HOCHSCHULE BREMERHAVEN

Steuerung eines Roboterarmes mit einem Mikrocontroller

Spezielle Kapitel zu Rechnersystemen

Hausarbeit
im Studiengang Informatik
an der Hochschule Bremerhaven

eingereicht bei
Prof. Dr. Adolf-Horst Haltof

vorgelegt von

Hauke Hoffmann

Torben Fiedler

Sebastian Albrecht

Bremerhaven, 28. Februar 2005

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Im Rahmen des Studienfaches „Spezielle Kapitel zu Rechnersystemen“ soll die Realisierung eines I/O-Beispiels unter Verwendung des PicMicroBoards von Microchip erfolgen.
Die Entscheidung fiel auf eine Steuereinheit für einen Roboterarm, der damit von einem PC bspw. über die Tastatur gesteuert werden kann.

In unserer Gruppe gab es keine feste Aufgabenteilung. Anfangs hat jeder mal gelötet und auch die Planung lief eher chaotisch ab. Im Laufe der Zeit hat es sich dann einfach ergeben, dass jeder eine Aufgabe hatte, um die er sich hauptsächlich kümmerte, Aufgaben, bei denen er die anderen unterstützte und es gab auch weiterhin Aufgaben, die gemeinsam erledigt wurden.

Sebastian Albrecht war hauptsächlich mit dem Optokoppler-Board und dem Roboterarm beschäftigt.
Hauke Hoffmann hat sich intensiv mit der Programmierung befasst.
Torben Fiedler hat sich um das PIC-Board und den Brenner und alles, was mit Löten zu tun hatte gekümmert.

2. Übersicht

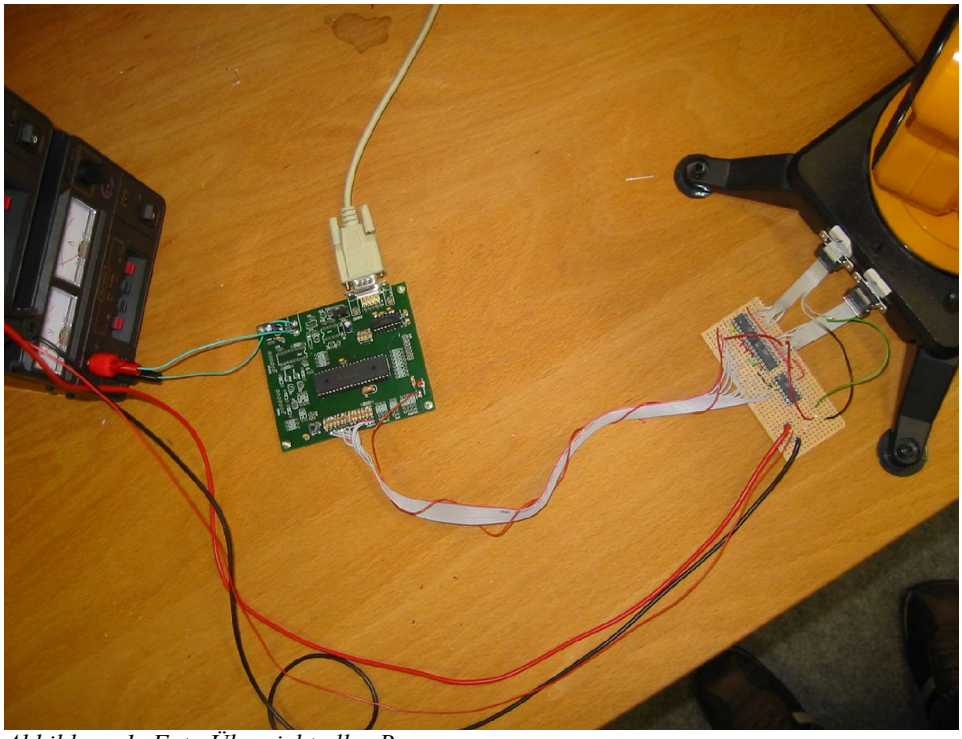

Abbildung 1: Foto Übersicht aller Baugruppen

Abbildung 1 zeigt (fast) allen Komponenten, die zur Realisierung dieses Projektes benötigt wurden. Links im Bild sind die Netzteile für die Stromversorgung des PIC-Boards und des Roboterarmes zu erkennen. Die grüne Platine ist das PIC-Board. Der Stecker am oberen Rand verbindet das PIC-Board mit dem Notebook (nicht im Bild). Das breite Flachkabel sind die Steuerleitungen, die vom PIC-Board zum Opto-koppler-Board verlaufen. Dieses ist die kleine Platine rechts im Bild. Sie ist über zwei serielle Kabel mit DB-9 Steckern mit dem Roboterarm verbunden.

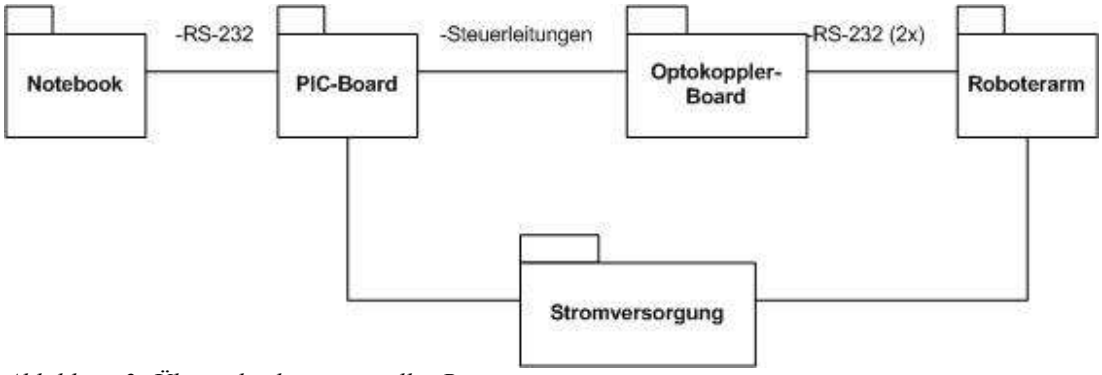

Abbildung 2: Übersichtsdiagramm aller Baugruppen

Abbildung 2 zeigt eine abstrakte Übersicht aller Geräte und der Verbindungen zwischen ihnen. Auf dem Notebook läuft ein Java-Programm, dass bestimmte Tastendrücke der Tastatur erkennt und je nach erkannter Tastenkombination entsprechende Bytes (Kommandos) an die serielle Schnittstelle (RS-232) sendet. Der PIC-Prozessor auf dem PIC-Board empfängt diese Bytes und ändert entsprechend seinem internen Programm die Spannungspegel an den Ports A und B. Die Ausgänge dieser Ports sind wiederum über ein Flachkabel mit der Optokoppler-Platine verbunden. Die Optokoppler schalten bzw. schalten nicht, je nach Kommando. Damit werden die Motoren des Roboterarmes ein- bzw. ausgeschaltet.

3. Hardware-Komponenten

3.1. Roboterarm

Der für das Projekt verwendete Roboterarm ist ein Quickshot SVI-2000 Robotarm (Abbildung 3). Er besitzt fünf bewegliche Achsen, die ursprünglich mit zwei digitalen Atari-Joysticks gesteuert werden können. Außerdem ist es über ein entsprechendes Interface-Modul möglich, den Roboterarm mit einem Atari-Computer zu steuern. Der Roboterarm selbst liefert in keiner Weise eigene Daten über Position oder Lage der Achsen zurück.

Abbildung 3: Foto Quickshot SVI-2000 Robotarm

Die beweglichen Achsen des Roboterarmes sind folgende:
- Achse 1 auf dem Sockel: Drehung im und gegen den Uhrzeigersinn
- Achse 2 (erstes Gelenk): Vor und zurück
- Achse 3 (zweites Gelenk): Hoch und runter
- Achse 4 (Greifer): Drehung des Greifers im und gegen den Uhrzeigersinn
- Achse 5 (Greifer): Öffnen und Schließen des Greifers

Bis auf die Drehung des Greifers (Achse 4) sind sämtliche Achsen in ihrer Bewegung begrenzt, d. h. sie können nur bis zu einem bestimmten Anschlag bewegt werden.

Für jede Achse ist ein Gleichstrommotor im Roboterarm zuständig, an welchen entweder -3 V oder + 3 V angelegt werden. Die unterschiedlichen Spannungen werden durch eine Verschaltung von vier im Roboterarm liegenden Mono-Batterien (je 1,5 V) erzeugt. Jeder Motor dreht entsprechend der Polarität vor oder zurück und bewegt somit seine Achse.
Da die benötigten digitalen Joysticks prinzipiell nur aus einfachen Tastern bestehen, ist es sehr einfach möglich, mit einem kleinen Kabel den Roboterarm an den Anschlussbuchsen durch „Überbrücken“ zu steuern, bzw. das „Überbrücken“ durch den Einsatz von Relais oder Optokopplern zu automatisieren. Da insgesamt fünf Achsen und pro Achse zwei Pins zur Steuerung (hin oder her) vorhanden sind, ist eine Lösung erforderlich, die mindestens zehn verschiedene Zustände zulässt.

Eine beispielhafte Zeichnung der internen Verschaltung des Roboterarmes zeigt Abbildung 4. Es ist nur ein Motor dargestellt und die Überbrückung als Taster. Darüber hinaus enthält der Roboterarm eine größere Schaltplatine mit Treibertransistoren zum Ansteuern der Motoren, welche hier jedoch keine Aufmerksamkeit erhalten soll.

