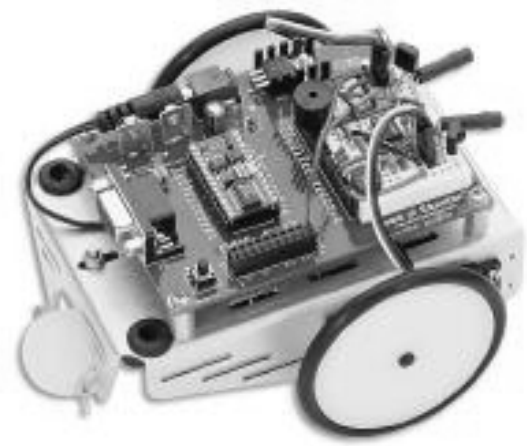
UNIVERSITÄT SIEGEN

Fakultät IV - Department Maschinenbau
Mess- und Regelungstechnik – Mechatronik
Prof. Dr.-Ing. Oliver Nelles

Maschinenlabor

Einführung in die Mikrocontroller-Programmierung am Beispiel eines autonomen Fahrzeugs



Versuch 10

Ort:	PB-H0109 (ZESS, im Untergeschoss)
Betreuer:	Dr.-Ing. Geritt Kampmann PB-A250 Tel.: 0271/740-2492 Dipl.-Ing. Julian Belz PB-A250 Tel.: 0271/740-4664

Inhaltsverzeichnis

1 Motivation und Einführung

Ziel dieses Laborversuchs ist es, einen kleinen mobilen Roboter (Bild siehe Deckblatt) so zu programmieren, dass er einfache Navigationsaufgaben durchführen kann. So soll er sich zum Beispiel frei durch einen Raum bewegen können und Hindernissen ausweichen. In einem weiteren Versuch soll der Roboter einer Bodenmarkierung (schwarze Linie) folgen.

Solche Navigationsaufgaben müssen in der Industrie beispielsweise von autonomen Transportsystemen erfüllt werden, die Produkte zwischen einzelnen Fertigungsstationen befördern. Aber auch in der Automobilindustrie gibt es Bemühungen, PKW das autonome Fahren beizubringen, um den Fahrer zu entlasten, bzw. um zusätzliche Sicherheitsreserven in Notsituationen zu schaffen.

Ein autonomes Fahrzeug benötigt, neben einem regelbaren Antrieb, geeignete Sensoren, die die Umwelt erfassen können und eine Steuereinheit, die die Signale der Sensoren in eine entsprechende Bewegung des Fahrzeugs umsetzt. Im Laufe des Versuches werden Sie zwei einfache Sensoren und eine einfach zu programmierende Steuereinheit kennen lernen.

Bei dieser Steuereinheit handelt es sich um einen sogenannten **Mikrocontroller**. Diese Geräte sind in unzähligen Produkten des täglichen Lebens unverzichtbar geworden. Das Einsatzgebiet reicht von Kinderspielzeugen über Haushaltsgeräte und Geräte der Unterhaltungsindustrie, bis hin zu Fahrzeugen und Produkten der Luft- und Raumfahrt.

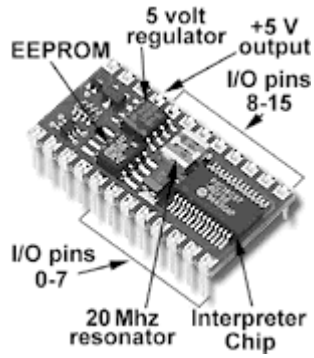
Mikrocontroller sind kompakte Computer, die zur Steuerung und Regelung von Maschinen aller Art verwendet werden. Da die Anwendungsgebiete sehr unterschiedlich sind, gibt es eine große Zahl von verschiedenen Microcontollern, die sich insbesondere in ihrer Leistungsfähigkeit und somit im Preis unterscheiden. Viele Merkmale, die ein Mikrocontroller auf-

weist, finden sich auch im PC wieder. Beide benötigen z.B. einen Prozessor und einen Arbeitsspeicher. In vielen Punkten gibt es jedoch Unterschiede, so benötigt ein Mikrocontroller keinen Massenspeicher, wie z.B. eine Festplatte, und keine Grafikhardware. Außerdem werden in der Regel andere Prozessoren als im PC-Bereich verwendet, da eine geringere Anzahl an Prozessorbefehlen benötigt wird, dafür stehen aber Schnelligkeit, Zuverlässigkeit und Robustheit, sowie eventuell geringer Energieverbrauch im Vordergrund.

Im Folgenden wird zunächst der verwendete Mikrocontroller vorgestellt und dessen Aufbau und Funktionsweise kurz erklärt. Danach wird die Zusatz-Hardware erläutert, die nötig ist, um Sensoren und Antriebe an den Controller anzuschließen. Schließlich erfolgt eine Einführung in die Programmierung des Controllers. In dem anschließenden Kapitel werden die 4 Versuche vorgestellt. In den ersten 2 Versuchen erfahren Sie an Hand einfacher Beispiele, wie die Ein- und Ausgänge mit elektronischen Bauelementen beschaltet werden und wie man einfache Programme für den Mikrocontroller schreibt. Die abschließenden Versuche beschäftigen sich dann mit der Programmierung des mobilen Roboters. Empfohlene Literatur zum Versuch finden Sie im letzten Kapitel.

2 Der Mikrocontroller *BASIC Stamp II*

In diesem Laborversuch wird ein Mikrocontroller (im Folgenden µC abgekürzt) verwendet, der sehr einfach aufgebaut ist und speziell für die Ausbildung entwickelt wurde. Dieser Mikrocontroller von der Firma *PARALLAX* heißt *BASIC Stamp II*, weil er so klein wie eine Briefmarke (Stamp) ist und in einer BASIC-ähnlichen Sprache programmiert wird. **Bild 1** zeigt den Mikrocontroller *BASIC Stamp II* und einige technische Daten.



- 16 Input/Output Leitungen
- Serielle Schnittstelle bis 50kBaud
- 2048 Byte EEPROM als BASIC Programmspeicher
- Kapazität: ca. 500 BASIC Befehle
- 16 Worte RAM für Variablen (32 Byte)
- Geschwindigkeit: ca. 4000 Befehle/s
- PIC16C57 mit 20 MHz Taktfrequenz (Schwingquarz)
- Reset Controller überwacht die Betriebsspannung
- Spannung: 5,5 bis 15V über On-Board Spannungsregler oder direkt 5V
- Stromaufnahme: 7mA (Run), 50µA (Sleep)
- Größe: 24-pol. Dual Inline Package
- Programmierung beliebig oft über die serielle Schnittstelle möglich
- Direktanschluß von Tastern, LEDs, Potentiometern, Signalgebern...

Bild 1: *BASIC Stamp II*, Ansicht und technische Daten

2.1 Aufbau und Funktion des Mikrocontrollers

Wie **Bild 1** zeigt, besteht der µC aus einigen verschiedenen Komponenten, die auf einer Platine mit 24 Anschlussbeinchen (Pins) untergebracht sind. Diese Form der Platine bezeichnet man auch als 24 Pin DIP (Dual In-Line Package), weil sie zwei, jeweils in einer Linie angeordnete, Reihen von Pins aufweist. Die Komponenten sind in **Bild 2** noch einmal schematisch dargestellt.

Die zentrale Komponente ist der *Interpreter Chip* (U1), an den die meisten Komponenten angeschlossen sind und in dem das Basic-Programm abläuft. Er beinhaltet auch die zentrale Steuereinheit des Mikrocontrollers, den Prozessor.

