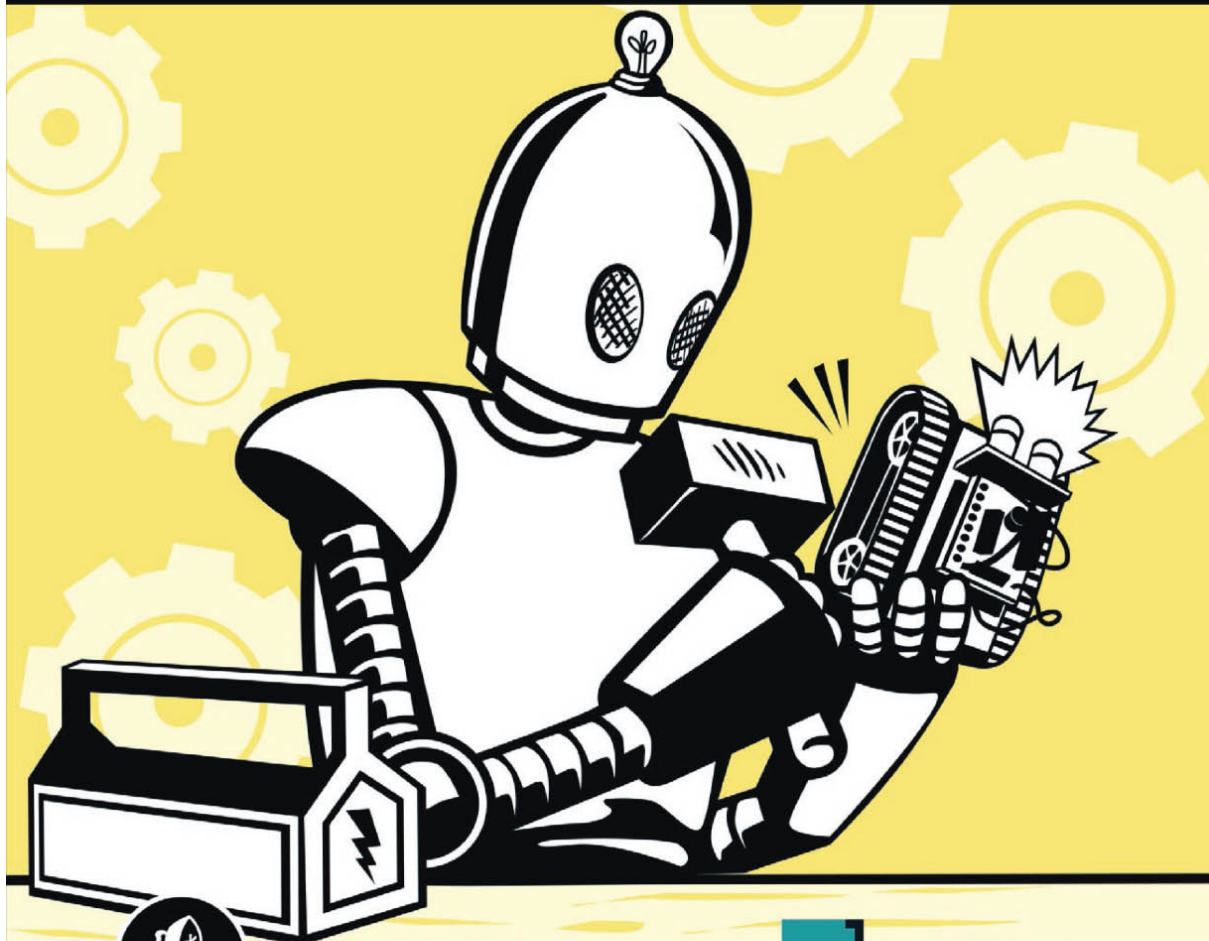



2. Auflage

Arduino-Workshops

Eine praktische Einführung mit 65 Projekten

John Boxall



 dpunkt.verlag

Inhalt

Cover

Über den Autor

Titel

Impressum

Widmung

Inhalt

Inhaltsverzeichnis

DANKSAGUNG

1 EINFÜHRUNG

Unendliche Möglichkeiten

Die Masse macht's

Teile und Zubehör

Benötigte Software

macOS

Windows 10

Ubuntu Linux

Sicherheit

Ausblick

2 EIN GENAUERER BLICK AUF DEN ARDUINO UND DIE IDE

Der Arduino

Die IDE

Der Befehlsbereich

Der Textbereich

Der Meldungsbereich

Ein erster Sketch in der IDE

Kommentare

Die Einrichtungsfunktion

Die Hardware steuern

Die Schleifenfunktion

Den Sketch überprüfen

Den Sketch hochladen und ausführen

Den Sketch bearbeiten

Ausblick

3 ERSTE SCHRITTE

Projekte planen

Elektrizität

Stromstärke

Spannung

Leistung

Elektronische Bauteile

Widerstände

Leuchtdioden (LEDs)

Steckplatinen

Projekt Nr. 1: LED-La-Ola

Der Algorithmus

Die Hardware

Der Schaltplan

Der Sketch

Den Sketch ausführen

Verwenden von Variablen

Projekt Nr. 2: Wiederholungen mit for-Schleifen

Die Helligkeit der LEDs durch Pulsbreitenmodulation ändern

Projekt Nr. 3: PBM-Beispiel

Weitere elektronische Bauteile

Transistoren

Gleichrichterioden

Relais

Schaltungen mit höherer Spannung

Ausblick

4 GRUNDBAUSTEINE

Schaltpläne

Symbole für die Bauteile

Leitungen in Schaltplänen

Schaltpläne analysieren

Kondensatoren

Die Kapazität von Kondensatoren

Kapazitätswerte ablesen

Arten von Kondensatoren

Digitale Eingänge

Projekt Nr. 4: Beispiel für digitale Eingänge

Der Algorithmus

Die Hardware

Der Schaltplan

Der Sketch

Den Sketch verstehen

Konstanten mit #define erstellen

Digitale Eingangspins messen

Entscheidungen mit if

Mehr Entscheidungsmöglichkeiten mit if-else

Boolesche Variablen

Logische Vergleichsoperatoren

Zwei und mehr Vergleiche

Projekt Nr. 5: Eine Verkehrsampel

Das Ziel

Der Algorithmus

Die Hardware

Der Schaltplan

Der Sketch

Den Sketch ausführen

Analoge und digitale Signale

Projekt Nr. 6: Ein Testgerät für Einzelzellenbatterien

Das Ziel

Der Algorithmus

Die Hardware

Der Schaltplan

Der Sketch

Rechnen mit dem Arduino

Fließkommavariablen

Vergleichsoperatoren für Berechnungen

Die Genauigkeit der Analogmessung durch eine Bezugsspannung verbessern

Externe Bezugsspannung

Interne Bezugsspannung

Regelbare Widerstände

Piezoelektrische Summer

Das Schaltplansymbol

Projekt Nr. 7: Einen Piezosummer ausprobieren

Projekt Nr. 8: Ein Thermometer mit Ampelanzeige

Das Ziel

Die Hardware

Der Schaltplan

Der Sketch

Ausblick

5 FUNKTIONEN

Projekt Nr. 9: Eine Funktion zur Wiederholung einer Aktion erstellen

Projekt Nr. 10: Eine Funktion mit einstellbarem Blinkvorgang erstellen

Funktionen zur Rückgabe von Werten

Projekt Nr. 11: Ein Thermometer mit Blinkcodeanzeige

Die Hardware

Der Schaltplan

Der Sketch

Daten vom Arduino im seriellen Monitor anzeigen

Der serielle Monitor

Projekt Nr. 12: Die Temperatur im seriellen Monitor anzeigen

Debugging im seriellen Monitor

Entscheidungen mit while-Anweisungen

while

do-while

Daten vom seriellen Monitor an den Arduino senden

Projekt Nr. 13: Eine Zahl mit 2 multiplizieren

long-Variablen

Projekt Nr. 14: long-Variablen verwenden

Ausblick

6 ZAHLEN, VARIABLEN UND ARITHMETISCHE OPERATIONEN

Zufallszahlen generieren

Zufallszahlen aus dem Umgebungsstrom generieren

Projekt Nr. 15: Einen elektronischen Würfel erstellen

Die Hardware

Der Schaltplan

Der Sketch

Den Sketch ändern

Schnellkurs in Binärzahlen

Binärzahlen

Bytevariablen

Erweitern der digitalen Ausgänge mit Schieberegistern

Projekt Nr. 16: Eine Binärzahlenanzeige aus LEDs bauen

Die Hardware

Der Schaltplan

Der Sketch

Projekt Nr. 17: Ein Binärzahlenquiz konstruieren

Der Algorithmus

Der Sketch

Arrays

Arrays definieren

Auf Werte in Arrays verweisen

In Arrays lesen und schreiben

Siebensegmentanzeigen

Die LEDs steuern

Projekt Nr. 18: Eine einstellige Anzeige konstruieren

Die Hardware

Der Schaltplan

Der Sketch

Zweistellige Zahlen anzeigen

Projekt Nr. 19: Zwei Siebensegmentanzeigen steuern

Die Hardware

Der Schaltplan

Modulo

Projekt Nr. 20: Ein Digitalthermometer konstruieren

Die Hardware

Der Sketch

Ausblick

7 DEN ARDUINO ERWEITERN

Shields

ProtoShield

Projekt 21: Ein individuelles Shield erstellen

Die Hardware

Der Schaltplan

Das Layout des ProtoShields

Das Design

Die Bauteile festlöten

Testen des ProtoShields

Sketche durch Bibliotheken erweitern

Eine Arduino-Bibliothek als ZIP-Datei herunterladen

Eine Arduino-Bibliothek mit dem Library Manager importieren

SD-Speicherkarten

Das Kartenmodul anschließen

Ihre SD-Karte testen

Projekt 22: Daten auf die Speicherkarte schreiben

Der Sketch

Projekt Nr. 23: Ein Gerät zur Temperaturlaufzeichnung konstruieren

Die Hardware

Der Sketch

Zeitmessung mit millis() und micros()

Projekt Nr. 31: Eine Stoppuhr konstruieren

Die Hardware

Der Schaltplan

Der Sketch

Interrupts

Interrupt-Modi

Interrupts einrichten

Interrupts aktivieren und deaktivieren

Projekt Nr. 25: Interrupts verwenden

Der Sketch

Ausblick

8 NUMERISCHE LED-ANZEIGEN UND MATRIZEN

Numerische LED-Anzeigen

Die Bibliothek installieren

Projekt Nr. 26: Digitale Stoppuhr

Projekt Nr. 27: LED-Matrix-Module verwenden

Installieren der Bibliothek

Die Display-Schriftart bearbeiten

Ausblick

9 FLÜSSIGKRISTALLANZEIGEN

LCD-Module

Ein LCD in einem Sketch verwenden

Text anzeigen

Variablen und Zahlen anzeigen

Projekt Nr. 28: Eigene Zeichen definieren

LCD-Grafikmodule

Das LCD-Grafikmodul anschließen

Das LCD verwenden

Die Anzeige steuern

Projekt Nr. 29: Textfunktionen in Aktion

Der Sketch

Den Sketch ausführen

Komplexere Anzeigeeffekte mit Grafikfunktionen erstellen

Projekt Nr. 30: Die Grafikfunktionen in Aktion

Der Sketch

Projekt Nr. 31: Eine Temperaturverlaufskurve aufzeichnen

Der Algorithmus

Die Hardware

Der Sketch

Den Sketch ausführen

Den Sketch ändern

Ausblick

10 IHRE EIGENEN ARDUINO-BIBLIOTHEKEN ERSTELLEN

Ihre erste Arduino-Bibliothek erstellen

Aufbau einer Arduino-Bibliothek

Die Header-Datei

Die Quelldatei

Die Datei KEYWORDS.TXT

Ihre neue Arduino-Bibliothek installieren

Erstellen einer ZIP-Datei mit Windows 7 und höher

Erstellen einer ZIP-Datei mit macOS oder später

Ihre neue Bibliothek installieren

Eine Bibliothek erstellen, die Werte annimmt, um eine Funktion auszuführen

Eine Bibliothek erstellen, die Sensorwerte verarbeitet und anzeigt

Ausblick

11 NUMERISCHE TASTENFELDER

Numerische Tastenfelder verwenden

Ein Tastenfeld anschließen

Programme für das Tastenfeld schreiben

Den Sketch testen

Entscheidungen mit switch-case

Projekt Nr. 32: Ein Schloss mit Tastenfeld konstruieren

Der Sketch

Funktionsweise

Den Sketch testen

Ausblick

12 BENUTZEREINGABEN ÜBER EINEN TOUCHSCREEN

Touchscreens

Den Touchscreen anschließen

Projekt Nr. 33: Bereiche auf dem Touchscreen ansprechen

Die Hardware

Der Sketch

Den Sketch testen

Den Touchscreen kalibrieren

Projekt Nr. 34: Einen Ein/Aus-Schalter mit zwei Zonen entwerfen

Der Sketch

Funktionsweise

Den Sketch testen

Die Funktion map()

Projekt Nr. 35: Einen Schalter mit drei Berührungszonen entwerfen

Die Touchscreen-Karte

Der Sketch

Den Sketch verstehen

Ausblick

13 GESTATTEN, FAMILIE ARDUINO!

Projekt Nr. 36: Einen eigenen Steckplatinen-Arduino bauen

Die Hardware

Der Schaltplan

Den Sketch ausführen

Die zahlreichen verschiedenen Arduino-Platinen

Arduino Uno

Freetronics Eleven

Der Adafruit Pro Trinket

Der Arduino Nano

Der Arduino LilyPad

Der Arduino Mega 2560

Der Freetronics EtherMega

Der Arduino Due

Ausblick

14 MOTOREN UND BEWEGUNG

Kleine Bewegungen mithilfe von Stellmotoren

Stellmotoren auswählen

Einen Stellmotor anschließen

Den Stellmotor in Bewegung setzen

Projekt Nr. 37: Ein Zeigerthermometer bauen

Die Hardware

Der Schaltplan

Der Sketch

E-Motoren

Einen Motor auswählen

Der Darlington-Transistor TIP120

Projekt Nr. 38: Den Motor steuern

Die Hardware

Der Schaltplan

Der Sketch

Kleine Schrittmotoren

Projekt Nr. 39: Ein Roboterfahrzeug bauen und steuern

Die Hardware

Der Schaltplan

Das Motor-Shield anschließen

Der Sketch

Anschließen weiterer Hardware

Kollisionserkennung

Projekt Nr. 40: Kollisionen mithilfe eines Mikroschalters erkennen

Der Schaltplan

Der Sketch

Infrarotsensoren zur Abstandsmessung

Verkabelung

Den IR-Abstandssensor testen

Projekt Nr. 41: Kollisionen mithilfe eines IR-Abstandssensors verhindern

Der Sketch

Den Sketch modifizieren und weitere Sensoren anschließen

Ultraschallsensoren

Den Ultraschallsensor anschließen

Den Ultraschallsensor testen

Projekt Nr. 42: Kollisionen mithilfe eines Ultraschall-Abstandssensors verhindern

Der Sketch

Ausblick

15 GPS FÜR DEN ARDUINO

Was ist GPS?

Den GPS-Shield testen

Projekt Nr. 43: Einen einfachen GPS-Empfänger bauen

Die Hardware

Der Sketch

Den Sketch ausführen

Projekt Nr. 44: Eine genaue GPS-gestützte Uhr konstruieren

Die Hardware

Der Sketch

Projekt Nr. 45: Den Bewegungsverlauf eines Objekts aufzeichnen

Die Hardware

Der Sketch

Den Sketch ausführen

Ausblick

16 DRAHTLOSE DATENÜBERTRAGUNG

Preiswerte Module für die drahtlose Datenübertragung

Projekt Nr. 46: Eine drahtlose Fernbedienung konstruieren

Die Hardware des Senders

Der Schaltplan des Senders

Die Hardware des Empfängers

Der Schaltplan des Empfängers

Der Sketch des Senders

Der Sketch des Empfängers

LoRa-Funkdatenmodule für größere Reichweite und höhere Geschwindigkeit

Projekt Nr. 47: Fernsteuern über LoRa Wireless

Die Hardware des Senders

Der Schaltplan des Senders

Die Hardware des Empfängers

Der Schaltplan des Empfängers

Der Sketch des Senders

Der Sketch des Empfängers

Projekt Nr. 48: Fernsteuern über LoRa mit Bestätigung

Die Hardware des Senders

Der Schaltplan des Senders

Der Sketch des Senders

Der Sketch des Empfängers

Projekt Nr. 49: Übertragung von Sensordaten über eine drahtlose LoRa-Verbindung

Die Hardware des Senders

Die Hardware des Empfängers

Der Schaltplan des Empfängers

Der Sketch des Senders

Der Sketch des Empfängers

Ausblick

17 INFRAROT-FERNBEDIENUNGEN

Was ist Infrarot?

Vorbereitung für die Infrarotübertragung

Der IR-Empfänger

Die Fernbedienung

Der Testsketch

Den Sketch testen

Projekt Nr. 50: Den Arduino fernsteuern

Die Hardware

Der Schaltplan

Der Sketch

Den Sketch erweitern

Projekt Nr. 51: Ein Raupenfahrzeug fernsteuern

Die Hardware

Der Sketch

Ausblick

18 RFID-TRANSPONDER LESEN

Das Innenleben von RFID-Transpondern

Die Hardware testen

Der Schaltplan

Die Schaltung prüfen

Der Test-Sketch

Anzeigen der ID-Nummer der RFID-Transponder

Projekt Nr. 52: Ein einfaches RFID-Steuerungssystem konstruieren

Der Sketch

Funktionsweise

Daten im eingebauten EEPROM des Arduino speichern

Lesen und Schreiben im EEPROM

Projekt Nr. 53: Ein RFID-Steuerungssystem konstruieren, das sich die letzte Aktion merkt

Der Sketch

Funktionsweise

Ausblick

19 DATENBUSSE

Der I2C-Bus

Projekt Nr. 54: Einen externen EEPROM verwenden

Die Hardware

Der Schaltplan

Der Sketch

Das Ergebnis

Projekt Nr. 55: Einen IC zur Porterweiterung verwenden

Die Hardware

Der Schaltplan

Der Sketch

Der SPI-Bus

Die Anschlüsse

Den SPI-Bus nutzen

Daten an ein SPI-Gerät senden

Projekt Nr. 56: Ein Digitalpotenziometer verwenden

Die Hardware

Der Schaltplan

Der Sketch

Ausblick

20 ECHTZEITUHREN

Der Anschluss des RTC-Moduls

Projekt Nr. 57: Datum und Uhrzeit mit einem RTC-Modul einstellen und anzeigen

Die Hardware

Der Sketch

Funktionsweise

Projekt Nr. 58: Eine einfache Digitaluhr bauen

Die Hardware

Der Sketch

Funktionsweise

Projekt Nr. 59: Eine Stechuhr konstruieren

Die Hardware

Der Sketch

Funktionsweise

Ausblick

21 DAS INTERNET

Erforderliches Material

Projekt Nr. 60: Eine Fernüberwachungsstation konstruieren

Die Hardware

Der Sketch

Fehlersuche

Funktionsweise

Projekt Nr. 61: Einen Arduino-Tweeter konstruieren

Die Hardware

Der Sketch

Den Arduino vom Web aus steuern

Projekt Nr. 62: Eine Fernsteuerung für den Arduino einrichten

Die Hardware

Der Sketch

Den Arduino fernsteuern

Ausblick

22 MOBILFUNKKOMMUNIKATION

Die Hardware

Hardware-Konfiguration und -Test

Projekt Nr. 63: Bau eines Arduino-Wählers

Die Hardware

Der Schaltplan

Der Sketch

Den Sketch verstehen

Projekt Nr. 64: Bau eines Arduino SMS-Senders

Der Sketch

Den Sketch verstehen

Projekt Nr. 65: Eine SMS-Fernbedienung bauen

Die Hardware

Der Schaltplan

Der Sketch

Den Sketch verstehen

Ausblick

INDEX

Grundbausteine

In diesem Kapitel lernen Sie Folgendes:

- Lesen von Schaltplänen, der »Sprache« von elektronischen Schaltungen
- Grundlagen von Kondensatoren
- Arbeiten mit Eingangspins
- Verwenden von arithmetischen Operationen und Testwerten
- Entscheidungen mithilfe von if-Anweisungen
- Unterschied zwischen analog und digital
- Messen von analogen Spannungsquellen mit unterschiedlicher Genauigkeit
- Grundlagen von regelbaren Widerständen, piezoelektrischen Summern und Temperatursensoren
- Anwenden des Gelernten durch die Konstruktion einer Ampel, eines Batterieprüfgeräts und eines Thermometers

Der Stoff in diesem Kapitel hilft Ihnen, das wahre Potenzial des Arduino zu erkennen. Sie lernen noch mehr über Elektronik – über weitere Bauteile, über das Lesen von Schaltplänen (den »Landkarten« von elektronischen Schaltungen) und die Art der Signale, die Sie messen können. Außerdem sehen wir uns weitere Funktionen des Arduino an: das Speichern von Werten, die Durchführung mathematischer Operationen, das Füllen von Entscheidungen. Schließlich wenden Sie das Gelernte in praktischen Projekten an.

