

Inhaltsverzeichnis

Inhaltsverzeichnis	1
Installation	9
Download der Treiber und der Entwicklungsumgebung	9
Download der Fritzing-Software	9
Entwicklungsumgebung	9
Ein erstes Programm	11
Sketch	12
Kompilieren eines Programms und Übertragung auf den Arduino	12
Programmstruktur	12
setup()	13
loop()	13
Mikroprozessor	14
Strukturen von Mikroprozessoren	14
Von Neumann Architektur	14
Aufbau der CPU	14
Harvard Architektur	14
Das Arduino-UNO-Board	15
Einfache Ein- / Ausgabe Teil 1	16
Steckboard	16
Erklärung Steckboard:	16
Strom, Spannung, Widerstand und Leistung	16
Erklärung Strom:	16
Erklärung Spannung:	16
Erklärung Widerstand:	17
Erklärung Leistung:	17
Zusammenhang zwischen Strom, Spannung und Widerstand:	17
Berechnung der Leistung:	17
Erklärung Reihen- und Parallelschaltung	17
Reihenschaltung:	18
Parallelschaltung:	18
LED ansteuern	19
LED	19

Programmierung Teil 1.....	21
Variablen:	21
Variablentypen (hier nur die im Skript verwendeten):	21
Verwendung von Variablen:.....	21
Rechnen / Vergleichen von Variablen.....	22
Arithmetische Operatoren	22
Vergleichs Operatoren	22
Boolsche Operatoren	22
Bedingte Entscheidungen (if - else).....	22
Einfache Ein-/ Ausgabe Teil 2	23
Taster abfragen	23
Pullup-Widerstand.....	23
Einlesen eines Tasters / Programmierung.....	24
"Entprellen" eines Tasters / Flankenabfrage.....	24
Entprellen eines Tasters	24
Flankenabfrage.....	24
Statische Variablen.....	25
Kontrollausgaben / Serielle Schnittstelle Teil 1.....	25
Serieller Monitor:	26
Analoge Werte einlesen	26
Analoge Eingänge:.....	26
Umrechnung des eingelesenen analogen Wert.....	27
Erklärung Potentiometer:.....	28
Serieller Plotter:	28
Analoge Werte ausgeben	29
Analoge Ausgabepins:	29
Puls-Weiten-Modulation (PWM):.....	29
Ansteuern der analogen Ausgabepins:	30
Dreiecksspannung:	30
Sinusspannung:.....	31
Programmieren Teil 2	32
Schleifen	32
Kopfgesteuerte Schleife	32
Struktur der kopfgesteuerten Schleife:	32

Erklärung der kopfgesteuerten Schleife:.....	32
Fußgesteuerte Schleife.....	32
Struktur der fußgesteuerten Schleife:.....	32
Erklärung der fußgesteuerten Schleife:.....	32
Zählschleife.....	32
Struktur der Zählschleife:	32
Erklärung der Zählschleife:	32
Funktionen.....	34
Struktur von Funktionen	34
Zugriff auf Funktionen	34
Überladen von Funktionen.....	34
Gültigkeitsbereich von Variablen	35
Bibliotheken	35
Sensoren.....	36
Fotowiderstand	36
Auswerten der Helligkeit: Schaltung, Programm	36
Temperaturfühler.....	37
Tiltsensor	37
Aktoren.....	38
Mehrfarbige LED.....	38
Summer	39
Servo.....	39
Transistor.....	41
Relais	42
Digitaltechnik.....	43
Digitaltechnische Grundschaltungen	43
UND / AND.....	43
ODER / OR.....	43
NICHT / NOT	44
Exklusiv-ODER.....	44
Negierte Funktionen.....	44
NAND	44
NOR.....	44
Komplexere digitale Schaltungen:.....	45

Halbaddierer.....	45
Register direkt ansprechen	45
Interrupts.....	47
1. Polling	47
2. Interrupts.....	47
attachInterrupt(interruptnummer, funktion, modus)	48
Interruptbelegungen des Arduino Uno	48
Modus der Erkennung des Interrupts:	48
detachInterrupt(interruptnummer)	49
Fritzing	49
Arduino Simulator	49
Informationen / Beispiele.....	50
Hinzufügen einer Bibliothek.....	50
Ansteuern eines Schrittmotors	50
Das Prinzip eines Schrittmotors	50
Arten von Schrittmotoren	51
Bipolarer Schrittmotor	51
Unipolarer Schrittmotor	52
L293 D	53
Ansteuerung eines unipolaren Schrittmotors mit dem Arduino.....	54
Ansteuerung eines bipolaren Schrittmotors mit dem Arduino.....	55
Der Quellcode des Vollschrittbetriebes:	56
Motorshields	57
Shield mit dem L293D	57
Zeitmessung	58
Zeitmessung eines Impulses.....	59
Entfernung messen mit Ultraschall.....	59
Schaltung zur Entfernungsmeßung:	60
Kommentierter Programmcode zur Entfernungsmeßung:	60
Infrarot	61
Infrarotsignale empfangen.....	61
Anschlussplan des Infrarotempfängers.....	61
Programmierung des Infrarotempfängers	62
Quellcode zum Auslesen einer Fernbedienung.....	62

Ansteuern von LCD-Displays.....	62
Anschlussplan für das LCD-Display:.....	63
Beispielprogramm für das LCD-Display:	63
Befehlsübersicht der Biliothek: <code>LiquidCrystal.h</code>	64
SPI-Bus.....	65
Programmierung eines digitalen Potentiometers MCP 4151	66
Ansteuern einer LED-Matrix oder von mehreren 7-Segment-Anzeigen	68
MAX 7219	68
Programmierung des MAX 7219	69
Programm zur Absteuerung einer LED-Matrix	70
Schaltplan zur Ansteuerung einer 5x7-LED-Matrix mit dem MAX 7219	71
Eine Sieben-Segment-Anzeige.....	71
Der I ² C-Bus.....	72
Das Busprinzip:.....	72
PCF8591.....	73
Schaltplan Analogmessung mit einem PCF 8591	76
Beschleunigungsmesser und Gyroskop: MPU 6050 auf dem Modul GY-521	79
Beschaltung Arduino und MPU6050	80
LCD-Display über I ² C	81
Schaltung zum LCD-Display über I ² C:	82
Programm für den Arduino mit einem I ² C-LCD-Display	82
Wetterdaten	83
Luftdruck, Temperatur und Höhe mit dem bmp180 messen	83
Luftfeuchtigkeit mit einem DHT11 messen	86
Shields	87
Netzwerkshield.....	87
Themen.....	91
Bewegungsmelder PIR.....	91
RFID (Technologie vereinfacht dargestellt!).....	92
Arduino im WLAN.....	95
CC3000.....	95
ESP8266.....	98
Arduino und Bluetooth.....	99
Umbenennen des Bluetoothmoduls	102

