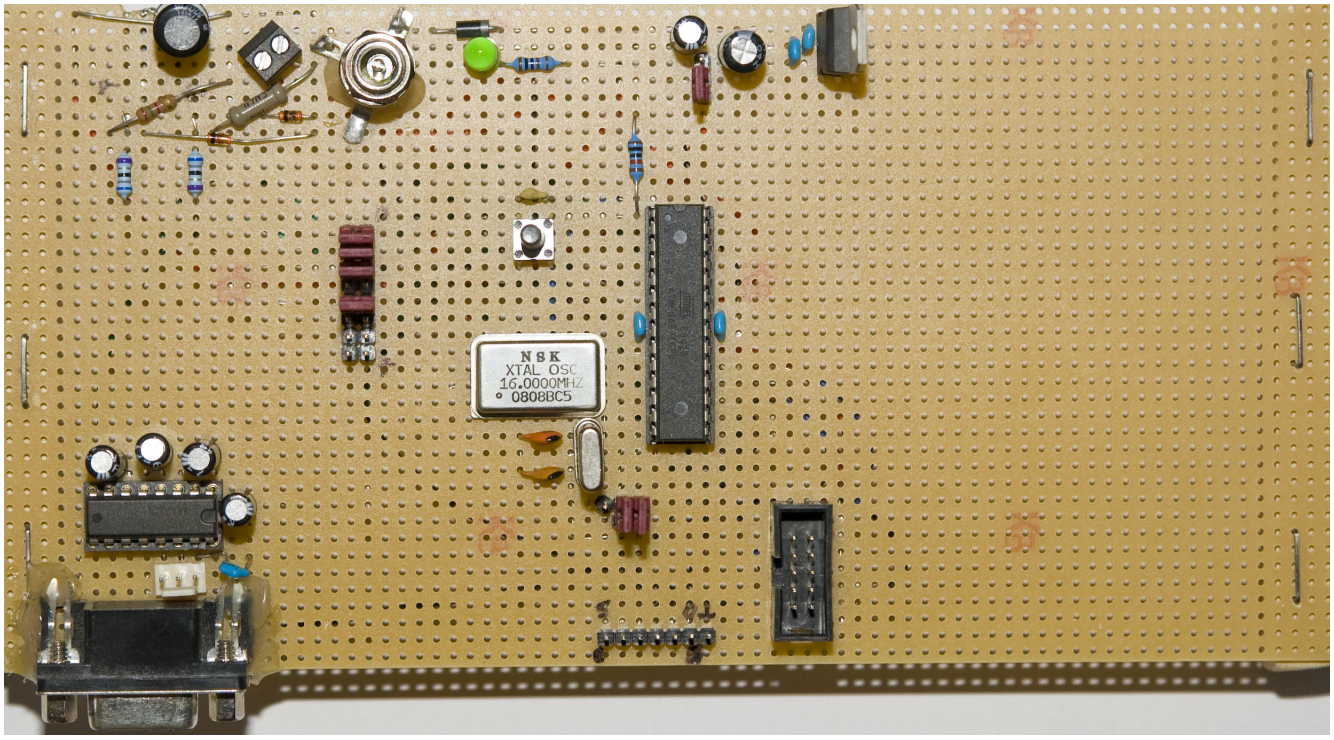

Abbildung 3.2.: Evaluation-Board - BA Robot

4. Firmware

Die Firmware des „BA Robot“ besteht im wesentlichen aus drei getrennten Modulen: Dem USB-Modul, dem UART-Modul und dem Servo-Modul. Diese Module sind weitgehend voneinander getrennt, um den Quelltext übersichtlicher zu gestalten und die Wartung zu vereinfachen. Die einzige Gemeinsamkeit ist der Zugriff auf die globalen Variablen der Servoattribute, da diese von allen drei Modulen geändert und gelesen werden müssen. Die Firmware lässt sich in ihrer Konfiguration sehr einfach über Präprozessordirektiven anpassen. Zum Beispiel kann eine andere Taktrate verwenden werden, die Anzahl der verwendeten Servos von 1-8 definieren, die Auflösung des erzeugten PWM-Signals anpassen und die Intervallgrenzen der ausgegebenen Signale. Außerdem kann die Baudrate für die UART Übertragung definiert, angeben welcher Ausgabeport für das PWM-Signal benutzt wird und welche Pins dort zu verwenden sind. Des Weiteren ist es möglich, die Firmware nur mit USB oder RS232/UART zu kompilieren. Die möglichen wählbaren Taktraten für die MCU sind bei Benutzung des USB-Moduls durch den verwendeten USB-Treiber limitiert. Dieser unterstützt in der aktuellen Revision die Taktraten: 12,12.8,15,16,16.5,18 und 20MHZ. In der aktuellen Version von V-USB unterstützt aber nur die Variante mit 18MHZ eine CRC-Prüfung, daher ist es sinnvoll diese Taktung zu verwenden, da so die Integrität der übertragenen Daten garantiert werden kann (siehe aber auch Abschnitt 3.1 auf Seite 6).

In den folgenden Abschnitten werden die einzelnen Hauptmodule noch weiter erläutert.

4.1. Kommunikationsbefehle

Die möglichen Kommunikationsbefehle sind für beide Kommunikationsschnittstellen im Prinzip gleich. Es wurden folgende Befehle festgelegt: Setzen, sowie Lesen von Geschwin-

digkeit und Winkel eines einzelnen Servos oder aller Servos gleichzeitig, Schreiben oder Lesen der konfigurierten minimalen und maximalen Winkelwerte, sowie eine Abfrage, ob sich alle Servos gerade in einer festen Position befinden oder ob sie gerade von einer Winkelstellung zu einer Weiteren bewegen.

Für die UART-Kommunikation existieren zwei weitere Befehle, welche aber nur zum Debuggen benutzt wurden. Ein Befehl, welcher nur eine Testantwort auslöst und ein Befehl, welcher den FIFO leert. Diese Befehle sind aber nicht im PC-Treiber implementiert, sondern können nur mit einem Terminalprogramm verwendet werden.