Schaltpläne

In Kapitel 3 haben wir den Aufbau der Schaltungen anhand von fotografischen Plänen erklärt, auf denen die Steckplatine und die darauf montierten Bauteile zu

sehen waren. Das mag zwar wie die einfachste Möglichkeit aussehen, um eine Schaltung darzustellen, aber je mehr Komponenten Sie hinzufügen, umso unübersichtlicher wird dies. Da die Schaltungen, die wir im Folgenden aufbauen, schon etwas komplizierter sind, verwenden wir zu ihrer Darstellung ab jetzt *Schaltpläne* wie den aus Abbildung 4–1.

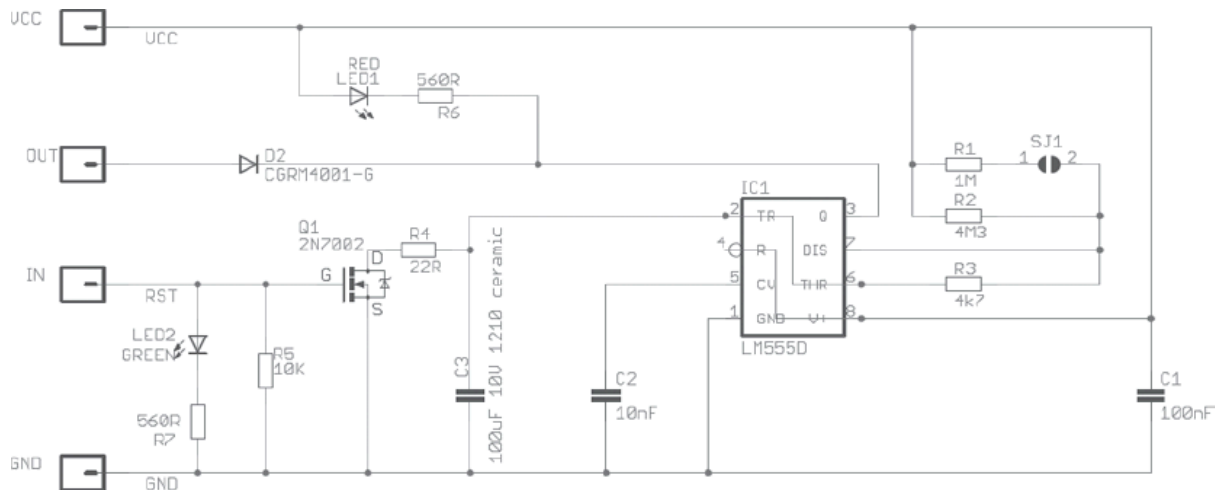


Abb. 4-4 Beispiel eines Schaltplans

Schaltpläne sind einfache »Landkarten« elektronischer Schaltungen, die den Weg zeigen, den der Strom durch die einzelnen Bauteile nimmt. Anstelle fotografischer Darstellungen der Elemente und Drähte werden Symbole und Linien verwendet.

Symbole für die Bauteile

Wenn Sie wissen, was die einzelnen Symbole bedeuten, ist das Lesen eines Schaltplans ganz einfach. Als Erstes wollen wir uns die Symbole für die Bauteile ansehen, die wir bereits verwendet haben.

Der Arduino

Abbildung 4–2 zeigt das Symbol für den Arduino. Darin sind alle Anschlüsse eingezeichnet und beschriftet.

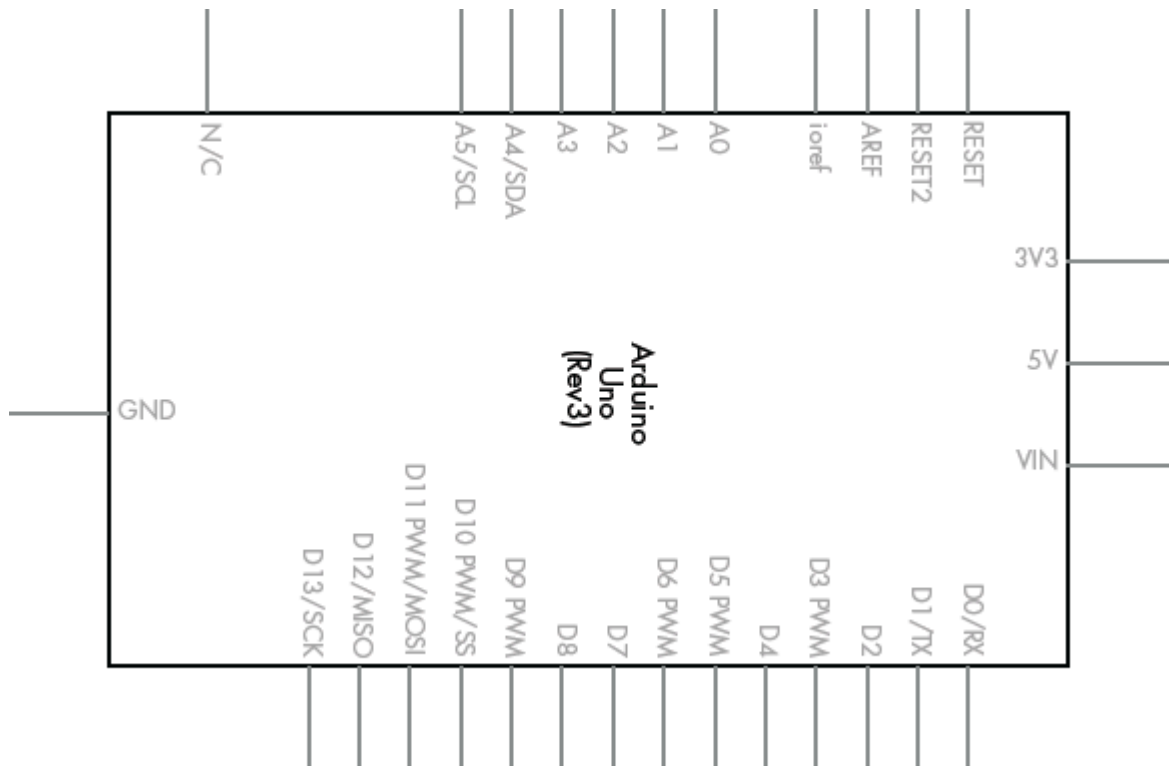


Abb. 4-4 Das Symbol für den Arduino Uno

Widerstände

Das Symbol für einen Widerstand sehen Sie in Abbildung 4-3.



Abb. 4-4 Das Symbol für einen Widerstand

Es ist sinnvoll, neben dem Symbol auch den Widerstandswert und die Bezeichnung des Bauteils anzugeben (in diesem Beispiel $220\ \Omega$ und R1). Das macht es leichter, dem Schaltplan zu folgen – sowohl für andere als auch für Sie selbst. Es kann vorkommen, dass Sie in Schaltplänen auch R statt Ω lesen, also beispielsweise eine Angabe wie 220 R.

Gleichrichterdioden

Das Symbol für eine Gleichrichterdiode sehen Sie in Abbildung 4-4.

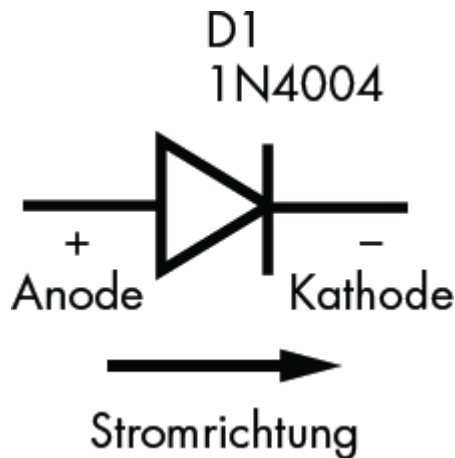


Abb. 4-4 Das Symbol für eine Gleichrichterdiode

Wie Sie in Kapitel 3 gelernt haben, sind Dioden gepolt: Der Strom fließt von der Anode zur Kathode. Auf dem Symbol in Abbildung 4-4 befindet sich die Anode links und die Kathode rechts. Das lässt sich leicht merken, indem Sie sich das Dreieck als eine Pfeilspitze vorstellen, die die Stromrichtung angibt. In die umgekehrte Richtung kann der Strom nicht fließen, da er von der senkrechten Linie »blockiert« wird.

Leuchtdioden

Das Symbol für eine LED sehen Sie in Abbildung 4-5.

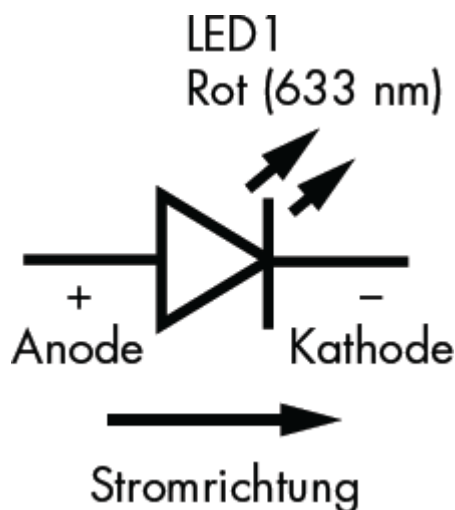


Abb. 4-4 Das Symbol für eine Leuchtdiode

Alle Arten von Dioden haben das gleiche Grundsymbol, das aus einem Dreieck und einer senkrechten Linie besteht. Bei LEDs sind jedoch zusätzlich zwei parallele Pfeile zu sehen, die von dem Dreieck wegzeigen. Damit wird das ausgestrahlte Licht symbolisiert.

Transistoren

Das Symbol für einen Transistor sehen Sie in Abbildung 4-6.

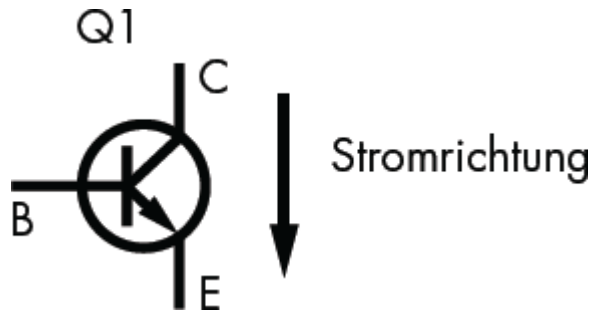


Abb. 4-4 Das Symbol für einen Transistor

Die mit *C* beschriftete senkrechte Linie oben am Symbol steht für den Kollektor, die waagerechte Linie auf der linken Seite (*B*) für die Basis und die untere Linie (*E*) für den Emitter. Da der Pfeil innerhalb des Symbols nach unten rechts zeigt, haben wir es hier mit einem *NPN-Transistor* zu tun, bei dem der Strom vom Kollektor zum Emitter fließt. (Bei *PNP-Transistoren* läuft der Strom vom Emitter zum Kollektor.)

Bei der Durchnummerierung von Transistoren verwenden wir den Buchstaben *Q* (wie das *R* für Widerstände).

Relais

Das Symbol für ein Relais sehen Sie in Abbildung 4-7.

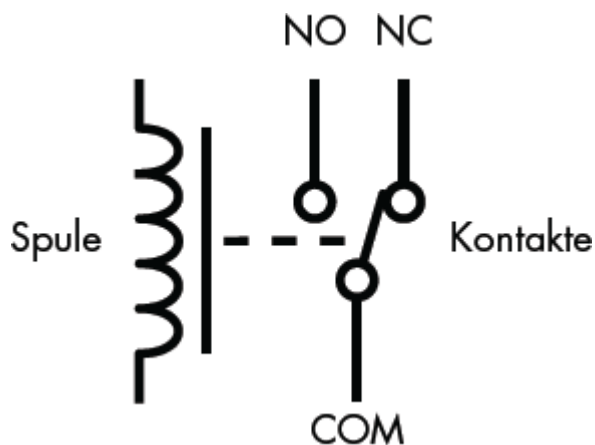


Abb. 4-4 Das Symbol für ein Relais

Relaisymbole gibt es in verschiedenen Formen und sie können auch mehr als einen Satz Kontakte aufweisen. Ihnen allen sind aber bestimmte Elemente gemeinsam, nämlich erstens die *Spule*, die als gewundene senkrechte Linie auf der linken Seite dargestellt wird, und zweitens die *Kontakte*. Der Kontakt *COM*

(für *common*, also »allgemein« oder »gemeinsam«) dient meistens als Eingang, die Kontakte *NO* (*normally open*, »im Normalzustand geöffnet«) und *NC* (*normally closed*, »im Normalzustand geschlossen«) als Ausgänge.

Das Relaisymbol wird stets im ausgeschalteten Zustand gezeigt, in dem die Spule nicht *bestromt* ist. Damit sind die Anschlüsse *COM* und *NC* miteinander verbunden. Bei bestromter Spule sind *COM* und *NO* verbunden.

Leitungen in Schaltplänen

Sich kreuzende oder miteinander verbundene Drähte werden in Schaltplänen auf eine ganz bestimmte Weise gezeichnet, wie die folgenden Beispiele zeigen.

Nicht verbundene kreuzende Leitungen

Wenn sich zwei Leitungen kreuzen, aber nicht miteinander verbunden sind, wird dies auf die beiden in Abbildung 4–8 dargestellten Weisen dargestellt. Keine davon ist falsch, es handelt sich nur um verschiedene Varianten.

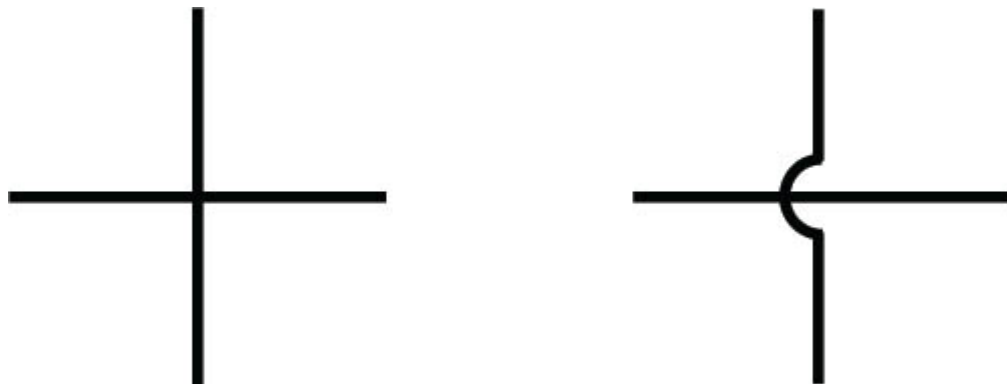


Abb. 4-4 Nicht verbundene kreuzende Leitungen

Verbundene Leitungen

Sind Leitungen physisch verbunden, wird der Kreuzungspunkt durch einen *Verbindungspunkt* gekennzeichnet, wie Sie in Abbildung 4–9 sehen.

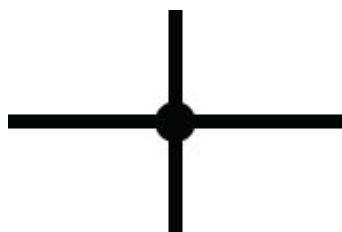


Abb. 4-4 Zwei miteinander verbundene Leitungen

Masseanschluss

Ist eine Leitung mit Erde oder Masse verbunden (*GND*), wird dies mit dem Symbol aus Abbildung 4–10 ausgedrückt.



Abb. 4–4 Das *GND*-Symbol

Wenn dieses Symbol in einem Schaltplan am Ende einer Linie steht, heißt dies, dass der Draht physisch an den *GND*-Pin des Arduino angeschlossen ist.

Schaltpläne analysieren

Nachdem Sie jetzt die Symbole für die verschiedenen Bauteile und Verbindungen kennen, wollen wir uns den Schaltplan für Projekt 1 ansehen. Dabei ging es darum, fünf LEDs nacheinander vorwärts und rückwärts aufleuchten zu lassen.

Wenn Sie den Schaltplan aus Abbildung 4–11 mit der Darstellung in Abbildung 3–13 vergleichen, werden Sie mir wahrscheinlich zustimmen, dass der Schaltplan die übersichtlichere Form der Darstellung bildet.

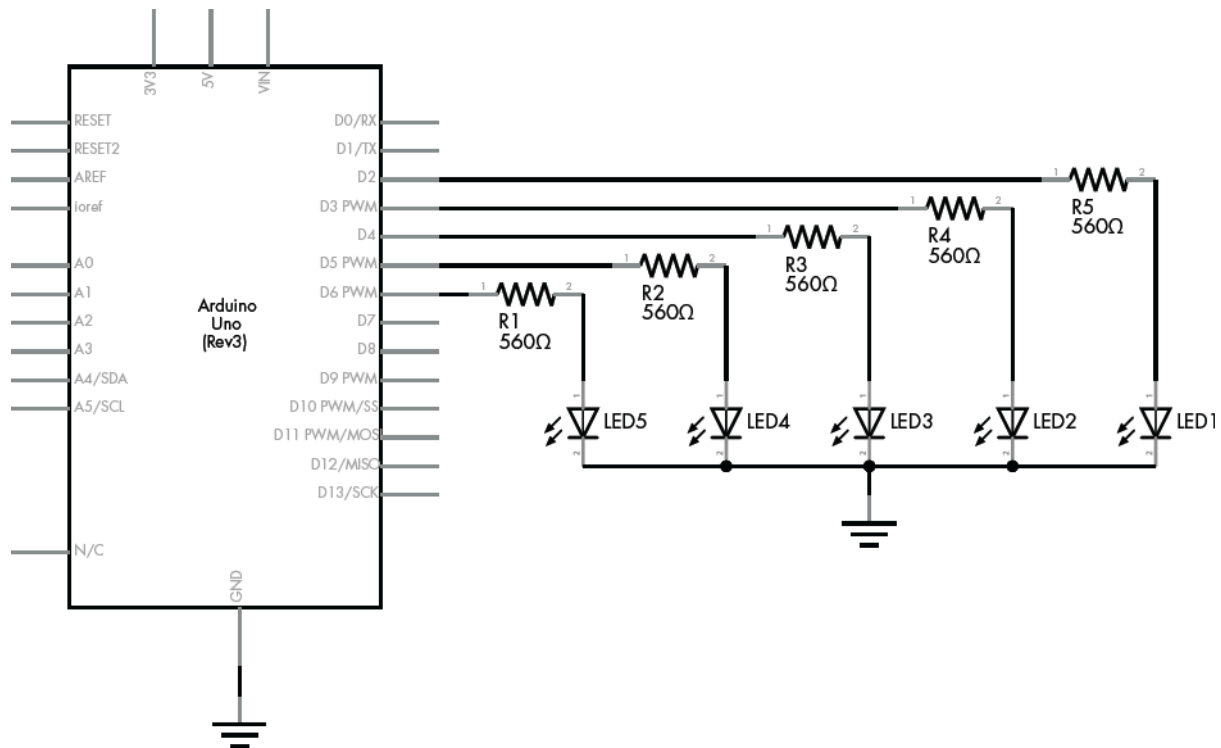


Abb. 4-4 Schaltplan für Projekt 1

Von jetzt an werden wir den Aufbau von Schaltungen anhand solcher Schaltpläne beschreiben. Bei der Einführung weiterer Bauteile lernen Sie auch gleich die Symbole dafür kennen.

HINWEIS

Wenn Sie Schaltpläne am Computer erstellen möchten, probieren Sie dazu das Programm *Fritzing* aus, das Sie kostenlos auf <http://www.fritzing.org/> erhalten.

Kondensatoren

Ein *Kondensator* ist ein Gerät, das eine elektrische Ladung speichert. Er besteht aus zwei Metallplatten mit einer Isolierschicht dazwischen, die es ermöglicht, dass sich zwischen den beiden Platten eine elektrische Ladung aufbaut. Fließt kein Strom mehr, verbleibt die Ladung dort und kann aus dem Kondensator austreten (man spricht hier vom *Entladen* des Kondensators), sobald ein neuer Weg eröffnet wird, den der Strom nehmen kann.

Die Kapazität von Kondensatoren

Die *Kapazität* eines Kondensators – also die Ladung, die er bei der anliegenden Spannung aufnehmen kann –, wird in *Farad* angegeben, wobei ein Farad (1 F)

schon eine sehr große Menge ist. Daher wird die Kapazität von Kondensatoren meistens in Pico- oder Mikrofarad ausgedrückt. Ein *Picofarad* (pF) entspricht 0,000000000001 F, ein *Mikrofarad* (μF) 0,000001 F. Bei der Fertigung werden Kondensatoren auf eine bestimmte Maximalspannung ausgelegt. In diesem Buch arbeiten wir ausschließlich mit Kondensatoren für geringe Spannung, die nicht mehr als 10 V verkräften können. Es ist jedoch kein Problem, in Schaltungen niedriger Spannungen auch Kondensatoren zu verwenden, die für höhere Spannungen gedacht sind. Übliche Werte sind 10, 16, 25 und 50 V.