Ein Gerät bauen.....	103
Bootloader.....	103
1. Interner oder externen Takt	103
Arduino als Programmer	105
2. Installation des Bootloaders.....	106
Änderung der Datei avrdude.conf.....	107
AtMega 328 P mit externem Takt	108
3. Programmierung des Atmega mittels FTDI:	108
RGB-Sensor TCS 230	109
Zeitermittlung mit dem DCF-77 – Protokoll.....	112
GPS	113
LCD-Keypad-Shield	115
LED&Keys.....	117
Aufgabensammlung	118
Aufgaben: Einfache Ein- / Ausgabe	118
Aufgabe 1 Einfache Ein- / Ausgabe.....	118
Aufgabe 2 Einfache Ein- / Ausgabe.....	118
Aufgabe 3 Einfache Ein- / Ausgabe.....	118
Aufgabe 4 Einfache Ein- / Ausgabe.....	118
Aufgabe 5 Einfache Ein- / Ausgabe.....	118
Aufgabe 6 Einfache Ein- / Ausgabe.....	118
Aufgabe 7 Einfache Ein- / Ausgabe.....	118
Aufgabe 8 Einfache Ein- / Ausgabe.....	118
Aufgabe 9 Einfache Ein- / Ausgabe.....	118
Aufgabe 10 Einfache Ein- / Ausgabe.....	119
Aufgaben Kontrollausgaben.....	119
Aufgabe 1 zu Kontrollausgaben.....	119
Aufgabe 2 zu Kontrollausgaben.....	119
Aufgabe 3 zu Kontrollausgaben.....	119
Aufgaben Einlesen analoger Werte.....	119
Aufgabe 1 zum Einlesen analoger Werte:	119
Aufgabe 2 zum Einlesen analoger Werte:	119
Aufgabe 3 zum Einlesen analoger Werte:	119
Aufgaben zu den analogen Ausgabepins:	120

Aufgabe 1 zu den analogen Ausgabepins:.....	120
Aufgabe 2 zu den analogen Ausgabepins (Dimmer):	120
Aufgabe 3 zu den analogen Ausgabepins (Signalgenerator I):.....	120
Aufgabe 4 zu den analogen Ausgabepins (Signalgenerator II):.....	120
Aufgaben zum LDR	120
Aufgabe 1 zum LDR.....	120
Aufgabe 2 zum LDR (Nachtlicht).....	120
Aufgaben zum Temperatursensor.....	120
Aufgabe 1 zum Temperatursensor.....	120
Aufgabe 2 zum Temperatursensor.....	121
Aufgaben zum Tiltsensor.....	121
Aufgabe 1 zum Tiltsensor	121
Aufgabe 2 zum Tiltsensor	121
Aufgabe 3 zum Tiltsensor (Pedometer).....	121
Aufgabe 4 zum Tiltsensor (Pedometer).....	121
Aufgabe zu Aktoren.....	121
Aufgaben zur RGB-LED	121
Aufgabe 1 zur RGB-LED.....	121
Aufgabe 2 zur RGB-LED.....	121
Aufgaben zum Summer	121
Aufgabe 1 zur Summer	121
Aufgabe 2 zur Summer	121
Aufgaben zum Servo.....	122
Aufgabe 1 zur Servo.....	122
Aufgabe 2 zur Servo.....	122
Aufgaben zum Transistor	122
Aufgabe 1 zum Transistor.....	122
Aufgabe 2 zum Transistor.....	122
Aufgaben zum Relais	122
Aufgabe 1 zum Relais	122
Aufgaben zur Digitaltechnik	122
Aufgabe 1 zur Digitaltechnik	122
Aufgabe 2 zur Digitaltechnik	122
Aufgabe 3 zur Digitaltechnik	123

Aufgabe 4 zur Digitaltechnik	123
Aufgaben zu Interrupts.....	123
Aufgabe 1 zu Interrupts.....	123
Aufgabe 2 zu Interrupts.....	123
Aufgabe 3 zu Interrupts.....	123
Aufgabe 4 zu Interrupts.....	123
Aufgaben zum Schrittmotor	123
Aufgabe 1 zum Schrittmotor	123
Aufgabe 2 zum Schrittmotor	124
Aufgabe 3 zum Schrittmotor	124
Aufgaben zur Zeit- / Entfernungsmessung.....	124
Aufgabe 1 zur Zeit - / Entfernungsmessung	124
Aufgaben zum IR-Empfänger.....	124
Aufgabe 1 zum IR-Empfänger.....	124
Aufgabe 2 zum IR-Empfänger.....	124
Aufgabe 3 zum IR-Empfänger.....	124
Aufgaben zum LCD-Display.....	124
Aufgabe 1 zum LCD-Display.....	124
Aufgabe 2 zum LCD-Display.....	124
Aufgabe 3 zum LCD-Display.....	124
Aufgabe 4 zum LCD-Display.....	125
Aufgaben zum SPI-Bus.....	125
Aufgabe 1 zum SPI-Bus	125
Aufgaben zum MAX 7219	125
Aufgabe 1 zum MAX 7219	125
Aufgabe 2 zum MAX 7219	125
Aufgaben zum I ² C-Bus	125
Aufgabe 1 zum I ² C-Bus	125
Aufgabe 2 zum I ² C-Bus	125
Aufgabe 3 zum I ² C-Bus	125
Aufgabe 4 zum I ² C-Bus	125
Aufgaben zum Ethernetshield	125
Aufgabe 1 zum Ethernetshield	125
Aufgaben zum Bewegungsmelder PIR:	126

Aufgabe 1 zum Bewegungsmelder.....	126
Aufgaben zu RFID:	126
Aufgabe 1 zu RFID.....	126
Aufgabe 2 zu RFID.....	126
Aufgaben zum RGB-Sensor TCS320:.....	126
Aufgabe 1 zu RFID.....	126
Aufgabe 2 zu RFID.....	126
Aufgaben zum DCF-77:.....	126
Aufgabe 1 zu DCF-77	126
Aufgabe 2 zu DCF-77	126
Aufgaben zum GPS-Modul:	126
Aufgabe 1 zum GPS-Modul.....	126
Aufgabe 2 zu GPS-Modul.....	127
Aufgaben zum LCD-Keypad-Shield:	127
Aufgabe 1 zum LCD-Keypad-Shield	127
Projektideen	127
Kapazitätsmessung eines Kondensators	127
Widerstandsmessung	127
LED-Tester	127
Quellen:	127

Installation

Download der Treiber und der Entwicklungsumgebung:

Laden Sie die Entwicklungsumgebung von: <http://arduino.cc/en/main/software> herunter und installieren Sie die Software:

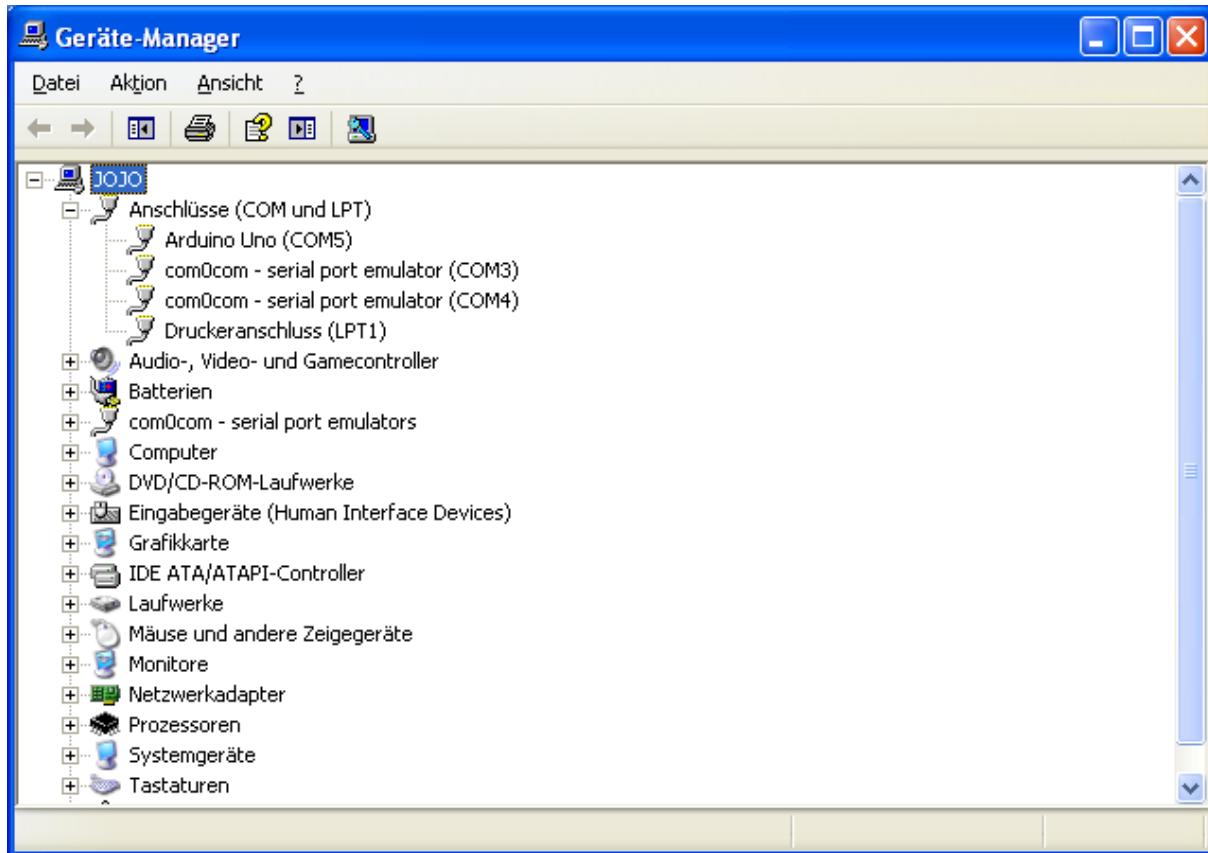
Download der Fritzing-Software von: <http://fritzing.org/download/>

