

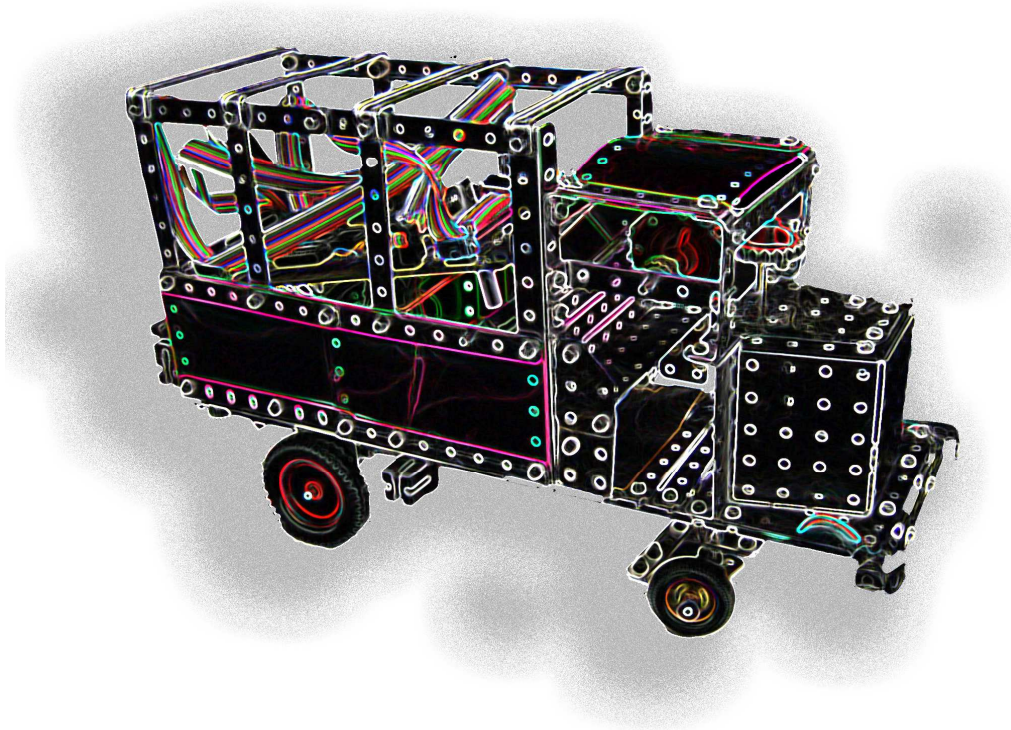
Deutsches Gymnasium Biel

Maturjahrgang 2005

Mobiler Roboter

gesteuert von einem Microcontroller

(Kurztitel: Mobiler Roboter)



Eine Maturaarbeit von
Michael Bögli, Klasse 1e

Betreuung: W. Schöchlin

1. Inhaltsübersicht

	Seite
2. Vorwort	3
3. Microcontroller.....	4
3.1 Was ist ein Microcontroller?	4
3.2 Geschichte der Microcontroller.....	4
3.3 Aufbau eines Microcontrollers.....	5
3.4 Schematische Darstellung des Microcontroller – Boards	6
3.5 Programmierung der Microcontroller	6
3.5.1 Assemblersprache.....	6
3.5.2 Das binäre und hexadezimale Zahlensystem	7
3.5.3 Die wichtigsten Befehle für meine Arbeit.....	8
4. Mobiler Roboter mit Microcontroller gesteuert	10
4.1 Ziel	10
4.2 Materialien	10
4.3 Verstärkerschaltung.....	11
4.4 Grundgerüst.....	12
4.5 Sensorschaltung.....	12
4.6 Relais-Schaltung.....	14
4.7 Programmierung der Software	15
5. Schlussfolgerungen	16
6. Zusammenfassung.....	17
7. Quellenverzeichnis.....	18
8. Anhang	20
8.1 Sensorschaltung.....	20
8.2 Quellcode des Programms zur Steuerung des Roboters.....	21
8.3 Quellen aus dem Internet.....	23

2. Vorwort

Da ich mich schon früh für die Technik von Maschinen und später vor allem für Computer interessierte, war es für mich naheliegend bei der Themenwahl der Maturaarbeit im Bereich der Informatik zu suchen. Ich war interessiert davon zu hören, dass das Deutsche Gymnasium Microcontroller besitzt, die zurzeit nicht in Gebrauch sind. Daher beschloss ich einen Microcontroller mit nach Hause zu nehmen um diverse Funktionen auszutesten. Ich entschied mich später im Rahmen der Maturaarbeit mich weiter mit Microcontroller zu befassen und daraus einen Roboter zu bauen.

Die nachfolgende Arbeit ist in zwei Schwerpunkte unterteilt. Zum einen erkläre ich das Prinzip und die Geschichte der Microcontroller und zum anderen beschreibe ich die praktische Anwendung des Microcontrollers am Beispiel des Roboters, den ich konstruiert habe.

3. Microcontroller

3.1 Was ist ein Microcontroller?

Ein „normaler“ Computer besteht aus einem Prozessor und vielen anderen Komponenten, wie zum Beispiel Schnittstellen oder Speicher. Schnittstellen verarbeiten Daten, damit sie von weiteren Geräten (z.B. Drucker, Bildschirme usw.) genutzt werden können. Dank der heutigen hohen Integrationsdichte können sämtliche Geräte auf einem einzigen Chip untergebracht werden. Man spricht dabei von einem Einchipcomputer oder eben einem Microcontroller. [1]

Der Microcontroller ist meistens in einem sogenannten Entwicklungssystem eingebaut, bevor er richtig zum Einsatz kommt. In einem Entwicklungssystem sind alle extern benötigten Geräte übersichtlich und praktisch angeordnet (wie im Bild 1). So kann man auf einfache Weise den Chip programmieren und die Hardware verändern und testen. Ist die Entwicklung einmal abgeschlossen, wird der Microcontroller in ein kompakteres System (z.B. in ein Mobiltelefon) eingebaut. Ein solches System ist viel unübersichtlicher und nicht so einfach umzubauen und somit auch nicht geeignet für die Entwicklung.