4.2. USB-Modul

Das USB-Modul ist für die USB-Kommunikation zuständig. Wie bereits erwähnt, setzt dieses Modul auf die freie (GPL2/3) Softwareimplementierung eines USB-Gerätes. Dieser USB-Treiber der Firma Objective Development nennt sich V-USB. Die Implementierung des Treibers setzt auf den USB Low-Speed Standard, da höhere Standards aufgrund der benötigten Geschwindigkeit nicht softwareseitig in den üblichen AVR8 Taktfrequenzen realisierbar wären. Die Verwendung des langsamsten und ältesten USB-Standards hat für die Anwendung in diesem Projekt keine Relevanz, da alle späteren Standards abwärtskompatibel sind und die Übertragungsgeschwindigkeit (Low-Speed (1,5 Mbit/s)) für diese Anwendung vollkommen ausreichend ist. V-USB ist sehr stark modularisiert und lässt sich über Präprozzesordirektiven und Makros vielseitig anpassen. Daraus ergibt sich, dass nur genau die Funktionalitäten verwendet werden, welche auch benötigt werden. Die Benutzung des V-USB ist sehr einfach, es muss nur die Headerdatei usbconfig.h nach den eigenen wünschen angepasst werden. Dort können zum Beispiel der Name und die Seriennummer des Gerätes festgelegt werden, die verwendeten Pins für die beiden USB-Datenleitungen D+ und D-. Zusätzlich müssen je nach Konfiguration definierte Hook-Up-Methoden selber implementiert werden. Im Falle des BA Robot sind dies genau zwei Funktionen: „usbFunctionSetup“ ist die Funktion die bei jedem USB-Interrupt-Aufruf vom Host-PC ausgeführt wird und die die im USB-Setup-Paket enthaltenen Informationen auswertet und dann eine entsprechende Handlung dafür ausführt. Die Funktion „usbFunctionWrite“ ist die Funktion die bei einem eingehenden Datentransfer nach der

Funktion „usbFunctionSetup“ ausgeführt wird, um die empfangenen Daten schrittweise zu verarbeiten. In der Funktion „usbFunctionSetup“ werden alle Leseanfragen direkt beantwortet. Dies ist notwendig, da der USB-Host eine direkte, synchrone Antwort erwartet. Ankommende Daten, also sozusagen Schreibanfragen werden nur in einem Puffer zwischengespeichert (auch mit Hilfe der Funktion „usbFunctionWrite“), welcher dann in einer weiteren Methode in diesem Modul „handleUSBCommands“ ausgewertet werden. Diese Methode muss immer wieder in der Hauptschleife der Firmware aufgerufen werden, um die eingegangen Daten zu verarbeiten. Dieses Verfahren wird gewählt, um die Abhandlung der USB-Interrupts möglichst schnell zu gestalten, da die Verarbeitung der eingehenden Daten relativ langwierig ist.

Die übertragenen Befehle sind jeweils als ein Byte im Setup-Paket enthalten und die zu lesenden und schreibenden Daten werden jeweils darauf folgend binär in definierter Reihenfolge übertragen.

Da die Kommunikation über USB im Vergleich zu RS232 sehr schnell abläuft, darf der USB-Interrupt bei 18MHZ Takt nur ca. 38 Takte unterbrochen werden, um eine fehlerfreie Kommunikation zu gewährleisten. Dies verursacht den grundlegenden Zeitkonflikt innerhalb der Firmware, welcher in Abschnitt 4.4.1 auf Seite 15 näher betrachtet wird.

4.3. UART-Modul

Die UART-Kommunikation geschieht, wie die USB-Kommunikation auch interruptgesteuert. Im Unterschied zur USB-Kommunikation müssen eingehende Leseanfrage nicht sofort beantwortet werden, die Übertragung erfolgt also asynchron. Dies hat den großen Vorteil, dass die gesamte Übertragung gepuffert und später in Ruhe verarbeitet werden kann. Diese asynchrone Übertragung hat aber auch den Nachteil, das der Host-PC länger auf seine Daten warten muss, jedoch ist diese Verzögerung bei dem geringen übertragenen Datenvolumen nicht relevant. Aus diesem Grund werden alle Anfragen in einen FIFO geschrieben und analog zum USB-Modul später in einer Handler-Methode, welche in der Hauptschleife aufgerufen werden sollte, ausgewertet.

Die Übertragung der Befehle und Daten erfolgt in Klartext ASCII-Zeichen. Dieses Verfahren erleichtert zum Einen das Debuggen, da die Befehle einfach in einem beliebigen

Terminalprogramm gelesen und geschrieben werden können und zum Anderen vereinfacht es die Übertragungslogik, da keine komplizierten Handshakes oder Setup-Pakete benötigt werden.

Ein Befehl besteht aus folgenden Komponenten: Das erste Zeichen beinhaltet den auszuführenden Befehl an das Gerät, darauf folgt, falls benötigt, eine Reihe von Zahlen, wobei jede Zahl maximal drei Stellen haben darf und durch ein Semikolon abgeschlossen wird. Am Ende des Befehls folgt immer ein Unix-typisches Newline-Zeichen. Daraus ergibt sich, dass jeder Befehl mindestens aus zwei ASCII-Zeichen bestehen muss, dem Befehl und dem Newline-Zeichen. Die Firmware geht davon aus, dass die übertragende dreistellige Zahl kleinergleich 254 ist, prüft dies jedoch nicht. Diese Überprüfung wird im Treiber vorgenommen.

Die Antwort vom Gerät erfolgt entweder mit der Bestätigung mit dem Befehl „k\n“, bei einem Schreibbefehl oder durch Rückgabe von angeforderten Zahlenwerten, welche in gleicherweise getrennt werden, wie oben beschrieben.

4.4. Servo-Modul

Die wichtigste Aufgabe des Servo-Moduls ist die Generierung des PWM-Signals, dieser Bestandteil und dessen Problematik wird im folgenden Unterabschnitt (Abschnitt 4.4.1 auf der nächsten Seite) behandelt. Des weiteren ist dieses Modul dafür zuständig, dass sich die einzelnen Servos mit einer definierten Geschwindigkeit von einer Winkelstellung in die Nächste bewegen und die konfigurierten Minimal- und Maximalwerte für die Servos im EEPROM gespeichert werden.

Jeder Servo verfügt über folgende Eigenschaften, welche jeweils durch ein Byte definiert werden (wobei der höchste Werte 255 entspricht und nicht verwendet wird): Er hat eine Winkelstellung, eine Geschwindigkeit und eine minimalen und maximalen Wert für die Winkelstellung. Die Minimal- und Maximalwertkonfiguration wird im EEPROM gespeichert. Das bedeutet, sie sind auch nach einer Unterbrechnung der Stromversorgung wieder verfügbar. Es können insgesamt 1-8 Servos existieren, dies ist abhängig von der Konfiguration, welche zum Zeitpunkt der Kompilierung gewählt wurde. Die gewählten Intervallgrenzen für das PWM-Signal gelten für alle Servos gleichermaßen. Dies lässt sich

nicht umgehen, da alle PWM-Ausgaben in einer Interruptserviceroutine generiert werden.