Hat man die Software installiert, so schließt man das Arduino-Board an einen USB-Anschluss an. Das System sollte nun den Treiber installieren, auch wenn die Kompatibilitätsprüfung von Windows mosert. Nun startet man die Entwicklungsumgebung vom Desktop aus.

Entwicklungsumgebung

Als erstes muss der serielle Port eingestellt werden, an dem der Arduino angeschlossen ist. Auf dem Board befindet sich ein Chip: FT 232. Dieser Chip stellt einen seriellen Port über den gewählten USB-Anschluss zur Verfügung. Damit die Kommunikation von der Entwicklungsumgebung zum Board

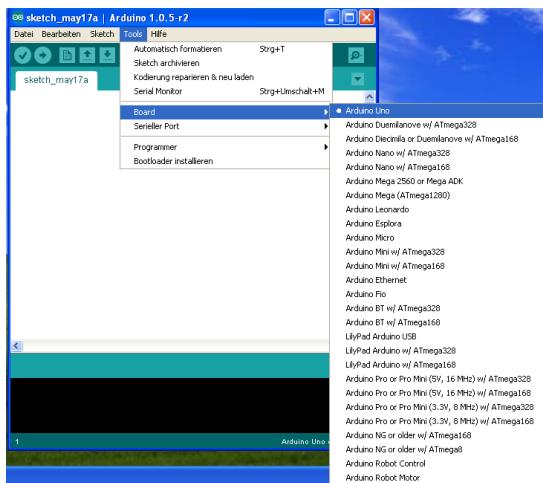
funktioniert, muss dieser richtig gewählt werden. Im Gerätemanager des Systems kann der entsprechende Port identifiziert werden (hier: COM5):



Diesen wählt man in der Entwicklungsumgebung unter dem Menüpunkt: Tools→Serieller Port aus:

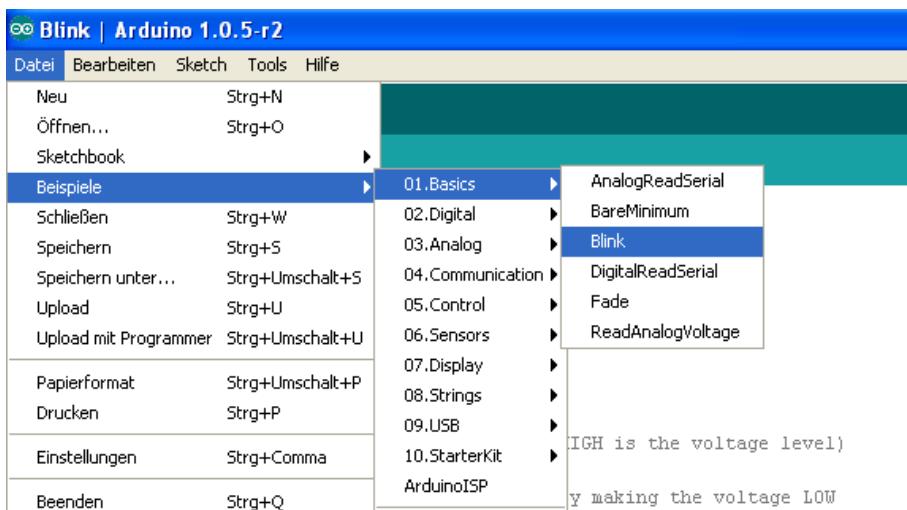


Im Menüpunkt Tools→Board ist nun noch der Arduino UNO zu wählen (es sei denn, sie verwenden ein anderes Board, welches unterstützt wird):



Ein erstes Programm

Nun kann es auch schon losgehen. Als einfachen Einstieg wählen wir ein Programm, welches bei den mitgelieferten Beispielen zu suchen ist. Das Programm heißt Blink:



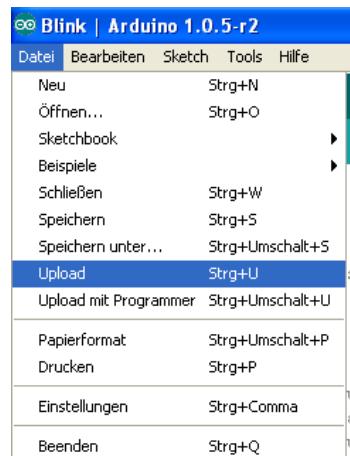
Sketch

Programme für den Arduino werden als Sketch bezeichnet. Lädt man ein fertiges Programm aus einer Datei, so wird immer ein neuer Sketch geöffnet.

Kompilieren eines Programms und Übertragung auf den Arduino:

Ist das Programm geschrieben, kann es nur kompiliert werden. Hierzu wird der Menüpunkt: Sketch → Überprüfen / Kompilieren ausgeführt. Der Compiler gibt etwaige Fehlermeldungen aus oder kompiliert das Programm komplett.

Einfacher geht es mit der Option: Datei → Upload:



Hierbei wird das Programm kompiliert und anschließend auf den Arduino übertragen.

Hat man dies geschafft, so blinkt auf dem Board eine LED.

Alternativ zur Vorgehensweise findet man in der Entwicklungsumgebung zwei grafische Symbole:



:kompiliert das Programm



:überträgt das Programm an den Arduino.

Programmstruktur

Das erste Programm Blink sieht wie folgt aus:

```
int led = 13;

void setup() {
    pinMode(led, OUTPUT);
}

void loop() {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
}
```

Das Programm besteht im Wesentlichen aus drei Bereichen.

Im oberen Teil werden globale Variablen definiert und gegebenenfalls mit Werten belegt. Dies bedeutet, dass man von jeder Stelle des gesamten Programms auf den Inhalt dieser Variablen zugreifen kann.

setup()

Die Funktion `setup()` wird beim Starten des Arduinos genau einmal ausgeführt. Hier werden die Einstellungen für die einzelnen Pins des Arduino vorgenommen. In diesem Fall wird der Pin mit der Nummer 13 als Ausgang definiert. Die zugehörige Funktion hierzu lautet: `pinMode()`. Die bekommt zwei Werte übergeben: die Nummer des Pins und die Eigenschaft. Die Nummer des Pins ist hier `led`. Da `led` aber eine globale Variable ist, nimmt die den Wert 13 an. Die Eigenschaft ist: `OUTPUT`. Man wird später noch andere Funktionen von Pins kennenlernen.

loop()

Die Funktion `loop()` ist die eigentliche Programmfunktion des Arduinos. Sie wird nach `setup()` aufgerufen und läuft automatisch in einer Endlosschleife. In ihr werden die eigentlichen Aktivitäten des Boards programmiert.