Der Mikroprozessor ist dagegen ein auf einen Chip konzipierter Prozessor. Der Mikroprozessor enthält keine weiteren Geräte wie Speicher oder Schnittstellen. Dafür hat der Mikroprozessor meistens eine höhere Taktfrequenz und eine grössere Registerbreite als ein Microcontroller. [2]

3.2 Geschichte der Microcontroller

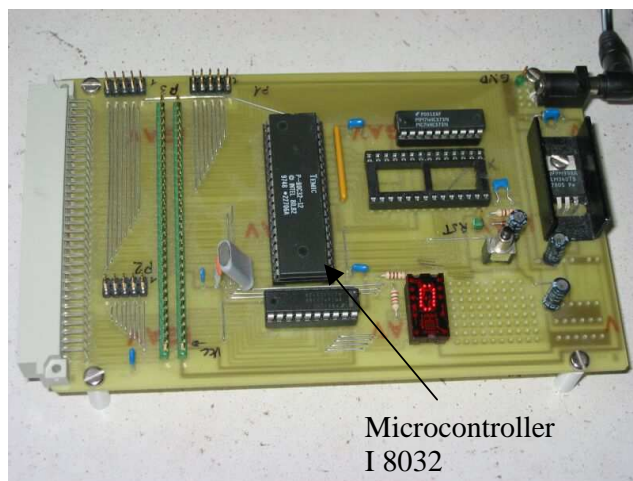


Bild 1: Microcontrollerboard

Die Grundidee des Microcontrollers war ein ganzes Computersystem auf einem einzigen Chip unterzubringen (System-On-a-Chip [SOC] Technology). Ursprünglich waren die sogenannten MPU's (MicroProcessor Unit) noch auf mehreren Chips aufgeteilt. Um eine höhere Geschwindigkeit und eine bessere Leistungsfähigkeit zu erreichen, wurde später alles auf einen Chip konzipiert.

Die Geschichte des Microcontrollers kann durch zwei parallele Entwicklungspfade verfolgt werden: die von Intel und die von Texas Instruments. Der erste Mikroprozessor war 1971 der Intel 4004 (2300 Transistoren, 1 MHz

Clock). Nach zehn Jahren Entwicklung und zahlreichen weiteren Controllern gab Intel den Microcontroller 8051 heraus. Mit dessen Nachfolgermodell (dem Intel 8032, siehe Bild 1) werde ich meine Maturaarbeit durchführen. Das erste wirklich auf einem Chip konzipierte Computersystem (Microcontroller) war der TMS1000 von Texas Instruments. Dieser beinhaltet neben einem ROM (Read Only Memory) und einem RAM (Random Access Memory) natürlich noch den Prozessor (MPU) und den Taktgeber in einem modernen Design (auf die einzelnen Begriffe komme ich später zurück). [1], [3]

3.3 Aufbau eines Microcontrollers

Der Zweck des Microcontrollers ist meistens die Prozesssteuerung oder eine spezifische Funktion. [3]

Ein Minimalsystem eines Microcontrollerboards besteht aus:

- einem RAM (Random Access Memory): eine temporäre Datenspeicherungsmöglichkeit, welche nach abschalten des Stromes jedes mal gelöscht wird.
- einem ROM (Read Only Memory): eine Datenspeicherungsmöglichkeit, welche nur gelesen und nichts darauf geschrieben werden kann. Üblicherweise wird das Programm darin gespeichert, denn dies wird schon bei der Fabrikation in das ROM geschrieben und beim Gebrauch muss das Programm nur noch gelesen und nicht mehr verändert werden können.

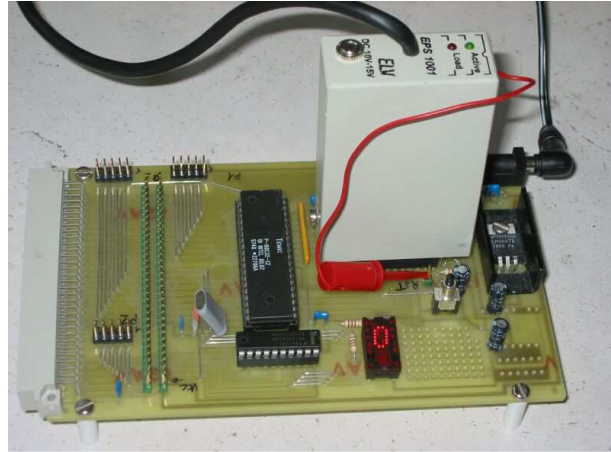


Bild 2: Microcontrollerboard mit aufgestecktem EPROM-Simulator

Ein Beispiel eines ROM ist, wie es der Name schon sagt, die CD-ROM. Bei einer CD-ROM wird das Programm schon bei der Fabrikation geschrieben und kann später nur noch gelesen werden. Eine spezielle Form von ROMs sind die so genannten PROMs (Programmable Read only Memory). Diese können genau einmal beschrieben werden und danach nur noch gelesen werden (ähnlich wie eine CD-R). Häufig werden in Microcontroller aber EPROMs (Erasable Programmable ROM) verwendet. Diese können mit etwas Aufwand wieder gelöscht werden, wenn das Programm abgeändert werden muss.