Kapazitätswerte ablesen

Den aufgedruckten Wert eines Keramikkondensators abzulesen, erfordert ein wenig Übung, da die Angabe gewissermaßen in einem Code vorliegt. Die ersten beiden Stellen stehen für einen Wert in Picofarad, die dritte gibt den Multiplikator in Vielfachen von Zehn an. Der Kondensator in Abbildung 4–12 beispielsweise ist mit *104* beschriftet. Das steht für eine Zehn, gefolgt von vier Nullen, also 100.000 pF. Dieser Wert kann auch als 100 nF (Nanofarad) oder 0,1 μF (Mikrofarad) ausgedrückt werden.



Abb. 4–4 Ein Keramikkondensator mit einer Kapazität von 0,1 μF

HINWEIS

Die Umrechnung zwischen den verschiedenen dezimalen Vielfachen kann etwas verwirrend sein. Auf <http://www.justradios.com/uFnFpF.html> finden Sie eine hervorragende Umrechnungstabelle, die Sie sich ausdrucken können.

Arten von Kondensatoren

In unseren Projekten verwenden wir zwei Typen von Kondensatoren: Keramik- und Elektrolytkondensatoren.

Keramikkondensatoren

Keramikkondensatoren wie der aus Abbildung 4–12 sind sehr klein und können daher nur wenig Ladung aufnehmen. Sie sind nicht gepolt und lassen sich daher in beiden Richtungen verwenden. Das Schaltplansymbol für nicht gepolte Kondensatoren sehen Sie in Abbildung 4–13.

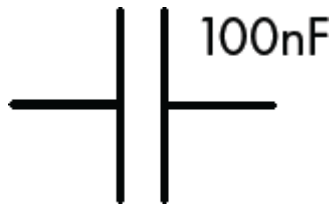


Abb. 4–4 Das Symbol für einen nicht gepolten Kondensator mit der Angabe der Kapazität oben rechts

Keramikkondensatoren eignen sich sehr gut für Hochfrequenzschaltungen, da sie aufgrund ihrer geringen Kapazität schnell geladen und entladen werden können.

Elektrolytkondensatoren

Elektrolytkondensatoren wie der aus Abbildung 4–14 haben eine größere Bauform als die Keramikvariante. Sie bieten eine höhere Kapazität, sind aber gepolt. Auf der Ummantelung ist entweder die positive oder die negative Seite markiert. In Abbildung 4–14 können Sie den Streifen mit dem Minussymbol erkennen, der die negative Seite kennzeichnet. Wie bei Widerständen müssen Sie auch bei den Werten von Kondensatoren mit einer gewissen Toleranz rechnen. Das Bauteil in Abbildung 4–14 hat eine Kapazität von 100 μF und eine Toleranz von 20 %.

Das Symbol für Elektrolytkondensatoren sehen Sie in Abbildung 4–15. Das Pluszeichen dient dazu, die Polung anzugeben.

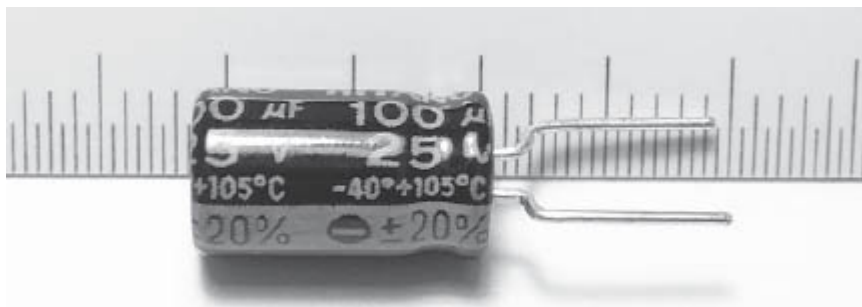


Abb. 4–4 Elektrolytkondensator

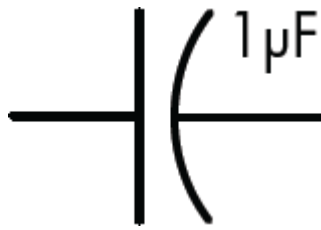


Abb. 4-4 Symbol für einen gepolten Kondensator

Elektrolytkondensatoren werden oft verwendet, um größere elektrische Ladungen zu speichern und die Spannung von Stromquellen auszugleichen. Wie kleine Kurzzeitbatterien können sie in der Nähe von Schaltkreisen oder Bauteilen, die in kurzer Zeit hohe Stromstärken von der Quelle beziehen, für eine gleichmäßige Versorgung und für Stabilität sorgen. Das verhindert unerwartete Aussetzer und Rauschen in Schaltungen. Dankenswerterweise sind die Kapazitätswerte von Elektrolytkondensatoren in Klartext auf der Ummantelung aufgedruckt und erfordern keine Dekodierung oder Deutung.

Am Beispiel der LEDs haben Sie schon den grundlegenden Umgang mit den Ausgängen des Arduino kennengelernt. Jetzt ist es an der Zeit, Signale von außen an den Arduino zu senden und die Messwerte an den digitalen Eingängen für Entscheidungen heranzuziehen.

Digitale Eingänge

In Kapitel 3 haben wir die digitalen E/A-Pins als Ausgänge genutzt, um LEDs ein- und auszuschalten. Dieselben Anschlüsse lassen sich aber auch verwenden, um Eingangssignale zu registrieren, beispielsweise um zu erkennen, ob eine Drucktaste betätigt wurde.

Wie die Ausgänge können auch die digitalen Eingänge zwei Zustände aufweisen, *high* und *low*. Die einfachste Form eines digitalen Eingangssignals liefern *Drucktasten* wie die in Abbildung 4-16. Sie können sie unmittelbar in die Steckplatine einsetzen. Wird eine solche Taste gedrückt, kann eine Spannung oder ein Strom durchgehen. An den digitalen Eingangspins können Sie dann messen, ob eine Spannung anliegt, und somit bestimmen, ob die Taste gedrückt wurde.

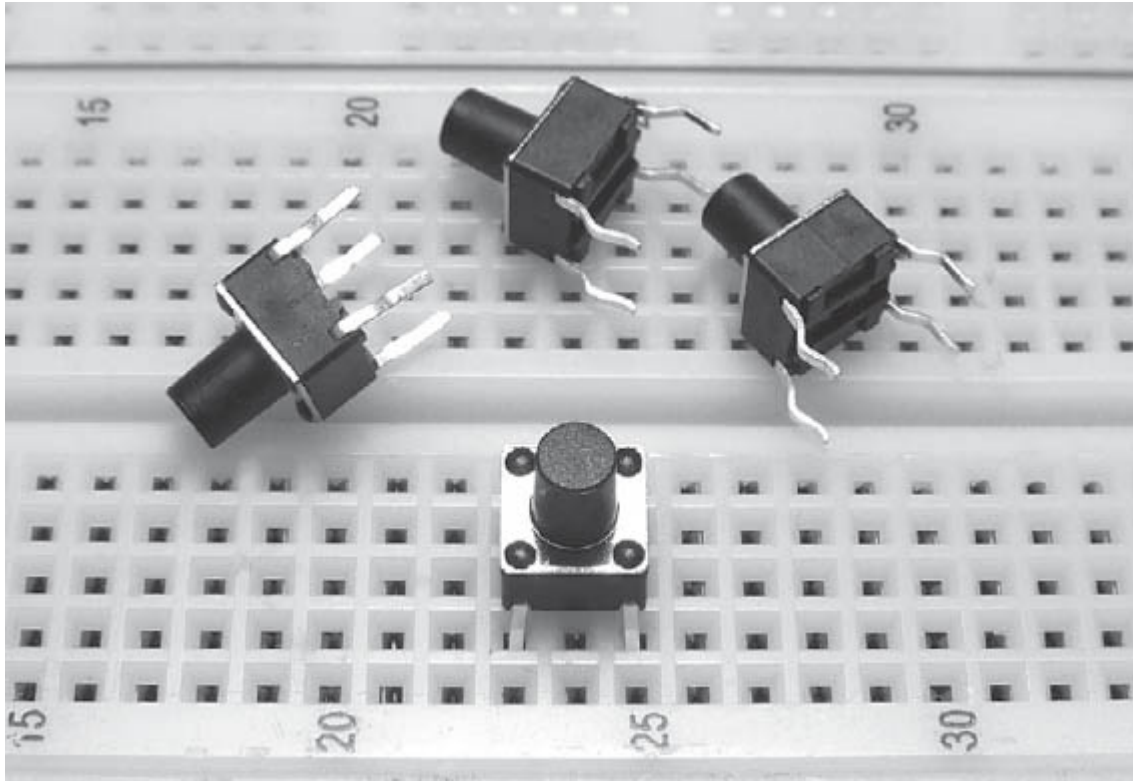


Abb. 4-4 Einfache Drucktasten auf einer Steckplatine

Beachten Sie, wie die Taste unten in der Abbildung in die Steckplatine eingeführt ist und dabei die Zeilen 23 und 25 überbrückt. Bei Betätigung der Taste werden die beiden Reihen verbunden. Das Schaltplansymbol für diese Art von Drucktaste sehen Sie in Abbildung 4-17. Es zeigt die beiden Seiten der Taste. Wird die Taste gedrückt, werden die beiden Hälften verbunden, sodass Spannung bzw. Strom durchgelassen wird. Durchnummeriert werden Tasten mit dem Präfix S.

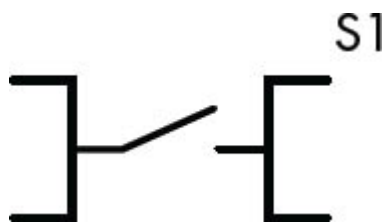


Abb. 4-4 Das Symbol für eine Drucktaste

KONTAKTPRELLEN MIT EINEM DIGITALEN SPEICHEROSZILLOSKOP SICHTBAR MACHEN

Drucktasten weisen ein besonderes Verhalten auf, das sogenannte Kontaktprellen oder kurz Prellen: Sie neigen dazu, mehrmals ein- und auszuschalten, wenn der Benutzer sie ein einziges Mal drückt. Die Metallkontakte im Inneren der Taste sind

so winzig, dass sie nach der Freigabe der Taste vibrieren, weshalb es zu den schnellen Schaltvorgängen kommt.

Das Kontaktprellen können Sie mit einem digitalen Speicheroszilloskop sichtbar machen. Dieses Gerät zeigt Spannungsänderungen im Verlauf der Zeit an. In Abbildung 4-18 sehen Sie das Phänomen des Kontaktprellens auf einem solchen Messinstrument.

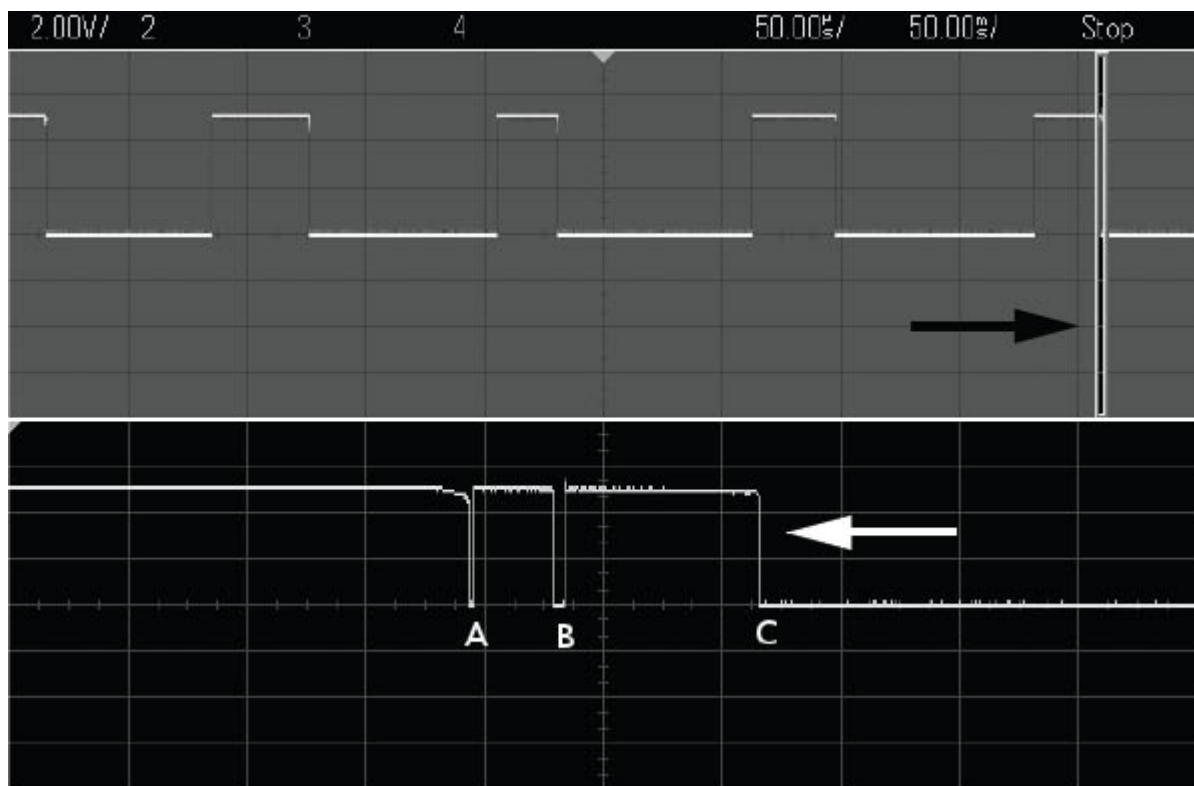


Abb. 4-4 Kontaktprellen messen

In der oberen Hälfte der Anzeige aus Abbildung 4-18 sehen Sie, was passiert, wenn die Taste mehrmals gedrückt wird. An den oberen Auslenkungen der Spannungskurve (bei 5 V) befindet sich die Taste im Zustand **Ein**, sodass die Spannung durchgeht. Unter dem Wort **Stop** in der Anzeige ist durch die beiden senkrechten Linien und den Pfeil die Stelle markiert, an der die Taste ausgeschaltet wurde. Der Verlauf der Spannung ist in der unteren Hälfte der Abbildung mit gestreckter Zeitachse dargestellt. An Punkt A lässt der Benutzer die Taste los, weshalb die Spannung auf 0 V sinkt. Aufgrund der Vibration der Kontakte steigt die Spannung wieder auf 5 V, bis Punkt B erreicht ist, wo der Kontakt zurückschwingt. Bei Punkt C kommen die Kontakte endlich zur Ruhe, sodass die Spannung auf 0 V fällt und endlich der Zustand **low** (also **Aus**) erreicht

ist. Anstatt also das Signal für eine Tastenbetätigung an den Arduino zu senden, haben wir in diesem Fall unbeabsichtigt gleich drei geschickt.

Projekt Nr. 4: Beispiel für digitale Eingänge

Bei diesem Projekt besteht das Ziel darin, dass eine LED bei Betätigung einer Taste eine halbe Sekunde lang aufleuchtet.

Der Algorithmus

Unser Algorithmus sieht wie folgt aus:

1. Prüfe, ob die Taste gedrückt wurde.
2. Wenn die Taste gedrückt wurde, schalte die LED eine halbe Sekunde lang ein und dann wieder aus.
3. Wenn die Taste nicht gedrückt wurde, tue gar nichts.
4. Wiederhole den Vorgang unendlich oft.

Die Hardware

Für dieses Projekt brauchen Sie folgende Teile:

- eine Drucktaste
- eine LED
- einen 560- Ω -Widerstand
- einen 10-k Ω -Widerstand
- einen 100-nF-Kondensator
- Verbindungsdrähte
- eine Steckplatine
- Arduino und USB-Kabel

Der Schaltplan

Als Erstes bauen wir auf der Steckplatine den Schaltkreis nach dem Schaltplan aus Abbildung 4–19 auf. Beachten Sie, wie der 10-k Ω -Widerstand zwischen *GND* und Digitalpin 7 angeordnet ist. Es handelt sich hierbei um einen *Pulldown*-Widerstand, da er die Spannung an dem Digitalpunkt auf fast 0 V senkt

(»herabzieht«). Durch den 100-nF-Kondensator, den der 10-k Ω -Widerstand überspannt, haben wir außerdem eine einfache *Entprellschaltung* gebaut, die das Kontaktprellen der Taste auszufiltern hilft. Wird die Taste gedrückt, geht das Signal am digitalen Pin sofort auf »high«. Beim Loslassen der Taste wird der Pin über den 10-k Ω -Widerstand auf *GND*-Niveau gesenkt, wobei der Kondensator für eine kleine Verzögerung sorgt. Da das Sinken der Spannung auf *GND*-Niveau verlangsamt wird, werden effektiv alle Prellpulse abgedeckt und dadurch die meisten Falschmessungen aufgrund schwankender Spannung und unvorhergesehenen Tastenverhaltens ausgeschlossen.

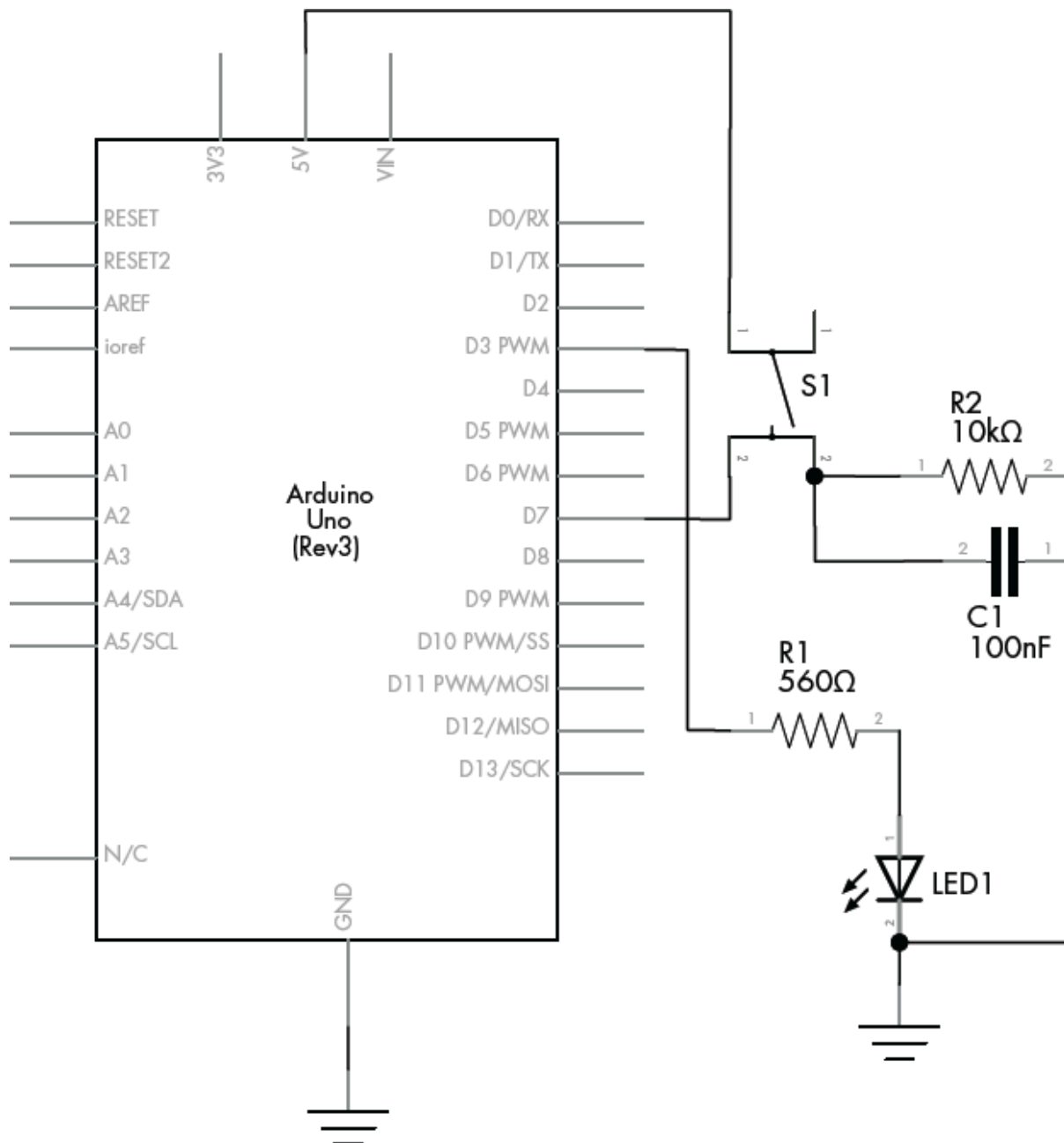


Abb. 4-4 Schaltplan für Projekt 4

Da dies das erste Mal ist, dass Sie eine Schaltung nach einem Schaltplan aufbauen, geben wir zusätzlich die folgende schrittweise Anleitung, mit deren Hilfe Sie genauer nachvollziehen können, wie die einzelnen Teile verbunden sind:

1. Bringen Sie die Drucktaste an der Steckplatine an, wie in Abbildung 4-20 gezeigt.

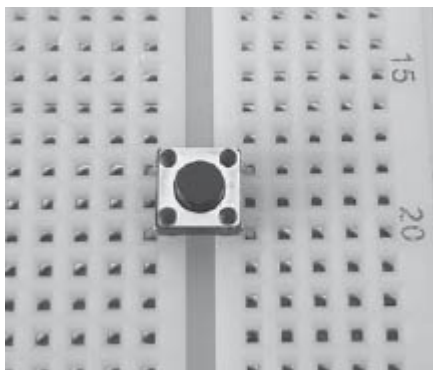


Abb. 4-4 Die Drucktaste auf der Steckplatine

2. Drehen Sie die Steckplatine um 90° gegen den Uhrzeigersinn und bauen Sie, wie in Abbildung 4-21 gezeigt, den 10-k Ω -Widerstand, einen kurzen Verbindungsdraht und den Kondensator ein.