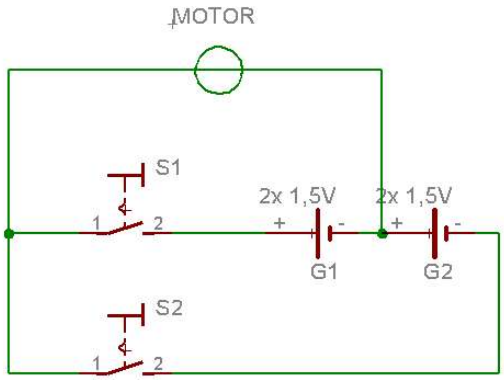

Abbildung 4: Beispielhafte interne Verschaltung eines Motors im Roboterarm

Ein wesentlicher Nachteil des Roboterarmes wurde uns während des Projektes bewusst. Die Abhängigkeit von Batterien ist nicht nur kostenintenstiv und umweltbelastend, sondern führt auch dazu, dass der Roboterarm Bewegungen mit der Zeit immer langsamer durchführt.

Aus diesen Gründen war es notwendig, eine externe Spannungsversorgung zu schaffen. Problematisch war die Anforderung negative und positive 3 V zugleich zu erhalten. Der Roboterarm bewegt sich beim Verbrauch zusätzlich oftmals über der 1 Ampere-Marke, sodass kleine Tischnetzteile kaum zum Erfolg führen. Auch der Versuch, mit einem sehr kleinen Widerstand einen Spannungsteiler herzustellen und an einem einzigen Netzteil zu betreiben, schlug aufgrund der zu geringen Leistungsfähigkeit der Widerstände fehl.

Erst der Betrieb mit zwei Netzteilen funktionierte problemlos. Diese wurden so verschaltet, dass der Pluspol des einen Netzteiles mit dem Minuspol des anderen direkt verbunden wurde. Beide Netzteil müssen jedoch unbedingt auf die gleiche Spannung (3 V) eingestellt sein.

3.2. Optokoppler-Board

Das Optokoppler-Board fungiert als Schnittstelle zwischen dem Roboterarm und dem PicMicroBoard. Durch die an den Ports des PicMicroBoards angelegten TTL-Pegel schalten die Optokoppler durch oder eben nicht und ermöglichen somit das „Überbrücken“ der Kontakte am Roboterarm und letztendlich die Bewegung der Achsen. In der folgenden Abbildung ist das Optokoppler-Board dargestellt.

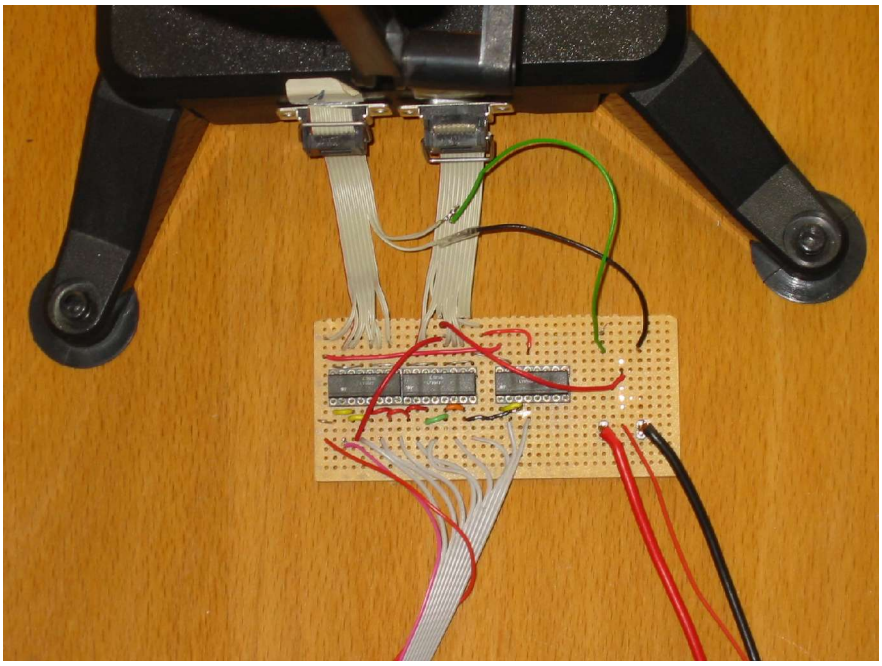

Abbildung 5: Foto Optokoppler-Board

Das 9-polige Flachbandkabel am unteren Bildrand wurde um eine zusätzliche Ader erweitert (roter Draht) und ist mit dem PicMicroBoard verbunden. Jede Ader führt zu einem Eingang an einen der Optokoppler. Es wurden vier Optokoppler-ICs vom Typ PC847 verwendet, welche jeweils vier Optokoppler-Einheiten beinhalten. Von den insgesamt 12 Optokopplern finden jedoch nur 10 Verwendung. Die Ausgänge der Optokoppler sind über zwei 9-polige DSUB-Kabel mit den Buchsen des Roboterarmes verbunden. Am rechten Platinenrand ist außerdem die nachträglich entwickelte externe Stromversorgung des Roboterarmes untergebracht. Die drei Kabel nach unten (rot, rot, schwarz) führen zu den Netzteilen und die drei dünneren (grün, rot, schwarz) sind direkt mit Pins des Roboterarmes verschaltet.

Im Folgenden ist der Schaltplan des Optokoppler-Boards abgebildet. Links ist der Anschluss am PicMicroBoard sichtbar und rechts die Anschlüsse am Roboterarm. Die Spannungsversorgung des Roboterarmes ist rechts mittig dargestellt.

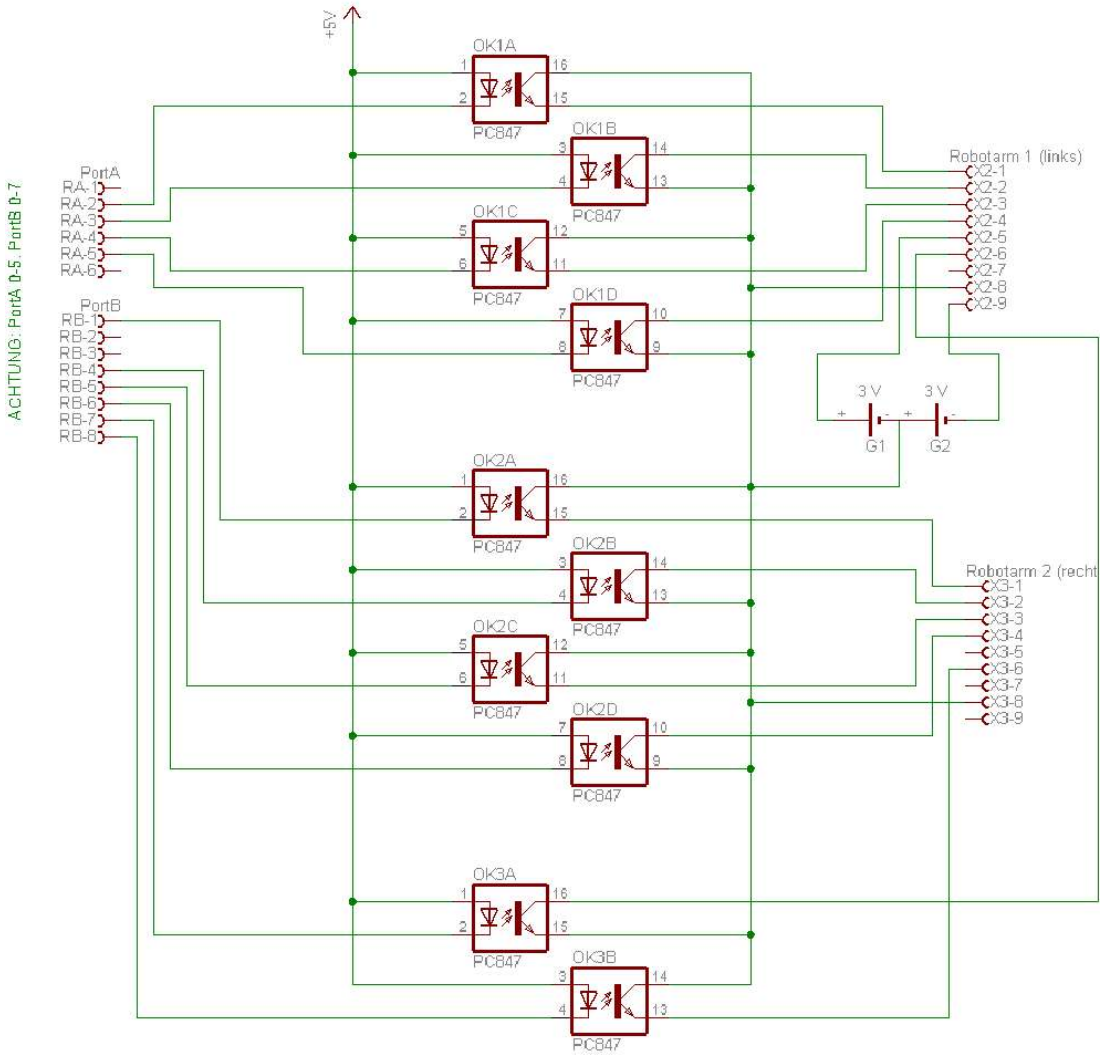

Abbildung 6: Schaltplan des Optokoppler-Boards

Die Verwendung der Optokoppler hat den großen Vorteil, dass die Stromkreise vom PIC und die vom Roboterarm galvanisch getrennt sind und sich somit niemals gegenseitig beeinflussen können. Gegen eine denkbare Schaltung mit Relais hätte gesprochen, dass der PIC vermutlich nicht genügend Strom zum Schalten der Relais bereitstellen kann, Relais noch länger zum Schalten brauchen und zudem ständig hörbare Geräusche produzieren. Eine einfache Transistorschaltung hätte dem Prinzip der galvanischen Trennung widersprochen, wäre aber dennoch möglich gewesen.