Die Geschwindigkeit der Servowinkeländerung wird darüber realisiert, dass ein Timer eine definierte Zeitspanne zählt und bei jedem Umlauf für jeden einzelnen Servo einen Geschwindigkeitszähler erhöht. Wenn dieser Geschwindigkeitszähler gleich der definierten Geschwindigkeit ist, dann wird der Zähler wieder 0 gesetzt und für den gewählten Servo geprüft, ob der aktuelle Winkelwert von dem gepufferten neuen Winkelwert abweicht. Falls dies der Fall ist, wird dieser in die jeweilige Richtung iteriert. Sehr wichtig bei dieser interruptgesteuerten Funktionalität ist, dass der globale Interrupt sofort wieder freigegeben wird, damit weder die Kommunikation, noch die Generierung des PWM-Signals gestört werden.

4.4.1. PWM-Signal

Die größte Schwierigkeit bei der Entwicklung der Firmware bestand darin, die Echtzeitanforderungen der Kommunikation mit den Anforderungen der PWM-Signalgenerierung innerhalb einer MCU miteinander zu vereinbaren. Aus diesem Grund wurden auch alle Interruptserviceroutinen (ISR) in Assembler geschrieben, um das Zeitverhalten zu optimieren.

Bedingt durch die Echtzeitanforderungen des V-USB ist es nur möglich, den Eingangs-Interrupt des USB-Signals für maximal ca. 38 Takte zu unterbrechen bei einem MCU-Takt von 18MHZ.

Der grundlegende Algorithmus der softwareseitigen Echtzeitgenerierung des Signals funktioniert folgendermaßen (Es wird davon ausgegangen, dass ein PWM-Signal in der Form wie in Abschnitt 1.3 auf Seite 2 erzeugt werden soll): Ein 16Bit Timer (z.B. Timer1) zählt bis eine Zeitspanne von 20ms erreicht ist, dann erfolgt der erste Interrupt (z.B. Timer1 Compare B), hier werden alle Servopins auf ein (high) gesetzt. Nach weiteren 0,8ms, das ist der Zeitraum, indem die Ausgänge auf jeden Fall high bleiben müssen, wird in ein weitere Interruptserviceroutine (z.B. CompareA) gesprungen, in dieser muss dann für die nächsten 1,4ms Alles abgehandelt werden, um alle $\frac{1{,}4\text{ms}}{255\text{Schritte}}$ eine mögliche Änderung des Ausgangssignals zu bewirken.

$$\text{Taktzyklen} = \frac{\text{Taktfrequenz in HZ}}{1000} \cdot \frac{\text{PWM-Max in ms} - \text{PWM-Min in ms}}{\text{PWM-Auflösung}} \tag{4.1}$$

Beispiel: (4.2)

$$\text{Taktzyklen} = \frac{18000000}{1000} \cdot \frac{2,2 - 0,8}{255} \tag{4.3}$$

$$\text{Taktzyklen} \approx 99 \tag{4.4}$$

Die Problematik hierbei ist, dass wenn die verfügbare Zeitspanne zwischen zwei möglichen Signaländerungen berechnet wird, bei 18MHZ Takt und der Konfiguration wie in Abschnitt 1.3 auf Seite 2 ca. 99 Taktzyklen ermöglicht werden (Gleichung (4.1)). Dies ist keine großer Zeitraum und so wurde in der 1. Implementierung, welche sich an dem Ergebnis der Projektwoche aus dem 3. Semester orientiert, versucht die gesamte Generierung des PWM-Signals in einer Schleife zu erledigen (siehe Abbildung 4.1 auf der nächsten Seite). Das bedeutet, dass die Interruptserviceroutine (ISR), welche z.B. von CompareA ausgelöst wird, eine maximale Laufzeit von 1,4ms hat. Innerhalb dieser Schleife benötigte eine einzelne Iteration ca. 80 von den 99 verfügbaren Takten (bei 5 Servos, es wären max. 6 Servos möglich). Aus diesem Grund, kann die ISR auch nicht verlassen werden. Die Iteration braucht so viele Takte, da die Servowerte in einem Array gespeichert werden und diese in einer Schleife durchlaufen werden müssen. Die Zeigeroperationen auf dem Array müssen in 16Bit erfolgen und sind daher auf einem AVR8 sehr langsam.

Die große Schwierigkeit in dieser Implementierung liegt darin, dass die ISR, welche das PWM-Signal generiert, sehr lange läuft und daher alle weiteren Funktionen stark ausbremst. So müssen die globalen Interrupts wieder eingeschaltet werden, damit sowohl die Interrupts der USB-Kommunikation und der UART-Kommunikation funktionieren, da sonst keine Verbindung vom Host-PC hergestellt werden kann. Dies hat dann zur Folge, dass bedingt durch die teilweise recht zeitaufwendige USB-Kommunikation das PWM-Signal stark verfälscht wird. Bei der Kommunikation via UART tritt das Problem nicht auf, da hier die Interrupts immer sehr schnell sind und die komplette Abarbeitung der empfangenen Befehle separat erfolgen kann.