Die mit P0 bis P15 beschrifteten Pins dienen dem µC als *Ein-* und *Ausgänge*, deren Zustand programmgesteuert entweder eingelesen oder gesetzt werden kann. Der µC kennt an den Ein-/Ausgängen nur die binären Zustände high oder low (1 oder 0), das entspricht einer anliegen-

den Spannung von 5V oder 0V. Genau genommen unterscheidet der μC beim Eingang nur zwischen größer (high) und kleiner (low) als 1,4V (Schwellenspannung). Um eine unnötige Belastung des μC durch Verlustströme zu vermeiden, sollten die Eingangsspannungen möglichst wenig von 0 bzw. 5V abweichen. In diesen Fällen nimmt ein Eingang des μC weniger als 1μA auf und beeinflusst den angeschlossenen Stromkreis kaum.

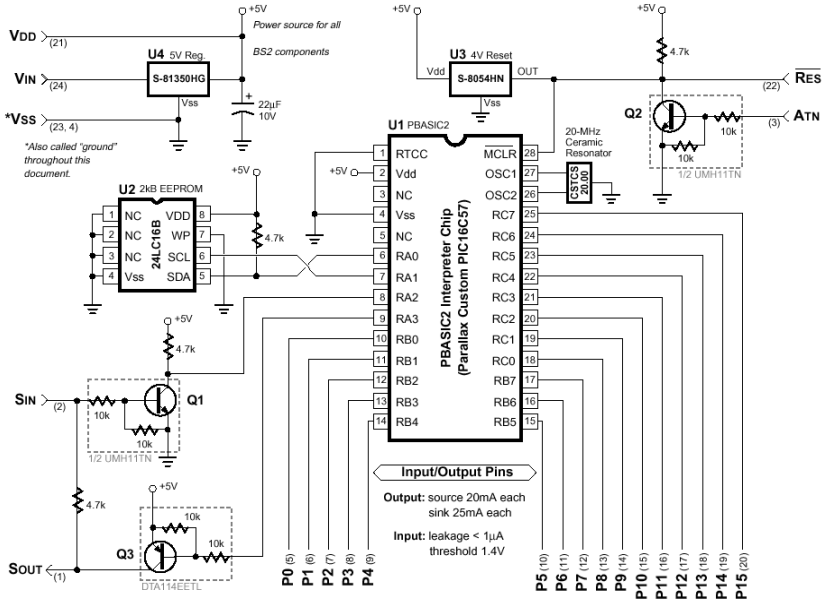


Bild 2: Aufbau der *BASIC Stamp II*

Der Basic Interpreter Chip verfügt über einen eingebauten *Arbeitsspeicher* (RAM, Random Access Memory) für Programmvariablen. Diese Variablen können während des Programmablaufs durch das Programm selbst geändert werden, z.B. durch mathematische Operationen. Die Größe dieses Speichers beträgt 32 Bytes, wovon nur 26 Bytes für das Programm zur Verfügung stehen, denn 6 Bytes werden für die Ansteuerung der Ein-/Ausgänge benötigt.

Der μC benötigt für den störungsfreien Betrieb eine Spannungsversorgung von 5V. Dies wird durch den *Spannungsregler* (U4) sichergestellt. An den Eingang des Spannungsreglers wird die Spannung V_{IN} angeschlossen, die 5,5 bis 15V betragen darf und beispielsweise von einer Batterie oder einem Netzteil geliefert wird. Der Regler stabilisiert diese Spannung, unabhängig von der Belastung des μC auf die Spannung V_{DD}, die 5V beträgt.

Sollte V_{IN} weniger als 5,5V betragen, zum Beispiel durch die Verwendung verbrauchter Batterien, so kann die Spannung V_{DD}=5V nicht aufrechterhalten werden und es kommt zu Fehlern bei der Programmausführung. Aus diesem Grund ist ein *Brown-Out Detektor* (U3) auf dem μC vorhanden. Dieses Bauteil führt einen Reset des Basic Interpreters durch, wenn V_{DD} zu niedrig ist. D.h. das laufende Programm wird beendet und neu gestartet. Dieser Vorgang wiederholt sich und es kommt zu keiner Programmausführung mehr. Der μC stellt auch einen Pin zur Verfügung mit dessen Hilfe ein Reset manuell, z.B. über einen Taster, ausgelöst werden kann.

Mit V_{SS} wird die Bezugsmasse der Schaltung (auch Erde, Ground) bezeichnet. Alle Spannungsangaben werden bezüglich dieser Masse gemessen und angegeben. Mit V_{SS} beschriftete Kontakte sind also direkt mit dem Minuspol der Stromversorgung verbunden.

Das mit U2 gekennzeichnete Bauteil ist der *Programmspeicher*. Dort werden die Befehle der selbstgeschriebenen Basicprogramme abgelegt und bei Bedarf vom Interpreter Chip aus-

gelesen. Bei diesem Bauteil handelt es ich um ein EEPROM (Electrical Eraseable and Programmable Read Only Memory) mit 2 kB (2048 Byte) Speicherkapazität. Dieser Speicher kann nahezu beliebig oft gelöscht und neu beschrieben werden (ca. 10 Millionen Zyklen). Gespeicherte Programme bleiben bis zur nächsten Löschung erhalten, auch wenn das EEPROM nicht mit Strom versorgt wird. Dieser Speicher ist nicht mit dem Arbeitsspeicher des Interpreters (RAM) zu verwechseln, der nur 32 Byte groß ist. Während auf das RAM durch das Basicprogramm selbst zugegriffen werden kann, ist das EEPROM dazu gedacht, den während des Programmablaufs unveränderlichen Programmcode aufzunehmen. Der Zugriff auf den EEPROM-Speicher geschieht wesentlich langsamer als ein Zugriff auf das RAM.

Wie gelangen nun die selbstgeschriebenen Programme in das EEPROM? Die Programme werden mit einem speziellen *Editor* an einem PC geschrieben. Durch eine serielle Kabelverbindung kann ein fertiggestelltes Programm über den Interpreter Chip in das EEPROM heruntergeladen werden. Der µC benötigt dazu drei Pins (S_{IN}, S_{OUT}, ATN), die über die Bauelemente Q1, Q2 und Q3 angeschlossen sind. Diese Transistorschaltungen sind nötig, weil die serielle Schnittstelle des PC (RS-232) die logischen Zustände 0 und 1 (low und high) durch die Spannungen +12V, und –12V darstellt (inverse Logik), der µC jedoch, wie zuvor erwähnt, durch 0 und 5V.

Die letzte Komponente des µC ist der 20 MHz Schwingquarz, der an die Anschlüsse 26 und 27 des Interpreter Chips angeschlossen ist. Er liefert den nötigen Takt, um die Programmbefehle schrittweise abzuarbeiten bzw. Zeiten zu messen.

2.2 Verbindung mit der Außenwelt durch das Board of Education (BOE)

Die einfachste Methode, den µC an eine Stromversorgung anzuschließen und eine Verbindung zwischen den 16 Ein-/Ausgängen des µC und anderen elektronischen Komponenten herzustellen, ist das in der **Bild 3** dargestellte Board of Education (BOE).