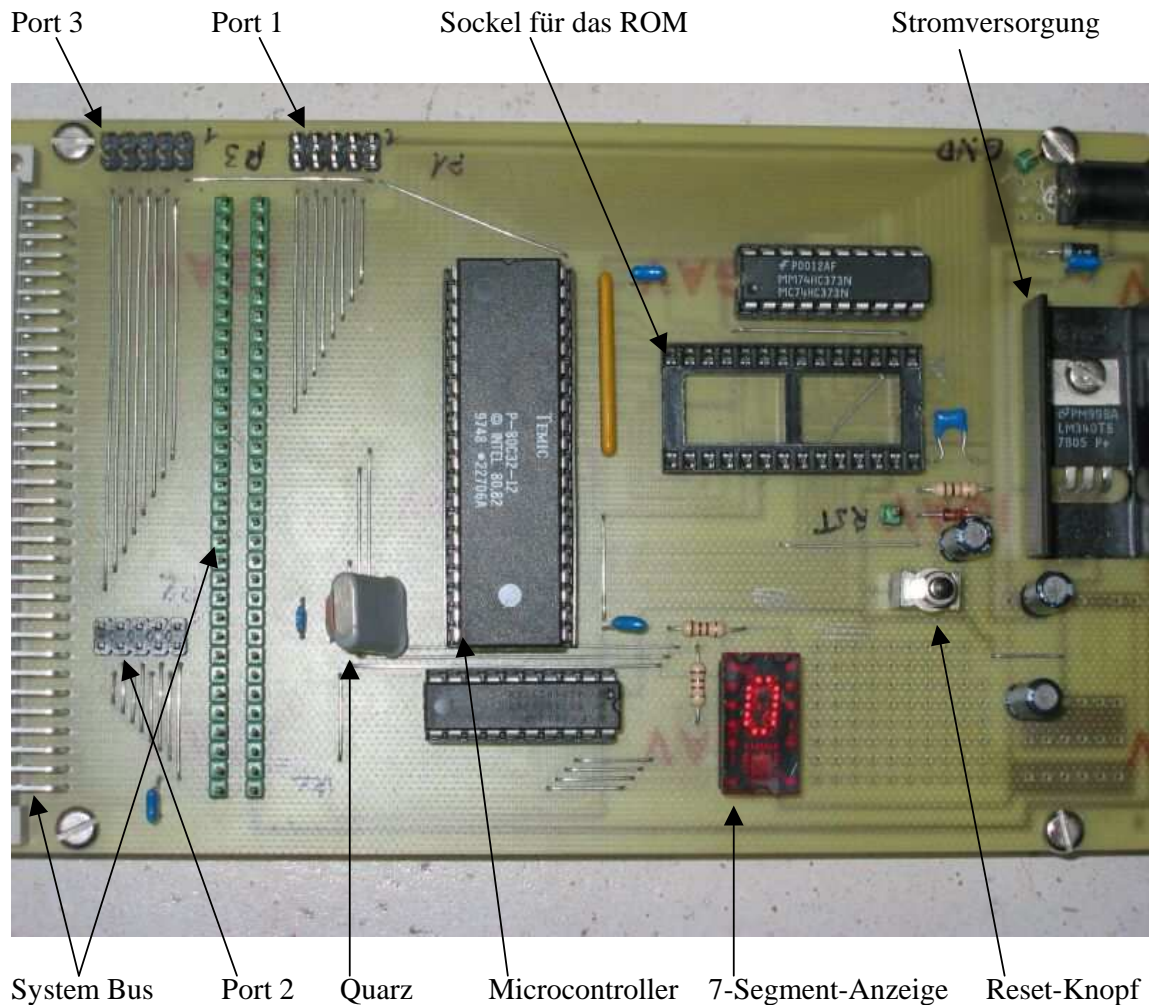
Wenn das Programm noch in der Entwicklungsphase ist oder es noch Fehler enthält, dann muss man für jeden Versuch das EPROM löschen und das Programm neu laden. Aber auch dafür gibt es eine Lösung: es gibt EPROM-Simulatoren. Diese Simulatoren werden wie ein ‚normales‘ EPROM auf das Microcontrollerboard aufgesteckt und sind durch ein Kabel mit der seriellen Schnittstelle eines PCs verbunden. Das Programm wird solange simuliert, bis es keine Fehler mehr enthält und wird erst danach auf ein definitives EPROM gebrannt.

- einem Taktgeber
- einem Prozessor (MPU)
- einer Stromversorgung (Netzgerät, Batterie etc.)

Normalerweise hat ein Microcontrollerboard zusätzlich noch einige Peripheriegeräte, wie:

- Ereigniszähler (Counter / Timer)
- weitere Datenübertragungsmöglichkeiten
- weitere Datenspeicherungsmöglichkeiten
- analog zu digital Übersetzer oder / und digital zu analog Übersetzer
- eine andere spezielle Ausgabe (zum Beispiel eine 7-Segment Anzeige) [4]

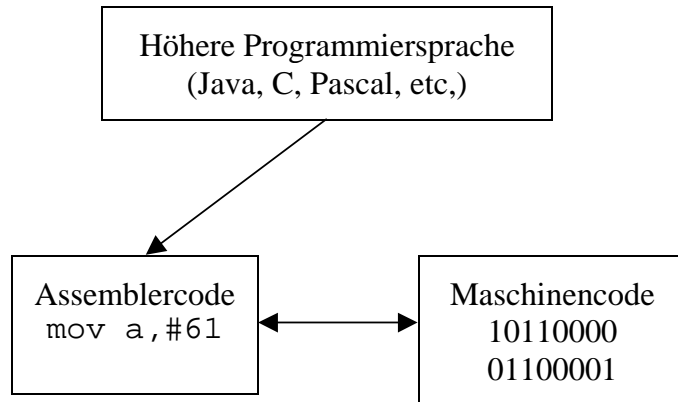
3.4 Schematische Darstellung des Microcontroller – Boards



3.5 Programmierung der Microcontroller

3.5.1 Assemblersprache

Der Microcontroller wird in einer sogenannten Assembler-Sprache programmiert. Eine Assembler-Sprache ist eine spezielle Programmiersprache, die den Maschinencode (welcher nur aus Nullen und Einsen besteht) für den Menschen verständlich macht, da jedes dieser Maschinencode-Zahlenmuster (z.B. 01110011 10100100) einem Assembler-Befehl entspricht. Die Programmierung wird dadurch viel einfacher, denn der Programmierer braucht nicht mehr den Maschinencode auswendig zu lernen, sondern kann einfach einen Assembler-Befehl eingeben. Schreibt man ein Programm, so wird jeder Befehl (der normalerweise aus drei Buchstaben besteht) vom Assembler zu Maschinencode umgewandelt (assembliert). Zum Beispiel entspricht der Befehl `mov a, #61` dem Maschinencode 10110000 01100001. Somit ist es auch möglich zu disassemblieren, das bedeutet, dass aus Maschinencode wieder Assemblercode wird. Maschinencode und Assemblercode sind folglich gleichwertig (siehe Grafik 1), also kann



Grafik 1: Unterschied höhere Programmiersprache / Assembler

Programmierersprache zu erhalten.

aus Assemblercode jederzeit einen Maschinencode erstellt werden und umgekehrt. Will man jedoch eine höhere Programmiersprache (Java, C, Pascal, etc.) in Maschinencode übersetzen, so wird zuerst anhand bestimmter Regeln ein Assemblercode erzeugt und erst dieser kann dann direkt in Maschinencode umgewandelt werden. Deshalb ist es (fast) nicht möglich aus Maschinencode wieder den ursprünglichen Befehl der höheren

3.5.2 Das binäre und hexadezimale Zahlensystem

Binäre Zahl	Dezimale (Hexadezimal) Zahl
11111	-1 (FFFFF)
00000	0
00001	1
00010	2
00011	3
00100	4
00101	5
00110	6
00111	7
01000	8
01001	9
01010	10 (A)
01011	11 (B)
01100	12 (C)
01101	13 (D)
01110	14 (E)
01111	15 (F)
10000	16 (10)
10001	17 (11)
Etc.	Etc.