Als Lösung für dieses Problem wurde zunächst ein neuer Algorithmus implementiert

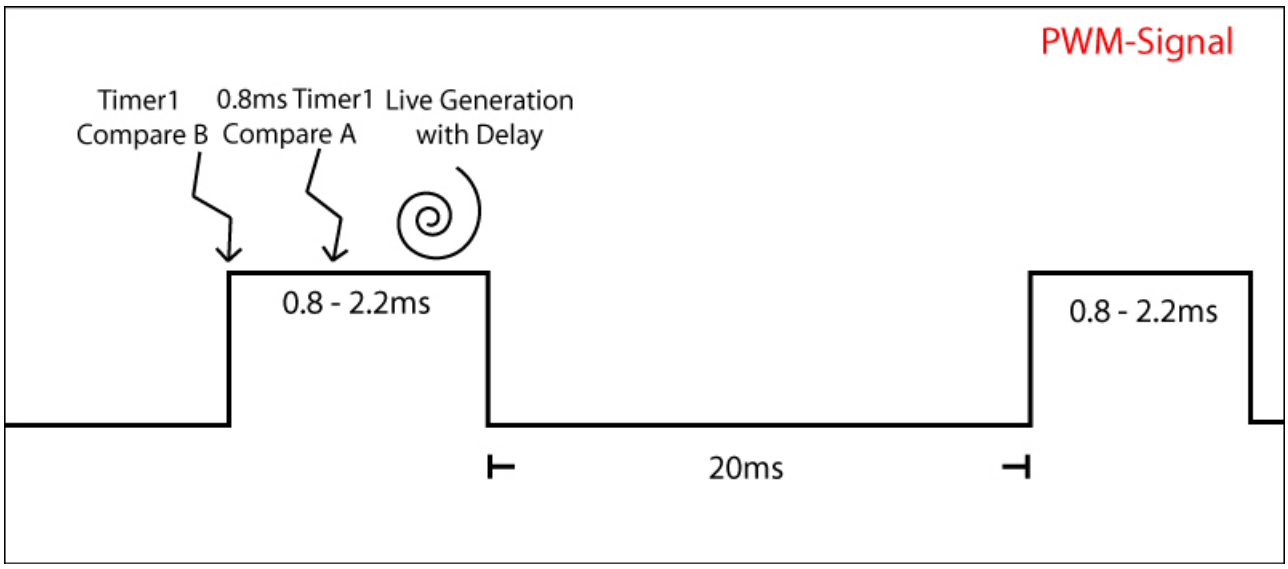

Abbildung 4.1.: PWM-Signalgenierung mit Interrupts 1

(siehe Abbildung 4.2 auf der nächsten Seite), welcher darauf basiert, dass alle PWM-Signale in einem doppelten Puffer in der Main-Loop vorberechnet werden und dann nicht das Signal in einer einzigen Schleife ausgegeben wird, sondern die ISR in der vorher die Generierungsschleife war, wird nur ein weiterer Timer-Interrupt (z.B. Timer2 Compare) für einen weiteren Timer initialisiert, welcher dann zum Beispiel genau alle 99 Takte ausgelöst wird. Diese weitere ISR wird nun 255 mal ausgelöst und iteriert nun über den Doppelpuffer. Dieses Verfahren ist sehr schnell und unabhängig von der Anzahl benutzter Servos und benötigt nur ca. 30 Takte von den 99 verfügbaren, was die Kommunikation nicht beeinträchtigt.

Das Problem bei diesem Algorithmus ist nun, dass die Vorberechnung aller PWM-Signale zum Einen die Ausgabe der letzten Befehls stark verzögert und zum Anderen das implementierte Geschwindigkeitsverhalten der Servos nutzlos macht, da die Aktualisierung des Puffers zu lange dauert.

Die Lösung der Echtzeitproblematik konnte durch eine dritte Implementierung des PWM-Algorithmuses erreicht werden. Dieser ist eine Mischform aus den ersten Beiden. Er funktioniert im Prinzip wie der 2. Algorithmus (siehe auch Abbildung 4.2 auf der

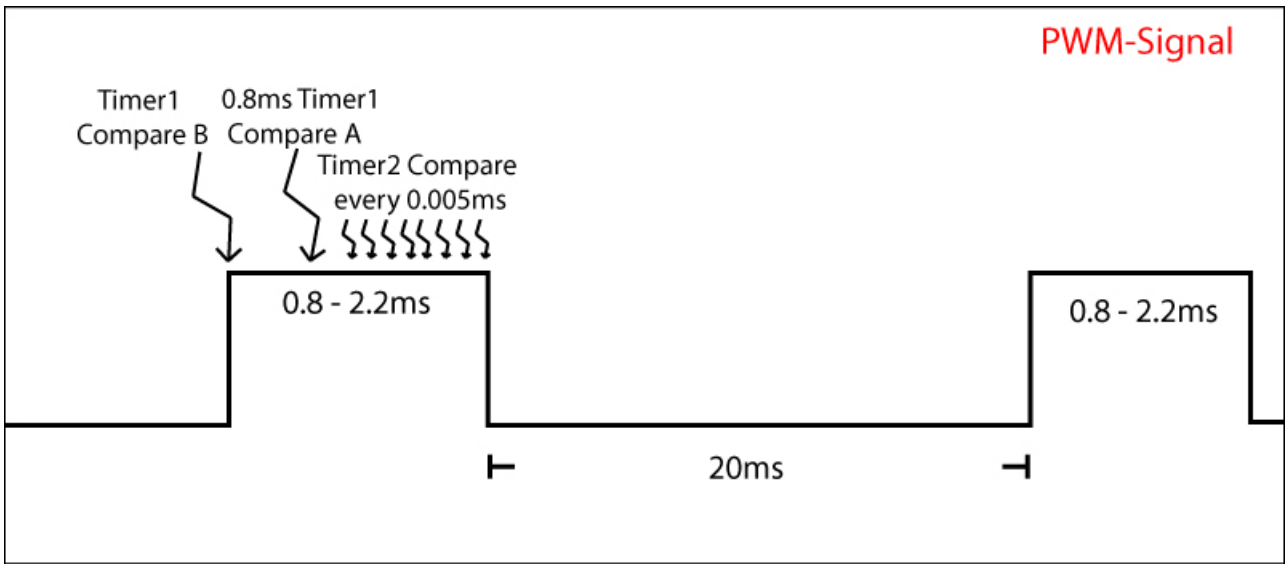

Abbildung 4.2.: PWM-Signalgenierung mit Interrupts 2

nächsten Seite), nur das hier die Änderungen des PWM-Signals nicht aus einem Puffer gelesen, sondern aktiv berechnet werden. Diese aktive Berechnung wird zeitlich dadurch möglich, dass kein Array für die Speicherung der Servozustände verwendet wird, sondern den Servowinkelwerten feste Speicherplätze zugewiesen werden und auch keine Schleife für die Berechnung des Signals benutzt wird, sondern redundanter Quelltext. Um die Implementierung trotzdem dynamisch und konfigurierbar zu halten, zum Beispiel für eine variierende Anzahl von Servos usw., sind Präprozssesormakros verwendet worden. Dieser dritte Algorithmus benötigt bei 8 Servos ca. 65 von 99 Takten und bei 5 Servos ca. 50 Takte.