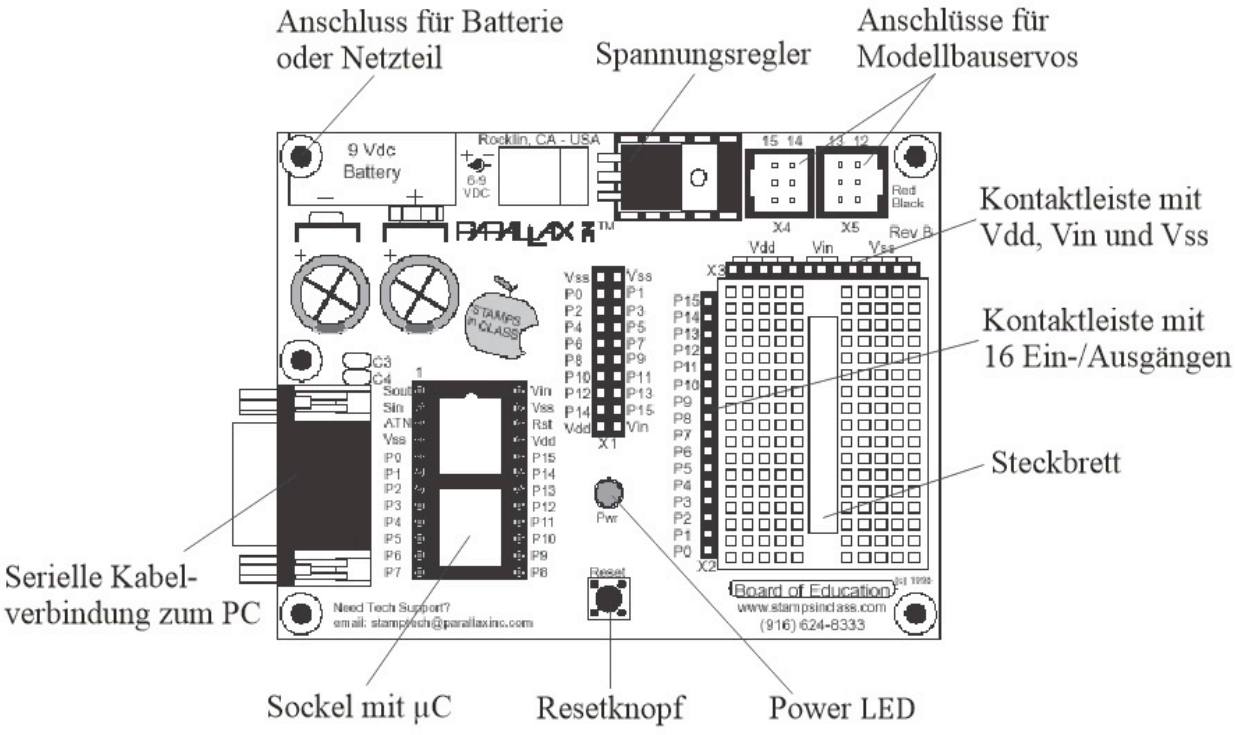


Bild 3: Board of Education (BOE)

Das BOE stellt ein Steckbrett (rechts) für eigene elektronische Schaltungen zur Verfügung. Das Brett hat 170 Steckkontakte, wobei allerdings jeweils 5 nebeneinander liegende Kontakte eine Gruppe bilden und miteinander verbunden sind. Es ergeben sich also insgesamt 34 Kontaktgruppen, die nicht untereinander verbunden sind.

Das BOE bietet einen Anschluss für eine 9V Batterie oder ein 6-9V Netzteil, sowie einen eigenen Spannungsregler, der für eine stabile Spannungsversorgung mit 5V sorgt. An der linken Seite ist die Anschlussbuchse für die serielle Kabelverbindung zum PC zu erkennen. Direkt rechts davon befindet sich der Sockel, in den der μC hineingesteckt wird.

Nach dem Einstecken ist der μC mit der Stromversorgung, der seriellen Schnittstelle und einem Resetknopf verbunden. Auch die 16 Ein-/Ausgänge P0 bis P15 des μC können nun auf dem BOE an verschiedenen Stellen abgegriffen werden. Am häufigsten wird dafür die mit X2 beschriftete schmale Kontaktleiste benötigt, die sich direkt links neben dem Steckbrett des BOE befindet. Direkt über dem Steckbrett können an der Kontaktleiste X3 Verbindungen zur geregelten 5V Stromversorgung V_{DD} (5 Kontakte links), zur ungeregelten Stromversorgung V_{IN} (3 Kontakte Mitte, nicht für Schaltungen verwenden) und zur Masse V_{SS} (5 Kontakte rechts) hergestellt werden.

Weiterhin von Bedeutung sind die Anschlüsse X4 und X5, die es ermöglichen, jeweils 2 Modellbauservos (kleine Stellmotoren, die sich meist max. 45° nach links und rechts drehen können) über jeweils 2 Steckverbindungen an den μC anzuschließen. Die Servos benötigen zum Betrieb jeweils 3 Kontakte. Die oberen Kontakte in den Anschlüssen X4 und X5 stellen das Steuersignal zur Vorgabe des Drehwinkels zur Verfügung und sind mit den Pins 12, 13, 14 oder 15 des μC verbunden, je nachdem welche der 4 Anschlussmöglichkeiten gewählt wurde. Die mittleren Kontakte versorgen die Servos mit Strom und sind mit V_{IN} verbunden, die unteren Kontakte stellen einen Anschluss zur Masse V_{SS} zur Verfügung.

2.3 Programmierung des Mikrocontrollers

Die Programmierung des μC erfolgt mit Hilfe eines speziellen Editors unter Windows, der auch eine Überprüfung des geschriebenen Programms auf Fehler und das Senden des fertigen Programms an den μC ermöglicht. Weiterhin stellt der Editor ein Debug-Fenster zur Verfügung auf dem beliebige Daten dargestellt werden können, die der μC an den PC zurücksendet, solange eine serielle Kabelverbindung zum PC besteht. Auch eine grafische Darstellung des EEPROM- und RAM-Speichers, z.B. zur Beurteilung der freien Ressourcen, ist möglich.

2.3.1 Variablen

Der μC kann nur Integeroperationen durchführen, d.h. es sind nur Operationen mit ganzen Zahlen möglich, die zudem maximal 2 Byte groß sein dürfen. In den Variablen können also auch nur ganze Zahlen mit maximal 2 Byte (= 1 Word) Größe abgespeichert werden (0 bis 65535, bzw. +/-32767). Am Anfang eines Programmes müssen zunächst alle Variablen, die im Programm verwendet werden, definiert werden. Dies geschieht durch folgende Syntax:

Variablenname `var` *Typ*

An dem Schlüsselwort `var` erkennt der Basic Interpreter, das eine Variable mit der Bezeichnung *Variablenname* und der durch *Typ* bestimmten Größe definiert werden soll. Es gibt 4 verschiedene Variablentypen, die in **Tabelle 1** zusammengefasst sind.

Sie unterscheiden sich in der maximalen Größe der Zahlen, die in ihnen abgespeichert werden können (Wertebereich) und somit auch im Speicherplatz, den sie im RAM benötigen. Da im RAM nur 26 Bytes (oder $26 \cdot 8 = 208$ Bit) zur Verfügung stehen, sollte der Variablentyp angepasst an die Größe des zu speichernden Zahlenwertes gewählt werden, um Platz zu sparen.

Soll z.B. in der Variable `meinevariable` eine Zahl gespeichert werden, die maximal den Wert 10 annimmt, so reicht es aus ein Nibble im Speicher zu reservieren:

```
meinevariable       var  nib
```

Typ	Wertebereich	benötigter RAM-Speicher
`bit`	0 oder 1	1 Bit
`nib`	0 bis 15	4 Bit = 1 Nibble
`byte`	0 bis 255	8 Bit = 1 Byte
`word`	0 bis 65535	16 Bit = 2 Byte = 1 Word
max. Speicherplatz (RAM)		208 Bit = 26 Byte

Tabelle 1: Variablentypen

2.3.2 Zahlenoperationen

Mit den Zahlenwerten der Variablen können natürlich auch mathematische Operationen durchgeführt werden. Die wichtigsten, die in diesem Versuch benötigt werden, zeigt nachfolgende **Tabelle 2**. Wie zu erwarten, gibt es die vier Grundrechenarten, abgekürzt durch die üblichen Symbole (+, -, *, /). Bei der Division ist zu beachten, dass das Ergebnis immer ganzzahlig ist. Es findet keine Rundung statt, Nachkommastellen werden abgeschnitten!