Grafik 2: Binäres, dezimales und (hexadezimals) Zahlensystem

Das binäre Zahlensystem ist bei der Microcontroller-Programmierung sehr wichtig, da vielen Schnittstellen (zum Beispiel einem Port) eine Binärzahl übergeben werden muss. Ein Port besteht aus acht Pins und kann somit mit einer achtstelligen Binärzahl angesteuert werden, denn jede Ziffer dieser Zahl entspricht einem Pin am Port. 0 bedeutet ausgeschaltet und 1 bedeutet eingeschaltet. Die Zahl 00100100 bedeutet in dem Fall, dass der dritte und drittletzte Pin eingeschaltet ist.

Binär bedeutet, dass das ganze System nur aus zwei Ziffern besteht: 1 und 0. Die erste Zahl ist 0. Wird diese um eins erhöht, entsteht eine 1. Die nächste Zahl wäre eine 2, aber die gibt es in diesem System nicht, also wird eine neue Stelle eingeführt: 10 (sprich: „eins null“). Die nächste Zahl 3 wird zu 11 im Binärsystem usw. (siehe Grafik 2).

Will man zwei Binärzahlen addieren, so rechnet man wie bei der schriftlichen Addition (Siehe Grafik 3). Zuerst werden die zwei hintersten Zah-

$$\begin{array}{r}
 1. \quad 0111 \\
 + \quad 1 \\
 \hline
 \quad 0 \\
 \hline
 \\
 2. \quad 0111 \\
 + \quad 1 \\
 \hline
 \quad 00 \\
 \hline
 \\
 3. \quad 0111 \\
 + \quad 1 \\
 \hline
 \quad 1000 \\
 \hline
 \end{array}$$

Grafik 3: Schriftliche Addition

len addiert. $1 + 1$ ergibt 10, also 0 behalte 1. Dieses Behalte 1 wird im nächsten Schritt zu einer weiteren 1 dazugezählt und ergibt wieder 0 behalte 1. Im letzten Schritt wird das Behalte 1 zu einer 0 dazugezählt, somit erhält man dort eine 1. Das Resultat ist also 1000.

Minus Eins entspricht binär lauter Einsen. Wird zu all den Einsen noch 1 dazugezählt, so wird zuerst die hinterste Eins zu einer Null, behalte 1. Dieses Behalte 1 wird dann zur nächsten Ziffer addiert, was wieder zu einer Null behalte 1 führt. Somit entstehen unendlich viele Nullen. In Realität ist es so, dass bei minus Eins der ganz Speicher mit Einsen gefüllt wird. Wird dann dort noch 1 dazugezählt, so entstehen im ganzen Speicher Nullen und dann gäbe es noch eine 1 ausserhalb des Speichers, diese wird aber nicht mehr erfasst. Folglich bleiben nur noch die Nullen übrig.

Um eine binäre Zahl ins dezimales Zahlensystem umzurechnen, berechnet man die Summe aller Ziffern von rechts nach links gelesen, welche mit der entsprechenden Zweierpotenzen multipliziert werden. Ein Beispiel: 10011101 gibt also im Dezimalsystem:

$$(1*2^0=1)+(0*2^1=0)+(1*2^2=4)+(1*2^3=8)+(1*2^4=16)+(0*2^5=0)+(0*2^6=0)+(1*2^7=128)=157$$

Das Hexadezimale System ermöglicht es bis 15 zu zählen, ohne eine weitere Stelle einzuführen, indem die Zahlen 10, 11, 12, 13, 14 und 15 durch A, B, C, D, E und F ersetzt werden. Dies ist praktisch, weil die neuen Stellen bei 16, 256, usw. eingeführt werden. Wird ein Port mit vier Pins angesteuert, gibt es genau 16 verschiedene Kombinationen, wie die Pins gesetzt sind. Somit kann man diese vier Pins nur mit einer Hexadezimal-Stelle bedienen. Bei einem Acht-Pin-Port ist die höchstmögliche Binär-Zahl 1111 1111 (=255), dies entspricht gerade FF im Hexadezimalsystem. Ein solcher Port wird mit einer Zwei-Stellen-Hexadezimal-Zahl angesteuert.

3.5.3 Die wichtigsten Assembler-Befehle für meine Arbeit

Im Assemblercode sind sowohl Maschinenbefehle als auch Assemblerdirektiven enthalten. Assemblerdirektiven sind Befehle, die nicht in die Maschinensprache übersetzt werden, sondern dem Übersetzungsprogramm (Assembler) Hinweise geben. Diese Hinweise sind zum Beispiel, dass die Datei an dieser Stelle zu Ende ist oder das ;-Zeichen, welches bewirkt, dass die nachfolgenden Wörter in dieser Zeile vom Assembler nicht als Befehle sondern als Kommentare interpretiert werden.

```
mainStart:  mov     a, #1
```

mainStart ist eine Assemblerdirektive, die diesem Ort einen Namen gibt. Damit kann man dem Assembler immer die Anweisung geben zu mainStart zu springen.

mov schiebt den hinteren Parameter in den vorderen. In diesem Fall 1 in das Register a.

```
setb     p2.0
```

Gibt den Wert 1 an der Stelle, die im Parameter angegeben wird (in dem Fall am ersten Pin (Pin0) des zweiten Ports) aus.

```
clr     p1.6
```

Gibt den Wert 0 an der Stelle, die im Parameter angegeben wird (in dem Fall am fünften Pin (Pin6) des ersten Ports) aus

```
inc    p1
```

Erhöht den Wert an der Stelle, die im Parameter angegeben wird um eins (binär) (ist p1[Port1]=0 wird das letzte Pin aktiviert)

```
dec    p3
```