Es zeigte sich in praktischen Tests, dass bei dieser Implementierung bei fünf Servos, trotz einer kleinen Überschreitung der maximalen Interruptsperrung von 38 Takten, fast immer eine fehlerlose Kommunikation möglich war. Der Unterschied, ob die Übertragung fehlerfrei funktionierte oder nicht, war bedingt durch den verwendeten USB-Host. Dieses rätselhafte Verhalten konnte nach Rücksprache mit einem Objective Development Mitarbeiter darauf zurückgeführt werden, dass bei den heutigen USB 2.0 (Highspeed) Geräten die Abwärtskompatiblität zu USB 1.1 (Lowspeed) jeweils anders implementiert sein kann

und daher auch ein sehr unterschiedliches Zeitverhalten auslösen kann. Diese Problematik konnte nun im Endeffekt dadurch umgangen werden, dass in den 2,2ms in den das PWM-Signal generiert wird, der USB-Interrupt komplett gesperrt und diese verminderte Erreichbarkeit im Treiber auf dem Host-PC abgefangen wird. Dies ermöglicht auch, dass bei der letzten Implementierung des PWM-Algorithmus problemlos acht Servos angesteuert werden können.

Diese endgültige Lösung des Problems könnte im Prinzip auch auf den ersten Algorithmus angewandt werden, jedoch hätte dieses insgesamt ein viel schlechteres Zeitverhalten (für die weiteren Funktionen der Firmware) und würde auch maximal nur 6. Servos bei 18MHZ ermöglichen.

5. Treiber

Die dynamische Treiberbibliothek für den BA Robot ermöglicht es, das erstellten Gerät mit den in Listing 5.1 auf Seite 22 dargestellten Funktionen zu steuern (zur näheren Erläuterung siehe auch Abschnitt 4.1 auf Seite 11). Der Rückgabewert der einzelnen Funktionen gibt an, ob die Funktion erfolgreich ausgeführt wurde (Rückgabe 0) oder nicht (Rückgabe Fehlercode). Die Fehlercodes sind in einer beiliegenden Readme-Datei erklärt. Die Treiberbibliothek ist in C geschrieben und setzt zum Einen auf der freie Bibliothek LibUSB (unter Windows LibUSB32) auf und für die RS232-Kommunikation auf eine für Windows und Linux abstrahierte Funktionssammlung von Teuniz van Beelen(GPL). Da LibUSB unter allen Linux-Distributionen die Standard-Bibliothek zur Kommunikation mit USB-Geräten ist, werden zur Verwendung des BA Robot nichts weiter als die mit dem GCC kompilierte Bibliothek benötigt.

Da unter Windows die Portierung der LibUSB (LibUSB32) verwendet wurde, muss dort zum Einen, zum Kompilieren das Entwicklungspaket installieren sein und zum Anderen muss eine für Windows benötigte Treiberdatei (INF-Datei) generiert worden sein. Dieser spezifische USB-Treiber ist notwendig, da das erstellte USB-Gerät nicht zu einer standard Geräteklasse gehört. Diese INF-Datei muss jedoch nur ein einziges mal generiert werden und kann mit einem Dialog geführten Werkzeug aus dem LibUSB32 Entwicklungspaket erstellt werden. Ein komplettes Treiberpaket für Windows liegt auf der CD bei. Diese INF-Datei sorgt dafür, dass eine benötigte DLL aus dem LibUSB32 Paket in den Windowsordner kopiert wird. Auf diese INF-Datei muss dann beim ersten anstecken des BA Robot an den USB-Port verwiesen werden, wenn nach einem Treiber gefragt wird. Für die Kommunikation über RS232 sind weder unter Windows noch unter Linux weitere Bibliotheken oder Treiber notwendig, da hier die Kommunikation über Standard Systemaufrufe erfolgt.

Da die Treiberbibliothek beide Kommunikationsschnittstellen (USB und RS232) unterstützt und diese von der Endanwendung abstrahiert behandelt, muss von der Endanwendung festgelegt werden, welche der Schnittstellen durch die gegebenen Funktionen benutzt wird. Dies geschieht mit den Funktionen setUSB bzw. setUART (erwartet noch die Nummer des Ports, beginnend bei Null), falls keine dieser Funktionen aufgerufen wird, wird standardmäßig die USB-Verbindung benutzt.

Der Aufbau der Bibliothek ist sehr einfach, es gibt ein Modul, welches die Funktionen enthält, welche von den externen Anwendungen aufgerufen werden und diese verweisen dann je nach gewählter Schnittstelle auf jeweils ein Modul für die USB- oder RS232-Kommunikation, welche dann wiederum auf die oben genannten Funktionssammlungen bzw. Bibliotheken aufsetzen. Es gibt dann noch ein weiteres Modul, welches allgemeine Funktionen enthält.

Listing 5.1: Funktionen der Treiberbibliothek zur Steuerung der Servos

```
void setUSB();

void setUART(unsigned char comport);

unsigned char setServos(unsigned char servoValues[], unsigned
    char servoSpeeds[], unsigned char numberOfServos);

unsigned char getServos(unsigned char servoValues[], unsigned
    char servoSpeeds[], unsigned char numberOfServos);

unsigned char getServo(unsigned char servoNr, unsigned char*
    servo, unsigned char* servoSpeed);

unsigned char setServo(unsigned char servoNr, unsigned char
    servo, unsigned char servoSpeed);

unsigned char isNotInMotion(unsigned char* isNotInMotionBool);

unsigned char getConfigServo(unsigned char servoNr, unsigned
    char* servo, unsigned char* servoSpeed);

unsigned char setConfigServo(unsigned char servoNr, unsigned
    char servoIn, unsigned char servoSpeedIn);
```

6. Applikationen

Es wurden drei Applikationen zum Nachweis der Funktionalität der entwickelten Treiberbibliothek erstellt. Eine Konsolenapplikation und eine Excel-VBA-Applikation, wie in Abschnitt 2.3 auf Seite 5 beschrieben. Zusätzlich wurde aus Eigeninteresse heraus noch eine Applikation erstellt, welche eine grafische Oberfläche bietet und im dotnet Umfeld entwickelt wurde. Alle Applikationen können, bedingt durch die flexible Implementierung der Treiberbibliothek, sowohl via USB als auch via RS232 kommunizieren.

6.1. Konsolen Applikation

Die Konsolenapplikation (siehe Abbildung 6.1 auf der nächsten Seite) ist in der Programmiersprache C geschrieben und ist sehr einfach gehalten. Sie ermöglicht es, nur fünf Servos anzusteuern (entspricht den Anforderungen der Roboterarms). Sie lässt sich mit dem GCC unter Linux kompilieren oder mit dem MinGW unter Windows. Es wurden alle von der Treiberbibliothek bereitgestellten Funktionen implementiert. Die Bedienung ist in einer beiliegenden Readme-Datei erklärt bzw. bei Fehleingaben wird eine Kurzhilfe angezeigt. Mit dieser Anwendung wurden die meisten Tests des Gerätes und der Treiberbibliothek durchgeführt.