Symbol	Bedeutung
`+`	Addition
`-`	Subtraktion
`*`	Multiplikation
`/`	Division (liefert nur den ganzzahligen Teil, z.B. `5/2` ergibt 2)
`=`	Wertzuweisung an eine Variable (`zahl=2+4`, speichert 6 in der Variable `zahl`)
`abs`	Absolutwert einer Zahl (z.B. `abs(-5)` ergibt 5)

Tabelle 2: Zahlenoperationen

Das Gleichheitszeichen in dieser Tabelle ist nicht als mathematische Operation aufzufassen, sondern stellt die Zuweisung eines Zahlenwertes zu einer Variable dar. Das gleiche Gleichheitszeichen kann in anderem Kontext aber auch als (logischer) Vergleichsoperator verwendet werden (siehe **Tabelle 4** im Abschnitt 2.3.4). Weiterhin ist noch der Befehl `abs` von Interesse, der den Absolutwert (Betrag) einer Zahl liefert.

2.3.3 Ein- und Ausgänge

Es existiert eine Gruppe von Variablen, die nicht definiert werden müssen, um sie benutzen zu können. Es handelt sich um die Variablen zur Ansteuerung und Abfrage der Ein- und Ausgänge. Die an den 16 Eingängen anliegende Spannung wird von dem µC in den Variablen `in0` bis `in15` abgelegt. Hat die Variable `in5` z.B. den Wert 1, so liegen an Eingang 5 (Pin 5, P5) 5V an. Falls 0V anliegen hat `in5` den Wert 0. Entsprechend bestimmen die Variablen `out0` bis `out15` welche Spannungen vom µC ausgegeben werden. Wenn am Ausgang 5 (Pin 5, P5) 5V anliegen sollen, müssen Sie `out5` den Wert 1 zuweisen, anderenfalls 0.

Um zu bestimmen, ob ein Pin ein Eingang oder ein Ausgang ist, werden die vordefinierten Variablen `dir0` bis `dir15` verwendet. Z.B. ist Pin 5 ein Eingang, wenn die Variable `dir5` den Wert 0 hat. Hat sie den Wert 1, so ist er ein Ausgang. Setzt man z.B. `dir5` auf 1, so werden in Abhängigkeit des in `out5` gespeicherten Wertes 0 oder 5V an Pin 5 gelegt. Beim Programmstart setzt der µC zunächst die Variablen `dir0` bis `dir15` auf 0. D.h. alle Pins sind Eingänge und die anliegende Spannung kann anhand der Variablen `in0` bis `in15` abgelesen

werden. Die `dir`-Variablen müssen in der Regel nicht direkt gesetzt werden, sondern es gibt geeignete Basic-Befehle, die dies übernehmen. Diese Befehle und weitere zur Ansteuerung von Ein- und Ausgängen entnehmen **Tabelle 3**.

Befehl	**Bedeutung**
`input` *pin*	Macht den angegebenen Pin (`0-15`) zum Eingang, `dir`-Variable wird auf 0 gesetzt.
`output` *pin*	Macht den angegebenen Pin (`0-15`) zum Ausgang, `dir`-Variable wird auf 1 gesetzt.
`high` *pin*	Macht den angegebenen Pin (`0-15`) zum Ausgang und setzt ihn auf high (5V), `out`-und `dir`-Variablen werden auf 1 gesetzt
`low` *pin*	Macht den angegebenen Pin (`0-15`) zum Ausgang und setzt ihn auf low (0V), `out`-Variable wird auf 0 und `dir`-Variable wird auf 1 gesetzt.
`pulsout` *pin*, *time*	Macht den angegebenen Pin (`0-15`) zum Ausgang und gibt dort einen Puls der Länge *time* (`0-65535`, in 2µs Schritten) aus (d.h. der Status des Ausgangs wird invertiert und nach der angegebenen Zeit wieder zurückgesetzt).
`freqout` *pin*, *time*, *frq*	Macht den angegebenen Pin (`0-15`) zum Ausgang und gibt dort eine Schwingung der Dauer *time* (in ms) und der Frequenz *frq* (in Hz) aus.
`rctime` *pin*, *state*, *result*	Macht den angegebenen Pin (`0-15`) zum Eingang und misst die Zeit, die vergeht, bis der Status des Pins von 0 auf 1 (*state*=0) oder von 1 auf 0 (*state*=1) wechselt. Das Ergebnis wird in der Variable *result* (0-65535, in 2µs Schritten) abgespeichert.
`debug` *home*, "Text", *ft val*	Anzeige der Variablen *val* im Debug-Fenster: *home* bringt den Cursor in die obere linke Bildschirmecke. "Text" kann optional eingefügt werden, um beliebige Textinformationen vor der Variable anzuzeigen. *ft* gibt an, wie die Darstellung der Variablen formatiert wird (`dec` Dezimalzahl, `bin` Binärzahl), die Stellenzahl kann vorgegeben werden (z.B. `dec5:` Ausgabe als Dezimalzahl mit 5 Stellen, ggf. mit voranstehenden Nullen). Weitere Variablen können mit Komma getrennt angefügt werden: `debug home,"Links ",bin1 varL," Rechts ",bin1 varR`
`debug` *cls*	Das Debug-Fenster wird gelöscht

Tabelle 3: Ansteuerung von Ein-/Ausgängen

Eine besondere Ausgabeanweisung stellt der Befehl `debug` dar. Mit diesem Befehl können beliebige Programmvariablen über die serielle Verbindung an den PC übermittelt werden. Sie werden im Debug-Fenster des Programm-Editors dargestellt. Dies ist nicht nur sinnvoll um Programmfehler zu finden (debugging), sondern ermöglicht z.B. auch die Visualisierung von Messdaten, die mit Hilfe des µC ermittelt wurden.

2.3.4 Programmablaufsteuerung

Normalerweise wird ein Programm schrittweise vom Anfang bis zum Ende abgearbeitet. Es ist jedoch meist notwendig, von dieser Reihenfolge abzuweichen. Dafür stehen u.a. die in **Tabelle 4** angegebenen Befehle zur Verfügung.

Mit den Befehlen `goto` *marke* und der Abfrage `if ... then` *marke* kann zu beliebigen Stellen im Programm gesprungen werden, die durch den Begriff *marke* definiert werden. Diese

Sprungmarke kann eine beliebige Bezeichnung haben, die jedoch nicht bereits als Variablenname verwendet wird und endet immer mit einem Doppelpunkt.