Zur Erklärung des Programms Blink:

Das Programm schreibt zuerst den Wert `HIGH` auf den Pin mit der Nummer 13 (`led`). Die Leuchtdiode leuchtet.

```
digitalWrite(led, HIGH);
```

Nun wartet das Programm 1000ms.

```
delay(1000);
```

Nun wird der Wert an Pin 13 auf `LOW` gesetzt. Die Leuchtdiode geht aus.

```
digitalWrite(led, LOW);
```

Das Programm wartet wieder 1000ms.

```
delay(1000);
```

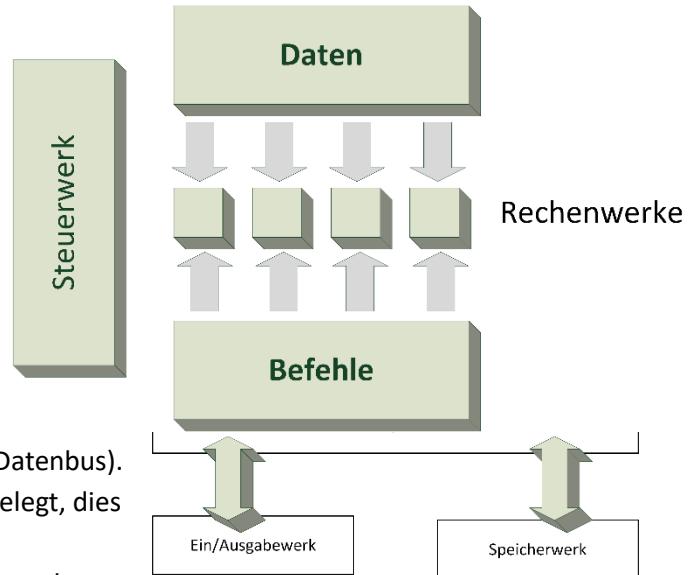
Die Funktion `loop()` wird wiederholt.

Mikroprozessor

Strukturen von Mikroprozessoren

Von Neumann Architektur

Diese Architektur, benannt nach dem Mathematiker John von Neumann beschreibt den Aufbau eines Mikroprozessorsystems. Es besteht aus einer CPU (Control Processing Unit), einem Speicherwerk und einem Ein-/Ausgabewerk. Diese werden über einen Bus angesteuert (Steuerbus) und mit Daten versorgt (Datenbus). Der Steuerbus ist ursprünglich unidirektional ausgelegt, dies bedeutet, es können nur Geräte von der CPU aus angesteuert werden, aber nicht umgekehrt. Der Datenbus ist bidirektional, da man auch Daten von den Geräten lesen möchte. Innerhalb der CPU befinden sich ein Rechenwerk und ein Steuerwerk. Das Rechenwerk verarbeitet Daten, das Steuerwerk sorgt dafür, dass die richtigen Geräte über den Steuerbus angesprochen werden.

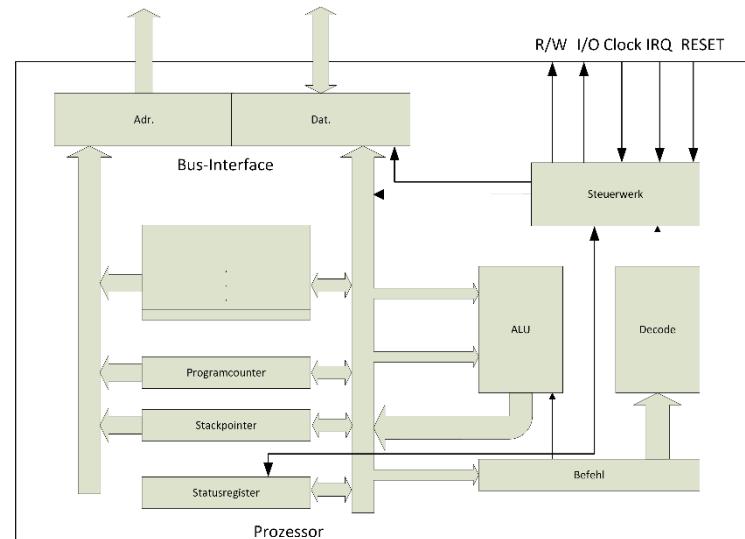


Aufbau der CPU

Nachfolgend ist der vereinfachte Aufbau einer CPU dargestellt.

Wichtig sind:

In der ALU (Arithmetik Logic Unit) wird gerechnet (binäre UND und ODER, sowie Additionen und Subtraktionen). Der Programmzähler sorgt dafür, dass immer der richtige Programmbebefl ausgeführt wird. Im Stack werden Daten gespeichert, das Prinzip ist LiFo. Der Befehlsdecoder sorgt dafür, dass alle



eingespeicherten Befehle eines Programms richtig ausgeführt werden.

Harvard Architektur

Der verwendete Prozessor ATMEGA 328P ist nach dieser Architektur aufgebaut:

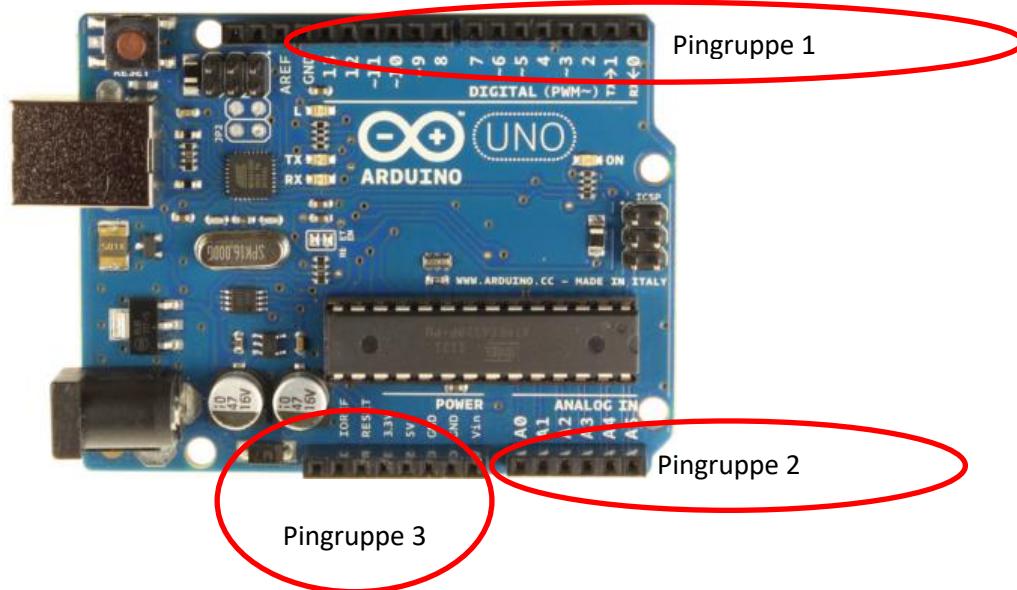
Der Unterschied zur von-Neumann-Architektur ist der, dass Datenspeicher und Befehlsspeicher sowohl logisch als auch physisch getrennt sind. Die logische Trennung erfolgt über unterschiedliche Adressräume, die physische Trennung durch zwei unabhängige Speicher. Dadurch ist es bei dieser Architektur möglich, Daten und Befehle gleichzeitig zu laden.