6.2. Excel-VBA Applikation

Die Excel-VBA Applikation (siehe Abbildung 6.2 auf Seite 25) bietet nur eine sehr rudimentäre Implementierung der Treiberbibliothek, denn sie dient nur dazu nachzuweisen, wie die erstellte Bibliothek eingebunden werden kann. Im Quelltextauszug Listing 6.1 auf

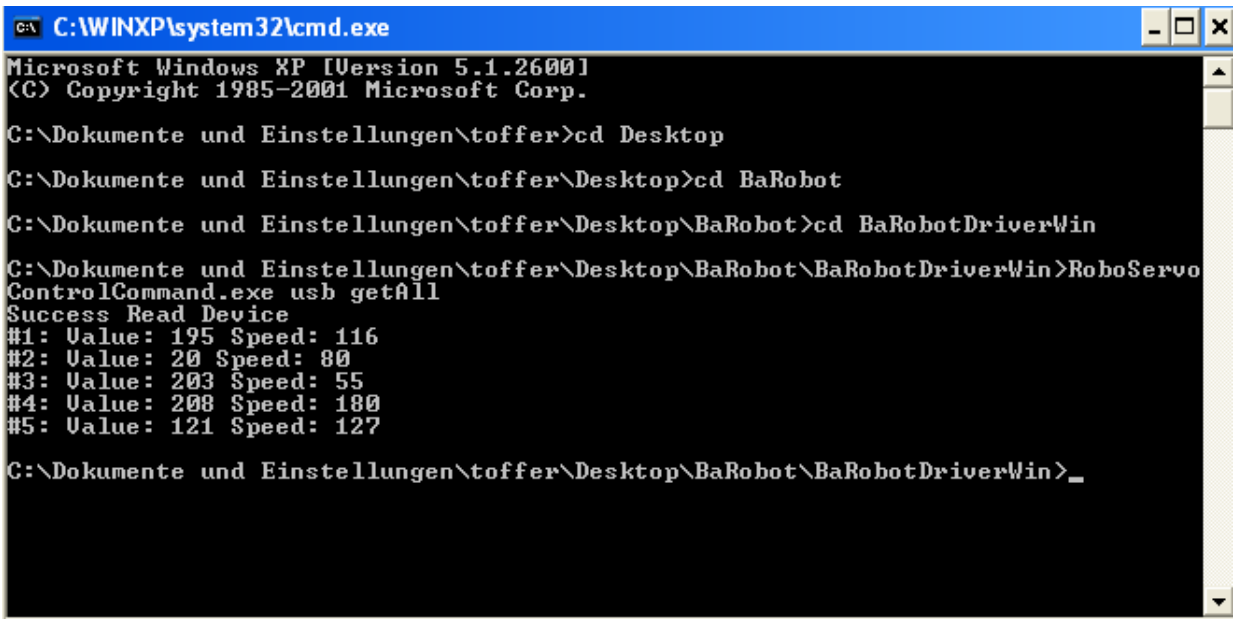

Abbildung 6.1.: Konsolenapplikation

Seite 26 ist zu sehen, wie die DLL in VBA eingebunden werden kann. Die Applikation besteht aus einer einzelnen Microsoft Excel Mappe mit mehreren enthaltenen VBA-Makros und lässt sich demzufolge nur unter Windows betreiben.

Da Excel-VBA eine Windows-API konforme DLL erwartet muss dies bei der Kompilierung mit MinGW beachtet werden. Dazu sind bestimmte Flags zu setzen, dies wird jedoch über Präprozzesormakros vereinfacht. Es ist aber zu beachten, dass MinGW an den Funktionsnamen in der DLL ein „@“ und die Länge der Eingangsparameter in Byte heran hängt. Dieser Umstand wird aber in einer, der Treiberbibliothek beigefügten, Readme-Datei noch näher erläutert.