Befehl	**Bedeutung**
`pause` *time*	Die Programmausführung wird für den Zeitraum *time* (0-65535, in Millisekunden) angehalten.
`end`	Beendet ein Programm, μC wird in Energiesparmodus versetzt.
`stop`	Beendet ein Programm.
`for` *var*=*start* `to` *end* (`step` *stepval*) ⋮ `next`	Wiederholt die Befehle zwischen der for- und der next-Anweisung. Die Variable *var* wird, beginnend bei dem Wert *start*, bei jedem Durchlauf um 1 (oder den Wert *stepval*) erhöht. Überschreitet *var* den Wert *end*, wird die Schleife abgebrochen.
Marke:	Ein mit einem Doppelpunkt endender Bezeichner stellt eine Marke dar, die von einem der folgenden Befehle angesprungen werden kann.
`goto` *marke*	Springt zur Sprungmarke *marke* und setzt die Programmausführung dort fort.
`return`	Führt zu einem Rücksprung der Programmausführung zu der Zeile bei der die letzte gosub-Anweisung erfolgte. Das Programm wird in der Zeile nach dem gosub-Befehl fortgesetzt. Sinnvoll für Unterprogrammaufrufe.
`if` ... `then` *marke*	Ist die zwischen if und then eingeschlossene logische Beziehung wahr, springt das Programm zur Sprungmarke *marke*, anderenfalls wird das Programm normal fortgesetzt. Zulässige Vergleichsoperatoren sind: <, >, =, <=, >=. Vergleiche können mit AND, OR, bzw. mit NOT negiert werden.

Tabelle 4: Befehle zur Programmablaufsteuerung

3 Versuche

Nachfolgend werden die 3 Versuche vorgestellt, die im Rahmen dieses Labors durchgeführt werden sollen. Im ersten Versuch werden zunächst einfache binäre Ein- und Ausgangssignale behandelt, die mit Tastern bzw. Leuchtdioden realisiert werden. Versuch 2 zeigt, wie ein analoges Eingangssignal (kontinuierlich veränderlicher Widerstand), im μC verarbeitet werden kann. Der dritte Versuch beschäftigt sich schließlich mit der Ansteuerung der Motoren der Fahrzeuge, dem Auslesen der Sensoren und der Erstellung einer Fahrstrategie basierend auf den Sensordaten.

3.1 Versuch 1: Einschalten einer Leuchtdiode mit einem Taster

In diesem Versuch wird zur Einführung gezeigt, wie man einen Ausgang in Abhängigkeit eines Eingangs setzt. Dazu wird Pin 0 des μC als Eingang definiert und ein handbetätigter Taster (Schließer) angeschlossen. Mit dem Ausgang (Pin 1) wird eine Leuchtdiode (LED) verbunden. Wie wird nun der Taster an den Pin 0 (P0) angeschlossen? Die einfachste Möglichkeit besteht darin, P0 auf dem Steckbrett direkt über den Taster mit V_{DD} zu verbinden (**Bild 4 A**). Diese Methode ist nicht empfehlenswert, da die Eingänge des μC einen hohen Innenwiderstand haben und somit vorhandene Ladungen an den Eingängen trotz geöffneten Tasters nicht abfließen. D.h. auch bei offenem Taster ermittelt der μC unter Umständen mehr

als 1,4V am Eingang, also einen high-Zustand. Eine Verbindung zur Masse V_{SS} über einen hochohmigen Widerstand (z.B. 10kΩ, Farbcodierung: braun, schwarz, orange) sorgt für das Abfließen von diesen Ladungen, bei gleichzeitig niedrigen Verlustströmen. Diese Schaltung ist im **Bild 4 B** dargestellt. Drücken des Tasters führt dazu, dass 5V an P0 anliegen (high ≙ 1), beim Loslassen des Tasters liegen 0V an (low ≙ 0). Sollen beim Drücken des Tasters 0V anliegen und im offenen Zustand 5V, ist Schaltung C (**Bild 4 C**) zu verwenden.

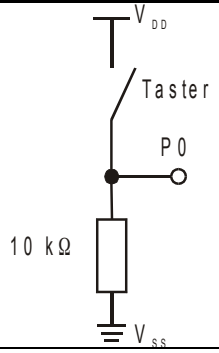
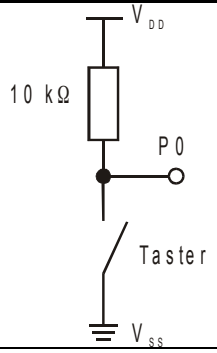
Schaltung A	**Schaltung B**	**Schaltung C**
 FALSCH!!!		

Bild 4: Beschaltungsmöglichkeiten des Eingangs P0

Als Nächstes muss nun die LED an P1 angeschlossen werden. Da die Ausgänge des μC maximal mit etwa 20mA belastet werden dürfen, ist auch hier eine Widerstandsschaltung nötig. Eine LED hat einen Widerstand, der sich in Abhängigkeit der angelegten Spannung ändert, im **Bild 5** ist dieser Zusammenhang durch eine Diodenkennlinie dargestellt.

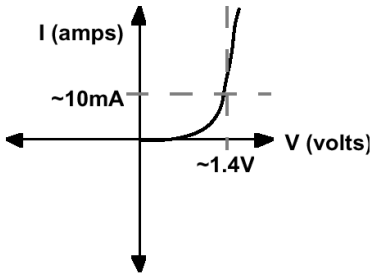
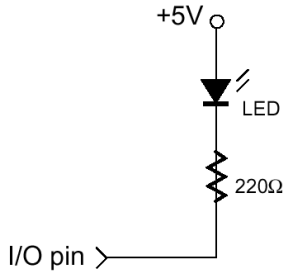
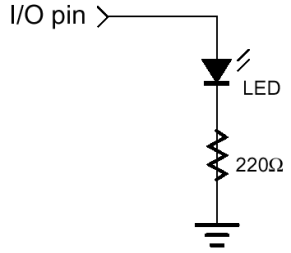
Diodenkennlinie	**Schaltung A**	**Schaltung B**
		

Bild 5: Beschaltung des Ausgangs P1

Die Kennlinie zeigt, dass bei niedrigen Spannungen (U) wenig Strom (I) durch die LED fließt (bei negativen überhaupt keiner). Mit zunehmender Spannung nimmt der Widerstand der LED stark ab, so dass der Strom exponentiell ansteigt. Damit die LED ordnungsgemäß funktioniert benötigt sie etwa 10mA Strom, wobei etwa 1,4V über ihr abfallen (siehe **Bild 5**). Da der μC am Ausgang aber 5V zur Verfügung stellt, wären die zulässigen 20mA stark überschritten und der μC könnte beschädigt werden. Deshalb ist es nötig, einen Widerstand R in Reihe zur LED zu schalten, dessen Größe sich wie folgt berechnet:

$$5V = U_R + U_{LED} = U_R + 1{,}4V \Leftrightarrow U_R = 3{,}6V \text{ und } R = \frac{U_R}{I} = \frac{3{,}6V}{10mA} = \underline{360\Omega}.$$