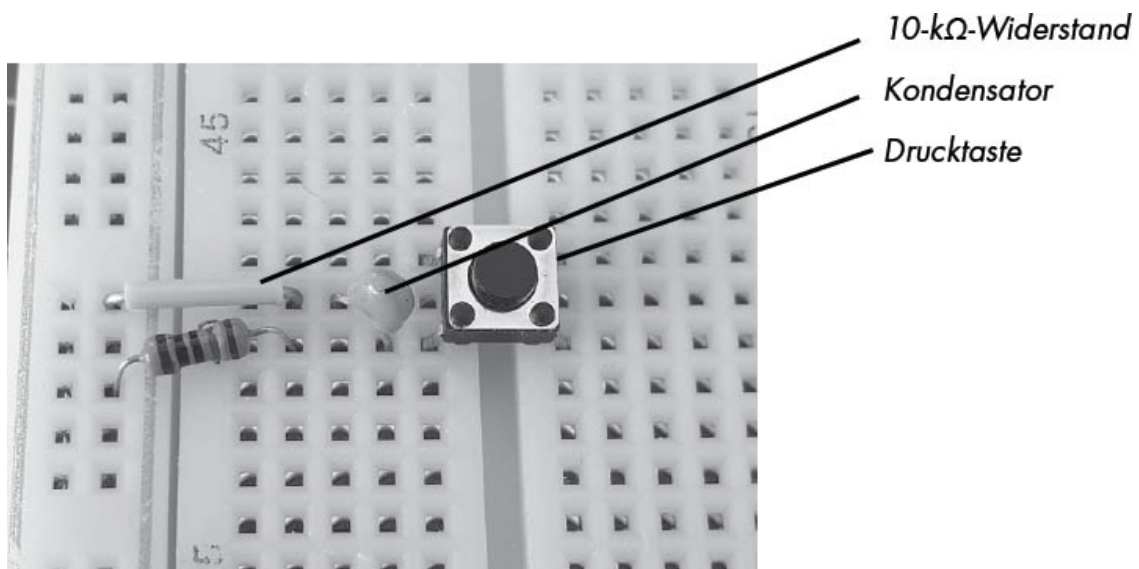


Abb. 4-4 10-k Ω -Widerstand, Kondensator und Drucktaste

3. Verbinden Sie das Kabel vom 5-Volt-Anschluss des Arduino mit der linken vertikalen Reihe der Steckplatine und das Kabel vom GND-Pin des Arduino mit der vertikalen Reihe rechts daneben (siehe Abb. 4-22).

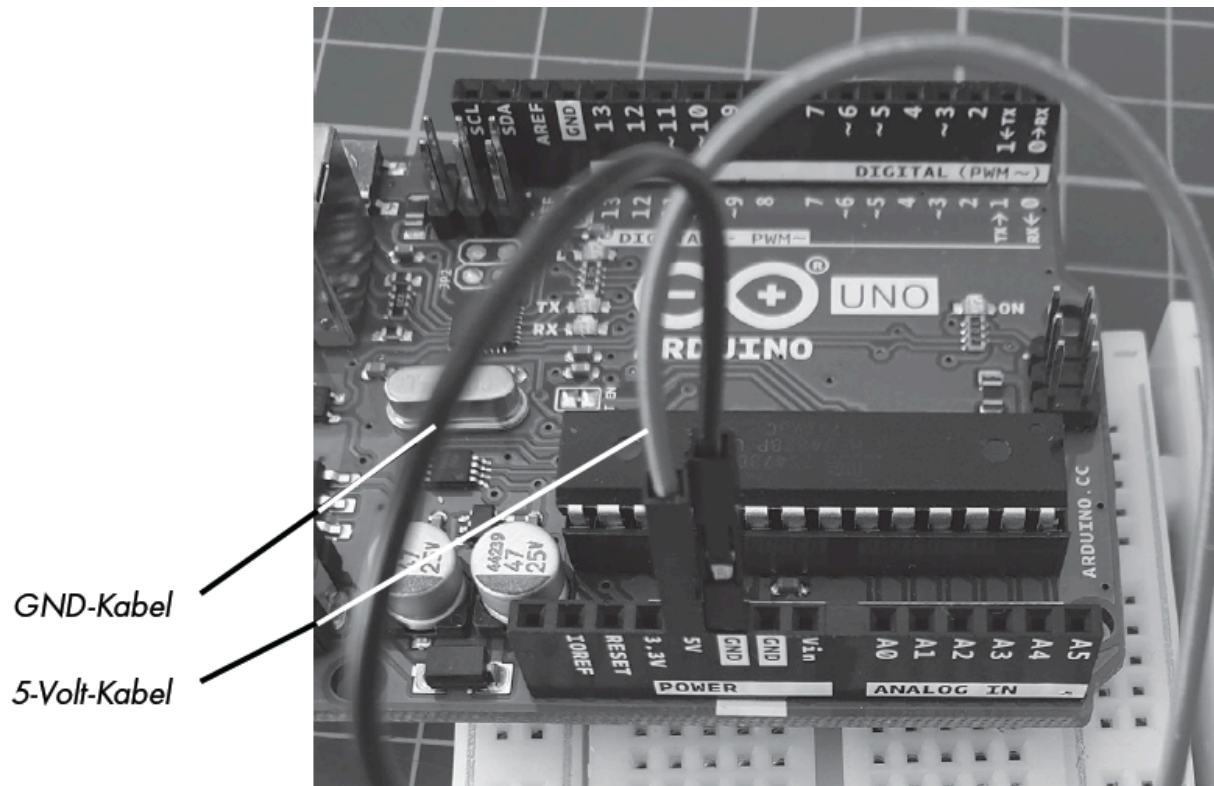


Abb. 4-4 Das 5-Volt-Kabel (rot) und das **GND**-Kabel (schwarz)

4. Führen Sie ein Kabel vom Digitalpin 7 des Arduino zur rechten oberen Ecke der Drucktaste auf der Steckplatine (siehe Abbildung 4-23).

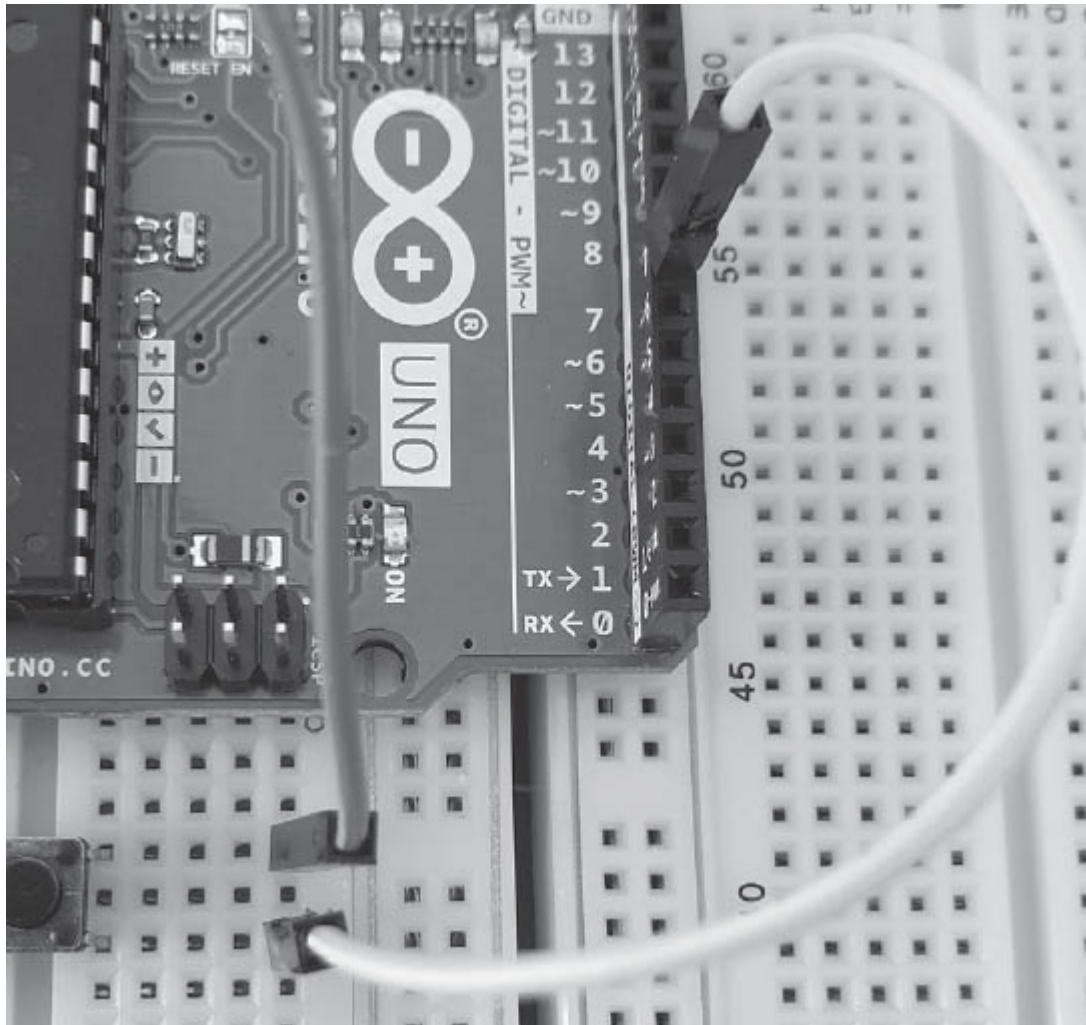


Abb. 4-4 Die Drucktaste mit dem digitalen Eingang verbinden

5. Stecken Sie die LED mit dem kurzen Bein (der Kathode) in die *GND*-Zeile der Platine und mit dem langen Bein (Anode) in eine Reihe rechts daneben. Bauen Sie anschließend den 560- Ω -Widerstand rechts neben der LED ein (siehe Abbildung 4-24).

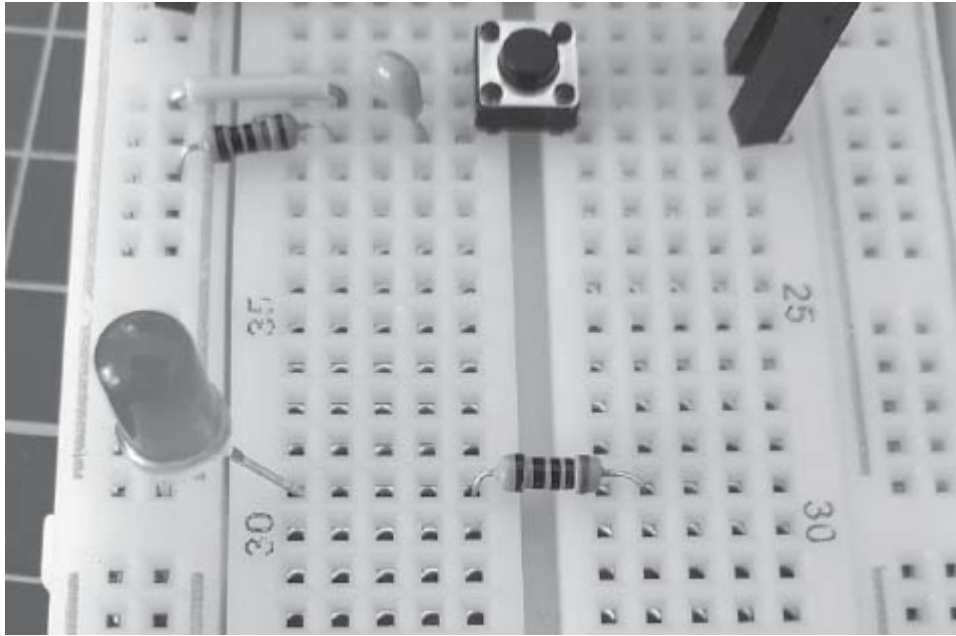


Abb. 4-4 Die LED und den 560-Ω-Widerstand einfügen

6. Führen Sie ein Kabel von der rechten Seite des 560-Ω-Widerstands zum Digitalpin 12 am Arduino (siehe Abb. 4-25).

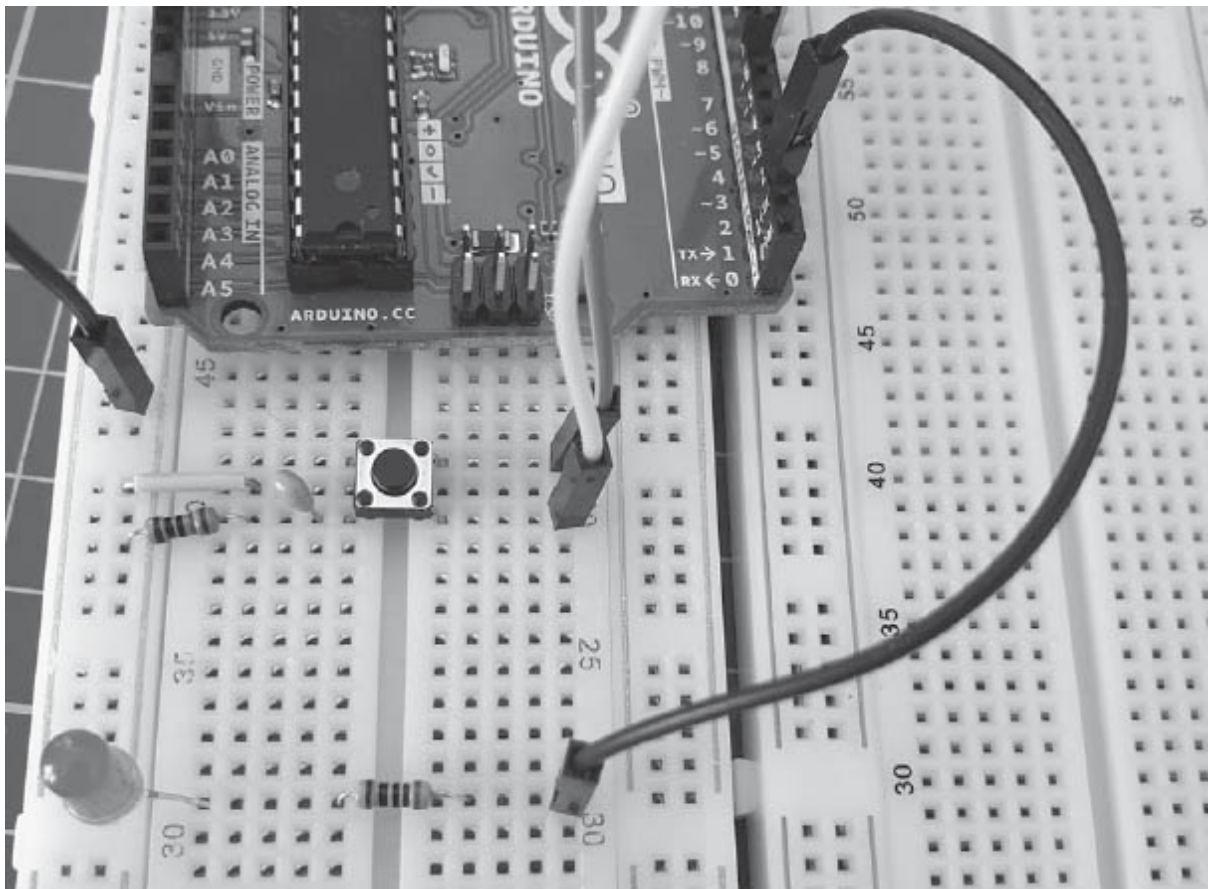


Abb. 4-4 Den LED-Zweig mit dem Arduino verbinden

Bevor Sie weitermachen, schauen Sie sich noch einmal den Schaltplan an und prüfen, ob Sie die Bauteile korrekt verdrahtet haben. Vergleichen Sie den Plan mit der Schaltung auf der Platine.

Der Sketch

Als Sketch geben Sie den Code von Listing 4–1 ein und laden ihn hoch:

```
// Projekt 4 - Beispiel für digitale Eingänge

#define LED    12    ❶

#define BUTTON 7

void setup()

{

    pinMode(LED, OUTPUT);    // Ausgang für die LED    ❷

    pinMode(BUTTON, INPUT); // Eingang für die Taste

}

void loop()

{

    if ( digitalRead(BUTTON) == HIGH )

    {

        digitalWrite(LED, HIGH); // Schaltet die LED ein

        delay(500);              // Wartet 0,5 s

    }

}
```

```
        digitalWrite(LED, LOW);    // Schaltet die LED aus
    }
}
```

Listing 4-1 *Digitale Eingänge*

Nachdem Sie den Sketch hochgeladen haben, tippen Sie kurz auf die Drucktaste. Die LED sollte jetzt eine halbe Sekunde lang aufleuchten.

Den Sketch verstehen

Als Nächstes sehen wir uns die neuen Elemente in dem Sketch von Projekt 4 detaillierter an – genauer gesagt, `#define`, die Befehle für die digitalen Eingangspins und die Funktion `if`.

Konstanten mit `#define` erstellen

Vor `void setup()` verwenden wir an der Stelle ❶ `#define`-Anweisungen, um feste Variablen zu erstellen. Wenn der Sketch kompiliert wird, ersetzt die IDE jedes Vorkommen des definierten Worts durch die darauf folgende Zahl. Beispielsweise erkennt die IDE bei ❷ `LED` und ersetzt dies durch die Zahl 3.

Im Grunde genommen verwenden wir den Befehl `#define` hier, um die Digitalpins für die LED und die Taste im Sketch mit Namen zu versehen. Beachten Sie, dass hinter der Wertangabe in `#define` kein Semikolon steht. Es ist eine gute Idee, Pinnummern und andere feste Werte (etwa eine zeitliche Verzögerung) auf diese Weise zu benennen, denn wenn sie in einem Sketch wiederholt vorkommen, müssen sie so bei einer Änderung nur an einer Stelle bearbeitet werden. In diesem Beispiel wird `LED` dreimal in dem Sketch verwendet. Um den Wert zu ändern, müssen wir lediglich einmal die Definition in der `#define`-Anweisung austauschen.

Digitale Eingangspins messen

Um den Status einer Drucktaste zu bestimmen, definieren wir zunächst in `void setup()` wie folgt einen digitalen E/A-Pin als Eingang:

```
pinMode(BUTTON, INPUT); // Eingang für die Taste
```

Um zu ermitteln, ob die Taste eine Spannung an den digitalen Eingang anlegt (ob sie also gedrückt ist), verwenden wir `digitalRead(Pin)`, wobei *Pin* für die Nummer des abzulesenden Digitalpins steht. Die Funktion gibt entweder HIGH (die Spannung am Pin liegt bei 5 V) oder LOW zurück (die Spannung am Pin liegt bei 0 V).

Entscheidungen mit if

Mit `if` können wir in unserem Sketch Entscheidungen fällen und den Arduino anweisen, je nachdem, wie die Entscheidung ausgefallen ist, unterschiedlichen Code auszuführen. Beispielsweise haben wir in Projekt 4 den Code aus Listing 4-2 verwendet:

```
if ( digitalRead(BUTTON) == HIGH )  
  
{  
  
    digitalWrite(LED, HIGH);    // Schaltet die LED ein  
  
    delay(500);                 // Wartet 0,5 s  
  
    digitalWrite(LED, LOW);    // Schaltet die LED aus  
  
}
```

Listing 4-2 Ein einfaches Beispiel für eine `if`-then-Anweisung

Die erste Zeile in diesem Code beginnt mit `if`, um eine Bedingung zu prüfen. Ist die Bedingung wahr (ist die Spannung also HIGH), so bedeutet dies, dass die Taste gedrückt wurde, weshalb der Code in den geschweiften Klammern ausgeführt wird.

Um zu bestimmen, ob die Taste gedrückt worden ist (ob also `digitalRead(BUTTON)` auf HIGH gesetzt ist), verwenden wir einen *Vergleichsoperator*, ein doppeltes Gleichheitszeichen (`==`). Wenn wir in dem

Sketch == durch != (nicht gleich) ersetzen, wird die LED beim Drücken der Taste abgeschaltet. Probieren Sie es aus!