3.3. PicMicroBoard

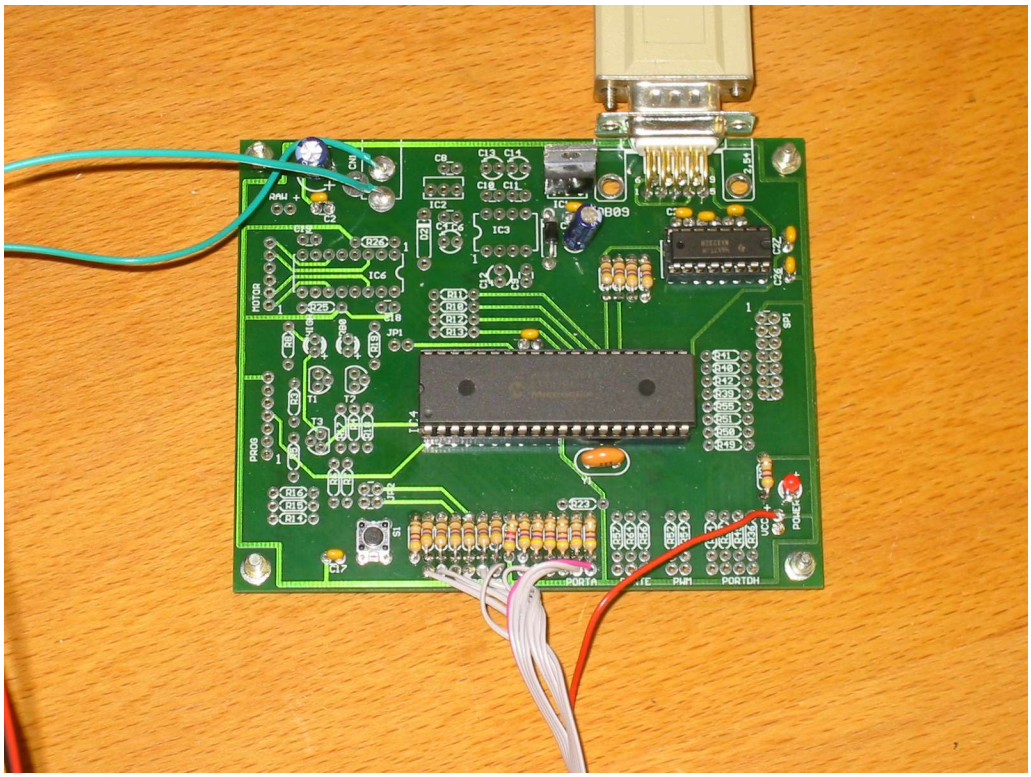

Abbildung 7: Foto PIC-Board groß

Abbildung 3 zeigt das fertiggestellte Board mit allen Anschlüssen und eingesetzten ICs. In der Mitte ist deutlich der PIC-Prozessor zu erkennen. Die beiden hellblauen Drähte links oben dienen der Stromversorgung. Rechts oben befindet sich die DB-9 Buchse der RS-232 Schnittstelle. Direkt darunter ist der Wandler IC zu erkennen. Am unteren Rand der Platine sind die 10 Datenleitungen zum Optokoppler-Board zu erkennen. Die Datenleitungen sind mit den Ports A und B des PIC verbunden.

Ein großer Vorteil war der großzügige Platz auf dem Board, der ein relativ unkompliziertes Löten ermöglichte. Auch die Beschaffenheit der Lötaugen war gerade für Anfänger gut geeignet (im Gegensatz zum Kupfer des Brenners).

Zwei Nachteile sind uns dennoch aufgefallen:

- Der PIC muß zur Programmierung jedes Mal aus dem Sockel genommen werden. Dies gestaltet sich recht schwierig und gelingt nur unter Zuhilfenahme eines Schraubendrehers. Dabei ist die Gefahr den PIC ernsthaft zu beschädigen relativ groß, ganz zu schweigen davon, dass eigentlich jedesmal ein paar Beinchen verbogen werden. Hier wäre der Einsatz eines Nullkraft-Sockels sicherlich sinnvoll.
- Die Lötaugen für die Ports A und B liegen alle schön nebeneinander. Leider sind sie unzureichend beschriftet und belegen die Ports nicht in der durch die Anordnung suggestierten linearen Abfolge. Es dauerte ziemlich lange bis wir darin unseren Fehler entdeckten.

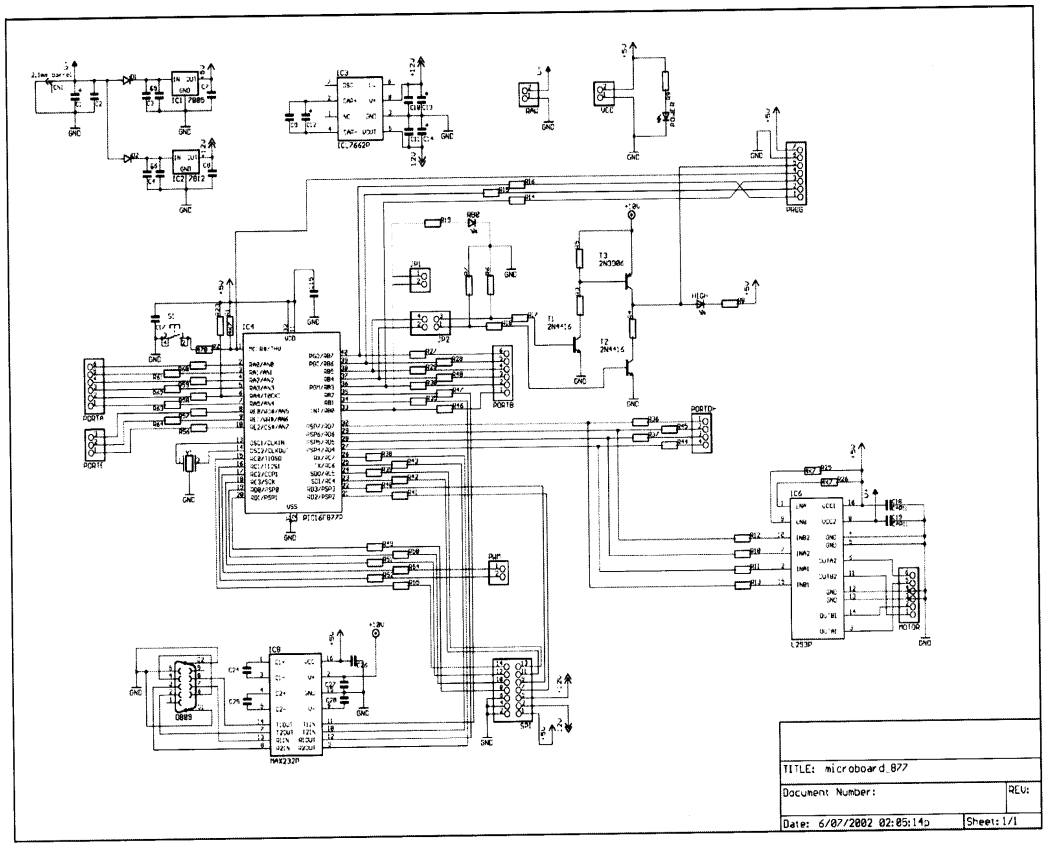

Abbildung 8: Schaltplan PIC-Board

Der in Abbildung 4 gezeigte Schalt-/Bestückungsplan für das PIC MicroBoard zeigt alle Bauteile, die überhaupt verwendet werden können. Im Rahmen dieser Aufgabe wurden nicht alle Teile verwendet. Neben dem PIC-Sockel als unverzichtbarem zentralen Bauteil sind vor allem die Stromversorgung (vergl. Abb. 5) und die RS-232 Schnittstelle (vergl. Abb. 6) realisiert worden.

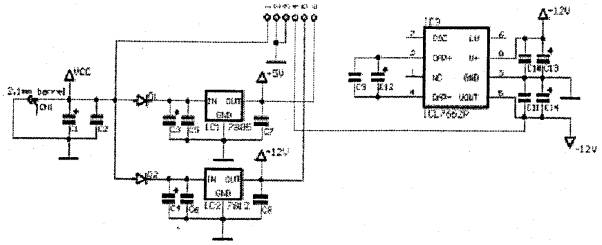

Abbildung 9: Schaltplandetail Stromversorgung PIC-Board

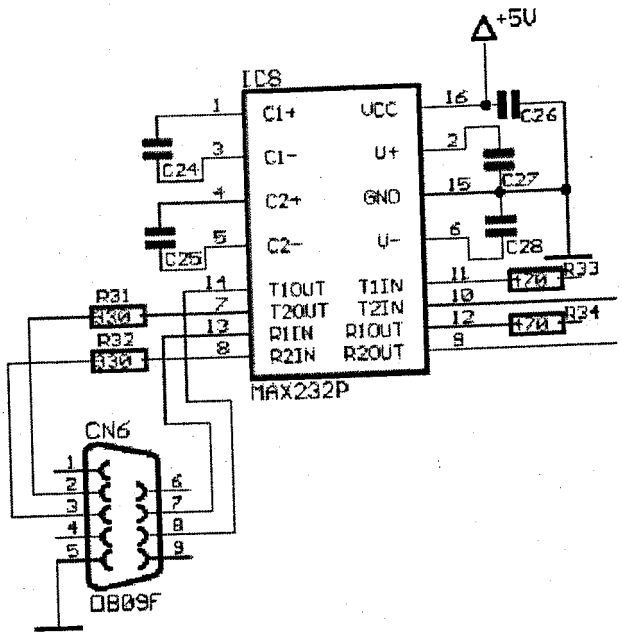

Abbildung 10: Schaltplandetail RS-232-Schnitstelle

4. Software-Komponenten

4.1. Testprogramm für den PIC

Nachdem die Hardware für die Steuerung aufgebaut war, wollten wir zunächst testen, ob wir den Roboterarm überhaupt mit dem PIC steuern können. Bevor wir uns also mit dem vollständigen Programm für den PIC beschäftigt haben, wurde zunächst ein Testprogramm geschrieben.

