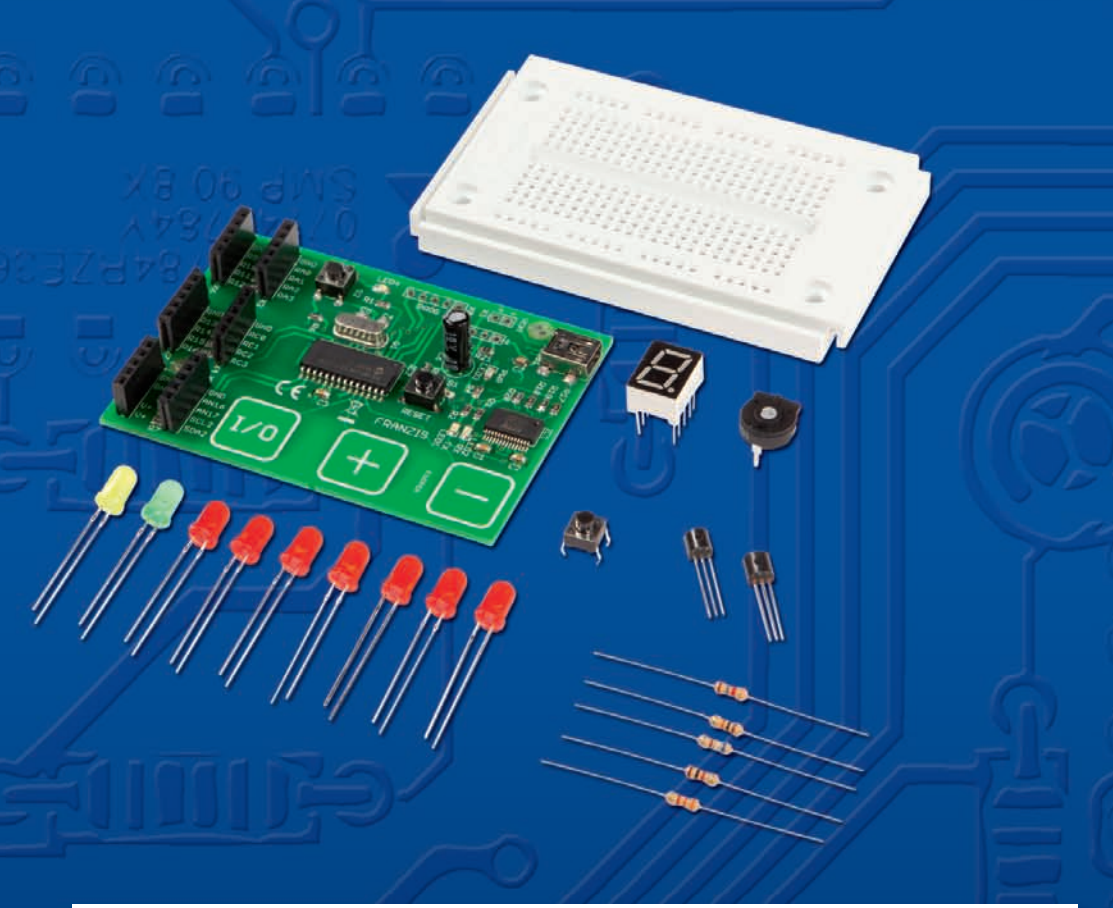
Lernpaket PIC-Mikrocontroller

Bearbeitet von
Michael Hofmann

1. Auflage 2011. Buch. 267 S.
ISBN 978 3 645 65069 4



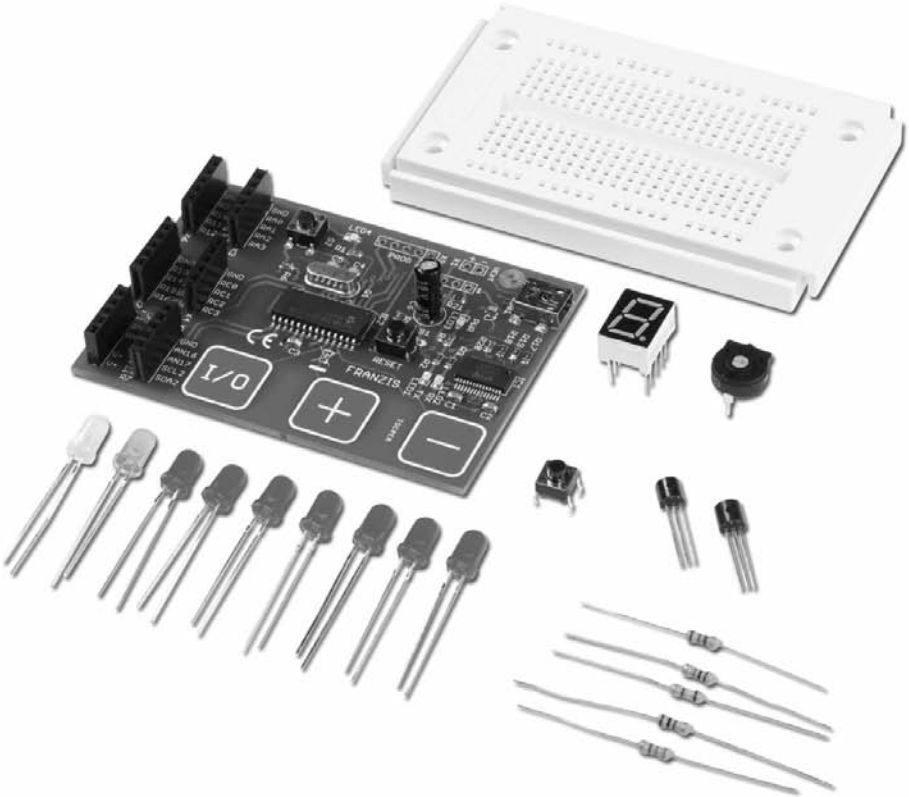
Experimentierhandbuch

Lernpaket PIC®-Mikrocontroller

Inklusive: Platine + Bauteile + CD-ROM + Handbuch mit 228 Seiten

FRANZIS

Lernpaket PIC®-Mikrocontroller



Experimentierhandbuch

Lernpaket PIC®-Mikrocontroller

Inklusive: Platine + Bauteile + CD-ROM + Handbuch mit 228 Seiten

FRANZIS

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.

Satz: DTP-Satz A. Kugge, München
art & design: www.ideehoch2.de

Vorwort

Mit einem Mikrocontroller kann man sehr komplexe Aufgaben mit nur einem Baustein lösen. Daher findet man ihn auch in nahezu jedem Gerät. Dieses Lernpaket erleichtert Ihnen den Einstieg in die Welt der Mikrocontrollerprogrammierung. Anhand verschiedener Beispiele wird beschrieben, welche Möglichkeiten ein Mikrocontroller bietet und wie die Funktionen des Bausteins angesprochen werden. Im Ihnen vorliegenden Lernpaket wird der PIC18F23K22 von Microchip verwendet. Dabei handelt es sich um einen kleinen aber dennoch leistungsfähigen Mikrocontroller, mit dem viele Schaltungen realisiert werden können.

Die Beispiele des Lernpakets beginnen bei der einfachen Ansteuerung einer Leuchtdiode und dem Abfragen eines Tasters. Im Lauf des Buchs werden sie etwas komplexer. Es wird beschrieben, wie man auf unterschiedliche Arten ein Lauflicht realisieren kann und wie eine 7-Segment-Anzeige angesteuert wird, um Ziffern oder Buchstaben anzuzeigen. Später wird gezeigt, wie man mit dem integrierten Analog-Digital-Wandler analoge Spannungen misst. Bei einem weiteren Beispiel wird über einen Temperatursensor die Temperatur gemessen und über die Schnittstelle an den PC übertragen. Neben den Beispielen für einen Treppenhausautomaten, der das Licht nach einer einstellbaren Zeit ausschaltet, und einer intelligenten Scheibenwischersteuerung wird auch gezeigt, wie man Touch-Tasten abfragen kann. Über diese Tasten, die auf der Leiterplatte des Lernpakets integriert sind, wird die Helligkeit einer Leuchtdiode gesteuert. Alle Beispielprogramme sind in der Programmiersprache C geschrieben, die bei der Programmierung von Mikrocontrollern sehr beliebt ist.

Sie sollten die Beispiele nacheinander ausprobieren, da sie teilweise aufeinander aufbauen. Versuchen Sie anschließend, die Beispiele nach Ihren Vorstellungen zu verändern und eigene Programme zu entwerfen. Ich wünsche Ihnen viel Spaß und Erfolg beim Experimentieren mit diesem Lernpaket.

Michael Hofmann

Inhaltsverzeichnis

1 Überblick über Mikrocontroller 13
1.1 Aufbau und Funktionsweise des PIC18F23K22 16
1.2 Blockschaltbild 16
1.2.1 Die Recheneinheit 18
1.2.2 Taktgenerator 18
1.2.3 Speicher 18
1.2.4 Multiplikationseinheit 19
1.2.5 Timer 19
1.2.6 EEPROM 19
1.2.7 MSSP 20
1.2.8 EUSART 20
1.2.9 CTMU 20
1.2.10 ADC 20
1.2.11 DAC 21
1.2.12 I/O-Ports 21
1.2.13 Konfigurationsbits 22

2 Die Leiterplatte des Lernpakets 23
2.1 Der Mikrocontroller 23
2.2 Taster und LED 25
2.3 Die Erweiterungsbuchsenleisten 26
2.4 Versorgungsspannung und USB-Anschluss 27
2.5 Sicherheitshinweise 29

3 Bauteile im Lernpaket 31
3.1 Steckbrett 31
3.2 Widerstand 31
3.3 Taster 34
3.4 Leuchtdioden (LED) 35
3.5 Transistor 37
3.6 Potenziometer 38
3.7 Temperatursensor 39
3.8 7-Segment-Anzeige 40

4 Die Programmierung mit MPLAB 41
4.1 Installation der Software 42
4.1.1 Installation von MPLAB 42
4.1.2 Installation des HI-TECH C Compilers 43
4.2 Beispiele des Lernpakets 43
4.3 Anlegen eines Projekts 44
4.4 Die Arbeitsoberfläche 47
4.5 Das View-Menü 54
4.5.1 Hardware Stack 54
4.5.2 Watches 55
4.5.3 Disassembly Listing 55
4.5.4 EEPROM 56
4.6 Breakpoints 57
4.7 Texteditor 58
4.8 Simulator 59
4.8.1 Grundeinstellungen 59
4.8.2 Asynchroner Stimulus 60
4.9 Logikanalysator 61
4.10 Programmieren des Mikrocontrollers 63

5 Hardware mit dem PC verbinden 65
5.1 Installation des USB-Treibers 65
5.2 Konfiguration des FT232R 65

6 Der Bootloader 71
6.1 Installation Bootloader PC-Tool 71
6.2 Funktionalität des Bootloader PC-Tools 71

7 Eine kleine Einführung in C 77
7.1 HI-TECH C Compiler 77
7.2 Der Compiler 78
7.3 Allgemeines 79
7.4 Aufbau eines C-Programms 79
7.4.1 Makros 81
7.4.2 Variablen 82
7.4.3 Zahlenformate 83
7.4.4 Variablen-Arrays 84
7.4.5 Operatoren 85
7.5 Kontrollstrukturen 86

7.5.1 if-Anweisung 86
7.5.2 switch-Anweisung 87
7.6 Schleifen 90
7.6.1 for-Schleife 90
7.6.2 while-Schleife 93
7.6.3 do-while-Schleife 94
7.7 Funktionen 95
7.7.1 Funktionsdefinition 95
7.7.2 Prototyp 96
7.7.3 Funktionsaufruf 97
7.8 Globale und lokale Variablen 98
7.9 Zeiger 99
7.9.1 Verwendung von Zeigern 100
7.9.2 Zeiger auf Arrays 100
7.9.3 Zeiger an Funktionen übergeben 102
7.10 Besonderheiten des HI-TECH C Compilers 102
7.10.1 Include-Datei 103
7.10.2 Mischen von C und Assembler 103
7.10.3 Lesen und Schreiben von Registern 104