HINWEIS

Ein häufig gemachter Fehler besteht darin, in einer Testanweisung nur ein einzelnes Gleichheitszeichen (=) zu verwenden, was aber »mach dies gleich« bedeutet. Benötigt wird aber ein doppeltes Gleichheitszeichen (==) für »prüfe, ob es gleich ist«. Möglicherweise erhalten Sie bei diesem Versäumnis keine Fehlermeldung, aber die if-Anweisung funktioniert dann trotzdem nicht.

Wenn Sie ein paar Erfolge verzeichnen können, versuchen Sie die Dauer, während der die LED angeschaltet ist, zu ändern. Andernfalls können Sie auch zurück zu Projekt Nr. 3 gehen und dort die Drucktastensteuerung implementieren. (Bitte nehmen Sie die Schaltung nicht auseinander, sie wird noch für das nächste Beispiel gebraucht.)

Mehr Entscheidungsmöglichkeiten mit if-else

Mit `else` können Sie einer `if`-Anweisung noch eine weitere Aktion hinzufügen. Wenn Sie den Code aus Listing 4-1 beispielsweise wie in Listing 4-3 mit zusätzlichem `else` umschreiben, wird die LED eingeschaltet, *wenn* (`if`) die Taste gedrückt wurde, *anderenfalls* (`else`) aber ausgeschaltet. Die Verwendung von `else` zwingt den Arduino dazu, einen anderen Codeausschnitt auszuführen, wenn die in der `if`-Anweisung geprüfte Bedingung nicht wahr ist.

```
// Listing 4-3

#define LED    3

#define BUTTON 7

void setup()

{

    pinMode(LED, OUTPUT);    // Ausgang für die LED
```

```

    pinMode(BUTTON, INPUT);    // Eingang für die Taste
}

void loop()

{

    if ( digitalRead(BUTTON) == HIGH )

    {

        digitalWrite(LED, HIGH);

    }

else

    {

        digitalWrite(LED, LOW);

    }

}

```

Listing 4-3 Ergänzung um `else`

Boolesche Variablen

Manchmal müssen Sie festhalten, ob sich irgendetwas in einem von zwei möglichen Zuständen befindet, etwa ein/aus oder heiß/kalt. Hierzu eignen sich *boolesche Variablen*, die legendären Computer-»Bits«, deren Wert nur 0 (falsch, `false`) oder 1 (wahr, `true`) sein kann. Daher sind boolesche Variablen so

nützlich: Sie können nur entweder wahr oder falsch sein. Wie alle anderen Variablen müssen wir sie zunächst deklarieren, bevor wir sie verwenden können:

```
boolean raining = true; // Erstellt die Variable "raining"
und
// setzt sie auf true
```

Innerhalb des Sketches können Sie den Zustand der booleschen Variable dann durch eine einfache Neuzuweisung ändern:

```
raining = false;
```

Die Verwendung von booleschen Variablen zur Entscheidungsfindung erfolgt ganz einfach mithilfe einer `if`-Teststruktur. Da boolesche Vergleiche entweder wahr oder falsch ergeben, können dafür wie im folgenden Beispiel die Vergleichsoperatoren herangezogen werden:

```
if ( raining == true )
{
    if ( summer != true )
    {
        // Es regnet, und es ist nicht Sommer
    }
}
```

Logische Vergleichsoperatoren

Um Entscheidungen über zwei oder mehr boolesche Variablen oder andere Zustände zu treffen, können wir verschiedene Operatoren heranziehen. Dazu gehören *not* (!), *and* (&&) und *or* (| |).

Not

Der Operator *not* wird durch ein Ausrufezeichen ausgedrückt und dient als Abkürzung, um zu prüfen, ob irgendetwas *nicht wahr* ist. Betrachten Sie dazu das folgende Beispiel:

```
if ( !raining )  
  
{  
  
    // Es regnet nicht (raining == false)  
  
}
```

And

Der logische Operator *and* wird durch && ausgedrückt. Er hilft die Anzahl einzelner if-Tests zu verringern. Betrachten Sie dazu das folgende Beispiel:

```
if (( raining == true ) && ( !summer ))  
  
{  
  
    // Es regnet und es ist nicht Sommer  
  
    // (raining == true and summer == false)  
  
}
```

Or

Der logische Operator *or* wird durch `||` ausgedrückt. Seine Verwendung ist ganz einfach. Betrachten Sie dazu das folgende Beispiel:

```
if (( raining == true ) || ( summer == true ))
{
    // Es regnet oder es ist Sommer
}
```

Zwei und mehr Vergleiche

Sie können auch zwei oder mehr Vergleiche in ein und derselben `if`-Anweisung unterbringen, wie das folgende Beispiel zeigt:

```
if ( snow == true && rain == true && !hot )
{
    // Es schneit, es regnet, und es ist nicht heiß
}
```

Die Operationsreihenfolge legen Sie mithilfe von Klammern fest. Im nächsten Beispiel wird zuerst der Vergleich in den Klammern verarbeitet und das Ergebnis, `true` oder `false`, dann mit dem Rest der `if-then`-Anweisung verglichen:

```
if (( snow == true || rain == true ) && hot == false))
{
```

```
    // Es regnet oder schneit, und es ist nicht heiß  
}
```

Ebenso wie in den Beispielen, in denen der *not*-Operator (!) unmittelbar vor einem Wert stand, können Sie auch einfache Tests auf `true` und `false` durchführen, ohne dabei jedes Mal ausdrücklich `== true` oder `== false` schreiben zu müssen. Der folgende Code funktioniert genauso wie das vorhergehende Beispiel:

```
if (( snow || rain ) && !hot )  
  
{  
  
    // Es regnet oder schneit, und es ist nicht heiß  
  
    // ( snow is true OR rain is true ) AND it is not hot  
  
}
```

Wie Sie sehen, kann der Arduino mithilfe von booleschen Variablen und Vergleichsoperatoren eine Menge von Entscheidungen treffen. Wenn Sie zu komplizierteren Projekten übergehen, wird sich diese Möglichkeit als äußerst nützlich erweisen.

Projekt Nr. 5: Eine Verkehrsampel

Wir wollen unsere neuen Kenntnisse jetzt zur Lösung eines hypothetischen Problems einsetzen. Stellen Sie sich vor, Sie sind Stadtplaner in einer ländlichen Gegend und haben ein Problem mit einer einspurigen Brücke über einen Fluss: Dort kommt es jede Woche zu ein oder zwei nächtlichen Unfällen, wenn übermüdete Fahrer über die Brücke rasen, ohne erst anzuhalten und nachzusehen, ob der Weg frei ist. Sie haben schon angeregt, Ampeln zu installieren, aber der Bürgermeister möchte erst eine Vorführung sehen, bevor er das Budget für die Anschaffung der Ampeln absegnet. Nun könnten Sie zwar eine mobile Ampelanlage leihen, aber das wäre zu teuer. Stattdessen bauen Sie

mithilfe von LEDs und einem Arduino ein Modell der Brücke mit funktionierenden Ampeln.

Das Ziel

Das Ziel besteht darin, an jedem Ende der einspurigen Brücke eine Dreifarben-Verkehrsampel aufzustellen, sodass der Verkehr immer nur in eine Richtung auf einmal fließen kann. Wenn Sensoren an einem Ende der Brücke erkennen, dass dort ein Auto an einer roten Ampel wartet, springen die Ampeln um, sodass der Verkehr in die entgegengesetzte Richtung fließen kann.

Der Algorithmus

Um die Fahrzeugsensoren an den beiden Enden der Brücke zu simulieren, verwenden wir zwei Drucktasten. Jede Ampel verfügt über eine rote, eine gelbe und eine grüne LED. Zu Anfang soll das System den Verkehr von Westen nach Osten durchlassen, sodass die westliche Ampel Grün zeigt und die östliche Rot.

Wenn sich ein Fahrzeug der Brücke nähert (was wir durch Drücken der Taste simulieren) und die Ampel rot ist, schaltet das System die Ampel auf der gegenüberliegenden Seite von Grün über Gelb auf Rot um und wartet dann einen zuvor festgelegten Zeitraum, damit Fahrzeuge, die sich schon auf der Brücke befinden, noch passieren können. Dann blinkt das gelbe Licht auf der Seite des wartenden Fahrzeugs, um dem Fahrer das Signal zu geben, sich fertig zu machen, und schließlich schaltet die Ampel auf Grün. Sie bleibt grün, bis sich ein Fahrzeug der anderen Seite nähert und sich der Vorgang wiederholt.

Die Hardware

Für dieses Projekt brauchen Sie folgende Teile:

- zwei rote LEDs (LED1 und LED2)
- zwei gelbe LEDs (LED3 und LED4)
- zwei grüne LEDs (LED5 und LED6)
- sechs 560- Ω -Widerstände (R1 bis R6)
- zwei 10-k Ω -Widerstände (R7 und R8)
- zwei 100-nF-Kondensatoren (C1 und C2)
- zwei Drucktasten (S1 und S2)

- eine mittelgroße Steckplatine
- Verbindungsdrähte
- Arduino und USB-Kabel

Der Schaltplan

Da wir nur sechs LEDs steuern und Eingangssignale von zwei Drucktasten empfangen, ist der Aufbau nicht allzu kompliziert. Abbildung 4–26 zeigt den Schaltplan für dieses Projekt.

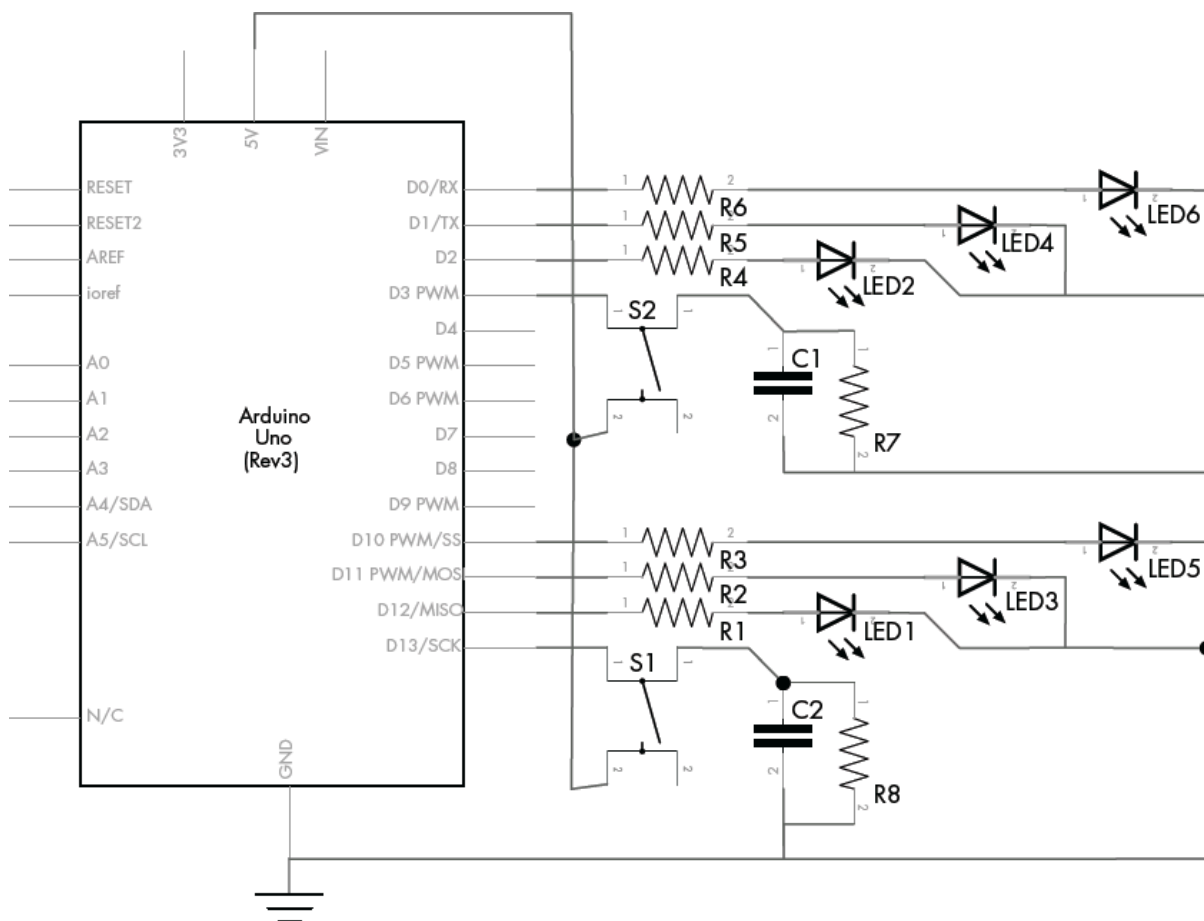


Abb. 4–4 Schaltplan für Projekt 5

Im Grunde genommen handelt es sich bei dieser Schaltung um eine erweiterte Version des Drucktasten- und LED-Aufbaus von Projekt 4 mit mehr Widerständen, mehr LEDs und einer weiteren Taste.

Achten Sie darauf, die LEDs in der richtigen Richtung einzubauen: Die Anoden müssen mit den Widerständen verbunden sein, die Kathoden mit dem *GND*-Pin des Arduino (siehe Abb. 4–27).

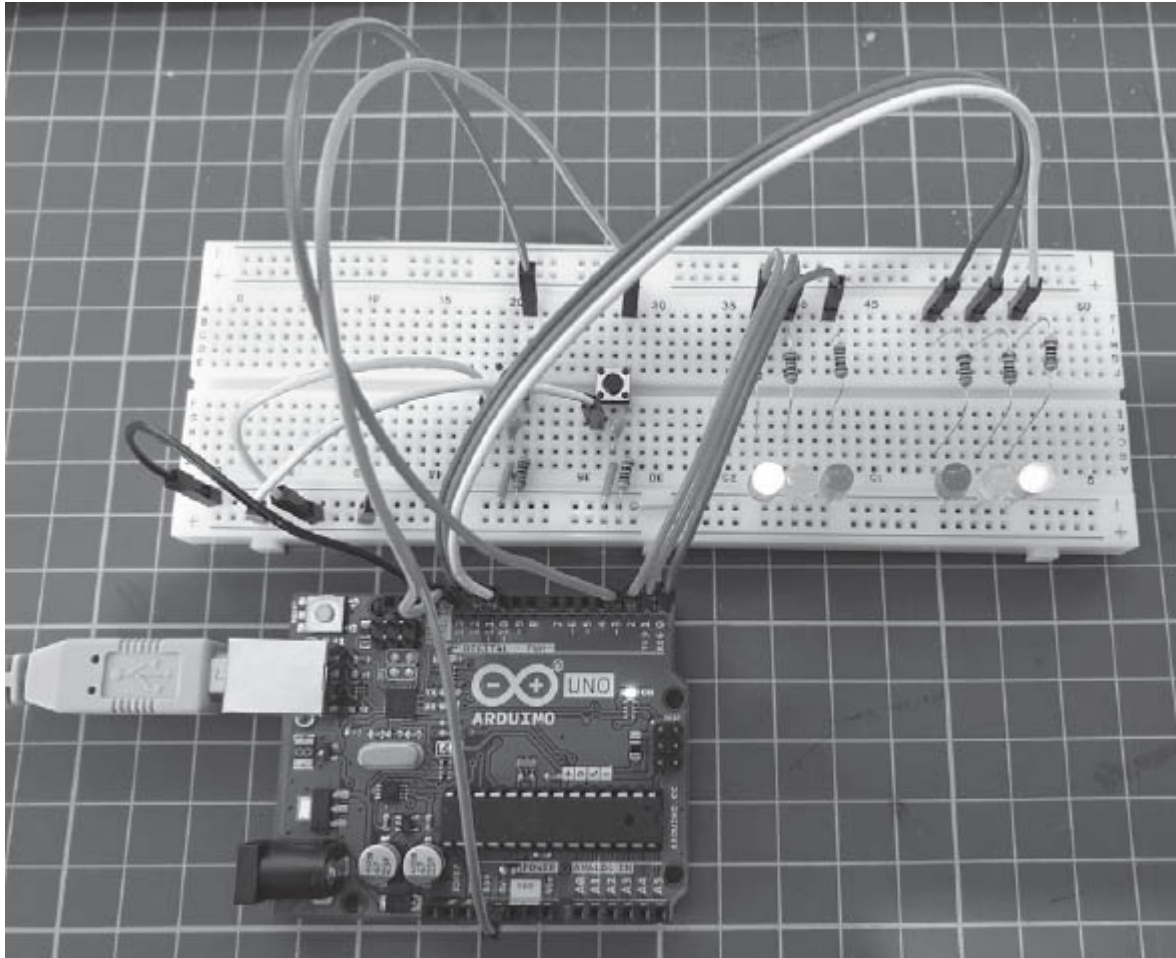


Abb. 4-4 Die vollständige Schaltung

Der Sketch

Kommen wir jetzt zum Sketch. Können Sie erkennen, wie er den Algorithmus umsetzt?

```
// Projekt 5 - Eine Verkehrsampel

// Definiert die Pins, mit denen die Tasten und LEDs
verbunden sind:

#define westButton 3    ❶

#define eastButton 13

#define westRed 2
```

```

#define westYellow 1

#define westGreen 0

#define eastRed 12

#define eastYellow 11

#define eastGreen 10

#define yellowBlinkTime 500 // Lässt die gelbe LED 0,5 s
blinker

boolean trafficWest = true; // West = true, Ost = false
❷

int    flowTime = 10000;    // Zeitraum, in dem der
Verkehr fließt ❸

int    changeDelay = 2000; // Zeitraum zwischen
Farbwechseln ❹

void setup()

{

    // Richtet die digitalen E/A-Pins ein

    pinMode(westButton, INPUT);

    pinMode(eastButton, INPUT);

    pinMode(westRed, OUTPUT);

```

```

pinMode(westYellow, OUTPUT);

pinMode(westGreen, OUTPUT);

pinMode(eastRed, OUTPUT);

pinMode(eastYellow, OUTPUT);

pinMode(eastGreen, OUTPUT);

// Legt den Anfangszustand der LEDs fest, im Westen
zuerst grün

digitalWrite(westRed, LOW);

digitalWrite(westYellow, LOW);

digitalWrite(westGreen, HIGH);

digitalWrite(eastRed, HIGH);

digitalWrite(eastYellow, LOW);

digitalWrite(eastGreen, LOW);

}

void loop()

{

    if ( digitalRead(westButton) == HIGH )    // Fordert
Verkehrsfluss

                                // von West nach Ost an

```

```

{

    if ( trafficWest != true )

        // Fährt nur fort, wenn der Verkehr

        // in die entgegengesetzte Richtung fließt (Osten)

        {

            trafficWest = true; // Ändert Verkehrsflussflag von
            West nach Ost

            delay(flowTime); // Räumt Zeit für den Verkehr ein

            digitalWrite(eastGreen, LOW);    // Schaltet die
            Ostampel

                                // von Grün über Gelb auf Rot

            digitalWrite(eastYellow, HIGH);

            delay(changeDelay);

            digitalWrite(eastYellow, LOW);

            digitalWrite(eastRed, HIGH);

            delay(changeDelay);

            for ( int a = 0; a < 5; a++ ) // Gelbe LED blinkt

                {

```

```

        digitalWrite(westYellow, LOW);

        delay(yellowBlinkTime);

        digitalWrite(westYellow, HIGH);

        delay(yellowBlinkTime);

    }

    digitalWrite(westYellow, LOW);

    digitalWrite(westRed, LOW); // Schaltet Westampel
    von Rot auf Grün

    digitalWrite(westGreen, HIGH);

}

}

if ( digitalRead(eastButton) == HIGH ) // Fordert
Verkehrsfluss

                                // von Ost nach West an

{

    if ( trafficWest == true )

        // Fährt nur fort, wenn der Verkehr in die
        entgegengesetzte

        // Richtung fließt (Westen)

```

```

{

    trafficWest = false; // Ändert Verkehrsflussflag
    von West nach Ost

    delay(flowTime); // Räumt Zeit für den Verkehr ein

    digitalWrite(westGreen, LOW);

    // Schaltet die Ostampel von Grün über Gelb auf Rot

    digitalWrite(westYellow, HIGH);

    delay(changeDelay);

    digitalWrite(westYellow, LOW);

    digitalWrite(westRed, HIGH);

    delay(changeDelay);

    for ( int a = 0 ; a < 5 ; a++ ) // Gelbe LED
    blinkt

    {

        digitalWrite(eastYellow, LOW);

        delay(yellowBlinkTime);

        digitalWrite(eastYellow, HIGH);

        delay(yellowBlinkTime);

    }
}

```

```

digitalWrite(eastYellow, LOW);

digitalWrite(eastRed, LOW); // Schaltet Westampel
von Rot auf Grün

digitalWrite(eastGreen, HIGH);

}

}

}

```

Unser Sketch beginnt bei ❶ mit `#define`, um die Nummern der digitalen Pins für die LEDs und die beiden Drucktasten mit Bezeichnungen zu versehen. Für die West- und die Ostseite der Brücke gibt es jeweils eine rote, eine gelbe und eine grüne LED sowie eine Taste. Mit der bei ❷ definierten booleschen Variable `trafficWest` halten wir fest, in welche Richtung der Verkehr fließt – bei `true` von Westen nach Osten, bei `false` von Osten nach Westen.