Aufgabe des Testprogrammes

Die Aufgabe des Testprogrammes ist es, alle Achsen des Roboterarmes jeweils in beide Bewegungsrichtungen zu bewegen.

Da der Roboterarm keine Sensoren besitzt, die die aktuelle Position ermitteln können, soll das Testprogramm den Roboterarm jeweils wieder in die Ausgangsposition zurückfahren, damit das Testprogramm mehrmals hintereinander durchlaufen kann. Um den Roboterarm möglichst exakt wieder in die Ausgangsposition zurück zu bewegen, werden Zeitintervalle verwendet. Das Programm steuert jede Achse des Roboterarmes zunächst für eine bestimmte Dauer in eine Richtung. Anschließend wird die Achse genau so lange in die entgegengesetzte Richtung gesteuert.

1. Der Ablauf des Testprogramms ist dabei wie folgt:
2. Stoppe alle Bewegungen
3. Bewege Achse 1 nach links für ca. 3 Sekunden
4. Bewege Achse 1 nach rechts für ca.3 Sekunden
5. Bewege Achse 2 nach unten für ca. 3 Sekunden
6. Bewege Achse 2 nach oben für ca. 3 Sekunden
7. Bewege Achse 3 nach unten für ca. 3 Sekunden
8. Bewege Achse 3 nach oben für ca. 3 Sekunden
9. Bewege Achse 4 gegen den Uhrzeigersinn für ca. 3 Sekunden
10. Bewege Achse 4 mit dem Uhrzeigersinn für ca. 3 Sekunden
11. Bewege Achse 5 zum Öffnen des Greifers für ca. 3 Sekunden
12. Bewege Achse 5 zum Schließen des Greifers für ca. 3 Sekunden
13. ca. 3 Sekunden warten
14. bei 1. weitermachen

Der Quellcode des Testprogramms

Im folgenden Abschnitt wird auf die Details des Assembler-Programmes eingegangen, welches zur Realisierung des Testprogrammes für den PIC geschrieben wurde.
Teilweise wurden die für das Testprogramm entwickelten Konzepte auch im eigentlichen Steuerprogramm wiederverwendet.

Um das eigentliche Programm übersichtlich und lesbar zu halten, wurden für die Steuerung des Roboterarmes Makros und Konstanten definiert. Dies erlaubte es die Ports und Pins, die für die Ansteuerung der jeweiligen Achse je Richtung zuständig sind, an einer zentralen Stelle im Quellcode zu ändern.
Dies war vor allem bei den ersten Tests sehr hilfreich, da einige der Pins nicht so mit dem PIC verbunden waren, wie wir es dachten. Durch die Zentralisierung der Port und Pin Adressen war eine Änderung des Programms jedoch ohne Probleme möglich.

```
#DEFINE ACHSE1_GEGENUHR    3
#DEFINE ACHSE1_MITUHR      2
#DEFINE ACHSE2_RUNTER      5
#DEFINE ACHSE2_RAUF        4

#DEFINE ACHSE3_RUNTER      0
#DEFINE ACHSE3_RAUF        3
#DEFINE ACHSE4_GEGENUHR    4
#DEFINE ACHSE4_MITUHR      5
#DEFINE ACHSE5_AUF         7
#DEFINE ACHSE5_ZU          6
```

Die Zahlen in der obigen Auflistung stehen jeweils für das Bit, das beim entsprechenden Port gesetzt bzw. gelöscht werden muss um eine Aktion zu stoppen bzw. zu starten.

Das folgende Codebeispiel zeigt, wie die Makros zum Steuern der Achse 1 aufgebaut sind. Der Aufbau der übrigen Steuerungsmakros ist nahezu identisch; es werden lediglich andere Achsen angesteuert.
Für die Namen der Makros wurden nach Möglichkeit sprechende Namen verwendet, um die Lesbarkeit des Quellcodes zu erhöhen.
Die folgenden vier Makros dienen zum Starten und Stoppen der Bewegungen je Bewegungsrichtung. Dazu werden innerhalb der Makros die entsprechenden Bit gesetzt

bzw. gelöscht. Im Ausgangszustand sind alle Bits auf 1 gesetzt. Um eine Bewegung zu starten müssen die Bits auf 0 gesetzt werden.
Der Befehl `bcf` ist für das Löschen eines Bits zuständig. Er kann somit zum Starten einer Bewegung verwendet werden. Der Befehl `bsf` setzt ein bestimmtes Bit auf 1. In unserem Fall kann damit die Bewegung des Roboterarmes gestoppt werden.

```
ACHSE1_LINKS_START    macro
      bcf PORTA, ACHSE1_GEGENUHR
      endm

ACHSE1_LINKS_STOP     macro
      bsf PORTA, ACHSE1_GEGENUHR
      endm

ACHSE1_RECHTS_START   macro
      bcf PORTA, ACHSE1_MITUHR
      endm

ACHSE1_RECHTS_STOP    macro
      bsf PORTA, ACHSE1_MITUHR
      endm
```

Der Hauptteil des Testprogrammes sieht wie folgt aus:

```
main

 STOP_ALL
 ACHSE1_LINKS_START
 call delay ; ca. 3s warten
 ACHSE1_LINKS_STOP
 ACHSE1_RECHTS_START
 call delay ; ca. 3s warten
 ACHSE1_RECHTS_STOP
 ACHSE2_RUNTER_START
 call delay ; ca. 3s warten
 ACHSE2_RUNTER_STOP
 ACHSE2_RAUF_START
 call delay ; ca. 3s warten
 ACHSE2_RAUF_STOP
 ACHSE3_RUNTER_START
 call delay ; ca. 3s warten
 ACHSE3_RUNTER_STOP
 ACHSE3_RAUF_START
 call delay ; ca. 3s warten
 ACHSE3_RAUF_STOP
 ACHSE4_GEGENUHR_START
 call delay ; ca. 3s warten
 ACHSE4_GEGENUHR_STOP
 ACHSE4_MITUHR_START
 call delay ; ca. 3s warten
 ACHSE4_MITUHR_STOP
```

```
ACHSE5_AUF_START
call delay ; ca. 3s warten
call delay
ACHSE5_AUF_STOP
ACHSE5_ZU_START
call delay ; ca. 3s warten
call delay
ACHSE5_ZU_STOP

call delay ; noch etwas warten...
goto  main ; ... und dann wieder zum Anfang
```

Neben den Makroaufrufen wird noch die Unterroutine `delay` aufgerufen. Diese beschäftigt den PIC für ca. 3 Sekunden mit Schleifen zählen.
Am Ende des vollständigen Durchlaufs springt das Programm wieder an den Anfang und es wird erneut durchlaufen. Dies geschieht solange, bis der PIC ausgeschaltet wird.

Der vollständige Quellcode des Assembler-Programmes ist im Anhang enthalten.

4.2. Steuerungsprogramm für den PIC

Gegenüber dem Testprogramm muss das eigentliche Steuerprogramm für den PIC auch in der Lage sein, Kommandos über die serielle Schnittstelle zu empfangen und zu verarbeiten. Dafür sind zusätzliche Behandlungsroutinen notwendig.

Die Struktur zur Ansteuerung der einzelnen Achsen wurde vom Testprogramm übernommen. In diesem Fall sind damit die Makros und Konstanten zur Ansteuerung und Identifizierung der Achsen gemeint.

Zur Kommunikation zwischen PC und PIC wurden die folgenden Kommandos in Form von konstanten Zahlenwerten definiert:

```
#DEFINE    KOMMANDO_ALLE_STOP                    D'1'

#DEFINE    KOMMANDO_ACHSE1_GEGENUHR_START        D'10'
#DEFINE    KOMMANDO_ACHSE1_MITUHR_START          D'11'
#DEFINE    KOMMANDO_ACHSE1_STOP                  D'12'

#DEFINE    KOMMANDO_ACHSE2_RUNTER_START          D'20'
#DEFINE    KOMMANDO_ACHSE2_RAUF_START            D'21'
#DEFINE    KOMMANDO_ACHSE2_STOP                  D'22'

#DEFINE    KOMMANDO_ACHSE3_RUNTER_START          D'30'
#DEFINE    KOMMANDO_ACHSE3_RAUF_START            D'31'
#DEFINE    KOMMANDO_ACHSE3_STOP                  D'32'

#DEFINE    KOMMANDO_ACHSE4_GEGENUHR_START        D'40'
#DEFINE    KOMMANDO_ACHSE4_MITUHR_START          D'41'
#DEFINE    KOMMANDO_ACHSE4_STOP                  D'42'

#DEFINE    KOMMANDO_ACHSE5_AUF_START             D'50'
#DEFINE    KOMMANDO_ACHSE5_ZU_START              D'51'
#DEFINE    KOMMANDO_ACHSE5_STOP                  D'52'
```

Anhand dieser Kommandos, die vom PC gesendet werden, stellt der PIC fest, welche Achse in welche Richtung bewegt werden soll bzw. welche Bewegung gestoppt werden soll.

Anstelle der Kommandotabelle hätten wir auch die genaue Belegung der Ports vom PC aus senden können und diese wäre einfach an den Ports angelegt worden. Diese Methode hat uns jedoch missfallen, da der PIC so im Grunde kaum eigene Logik zur Kontrolle der Achsen und Befehle besessen hätte. Ein weiterer Vorteil der jetzigen Lösung ist der, dass das Programm sehr einfach mit weiteren Befehlen, bspw. zur Ansteuerung weiterer Motoren (z. B. für die Bewegung einer Webcam) oder zum Einschalten von Lichtquellen erweitert werden kann.