8 Beispiele 107
8.1 Grundsätzlicher Aufbau der Beispiele 107
8.2 Vorbereitungen 109

9 Ein- und Ausgabe-Ports 111
9.1 Beispiel: Taster und LED 113
9.2 Beispiel: Blinker 114

10 Logik-Gatter 117
10.1 UND-Gatter 118
10.2 ODER-Gatter 120
10.3 NAND-Gatter 121
10.4 NOR-Gatter 122
10.5 XOR-Gatter 124
10.6 RS-Flip-Flop 125

11 Ansteuerung mehrerer LEDs 129
11.1 Einfaches Lauflicht 129
11.2 Knight-Rider-Lauflicht 133
11.3 Memory-Lauflicht 138
11.4 Würfel 142
11.5 Spiel Adler oder Zahl 146

12 Timer 151
12.1 Der Timer 0 151
12.2 Ampelschaltung 152

13 Tasterentprellung 159
13.1 Binärzähler 161

14 7-Segment-Anzeige 165

15 Watchdogtimer 173
15.1 Schlafmodus 173
15.2 Blitzlicht 174

16 Schalten hoher Ströme 177
16.1 Treppenhausautomat 177

17 Interrupts 185
17.1 Scheibenwischer 185

18 Verarbeitung analoger Signale 195
18.1 Analog-Digital-Wandler (ADC) 195
18.1.1 Berechnung des Spannungswerts 197
18.1.2 Aufteilung des digitalisierten Werts 198
18.2 Durchgangsprüfer 199

19 Temperatursensor 205
19.1 Temperaturmessung 207

20 Datenübertragung zum PC 211
20.1 Protokoll der USART-Schnittstelle 212
20.2 Software zur Datenübertragung 213
20.3 Einstellen der Baudrate 214

20.4 PC-Steuerung 216
20.4.1 Funktionen für die Kommunikation 217
20.4.2 Beispiel PC-Steuerung 219
20.5 Beispiel Thermometer 222

21 Internes EEPROM 229
21.1 Funktionen für das Schreiben und Lesen des EEPROM 229
21.2 Temperatur-Logger 231

22 Kapazitiver Touch-Sensor 239
22.1 Funktionsweise 239
22.2 Touchkey 241

23 Helligkeit einer LED über PWM steuern 251
23.1 Funktionsweise der PWM 251
23.2 Beispiel Touchdimmer 254

Quellenverzeichnis 263

Stichwortverzeichnis 265

1 Überblick über Mikrocontroller

Mikrocontroller findet man heute in nahezu jedem elektronischen Gerät. Sie werden in Thermometern zur Anzeige der Temperatur eingesetzt und sie steuern in Kaffeeautomaten die richtige Zusammensetzung des Kaffees. Mikrocontroller öffnen und schließen automatische Garagentore und unterstützen den Autofahrer in gefährlichen Situationen durch ABS und ESP.

Ein Mikrocontroller (kurz: µC) oder auch Mikrocomputer ist ein elektronischer Baustein, der Berechnungen und Steuerungen ähnlich einem PC ausführen kann. Allerdings hat er häufig nur die Aufgabe ein einzelnes Gerät zu steuern, während der PC viele unterschiedliche Geräte steuern und auswerten muss. Der PC wird für die Textverarbeitung genauso genutzt wie für ein Computerspiel. Er muss daher sehr flexibel sein und ausreichend Speicherplatz und Rechenleistung zur Verfügung stellen. Ein Mikrocontroller hingegen wird für eine spezielle Aufgabe ausgewählt. Er wird z. B. nur für die Steuerung einer Heizung verwendet, wozu keine hohe Rechenleistung erforderlich ist und wenig Speicher benötigt wird. Es muss lediglich die Temperatur gemessen und ausgewertet werden. Allerdings ist die Grenze, an der der µC aufhört und ein PC anfängt, fließend. So wird in einem Mobiltelefon der Mikrocontroller für das Wählen einer Rufnummer und den Aufbau eines Telefonates benötigt. Außerdem kann man damit auch E-Mails versenden, Videos aufzeichnen und Musik hören.

Da es nicht für jede spezielle Aufgabe einen speziellen Mikrocomputer gibt, kann man aus einer großen Anzahl von Bausteinen einen geeigneten auswählen. Es gibt unterschiedliche Hersteller, die viele Varianten von Controllern produzieren, die sich aber in der grundsätzlichen Funktionsweise kaum unterscheiden. Die Mikrocontroller verfügen über eine unterschiedliche Anzahl an Ein- und Ausgängen sowie über spezielle Hardware-Bausteine im Innern, mit denen verschiedene Aufgaben ver-

einfacht werden. So verfügt nahezu jeder µC über einen Timer, mit dem man Zeiten bestimmen und Signale definierter Länge generieren kann. Ebenso haben sehr viele Bausteine einen integrierten Analog-Digital-Wandler, der es ermöglicht, analoge Signale wie z. B. Temperatur oder Batteriespannung zu messen.

Die Größe des Bausteins wird hauptsächlich von der Anzahl der Ein- und Ausgänge bestimmt. So kann ein Mikrocontroller, der nur die Batteriespannung überwachen soll und beim Unterschreiten eines Schwellwerts ein Signal ausgeben soll, mit wenigen Pins auskommen. Ein Controller, der die Temperatur von mehreren Zimmern regeln soll und die Steuerung über ein Farbdisplay mit Touchscreen visualisiert, benötigt hingegen wesentlich mehr Ein- und Ausgänge. Daher findet man auch bei den Herstellern Mikrocontroller, die mit 6 Pins auskommen, und andere, die mehrere hundert davon haben. Ganz grob kann man sagen, dass auch die Rechenleistung mit der Anzahl der Pins steigt, da bei vielen Pins auch mehr Aufgaben bewältigt werden müssen. Auch die Anzahl der Bits, die gleichzeitig verarbeitet werden, steigt mit der Größe des Mikrocontrollers. So gibt es Bausteine mit einer Busbreite von 4 Bit bis hoch zu 32 Bit. Da im PC-Bereich auch schon 64-Bit-Prozessoren eingesetzt werden, wird es auch bei Mikrocontrollern nicht mehr lange dauern, bis diese mit 64 Bit breiten Werten rechnen. Ob dies nun ein Vorteil oder eher ein Nachteil ist, muss dann wohl jeder Entwickler selbst entscheiden. Die Mikrocontroller mit einer Wortbreite von 4 Bit findet man hauptsächlich in Museen und sehr alten Geräten. Sie werden aber auch heute noch verkauft, wenn es gilt, ein altes Gerät wieder funktionstüchtig zu machen. Für eine Neuentwicklung spielen diese Bausteine keine Rolle mehr. Am weitesten verbreitet sind Mikrocontroller mit einer Wortbreite von 8–16 Bit. Hier ist die Auswahl an verfügbaren Bausteinen sehr groß. Die 32-Bit-Controller werden hauptsächlich in Geräten eingesetzt, die ein Farbdisplay ansteuern, verschiedene Speichermedien wie USB oder SD-Karten unterstützen und zum Datenaustausch mit dem PC verbunden werden.

Für eine Vielzahl von Anwendungen sind 8-Bit-Mikrocontroller völlig ausreichend und zudem auch noch günstig zu erwerben. Es können auch kleine Schaltungen aufgebaut werden, ohne eine Leiterplatte herstellen zu lassen. Da die größeren Typen ähnlich funktionieren wie die Kleinen, wird im Folgenden hauptsächlich auf die Entwicklung einer Schaltung mit einem 8-Bit-Mikrocontroller eingegangen. Die vorgestellten Schaltungen und Beispielprogramme lassen sich auf diese Weise einfach aufbauen, simulieren und ausprobieren.

In diesem Lernpaket wird ein Controller der Firma Microchip verwendet. Dabei handelt es sich um den Typ PIC18F23K22. Die Abkürzung »PIC« steht für *Programmable Integrated Circuit* und bedeutet, dass es sich um einen programmierbaren Baustein handelt. Die Controller anderer Hersteller (z. B. Atmel, Motorola, Texas Instruments etc.) funktionieren nach dem gleichen Prinzip. Sie unterscheiden sich hauptsächlich in der Zusammenstellung der Features und der Bezeichnung der Register für die Steuerung der internen Hardware.

Warum in diesem Buch die Wahl auf die Produkte von Microchip fiel, liegt unter anderem daran, dass von Microchip eine komplette kostenlose Entwicklungsumgebung zur Verfügung gestellt wird. Die Artikel können vom Controller bei den bekannten Lieferanten für elektronische Bauteile gekauft werden. Sie erfreuen sich großer Beliebtheit, weshalb man viele weitere Beispiele und Diskussionen im Internet findet.

Bei dem Mikrocontroller in diesem Lernpaket wurde bereits ein Bootloader vorinstalliert. Ein Bootloader ermöglicht das Programmieren des Mikrocontrollers ohne ein spezielles Programmiergerät. Hierbei ist aber auch Vorsicht geboten. Wird der Bootloader durch einen Fehler im Programm versehentlich überschrieben oder verändert, kann es dazu führen, dass der Mikrocontroller nicht mehr programmiert werden kann. Dann ist ein Programmiergerät erforderlich, um das Programm oder den Bootloader erneut in den Controller zu laden.

Alle Beispiele für dieses Lernpaket sind in der Programmiersprache C programmiert. Zu jedem Beispiel findet man den kompletten funktionsfähigen und kommentierten Code auf der CD-ROM. Um ein optimales Lernen zu gewährleisten, sollte man aber versuchen, die Beispiele selbst zu programmieren und die Versionen auf der CD-ROM nur im Notfall zum Nachsehen verwenden. Die Beispiele sind nicht auf kürzeste Ausführungszeit oder kleinsten Speicherplatz optimiert. Sie demonstrieren lediglich die verschiedenen Features eines Mikrocontrollers und die Anwendung der Programmiersprache C.

1.1 Aufbau und Funktionsweise des PIC18F23K22

Der PIC18F23K22 gehört zu einer kleinen Familie von mehreren Mikrocontrollern, die über die gleichen Funktionen verfügen. Die anderen Typen haben allerdings etwas mehr Speicher und bieten somit mehr Platz für umfangreichere Programme. Die größeren Mikrocontroller verfügen über 40 oder 44 Pins und haben somit auch mehr Ein- und Ausgangs-Pins.

1.2 Blockschaltbild

Der Mikrocontroller PIC18F23K22 verfügt über verschiedene Peripherieteile wie z. B. Timer, AD-Wandler und I/O-Ports. Wie diese einzelnen Blöcke miteinander kommunizieren, ist im folgenden Blockschaltbild dargestellt.