HINWEIS

Beachten Sie, dass wir die Verkehrsrichtung mit einer einzigen Variable festhalten (`trafficWest`), die entweder `true` oder `false` ist. Diese Vorgehensweise anstatt der Verwendung von zwei Variablen (eine für Verkehr Richtung Osten, die andere für Verkehr Richtung Westen) bietet den Vorteil, dass nicht versehentlich beide Richtungen wahr sein können. Das hilft uns, Zusammenstöße zu vermeiden!

Die bei ❸ definierte Integervariable `flowTime` gibt an, wie viel Zeit die Fahrzeuge mindestens haben, um die Brücke zu überqueren. Wenn ein Fahrzeug an einer roten Ampel zum Stehen kommt, wartet das System diesen Zeitraum ab, um dem entgegenkommenden Verkehr die Gelegenheit zu geben, über die Brücke zu fahren. Die bei ❹ definierte Integervariable `changeDelay` enthält den Zeitraum, den die Ampeln brauchen, um von Grün über Gelb auf Rot umzuschalten.

Bevor der Sketch in den Abschnitt von `void loop()` eintritt, wird er in `void setup()` für die Verkehrsrichtung von West nach Ost eingerichtet.

Den Sketch ausführen

Wenn der Sketch läuft, tut er zunächst nichts, bis eine der Tasten gedrückt wird. Handelt es sich dabei um die Osttaste, sorgt die folgende Zeile dafür, dass die Ampeln nur dann umgeschaltet werden, wenn der Verkehr in die entgegengesetzte Richtung fließt:

```
if ( trafficWest == true )
```

Der Rest dieses Codeabschnitts besteht aus einer einfachen Abfolge von Wartevorgängen und Ein-/Aus-Schaltbefehlen für die einzelnen LEDs, um den Betrieb von Verkehrsampeln zu simulieren.

Analoge und digitale Signale

In diesem Abschnitt lernen Sie den Unterschied zwischen analogen und digitalen Signalen kennen und erfahren, wie Sie analoge Signale an den analogen Eingangspins messen.

Bis jetzt haben wir in unseren Sketchen nur digitale elektrische Signale verwendet, die ausschließlich zwei diskrete Pegel annehmen können: Wir haben mit `digitalWrite(pin, HIGH)` und `digitalWrite(pin, LOW)` eine LED zum Blinken gebracht und mit `digitalRead()` gemessen, ob an einem Digitalpin eine Spannung anliegt (HIGH) oder nicht (LOW). Abbildung 4-28 zeigt den Verlauf eines digitalen Signals, das abrupt zwischen hohem und niedrigem Pegel wechselt.

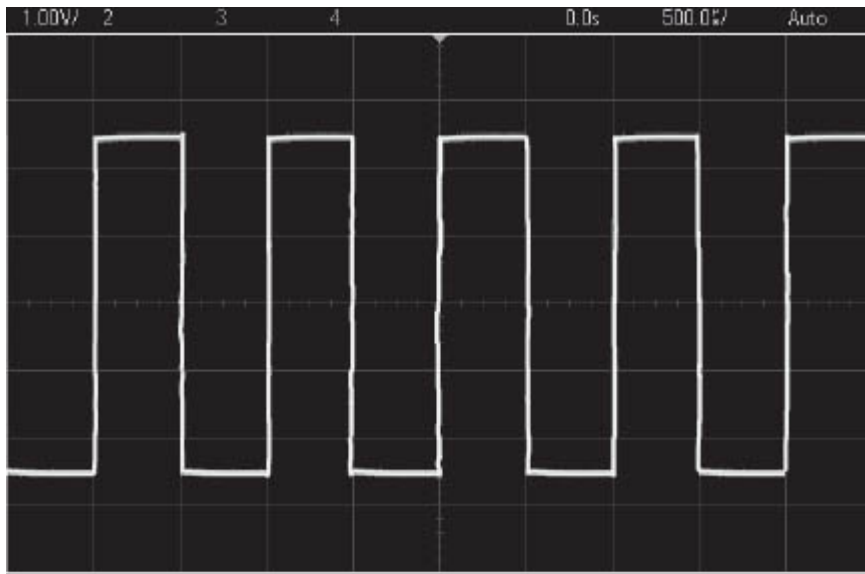


Abb. 4-4 Ein digitales Signal, wobei die beiden möglichen Pegel in Form von horizontalen Linien dargestellt werden, HIGH oben und LOW unten

Im Gegensatz zu den digitalen Signalen können die analogen unendlich viele Werte zwischen hohem und niedrigem Pegel annehmen. In Abbildung 4-29 sehen Sie als Beispiel ein analoges Signal in Form einer Sinuskurve. Im Verlauf der Zeit wechselt der Spannungspegel fließend zwischen hoch und niedrig.

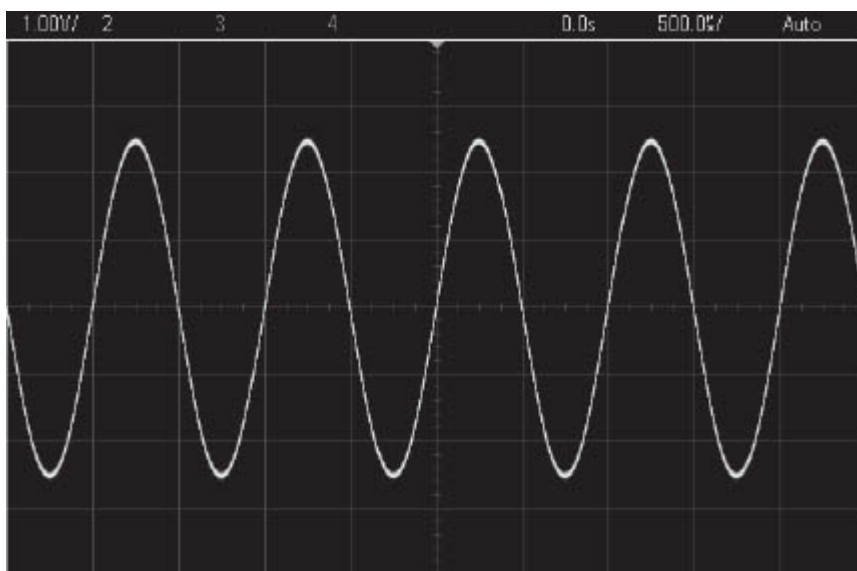


Abb. 4-4 Ein analoges Signal in Form einer Sinuskurve

Beim Arduino liegt der hohe Pegel näher an 5 V und der niedrige näher an 0 V oder GND. Die Spannungswerte von analogen Signalen können wir am Arduino an den sechs analogen Eingangspins messen, die Sie in Abbildung 4-30 sehen. Dies ist gefahrlos im Bereich zwischen 0 V (GND) und maximal 5 V möglich.

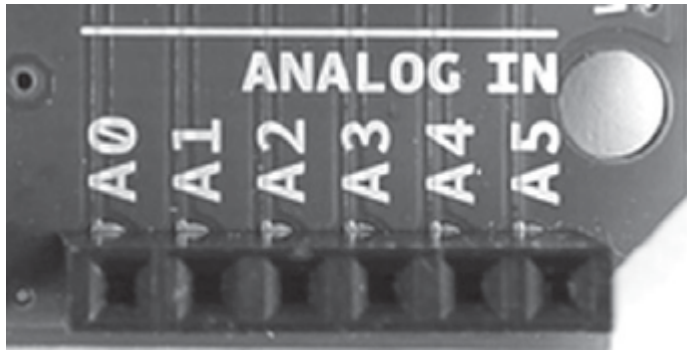


Abb. 4-4 Die analogen Eingangspins am Arduino Uno

Wenn Sie die Funktion `analogRead()` verwenden, gibt der Arduino proportional zu der an dem abgefragten Analogpin anliegenden Spannung eine Zahl zwischen 0 und 1023 zurück. Im folgenden Beispiel sehen Sie, wie Sie mit `analogRead()` den Wert des Analogpins 0 in der Integervariable `a` speichern:

```
a = analogRead(0); // Liest den analogen Eingangspin 0
(A0)

// Gibt einen Wert zwischen 0 und 1023
zurück,

// was gewöhnlich 0,000 bis 4,995 V
entspricht
```

Projekt Nr. 6: Ein Testgerät für Einzelzellenbatterien

Verbreitung und Verwendung der klassischen Batterien sind zwar zurückgegangen, aber in den meisten Haushalten gibt es immer noch einige Geräte, in denen AA-, AAA-, C- oder D-Zellen zum Einsatz kommen, z. B. für Fernbedienungen, Wanduhren oder Kinderspielzeuge. Die Spannung solcher Batterien liegt weit unter 5 V, weshalb wir sie mit unserem Arduino messen können, um den Zustand der Batterie zu ermitteln. In diesem Projekt bauen wir ein Batterietestgerät.

Das Ziel

Einzelzellenbatterien wie AA-Zellen haben im Neuzustand gewöhnlich eine Spannung von 1,6 V, die im Verlauf der Nutzungsdauer abnimmt. Wir wollen die

Spannung messen und den Zustand der Batterie optisch über LEDs anzeigen. Dazu rechnen wir das Messergebnis von `analogRead()` in Volt um. Die höchste Spannung, die wir messen können, beträgt 5 V, weshalb wir 5 durch 1024 dividieren (die Anzahl der möglichen Werte), was 0,0048 ergibt. Wenn `analogRead()` beispielsweise 512 zurückgibt, müssen wir diesen Wert mit 0,0048 multiplizieren, wodurch wir auf 2,4576 V kommen.

Der Algorithmus

Der Algorithmus für unseren Batterienprüfer sieht wie folgt aus:

1. Lies von Analogpin 0.
2. Multipliziere den Messwert mit 0,0048, um den Spannungswert in Volt zu erhalten.
3. Ist die Spannung größer oder gleich 1,6 V, dann schalte kurz die grüne LED ein.
4. Ist die Spannung größer als 1,4 V *und* kleiner als 1,6 V, dann schalte kurz die gelbe LED ein.
5. Ist die Spannung kleiner als 1,4 V, dann schalte kurz die rote LED ein.
6. Wiederhole den Vorgang unendlich oft.

Die Hardware

Für dieses Projekt benötigen Sie folgende Teile:

- drei 560- Ω -Widerstände (R1 bis R3)
- einen 2,2-k Ω -Widerstand (R4)
- eine grüne LED (LED1)
- eine gelbe LED (LED2)
- eine rote LED (LED3)
- eine Steckplatine
- Verbindungsdrähte
- Arduino und USB-Kabel

Der Schaltplan

Den Schaltplan für den Einzelzellenprüfer sehen Sie in Abbildung 4–31. Beachten Sie die beiden mit + und – bezeichneten Anschlusspunkte auf der linken Seite. Verbinden Sie sie mit dem *gleichnamigen* Pol der zu prüfenden Batterie, also Plus mit Plus und Minus mit Minus.

WARNUNG

Versuchen Sie auf keinen Fall, einen Wert von mehr als 5 V zu messen. Vertauschen Sie auch nicht die Pole! Dadurch beschädigen Sie den Arduino.

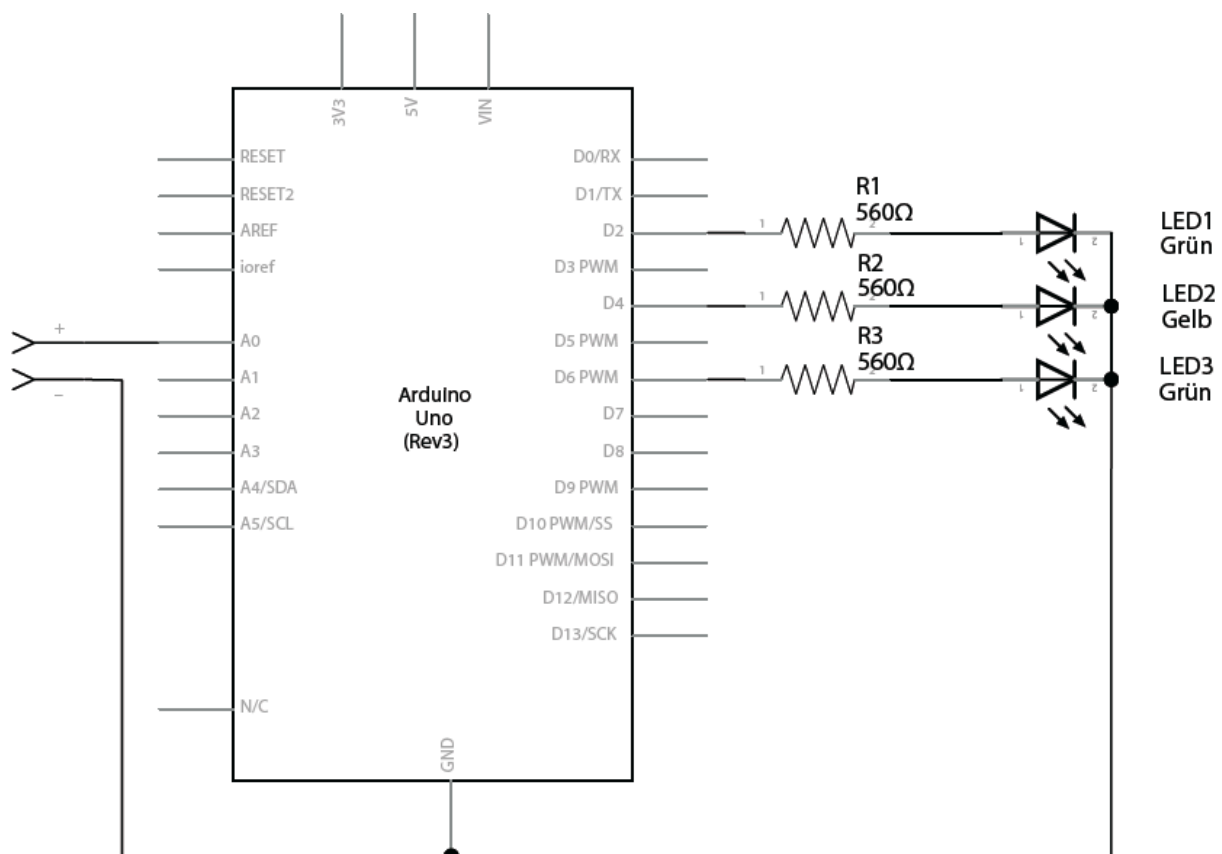


Abb. 4–4 Schaltplan für Projekt 6

Der Sketch

Und hier der Sketch: Da analoge Werte sich oft zwischen ganzen Zahlen bewegen, verwenden wir nun einen neuen Variablentyp namens **float**, welcher Bruch- oder Dezimalwerte enthalten kann.

```
// Projekt 6: Ein Testgerät für Einzelzellenbatterien
```

```
#define newLED 2    // Grüne LED "neu"

#define okLED 4    // Gelbe LED "ok"

#define oldLED 6   // Rote LED "alt"

int analogValue = 0;

float voltage = 0; ❶

int ledDelay = 2000;

void setup()

{

    pinMode(newLED, OUTPUT);

    pinMode(okLED, OUTPUT);

    pinMode(oldLED, OUTPUT);

}

void loop()

{

    analogValue = analogRead(0); ❷

    voltage = 0.0048*analogValue; ❸

    if ( voltage >= 1.6 ) ❹
```

```

{

    digitalWrite(newLED, HIGH);

    delay(ledDelay);

    digitalWrite(newLED, LOW);

}

else if ( voltage < 1.6 && voltage > 1.4 ) ⑤

{

    digitalWrite(okLED, HIGH);

    delay(ledDelay);

    digitalWrite(okLED, LOW);

}

else if ( voltage <= 1.4 ) ⑥

{

    digitalWrite(oldLED, HIGH);

    delay(ledDelay);

    digitalWrite(oldLED, LOW);

}

```

```
}
```

In dem Sketch für Projekt 6 liest der Arduino bei ② den Wert von Analogpin 0 und wandelt ihn bei ③ in einen Spannungswert um. Was es mit dem neuen Variablentyp `float` auf sich hat, der bei ① auftritt, erfahren Sie im nächsten Abschnitt, welcher nicht nur das Rechnen mit dem Arduino beinhaltet, sondern auch den Vergleichsoperatoren für Zahlenwerte.

Rechnen mit dem Arduino

Wie ein Taschenrechner kann der Arduino arithmetische Operationen wie Multiplikation, Division, Addition und Subtraktion durchführen:

```
a = 100;  
  
b = a + 20;  
  
c = b - 200;  
  
d = c + 80; // d ist gleich 0
```

Fließkommavariablen

Für Zahlen mit Dezimalstellen können Sie den Variablentyp `float` verwenden. Damit ist es möglich, Zahlen zwischen $3,4028235 \times 10^{38}$ und $-3,4028235 \times 10^{38}$ zu speichern. Die Genauigkeit ist gewöhnlich auf sechs bis sieben Dezimalstellen beschränkt. In Berechnungen können Sie ganze Zahlen (Integer) und Fließkommazahlen (`float`) mischen. Im folgenden Beispiel wird erst die Fließkommazahl `f` zum Integer `a` addiert und das Ergebnis dann in der `float`-Variable `g` gespeichert:

```
int a = 100;  
  
float f;
```

```
float g;  
  
f = a / 3; // f = 33.333333  
  
g = a + f; // g = 133.333333
```

Vergleichsoperatoren für Berechnungen

Für die if-Anweisungen und digitalen Eingangssignale in Projekt 5 haben wir Vergleichsoperatoren wie == und != verwendet. Daneben gibt es auch noch folgende Operatoren zum Vergleich von Zahlen und numerischen Variablen:

- < kleiner als
- > größer als
- <= kleiner oder gleich
- >= größer oder gleich

Diese Operatoren haben wir im Sketch von Projekt 6 zum Vergleich von Zahlenwerten in den Zeilen 4, 5 und 6 verwendet.

Die Genauigkeit der Analogmessung durch eine Bezugsspannung verbessern

Wie Sie in Projekt 6 gesehen haben, gibt die Funktion analogRead() einen Wert zurück, der proportional zur Spannung zwischen 0 V und 5 V ist. Der obere Wert (5 V) ist die *Bezugsspannung*, also die höchste Spannung, die der analoge Eingang des Arduino akzeptiert und für die er den höchsten Wert (1023) zurückgibt.

Um die Genauigkeit bei der Messung niedrigerer Spannungen zu erhöhen, können Sie eine geringere Bezugsspannung verwenden. Beträgt die Bezugsspannung 5 V, so stellt analogRead() mit den Werten zwischen 0 und 1023 den gesamten Bereich bis 5 V dar. Wollen wir aber nur Spannungen mit maximal 2 V messen, können wir die Ausgabe des Arduino so ändern, dass nur der Bereich bis 2 V durch die Werte von 0 bis 1023 wiedergegeben wird, was eine genauere Messung ermöglicht. Dazu können Sie sowohl eine externe als auch

eine interne Bezugsspannung heranziehen, wie Sie im folgenden Abschnitt erfahren.

Externe Bezugsspannung

Die erste Methode, um eine Bezugs- oder Referenzspannung vorzugeben, bietet der Pin *AREF* (für *analoge Referenz*), den Sie in Abbildung 4–32 sehen.



Abb. 4-4 Der **AREF**-Pin auf dem Arduino Uno

Um eine neue Bezugsspannung einzuführen, verbinden Sie die Spannungsquelle mit dem *AREF*-Pin und den zugehörigen *GND*-Anschluss mit *GND* auf dem Arduino. Beachten Sie, dass Sie damit nur eine niedrigere und keine höhere Bezugsspannung festlegen können, da eine am Arduino Uno angelegte Spannung nicht höher sein darf als 5 V. Eine einfache Möglichkeit, um eine niedrigere Bezugsspannung einzurichten, besteht darin, aus zwei Widerständen einen *Spannungsteiler* zu bauen, wie Sie in Abbildung 4–33 sehen.