Um festzustellen welches Kommando der PIC empfangen hat, wird der aktuelle Kommando-Wert nacheinander von jeder Kommando-Konstante abgezogen. Ergibt diese Subtraktion null, wurde das Kommando gefunden und die entsprechenden Aktionen werden ausgeführt, d. h. die aus dem Testprogramm bereits bekannten Makros werden aufgerufen.

```
kommando_1
     movf    Kommando, w         ; Kommando nach W kopieren
     sublw   KOMMANDO_ALLE_STOP  ; Wert von KOMMANDO_STOP
                                 ; abziehen
     btfss   STATUS, Z           ; Wenn Kommando -
                                 ; KOMMANDO_STOP != 0 dann
     goto kommando_2             ;    fahre mit dem nächsten
                                 ;    Kommando fort
                                 ; Sonst
     STOP_ALL                    ;    alle Achsen-
                                 ;    bewegungen anhalten

     goto kommando_verarbeitet
```

Damit der PIC die oben genannten Kommandos über die serielle Schnittstelle empfangen und verarbeiten kann, wurde zusätzlich eine Interruptbehandlungsroutine geschrieben. Diese wird aufgerufen sobald ein neues Datenbyte an der seriellen Schnittstelle empfangen wurde. Die Routine setzt ein entsprechendes Flag, das vom Hauptprogramm überprüft wird.

```
org 4
     ; Status und Arbeitsregister zwischenspeichern
     movwf    w_temp
     swapf    STATUS, w
     clrf     STATUS
     movwf    status_temp
     movf     PCLATH, w
     movwf    pclatch_temp
     clrf     PCLATH

     ; Wurde ein RS232-Empfänger-Interupt ausgelöst?
     btfss    PIR1,RCIF
     goto interrupt_ende        ; Interrupt kam von woanders

     movfw    RCREG    ; RS232-Register auslesen
     movwf    Kommando ; und in den Speicher
                       ; nach 'Kommando' schreiben
     bsf     NeuesKommandoEmpfangen, 0 ; Flag für gültige
                                       ; Daten setzen
     bcf     PIR1, RCIF

     ; Status und Arbeitsregister wieder zurückschreiben
     interrupt_ende
     movf     pclatch_temp, w
     movwf    PCLATH
     swapf    status_temp, w
     movwf    STATUS
     swapf    w_temp, 1
     swapf    w_temp, w

     retfie               ; zum Programm zurückspringen
```

Das Hauptprogramm initialisiert zunächst sich selbst, den Roboterarm (bzw. die entsprechenden Ausgänge am PIC) sowie die Kommunikation mit dem PC. Anschließend geht es in eine Endlosschleife über. Bei jedem Schleifendurchlauf wird geprüft, ob ein Kommando vom PC empfangen wurde. Ist diese der Fall, werden die entsprechenden Aktionen (wie oben beschrieben) gestartet.

Für die Initialisierung der Interruptbehandlung und der Kommunikation mittels RS232 sind die folgenden Befehle erforderlich:

```
      ; RS232-Schnittstelle initialisieren
      BANK1
    movlw    0x20          ; Sender: RS232
    movwf    TXSTA

      BANK0
    movlw    0x90          ; Empfänger: RS232
    movwf    RCSTA

      ; Baudrate einstellen
    BANK1
    movlw    D'25'         ; Set Baud rate 2.400 BPS bei 4 MHz
    movwf    SPBRG
    bcf      TXSTA, BRGH   ; BRGH=0


      ;Interrupts vorbereiten
      ; BANK 1
    bsf      PIE1,RCIE     ; Enable receive interrupts
    BANK0
    clrf     PIR1          ; alle Interruptflags löschen
    clrf     PIR2
    bsf      INTCON,GIE    ; generell Interrupts erlauben
    bsf      INTCON,PEIE   ; Interrupts von Peripheriegeräten
erlauben
```

Die weiteren Details des Programms können den Kommentaren im Quellcode entnommen werden. Der Quellcode des Steuerungsprogrammes befindet sich im Anhang.

4.3. PC-Steuerprogramm

Zur Steuerung des Roboterarmes bzw. zum Senden der Kommandos über die RS232-Schnittstelle ist ein Programm auf dem PC erforderlich, welches die entsprechenden Kommandokonstanten an die serielle Schnittstelle sendet. Sehr wichtig ist, dass die Baudrate entsprechend auf die im PIC fest programmierte Baudrate eingestellt wird. Die Hardwareflusssteuerung sollte deaktiviert werden.

Prinzipiell könnte für den Einsatz das HyperTerminal von Microsoft Windows verwendet werden. Der Nachteil ist, dass die Steuerung des Roboterarmes damit kaum intuitiv erfolgen kann, da die ASCII-Werte der Kommandos mit der Tastatur unter Windows nur sehr schwer erzeugbar sind.

Um eine etwas elegantere und komfortablere Lösung zu schaffen, wurde ein kleines Java-Programm entwickelt, welches das JavaCOMM-Paket (ein von Sun zur Verfügung gestelltes Paket zur Kommunikation mit diversen Schnittstellen[1]) verwendet.
Ein KeyListener fängt im Programm das Drücken bestimmter Tasten ab und leitet mit Hilfe des JavaCOMM-Paketes das entsprechende Kommando an die serielle Schnittstelle weiter. Im Java-Programm selbst ist die gleiche Kommandotabelle mit Konstanten enthalten wie im Assembler-Steuerprogramm für den PIC. Nur wenn diese Tabellen wertmäßig identisch sind, ist eine abgestimmte Kommunikation und somit eine korrekte Steuerung des Roboterarmes möglich.

In der folgenden Abbildung ist die Benutzeroberfläche des Programmes dargestellt.

1 [javacomm] (http://java.sun.com/products/javacomm/index.jsp)

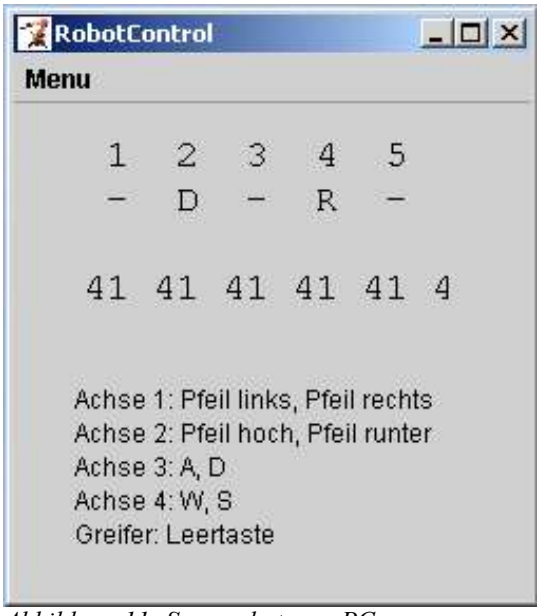

Abbildung 11: Screenshot vom PC-Steuerprogramm

Im oberen Teil des Fenster sind die jeweiligen Achsen durchnumeriert. In der Zeile darunter wird der Status einer Achse dargestellt, d. h. ob diese gerade still steht („-“) oder ob sich diese gerade bewegt („L“ für left, „R“ für right, „D“ für down, „U“ für up, „C“ für close und „O“ für open).
Die Zeile in der Mitte dient lediglich zur Kontrolle und enthält stets die zuletzt gesendeten Kommandos.
Im unteren Bereich des Programms ist lediglich als Hilfe aufgezeigt, welche Tasten für die Steuerung des Roboterarmes notwendig sind.

Die Entwicklung dieses Steuerprogrammes hat es uns ermöglicht, den Roboterarm in ersten Tests sehr präsize steuern zu können. Die Bedienung ist intuitiv und daher sehr schnell zu erlernen.

5. Ausblick

Während der Durchführung des Projektes sind uns zahlreiche Ideen gekommen, die noch mit dem Roboterarm verwirklicht werden könnten.

Durch die einfache Erweiterbarkeit des PIC-Programmes um weitere Kommandos wäre es möglich zusätzliche Beleuchtungen ein- und auszuschalten oder andere Motoren, bspw. zur Bewegung einer Webcam anzusteuern.

Durch den Anbau von Potentiometern und der Nutzung des A/D-Konverters im PIC wäre es denkbar, die Positionen und Lagen der Achsen des Roboterarmes an den PIC und ggf. an den PC zurückzusenden, um eine punktgenaue Steuerung zu erreichen. In diesem Zusammenhang wäre ebenfalls eine PIC-interne gotoPositionXYZ-Funktion denkbar, die die Spitze des Roboterarmes automatisch zu einer bestimmten Position bewegt.

Auch auf der PC-Seite könnten weitere Entwicklungen getätigt werden. So könnten die gedrückten Tastenkombinationen vom Javaprogramm aufgezeichnet und als Makro wiederabgespielt werden, sodass sich der Roboterarm von alleine bewegt.

Mit einer oder mehreren Webcams könnte die Roboterarmszene aufgenommen und als Videostream ins Internet gestellt werden. Mittels einer kleinen Client-Server-Anwendung könnte der Roboterarm dann über ein Netzwerk gesteuert und beobachtet werden.

6. Fazit

Die Bearbeitung der Aufgabe verlief weitestgehend problemlos. Lediglich die nicht durchgängige Nummerierung der Ports am PIC (sowohl Port B als auch Port A) und falsch eingestellte Baudraten bei der RS232-Kommunikation benötigten etwas Zeit zur Fehleraufdeckung, waren dann aber sehr schnell behoben.