Die Vorteile dieser Architektur bestehen in der Geschwindigkeit, da nur ein Buszyklus für das Laden eines Befehls und von Daten erforderlich ist. Weiter entsteht durch die getrennten Speicher ein

Sicherheitsvorteil, da der Befehlsspeicher im Betrieb nur lesbar ist, der Datenspeicher wird als RAM ausgelegt. Man kann also durch das Beschreiben von Daten keine Befehle zerschießen.

Hier liegt aber auch der Nachteil dieser Architektur: wird Befehlsspeicher nicht benötigt, so ist dieser im Betrieb weiter nicht verwendbar.

Das Arduino-UNO-Board



Das Board beinhaltet den Mikroprozessor (ATmega328), benötigte elektronische Bauteile, einen USB-Anschluss, einen Taster, einige LEDs sowie diverse Anschlussmöglichkeiten.

Das Board wird über den USB-Bus mit Betriebsspannung versorgt, so dass erst einmal kein externes Netzteil erforderlich ist. Schließt man später externe Geräte mit höherem Strombedarf an (Schrittmotoren etc.), so gibt es die Möglichkeit, über ein externes Netzteil das Board sowie die angeschlossenen Geräte mit Spannung und ausreichender Leistung zu versorgen.

Das Board besitzt 13 digitale Ein-/Ausgabepins (Pingruppe 1), die je nach Konfiguration in der Software als Eingabepin oder Ausgabepin konfiguriert werden können. Einige dieser Pins haben das Tildezeichen (~) vor der Pinnummer stehen. Diese Pins können über Pulsweitenmodulation (PWM, hierzu später mehr) auch als analoge Ausgabepins dienen.

An Pin 13 ist die auf dem Board vorhandene LED angeschlossen, PIN 0 und 1 dienen der Kommunikation über die serielle Schnittstelle.

Weiterhin gibt es sechs analoge Eingabepins (Pingruppe 2). Hier können analoge Spannungen eingelesen werden.

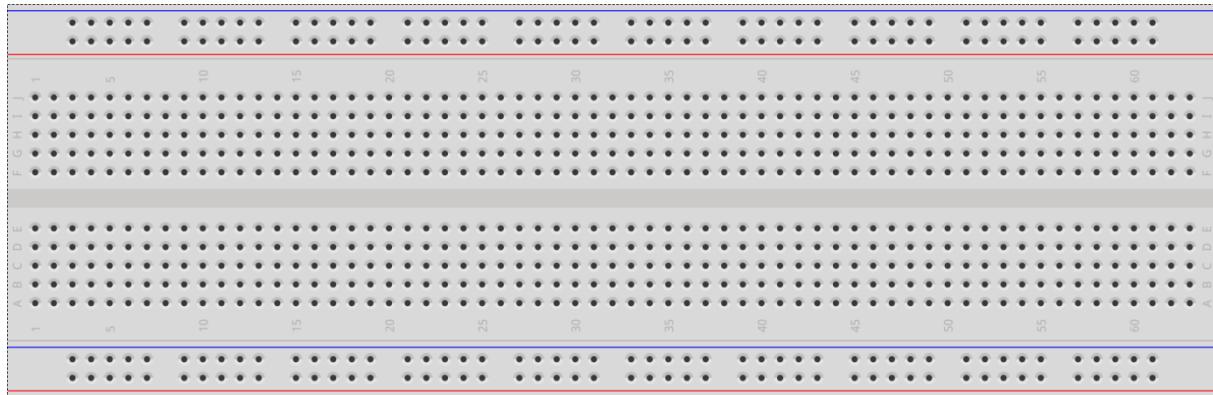
Über die Pingruppe 3 können die elektrische Masse (GND) sowie die internen Spannungen 5V und 3,3V sowie, falls angeschlossen, die externe Spannung abgenommen werden.

Einfache Ein- / Ausgabe Teil 1

Bevor man mit den ersten Aufbauten von Schaltungen beginnt, muss man sich mit der Steckplatine und (wohl oder übel) mit etwas elektrotechnischer Theorie beschäftigen:

Steckboard

Die mitgelieferte Steckplatine sieht wie folgt aus:



Erklärung Steckboard:

Das Board hat im **oberen und unteren Bereich** eine Reihe von Fünfergruppen an Anschläßen. Diese Fünfergruppen sind horizontal kurzgeschlossen. Sie eignen sich ideal, um eine Spannung (Pluspol an die rote Reihe, Minuspol an die blaue Reihe) zu verteilen.

Der **mittlere Bereich** ist in zwei identische Bereiche unterteilt. Hier liegen jeweils nummeriert Fünfergruppen nebeneinander. Jede Fünfergruppe ist für sich kurzgeschlossen, die Gruppen gegeneinander allerdings nicht. In diesen Bereichen baut man gewöhnlich seine Schaltungen auf.

Wichtig:

Beachten Sie immer beim Aufbau von Schaltungen, ob nicht Bauteile unwirksam sind, da sie fälschlicher Weise durch eine Gruppe kurzgeschlossen werden. Prüfen Sie notfalls Ihren Aufbau vor dem Einschalten mit einem Messgerät auf Richtigkeit.

Strom, Spannung, Widerstand und Leistung¹

Erklärung Strom:

Als Strom kann man die Elektronen bezeichnen, die sich in einer Schaltung bewegen. Bewegen sich viele Elektronen, so ist die Stromgröße höher, bewegen sich weniger Elektronen, so ist die Stromgröße kleiner. Die physikalische Einheit des Stroms ist Ampere (A), wobei wir hier uns sicherlich eher im Bereich von sehr kleinen Strömen aufhalten werden. Daher werden wohl eher Milliampere (mA, ein Tausendstel von einem Ampere) verwendet. Das technische Symbol des Stroms ist I.

Erklärung Spannung:

Damit Elektronen sich bewegen, benötigen sie einen Grund. Dieser stellt die elektrische Spannung dar. Die Spannung kann als Missverhältnis an zwei verschiedenen Stellen einer Schaltung angesehen

¹ Auch wenn mich mancher Elektrotechnik / Physiker hier gerne lynchen möchte, reichen die folgenden Erklärungen für den Hausgebrauch doch sicher aus.

werden. Dieses Missverhältnis (man spricht auch von dem Potenzial) fordert die Elektronen auf, sich zu bewegen. Die physikalische Einheit der Spannung ist Volt (V). Das technische Symbol ist U.

Erklärung Widerstand:

Der elektrische Widerstand bezeichnet die Eigenschaft des Materials, durch das sich die Elektronen bewegen müssen. Kommen nur wenige Elektronen voran, so ist der Widerstand groß und umgekehrt. Widerstand wird in OHM (Ω) gemessen, das technische Symbol ist R.

Erklärung Leistung:

Die elektrische Leistung ist eine Größe, die Auskunft über die Energie gibt, welche ein Bauteil, eine Schaltung oder ein Gerät verbraucht. Die Leistung wird in Watt (W) angegeben, das technische Symbol ist P.

Wichtig:

Elektrotechnische Bauteile haben eine maximale Leistung. Wird diese überschritten, so kann das Bauteil zerstört werden!

Zusammenhang zwischen Strom, Spannung und Widerstand:

Dieser Zusammenhang wird durch das ohmsche Gesetz dargestellt:

$$I = \frac{U}{R}$$

Hierbei bezeichnet U die an einer Schaltung angelegte Spannung, R den elektrischen Widerstand der Schaltung und I den daraus resultierenden Strom, der durch die Schaltung fließt. Das bedeutet, dass sich der Strom in einer Schaltung dann erhöht, wenn man die angelegte Spannung erhöht oder den Widerstand der Schaltung verringert. Das Gesetz wird oft in der Form: $U = RI$ dargestellt, allerdings ist die obige Form einsichtiger, da der elektrische Strom die abhängige Größe darstellt.