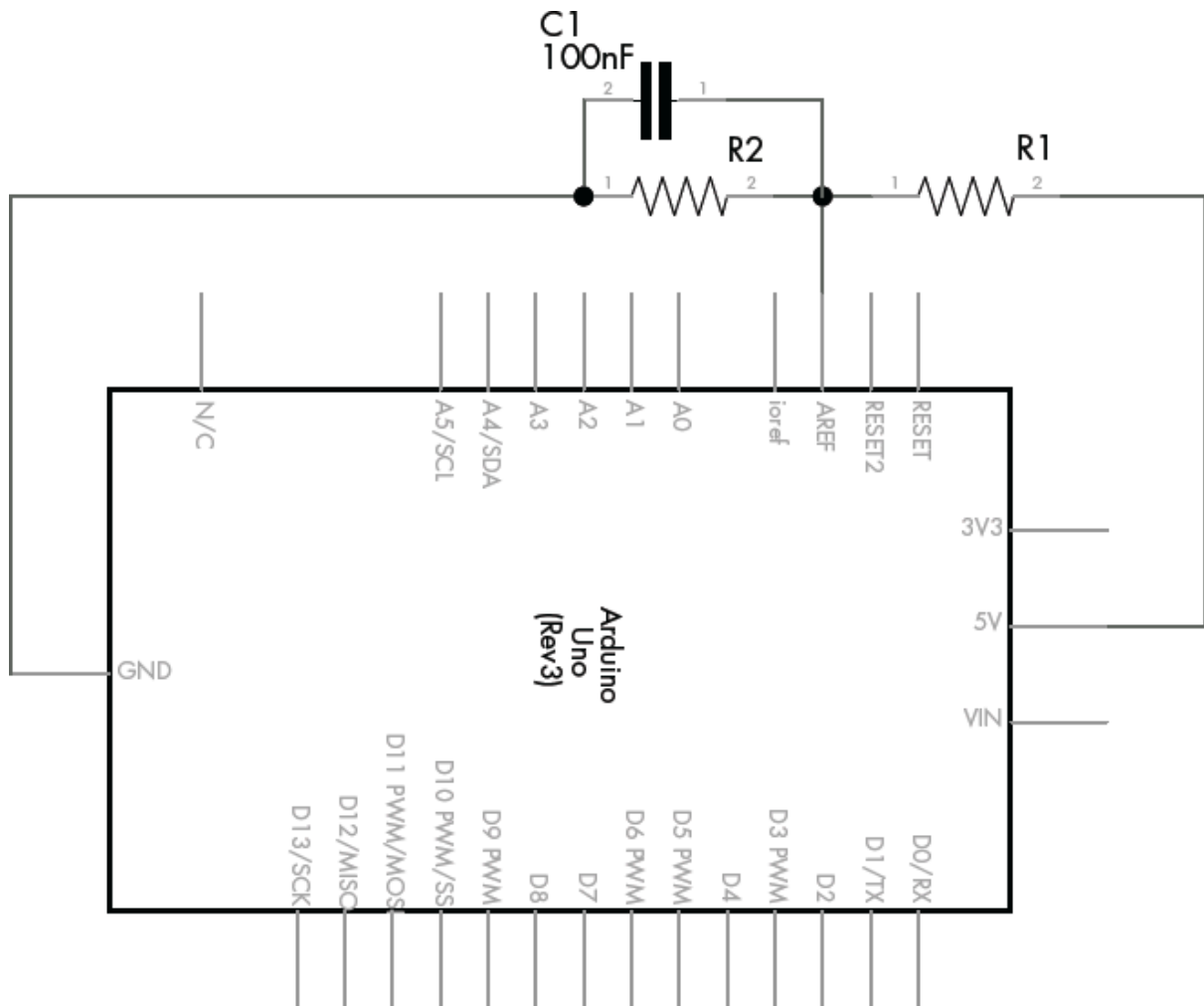


Abb. 4-4 Spannungsteiler

Die Werte $R1$ und $R2$ bestimmen die Bezugsspannung nach folgender Formel:

$$V_{\text{out}} = V_{\text{in}} \left(\frac{R2}{R1 + R2} \right)$$

Abb. 4-4 Formel für die Referenzspannung

Dabei ist V_{out} die Bezugsspannung und V_{in} die Eingangsspannung, in diesem Fall also 5 V. $R1$ und $R2$ sind die Widerstandswerte in Ohm.

Die einfachste Möglichkeit besteht darin, V_{in} zu halbieren, indem Sie $R1$ und $R2$ auf denselben Wert setzen, z. B. je 10 kΩ. Dabei sollen Sie Widerstände mit möglichst geringer Toleranz verwenden, z. B. 1 %. Messen Sie die tatsächlichen Widerstandswerte mit einem Multimeter und verwenden Sie diese Werte in den Berechnungen. Außerdem ist es sinnvoll, zwischen $AREF$ und GND einen 100-nF-Kondensator einzubauen, um ein Rauschen am $AREF$ und instabile analoge Messungen zu verhindern.

Bei der Verwendung einer externen Bezugsspannung müssen Sie in den Abschnitt `void setup()` im Sketch folgende Zeile einfügen:

```
analogReference(EXTERNAL); // Wählt den AREF-Pin als  
Bezugsspannung aus
```

Interne Bezugsspannung

Der Arduino Uno verfügt auch über eine interne 1,1-Volt-Bezugsspannung. Wenn das für Ihren Bedarf geeignet ist, brauchen Sie keine zusätzliche Hardware. Fügen Sie einfach folgende Zeile zu `void setup()` hinzu:

```
analogReference(INTERNAL); // Wählt die interne  
  
// 1,1-Volt-Bezugsspannung aus
```

Regelbare Widerstände

Regelbare Widerstände oder *Potenzimeter* («Poti») können im Allgemeinen von 0Ω bis zum Nennwert eingestellt werden. Das Schaltplansymbol sehen Sie in Abbildung 4–35.



Abb. 4–4 Schaltplansymbol für einen regelbaren Widerstand (*Potenzimeter*)

Bei einem regelbaren Widerstand finden Sie drei Kontaktstifte: einen in der Mitte und zwei an den Seiten. Wenn Sie die Mittelachse drehen, wird der Widerstand zwischen der einen Seite und der Mitte erhöht und der Widerstand zwischen der Mitte und der anderen Seite verringert.

Es gibt *linear* und *logarithmisch* regelbare Widerstände. Bei den linearen Modellen ändern sich der Widerstandswert beim Drehen mit konstanter Rate, bei den logarithmischen erst langsam und dann immer schneller. In Audioverstärkerschaltkreisen werden meistens logarithmische Potenziometer verwendet, da sie den Verlauf des menschlichen Hörvermögens nachbilden. Ob ein Potenziometer logarithmisch oder linear ist, können Sie an den

Markierungen auf der Rückseite ablesen: Ein A steht in diesem Fall für logarithmisch und ein B für linear.

Dagegen kommen in Arduino-Projekten eher linear regelbare Widerstände wie der aus Abbildung 4-36 zum Einsatz.



Abb. 4-4 Ein typischer linear regelbarer Widerstand

Es gibt auch Miniaturversionen von regelbaren Widerständen, sogenannte *Trimpotenzimeter* oder *Trimmer* (siehe Abb. 4-37). Aufgrund ihrer Größe sind sie vor allem für die Widerstandsregelung in »richtigen« Schaltkreisen sehr nützlich. Da sie eingesteckt werden können, eignen sie sich aber auch für die Arbeit auf der Steckplatine.



Abb. 4-4 Verschiedene Modelle von Trimpotenzimetern

HINWEIS

Schauen Sie sich die verschiedenen Modelle genau an, wenn Sie Trimpotenzio­meter kaufen. Wahrscheinlich sind für Ihre Zwecke die Varianten geeignet, die sich mit einem handelsüblichen Schraubendreher einstellen lassen. Außerdem sind die in Abbildung 4–37 gezeigten geschlossenen Typen haltbarer als die billigen Versionen mit offenen Kontakten.

Piezo­elektrische Summer

Ein *piezo­elektrischer Summer* ist ein kleines rundes Bauteil, mit dem Sie laute und nervtötende Töne erzeugen können – ideal für Alarmanlagen oder um Spaß zu haben. Abbildung 4–38 zeigt ein typisches Beispiel, den TDK PS1240, neben einer Münze, um Ihnen einen Eindruck von der Größe zu geben.

Im Gehäuse eines Piezosummers befindet sich ein sehr dünnes Plättchen, das sich beim Anlegen eines elektrischen Stroms bewegt. Bei Wechselstrom (ein ... aus ... ein ... aus ...) vibriert das Plättchen, was Schallwellen erzeugt.

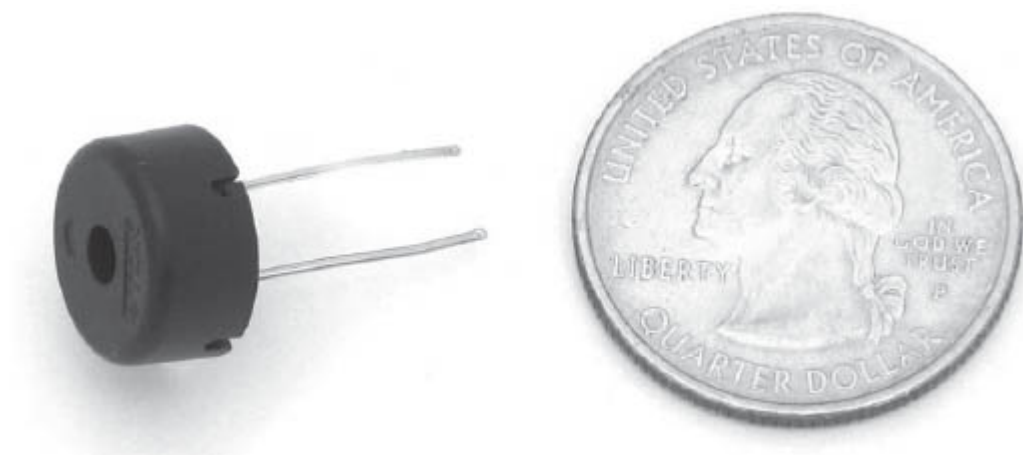


Abb. 4–4 Der Piezosummer TDK PS1240

Da sich Piezosummer wie LEDs ein- und ausschalten lassen, können Sie sie auf ganz einfache Weise in Arduino-Projekten verwenden. Diese Elemente sind nicht gepolt, sondern können in beliebiger Richtung eingebaut werden.

Das Schaltplansymbol

Das Schaltplansymbol für einen Piezosummer sieht wie ein Lautsprecher aus (siehe Abb. 4–39), was es sehr leicht erkennbar macht.

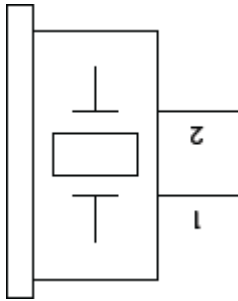


Abb. 4-4 Das Schaltplansymbol für einen Piezosummer

HINWEIS

Beim Einkauf der Piezosummer für dieses Projekt müssen Sie darauf achten, dass Sie einen Typ erwischen, der **nur aus dem Piezoelement** besteht. Es gibt auch Modelle, die zwar so aussehen wie das Bauteil in Abbildung 4-38, aber über einen Tonsteuerungsschaltkreis im Gehäuse verfügen. Da wir den Ton direkt vom Arduino aus steuern wollen, können wir diesen Summertyp nicht gebrauchen.

Projekt Nr. 7: Einen Piezosummer ausprobieren

Wenn Sie einen Piezosummer zur Hand haben und ausprobieren wollen, schließen Sie ihn zunächst zwischen Arduino-GND und den digitalen Pins D3 bis einschließlich D0 an. Laden Sie anschließend den folgenden Sketch auf Ihren Arduino hoch:

```
// Projekt 7: Einen Piezosummer ausprobieren

#define PIEZO 3 // Pin 3 ist PBM-fähig und eignet sich

                // für die Tonsteuerung

int del = 500;

void setup()

{

    pinMode(PIEZO, OUTPUT);
```

```

}

void loop()

{

    analogWrite(PIEZO, 128);    // Tastverhältnis 50% ❶

    delay(del);

    digitalWrite(PIEZO, LOW); // Schaltet den Piezosummer
    aus

    delay(del);

}

```

In diesem Sketch nutzen wir eine Pulsbreitenmodulation am Digitalpin 3. Um die Lautstärke zu ändern, geben Sie bei ❶ ein anderes Tastverhältnis in der Funktion `analogWrite()` an (zurzeit steht es auf 128, also 50%).

Wollen Sie die Maximallautstärke des Summers erhöhen, müssen Sie eine höhere Spannung anlegen. Sie ist zurzeit auf 5 V beschränkt, aber bei 9 V oder 12 V ist der Summer viel lauter. Da Sie höhere Spannungen nicht vom Arduino beziehen können, brauchen Sie eine externe Stromquelle für den Summer, etwa eine 9-Volt-Batterie, und müssen einen Transistor als elektronischen Schalter verwenden, um die Versorgung des Summers zu regeln. Die erforderliche Schaltung dafür sehen Sie in Abbildung 4-40. Den Sketch für dieses Projekt müssen Sie dazu nicht verändern.

Der mit 12 V beschriftete Teil des Schaltplans stellt den Pluspol der externen Spannungsquelle dar. Den Minuspol müssen Sie mit dem *GND*-Pin des Arduino verbinden.

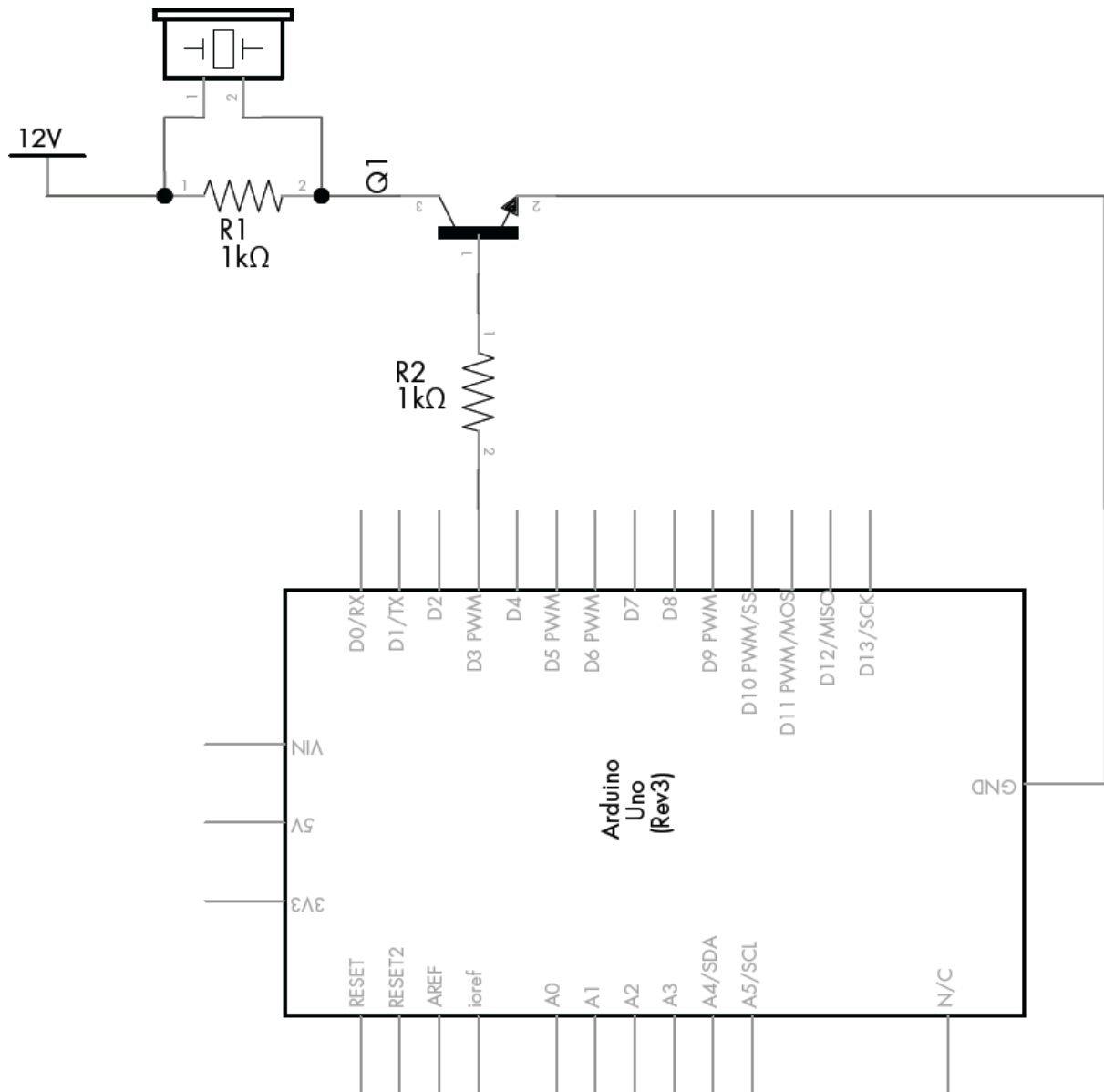


Abb. 4-4 Schaltplan für Projekt 7

Projekt Nr. 8: Ein Thermometer mit Ampelanzeige

Temperatur kann durch ein analoges Signal dargestellt werden. Zur Messung verwenden wir den Temperatursensor TMP36 mit Spannungsausgang von Analog Devices (<http://www.analog.com/tmp36/>), den Sie in Abbildung 4-41 sehen.

- eine rote LED (LED1)
- eine grüne LED (LED2)
- eine blaue LED (LED3)
- einen Temperatursensor TMP36
- eine Steckplatine
- Verbindungsdrähte
- Arduino und USB-Kabel

Der Schaltplan

Der Aufbau der Schaltung ist einfach. Wenn Sie auf die beschriftete Seite des TMP36 blicken, gehört der linke Anschluss an den 5-Volt-Ausgang, der mittlere (der Spannungsausgang) an den Analogeingang und der rechte an *GND* (siehe Abb. 4–42).

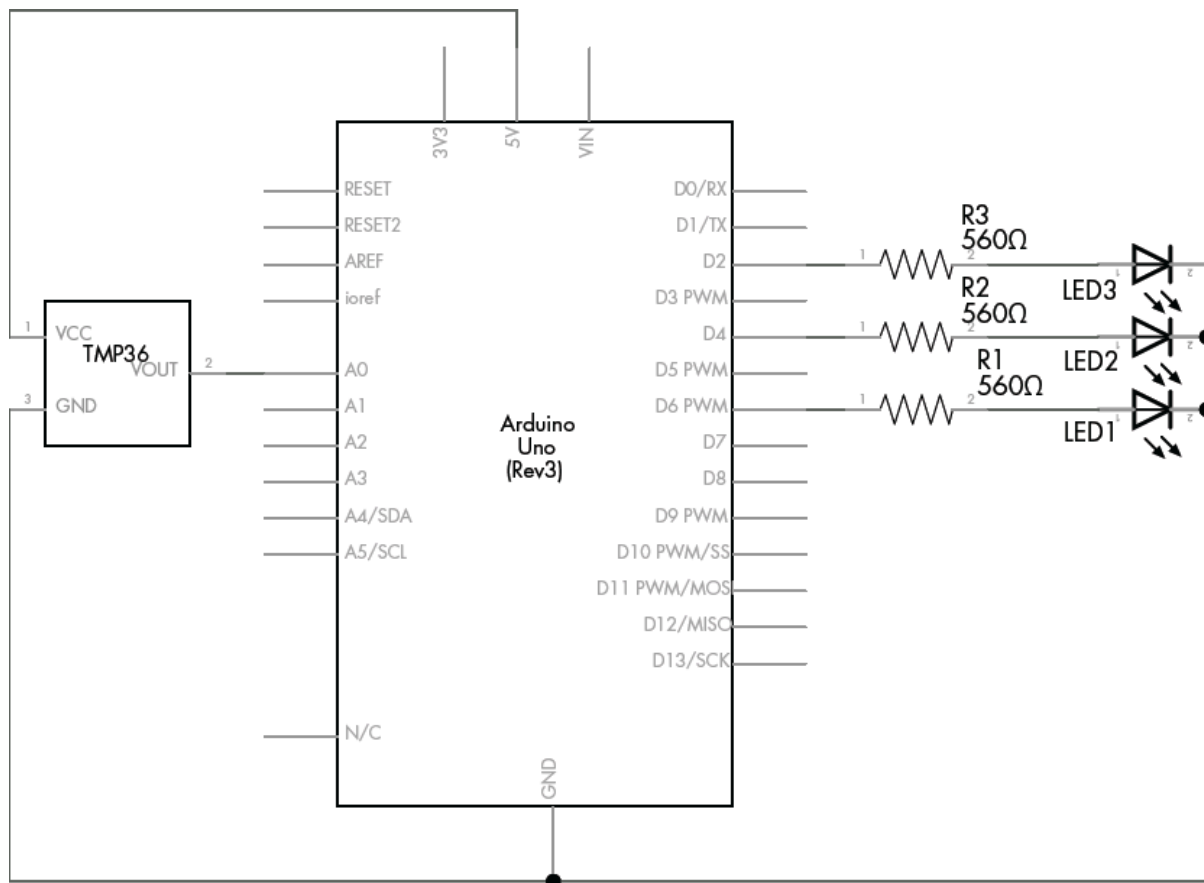


Abb. 4–4 Schaltplan für Projekt 8

Der Sketch

Hier sehen Sie den Sketch:

```
// Projekt 8: Ein Thermometer mit Ampelanzeige

// Definiert die Pins, mit denen die LEDs verbunden sind:

#define HOT 6

#define NORMAL 4

#define COLD 2

float voltage = 0;

float celsius = 0;

float hotTemp = 26;

float coldTemp = 20;

float sensor = 0;

void setup()

{

    pinMode(HOT, OUTPUT);

    pinMode(NORMAL, OUTPUT);

    pinMode(COLD, OUTPUT);

}
```

```

void loop()

{

    // Liest den Temperatursensor ab und rechnet das Ergebnis

    // in Celsiuswerte um

    sensor = analogRead(0);    ❶

    voltage = (sensor*5000)/1024;    // Rechnet Sensorrohwert
    in mV um

    voltage = voltage-500;        // Entfernt Versatz

    celsius = voltage/10;        // Rechnet mV- in
    Celsiuswert um

    // Reagiert je nach Temperaturbereich

    if ( celsius < coldTemp )    ❷

    {

        digitalWrite(COLD, HIGH);

        delay(1000);

        digitalWrite(COLD, LOW);

    }

    else if ( celsius > coldTemp && celsius <= hotTemp )    ❸

```

```

    {

        digitalWrite(NORMAL, HIGH);

        delay(1000);

        digitalWrite(NORMAL, LOW);

    }

else

{

    // Temperatur ist > hotTemp

    digitalWrite(HOT, HIGH);

    delay(1000);

    digitalWrite(HOT, LOW);

}

}

```

In diesem Sketch wird als Erstes die Spannung vom Sensor TMP36 abgelesen und dann bei ❶ in Celsius-Temperaturwerte umgerechnet. Danach vergleicht er mit den `if-else`-Funktionen bei ❷ und ❸ die vorliegende Temperatur mit den Werten für heiß und kalt und schaltet die jeweils zugehörige LED ein. Mit `delay(1000)` wird verhindert, dass die Leuchtdioden hektisch blinken, wenn die Temperatur schnell zwischen zwei Bereichen wechselt. Sie können mit dem Thermometer experimentieren, indem Sie kühle Luft darüber blasen, um die Temperatur zu senken, oder indem Sie mit zwei Fingern den TMP36-Sensor reiben, um Wärme zu erzeugen.