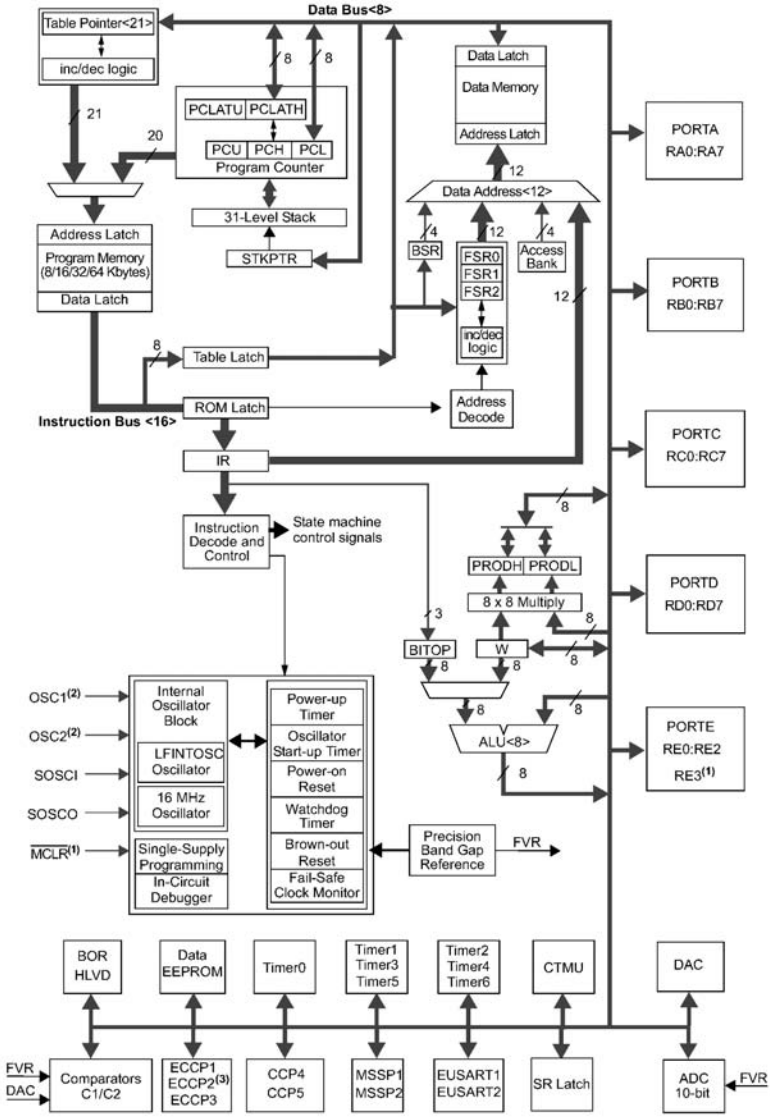


Bild 1.1: Blockschaltbild des PIC18F23K22 (Quelle: Datenblatt Microchip)

1.2.1 Die Recheneinheit

Die eigentliche Recheneinheit (ALU = Arithmetic Logic Unit), mit der Additionen und logische Verknüpfungen durchgeführt werden, nimmt nur einen vergleichsweise geringen Platz in Anspruch. Um z. B. einen analogen Wert an einem Eingang zu bestimmen, muss man der Peripherie nur mitteilen, von welchem Pin gelesen werden soll. Die interne Hardware des ADC (Analog Digital Converter) kümmert sich dann um die Wandlung des analogen Signals in einen digitalen Wert. Liegt dieser Wert im entsprechenden Register vor, wird ein Flag (Flagge/Zeichen) gesetzt, damit das Programm erkennt, dass die Wandlung beendet ist und der ermittelte Wert weiterverarbeitet werden kann.

1.2.2 Taktgenerator

Besonders wichtig ist auch der Taktgenerator, ohne den kein Mikrocontroller läuft. In der Regel befindet sich auf der Schaltung direkt neben dem Mikrocontroller ein Quarz. Er wird vom Mikrocontroller zum Schwingen angeregt, schwingt dann mit konstanter Frequenz und gibt so den Takt vor. Es besteht auch die Möglichkeit, dass der Takt im Innern des Mikrocontrollers erzeugt wird. Er unterliegt allerdings größeren Schwankungen. Die Schwankungen sind für viele Anwendungen von geringer Bedeutung, können aber zu Problemen führen, wenn genaue Zeiten bestimmt werden sollen. Der Mikrocontroller führt bei jedem Takt intern eine Aktion aus und bearbeitet so die Befehle des Programms.

1.2.3 Speicher

Des Weiteren verfügt der Controller noch über einen Flash-Speicher (nicht flüchtiger Speicher/Program Memory), in dem die einzelnen Befehle für die Programmausführung gespeichert werden. Dieser Speicher verliert seinen Inhalt nach dem Ausschalten der Versorgungsspannung nicht. Im RAM-Speicher (flüchtiger Speicher/Data Memory; Arbeitsspeicher) werden die Variablen gespeichert, die während des Programms geändert werden. Der Inhalt des Arbeitsspeichers wird gelöscht, sobald der Mikrocontroller ausgeschaltet wird. Beim PIC18F23K22 sind 8 KBytes Flash-Speicher und 512 Bytes Arbeitsspeicher vorhanden. Das erscheint sehr wenig im Vergleich zu modernen PCs, aber man wird

merken, dass auch mit wenigen Ressourcen umfangreiche und komplexe Programme realisiert werden können.

1.2.4 Multiplikationseinheit

Eine kleine Besonderheit gegenüber kleineren PIC-Mikrocontrollern ist die Multiplikationseinheit (8x8 Multiply). Dadurch können Multiplikationen schneller durchgeführt werden. Bei den kleinen Mikrocontrollern, z. B. denen der PIC16F-Reihe, müssen Multiplikationen in Form von Software-Algorithmen gelöst werden und sind daher auch relativ zeitaufwendig.

1.2.5 Timer

Die Einsatzbereiche für Timer sind unterschiedlich und vielseitig. Sie werden verwendet, um genaue Zeiten zu bestimmen oder zu erzeugen. Soll z. B. jede Sekunde ein Impuls ausgegeben werden, kann dies auf einfache Weise mit einem Timer realisiert werden. Ein Timer kann mit einem Countdown verglichen werden, der ein Signal abgibt, wenn die Zeit abgelaufen ist. Nach dem Ablauf der eingestellten Zeit wird ein Interrupt (Programmunterbrechung) ausgelöst und der Impuls kann auf einem Ausgangs-Pin ausgegeben werden. Das Programm wird ganz normal bearbeitet, während der Timer im Hintergrund läuft.

1.2.6 EEPROM

Im internen EEPROM können bis zu 256 Bytes gespeichert werden. Das ist nicht sehr viel, ist aber in der Regel ausreichend, um Kalibrierwerte, z. B. für einen Temperatursensor, zu speichern. Wird mehr Speicherplatz benötigt, um z. B. die Temperatur über einen längeren Zeitraum aufzuzeichnen, ist es vermutlich erforderlich, den Speicher mit einem externen EEPROM zu erweitern. Diese externen EEPROMs können über den I²C-Bus (Inter-Integrated Circuit) oder über SPI (Serial Peripheral Interface) an den Mikrocontroller angeschlossen werden.

1.2.7 MSSP

MSSP steht für *Master Syncronous Serial Port* und ermöglicht die Kommunikation über I²C und SPI zu externen Bausteinen. Diese Schnittstellen werden von vielen Bauteilen wie z. B. EEPROMs oder externen AD-Wandlern unterstützt. Um die Schnittstellen zu verwenden, werden die Register des Moduls entsprechend parametriert und die interne Hardware kümmert sich um die Datenübertragung. Anhand von Flags wird man über Fehler oder eine erfolgreiche Datenübertragung informiert.

1.2.8 EUSART

Das EUSART-Modul (Enhanced Universal Synchronous Asynchronous Receiver Transmitter) wird für serielle Datenübertragungen verwendet. Eine häufige Anwendung ist die Kommunikation über die RS-232-Schnittstelle. Dabei handelt es sich um eine asynchrone Datenübertragung, bei der kein eigenes Taktsignal geliefert wird. Der Sender und der Empfänger müssen die gleiche Taktrate (auch *Baudrate* genannt) verwenden. Um dies zu realisieren, verfügt das Modul über einen Baudrate-Generator, der aus dem Oszillatortakt die entsprechende Baudrate ableitet.

1.2.9 CTMU

Eine Besonderheit dieses Mikrocontrollers ist die CTMU (Charge Time Measurement Unit). Damit ist es möglich, sehr genau differenzielle Zeiten zu bestimmen. So kann man Kapazitäten und Kapazitätsänderungen ermitteln. In diesem Lernpaket wird das CTMU-Modul für die Realisierung eines kapazitiven Touchs verwendet. Durch das Berühren einer Sensorfläche auf der Leiterplatte wird die Kapazität erhöht. Diese Kapazitätserhöhung kann bestimmt werden. Bei entsprechender Höhe kann ein Schaltereignis, z. B. Anschalten einer LED, ausgelöst werden.

1.2.10 ADC

Fast alle Mikrocontroller verfügen über einen Analog-Digital-Wandler (ADC = Analog Digital Converter). Mit ihm ist es möglich, eine analoge Spannung, z. B. von einem Temperatursensor, auszulesen und den Wert in digitaler Form abzuspeichern. Bei dem PIC18F23K22 sind 19 Kanäle

vorhanden. Diese 19 Kanäle werden über einen Multiplexer dem eigentlichen ADC-Modul zugeführt. Eine gleichzeitige Wandlung von zwei oder mehr Kanälen ist leider nicht möglich. Nachdem ein Kanal gewandelt wurde und das Ergebnis vorliegt, kann der nächste Kanal gewandelt werden. Der integrierte AD-Wandler löst die analoge Spannung mit 10 Bit auf. Eine Auflösung von 10 Bit entspricht 1.024 Stufen. Bei einer Versorgungsspannung von 5 V resultiert daraus eine minimale Stufe von ca. 4,88 mV. Das bedeutet, dass Signale kleiner als 4,88 mV nicht aufgelöst werden können.

1.2.11 DAC

Die Auflösung des Digital-Analog-Wandlers (DAC = Digital Analog Converter) ist im Vergleich zum ADC deutlich geringer. Der interne DAC verfügt nur über 32 Stufen, was einer Auflösung von 5 Bit entspricht. Um die analoge Spannung zu generieren, wird eine Referenzspannung (kann auch die Versorgungsspannung sein) an 32 in Serie geschaltete Widerstände gelegt. Dadurch ergibt sich an jedem Verbindungspunkt der Widerstände eine andere Spannung. Diese einzelnen Spannungen können über einen Multiplexer abgegriffen und auf dem Ausgang ausgegeben werden.

1.2.12 I/O-Ports

Das wahrscheinlich Wichtigste an Mikrocontrollern sind die Ein- und Ausgangsports, auch *I/O-Ports* genannt. Ohne diese Ports könnten keine externen Signale an den Mikrocontroller angelegt werden. Über die Register der I/O-Ports wird dem Mikrocontroller mitgeteilt, bei welchem Pin es sich um einen Ein- oder Ausgang handelt. Zusätzlich muss dem Mikrocontroller noch mitgeteilt werden, ob es sich bei einem Eingang um einen digitalen Eingang (nur 0 und 1) oder einen analogen Eingang handelt. Nach dem Einschalten des Mikrocontrollers oder nach einem Reset sind alle Pins als analoge Eingänge geschaltet. Will man z. B. einen Taster abfragen, muss man zuerst den entsprechenden Pin als digitalen Eingang definieren. Die Ports sind in Gruppen mit je 8 Ein- oder Ausgängen zusammengefasst (Port A bis Port E). Da die kleineren Mikrocontroller über weniger Pins verfügen als die größeren, sind nicht alle

Portpins verfügbar. Bei dem PIC18F23K22 gibt es nur die Portpins RA0 bis RA5. Die Pins RA6 und RA7 sind nicht vorhanden.

1.2.13 Konfigurationsbits

Neben den Registern, die während der Laufzeit des Programms beschrieben und gelesen werden, verfügt der Mikrocontroller auch noch über Konfigurationsbits, die auch als Fuses (Sicherungen) bezeichnet werden. Diese Bits werden beim Programmieren des Mikrocontrollers entsprechend den Anforderungen gesetzt und werden während der Laufzeit nicht geändert. Über diese Konfigurationsbits wird dem Mikrocontroller z. B. mitgeteilt, welcher Oszillatortyp verwendet werden soll und ob bestimmte Bereiche gegen unbefugtes Lesen oder Schreiben geschützt werden sollen.

3 Bauteile im Lernpaket

Im Lernpaket sind verschiedene Bauteile enthalten, mit denen unterschiedliche Beispiele aufgebaut werden können. Im Folgenden werden die Funktionsweise und die entsprechende Belegung der Anschlüsse erklärt.

3.1 Steckbrett

Für die Beispiele in diesem Lernpaket ist es erforderlich, die Bauteile in unterschiedlichen Anordnungen zusammenzuschalten. Auf dem beiliegenden Steckbrett (SYB-46) können die Bauteile gesteckt und über Drähte untereinander und mit dem Mikrocontroller verbunden werden. Die Buchsen des Steckbretts sind teilweise miteinander verbunden, um die Verschaltung zu vereinfachen. In Bild 3.1 erkennt man, welche Buchsen miteinander verbunden sind.