Bei Verwendung dieses Widerstandes beträgt der Strom an Pin1 ca. 10mA, aber auch bei der Verwendung eines 220Ω (Farbcode: rot, rot, braun) oder eines 470Ω (Farbcode: gelb, violett, braun) Widerstandes funktioniert die Schaltung problemlos. Da LEDs nur in einer Stromrichtung leitend sind, müssen sie an die korrekten Pole angeschlossen werden. Die abgeflachte Seite des LED-Gehäuses bzw. der kürzere Pin kennzeichnen diejenige Seite, die an den Mi-

nuspol (VSS) angeschlossen werden muss. Es gibt nun zwei Möglichkeiten die LED an den µC anzuschließen, diese sind im **Bild 5** dargestellt. Die Schaltungen unterscheiden sich darin, dass bei Schaltung A die LED leuchtet, wenn der Ausgang des µC auf 0V gesetzt ist, während die LED in Schaltung B leuchtet, wenn der Ausgang auf 5V gesetzt ist. In den anderen Fällen fließt kein Strom durch die LED, weil auf beiden Seiten der LED die gleiche Spannung (5V in Schaltung A bzw. 0V in B) anliegt.

```
'Maschinenlabor Mobiler Roboter: Taster


'----- Deklarationen --------------

taster var bit                'Definition einer Bit-Variablen taster


'----- Initialisierung ------------

debug cls                     'Debug-Bildschirm löschen
input 0                       'Pin 0 als Eingang definieren
output 1                      'Pin 1 als Ausgang definieren
taster=0                      'Variable taster Null setzen


'----- Hauptprogramm --------------

main:                         'Sprungadresse des Hauptprogramms

taster=in0                    'Eingang abfragen und in taster speichern

if taster=1 then einschalten         'Sprung in die Unterprogramme
if taster=0 then ausschalten         'um LED Ein-/Auszuschalten

zurueck:                             'Rücksprung aus Unterprogrammen

debug home, "Knopf= ", bin1 out1     'Ausgang darstellen in Debug-Fenster.
                                     'Home bringt den Cursor in die obere linke
                                     'Bildschirmecke
goto main                            'Hauptprogramm Schleife


'-------- Unterprogramme --------------

einschalten:                  'LED ein
 out1=1                       'oder high 1 verwenden
 goto zurueck

ausschalten:                  'LED aus
 out1=0                       'oder low 1 verwenden
 goto zurueck
```

Programmlisting 1: Taster

Aufgaben:

1. Verbinden Sie nun auf dem Steckbrett den Taster mit P0 und die LED mit P1, verwenden Sie dabei jeweils die Schaltungen B.
2. Betrachten Sie das oben aufgeführte **Programmlisting 1**, es ist etwas aufwändiger konzipiert, als für die einfache Anwendung nötig, zeigt dafür aber alle wichtigen Konzepte wie Sprungmarken und Unterprogramme. Was bewirkt das Programm?
3. Erweitern Sie das Programm in der Weise, dass der Taster eine Haltefunktion erhält. D.h. beim ersten Tastendruck wird die LED eingeschaltet, beim zweiten ausgeschaltet, usw.. Welches Problem tritt dabei auf?

3.2 Versuch 2: Blinker

Im vorherigen Versuch wurde ein Taster dazu verwendet, ein Eingangssignal zu erzeugen. Dieses Signal war lediglich binär, es gab nur die Zustände 0 und 1 (0V und 5V). Es soll nun gezeigt werden, dass auch analoge Signale (z.B. ein veränderlicher Widerstand) als Eingangssignale verwendet werden können. Um dies zu ermöglichen, benötigt man eine Reihenschaltung aus einem Kondensator *C* und einem veränderlichen Widerstand *R*, ein sogenanntes RC-Glied, wie es **Bild 6 (links)** zeigt. Der veränderliche Widerstand kann z.B. ein Potentiometer oder ein Photowiderstand sein. Zwischen *R* und *C* wird ein Ein-/Ausgang des μC geschaltet.

Um die Größe des verstellbaren Widerstandes *R* zu ermitteln, wird der Kondensator *C* über den Widerstand *R* mit einer Spannung von 5V aufgeladen. Während des Ladens steigt die über dem Kondensator *C* abfallende Spannung kontinuierlich von 0V (Kondensator leer) auf 5V (Kondensator voll) an, während gleichzeitig die Spannung über dem Widerstand *R* von 5V auf 0V sinkt. Die Summe beider Spannungen ergibt zu jedem Zeitpunkt 5V (Maschenregel). Zum besseren Verständnis der Schaltung sollten Sie sich klarmachen, dass ein leerer Kondensator keinen Widerstand für den Strom darstellt und somit die 5V vollständig über dem Widerstand anliegen. Ein voller Kondensator hat hingegen einen unendlich großen Widerstand, das heißt, es fließt kein Strom mehr durch die Schaltung, auch nicht durch den Widerstand *R*. Nach dem ohmschen Gesetz kann dann an *R* keine Spannung mehr anliegen.

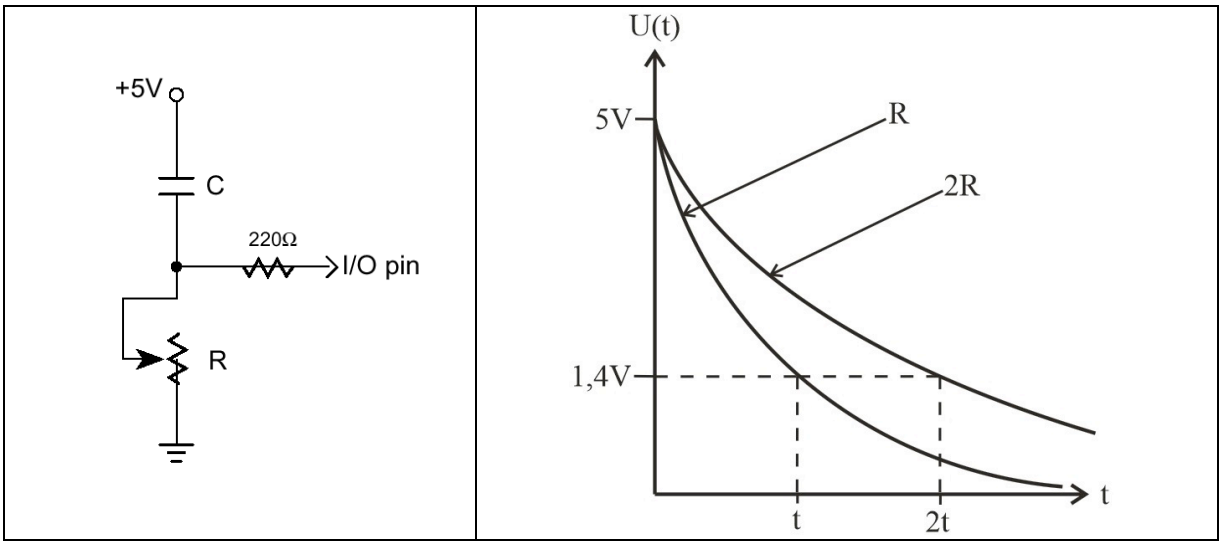


Bild 6: RC-Glied (links), Zusammenhang zwischen Widerstand R und Entladezeit t_e (rechts)

Allgemein, bzw. für unsere spezielle Anwendung, berechnet sich die Zeit, die vergeht, bis eine am Widerstand *R* anliegende Spannung U_1 während des Ladevorgangs auf eine Spannung U_2 gesunken ist, nach folgenden Gleichungen:

$$t = R \cdot C \cdot \ln\left(\frac{U_1}{U_2}\right) \Rightarrow t = 1{,}27 \cdot 10^{-8} \frac{s}{\Omega} \cdot R$$

Dies ergibt sich mit folgenden Zahlenwerten:

- $C = 0.01\mu F = 10^{-8} F$ (verwendeter Kondensator)
- $U_1 = 5V$, weil zu Beginn der Kondensator leer ist und somit die gesamte Spannung über R abfällt.
- $U_2 = 1{,}4V$, da bei dieser Schwellenspannung der Eingang des μC von 1 (high) auf 0 (low) abfällt.

Wie obige Gleichung zeigt, besteht ein linearer Zusammenhang zwischen dem Widerstands *R* und der Aufladezeit *t*. Je größer der Widerstand *R* gewählt/eingestellt wird, desto länger benötigt der Kondensator *C* um sich auf 5V aufzuladen, analog dazu vergeht auch mehr Zeit bis der Eingang des µC von 1 auf 0 abfällt (**Bild 6 rechts**). Um diese Zeit *t* zu messen, wird der im Abschnitt 2.3.3, **Tabelle 3** beschriebene Befehl *rctime* benutzt.