Ausblick

Damit sind wir am Ende von Kapitel 4 angelangt. Sie haben jetzt eine Menge weiterer Möglichkeiten kennengelernt, darunter digitale Ein- und Ausgänge, neue Arten von Variablen und verschiedene mathematische Funktionen. Im nächsten Kapitel werden Sie noch mehr mit LEDs arbeiten, lernen, wie Sie eigene Funktionen schreiben können, ein Computerspiel und einen elektronischen Würfel konstruieren usw.

Index

Symbole

! 72

&& 72

!= 70

= 70

|| 72

74HC595 123

#define 69

A

Abstandsmessung

 Infrarot 309

 Ultraschall 318

Analoge Signale 80

Analogpins

 Ausgangswert für Zufallszahlen 116

 Bezugsspannung 85

 Eigenbau-Arduino 266

 Eingänge 80

 Messgenauigkeit verbessern 85

analogRead() 81

 Bezugsspannung 85

 Genauigkeit verbessern 85

analogWrite() 45,90

And 72

Arbeitsspeicher 273

Arduino

 Anbieter 5

Arbeitsspeicher 273
Arduino Due 277
Arduino LilyPad 276
Arduino Mega 276
Arduino Nano 275
Arduino Uno 12, 273
Aufbau der Platine 12
Bootloader 265
Digitalpins 14
Drahtlose Fernbedienung 341
Eigenbau auf Steckplatine 261
Erweiterung 15, 145
Fernsteuerung über Infrarot 375
Fernsteuerung über Teleduino 450
Freeronics Eleven 273
Freeronics EtherMega 277
Geschichte 1
IDE 6
LEDs 14
Mikrocontroller 13, 264
Modelle 272
Reset-Schalter 15
Schaltplansymbol 54
Serieller Port 14
Sicherheit 8
Stromanschluss 13
USB-Anschluss 13
AREF-Pin 86
Arithmetische Operationen 84
Arrays 131
 Transpondernummern vergleichen 392
 Zeichen für LCD definieren 200
ATmega328 264, 272
Auffangregister 125
Autoscroll 105

B

Basis (Transistor) 47

Bauteile

Chipwiderstände 32

Darlington-Transistor 287

Digitalpotenziometer 413

Dioden 48

Drucktasten 61

Elektrolytkondensatoren 60

Gleichrichterioden 48

Infrarotsensoren 309

Keramikkondensatoren 60

Kondensatoren 59

LCD-Grafikmodule 202

LCD-Zeichenmodule 195

LEDs 34

Mikrocontroller 264

Mikroschalter 305

Numerische Tastenfelder 239

Piezosummer 89

Potenziometer 87

Quarzoszillatoren 263

Relais 49

RF-Link-Module 339

RTC-Module 419

Siebensegmentanzeigen 133

Spannungsregler 263

Symbole in Schaltplänen 54

Tastfelder 239

Temperatursensoren 92

Touchscreens 247

Transistoren 47

Widerstände 30

BC548 47

BCD 424

beginTransmission() 403
Bezugsspannung 85
Bibliotheken
 I²C 403
 Infrarotübertragung 372
 Installation unter macOS oder später 225
 Installation unter Windows 7 und höher 224
 IRremote 372
 LCD-Modul 197
 SPI 412
 Teleduino 449
 TinyGPS 336
 Twitter 445
 VirtualWire 341
 Wire 403
Binärkodiertes Dezimalformat 424
Binärzahlen 121
Bitreihenfolge 121, 412
Blockierstrom 287
Boolesche Variablen 71
Bootloader 265
Breadboard 148
Bytevariablen 122

C

Chipwiderstände 32

D

Datenbus

 I²C 401
 main device 402
 secondary device 402
 SPI 413
Debugging 108
delay() 23
Digitale Signale 79

Digitalpins

- Als Ausgang festlegen 21
- Eigenbau-Arduino 266
- Eingänge 61
- Eingang messen 69
- Ein- und ausschalten 23
- Erweitern 123
- Interrupt-Modi 173
- Lage 14
- Porterweiterung 408
- Pulsbreitenmodulation 45, 290
- Steuern über Teleduino 447
- Steuer-URL von Teleduino 450
- Zeit für den Pegelwechsel messen 167

Digitalpotenziometer 413

digitalRead() 69

digitalWrite() 23

Dioden 48

do-while 109

Drahtlose Datenübertragung

- Bibliothek 341
- Empfängerschaltung 342
- Empfängersketch 346
- Fehlerprüfung 341
- Fernbedienung 341
- Preiswerte Module 339
- RF-Link-Module 339
- Senderschaltung 341
- Sendersketch 344
- VirtualWire 341

Drehmoment 282

Drucktasten

- Einführung 61
- Interrupts 173
- Prellen 62

Status prüfen 69

E

Echtzeit-Modul Siehe RTC

EEPROM

Busadresse 404

Daten schreiben und lesen 393

Einführung 273

Extern 404

Lebensdauer 393

Status speichern 395

Variablen speichern 392

einlöten 152

Elektrolytkondensatoren 60

Emitter (Transistor) 47

E-Motoren

Drehzahlregelung 288

Eigenschaften 287

Pulsweitenmodulation 290

endTransmission() 403

Entprellschaltung 64

Ethernet

Freeronics EtherMega 277

F

Farad 59

Feature creep 28

Fernsteuerung

Arduino 375

Arduino vom Web aus steuern 447

Befehls-URL (Teleduino) 450

Codes von Sony-Fernbedienungen 374

Drahtlose Fernbedienung 341

Fernüberwachungsstation 439

Infrarot-Fernbedienung 371

Portweiterleitung 444

Raupenfahrzeug 379
Sensor Daten auf einer Webseite anzeigen 439
SMS-Fernsteuerung 463
Teleduino 447, 450
Flash-Speicher 273
Fließkommavariablen 84
float 85
Flüssigkristallanzeigen Siehe LCD
for-Schleifen 43
Freetronics Eleven 273
Freetronics EtherMega 277
Fritzing 58
FTDI-Kabel 270
Funktionen
 Aktionen wiederholen 98
 analogRead() 81
 analogWrite() 45, 90
 Auskommentieren 424
 beginTransmission() 403
 delay() 23
 digitalRead() 69
 digitalWrite() 23
 Eigene Funktionen schreiben 98
endTransmission() 403
Interrupt-Funktionen 172
Parameter 99
pinMode() 21
randomSeed() 116
read() 403
readKeypad() 245
requestFrom() 403
Serial.flush() 111
void 21
Vorteile 108
vw_send() 345

Werte zurückgeben 100

write() 403

G

Gleichrichterdioden 48, 55

GMT 333

GND

Schaltplansymbol 57

GPS

Bewegungsverlauf aufzeichnen 333

Definition 323

Empfängermodul 328

Empfangsgerät bauen 328

GPS Visualizer 337

Positionsanzeige auf LCD-Bildschirm 330

Standorte auf Landkarte anzeigen 336

Standort in Google Maps bestimmen 330

Testlauf 326

TinyGPS 336

Uhr 331

H

Hexadezimalzahlen 378

HTML 443

I

I²C

Befehle über Teleduino senden 447

Bibliothek 403

Busadresse 404

Daten anfordern 403

Einführung 401

Externer EEPROM 404

Porterweiterung 408

Spannung 402

IC-Abzieher 269

ICs

Porterweiterung 408

RTC-Modul 419

Schieberegister 123

IDE

Befehlsbereich 17

Fehlermeldungen 24

Meldungsbereich 19

Menü 18

Oberfläche 17

Platinentyp ändern 271

Serieller Monitor 105

Sketche hochladen 24

Sketche speichern 20

Sketche überprüfen 23

Symbolleiste 18

Textbereich 19

Upload 24

Verify 23

if 69

Mehrere Vergleiche 73

if-then-else 70

Infrarot

Empfänger 372

Fernsteuerung für Raupenfahrzeug 379

Grundlagen 371

Sony-Codes 374

TV-Fernbedienung 373

Infrarotsensoren 309

ino 21

int 112

Integrierte Schaltkreise Siehe ICs; *Siehe* ICs

Internet

Fernüberwachungsstation 439

Freertronics EtherMega 277

IP-Adresse 438, 442

Portweiterleitung 444
Interrupts 172, 306
IP-Adresse 438, 442
IRremote 372

K

Keramikkondensatoren 60
Kollektor (Transistor) 47
Kollisionswarnung
 Infrarotsensoren 309
 Mikroschalter 305
 Ultraschallsensoren 316
Kommentare 20
Kondensatoren
 Elektrolytkondensatoren 60
 Kapazität 59
 Keramikkondensatoren 60
Konstanten 69
Kontaktprellen 62

L

Latch-Pin 127
LCD
 Anzeige steuern 197
 Bibliothek 197
 Eigene Zeichen definieren 200
 Grafikmodul anschließen 203
 Grafikmodule 202
 Textfunktionen 206
 Variablen anzeigen 199
 Zeichenmodule 195
Least Significant Bit 122
LEDs
 Begrenzungswiderstand 35
 Binärzahlenanzeige 124
 Blinkcodeanzeige 101

Blinken lassen 19
Einführung 34
Helligkeit ändern 44
Helligkeitsregelung auf Touchscreen 256
Helligkeitsregelung mit Digitalpotenziometer 414
Lage auf dem Arduino 14
LED-La-Ola 39
Polung 34
Schaltplansymbol 56
Siebensegmentanzeigen 133
Teleduino-Statuscodes 450
Verkehrsampel 74
Leerlaufstrom 287
Leistung 29
Leuchtdioden Siehe LEDs
LilyPad 276
Linearer Spannungsregler 263
Logische Vergleichsoperatoren 72
long 112
loop() 22
LSB 122

M

MAC-Adresse 444
main device 402
Masseanschluss Siehe GND
Maxim DS3231 419
MCP4162 414
MCP23017 408
micros() 167
microSD
 Bewegungsverlauf aufzeichnen 333
 Freetronics EtherMega 277
 Freetronics EtherTen 438
 Stechuhr 429
 Temperaturdaten aufzeichnen 164

Mikrocontroller

ATmega328 264

Aufkleber mit Pinbelegung 267

Austauschen 268

Bootloader 265

Definition 13

Modelle 272

Pins 267

Mikroschalter 305

millis() 167

Mobilfunk

Voraussetzungen 453

Modulo 140

Most Significant Bit 122

Motoren

Blockierstrom 287

Drehzahlregelung 288

Elektromotoren 287

Leerlaufstrom 287

Stellmotoren 281

Stromquelle 289

MSB 122

MSBFIRST 412

MSBLAST 412

Multimeter 33

N

Not 72

Numerische Tastenfelder Siehe

Tastfelder

O

Ohmsches Gesetz 36

Open-Source-Hardware 278

Operatoren

! 72

- && 72
- <, >, <=, >= 85
- != 70
- == 70
- || 72
- And 72
- Größer/kleiner 85
- Klammern 73
- Logische Vergleichsoperatoren 72
- Mehrere Vergleiche 73
- Modulo 140
- Not 72
- Or 72
- Reihenfolge 73
- Vergleichsoperatoren 70, 85
- Or 72

P

- Parallax Ping))) 316
- Piezosummer 89
 - Maximallautstärke 90
- pinMode() 21
- Porterweiterung 408
- Portweiterleitung 444
- Potenziometer 87
 - Digitalpotenziometer 413
 - Trimmer 88
- Prellen 62
- print() 106
- println() 106
- Projekte
 - Algorithmus 28
 - Arduino-Fernsteuerung 375, 448
 - Arduino-Tweeter 444
 - Arduino-Wahlgerät 458
 - Batterietestgerät 81

Bereiche auf einem Touchscreen ansprechen 249
Bewegungsverlauf aufzeichnen 333
Bilder auf LED-Matrix 191
Binärzahlenanzeige 124
Binärzahlenquiz 128
Daten auf microSD-Karten schreiben 162
Datenübertragung mit LoRa-Modulen 348
Datum und Uhrzeit auf einem RTC-Modul 420
Digitalpotenziometer 413
Digitalthermometer 141
Digitaluhr bauen 425
Drahtlose Fernbedienung 341
Eigene Zeichen für LCD definieren 200
Ein/Aus-Schalter auf einem Touchscreen 252
Eingaben im seriellen Monitor multiplizieren 110
Einstellige Siebensegmentanzeige 136
Elektronischer Würfel 118
Feature creep 28
Fernüberwachungsstation 439
GPS-Empfangsgerät 328
GPS-Uhr 331
Interrupts 174
Kollisionserkennung mit Mikroschalter 305
Kollisionsverhinderung mit IR-Sensor 313
Kollisionsverhinderung mit Ultraschallsensor 318
LED-La-Ola 39
Piezosummer 90
Planung 28
ProtoScrewShield 429
Raupenfahrzeug 295
RFID-Steuerungssystem 389
RFID-Steuerungssystem mit Erinnerung an die letzte Aktion 395
Schloss mit Tastenfeld 243
Stechuhr 429
Steckplatinen-Arduino 261

Stoppuhr 169
Teleduino 448
Temperaturaufzeichnung 164
Temperaturverlaufskurve 212
Textfunktionen für LCD 206
Textnachrichten senden 461
Thermometer mit Ampelanzeige 92
Thermometer mit Blinkcodeanzeige 101
Touchscreen-Schalter mit drei Zonen 256
 Twitter 444
 Verkehrsampel 74
 Zeigerthermometer 284
 Zweistellige Siebensegmentanzeige 138
ProtoShield 148
 Lötarbeiten 152
Pull-down-Widerstand 64
Pulsweitenmodulation 44, 90, 290

Q

Quarzoszillatoren 263

R

randomSeed() 116
Raupenfahrzeuge
 Bausatz 296
 Chassis 296
 Fernsteuerung 379
 Kollisionserkennung 305
 Kollisionsverhinderung mit IR-Sensor 313
 Kollisionsverhinderung mit Ultraschallsensor 318
 Stromversorgung 297
 Wendemanöver 300
read() 403
readKeypad() 245
Relais 49, 56
requestFrom() 403

Reset-Schalter 15
Restladungen 51, 291

RFID

Definition 385
Lesegerät 386
Status merken 395
Stechuhr 429
Test 388
Transponder 386
Zugriffssteuerung 389

RF-Link-Module 339

RTC

Binärkodiertes Dezimalformat 424
Digitaluhr 425
IC 419
Modul anschließen 420
Stechuhr 429
Uhr stellen 424
Uhrzeit- und Datumsformat 424

S

SAM3X8E 272

Schaltpläne 54

Bauteilsymbole 54
Konstruktionsprogramm Fritzing 58
Leitungen 57

Schieberegister 123

SCL 402

SDA 402

secondary device 411

Serial.available() 111

Serial.flush() 111

Serieller Monitor

Autoscroll 105
Daten an den Arduino senden 110
Debugging 108

- Öffnen 105
- Puffer leeren 111
- Starten in setup() 105
- Text senden 106
- Variableninhalt anzeigen 106
- Serieller Port
 - Lage 14
- Servomotoren Siehe Stellmotoren
- setup() 21
- Shields 148
 - Einführung 15
 - Freertronics EtherMega 277
 - GPS-Shield 325
 - IP-Adresse 442
 - LCD- und Tastenfeld-Shield von Freertronics 328
 - microSD-Karten-Shield 160
 - Motor-Shield 298
 - ProtoScrewShield 429
 - ProtoShield 148
 - Stapeln 16
- Sicherheit
 - Allgemeine Hinweise 8
 - Netzstrom steuern 51
- Siebensegmentanzeigen
 - Aufbau 133
 - Einstellig 136
 - Mehrstellig 138
 - Pinbelegung 134
 - Steuern 134
- Signifikantestes Bit 121
- Sketche
 - Bibliotheken 155
 - Dateinamenerweiterung 21
 - Debugging 108
 - #define 69

- Definition 19
- Eigene Funktionen schreiben 97
- Entscheidungen 69, 108
- Fehlermeldungen 24
- for-Schleifen 43
- Hochladen 24
- Hochladen ohne USB-Anschluss 268
- if 69
- if-then-else 70
- Kommentare 20
- Konstanten 69
- LEDs blinken lassen 19
- loop() 22
- setup() 21
- Sketch 449
- Speichern 20
- Überprüfen 23
- Variablen 42
- Vergleichsoperatoren 70
- Zeitmessung 167
- Slave 402
- Spannung 29
- Spannungsregler 263
- Spannungsteiler 86
- SPI
 - Bibliothek 412
 - Bitreihenfolge 412
 - Daten an SPI-Geräte senden 413
 - Spannung 412
- SRAM 273
- SS 413
- Steckplatinen 36
 - Arduino-Nachbau auf Steckplatine 261
- Stellmotoren
 - Anschließen 283

- Bewegen 283
- Eigenschaften 282
- Über Teleduino steuern 447
- Zeigerthermometer 284

- Stoppuhr 169
- Stromstärke 29

T

Tastenfelder

- Anschließen 240
- Einführung 239
- Programmierung 240
- Schloss 243

- Tastverhältnis 45, 90

- TDK PS1240 89

Teleduino 447

- Befehls-URL 450
- Bibliothek 449
- Schlüssel 448
- Sketch 449
- Statuscodes 450

Temperatur

- Aufzeichnung auf microSD-Karte 164
- Verlaufskurve 212
- Werte umrechnen 95

Temperatursensor 92

- Daten auf Webseite anzeigen 439

- TinyGPS 336

- TIP120 288

- TMP36 92

Touchscreen

- Anschließen 248
- Berührungspunkt ermitteln 249
- Drei-Zonen-Touch-Schalter 256
- Einführung 247
- Helligkeitsregler für LED 256

- Kartierung 251
- Transistoren 47, 56
 - Darlington-Transistoren 287
- Transponder Siehe RFID
- Trimmer 88
- Twitter
 - Bibliothek 445
 - Statuscodes 446
 - Token 444
 - Tweets vom Arduino senden 444

U

- Ultraschallsensoren 316
- Upload 24
- USB
 - Anschluss 13
- UTC 333

V

- Variablen
 - Auf LCD anzeigen 199
 - Boolesche Variablen 71
 - Bytevariablen 122
 - Dauerhaft im EEPROM speichern 392
 - Einführung 42
 - Fließkommavariablen 84
 - Inhalt anzeigen 106
 - Inhalt drahtlos senden 345
 - int 112
 - long 112
- Vergleichsoperatoren 70, 85
- Verify 23
- VirtualWire 341
- vw_send() 345

W

- while 108

Widerstände

Begrenzungswiderstand 35

Chipwiderstände 32

Digitalpotenziometer 413

Einführung 30

Farbcode 31

Leistungsangabe 33

Ohmsches Gesetz 36

Potenziometer 87

Pulldown-Widerstand 64

Regelbare Widerstände 87

Schaltplansymbol 55

Spannungsteiler 86

Trimmer 88

Wire 403

write() 403

Z

Zeitmessung

Digitaluhr 425

GPS-Uhr 331

RTC-Module 419

Stechuhr 429

Zeitzone 333

Zufallszahlen 116