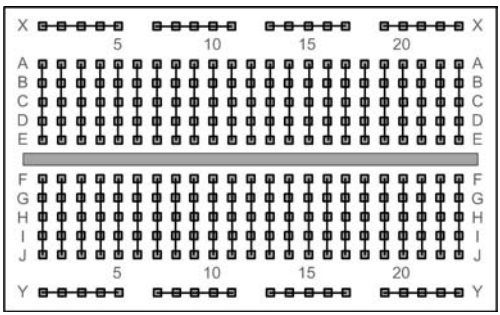


Bild 3.1: Steckbrett mit internen Verbindungen

3.2 Widerstand

Widerstände sind Strom begrenzende Bauteile. Sie sorgen dafür, dass man die Ströme und Spannungen in einer Schaltung entsprechend ein-

stellen kann. Sie werden z. B. eingesetzt, um den Strom durch eine Leuchtdiode zu begrenzen. Würde man eine Leuchtdiode (LED = Light Emitting Diode) ohne Vorwiderstand betreiben, würde diese sehr schnell überlastet und durchbrennen. In Bild 3.2 ist ein Widerstand mit seinem zugehörigen Schaltplansymbol dargestellt.

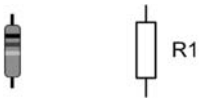


Bild 3.2: Widerstand und Schaltplansymbol

Widerstände gibt es in vielen verschiedenen Werten, üblicherweise zwischen 1 Ω und 10 MΩ. Die Zwischenwerte sind, je nach Toleranz, gröber oder feiner abgestuft. Um die unterschiedlichen Werte voneinander unterscheiden zu können, haben die bedrahteten Widerstände einen aufgedruckten Farbcode.

Tabelle 3.1: Farbcodes von Widerständen

Farbe	*1. Ring*	*2. Ring*	*Multiplikator*	*Toleranz*
Schwarz	0	0	$x10^0$	-
Braun	1	1	$x10^1$	+/–1 %
Rot	2	2	$x10^2$	+/–2 %
Orange	3	3	$x10^3$	-
Gelb	4	4	$x10^4$	-
Grün	5	5	$x10^5$	+/–0,5 %
Blau	6	6	$x10^6$	+/–0,25 %
Violett	7	7	$x10^7$	+/–0,1 %
Grau	8	8	$x10^8$	-
Weiß	9	9	$x10^9$	-
Gold	-	-	$x10^{-1}$	+/–5 %
Silber	-	-	$x10^{-2}$	+/–10 %

In diesem Lernpaket werden Widerstände mit einer Toleranz von +/–5% verwendet. Sie haben insgesamt 4 Ringe. Widerstände mit genauerer

Toleranz können auch mehr als 4 Ringe haben. Die ersten beiden Ringe stehen für die ersten beiden Ziffern des Werts. Der dritte Ring gibt den Multiplikator an, mit dem die beiden Ziffern multipliziert werden müssen, um auf den Widerstandswert zu kommen. Man kann sich auch merken, dass der 3. Ring die Anzahl der Nullen angibt, die den ersten beiden Ziffern hinzugefügt werden müssen. Der 4. Ring gibt die Toleranz des Werts an. Bei einer Toleranz von +/–5 % kann ein Widerstand, der einen nominalen Wert von 1 kΩ hat, in der Realität zwischen 0,95 kΩ und 1,05 kΩ liegen.

In Bild 3.3 ist ein Beispiel für einen 1,5-kΩ-Widerstand mit einer Toleranz von 5 % gezeigt.

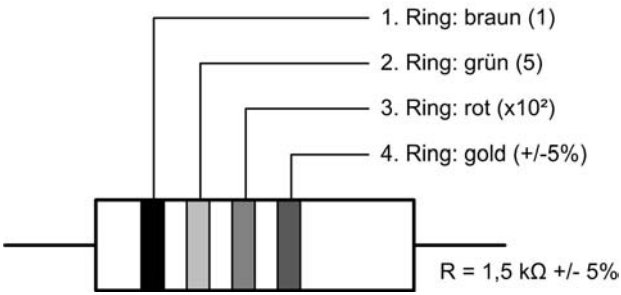


Bild 3.3: Beispiel für einen Farbcode

Der Farbcode ist in Form von farbigen Ringen auf den Widerstand gedruckt. Um den Code zu lesen, beginnt man bei dem Ring, der näher an der Außenseite des Widerstands ist. Die ersten drei Ringe geben den Widerstandswert an und der vierte Ring kennzeichnet die Toleranz des Widerstands. Als Anfänger ist es etwas schwierig, sich die Farben und die dazugehörigen Werte zu merken. Eine kleine »Eselsbrücke« erleichtert das:

*S*chwarz	0	*S*top
*B*raun	1	*B*ei
*R*ot	2	*R*ot
*O*range	3	*O*der
*G*elb	4	*G*elb
*G*rün	5	*G*rün

*B*lau	6	*B*ietet
*V*iolett	7	*V*iel
*G*rau	8	*G*efahrloseren
*W*eiß	9	*W*eg

Mithilfe des Merksatzes: »Stop bei Rot oder Gelb, Grün bietet viel gefahrloseren Weg«, kann man sich die Reihenfolge der einzelnen Farben merken. Die Anfangsbuchstaben des Merksatzes stehen für die Anfangsbuchstaben der jeweiligen Farbe.

3.3 Taster

Durch Taster wird es dem Benutzer der Schaltung auf einfache Weise ermöglicht, mit der Schaltung zu kommunizieren. Wird das Drücken eines Tasters von der Schaltung erkannt, kann eine Funktion ausgelöst werden – im einfachsten Fall das Anschalten eines Lichts oder einer LED. Ein Taster hat mindestens 2 Anschlüsse, die bei Betätigung des Tasters miteinander verbunden werden. In Bild 3.4 ist der Taster, der diesem Lernpaket beigelegt ist, dargestellt.

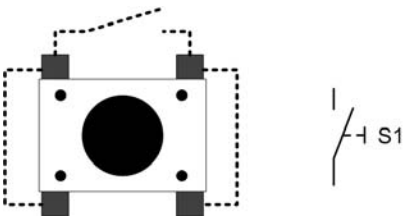


Bild 3.4: Taster mit Schaltplansymbol

Dieser Taster verfügt auf den ersten Blick über 4 Anschlüsse. Allerdings sind jeweils 2 von diesen Anschlüssen miteinander verbunden und der Schaltkontakt liegt zwischen den gebrückten Anschlüssen. Die 4 Anschlüsse haben den Vorteil, dass der Taster sehr gut auf einer Leiterplatte aufgelötet und betätigt werden kann. Ein Taster mit nur 2 Anschlüssen ist mechanisch nicht so stabil und kann bei Betätigung wackeln.

Ein großer Nachteil von vielen Tastern ist, dass sie Prellen. Das bedeutet, dass der Kontakt durch Betätigung, aufgrund eines federnden Verhal-

tens, mehrmals geschlossen und geöffnet wird, bis sich ein stabiler Zustand eingestellt hat. Dieser Effekt kann mehrere Millisekunden dauern. Ein Mikrocontroller kann Eingänge sehr viel schneller abfragen und es kann zu fehlerhaften Interpretationen kommen, die dann zu einer ungewollten Funktionalität führen. Eine Lösung für dieses Problem wird in Kap. 13 vorgestellt.

3.4 Leuchtdioden (LED)

Leuchtdioden sind Halbleiter-Bauelemente, die bei einem Stromfluss Licht abgeben. Die Abkürzung LED steht für *Light Emitting Diode*. Die Ansteuerung von LEDs ist sehr einfach und aufgrund der geringen Leistungsaufnahme auch direkt mit einem Mikrocontroller möglich. Bild 3.5 zeigt eine LED mit dem entsprechenden Schaltplansymbol.

Bild 3.5: Leuchtdiode mit Schaltplansymbol

Da es sich um einen Halbleiter handelt, ist bei dem Einbau der Leuchtdiode auf die Polarität zu achten. Die beiden Anschlüsse sind mit A (Anode) und K (Kathode) gekennzeichnet. Damit man die Anschlüsse der LED auseinanderhalten kann, gibt es mehrere Merkmale. Hält man eine neue LED in den Händen, ist der Anschlussdraht für die Kathode etwas kürzer als der der Anode (*K* wie ***K**athode* oder ***k**urz*). Werden diese Anschlüsse mit einem Seitenschneider auf die gleiche Länge gebracht, fällt dieses Entscheidungsmerkmal weg. Als weiteres Merkmal findet man am Gehäuse der LED eine abgeflachte Seite, während die andere rund ist. Die abgeflachte Seite kennzeichnet ebenfalls die Kathode. Die letzte Möglichkeit sieht man, wenn man die LED gegen das Licht hält. Im Innern der LED sind zwei unterschiedlich große Anschlüsse. Der größere Anschluss ist die Kathode.

Leuchtdioden dürfen immer nur über einen Vorwiderstand betrieben werden, da der Stromfluss durch die LED begrenzt werden muss. Ein Widerstand ist die einfachste Art der Strombegrenzung. Sie kann aber auch über andere Schaltungsteile, z. B. Konstantstromquellen, ermöglicht

werden. Um den Vorwiderstand zu ermitteln, muss man die Vorwärtsspannung der Leuchtdiode kennen. Diese ist, je nach Farbe, etwas unterschiedlich. Als grobe Näherung kann man folgende Werte verwenden:

Tabelle 3.2: Vorwärtsspannungen verschiedenfarbiger LEDs

Farbe der LED	*Vorwärtsspannung U_F*
Rot	2,1 V
Gelb	2,0 V
Grün	2,4 V
Blau	3,9 V

Diese Werte können, je nach LED-Typ und Hersteller, schwanken. Für die Berechnung eines einfachen Vorwiderstands sind diese Werte in der Regel ausreichend. Muss für spezielle Anwendungen die genaue Vorwärtsspannung/der erlaubte Strom bekannt sein, muss man diese Werte dem Datenblatt des jeweiligen Herstellers entnehmen.

Um nun den Vorwiderstand für eine LED zu berechnen, muss zuerst der Strom durch die LED festgelegt werden. Je höher der Strom ist, desto heller leuchtet die LED. Ist der Strom zu hoch, leuchtet die LED nur einmal kurz auf und brennt dann durch. In der Regel sind 5–10 mA ausreichend und führen zu keiner Beschädigung der LED. Es gibt auch Low-Power-LEDs, die bereits bei geringeren Strömen ausreichende Helligkeit abgeben.

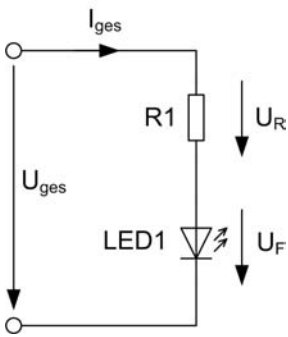


Bild 3.6: Schaltung LED mit Vorwiderstand

Bei einer Versorgungsspannung von 5 V und einem gewünschten Strom von 5 mA kann für eine rote LED der Vorwiderstand wie folgt berechnet werden:

$$R1 = \frac{U_{ges} - U_F}{I_{ges}} = \frac{5\,V - 2{,}1\,V}{5\,mA} = 580\,\Omega$$

Formel 3.1

Möchte man einen Widerstandswert aus der E24-Normreihe verwenden, wird man einen Widerstandswert von 580 Ω nicht finden. Daher muss man sich für einen Wert von 560 Ω (etwas höherer Strom) oder 620 Ω (etwas niedrigerer Strom) entscheiden.

3.5 Transistor

Bei dem Transistor im Lernpaket handelt es sich um einen NPN-Transistor vom Typ BC548. Das ist ein Kleinsignaltransistor, mit dem man keine allzu großen Ströme schalten kann.

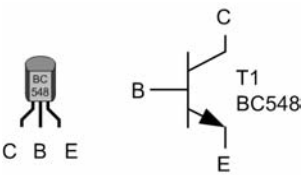


Bild 3.7: Transistor BC548

Mit einem Transistor hat man die Möglichkeit, durch einen kleinen Basisstrom (I_B) einen größeren Kollektorstrom (I_C) fließen zu lassen. Bei einem BC548B liegt die Stromverstärkung zwischen 200 und 450. Das bedeutet, dass ein Basisstrom von 1 mA einen Kollektorstrom von 200–450 mA fließen lassen kann. Der maximale Strom darf allerdings 300 mA nicht überschreiten, da sonst der Transistor zerstört werden könnte. Daher muss der Strom durch den Transistor über Widerstände oder einen Verbraucher begrenzt werden.