An den vielen denkbaren Ideen, die im Ausblick geschildert wurden, erkennt man, dass diesem Projekt im Prinzip kaum Grenzen gesetzt sind.
Und das Schönste ist: Für jeden war etwas dabei. Wer sich gerne mit der Hardware befasst und auch vor dem Einsatz eines Lötkolben nicht zurückscheut kann eben so viele und gute Ideen einbringen wie jemand der sich lieber auf die Programmierung von PC-Software beschränkt.

Dennoch hat dies Projekt für uns vorerst ein Ende und wir blicken auf eine sehr interessante und praktische Vorlesung (so sollte man es eigentlich kaum nennen) zurück, welche uns (endlich) tiefe Einblicke in den Aufbau und die Funktionsweise von Mikrochips gegeben hat und uns die Kenntnisse aus der Physik und der Rechnerarchitektur anwenden ließ. Wir hoffen, dass diese Vorlesung auch zukünftigen Semestern weiterhin angeboten werden kann.

7. Quellen

[javacomm]
Java Core - Java Communications
http://java.sun.com/products/javacomm/index.jsp

[sprut]
Sprut PIC Microchip Controller
http://www.sprut.de/electronic/pic/

8. Anhang

Ausdruck:

- Assembler-Listings
    - Steuerungsprogramm PIC
    - Testprogramm PIC

CD-ROM:

- Ausarbeitung: Dokumentation_Robo.pdf
- Präsentation: Referat_RS232.ppt
- Assembler-Listings
    - Steuerungsprogramm PIC
    - Testprogramm PIC
- PC-Steuerprogramm (Javaprojekt AppRobotControl)
- Fotos des Aufbaus
- Schaltpläne
    - Optokoppler-Board (eagle-Format)
    - Skizzen Roboterarm intern
- Datenblatt Optokoppler PC847