Vermindert den Wert an der Stelle, die im Parameter angegeben wird um eins (binär) (ist p1[Port1]=0 werden alle Pins aktiviert)

```
sjmp   mainStart
```

Lässt das Programm an den im Parameter angegebenen Ort springen (hier nach mainStart)

```
jb     p1.5, mainStart
```

Lässt das Programm an den im zweiten Parameter angegebenen Ort springen, falls der erste Parameter eins ist

```
jnb    p2.4, mainStart
```

Lässt das Programm an den im zweiten Parameter angegebenen Ort springen, falls der erste Parameter 0 ist

3.6 Anwendungen der Microcontroller

Da der Microcontroller speziell für ein System entwickelt wird, wird dieser sehr gut integriert und ist sehr stabil. Deshalb werden diese Microcontroller häufig in unbeaufsichtigten Maschinen verwendet, bei welchem eine hohe Zuverlässigkeit und die Fähigkeit, sich von einem Ausfall zu erholen, wichtig sind. [3]

Microcontroller finden wir zu Dutzenden in unserer Umgebung. Beispiele sind Radios, die Steuerung von CD-Playern, elektronische Armbanduhren, ABS-Bremsen, Waschmaschinen usw. Im Computer selber gibt es viele Microcontroller. Als Beispiel besitzt jede Tastatur einen MC8051 Microcontroller und der Schreib- und Lesevorgang der Festplatte wird von einem Microcontroller gesteuert. [1]

4. Mobiler Roboter mit Microcontroller gesteuert

4.1 Ziel

Mein Ziel war einen mobilen Roboter zu konstruieren, der ohne Verbindungskabel zum Computer läuft.

Um dies zustande zu bringen, musste ich zuerst einmal überlegen, was denn der Roboter genau machen sollte. Der Roboter soll gerade aus fahren können und auf ein eventuelles Hindernis reagieren und ausweichen. Die Hindernisse soll der Roboter mittels zweier Tastsensoren auf der Vorderseite wahrnehmen. Diese Tastsensoren müssen ein Signal an den Microcontroller schicken, welches dann bewirkt, dass der Roboter zurückfährt und dem Hindernis ausweicht.

Damit der Roboter dann auch wirklich mobil ist, muss das Programm später auf ein richtiges EPROM gebrannt werden (also nicht auf den Simulator). Weiter darf auch der Microcontroller nicht mehr mittels eines Netzgeräts den Strom beziehen, sondern soll an eine Batterie angeschlossen werden.

4.2 Materialien

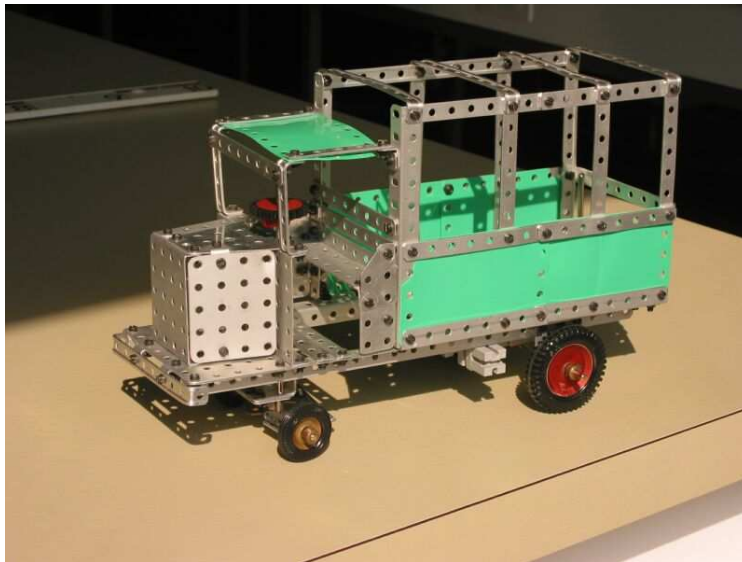


Bild 3: Grundgerüst

einen Motor zu betreiben. Also muss dieser Strom mittels einer Transistorschaltung verstärkt werden (Siehe Kapitel 4.3 Verstärkerschaltung). Weiter benutzte ich zwei 9 Volt Batterien um die Transistorschaltung und das Micocontroller-Board mit Strom zu versorgen. Ausserdem braucht es ein Grundgerüst (Bild 3), auf welchem der Microcontroller, die Schaltungen, die Motoren, die Batterien und die Sensoren untergebracht werden können. Dieses Grundgerüst baute ich mit Stokis. Zu Guter Letzt brauche ich noch Sensoren, die ich aus zwei Metallplatten anfertigte. Trifft der Roboter auf ein Hindernis, berührt die Metallplatte das Grundgerüst und leitet einen Strom, der dann vom Microcontroller gemessen wird.

Wie bereits erwähnt, ist die wichtigste Komponente der Arbeit der Microcontroller. Dazu kommen zwei Motoren: der Eine um den Roboter anzutreiben und um die Achsen zu bewegen, der Andere ermöglicht es dem Roboter eine Kurve zu fahren. Die Motoren und die Untersetzungen, die ich benutzte stammen von einem Fischer-Technik-Set. Ein Problem das sich stellt ist, dass der Strom, den der Microcontroller ausgibt nicht reicht, um

4.3 Verstärkerschaltung

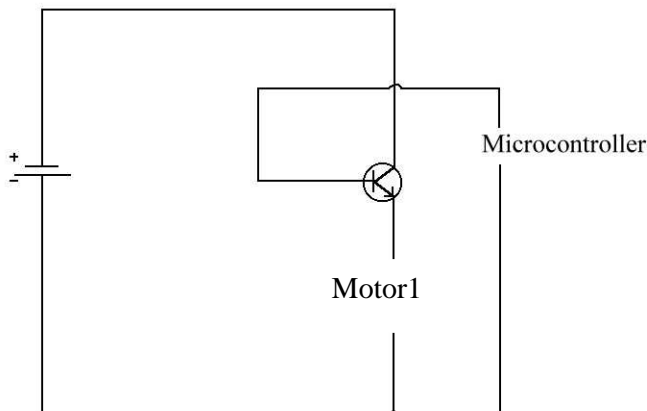


Bild 4: Transistorschaltung

Nachdem ich mich mit dem Microcontroller vertraut gemacht hatte, merkte ich bald, dass der Strom, den der Microcontroller lieferte, zu schwach war um einen Motor zu betreiben. Ich erinnerte mich an den Physikunterricht in der Tertia und entwarf eine Transistorschaltung (siehe Bild 4). Dabei dient der Motor zusätzlich als Widerstand, um dem Microcontroller nicht kurzzuschliessen.