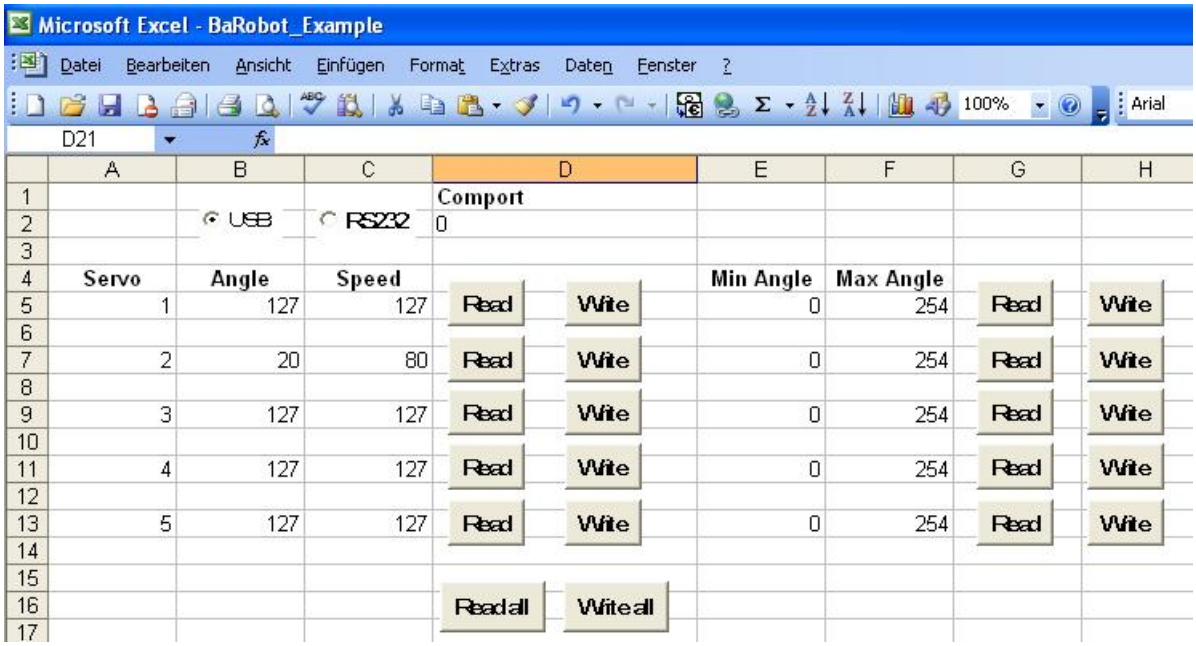

Abbildung 6.2.: Grafische Excel-VBAApplikation

Listing 6.1: Deklarierung der Bibliotheksfunktionen in VBA

```
Declare Sub setUSB Lib "libroboServoControlWin.dll" Alias "
    setUSB@0" ()

Declare Sub setUART Lib "libroboServoControlWin.dll" Alias "
    setUART@4" (ByVal comport As Byte)

Declare Function getServo Lib "libroboServoControlWin.dll" Alias
     "getServo@12" (ByVal servoNr As Byte, ByRef servo As Byte,
    ByRef servoSpeed As Byte) As Byte

Declare Function setServo Lib "libroboServoControlWin.dll" Alias
     "setServo@12" (ByVal servoNr As Byte, ByVal angle As Byte,
    ByVal speed As Byte) As Byte

Declare Function isNotInMotion Lib "libroboServoControlWin.dll"
    Alias "isNotInMotion@4" (ByRef isNotInMotionBool As Byte) As
    Byte

Declare Function getConfigServo Lib "libroboServoControlWin.dll"
     Alias "getConfigServo@12" (ByVal servoNr As Byte, ByRef
    angle As Byte, ByRef speed As Byte) As Byte

Declare Function setConfigServo Lib "libroboServoControlWin.dll"
     Alias "setConfigServo@12" (ByVal servoNr As Byte, ByVal
    angle As Byte, ByVal speed As Byte) As Byte

Declare Function setServos Lib "libroboServoControlWin.dll"
    Alias "setServos@12" (angle() As Byte, speed() As Byte,
    numberOfServos As Byte) As Byte
```

```
Declare Function getServos Lib "libroboServoControlWin.dll"
    Alias "getServos@12" (angle() As Byte, speed() As Byte,
    numberOfServos As Byte) As Byte
```

6.3. Applikation mit grafischer Oberfläche

Die Applikation mit grafischer Oberfläche (siehe Abbildung 6.3) wurde in C-sharp innerhalb des dotnet Umfelds erstellt. Daraus ergibt sich, dass sie ohne weitere Anpassungen nur unter Windows lauffähig ist. Jedoch müsste sie theoretisch auch durch geringe Anpassungen unter der quelloffenen Umgebung Mono kompilierbar und lauffähig sein.

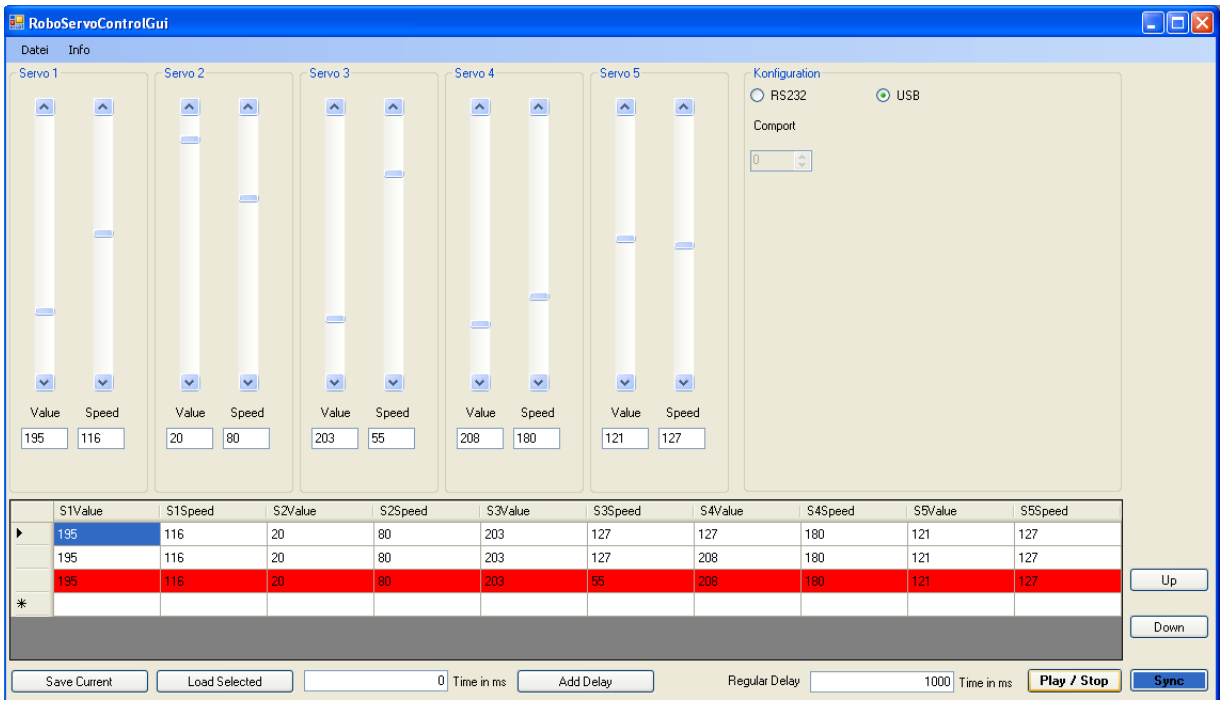

Abbildung 6.3.: Applikation mit grafischer Oberfläche

Die Applikation bietet die Möglichkeit fünf Servos mit Schiebereglern und Eingabefeldern für Geschwindigkeit und Winkelstellung zu steuern. Des Weiteren ist es möglich, die

aktuelle Einstellung in einer fortlaufenden Liste zu speichern. Diese Liste kann mit Pausen ergänzt werden und auch zwischen der Ausführung der verschiedenen Bewegungen können standardisierte Pausen gesetzt werden. Diese Liste kann dann abgespielt werden, womit sich auch gut komplexe Bewegungsabläufe modellieren lassen. Diese Liste kann außerdem extern in einer XML-Datei gespeichert werden, woraus sie auch wieder geladen werden kann.

7. Fazit

Das Projekt BA Robot konnte erfolgreich abgeschlossen werden. Alle gestellten Anforderungen konnten umgesetzt und deren Funktion im Rahmen einer Präsentation nachgewiesen werden. Das wichtigste Ziel, es zu ermöglichen, dass der an der BA-Berlin entwickelte Roboterarm mit fünf Freiheitsgraden in Zukunft effektiv zu Lehrzwecken einsetzten werden kann, wurde komplett erreicht. Außerdem bietet das erstellte Gerät zusammen mit seiner Treiberbibliothek auch für zukünftige Projekte an der BA-Berlin genug Reserven, um zum Beispiel andere Roboter, besonders welche mit noch mehr Freitheitsgeraden, zu steuern. Das zurückliegende Projekt wurde als sehr lehrreich angesehen, verlief jedoch auch nicht ohne gewisse Schwierigkeiten.