Soll ein Relais über einen Mikrocontroller geschaltet werden, ist das nicht mehr direkt über einen Ausgang möglich. Der erforderliche Strom

kann für die Ansteuerung der Relaisspule nicht geliefert werden. Um dieses Problem zu lösen, verwendet man einen Transistor, der das Relais schalten kann.

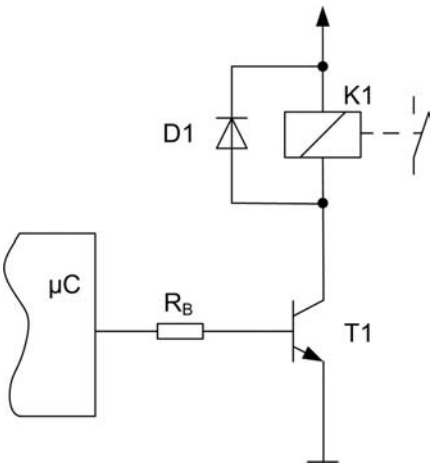


Bild 3.8: Ansteuerung eines Relais über den Mikrocontroller

Über den Basiswiderstand R_B wird der Basisstrom des Transistors begrenzt. Dieser liegt bei einem bipolaren Schalttransistor in der Regel zwischen 0,1 mA und 1 mA. Durch den geringen Basisstrom fließt ein höherer Kollektorstrom, der das Relais K1 schalten lässt. Bei der Diode D1 handelt es sich um eine sogenannte Freilaufdiode, die dafür sorgt, dass beim Ausschalten des Relais keine hohen Spannungen am Transistor und der restlichen Schaltung entstehen.

3.6 Potenziometer

Bei einem Potenziometer (kurz: Poti) handelt es sich um einen einstellbaren Widerstand. Der Widerstand kann im Allgemeinen durch Drehen oder Schieben verändert werden. Potenziometer werden häufig in einem Gerät eingebaut, um sie während des Betriebs zu verändern. In der kleineren Bauform werden Potenziometer als Trimmwiderstand (kurz: Trimmer) bezeichnet. Ein Trimmer wird verwendet, um bestimmte Werte in der Schaltung einmalig abzugleichen und während

des Betriebs nicht mehr zu verändern. Daher sind Trimmwiderstände häufig auch nur mit einem Schraubendreher zu verstellen.

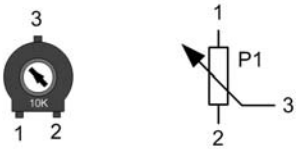


Bild 3.9: Potenziometer

Das Potenziometer oder der Trimmer verfügt über drei Anschlüsse. Zwischen den beiden äußeren Anschlüssen (1 und 2) liegt der gesamte Widerstand, der über einen Schleifer (3) abgegriffen wird. Der Schleifer liegt daher aus mechanischen Gründen in der Mitte der drei Pins.

3.7 Temperatursensor

Mikrocontroller werden oft verwendet, um verschiedene physikalische Größen zu messen, auszuwerten und dem Benutzer entsprechend anzuzeigen. In diesem Lernpaket wird beispielhaft die Temperatur gemessen, wofür ein Temperatursensor erforderlich ist. Es gibt sehr viele verschiedene Varianten von Temperatursensoren, die auch sehr unterschiedlich ausgewertet werden müssen. Bei dem hier vorliegenden Temperatursensor handelt es sich um den LM335 von National Semiconductor.

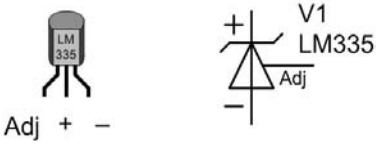


Bild 3.10: Temperatursensor LM335

Der LM335 erzeugt eine Spannungsänderung von 10 mV pro K oder °C. Lässt man einen Strom von 1 mA fließen, ergibt sich bei einer Temperatur von 25 °C eine Spannung von 2,98 V zwischen Kathode (+) und Anode (–). Diese Spannung ändert sich dann entsprechend um 10 mV bei einer Temperaturänderung um 1 °C. Die Genauigkeit dieses Sensors beträgt ca. +/–2 °C im nicht kalibrierten Fall und kann daher nicht für genaue Temperaturmessungen eingesetzt werden. Über den

Adjust-Pin (Adj) kann der Temperatursensor noch abgeglichen werden und man kann eine Genauigkeit von ca. +/–1 °C erreichen.

3.8 7-Segment-Anzeige

Eine 7-Segment-Anzeige ist eine relativ einfache Möglichkeit um Ziffern und einige Buchstaben darzustellen. Die einzelnen Segmente sind aus Leuchtdioden gefertigt, die bei Ansteuerung leuchten. Durch die Ansteuerung von mehreren Segmenten zur gleichen Zeit kann die gewünschte Ziffer angezeigt werden.

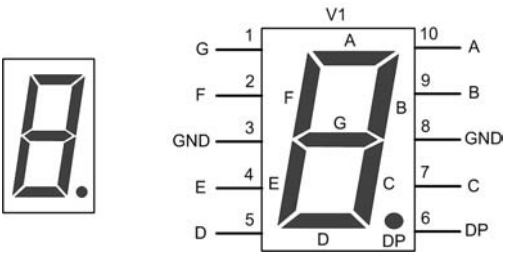


Bild 3.11: 7-Segment-Anzeige mit gemeinsamer Kathode

Bei 7-Segment-Anzeigen unterscheidet man zwischen zwei Varianten. Es gibt sie mit gemeinsamer Anode oder gemeinsamer Kathode. Das bedeutet, dass alle Anoden oder Kathoden der internen Leuchtdioden miteinander verbunden sind. Im Lernpaket wird eine 7-Segment-Anzeige mit gemeinsamer Kathode verwendet. Diese wird auf GND gelegt und die einzelnen Segmente werden über einen High-Pegel zum Leuchten gebracht. Für die Ansteuerung ist, genauso wie bei der Ansteuerung von LEDs, ein Vorwiderstand erforderlich. In Bild 3.11 sieht man, welche Pins mit welchem Segment der Anzeige verbunden sind. Um eine 7 darzustellen, muss an die Pins 10 (A), 9 (B) und 7 (C) ein High-Pegel gelegt werden.

10 Logik-Gatter

Die vorherigen Beispiele konnten noch alle mit der Leiterplatte des Lernpakets ohne zusätzliche Bauteile ausprobiert werden. Um die Beispiele für die Logikgatter auszuprobieren, wird ein zusätzlicher Taster benötigt. Er muss noch an die Leiterplatte über das Steckbrett angeschlossen werden. In Bild 10.1 zeigt die Schaltung, wie der zweite Taster angeschlossen werden muss, damit das Beispiel auf der CD-ROM funktioniert.

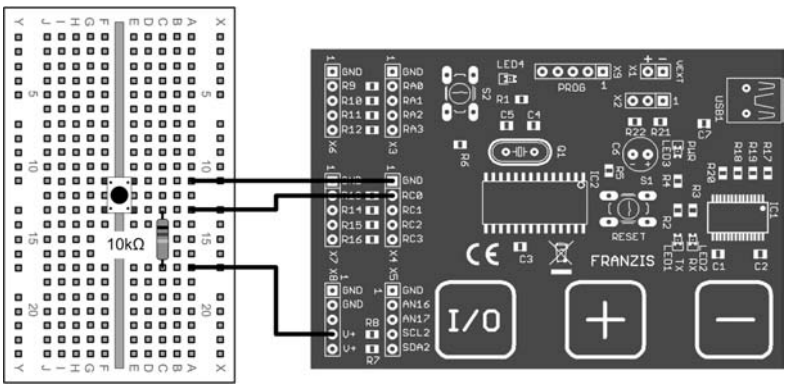


Bild 10.1: Anschluss des zusätzlichen Tasters

Für die Verdrahtung werden die folgenden Bauteile verwendet:

1 x Widerstand 10 kΩ (braun, schwarz, orange)

1 x Taster

3 x Verbindungsleitung ca. 15 cm

Der Schaltplan zu der Verdrahtung sieht folgendermaßen aus:

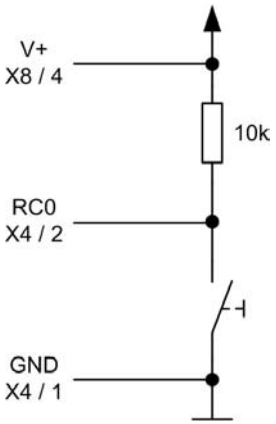


Bild 10.2: Schaltplan für zusätzlichen Taster

Der Widerstand hat einen Wert von 10 kΩ und dient als Pull-up-Widerstand, der den Pegel am Eingang des Mikrocontrollers auf einen High-Pegel legt, wenn der Taster nicht gedrückt ist. Fehlt dieser Widerstand, kann man nicht sicher sagen, welchen Pegel der Mikrocontroller vom Eingang liest. Der Taster schaltet den Pin RC0 auf Low-Pegel, sobald dieser gedrückt wird. Nach der Verdrahtung kann man alle Beispiele der Logik-Gatter ausführen, ohne die externe Beschaltung zu ändern.

10.1 UND-Gatter

Bei dem Beispiel des UND-Gatters wird eine LED geschaltet, wenn beide Taster gedrückt sind. Ist einer der beiden Taster nicht gedrückt, bleibt die LED dunkel. Diese Funktionalität verdeutlicht die Wahrheitstabelle:

TASTER1	*TASTER2*	*LED*
Nicht gedrückt	Nicht gedrückt	Aus
Nicht gedrückt	Gedrückt	Aus
Gedrückt	Nicht gedrückt	Aus
Gedrückt	Gedrückt	Ein

Das Beispielprogramm für ein UND-Gatter prüft in der Hauptschleife, ob beide Taster gedrückt sind, und schaltet die LED an oder aus.

```
/** D E F I N E S
*****************************************/
#define TASTER1   PORTBbits.RB0 //Taster1 liegt an Port B RB0
#define TASTER2   PORTCbits.RC0 //Taster2 liegt an Port C RC0
#define LED       LATAbits.LATA4//LED liegt an Port A RA4

/** H A U P T P R O G R A M M
*********************************/
void main(void)
{
   //Port A
   LATA = 0x00;
   TRISA = 0x00;    //Alle Pins von Port A sind Ausgänge
   ANSELA = 0x00;   //Alle Pins von Port A sind digitale I/Os
   //Port B
   LATB = 0x00;
   TRISB = 0xFF;    //Alle Pins von Port B sind Eingänge
   ANSELB = 0x00;   //Alle Pins von Port A sind digitale I/Os
   //Port C
   LATC = 0x00;
   TRISC = 0xFF;    //Alle Pins von Port C sind Eingänge
   ANSELC = 0x30;   //Nur RC4 und RC5 sind analoge Eingänge

   while(1) {              //Hauptschleife
      //Prüfen, ob beide Taster gedrückt sind
      if( (TASTER1==0) && (TASTER2==0) ) {
         LED = 1;          //Beide Taster wurden gedrückt
                           //LED einschalten
      } else {
         LED = 0;          //LED ausschalten
      }
   }
}
```

In der if-Abfrage wird über den Operator == geprüft, ob der Eingang einen Low-Pegel hat. Die beiden Bedingungen werden über den Operator `&&` miteinander verknüpft. Der Ausdruck ist nur wahr, wenn beide Taster gedrückt sind.

10.2 ODER-Gatter

Mit der ODER-Verknüpfung wird geprüft, ob einer der beiden Taster gedrückt ist. In diesem Fall wird die LED eingeschaltet. Die Leuchtdiode bleibt nur ausgeschaltet, wenn kein Taster gedrückt ist.