Für jedes Pin am Port gibt es einen Transistor, damit jedes Pin einzeln verstärkt wird und somit jeder Motor einzeln angesteuert werden kann. Diese Schaltung wurde dann auf eine Platte gelötet und mit dem Microcontroller verbunden. Um die Schaltung mit dem Microcontroller richtig zu verkabeln, musste ich zuerst herausfinden welcher Pin welche Funktion an den einzelnen Ports hat. Ich fand heraus, dass der erste Pin immer 5 Volt lieferte, der zweite GND (also 0 Volt) war und der dritt bis zehnte dem px.0-px.7 beim Programmieren entsprach (wobei x durch die Portnummer ersetzt wird [also 1 bis 3]). Ich entwarf ein Schema, wie ich die und die Kabel von den Pins Transistoren anlöten sollte. Dabei war darauf zu achten, dass der zweite Pin (GND) mit dem Minus der Batterie verbunden sein musste.

Der erste Test der Schaltung war wenig erfolgreich. Laut dem Multimeter floss ein Strom, aber die Motoren drehten sich nicht. Das Problem war, dass die Transistoren zu schwach waren und somit zu wenig Strom floss, um die Motoren in Gang zu bringen. Nachdem ich jedoch neue Transistoren angelötet hatte und alles richtig angeschlossen war, konnte ich die beiden Motoren wie vorgesehen steuern. Ein (kleiner) erster Teil des Roboters war entstanden.

4.4 Grundgerüst

Als nächster Schritt baute ich das Fahrgestell. Dieses war mit Rädern versehen und so konzipiert, dass der Microcontoller, die Schaltungen, die Motoren, die Batterien und die Sensoren darin untergebracht werden konnten. Später fügte ich dem Grundgerüst noch einen Boden hinzu, weil die langen Kabel immer unter den Fahrzeug zum Vorschein kamen und sich in den Rädern verfangen.

Danach ging es zum ersten Mal an die Programmierung der Software. Als erstes kontrollierte ich die Schaltung, indem ich an jedem Pin des Ports p1 ein Bit ausgab. Nach einigen Korrekturen gelang es mir das Rad und das Steuer zum Drehen zu bringen. Damit der Roboter aber wirklich fahren konnte, musste er noch mehr untersetzt werden, da der Strom zu schwach war. Die Folge war, dass der Roboter nur langsam fahren konnte, dafür war es ihm möglich, alle Batterien und das ganze Microcontoller-Board mitzutragen.

Ein weiteres Problem, das sich noch ergab war, dass das Zahnrad auf der Achse drehen konnte, ohne dass die Kraft an die Achse und die Räder weitergegeben wurde. Das Zahnrad musste also besser befestigt werden. Anlöten war nicht möglich, da das Zahnrad aus Plastik bestand. Also versuchte ich das Zahnrad festzukleben. Ich testete verschiedene Klebstoffe, aber keiner haftete an der Metallachse. Schlussendlich nahm ich einen feinen Metalldraht und platzierte ihn zwischen der Achse und dem Zahnrad und klebte diesen mit Klebstreifen an der Achse fest. Nun war es nicht mehr möglich, dass das Zahnrad an der Achse vorbeidrehen konnte.

Als Letztes musste ich die Motoren besser befestigen, da sich die Fischer-Technik-Bausteine immer verschoben. Dieses Problem löste ich, indem ich kleine Balken auf der Seite der Bausteine befestigte.

4.5 Sensorschaltung

Jetzt fehlte aber noch etwas: die Sensoren. Das erste Problem war schon, dass ich nicht wusste, wie ich die Sensoren anzuschliessen hatte. Das einzige was ich wusste war, dass ich den Port 3 (p3) für die Motoren brauchte (bzw. schon für die Motoren verlötet hatte). Also blieb noch Port 1 (Port 2 lag zu weit weg, wäre aber natürlich auch noch da gewesen). Der erste Pin, an dem immer 5 Volt sind, wurde an einen Schutzwiderstand von $1k\Omega$ angeschlossen, damit nicht ein Kurzschluss entsteht wenn der Sensor geschlossen ist, (Siehe Bild 6). Von diesem Widerstand führte ein Kabel zum Sensor und von dort zu einem vordefinierten Pin am Port 3. Nun konnte ich ein Programm schreiben, welches „0“, „1“ oder „2“ am Display vom Microcontoller ausgab, je nach dem ob keiner, der linke oder der rechte Sensor aktiviert war.

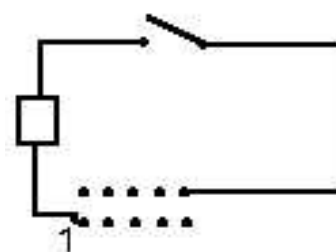


Bild 6: Sensorschaltung

Das Programm funktioniert so, dass wenn sich der Schalter schliesst ein Bit vom Pin 2 kommt (Pin 2 = „1“) und wenn sich der Schalter wieder öffnet null Bit am Pin2

kommt. Registriert der Microcontroller ein Bit am Pin 2, so springt das Programm an die Stelle rechts, wodurch eine Ausgabe am Display erfolgt.