Berechnung der Leistung:

Die elektrische Leistung berechnet sich wie folgt: $P = U \cdot I$.

Erklärung Reihen- und Parallelschaltung

Schaltungen bestehen aus mehreren Bauteilen, die in Reihenschaltungen und Parallelschaltungen kombiniert verbunden sind. Um auszurechnen, welcher Strom in einer Schaltung fließt, oder welche Spannung an einem Bauteil abfällt, muss man die folgenden elektrotechnischen Zusammenhänge wissen:

Reihenschaltung:

Zwei Widerstände werden in Reihe verschaltet. An die gesamte Schaltung wird eine Spannung von 6V angelegt. Wie hoch ist der Strom, der durch die einzelnen Bauteile fließt? Welche Spannung fällt an den einzelnen Bauteilen ab?



Gesetzmäßigkeiten:

In einer Reihenschaltung ist die Stromstärke in allen Bauteilen gleich.

In einer Reihenschaltung addieren sich die Widerstände zum Gesamtwiderstand.

In einer Reihenschaltung teilt sich die Spannung entsprechend der Größe der einzelnen Bauteile auf!

Was bedeutet dies?

Der Gesamtwiderstand der Schaltung berechnet sich aus der Addition der einzelnen Widerstände, also: $R_{ges} = R_1 + R_2 = 1k\Omega + 2k\Omega = 3k\Omega$

Nach dem ohmschen Gesetz ist der Strom, der durch die gesamte Schaltung fließt: $I_{ges} = \frac{U}{R_{ges}} = \frac{6V}{3k\Omega} = 5mA$

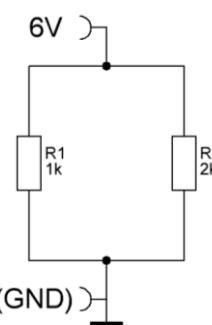
In einer Reihenschaltung ist die Stromstärke in allen Bauteilen gleich $\rightarrow I_1 = I_2 = I_{ges} = 5mA$.

In einer Reihenschaltung teilt sich die Spannung entsprechend der Größe der einzelnen Teile auf. Dies bedeutet, dass 1 Teil von 6V am ersten Widerstand und 2 Teile von 6V am zweiten Widerstand abfallen. Also ist: $U_1 = 2V$ und $U_2 = 4V$. Als Formel hierfür kann man die folgende verwenden:

$$U_n = U \cdot \frac{R_n}{R_{ges}}$$

Parallelschaltung:

Zwei Widerstände werden parallel verschaltet. An die gesamte Schaltung wird eine Spannung von 6V angelegt. Wie hoch ist der Strom, der durch die einzelnen Bauteile fließt? Welche Spannung fällt an den einzelnen Bauteilen ab?



Gesetzmäßigkeiten:

Die Gesetzmäßigkeiten sind in der Parallelschaltung leider nur bei der Spannung trivial:

Die Spannung ist an allen Bauteilen gleich.

Die Ströme durch die einzelnen Bauteile addieren sich zum Gesamtstrom.

Der Gesamtwiderstand berechnet sich wie folgt: $R_{ges} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$

Keine Angst, die Eigenschaft, dass die Spannung überall gleich ist und das ohmsche Gesetz erleichtern die Berechnung:

Berechnung der Spannungen:

$$U = U_1 = U_2 = 6V$$

Berechnung der Ströme:

$$I_1 = \frac{U}{R_1} = \frac{6V}{1k\Omega} = 6mA ; I_2 = \frac{U}{R_2} = \frac{6V}{2k\Omega} = 3mA ; I_{ges} = I_1 + I_2 = 6mA + 3mA = 9mA$$

Berechnung des Gesamtwiderstands:

$$R_{ges} = \frac{U}{I_{ges}} = \frac{6V}{9mA} = 666,67\Omega$$

So, jetzt geht's aber richtig los.

LED ansteuern

Eine LED auf dem Steckboard soll angesteuert werden. Dies ging im Falle des ersten Programms sehr leicht, eine Außenbeschaltung war nicht nötig. Der Grund dafür ist, dass die Außenbeschaltung der LED auf dem Arduino-Board schon vorhanden ist. Wenn aber außerhalb auf dem Steckboard eine LED angesteuert werden soll, so sind ein paar Vorüberlegungen nötig:

LED

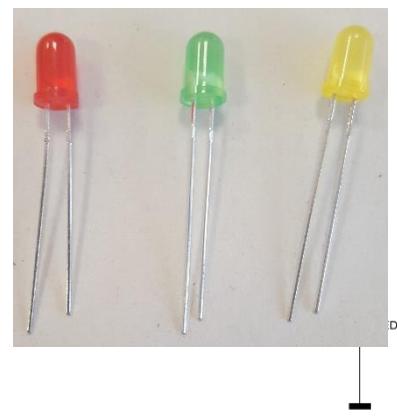
Eine LED (light-emitting diode) ist wie der Name schon sagte eine Diode. Dies bedeutet, dass dieses Bauteil folgende Eigenschaften hat:

Sie lässt Strom nur in einer Richtung durch. Ist dies der Fall, so fällt an ihr eine feste Spannung ab. Fließt zu viel Strom durch die Diode, so wird sie zerstört!

Die Diode hat also verschiedene Anschlüsse. Wie dargestellt, ist das eine Beinchen der LED länger als das andere. Dieser Anschluss heißt Anode und wird an die positivere Spannung angeschlossen, der kürzere Anschluss heißt Kathode und wird an die negativere Spannung angeschlossen. Dem Datenblatt einer solchen Diode entnimmt man, dass sie bei ca. 2,1V arbeitet und einen maximalen Strom von 20mA vertragen kann.

Die digitalen Ausgänge des Arduino liefern bei LOW 0V und bei HIGH 5V, also zu viel Spannung für die LED. Auch können die Ausgänge ca. 40mA liefern, auch zu viel für die Diode. Sie muss also weiter beschaltet werden. Da bei der Parallelschaltung die Spannung an allen Bauteilen gleich ist, kommt diese nicht in Frage. Eine Reihenschaltung ist die richtige Wahl:

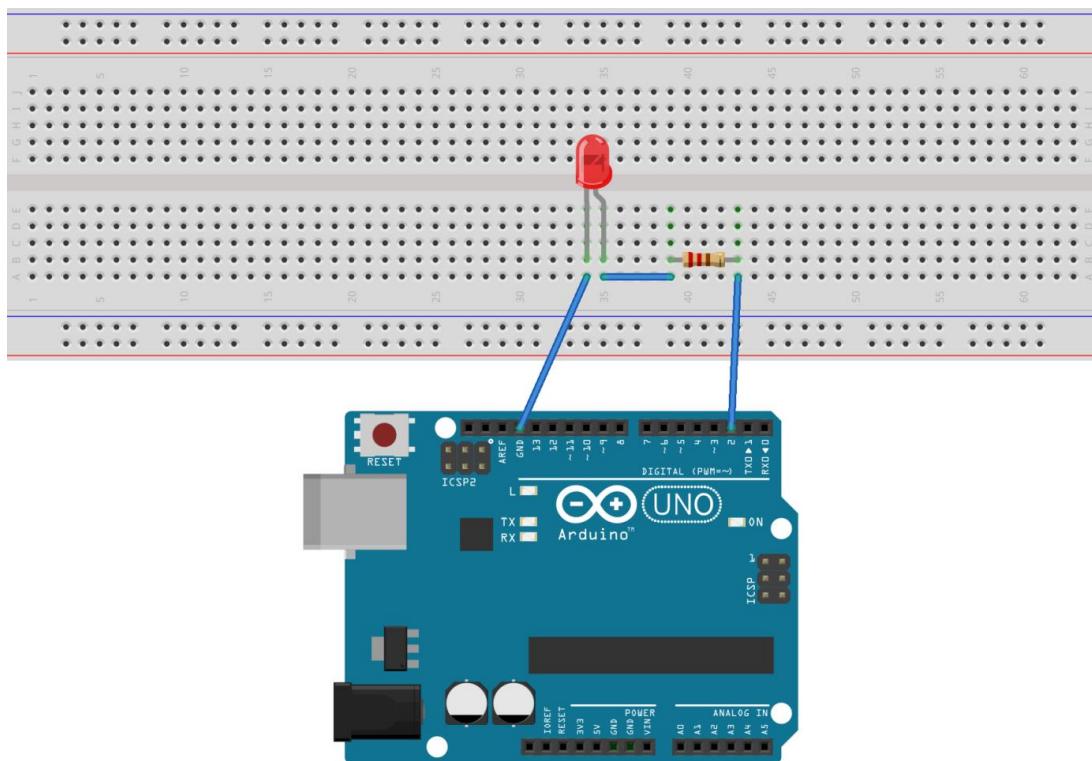
Die Schaltung wird an Pin 2 des Arduino angeschlossen (möglich wären natürlich auch die anderen digitalen Ausgänge, jedoch sind 0 und 1 auch für die serielle Kommunikation zuständig).