TASTER1	*TASTER2*	*LED*
Nicht gedrückt	Nicht gedrückt	Aus
Nicht gedrückt	Gedrückt	Ein
Gedrückt	Nicht gedrückt	Ein
Gedrückt	Gedrückt	Ein

Ähnlich wie auch beim UND-Gatter wird in der if-Anweisung geprüft, ob einer der beiden Taster gedrückt wurde.

```
/** D E F I N E S
*****************************************/
#define TASTER1   PORTBbits.RB0 //Taster1 liegt an Port B RB0
#define TASTER2   PORTCbits.RC0 //Taster2 liegt an Port C RC0
#define LED       LATAbits.LATA4//LED liegt an Port A RA4

/** H A U P T P R O G R A M M
*********************************/
void main(void)
{
   //Port A
   LATA = 0x00;
   TRISA = 0x00;    //Alle Pins von Port A sind Ausgänge
   ANSELA = 0x00;   //Alle Pins von Port A sind digitale I/Os
   //Port B
   LATB = 0x00;
   TRISB = 0xFF;    //Alle Pins von Port B sind Eingänge
   ANSELB = 0x00;   //Alle Pins von Port B sind digitale I/Os
   //Port C
   LATC = 0x00;
   TRISC = 0xFF;    //Alle Pins von Port C sind Eingänge
   ANSELC = 0x30;   //Nur RC4 und RC5 sind analoge Eingänge

   while(1) {              //Hauptschleife
```

```
      //Prüfen, ob einer oder beide Taster gedrückt sind
      if( (TASTER1==0) || (TASTER2==0) ) {
         LED = 1;          //Min. ein Taster wurde gedrückt
                           //LED einschalten
      } else {
         LED = 0;          //LED ausschalten
      }
   }
}
```

Um eine ODER-Verknüpfung zu realisieren, wird der Operator || verwendet. Man muss beachten, dass der Operator aus zwei senkrechten Strichen besteht und nicht aus einem, wie bei der ODER-Verknüpfung von zwei Variablen, da dies einer bitweisen Verknüpfung entspricht.

10.3 NAND-Gatter

Das NAND-Gatter schaltet die LED immer ein, außer es werden beide Taster gleichzeitig gedrückt. Die NAND-Verknüpfung ist das gleiche wie die UND-Verknüpfung mit einem nachgeschalteten Inverter.

TASTER1	*TASTER2*	*LED*
Nicht gedrückt	Nicht gedrückt	Ein
Nicht gedrückt	Gedrückt	Ein
Gedrückt	Nicht gedrückt	Ein
Gedrückt	Gedrückt	Aus

```
/** D E F I N E S
*****************************************/
#define TASTER1   PORTBbits.RB0 //Taster1 liegt an Port B RB0
#define TASTER2   PORTCbits.RC0 //Taster2 liegt an Port C RC0
#define LED       LATAbits.LATA4//LED liegt an Port A RA4

/** H A U P T P R O G R A M M
*********************************/
void main(void)
{
```

```
   //Port A
   LATA = 0x00;
   TRISA = 0x00;    //Alle Pins von Port A sind Ausgänge
   ANSELA = 0x00;   //Alle Pins von Port A sind digitale I/Os
   //Port B
   LATB = 0x00;
   TRISB = 0xFF;    //Alle Pins von Port B sind Eingänge
   ANSELB = 0x00;   //Alle Pins von Port B sind digitale I/Os
   //Port C
   LATC = 0x00;
   TRISC = 0xFF;    //Alle Pins von Port C sind Eingänge
   ANSELC = 0x30;   //Nur RC4 und RC5 sind analoge Eingänge

   while(1) {              //Hauptschleife
      //Prüfen, ob beide Taster gedrückt sind
      if( !((TASTER1==0) && (TASTER2==0)) ) {
         LED = 1;          //LED einschalten
      } else {
         LED = 0;          //Beide Taster wurden gedrückt
                           //LED ausschalten
      }
   }
}
```

Man erkennt, dass der Ausdruck in der if-Anweisung fast identisch mit der UND-Verknüpfung ist. Der Ausdruck wurde noch zusätzlich in Klammern gesetzt und ein Ausrufezeichen »!« vorangestellt. Durch dieses Ausrufezeichen wird die Negation des Ausdrucks erreicht.

10.4 NOR-Gatter

Die NOR-Verknüpfung ist eine invertierte ODER-Verknüpfung. Dabei wird die LED nur eingeschaltet, wenn beide Taster nicht gedrückt werden. In allen anderen Fällen bleibt die LED ausgeschaltet.

TASTER1	*TASTER2*	*LED*
Nicht gedrückt	Nicht gedrückt	Ein
Nicht gedrückt	Gedrückt	Aus
Gedrückt	Nicht gedrückt	Aus
Gedrückt	Gedrückt	Aus

```
/** D E F I N E S
*****************************************/
#define TASTER1   PORTBbits.RB0 //Taster1 liegt an Port B RB0
#define TASTER2   PORTCbits.RC0 //Taster2 liegt an Port C RC0
#define LED       LATAbits.LATA4//LED liegt an Port A RA4

/** H A U P T P R O G R A M M
*********************************/
void main(void)
{
   //Port A
   LATA = 0x00;
   TRISA = 0x00;    //Alle Pins von Port A sind Ausgänge
   ANSELA = 0x00;   //Alle Pins von Port A sind digitale I/Os
   //Port B
   LATB = 0x00;
   TRISB = 0xFF;    //Alle Pins von Port B sind Eingänge
   ANSELB = 0x00;   //Alle Pins von Port B sind digitale I/Os
   //Port C
   LATC = 0x00;
   TRISC = 0xFF;    //Alle Pins von Port C sind Eingänge
   ANSELC = 0x30;   //Nur RC4 und RC5 sind analoge Eingänge

   while(1) {              //Hauptschleife
      //Prüfen, ob beide Taster nicht gedrückt sind
      if( !((TASTER1==0) || (TASTER2==0)) ) {
         LED = 1;          //LED einschalten
      } else {
         LED = 0;          //Mind. ein Taster wurde gedrückt
                           //LED ausschalten
      }
   }
}
```

Wie bei der ODER-Verknüpfung wird über den Operator || geprüft, ob einer der beiden Taster gedrückt wurde oder ob der Eingang auf einem Low-Pegel liegt. Diese Verknüpfung wird anschließend über das Ausrufezeichen »!« negiert und man erhält so die NOR-Verknüpfung.

10.5 XOR-Gatter

Die Abkürzung XOR bedeutet »exklusives ODER« (engl. eXclusiv OR). Ein solches Gatter liefert am Ausgang einen High-Pegel, wenn die Eingänge unterschiedlich sind – anders als beim ODER-Gatter, das auch einen High-Pegel liefert, wenn beide Eingänge High sind.

```
/** D E F I N E S
*****************************************/
#define TASTER1   PORTBbits.RB0 //Taster1 liegt an Port B RB0
#define TASTER2   PORTCbits.RC0 //Taster2 liegt an Port C RC0
#define LED       LATAbits.LATA4//LED liegt an Port A RA4

/** H A U P T P R O G R A M M
*********************************/
void main(void)
{
   //Port A
   LATA = 0x00;
   TRISA = 0x00;   //Alle Pins von Port A sind Ausgänge
   ANSELA = 0x00;  //Alle Pins von Port A sind digitale I/Os
   //Port B
   LATB = 0x00;
   TRISB = 0xFF;   //Alle Pins von Port B sind Eingänge
   ANSELB = 0x00;  //Alle Pins von Port B sind digitale I/Os
   //Port C
   LATC = 0x00;
   TRISC = 0xFF;   //Alle Pins von Port C sind Eingänge
   ANSELC = 0x30;  //Nur RC4 und RC5 sind analoge Eingänge

   while(1) {              //Hauptschleife
      //Prüfen, ob einer der Taster gedrückt ist
      if( (TASTER1==0) != (TASTER2==0) ) {
         LED = 1;          //Nur ein Taster wurde gedrückt
```

```
                          //LED einschalten
      } else {
         LED = 0;         //LED ausschalten
      }

   }
}
```

Der Operator für die exklusive ODER-Verknüpfung wird aus einem Ausrufezeichen, gefolgt von einem Gleichzeichen (!=) gebildet. Das Zeichen bedeutet ungleich (Negation von gleich) und prüft, ob nur einer der beiden Taster gedrückt wurde. Sind beide Taster gedrückt oder beide Taster nicht gedrückt, ist die Bedingung nicht wahr und die LED wird ausgeschaltet.

10.6 RS-Flip-Flop

Ein RS-Flip-Flop verfügt wie ein logisches Gatter über zwei Eingänge: einen zum Setzen des Ausgangs (S) und einen zum Rücksetzen des Ausgangs (R). Diese Eingänge können ebenfalls über die Taster simuliert werden. Mit einem Taster wird die LED eingeschaltet und mit dem anderen wieder ausgeschaltet.

```
/** D E F I N E S
*****************************************/
#define TASTER1   PORTBbits.RB0 //Taster1 liegt an Port B RB0
#define TASTER2   PORTCbits.RC0 //Taster2 liegt an Port C RC0
#define LED       LATAbits.LATA4//LED liegt an Port A RA4

/** H A U P T P R O G R A M M
*********************************/
void main(void)
{
   //Port A
   LATA = 0x00;
   TRISA = 0x00;   //Alle Pins von Port A sind Ausgänge
   ANSELA = 0x00;  //Alle Pins von Port A sind digitale I/Os
   //Port B
   LATB = 0x00;
```

```
   TRISB = 0xFF;   //Alle Pins von Port B sind Eingänge
   ANSELB = 0x00;  //Alle Pins von Port B sind digitale I/Os
   //Port C
   LATC = 0x00;
   TRISC = 0xFF;   //Alle Pins von Port C sind Eingänge
   ANSELC = 0x30;  //Nur RC4 und RC5 sind analoge Eingänge

   while(1) {              //Hauptschleife
      //Prüfen, welcher Taster gedrückt wurde
      if( TASTER1==0 ) {
         LED = 1;          //LED einschalten
      }
      if( TASTER2==0 ) {
         LED = 0;          //LED ausschalten
      }
   }
}
```

Die Taster werden im Beispielprogramm kurz hintereinander abgefragt. Wird der Taster 1 gedrückt und der Eingangspegel liegt somit auf Low, wird die LED eingeschaltet. Solange der Taster 2 nicht gedrückt wird, bleibt die LED eingeschaltet. Erst wenn der Taster 2 gedrückt wird, ist der Ausdruck der if-Anweisung wahr und die LED wird wieder ausgeschaltet. Werden beide Taster gleichzeitig gedrückt, führt dies zu einem undefinierten Verhalten und die LED wird sehr schnell ein- und ausgeschaltet. Um diesen Effekt zu vermeiden, kann man das Programm etwas umschreiben und dem Taster 1 für das Einschalten der LED eine Priorität geben.

```
   while(1) {              //Hauptschleife
      //Prüfen, welcher Taster gedrückt wurde
      if( TASTER1==0 ) {
         LED = 1;          //LED einschalten
      } else {
         if( TASTER2==0 ) {
            LED = 0;          //LED ausschalten
         }
      }
   }
```

Bei dieser kleinen Abwandlung wird der Eingang von Taster 2 nicht mehr geprüft, solange der Taster 1 gedrückt ist. Somit kann auch die LED nicht mehr ausgeschaltet werden. Wird Taster 1 losgelassen, springt das Programm immer in den else-Zweig und prüft, ob der Taster 2 gedrückt ist, um die LED auszuschalten.

18 Verarbeitung analoger Signale

Viele Mikrocontroller können nicht nur digitale, sondern auch analoge Signale auswerten. Diese stammen z. B. von Temperaturfühlern, die in Abhängigkeit von der Temperatur ein analoges Signal ausgeben. Häufig müssen die Signale noch verstärkt werden, damit sie vom Mikrocontroller verarbeitet werden können. Mit einem analogen Eingang kann z. B. auch die Batteriespannung überwacht und bei zu niedrigem Spannungspegel eine Warnung ausgegeben werden. Die analogen Eingänge können sehr gut für Signale verwendet werden, die sich langsam ändern (bis ca. 1–5 kHz). Schnellere Signale können auch möglich sein, allerdings hängt dann die Messfrequenz stark von der Verarbeitungsdauer ab. Sollen mit den gemessenen Werten noch Berechnungen durchgeführt werden, können auch nur 100 Hz oder weniger sinnvoll sein. Man erkennt an diesen Limitierungen, dass ein Mikrocontroller nicht für die Verarbeitung von Audiosignalen (20–20 kHz) geeignet ist. Für die Bestimmung der maximalen Messfrequenz ist daher nicht nur die Angabe der Wandlungszeit wichtig, sondern auch die Bestimmung der Verarbeitungszeit im Programm. Sollen z. B. verschiedene Werte im internen EEPROM aufgezeichnet werden, ist die maximale Messfrequenz 250 Hz, da das Speichern im internen EEPROM mindestens 4 ms in Anspruch nimmt. Da sich die Temperatur oder die Batteriespannung in der Regel nur sehr langsam ändert, ist es problemlos möglich, eine Messung im Sekunden- oder Minutentakt durchzuführen.