(Das Programm finden Sie im Anhang unter 7.2 Sensorschaltung und hier nachfolgend einige Ausschnitte):

```
mov    a, p1
```

Kopiert die Informationen von Port 1 in den Akkumulator.

```
subb   a, #16
```

Der Wert wird um eine ganze Stelle vermindert (Hexadezimal gibt es 16 Zahlen pro Stelle, (siehe Kap. 3.5.2 Das binäre und hexadezimale Zahlensystem).

```
subb   a, #1
```

Hier wird der Akkumulator noch um eins vermindert, weil der Sensor nicht am ersten, sondern am zweiten Pin angeschlossen ist. Somit wird die Information, ob ein Strom fließt in das erste Bit des Akkumulators verschoben.

```
jz     rechts
```

Die Funktion `jz` springt an die Position „rechts“ im Programm, falls das erste Bit des Akkumulators 0 ist. Dies ist der Fall, wenn der rechte Schalter geschlossen ist und ein Strom fließt. An der Position „rechts“ wird dem Display der Wert „3“ übergeben und danach springt das Programm wieder an den Anfang.

```
subb   a, #1
```

Falls das Programm nicht an die Position „rechts“ gesprungen ist, wird der Akkumulator noch einmal um eins verringert, damit die Information des linken Sensors an die erste Position verschoben wird.

```
jz     links
```

Spring an die Position „links“, falls der linke Schalter geschlossen ist. Bei „links“ wird dem Display der Wert „2“ übergeben und danach springt das Programm wieder an den Anfang.

```
mov    p2, #0
```

Falls weder der linke noch der rechte Schalter geschlossen ist, wird dem Display der Wert „0“ übergeben und danach springt das Programm wieder an den Anfang (mit `sjmp Start`)

Nachdem das Programm endlich funktionierte, erstellte ich die Sensoren am Roboter. Dazu schloss ich das Kabel mit den 5 Volt (Siehe Bild 6) zuerst am Widerstand und dann am Grundgerüst an. Das ganze Grundgerüst besteht aus Eisen und leitet somit. Das Kabel, welches am Pin 2 angeschlossen wurde befestigte ich an einer Metallplatte. Diese Metallplatte war genau so angebracht, dass sie das Grundgerüst nur dann berührt, wenn der Roboter auf ein Hindernis trifft. Ich prüfte die Schaltung und sie funktionierte.

4.6 Relais-Schaltung

Ein weiteres Problem war, dass ich mit dem Roboter nicht rückwärts fahren konnte und das Steuerrad nur in eine Richtung bewegen konnte. Der Microcontoller sendete ein Signal, das verstärkt wurde. Dieses verstärkte Signal kann nur positiv sein, weil a) der Microcontoller nur positive Ströme abgeben kann und b) weil der Transistor nur positive Ströme verstärken kann. Das bedeutet, dass der Strom nur entweder in die eine Richtung fließen kann oder gar nicht. Eine Lösung wäre, ein zweiter Motor in entgegengesetzter Richtung aufzustellen, der dann das Rückwärtsfahren übernimmt. Das Problem war hierbei, dass der andere Motor dann in der umgekehrten Richtung drehen und die Achse blockieren würde. Diese Blockierung kommt dadurch zustande, dass der Motor mit einem Schnecken-Getriebe untersetzt wird, welches nicht in der entgegengesetzten Richtung laufen kann.

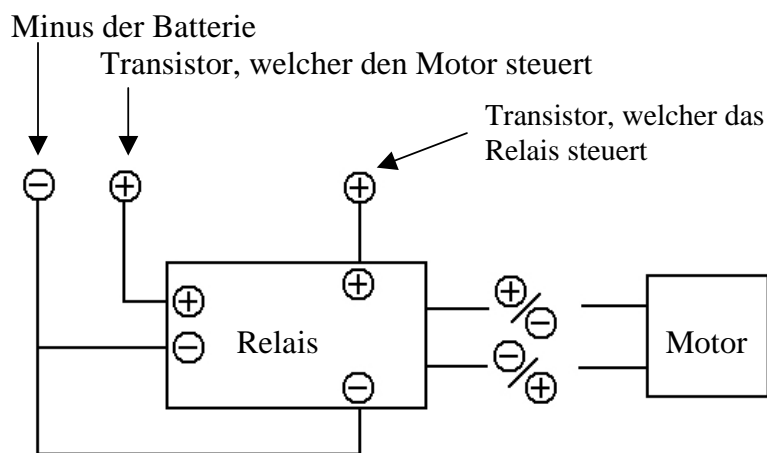


Bild 7: Anschluss des Relais

Ich kam auf die Idee, ein Relais einzusetzen, welches an einem weiteren Pin am Microcontoller angeschlossen wird (siehe Bild 7). Dieses Relais baute ich zwischen die Verstärkerschaltung und den Motor an. Ein Kabel, welches von der Verstärkerschaltung her kam, schloss ich oben an den weissen Knöpfen an (siehe Bild 8). Die Schwarzen Knöpfe wurden

nicht gebraucht. Die Halbkreise symbolisieren die Zusammenschaltung jeweils zweier Knöpfe. Der Motor wurde dann an den weissen Knöpfen unten angeschlossen. Position 1 ist die Ausgangslage. Wird ein Signal vom Microcontoller an das Relais gesandt, so werden die zwei Schalter (schwarze Linien) in Position 2 versetzt. Dadurch

wird unten die Polarität vertauscht. Dort wo der Motor angeschlossen ist, ist nun die genau umgekehrte Polarität gegenüber vorher. Bei jedem Signal an das Relais änderte sich dort die Polarität und somit auch die Richtung, in welcher der Motor dreht. Dadurch

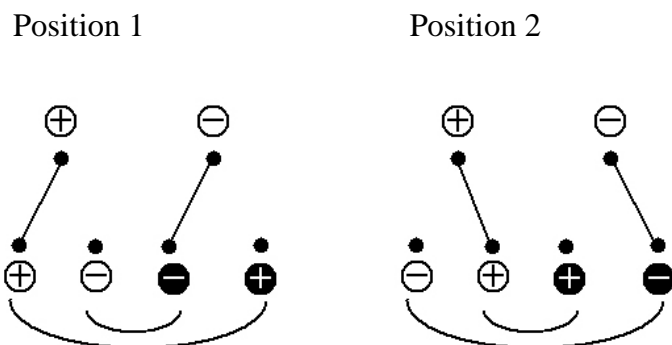


Bild 8: Prinzip eines Relais

ist es möglich, dass sich das Fahrzeug auch rückwärts bewegen kann und das Steuerrad in beide Richtungen gedreht werden kann.

4.7 Programmierung der Software

Der erste Teil des Programms war bereits vorhanden, da ich das Sensor-Testprogramm erweiterte (Siehe 4.5 Sensorschaltung und 8.1 Sensorschaltung). Das Prinzip bleibt das selbe: Sobald ein Sensor gedrückt wird, springt das Programm an eine bestimmte Stelle und führt dort einen Befehl aus. Bis jetzt bestand der Befehl lediglich daraus eine Zahl am Display auszugeben. Das Programm wird jetzt so verändert, dass der Roboter anhält, das Steuerrad in Richtung des Hindernisses dreht, rückwärts fährt, das Steuerrad vom Hindernis weg dreht und schliesslich abfährt. Danach springt das Programm wieder an den Anfang, wo sich der Roboter geradeaus bewegt.