Liefert der Arduino LOW-Signal, gibt es nichts zu berechnen, die Spannung beträgt 0V, der Strom 0A, die LED leuchtet nicht.

Liefert der Arduino HIGH-Signal, so liegen an der gesamten Schaltung 5V an. Die LED arbeitet mit 2,1 V, so dass für den Widerstand 2,9V übrigbleiben. Der Strom durch die Diode beträgt maximal 20mA. In der Reihenschaltung sind die Ströme gleich, daher fließen auch durch den Widerstand maximal 20mA. Nach dem ohmschen Gesetz berechnet sich der Widerstand also: $R = \frac{U}{I} = \frac{2,9V}{20mA} = 145\Omega$.

Einen solchen Widerstand ist in unserem Starterkit nicht vorhanden. Schaut man in die Stückliste des Starterkits nach, so entdeckt man 220Ω Widerstände. Da ein höherer Widerstand den Strom nur weiter begrenzt, kann man ihn hier gefahrlos einsetzen. Durch den niedrigeren Strom wird dann die LED ein wenig dunkler leuchten:



Das Programm zur LED:

```
int led = 2;

void setup() {
    pinMode(led, OUTPUT);
}

void loop() {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
}
```

Man sieht, dass im ersten Programm "Blink" nur die Zahl des Ausgabepins auf 2 geändert werden muss.

Bearbeiten Sie nun Aufgabe 2 Einfache Aus-/Eingabe.

Programmierung Teil 1

Der Arduino wird in einer modifizierten Sprache abgeleitet von C++/Java programmiert.

Variablen:

In Variablen können Werte gespeichert werden. Da es unterschiedliche Arten von Werten gibt (Text, Zahlen, Zahlen mit Komma, boolesche Werte etc.) gibt es auch unterschiedliche Variablentypen. Eine Variable bekommt immer einen Variablentyp und einen repräsentativen Namen.

Variablentypen (hier nur die im Skript verwendeten):

Variablentyp	Art des Wertes
int	Ganzzahlige Werte mit Vorzeichen
double	Fließkommazahlen mit Vorzeichen
boolean	Boolscher Wert (true oder false)
char	Ein einzelnes Zeichen
String	Text
void	Ein definiertes Nichts!

Verwendung von Variablen:

Beispiele:

```
int pin = 4;                      //der Wert der Variablen pin vom Typ integer wird
                                    //auf 4 gesetzt

int wert;                          //wert ist eine Integervariable, Deklaration
wert = 5;                           //wert bekommt erstmals einen Wert: Initialisierung

boolean programmLaeuft = true;     //der Wert der boolschen Variable programmLaeuft
                                    //wird auf true gesetzt

char zeichen = 't';                //chars werden immer mit Hochkommata eingeschlossen

String text = "Hallo Arduino";    //String wird groß geschrieben!!!
```

Anmerkung:

Eine Programmierzeile wird immer mit einem **Semikolon** abgeschlossen!

Einzelige Kommentare können über ein **//** eingeleitet werden.

Mehrzeilige Kommentare werden über **/*** eingeleitet und mit ***/** beendet.

Rechnen / Vergleichen von Variablen

Mit Variablen kann man rechnen oder sie mit anderen Werten vergleichen:

Arithmetische Operatoren

Operator	Beispiel	Funktion / Ergebnis
+	int i = 3 + 4;	Addition / i = 7
-	int i = 4 - 3;	Subtraktion / i = 1
*	double d = 3 * 5;	Multiplikation / d = 15
/	double d = 3 / 5;	Division / d = 0.6
%	int i = 10 % 6;	Modulo(Rest einer Division) / i = 4
++	int i = 4; i++;	Inkrementieren / i = 5
--	int i = 4; i--;	Dekrementieren / i = 3

Vergleichs Operatoren

Operator	Beispiel	Funktion / Ergebnis
==	boolean b; i = 3; b = (i == 4);	vergleichen / b = false
!=	boolean b; i = 3; b = (i != 4);	verungleichen / b = true
<	boolean b; i = 3; b = (i < 4);	ist kleiner als / b = true
>	boolean b; i = 3; b = (i > 4);	ist größer als / b = false
<=	boolean b; i = 4; b = (i <= 4);	ist kleiner gleich / b = true
>=	boolean b; i = 3; b = (i >= 4);	ist größer gleich / b = false

Boolsche Operatoren

Operator	Beispiel	Funktion / Ergebnis
&&	boolean b = (3>4) && (3<5);	Logisches Und / b = true
	boolean b = (3<4) (3<5);	Logisches Oder / b = true
!	boolean b; boolean c = true; b = !c;	Logisches Nicht / b = false

Bedingte Entscheidungen (if - else)

Möchte man in einem Programm auf Ereignisse reagieren, so muss man Entscheidungen treffen.

Beim Arduino geschieht dies mittels der if-else-Struktur:

```
if(Bedingung)
{
    Quelltext der ausgeführt wird, wenn Bedingung true war.
}
else
{
    Quelltext der ausgeführt wird, wenn Bedingung false war.
}
```

Beispiel:

```
int i = 0;
boolean iIstKleiner50;
void loop(){
    if(i < 50){
        iIstKleiner50 = true;
    } //end of if
    else{
        iIstKleiner50 = false;
    } //end of else
}
```

Einfache Ein-/ Ausgabe Teil 2

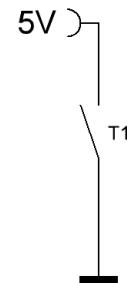
Taster abfragen

Ein Taster ist ein Schalter, der allerdings seinen ursprünglichen Zustand beim Loslassen wieder einnimmt. Einen Taster kann man nicht so einfach an einen digitalen Eingang des Arduino anschließen. Begründung:

Der Eingang liefert keine Spannung, eine externe Spannung ist notwendig!

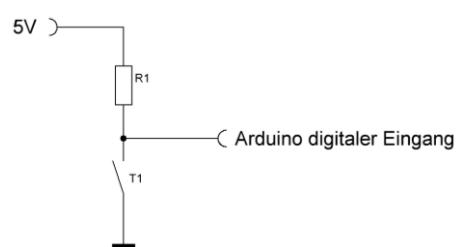
Problematisch ist hier, dass man mittels eines Tasters eine Spannung kurzschließen kann:

Dieser Kurzschluss führt dazu, dass die Spannungsquelle, die die 5V liefert zusammenbricht. Da in unserem Fall dies das Arduino-Board ist, wird die Funktion gestört.