18.1 Analog-Digital-Wandler (ADC)

Ein analoges Signal kann nicht direkt in einem Register abgespeichert werden und muss daher gewandelt werden. Da ein analoges Signal aus unendlich vielen kleinen Abstufungen besteht, würde man im Mikrocontroller auch ein unendlich großes Register und unendlich viel

Rechenleistung benötigen. Das ist praktisch nicht realisierbar und auch wenig sinnvoll. Daher wird die analoge Spannung in kleine Stufen aufgeteilt. Die Bereiche zwischen diesen Stufen erhalten einen definierten digitalen Wert. Je feiner die Stufen sind, desto größer sind die Genauigkeit und auch die Auflösung. Die Auflösung eines AD-Wandlers wird in Bit angegeben. Beim PIC18F23K22 kann man analoge Signale mit einer Auflösung von 10 Bit wandeln. Dabei entsprechen 10 Bit gleich 2^{10} Stufen gleich 1.024 Werten. Es kann demnach eine Spannung in 1.024 kleine Bereiche eingeteilt werden, die dann einem digitalen Wert zugeordnet werden. Wie groß ein solcher Bereich ist, hängt von der Referenzspannung ab, auf die der digitale Wert bezogen wird. Beträgt der maximale Wert der analogen Spannung 5 V, würde dies, bei einer 10-Bit-Auflösung, dem digitalen Wert 0x3FF entsprechen. Die Größe einer Stufe entspricht dann ca. 4,88 mV (= 5 V /1.024). Das heißt, dass sich bei einer Erhöhung der analogen Spannung um 4,88 mV der digitale Wert um 1 Bit (1 LSB) ändert.

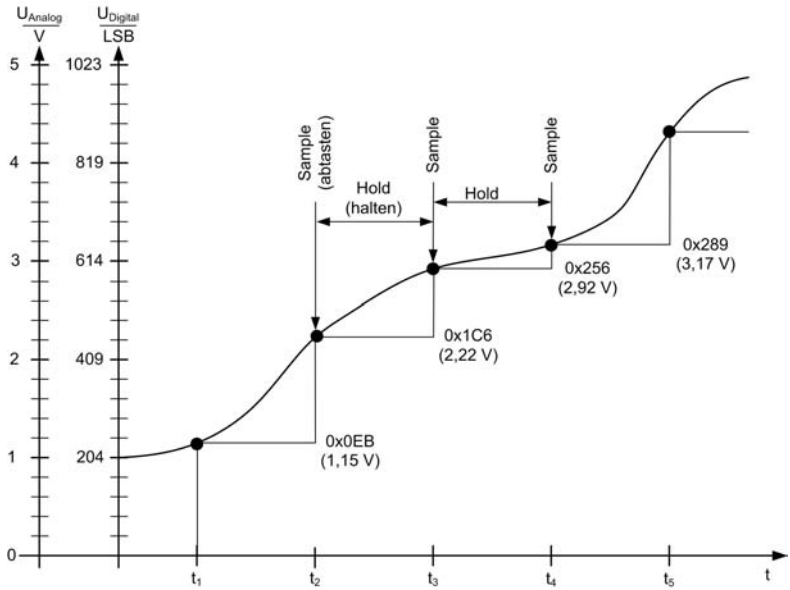


Bild 18.1: Abtastung eines analogen Signals

18.1.1 Berechnung des Spannungswerts

Der digitale Wert hat immer eine Ungenauigkeit von 1 LSB. Ist der analoge Wert genau 2 LSB groß, kann man nicht sicher sein, ob der digitale Wert noch 0x001 oder schon 0x002 ist, da genau hier die Umschaltschwelle liegt. Daher ist das niederwertigste Bit immer unsicher. Um den digitalen Wert zu berechnen, kann man die Formel 18.1 verwenden.

$$\text{Registerinhalt} = \left(\frac{U_{\text{Analog}}}{U_{\text{Ref}}} \cdot 1024 \right) - 0{,}5$$

Formel 18.1

Nach der Berechnung des Registerinhalts muss der Wert auf einen ganzzahligen Wert auf- oder abgerundet werden. Entsprechend kann der analoge Wert durch Umstellen von Formel 18.1 mit Formel 18.2 berechnet werden.

$$U_{\text{Analog}} = \frac{\text{Registerinhalt} + 0{,}5}{1024} \cdot U_{\text{Ref}}$$

Formel 18.2

Beispiel:

Im Beispiel wird angenommen, dass eine analoge Spannung von 0,8 V am Eingang anliegt und die Referenzspannung gleich der Betriebsspannung (5 V) ist.

$$\text{Registerinhalt} = \left(\frac{0{,}8\,\text{V}}{5\,\text{V}} \cdot 1024 \right) - 0{,}5 = 163{,}34 = 163$$

$$U_{\text{Analog}} = \frac{163 + 0{,}5}{1024} \cdot 5\,\text{V} = 0{,}798\,\text{V}$$

Bei einem Registerinhalt von 163 lag die gemessene analoge Spannung zwischen 0,796 V und 0,801 V. Die berechnete analoge Spannung liegt mit 0,798 V genau in der Mitte der beiden Werte. Das Ergebnis ist demnach:

$$U_{\text{Analog}} = 0{,}798\,\text{V} \pm 0{,}5\,\text{LSB} = 0{,}798\,\text{V} \pm 2{,}44\,\text{mV}$$

Man sieht, dass der digitale Wert immer einen gewissen Unsicherheitsfaktor hat. Diese Toleranz muss man bei der Anzeige und bei den weiteren Berechnungen beachten. Einen ebenfalls großen Einfluss auf die Genauigkeit des digitalen Werts hat die Referenzspannung. Schwankt diese um 1%, ist auch das Ergebnis der AD-Wandlung um 1% falsch. Es ist sehr einfach, die Betriebsspannung als Referenzspannung zu verwenden. Allerdings sollte man bei einer analogen Messung darauf achten, dass die Betriebsspannung während der Abtastung nicht durch große Verbraucher belastet wird. Dann ist es möglich, dass die Betriebsspannung absinkt und dadurch eine geringere Referenzspannung anliegt.

18.1.2 Aufteilung des digitalisierten Werts

Der PIC-Mikrocontroller verfügt nur über 8-Bit-Register, der digitale Wert hat aber eine Breite von 10 Bit. Daher muss man den Wert in 2 Registern abspeichern (ADRESH und ADRESL). Es gibt nun zwei Möglichkeiten, den Wert in das 16-Bit breite Doppelregister zu speichern: links- oder rechtsbündig. Welche Variante gewählt wird, wird über das Bit ADFM im Register *ADCON2* festgelegt. In der Regel hängt die Wahl der Variante von der späteren Weiterverarbeitung ab. Soll z. B. die Batteriespannung gemessen werden und man kann auf die beiden unteren Bits verzichten, wird man den AD-Wert linksbündig abspeichern und verwendet nur das Register *ADRESH*. Soll allerdings eine Konstante zum AD-Wert addiert werden, ist die rechtsbündige Variante sinnvoller, da keine Schiebeoperationen mehr ausgeführt werden müssen. Wie man Bild 18.2 entnehmen kann, werden die nicht verwendeten Bits der Register *ADRESH* und *ADRESL* mit Nullen aufgefüllt.

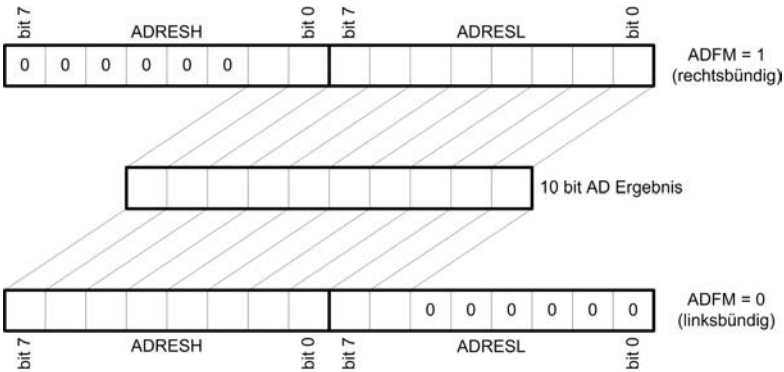


Bild 18.2: Aufteilung des digitalisierten Werts

18.2 Durchgangsprüfer

Wie die AD-Wandlung in der Praxis funktioniert, wird am Beispiel eines Durchgangsprüfers gezeigt. Ein Durchgangsprüfer ist ein stark vereinfachtes Ohmmeter und misst den elektrischen Widerstand. Bei einem Ohmmeter wird der exakte Widerstandswert angezeigt, wohingegen bei dem Durchgangsprüfer meist durch ein akustisches Signal niedriger Widerstandswert (z. B. <10 Ohm) signalisiert wird. Bei diesem Beispiel wird der Widerstandswert in 4 Stufen über 4 unterschiedliche LEDs angezeigt. Dadurch kann man eine grobe Abschätzung machen, in welchem Bereich der Widerstandswert liegt.

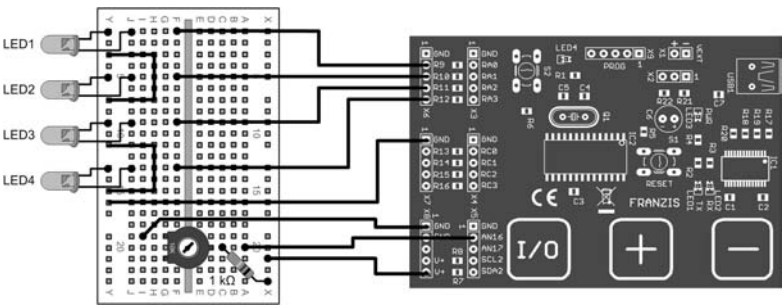


Bild 18.3: Anschluss für den Durchgangsprüfer

Für die Verdrahtung werden die folgenden Bauteile verwendet:

4 x LED rot

1 x Widerstand 1 kΩ (braun, schwarz, rot)

1 x Trimmpotenziometer 10 kΩ (braun, schwarz, orange)

8 x Verbindungsleitung ca. 15 cm

2 x Verbindungsleitung ca. 7 cm

Bild 18.4 zeigt den Schaltplan für den Durchgangsprüfer.

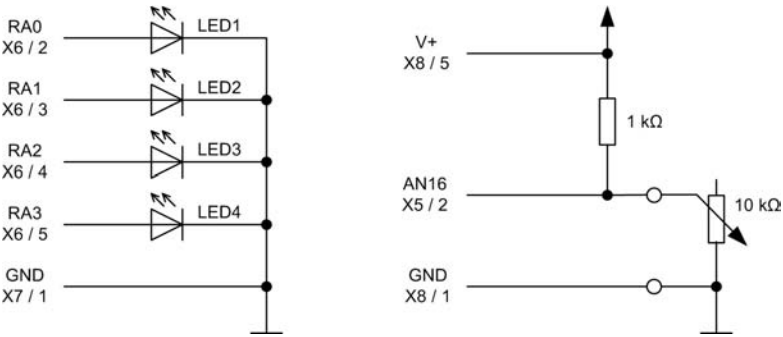


Bild 18.4: Schaltplan für den Durchgangsprüfer

Bei dem Aufbau der Schaltung sollte man darauf achten, dass der 1-kΩ-Widerstand richtig angeschlossen ist, da er den Stromfluss durch das Potenziometer begrenzt. Wird dieser Widerstand kurzgeschlossen, kann ein sehr hoher Strom fließen, wenn der Widerstand des Potis auf 0 Ω eingestellt wird.