Sie finden den vollständigen Quellcode mit genügend Kommentaren im Anhang unter „8.2 Quellcode des Programms zur Steuerung des Roboters“, deshalb werde ich an dieser Stelle nicht auf die einzelnen Befehle eingehen.

Das Programm funktionierte (erstaunlicherweise) gleich beim ersten Mal. Nur das Zeitintervall musste noch einige Male anders eingestellt werden, denn der Roboter fuhr anfangs zu wenig weit vom Hindernis weg. Dies deshalb, weil der Roboter langsam fährt und somit auch mehr Zeit braucht.

Nachdem das erste provisorische Programm entstanden war, versuchte ich den Roboter einmal ohne Verbindungskabel zum PC laufen zu lassen. Dafür musste ich alle Geräte (Relais, Batterien, Transistorschaltung und vor allem den Microcontoller mit dem EPROM-Simulator) in das Grundgerüst hineinpacken. Dies war sehr schwierig, da sich überall freie Drähte befanden und diese sich nicht berühren durften. Eine zusätzliche Schwierigkeit bestand darin, dass das ganze Grundgerüst aufgrund der Sensorschaltung am 5 Volt Pin des Microcontollers angeschlossen war. Somit durfte auch nichts das Grundgerüst berühren. Dies war vor allem bei der Transistorschaltung schwierig, da diese auf allen Seiten irgendwelche Kontakte hatte. Nach einiger Zeit gelang es mir, dass sich der Roboter frei bewegte. Alle Schaltungen funktionierten, das Programm funktionierte und der Roboter lief eine Weile. Leider war dann aber auch schon die Batterie am Ende, weil ich diese sehr viel gebraucht hatte, als ich die Transistorschaltung und die Motoren testete. Weitere Probleme ergaben sich an den Lötstellen, da sich Kabel vom Relais oder aus der Transistorschaltung lösten.

Das letzte Problem war, dass nachdem der Roboter das Steuerrad eine bestimmte Zeit lang in eine Richtung gedreht hatte und es dann später gleich lang wider zurückbewegte, war das Steuerrad nicht mehr ganz genau in der Mitte. Folglich fuhr der Roboter nicht mehr gerade aus. Sobald also der Roboter einmal das Steuer bewegt hatte, konnte er nie mehr gerade aus fahren. Das Problem könnte gelöst werden, indem der Roboter einen weiteren Tastsensor in der Mitte des Steuerrades erhalten würde. So könnte programmiert werden, dass das Rad solange gedreht wird, bis es den Sensor wieder berührt und wieder in der Position wäre um gerade aus fahren zu können.

5. Schlussfolgerungen

Der Entwurf und vor allem die Ausführung eines Roboters, auch wenn er im Vergleich nur so simpel war wie meiner, ist äussert zeitaufwändig. Das Löten, welches eigentlich das Einfachste gewesen wäre, nahm am meisten Zeit in Anspruch. Das Problem ist, dass es sehr schwierig ist die richtige Dosis Lötzinn zu verwenden. Benutzt man zu viel, dann überläuft der Lötzinn Teile der Schaltung und wenn man zu wenig braucht, dann hält das Kabel oder der Draht nicht richtig und reisst bei der kleinsten Belastung ab. Weiter war es problematisch, da ich am Anfang noch andere Transistoren gebrauchte die dann ersetzt werden mussten. Dadurch musste ich noch einmal von vorne mit Löten beginnen.

Die Programmierung war bis zu dem Zeitpunkt schwierig, bevor ich herausfand, wie die Sensorschaltung abgefragt werden kann. Zeitweise war es ziemlich entmutigend, da ich wirklich alles Mögliche versucht hatte und es trotzdem nicht funktionierte.

Ich habe allerdings viel gelernt, vor allem über den Microcontroller und dessen Programmierung und im Grossen und Ganzen kann ich sagen, dass die Arbeit sehr interessant, aber zeitaufwändig war.

6. Zusammenfassung

Die Arbeit befasst sich in einem ersten Teil intensiv mit Microcontoller. Microcontoller sind Chips, in welchen viele Komponenten (zum Beispiel Prozessor, RAM u.s.w.) untergebracht werden. Somit ist er sehr kompakt und ideal geeignet für selbständige Systeme, wie zum Beispiel Mobiltelefone, Radios, elektronische Armbanduhren etc. Microcontoller haben aber selbst kein Betriebssystem, was bedeutet, dass diese in der Maschinensprache programmiert werden müssen. Diese Maschinensprache heisst Assembler. Im Rahmen meiner Arbeit wird ein kleiner Einblick in diese Sprache geboten.

In einem zweiten Teil wird meine praktische Arbeit beschrieben. Mein Ziel war es einen Roboter zu bauen, welcher ohne Verbindung zu einem Computer fahren kann und mit zwei Tastsensoren ausgerüstet ist. Trifft der Roboter auf ein Hindernis so soll er dieses möglichst gut umfahren. Der Roboter funktioniert folgendermassen: Der Microcontoller überprüft den Tastsensor um festzustellen, ob der Roboter an ein Hindernis gestossen ist. Danach gibt der Microcontoller ein Signal aus, das von einem Transistor verstärkt wird und so zum Motor gelangt, der den Roboter steuert.

7. Quellenverzeichnis

- [1] http://www.goblack.de/desy/mc8051chip/theorie/stw_mikrocontroller.html
- [2] www.blumen-debus.de/markus-debus/projects/prozessoren.pdf
- [3] <http://www.chipcenter.com/eexpert/dgilbert/dgilbert004.html>
- [4] Ordner von Herrn Plüss im Besitz des Deutschen Gymnasiums Biel

Eine Kopie der Quellen aus dem Internet finden Sie unter 8.3 Quellen aus dem Internet

Die folgenden Seiten finden Sie im Internet unter:

<http://maturaarbeit.boegli.tk>

8. Anhang

8.1 Sensorschaltung

8.2 Quellcode des Programms zur Steuerung des Roboters

8.3 Quellen aus dem Internet