Dabei ist folgendermaßen vorzugehen. Wenn Sie z.B. den Pin P6 für die Schaltung verwenden, dann setzen Sie P6 zunächst auf high (`high 6`) und warten etwa 3ms (`pause 3`). Dadurch liegen an beiden Anschlüssen des Kondensators 5V an und der Kondensator muss sich auf Grund der fehlenden Spannungsdifferenz über den Widerstand entladen. Der Zeitraum von 3ms reicht für den Entladungsvorgang völlig aus. Durch den Befehl

```
rctime 6,1,zeit
```

wird P6 nun zum Eingang und beeinflusst somit die Schaltung nicht mehr. Der Kondensator lädt sich auf. An P6 liegen zu Beginn 5V an (high). Sobald die Spannung an P6 auf 1,4V gefallen ist, wechselt der Zustand des Eingangs P6 auf low und die bis zu dem Wechsel vergangene Zeit wird durch den Befehl `rctime` in der Variable `zeit` abgespeichert (`zeit` bitte als `word` definieren). Diese Befehle werden nun innerhalb einer Programmschleife ständig wiederholt, so dass kontinuierlich eine neue Messung des Widerstands *R* vorliegt.

Aufgaben:

1. Erstellen Sie obiges RC-Glied mit einem 0,01µF Kondensator, einem 10kΩ Potentiometer und einem 220Ω Widerstand als Schutz für den µC. Verwenden Sie z.B. den Anschluss P6.
2. Schreiben Sie ein Programm, das mit Hilfe des Befehls `rctime 6,1,zeit` die Aufladezeit bestimmt und im Debug-Fenster anzeigt.
3. Schließen Sie eine LED an P0 an, die in Abhängigkeit der Potentiometereinstellung mit unterschiedlicher Frequenz blinken soll (z.B. den Befehl `pause 10*zeit` verwenden).

3.3 Vorbemerkungen zu Versuch 3 und 4: Der BOE-Bot

Die letzen beiden Versuche werden mit dem BOE-Bot durchgeführt. Hierbei handelt es sich um den auf dem Deckblatt dieser Versuchbeschreibung abgebildeten mobilen Kleinroboter. Der Antrieb des BOE-Bot erfolgt über zwei leicht modifizierte Modellbauservos, die direkt mit zwei großen Rädern an den Seiten des BOE-Bot verbunden sind. Eine Kugel am Heck des Roboters dient als Stützrad. Auf dem Roboter ist ein BOE befestigt, mit dessen Hilfe die Ansteuerung der Servos erfolgt.

Für die restlichen Versuche sind alle elektronischen Schaltungen auf dem BOE bereits vorbereitet, es müssen lediglich noch die notwendigen Programme geschrieben werden.

Bevor es weitergehen kann, muss die Frage geklärt werden, wie die Antriebe angesteuert werden können. Modellbauservos werden über Pulsfolgen reguliert, die an die Steuerleitung des Servos übergeben werden (Pulsweitenmodulation, PWM). Zu diesem Zweck ist der rechte Antrieb des Roboters an P14 und der linke an P15 an des µC angeschlossen (Stecker X4 auf dem BOE). Der Servo erwartet etwa alle 20ms (Zeiten zwischen 10 und 40ms sind auch noch akzeptabel) einen 5V-SpannungsPuls von 1,3 bis 1,7ms Dauer (**Bild 7**).

Normalerweise rotiert ein Servo bei einer Pulslänge von 1,3ms mit maximaler Geschwindigkeit um 45° nach rechts, bei einer Pulslänge von 1,7ms um 45° nach links. Bei einer Pulslänge von 1,5ms bleibt er in der Mittellage. Bei dazwischen liegenden Pulslängen ergibt sich eine entsprechend proportionale Geschwindigkeit und Drehung. Durch Deaktivierung des

eingebauten Winkelmessers, drehen die Servos nicht mehr um nur 45°, sondern kontinuierlich wie ein Motor. Die Pulslänge bestimmt jetzt nur noch die Geschwindigkeit der Drehung.

Pulse lassen sich einfach mit dem Befehl `pulsout`, der im Abschnitt 2.3.3, Tabelle 3 beschrieben ist, erzeugen. Die Länge des Pulses wird als zweites Argument in 2µs an den Befehl übergeben. Die Zeiten 1,3ms, 1,5ms und 1,7ms entsprechen also den Zahlenwerten 650, 750 und 850. **Tabelle 5** zeigt die Befehle, die nötig sind, um die Basisbewegungen zu erzeugen.

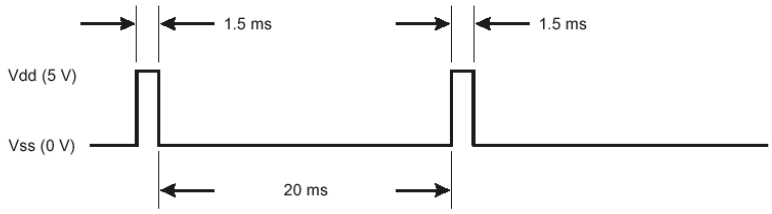


Bild 7: Pulsfolge zur Servosteuerung

Bewegung	**rechter Servo**	**linker Servo**
Stillstand	`pulsout 14,750`	`pulsout 15,750`
vorwärts fahren (max. Geschwindigkeit)	`pulsout 14,650`	`pulsout 15,850`
rückwärts fahren (max. Geschwindigkeit)	`pulsout 14,850`	`pulsout 15,650`
Rechtsdrehung auf der Stelle (max. Geschw.)	`pulsout 14,850`	`pulsout 15,850`
Linksdrehung auf der Stelle (max. Geschw.)	`pulsout 14,650`	`pulsout 15,650`

Tabelle 5: Bewegungen des BOE-Bot

```
'Maschinenlabor Mobiler Roboter: Fahren

'----------- Deklarationen -----------------------------

zaehler var   word           'Zählervariable

'------------ Initialisierungen --------------------------

low 14                       'Ausgänge definieren und
low 15                       'auf 0V setzen

'------------ Hauptprogramm ------------------------------

for zaehler=1 to 100         'Schleife wird 100 mal durchlaufen
  goto geradeaus             'Sprung zum Unterprogramm geradeaus
next

end                          'Programmende

'----- Unterprogramme - Fahren ------------------------

geradeaus:
 pulsout 14,650
 pulsout 15,850
 pause 20                    'Vor dem nächsten Puls 20ms warten
 return                      'Rücksprung zur Zeile unterhalb von goto geradeaus
```

Programmlisting 2: Fahren

Um den BOE-Bot etwa 2 Sekunden vorwärts fahren zu lassen, ist zum Beispiel das im **Programmlisting 2** dargestellte Programm nötig. Durch dieses Programm wird 100 mal das Unterprogramm *geradeaus* aufgerufen. Dieses Unterprogramm sendet die entsprechenden Pulse an die beiden Servos, um eine Vorwärtsbewegung zu ermöglichen (Tabelle 5) und sorgt für eine Pause von mindestens 20 ms bevor neue Pulse gesendet werden können.

3.4 Versuch 3: Navigation im Raum

Für diesen Versuch wird der Roboter mit je zwei Infrarot-LEDs (IR-LEDs) und Infrarot-Detektoren ausgestattet, um Hindernisse auf seinem Weg erkennen zu können. **Bild 8** zeigt die dazu nötige Schaltung (Anmerkung: In dem Bild ist zusätzlich ein Piezo-Lautsprecher dargestellt, mit dem Tonsignale ausgegeben werden können, z.B. bei Programmstart). Der Roboter soll so programmiert werden, dass er geradeaus durch einen Raum fährt und Hindernissen ausweicht, indem er sich von ihnen wegdreht.