Beispielprogramm: Durchgangsprüfer

Öffnet man dieses Beispiel in MPLAB, erkennt man, dass die Funktionen für die Auswertung der analogen Signale in einer eigenen Datei (adcLP.h) zusammengefasst sind. Dadurch wird die Wiederverwendung in anderen Programmen erleichtert. Es muss dann nur die Datei in das Projekt eingebunden werden und man kann die Funktionen in den Dateien verwenden.

```
//Initialisiert den AD-Wandler
void initADC( void ) {
   ADCON0 = 0b01000000;
   ADCON1 = 0;
   ADCON2 = 0b10100010;
   ADIF = 0;              //Bit für ADC Interrupt zurücksetzen
   ADIE = 0;              //ADC Interrupt ausschalten
   ADON = 1;              //AD-Wandler einschalten
   return;
}

//Prüft, ob die AD-Wandlung noch aktiv ist.
//Ist der AD-Wandler noch beschäftigt, wird eine 1
//zurückgegeben
//Wenn die Wandlung beendet ist, wird 0 zurückgegeben
char BusyADC(void) {
  return(ADCON0bits.GO);
}

//Schaltet den AD-Wandler wieder aus
void CloseADC(void) {
  ADCON0bits.ADON = 0;
  PIE1bits.ADIE = 0;
}

//Startet die Konvertierung
void ConvertADC(void) {
  ADCON0bits.GO = 1;
}

//Liest den gewandelten Wert aus
int ReadADC(void) {
  return (((unsigned int)ADRESH)<<8)|(ADRESL);
}

//Wählt den Kanal aus, der für die Wandlung
//verwendet werden soll
void SetChanADC(unsigned char channel)
{
  ADCON0 = ((channel << 2) & 0b00111100) |
           (ADCON0 & 0b11000011);
}
```

Die Funktion *initADC()* wird verwendet, um den AD-Wandler zu initialisieren und einzuschalten. Dazu werden über die ADCON-Register die Wandlungszeit und die Referenzspannung eingestellt. Anschließend werden noch der Interrupt aus- und der AD-Wandler eingeschaltet. Mit der Funktion *BusyADC()* kann geprüft werden, ob die AD-Wandlung beendet ist. *CloseADC()* beendet die Wandlung, indem der AD-Wandler ausgeschaltet wird. Um die Konvertierung zu starten, muss das Bit *GO* im Register *ADCON0* auf 1 gesetzt werden. Das wird durch die Funktion *ConvertADC()* erledigt. Liegt das Ergebnis vor, kann der Wert aus den Registern *ADRESH* und *ADRESL* über die Funktion *ReadADC()* ausgelesen werden. Da immer nur ein Kanal zur gleichen Zeit gemessen werden kann, wird mit der Funktion *SetChanADC()* der gewünschte Kanal ausgewählt, der für die nächste Wandlung verwendet werden soll.

```
/** D E F I N E S
*****************************************/
#define TASTER    PORTBbits.RB0  //Taster liegt an Port B RB0
#define LED       LATAbits.LATA4 //LED liegt an Port A RA4

//LEDs für die Anzeige des Widerstandswerts
#define LED_1     LATAbits.LATA0 //LED liegt an Port A RA0
#define LED_2     LATAbits.LATA1 //LED liegt an Port A RA1
#define LED_3     LATAbits.LATA2 //LED liegt an Port A RA2
#define LED_4     LATAbits.LATA3 //LED liegt an Port A RA3

//Schwellwerte für die Auswertung
#define R_10_OHM     10
#define R_100_OHM    93
#define R_1000_OHM   512

/** P R O T O T Y P E S
************************************/
void delay( int ms );

/** H A U P T P R O G R A M M
*********************************/
void main(void)
{
   int result=0;
```

```
//Port A
LATA = 0x00;
TRISA = 0x00;   //Alle Pins von Port A sind Ausgänge
ANSELA = 0x00;  //Alle Pins von Port A sind digitale I/Os
//Port B
LATB = 0x00;
TRISB = 0xFF;   //Alle Pins von Port B sind Eingänge
ANSELB = 0x00;  //Alle Pins von Port B sind digitale I/Os
//Port C
LATC = 0x00;
TRISC = 0xF0;   //RC0..RC3 = Ausgänge, RC4..RC7 = Eingänge
ANSELC = 0x30;  //Nur RC4 und RC5 sind analoge Eingänge

while(1) {              //Hauptschleife
   initADC();           //AD-Wandler initialisieren
   SetChanADC(16);      //Kanal einstellen, der für die
                        //AD-Wandlung verwendet werden soll
   ConvertADC();        //Startet die Wandlung
   while (BusyADC());   //Warten, bis Wandlung beendet ist
   result = ReadADC(); //Auslesen des AD-Werts
   CloseADC();          //AD-Wandler ausschalten

   //Auswerten des AD Ergebnisses
   if( (result >= 0) && (result < R_10_OHM) ) {
      //Wert ist kleiner als 10 Ohm
      LED_1 = 1;
      LED_2 = 0;
      LED_3 = 0;
      LED_4 = 0;
   } else if( (result>=R_10_OHM)&&(result<R_100_OHM) ) {
      //Wert liegt zwischen 10 und 100 Ohm
      LED_1 = 0;
      LED_2 = 1;
      LED_3 = 0;
      LED_4 = 0;
   } else if( (result>=R_100_OHM)&&(result<R_1000_OHM) ) {
      //Wert liegt zwischen 100 und 1000 Ohm
      LED_1 = 0;
      LED_2 = 0;
      LED_3 = 1;
```

```
         LED_4 = 0;
      } else {
         //Wert liegt über 1000 Ohm
         LED_1 = 0;
         LED_2 = 0;
         LED_3 = 0;
         LED_4 = 1;
      }
      delay(50);           //kurze Wartezeit

   }
}
```

Vor dem Beginn des Hauptprogramms werden über die Definitionen R_10_OHM, R_100_OHM und R_1000_OHM die Werte des AD-Wandlers festgelegt, bei denen der entsprechende Widerstand erreicht ist. Ein Widerstandswert von 100 Ω entspricht einem AD-Wert von 93.

In der Hauptschleife wird nach der Initialisierung des AD-Wandlers der gewünschte Kanal gewählt. In diesem Fall soll die Spannung von Kanal 16 (AN16) gemessen werden. Nun wird die Wandlung gestartet und mithilfe der Funktion *BusyADC()* wartet das Programm, bis der Wert vorliegt. Liegt der Wert vor, wird er mit der Funktion *ReadADC()* in die Variable *result* gespeichert. Im Anschluss wird geprüft, in welchem Bereich der Wert der Variablen *result* liegt. Ist der Wert kleiner als R_10_OHM (10) wird die LED 1 angeschaltet. Wird das Potenziometer etwas höher gestellt und der Wert von result steigt entsprechend, geht die LED 2 an und zeigt so, dass der Widerstandswert zwischen 10 Ω und 100 Ω liegt. Nachdem die Leuchtdiode eingeschaltet wurde, folgt eine kurze Zeitverzögerung, um die Messung etwas zu verlangsamen. Danach beginnt die Prozedur wieder von vorn.

Stichwortverzeichnis

7
7-Segment-Anzeige 40, 165

A
ADC 20
AD-Wandler 196
Arbeitsoberfläche 47
Array 84
Assembler 103
Auflösung 196

B
Baudrate 214
Bauteile 31
Bitfehler 212
Blockschaltbild 16
Bootloader 15, 71
Bootloaderbereich 74
Breakpoint 57

C
Compiler 41, 78
CTMU 20, 240

D
DAC 21
Datentypen 82
Debugger 51
Disassembler 55

E
EEPROM 19, 56, 229
Entwicklungsumgebung 41
Erweiterungsbuchsenleisten 26
EUSART 20

F
Farbcode 32
Flash 18
for-Schleife 90
Funktionen 95
Funktionsprototyp 96

G
Globale Variablen 98
Grundeinstellungen 59

H
Hauptprogramm 109
HI-TECH 43

I
I/O-Ports 21, 111
IDE 42
if-Anweisung 86
Interruptroutine 193
Interrupts 185

K
Konfigurationsbits 22
Kontrollstrukturen 86

L
Lauflicht 129
Leiterplatte 23
Leuchtdioden 35
Logikanalysator 61

M
Makros 81
Microchip 41
MPLAB 41
Multiplikationseinheit 19

O
Ohmmeter 199
Operatoren 85

P
Periodendauer 252
PIC 15
PIC18F23K22 23
Potentiometer 38
Prellen 159
Prescaler 151
Programmiergerät 63
Programmierung 25
Projekt 44
PWM 251

Q
Quarz 24

R
RAM 18
Referenzspannung 207
Register 104
Relais 37
RS232 29, 211

S
Schaltung 23
Schlafmodus 173
Schleifen 90
Schnittstellenkonverter 29
Schrittgeschwindigkeit 213
Sicherheitshinweise 29
Simulator 59
Speicher 13, 16, 18, 19, 63, 71, 75, 78, 230, 231, 237
Statemachine 89, 157
Steckbrett 31
Stimulus 60
switch-Anweisung 87

T
Taktgenerator 18
Taster 34
Tastgrad 252
Temperatursensor 39, 205
Texteditor 58
Timer 151
Touch 239
Transistor 37, 177

U
USART 211
USB-Schnittstelle 27

USB-Treiber 65

V
Variablen 82
Versorgungsspannung 27
Vorlage 49
Vorwärtsspannung 36

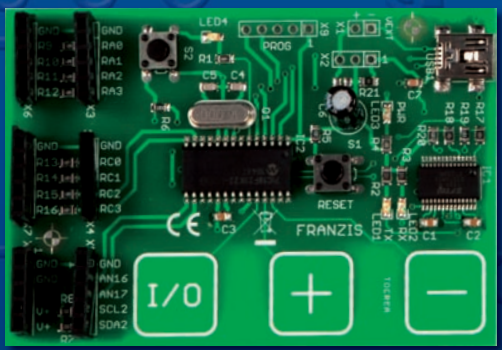
W
Watchdog 173
Watch-Fenster 55
while-Schleife 93
Widerstand 31

Z
Zahlenformate 83
Zeiger 99
Zufallszahl 142, 149
Zustandsmaschine 157

Lernpaket PIC®-Mikrocontroller

Die Platine ist so gestaltet, dass auch zusätzliche Pins auf die Buchsenleiste geführt sind und Sie dadurch ohne größeren Aufwand Ihre eigenen weiterführenden Experimente aufbauen können.

Anhand von verschiedenen Beispielen werden die einzelnen Funktionen des Mikrocontrollers angesteuert und erklärt wie die Register beschrieben werden müssen. Die Beispiele beginnen bei dem einfachen Schalten einer LED über einen Taster. Bei den weiteren Beispielen erfahren Sie wie ein Lauflicht auf unterschiedliche Weise realisiert werden kann und wie eine 7-Segment-Anzeige angesteuert wird.



Sie lernen wie analoge Spannungen gemessen werden und wie diese als Temperatur oder Widerstand interpretiert werden. Es wird erklärt, wie Daten in dem internen EEPROM gespeichert werden und anschließend über die USB-Schnittstelle an einen PC übertragen werden. Im Weiteren erfahren Sie wie der Mikrocontroller in den Schlafmodus versetzt werden kann und so kaum noch Energie verbraucht und wie Programmunterbrechungen mit Interrupts funktionieren. Zum Abschluss wird als kleines Highlight noch erklärt wie ein kapazitiver Touch funktioniert, den man von modernen Kochfeldern oder Nachttischlampen kennt.

Der Aufbau der Beispiele ist im beiliegenden Buch anhand von Bildern und Schaltplänen genau erklärt, und auf der CD finden Sie die Datenblätter der verwendeten Bauteile. Alle Beispiele sind getestet und können direkt mit einem Bootloadertool in den Mikrocontroller auf der Platine programmiert und ausprobiert werden.

Aus dem Inhalt:

- Mikrocontroller Grundlagen
- Hardwarefunktionen des PIC18F23K22
- Funktionalität der Platine und der Bauteile für die Experimente
- Einführung in die Programmiersprache C
- Entwicklungsumgebung MPLAB®
- Simulation mit MPLAB®
- Konfiguration des USB-Controllers FT232RL
- Tasterabfrage
- LED-Ansteuerung
- Hardwaretimer
- 7-Segment-Anzeige
- Tasterentprellung
- Temperaturmessung mit LM335
- Interrupts
- Watchdogtimer
- Analog-Digital-Wandler
- Kommunikation mit PC
- Internes EEPROM
- Kapazitiver Touchsensor
- Helligkeitssteuerung mit PWM