Assembler-Listing Steuerungsprogramm PIC

```
;*******************************************************************************
;*******************************************************************************
;******* PROGRAMM ZUR STEUERUNG EINES ROBOTERARMES UNTER *********************
;******* VERWENDUNG DER RS232-SCHNITTSTELLE **********************************
;*******************************************************************************
;*******************************************************************************

;******** Beschreibung ********************************************************



;******* Header ****************************************************************
 PROCESSOR 16F877
 INCLUDE P16F877.INC
 RADIX DEC
               __IDLOCS H'0100'
               __CONFIG _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC & _LVP_OFF


;******* Konstanten festlegen **************************************************

; Initialisierungszustände
#DEFINE    INIT_PORTA                          B'00111100'
#DEFINE    INIT_PORTB                          B'11111001'

#DEFINE    INTERRUPTED                         D'0'
#DEFINE    MAINTEST                            D'1'

; Zu setzende Bits PORTA
#DEFINE    ACHSE1_GEGENUHR                     D'3'
#DEFINE    ACHSE1_MITUHR                       D'2'
#DEFINE    ACHSE2_RUNTER                       D'5'
#DEFINE    ACHSE2_RAUF                         D'4'

; Zu setzende Bits PORTB
#DEFINE    ACHSE3_RUNTER                       D'0'
#DEFINE    ACHSE3_RAUF                         D'3'
#DEFINE    ACHSE4_GEGENUHR                     D'4'
#DEFINE    ACHSE4_MITUHR                       D'5'
#DEFINE    ACHSE5_AUF                          D'7'
#DEFINE    ACHSE5_ZU                           D'6'


; Kommando-Konstanten
; (identisch mit den im PC-Steuerprogramm verwendeten Konstanten)
#DEFINE    KOMMANDO_ALLE_STOP                   D'1'

#DEFINE    KOMMANDO_ACHSE1_GEGENUHR_START      D'10'
#DEFINE    KOMMANDO_ACHSE1_MITUHR_START        D'11'
#DEFINE    KOMMANDO_ACHSE1_STOP                D'12'

#DEFINE    KOMMANDO_ACHSE2_RUNTER_START        D'20'
#DEFINE    KOMMANDO_ACHSE2_RAUF_START          D'21'
#DEFINE    KOMMANDO_ACHSE2_STOP                D'22'

#DEFINE    KOMMANDO_ACHSE3_RUNTER_START        D'30'
#DEFINE    KOMMANDO_ACHSE3_RAUF_START          D'31'
#DEFINE    KOMMANDO_ACHSE3_STOP                D'32'

#DEFINE    KOMMANDO_ACHSE4_GEGENUHR_START      D'40'
#DEFINE    KOMMANDO_ACHSE4_MITUHR_START        D'41'
#DEFINE    KOMMANDO_ACHSE4_STOP                D'42'

#DEFINE    KOMMANDO_ACHSE5_AUF_START           D'50'
#DEFINE    KOMMANDO_ACHSE5_ZU_START            D'51'
#DEFINE    KOMMANDO_ACHSE5_STOP                D'52'


;******* Variablen deklarieren *************************************************

; Variablen zum Zwischenspeichern von Status und W
w_temp                  equ 0x20
status_temp             equ 0x21
pclatch_temp            equ 0x22

; Variablen für die Achsensteuerung
Kommando                equ 0x30
NeuesKommandoEmpfangen  equ 0x31

;count equ 0x40
;count197118 equ 0x41
;count768 equ 0x42

;******* Programmspeicherzelle 0 ***********************************************
```

```
org 0x00
                goto init ; springt zur Initialisierung


;******* Makrodefinition ****************************************************

; Makros zum Wechseln der Speicherbänke
BANK0           macro
                bcf          STATUS,RP0    ;select page 0
                bcf    STATUS,RP1          ;
                endm

BANK1           macro
                bsf          STATUS,RP0    ;select page 1
                bcf          STATUS,RP1    ;
                endm

BANK2           macro
                bcf    STATUS,RP0          ;select page 2
                bsf          STATUS,RP1    ;
                endm

BANK3           macro
                bsf          STATUS,RP0    ;select page 3
                bsf          STATUS,RP1    ;
                endm


;******* Achsen-Bewegungen **************************************************

; Stoppt sämtliche Bewegungen des Roboterarmes
; (Startzustand)
STOP_ALL        macro
                movlw    INIT_PORTA        ;B'00001111' ;0x0F
                movwf    PORTA
                movlw    INIT_PORTB        ;0x3F
                movwf    PORTB
                endm

;----------------------------
; Achsen-Bewegungen:
; Für jede Achse gibt es zwei Makros ACHSEx_richtung_START, welche die
; jeweilige Bewegung in eine Richtung aktivieren. Mit dem Makro
; ACHSEx_STOP kann die Bewegung der Achse gestoppt werden.
; Die Makros setzen jeweils entsprechende Bits an PORTA und PORTB, wobei
; eine 1 für Stillstand, eine 0 für Bewegung steht.

;----------------------------
; Bewegung von Achse 1
; (Achse am Sockel)
; gegen den Uhrzeigersinn
ACHSE1_GEGENUHR_START    macro
                bcf PORTA, ACHSE1_GEGENUHR
                endm

ACHSE1_MITUHR_START      macro
                bcf PORTA, ACHSE1_MITUHR
                endm

ACHSE1_STOP     macro
                bsf PORTA, ACHSE1_GEGENUHR
                bsf PORTA, ACHSE1_MITUHR
                endm

;----------------------------
; Bewegung von Achse 2
; nach vorne (bzw. unten)
ACHSE2_RUNTER_START      macro
                bcf PORTA, ACHSE2_RUNTER
                endm

ACHSE2_RAUF_START        macro
                bcf PORTA, ACHSE2_RAUF
                endm

ACHSE2_STOP     macro
                bsf PORTA, ACHSE2_RUNTER
                bsf PORTA, ACHSE2_RAUF
                endm

;----------------------------
; Bewegung von Achse 3
; nach unten
ACHSE3_RUNTER_START      macro
                bcf PORTB, ACHSE3_RUNTER
                endm

ACHSE3_RAUF_START        macro
                bcf PORTB, ACHSE3_RAUF
                endm
```

```
ACHSE3_STOP     macro
                bsf PORTB, ACHSE3_RUNTER
                bsf PORTB, ACHSE3_RAUF
                endm

;----------------------------
; Bewegung von Achse 4
; (Drehgelenk am Greifer)
; gegen den Uhrzeigersinn
ACHSE4_GEGENUHR_START    macro
                bcf PORTB, ACHSE4_GEGENUHR
                endm

ACHSE4_MITUHR_START      macro
                bcf PORTB, ACHSE4_MITUHR
                endm

ACHSE4_STOP     macro
                bsf PORTB, ACHSE4_GEGENUHR
                bsf PORTB, ACHSE4_MITUHR
                endm

;----------------------------
; Öffnen vom Greifer
; (Achse 5)
ACHSE5_AUF_START macro
                bcf PORTB, ACHSE5_AUF
                endm

ACHSE5_ZU_START  macro
                bcf PORTB, ACHSE5_ZU
                endm

ACHSE5_STOP      macro
                bsf PORTB, ACHSE5_AUF
                bsf PORTB, ACHSE5_ZU
                endm


;******* Programmspeicherzelle 4 (Interruptbehandlung) ************************

; Wenn ein Byte vom PC empfangen wird, wird es in der
; Variablen Kommando hinterlegt. Durch das Setzen eines
; Bits in der Variable NeuesKommandoEmpfangen kann die
; Hauptprogrammschleife im Anschluss registrieren, dass
; ein neues Kommando vorliegt.

org 4

; Status und Arbeitsregister zwischenspeichern
                movwf    w_temp
    swapf    STATUS, w
                clrf     STATUS
    movwf    status_temp
                movf     PCLATH, w
                movwf    pclatch_temp
                clrf     PCLATH

; Wurde ein RS232-Empfänger-Interupt ausgelöst?
    btfss    PIR1,RCIF
    goto interrupt_ende                  ; Interrupt kam von woanders

    movfw    RCREG                       ; RS232-Register auslesen
    movwf    Kommando                    ; und in den Speicher nach 'Kommando' schreiben
    bsf      NeuesKommandoEmpfangen, 0   ; Kennzeichen für gültige Daten setzen
                bcf      PIR1, RCIF

; Status und Arbeitsregister wieder zurückschreiben
interrupt_ende
                movf     pclatch_temp, w
                movwf    PCLATH
    swapf    status_temp, w
    movwf    STATUS
    swapf    w_temp, 1
    swapf    w_temp, w

    retfie                               ; zum Programm zurückspringen
```

```
;******* Initialisierung *********************************************************

; Initialisierung von PORTA und PORTB. Die jeweiligen Pins
; werden als Ausgänge gesetzt und es wird der Ausgangszustand
; (keine Achsenbewegung) an den Ports angelegt.

init

                BANK0

                ; Initialisierung von PORTA
                movlw    INIT_PORTA
                movwf    PORTA

                ; Alle PORTA-Pins als digitale Outputs konfigurieren
                BANK1

                movlw    0x06
                movwf    ADCON1
                movlw    B'00000000'
                movwf    TRISA

                ; Initialisierung von PORTB
                BANK0

                movlw    INIT_PORTB
                movwf    PORTB

                ; Alle PORTB-Pins als digitale Outputs konfigurieren
                BANK1

                movlw    B'00000000'
                movwf    TRISB

                ; RS232-Schnittstelle initialisieren
                BANK1
                movlw    0x20             ; Sender: RS232
                movwf    TXSTA

                BANK0
                movlw    0x90             ; Empfänger: RS232
                movwf    RCSTA

                ; Baudrate einstellen
                BANK1
                movlw    D'25'                     ; Set Baud rate 2.400 BPS bei 4 MHz
                movwf    SPBRG
                bcf      TXSTA, BRGH       ; BRGH=0


                ;Interrupts vorbereiten
                ; BANK 1
                bsf      PIE1,RCIE     ; Enable receive interrupts
                BANK0
                clrf     PIR1          ; alle Interruptflags löschen
                clrf     PIR2
                bsf      INTCON,GIE    ; generell Interrupts erlauben
                bsf      INTCON,PEIE   ; Interrupts von Peripheriegeräten erlauben


 goto main ; Hauptprogrammschleife ausführen
```

```
;******* Hauptprogramm *********************************************************

main
                btfss    NeuesKommandoEmpfangen, 0
                goto main ; ... und dann wieder zum Anfang


kommando_ausfuehren


; Nun wird das empfangene Kommando ausgeführt.
; Durch die Subtraktion des empfangenen Kommandos von einer vordefinierten
; Konstante aus der Kommando-Tabelle kann festgestellt werden, um welches
; jeweilige Kommando es sich handelt und die entsprechende Achsenbewegung
; kann durchgeführt werden.
; Beim Starten einer Achsenbewegung wird jeweils sicherheitshalber die
; möglicherweise bereits aktivierte gegenläufige Bewegung gestoppt.


;-------------------------------------------------------------------------------
; Kommando STOP_ALL
kommando_1
                movf    Kommando, w                ; Kommando nach W kopieren
                sublw   KOMMANDO_ALLE_STOP         ; Wert von KOMMANDO_STOP abziehen
                btfss   STATUS, Z                  ; Wenn Kommando - KOMMANDO_STOP != 0 dann
                goto kommando_2                    ;       fahre mit dem nächsten Kommando fort
                                                   ; Sonst
                STOP_ALL                           ;       halte alle Achsenbewegungen anhalten

                goto kommando_verarbeitet


;-------------------------------------------------------------------------------
; Kommando ACHSE1_GEGENUHR_START
kommando_2
                movf     Kommando, 0               ; Kommando nach W kopieren
                sublw    KOMMANDO_ACHSE1_GEGENUHR_START     ; Wert von Kommando-Konstante abziehen
                btfss    STATUS, Z                 ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_3                    ;   fahre mit dem nächsten Kommando fort

                ; Sonst
                ACHSE1_STOP
                ;   halte mögliche Bewegung an
                ACHSE1_GEGENUHR_START              ;   führe Achsenbewegung durch

                goto kommando_verarbeitet

;--------------------------------------
; Kommando ACHSE1_MITUHR_START
kommando_3
                movf     Kommando, w                       ; Kommando nach W kopieren
                sublw    KOMMANDO_ACHSE1_MITUHR_START      ; Wert von Kommando-Konstante abziehen
                btfss    STATUS, Z                 ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_4                    ;   fahre mit dem nächsten Kommando fort
                                                   ; Sonst
                ACHSE1_STOP                        ;   halte mögliche Bewegung an
                ACHSE1_MITUHR_START                ;   führe Achsenbewegung durch

                goto kommando_verarbeitet

;--------------------------------------
; Kommando ACHSE1_STOP
kommando_4
                movf    Kommando, w                ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE1_STOP       ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                  ; Wenn Kommandokonstante - Kommando != 0 dann
                goto kommando_5                    ;   fahre mit dem nächsten Kommando fort
                                                   ; Sonst
                ACHSE1_STOP                        ;   halte Achsenbewegung an

                goto kommando_verarbeitet


;-------------------------------------------------------------------------------
; Kommando ACHSE2_RUNTER_START
kommando_5
                movf    Kommando, w                       ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE2_RUNTER_START      ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                  ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_6                    ;   fahre mit dem nächsten Kommando fort
                                                   ; Sonst
                ACHSE2_STOP                        ;   halte mögliche Bewegung an
                ACHSE2_RUNTER_START                ;   führe Achsenbewegung durch

                goto kommando_verarbeitet
```

```
;--------------------------------------
; Kommando ACHSE2_RAUF_START
kommando_6
                movf    Kommando, w                         ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE2_RAUF_START          ; Wert von Kommando-Konstante abziehen
                btfss     STATUS, Z                 ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_7                     ;   fahre mit dem nächsten Kommando fort
                                                    ; Sonst
                ACHSE2_STOP                         ;   halte mögliche Bewegung an
                ACHSE2_RAUF_START                   ;   führe Achsenbewegung durch

                goto kommando_verarbeitet

;--------------------------------------
; Kommando ACHSE2_STOP
kommando_7
                movf    Kommando, w                 ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE2_STOP        ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                   ; Wenn Kommandokonstante - Kommando != 0 dann
                goto kommando_8                     ;   fahre mit dem nächsten Kommando fort
                                                    ; Sonst
                ACHSE2_STOP                         ;   halte Achsenbewegung an

                goto kommando_verarbeitet


;-------------------------------------------------------------------------------
; Kommando ACHSE3_RUNTER_START
kommando_8
                movf    Kommando, w                         ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE3_RUNTER_START        ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                   ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_9                     ;   fahre mit dem nächsten Kommando fort
                                                    ; Sonst
                ACHSE3_STOP                         ;   halte mögliche Bewegung an
                ACHSE3_RUNTER_START                 ;   führe Achsenbewegung durch

                goto kommando_verarbeitet

;--------------------------------------
; Kommando ACHSE3_RAUF_START
kommando_9
                movf    Kommando, w                         ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE3_RAUF_START          ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                   ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_10                    ;   fahre mit dem nächsten Kommando fort
                                                    ; Sonst
                ACHSE3_STOP                         ;   halte mögliche Bewegung an
                ACHSE3_RAUF_START                   ;   führe Achsenbewegung durch

                goto kommando_verarbeitet

;--------------------------------------
; Kommando ACHSE3_STOP
kommando_10
                movf    Kommando, w                 ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE3_STOP        ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                   ; Wenn Kommandokonstante - Kommando != 0 dann
                goto kommando_11                    ;   fahre mit dem nächsten Kommando fort
                                                    ; Sonst
                ACHSE3_STOP                         ;   halte Achsenbewegung an

                goto kommando_verarbeitet


;-------------------------------------------------------------------------------
; Kommando ACHSE4_GEGENUHR_START
kommando_11
                movf    Kommando, w                         ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE4_GEGENUHR_START      ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                   ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_12                    ;   fahre mit dem nächsten Kommando fort
                                                    ; Sonst
                ACHSE4_STOP                         ;   halte mögliche Bewegung an
                ACHSE4_GEGENUHR_START               ;   führe Achsenbewegung durch

                goto kommando_verarbeitet

;--------------------------------------
; Kommando ACHSE4_MITUHR_START
kommando_12
                movf    Kommando, w                         ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE4_MITUHR_START        ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                   ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_13                    ;   fahre mit dem nächsten Kommando fort
                                                    ; Sonst
                ACHSE4_STOP                         ;   halte mögliche Bewegung an
                ACHSE4_MITUHR_START                 ;   führe Achsenbewegung durch

                goto kommando_verarbeitet
```

```
;-------------------------------------
; Kommando ACHSE4_STOP
kommando_13
                movf    Kommando, w               ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE4_STOP      ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                 ; Wenn Kommandokonstante - Kommando != 0 dann
                goto kommando_14                  ;   fahre mit dem nächsten Kommando fort
                                                  ; Sonst
                ACHSE4_STOP                       ;   halte Achsenbewegung an

                goto kommando_verarbeitet


;-------------------------------------------------------------------------------
; Kommando ACHSE5_AUF_START
kommando_14
                movf    Kommando, w               ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE5_AUF_START ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                 ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_15                  ;   fahre mit dem nächsten Kommando fort
                                                  ; Sonst
                ACHSE5_STOP                       ;   halte mögliche Bewegung an
                ACHSE5_AUF_START                  ;   führe Achsenbewegung durch

                goto kommando_verarbeitet

;-------------------------------------
; Kommando ACHSE5_ZU_START
kommando_15
                movf    Kommando, w               ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE5_ZU_START  ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                 ; Wenn Kommandokonstante - Konstante != 0 dann
                goto kommando_16                  ;   fahre mit dem nächsten Kommando fort
                                                  ; Sonst
                ACHSE5_STOP                       ;   halte mögliche Bewegung an
                ACHSE5_ZU_START                   ;   führe Achsenbewegung durch

                goto kommando_verarbeitet

;-------------------------------------
; Kommando ACHSE5_STOP
kommando_16
                movf    Kommando, w               ; Kommando nach W kopieren
                sublw   KOMMANDO_ACHSE5_STOP      ; Wert von Kommando-Konstante abziehen
                btfss   STATUS, Z                 ; Wenn Kommandokonstante - Kommando != 0 dann
                goto kommando_verarbeitet         ;   fahre mit dem nächsten Kommando fort
                                                  ; Sonst
                ACHSE5_STOP                       ;   halte Achsenbewegung an

                goto kommando_verarbeitet



;-------------------------------------------------------------------------------
; Kommando wurde verarbeitet, nun wird noch etwas aufgeräumt und
; zur Hauptprogrammschleife zurückgesprungen
kommando_verarbeitet

    clrf    NeuesKommandoEmpfangen

    goto    main



;******* Vergleichsroutine *********************************************************

; Vergleicht die Variable KOMMANDO mit dem Wert im Arbeitsregister
; Sind beide gleich, wird

;******* Programmende ***********************************
 end


;*******************************************************************************
;*******************************************************************************
;*******************************************************************************
```