7.1. Probleme

Als größte Schwierigkeit bei dem Projekt stellten sich die nicht vorhandenen Vorkenntnisse im Bereich der Elektrotechnik heraus. Dies hat besonders am Anfang, wo die Entwicklung der Hardware im Vordergrund stand, einen sehr großen Einarbeitungsaufwand verursacht. Jedoch ist auch zu sagen, dass gerade in diesem Bereich ein enormer Lernerfolg zu verzeichnen ist.

Wie schon in der Einleitung deutlich gemacht wurde, war besonders die Echtzeitkonkurrenz, welche zwischen der USB-Kommunikation und der PWM-Signalgenerierung bestand mit die größte Schwierigkeit bei diesem Projekt. Besonders in diesem Bereich war ein ausgiebiges Testen notwendig, wobei sich besonders die Arbeit mit dem Oszilloskope bewährt hat.

Lange Zeit war auch das unterschiedliche Verhalten verschiedener USB-Hosts ein großes Rätsel, was jedoch, wie in Abschnitt 4.4.1 auf Seite 15 erwähnt, durch einen hilfreichen

Mitarbeiter der Firma Objective Development aufgeklärt werden konnte.

7.2. Ausblick

Da alle gestellten Anforderungen in diesem Projekt bewältigt werden konnten, sind in diesem Bereich keine weiterführenden Arbeiten notwendig. Es soll aber zeitgleich zur Abgabe dieser Arbeit das Projekt unter der GPL im Internet veröffentlicht werden und es wäre zu klären, ob in Kooperation mit der BA-Berlin einige Exemplare des Gerätes produziert werden sollen. Des weiteren ist es vorstellbar, das Projekt entweder durch neue Funktionen zu erweitern oder neue Projekte zu starten, welche auf BA Robot aufsetzen.

Als Erweiterung wäre denkbar, die erstellte Firmware zum Beispiel auf die Arduino-Plattform zu portieren, was durch den dort integrierten USB-Controller besonders im Bereich der USB-Kommunikation Vorteile bringen würde. Eine weitere interessante Erweiterung wäre auch die Portierung auf eine Mikrocontrollerarchitektur mit 16 oder 32Bit, da dort das PWM-Signal noch feiner aufgelöst werden könnte, was zu einer präziseren Ansteuerung der Motoren führen würde. In der jetzigen 8Bit Implementierung ist nur eine Aufteilung des kompletten Servo-Winkelbereichs in 255 Stufen ermöglicht.

Denkbare Projekte, welche auf den erstellten Grundlagen aufsetzen, wären zum Beispiel im Bereich Embedded Systems eine Zusatzplatine, welche es ermöglicht, eine Fernbedienung mit einem Joystick zur Steuerung des Roboterarms zu benutzen. Im Bereich der normalen PC-Entwicklung wäre es denkbar, die Treiberbibliothek um weitere Funktionen zu ergänzen, bzw. eine neue, aufsetzende Bibliothek zu schreiben, welche komplexere Ansteuerungsfunktionen für den Roboterarm enthält. Beispiele wären hier die Steuerung über die Angabe von Koordinaten aus Sicht von verschiedenen Koordinatensystem.

A. Anhang

A.1. Kopie der Internetquellen, erstellter Quellcode, und Bericht und Abbildungen in digitalisierter Form

Glossar

Kürzel	**Beschreibung**	
ASCII	American Standard Code for Information Interchange	13, 14
CRC	Cyclic redundancy check ist ein Verfahren zur Bestimmung eines Prüfwerts für Daten, um Fehler bei der Übertragung oder Speicherung erkennen zu können.	6, 11
DLL	Dynamic Link Library: Meist Programmbibliothek unter Microsoft Windows	20, 23, 24
EEPROM	Electrically Erasable Programmable Read Only Memory	2, 6, 14
FIFO	First in first out Pufferspeicher	12, 13
GCC	GNU Compiler Collection ist der Name der Compiler-Suite des GNU-Projekts	5, 23
GPL	Die GNU General Public License ist eine von der Free Software Foundation herausgegebene Lizenz mit Copyleft für die Lizenzierung freier Software.	12, 20, 30
ISP	In System Programmer	7

Kürzel	**Beschreibung**	
MCU	Micro Controll Unit	2, 6–8, 11, 15
MinGW	Minimalist GNU for Windows ist eine Softwareportierung der GNU-Entwicklerwerkzeuge (GCC, GDB) auf die Windows-Plattform	5, 23, 24
PWM	Pulsewellenmodulation	2, 4, 7, 11, 14–18, 29, 30
RAM	Random Access Memory	6
RS232	Recommended Standard 232: Standard für serielle Übertragung von Binärdaten	1, 2, 4, 5, 7, 8, 11, 13, 20, 21, 23
UART	Universal Asynchronous Receiver Transmitter	4, 6, 11–13, 16
USB	Universal Serial Bus: Serielles Bussystem zur Verbindung eines Computers mit externen Geräten	1, 2, 4–8, 11–13, 15, 16, 18, 20, 21,
VBA	Visual Basic for Applications ist eine zu den Microsoft-Office-Programmen gehörende Skriptsprache	5, 23, 24, 26

Literaturverzeichnis

[1] DLL Tutorial For Beginners, http://www.codeguru.com/cpp/cpp/cpp_mfc/tutorials/article.php/c9855, stand 10.11.2009

[2] Gietenbruch,Torsten: Das Experimentierboard, http://www.rclineforum.de/forum/thread.php?threadid=102929&sid=2056393f61e2b3ad42d79363ce6e7896, stand 15.09.2009

[3] Lewing: Der Pegelumsetzer MAX232, http://www.elektronik-magazin.de/page/der-pegelumsetzer-max232-15, stand 27.12.2009

[4] Nürnberg,Darwin: RS232, http://www.rn-wissen.de/index.php/RS232, stand 27.10.2009

[5] Objective Development: VUSB, http://www.obdev.at/products/vusb/index-de.html, stand 15.10.2009

[6] Peacock,Craig: USB in a NutShell, http://www.beyondlogic.org/usbnutshell/usb6.htm#SetupPacket, stand 27.12.2009

[7] Van Beelen, Teuniz: RS-232 for Linux and WIN32, http://www.teuniz.net/RS-232/, stand 24.11.2009

[8] Wachtler,Klaus: Dynamische Bibliotheken unter Windows und Linux, http://www.wachtler.de/dynamischeBibliotheken/dynamischeBibliotheken.html, stand 27.10.2009