Die IR-LEDs werden über P1 (rechte LED) und P7 (linke LED) an den µC angeschlossen, die Detektoren, die das von Hindernissen reflektierte Licht erfassen, über P0 (rechts) und P8 (links). Damit die Messung nicht durch Infrarotlicht aus anderen Quellen beeinträchtigt wird, sind die Detektoren nicht nur auf die Frequenz des Lichtes, das die LEDs aussenden, durch einen optischen Filter abgestimmt, sondern sie reagieren darüber hinaus nur auf Lichtquellen, die mit einer Frequenz von 38,5 kHz blinken.

Um z.B. die linke LED mit dieser Frequenz für eine Millisekunde blinken zu lassen, kann der Befehl `freqout 7,1,38500` (siehe auch Abschnitt 2.3.3, Tabelle 3) verwendet werden. Da die Detektoren etwas träge reagieren, kann nun im nächsten Programmschritt der Status des Einganges P8 abgefragt werden, obwohl die LED dann nicht mehr leuchtet. Empfängt der Detektor Licht, das von einem Hindernis auf der linken Seite des BOE-Bot reflektiert wurde, so fällt P8 auf 0V (`in8` = 0) ab. Anderenfalls liegen 5V an (`in8` = 1). Die Detektoren verhalten sich also im Prinzip wie „lichtbetätigte“ Taster nach **Bild 4** Schaltung C.

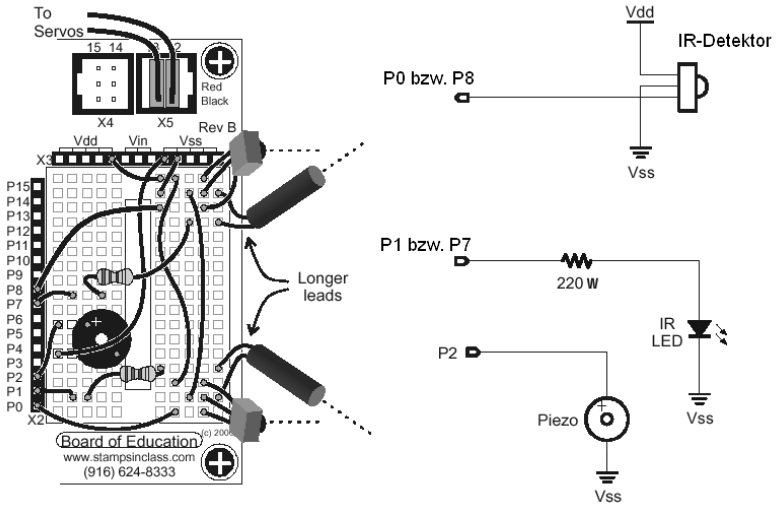


Bild 8: Schaltung für IR-Navigation.

Aufgaben:

1. Versuchen Sie die Funktionsweise der Schaltung zu verstehen.
2. Schreiben Sie ein Programm, das die LEDs blinken lässt, die Detektoren überprüft und das Ergebnis im Debug-Fenster darstellt. Prüfen Sie die Funktion der Hinderniserkennung.
3. Ergänzen Sie das Programm mit Fahrbefehlen (links, rechts, geradeaus) und Regeln, wann der Roboter links, rechts oder geradeaus fahren muss, um Hindernissen auszuweichen. Diese Vorgehensweise führt auf einen Dreipunktregler.

3.5 Versuch 4: Linienverfolgung

In diesem Versuch soll der BOE-Bot eine auf den Boden gezeichnete schwarze Linie verfolgen. Zu diesem Zweck sind an dem Roboter zwei Photowiderstände angebracht, deren Wi-

derstand sich mit größerem Lichteinfall vermindert. Mit zwei RC-Gliedern, wie sie bereits im Versuch 2 verwendet wurden, kann die Widerstandsänderung erfasst und somit eine Aussage über die Lage der schwarzen Linie gemacht werden. Den Aufbau der Schaltung zeigt **Bild 9**.

Aufgaben:

1. Versuchen Sie die Funktionsweise der Schaltung zu verstehen.
2. Schreiben Sie ein Programm, das mit Hilfe des Befehls `rctime` die Ladezeiten der Kondensatoren ermittelt. Geben Sie die Ergebnisse im Debug-Fenster aus. Testen Sie das Programm, indem Sie ein Blatt Papier mit einer schwarzen Linie unter den Photowiderständen hin und her schieben.
3. Ergänzen Sie das Programm mit Fahrbefehlen (links, rechts, geradeaus) und Regeln, wann der Roboter links, rechts oder geradeaus fahren muss, um der Linie zu folgen (linker Widerstand größer als rechter oder umgekehrt, bzw. beide gleich). Diese Vorgehensweise führt auf einen Dreipunktregler
4. Da die zwei Photowiderstände auch bei gleichem Lichteinfall niemals exakt den gleichen Widerstand aufweisen, wird der Roboter nie geradeaus fahren. Führen Sie also eine Tote Zone ein, in der der Roboter geradeaus fährt. Eventuell müssen Sie die Breite der Toten Zone und die Wendegeschwindigkeit des Roboters variieren, bis Sie ein zufriedenstellendes Ergebnis erhalten.

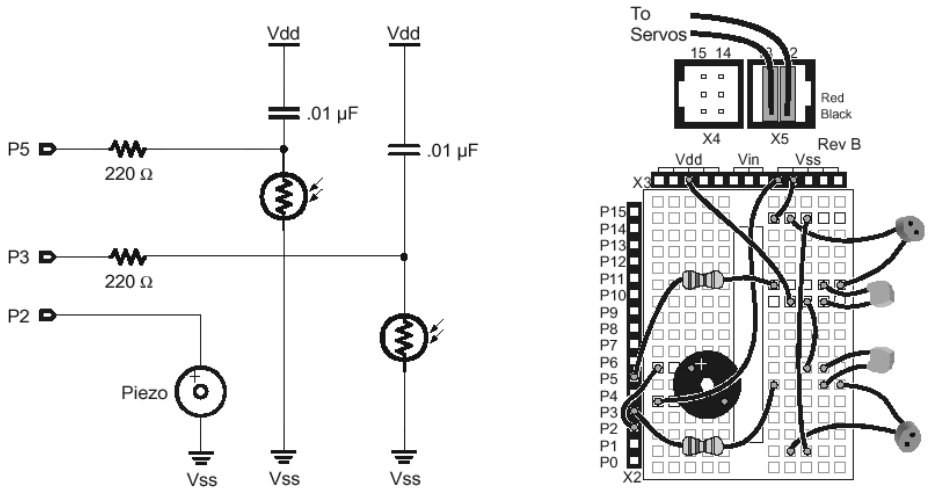


Bild 9: Schaltung für Linienverfolgung

4 Literaturhinweise

Einige Beispiel und Abbildungen in diesem Text wurden folgenden Quellen entnommen:

Parallax	***BASIC Stamp* Manual Version 1.9 Volume II** Benutzerhandbuch zur *BASIC Stamp II* von der Fa. Parallax
Parallax	**Robotics! Student Guide** Buch mit Experimenten für den BOE-Bot von der Fa. Parallax.
Paul H. Diez	**A Pragmatic Introduction to the Art of Electrical Engineering** Einführung in die Grundlagen elekronischer Schaltungen mit Hilfe des Board of Education

Obige Texte sind als PDF-Dateien kostenlos unter folgender Internetadresse zu erhalten:
http://www.parallax.com