Assembler-Listing Testprogramm PIC

```
 ;******* Header ***********
 PROCESSOR 16F877
 INCLUDE P16F877.INC
 RADIX DEC
      __IDLOCS H'0100'
      __CONFIG _CP_OFF & _PWRTE_ON & _WDT_OFF & _HS_OSC

;******* Konstanten festlegen
#DEFINE ACHSE1_GEGENUHR 3
#DEFINE ACHSE1_MITUHR 2
#DEFINE ACHSE2_RUNTER 5
#DEFINE ACHSE2_RAUF 4

#DEFINE ACHSE3_RUNTER 0
#DEFINE ACHSE3_RAUF 3
#DEFINE ACHSE4_GEGENUHR 4
#DEFINE ACHSE4_MITUHR 5
#DEFINE ACHSE5_AUF 7
#DEFINE ACHSE5_ZU 6

;******* Variablen deklarieren ****************************

; Zählvariablen
count equ         0x21
count768 equ      0x22
count197118 equ   0x23

; Variablen zum Zwischenspeichern von Status und W


;******* Makrodefinition **********************************

; Makros zum Wechseln der Speicherbänke
BANK0 macro
      bcf   STATUS,RP0  ;select page 0
      bcf   STATUS,RP1  ;
      endm

BANK1 macro
      bsf   STATUS,RP0  ;select page 1
      bcf   STATUS,RP1  ;
      endm

BANK2 macro
      bcf   STATUS,RP0  ;select page 2
      bsf   STATUS,RP1  ;
      endm

BANK3 macro
      bsf   STATUS,RP0  ;select page 3
      bsf   STATUS,RP1  ;
      endm
```

```
;******* Achsen-Bewegungen *********************************

; Stoppt sämtliche Bewegungen des Roboterarmes
; (Startzustand)
STOP_ALL    macro
      movlw B'00111100' ;B'00001111' ;0x0F
      movwf PORTA
      movlw B'11111001' ;0x3F
      movwf PORTB
      endm

;----------------------------
; Achsen-Bewegungen:
; Für jede Bewegung gibt es ein Start- und ein Stopmakro,
; welche das entsprechende Bit an PORTA oder PORTB
; auf 0 (Start) oder 1 (Stop) setzen.

;----------------------------
; Bewegung von Achse 1
; (Achse am Sockel)
; gegen den Uhrzeigersinn
ACHSE1_LINKS_START      macro
      bcf PORTA, ACHSE1_GEGENUHR
      endm

ACHSE1_LINKS_STOP macro
      bsf PORTA, ACHSE1_GEGENUHR
      endm

;----------------------------
; Bewegung von Achse 1
; mit dem Uhrzeigersinn
ACHSE1_RECHTS_START     macro
      bcf PORTA, ACHSE1_MITUHR
      endm

ACHSE1_RECHTS_STOP      macro
      bsf PORTA, ACHSE1_MITUHR
      endm

;----------------------------
; Bewegung von Achse 2
; nach vorne (bzw. unten)
ACHSE2_RUNTER_START     macro
      bcf PORTA, ACHSE2_RUNTER
      endm

ACHSE2_RUNTER_STOP      macro
      bsf PORTA, ACHSE2_RUNTER
      endm

;----------------------------
; Bewegung von Achse 2
; nach hinten (bzw. oben)
ACHSE2_RAUF_START macro
      bcf PORTA, ACHSE2_RAUF
      endm

ACHSE2_RAUF_STOP  macro
      bsf PORTA, ACHSE2_RAUF
      endm
```

```
;----------------------------
; Bewegung von Achse 3
; nach unten
ACHSE3_RUNTER_START     macro
      bcf PORTB, ACHSE3_RUNTER
      endm

ACHSE3_RUNTER_STOP      macro
      bsf PORTB, ACHSE3_RUNTER
      endm

;----------------------------
; Bewegung von Achse 3
; nach oben
ACHSE3_RAUF_START macro
      bcf PORTB, ACHSE3_RAUF
      endm

ACHSE3_RAUF_STOP  macro
      bsf PORTB, ACHSE3_RAUF
      endm

;----------------------------
; Bewegung von Achse 4
; (Drehgelenk am Greifer)
; gegen den Uhrzeigersinn
ACHSE4_GEGENUHR_START   macro
      bcf PORTB, ACHSE4_GEGENUHR
      endm

ACHSE4_GEGENUHR_STOP    macro
      bsf PORTB, ACHSE4_GEGENUHR
      endm

;----------------------------
; Bewegung von Achse 4
; mit dem Uhrzeigersinn
ACHSE4_MITUHR_START     macro
      bcf PORTB, ACHSE4_MITUHR
      endm

ACHSE4_MITUHR_STOP      macro
      bsf PORTB, ACHSE4_MITUHR
      endm

;----------------------------
; Öffnen vom Greifer
; (Achse 5)
ACHSE5_AUF_START  macro
      bcf PORTB, ACHSE5_AUF
      endm

ACHSE5_AUF_STOP         macro
      bsf PORTB, ACHSE5_AUF
      endm

;----------------------------
; Schließen vom Greifer
; (Achse 5)
ACHSE5_ZU_START         macro
      bcf PORTB, ACHSE5_ZU
      endm
```

```
ACHSE5_ZU_STOP          macro
      bsf PORTB, ACHSE5_ZU
      endm


;******* Initialisierung **********************************

 org 0x000 ; hex 0x000 ist erste Programmspeicherzelle
      BANK0

      ; Initialisierung von PORTA
      movlw B'00111100'
      movwf PORTA

      ; Alle PORTA-Pins als digitale Outputs konfigurieren
      BANK1

      movlw 0x06
      movwf ADCON1
      movlw 0x00
      movwf TRISA

      ; Initialisierung von PORTB
      BANK0

      movlw B'11111001'
      movwf PORTB

      ; Alle PORTB-Pins als digitale Outputs konfigurieren
      BANK1

      movlw B'00000000'
      movwf TRISB

      BANK0

; Hauptprogrammschleife ausführen
 goto main


;******* Hauptprogramm ************************************
main

 STOP_ALL
 ACHSE1_LINKS_START
 call delay ; ca. 3s warten
 ACHSE1_LINKS_STOP
 ACHSE1_RECHTS_START
 call delay ; ca. 3s warten
 ACHSE1_RECHTS_STOP
 ACHSE2_RUNTER_START
 call delay ; ca. 3s warten
 ACHSE2_RUNTER_STOP
 ACHSE2_RAUF_START
 call delay ; ca. 3s warten
 ACHSE2_RAUF_STOP
 ACHSE3_RUNTER_START
 call delay ; ca. 3s warten
 ACHSE3_RUNTER_STOP
 ACHSE3_RAUF_START
 call delay ; ca. 3s warten
 ACHSE3_RAUF_STOP
 ACHSE4_GEGENUHR_START
```

```
 call delay ; ca. 3s warten
 ACHSE4_GEGENUHR_STOP
 ACHSE4_MITUHR_START
 call delay ; ca. 3s warten
 ACHSE4_MITUHR_STOP
 ACHSE5_AUF_START
 call delay ; ca. 3s warten
 call delay
 ACHSE5_AUF_STOP
 ACHSE5_ZU_START
 call delay ; ca. 3s warten
 call delay
 ACHSE5_ZU_STOP

 call delay ; noch etwas warten...
 goto  main ; ... und dann wieder zum Anfang

; ***** delay *****
delay
 movlw 0x0F
 movwf count ; kopiere w nach count
repeat
 call delay197118 ;
 decfsz count,f ; counter dekrementieren falls 0 überspringen
 goto repeat ;
 return ; call delay verlassen
; ***** delay197118 *****
delay197118
 movlw 0xff ;
 movwf count197118 ;
repeat197118
 call delay768 ;
 decfsz count197118,f ; counter dekrementieren falls 0 überspringen
 goto repeat197118 ;
 return ; call delay197118 verlassen
; ***** delay768 *****
delay768 ; delay mit Gesamtdauer 768 Takte
 movlw 0xff ;
 movwf count768 ;
repeat768
 decfsz count768,f ;
 goto repeat768 ;
 return ;

;******* Programm Ende ******
 end
```