Pullup-Widerstand

Ein Widerstand löst dieses Problem. Ist der Taster nicht gedrückt, so fließt kein Strom. Oberhalb des Tasters werden die 5V abgenommen (Erklärung: Durch den Widerstand fließt kein Strom, also liegt am Widerstand nach dem ohmschen Gesetz keine Spannung an). Wird der Taster gedrückt, so fließt der Strom durch den Widerstand. Der digitale Eingang des Arduinos liegt damit auf Masse, also 0V.



Achtung: Durch interne Beschaltung auf dem Arduino-Board interpretiert der Arduino 5V als LOW-Signal und 0V als HIGH-Signal. Dieses Verhalten nennt man LOW-Active.

Der Widerstand heißt Pullup-Widerstand und ist möglichst groß zu wählen. Dadurch wird der Strom durch ihn klein und belastet das Arduino-Board wenig.

Einlesen eines Tasters / Programmierung

Zum Einlesen eines Tasters muss ein digitaler Pin als Eingang definiert werden. Über die Funktion `readDigital()` bekommt man den Zustand dieses Eingangs zurückgegeben und kann ihn in einer boolschen Variablen speichern:

```
int taster = 8;
boolean tasterGedrueckt;

void setup() {
    pinMode(taster, INPUT);
}

void loop() {
    tasterGedrueckt = digitalRead(taster);
    if(tasterGedrueckt == false){
        // mach irgendwas der Taster wurde gedrückt
    }
    else{
        // mach irgendwas der Taster wurde gedrückt
    }
}
```

Bearbeiten Sie nun Aufgabe 4 bis 8 Einfache Ein- / Ausgabe.

"Entprellen" eines Tasters / Flankenabfrage

Entprellen eines Tasters

Beim Ausführen des Programmes von Aufgabe 8 zur einfachen Ein - / Ausgabe kann es passieren, dass die LED mehrfach ihren Zustand wechselt, auch wenn man nur einmal den Taster drückt. Dies liegt daran, dass das Programm sehr schnell abläuft und in der Zeit, in der der Taster gedrückt ist mehrfach dessen Zustand abfragt. Um dies zu ändern gibt es zwei Möglichkeiten:

Man entprellt den Taster durch eine zusätzliche digitale Schaltung² (der Taster muss ein Wechseltaster sein, bei einfachen Tastern eignet sich eine analoge Schaltung).

Flankenabfrage

Durch geschickte Programmierung wird nicht nur der Zustand des Tasters abgefragt, sondern Änderungen der Flanke am digitalen Eingang gespeichert. Als Flanke wird der kurzzeitige Zustand des Tasters bezeichnet, in dem der Taster gedrückt, oder losgelassen wird. Wird der Taster gedrückt, so wechselt der Zustand von LOW nach HIGH, man spricht von einer positiven Flanke (rising edge). Wird der Taster losgelassen, so spricht man von einer negativen Flanke (falling edge), der Zustand wechselt von HIGH nach LOW.

Die meisten Programme erfordern es, dass nur ein einziges Mal beim Drucken des Tasters eine Aktion ausgeführt wird. Das könnte z.B. das Weiterschalten eines Menüpunkts sein oder das Umschalten eines Schaltzustands. Fragt das Programm lediglich ab, ob eine Taste gedrückt ist, dann

² Hiermit kann auch ein echtes Prellen (der Schalterzustand ändert sich, obwohl der Taster gedrückt ist) verhindert werden.

wird diese Aktion so oft ausgeführt, wie die Taste gedrückt ist. Mit einer Flankenerkennung erreicht man es nun, dass eine Aktion nur genau ein einziges Mal pro Tastendruck ausgeführt wird.

Statische Variablen

Für folgendes Programmbeispiel ist es notwendig, den Wert einer Variablen bei jedem Durchlauf der `loop()`-Funktion beizubehalten. Eine statische Variable tut genau das. Statische Variablen werden nur einziges Mal auf einen Wert initialisiert und behalten dann den Wert, auf den sie als letztes gesetzt wurden, zwischen verschiedenen Funktionsaufrufen bei.

Programmbeispiel statische Variablen

```
void f() {  
    static int s=0;  
    int i=0;  
    s++;  
    i++;  
}
```

Bei jedem Aufruf von `f()` erhöht sich `s` um 1 während `i` am Ende eines jeden Aufrufs den gleichen Wert hat.

Um nun eine Flanke, d.h. eine Änderung, in Software zu erkennen, muss darauf zurückgegriffen werden, was der Wert des Signals beim vorhergegangenen Durchlauf der Schleife war.

Eine Flankenerkennung kann also wie folgt aussehen.

```
void loop() {  
    int sw= ! digitalRead (swNr);  
    static int sw_alt=LOW;  
    if (sw_alt < sw) {  
        // steigende Flanke  
    } else if (sw_alt > sw) {  
        // fallende Flanke  
    } else {  
        // gar keine Flanke  
    }  
    sw_alt=sw;  
}
```

In der ersten Zeile dieses Codebeispiels wird der Wert des Schalters eingelesen. Da dieser aber Low-Active angeschlossen ist, wird der Wert invertiert. So enthält die Variable `sw` den tatsächlichen Zustand des Tasters. Die Variable `sw_alt` ist statisch und speichert den Wert von `sw` beim letzten Durchlauf der Schleife. Die Erkennung der Flanke selbst geschieht durch einen Vergleich dieser beiden Variablen.

Bearbeiten Sie nun Aufgabe 9 zur einfachen Ein- / Ausgabe.

Kontrollausgaben / Serielle Schnittstelle Teil 1

Der Arduino bietet keine optische Ausgabe, beispielsweise von Werten. Natürlich könnte man die Ausgabepins (analog / digital) für einige Ausgaben verwenden, aber sie sollen ja dem eigentlichen Programm dienen. Läuft nun ein Programm nicht wie gewünscht, so kann man kein Debugging betreiben oder anhand von Ausgaben den Wert von Variablen überprüfen. Hier kann man sich die serielle Schnittstelle des Arduino Boards zu Nutze machen:

Das Board besitzt mit den Pins 0 und 1 die beiden notwendigen Leitungen, um eine serielle Kommunikation zu realisieren. In der Grundbibliothek des Arduino gibt es ein fertiges Paket, mittels dem man über diese Schnittstelle kommunizieren kann:

Zuerst muss die Schnittstelle initialisiert werden:

```
Serial.begin(baudrate);
```

Dieser Befehl öffnet die Schnittstelle und setzt die Übertragungsrate auf den übergebenen Wert, beim Arduino 9600Bd (Baud: Bit pro Sekunde), also:

```
Serial.begin(9600);
```

Dieser Befehl muss nur einmal ausgeführt werden, also kann man ihn in die Funktion `setup()` einbinden.

Dann stehen die beiden Funktionen `Serial.print()` und `Serial.println()` zur Verfügung. Diese Funktionen schreiben den übergebenen Werte auf die Schnittstelle und können auf der "anderen Seite" abgefragt werden. Die Funktion `Serial.println()` fügt zusätzlich noch einen Zeilenumbruch hinzu. Die beiden Funktionen sind mehrfach überladen, so dass man ihnen sowohl Integerzahlen, Fließkommazahlen, Zeichen und Strings übergeben kann.

Serieller Monitor:

Die Kontrolle am Rechner funktioniert mit dem seriellen Monitor. Diesen findet man im Menüpunkt "Tools":



Bearbeiten Sie nun Aufgabe 1 bis 3 zu Kontrollausgaben.

Analoge Werte einlesen

Analoge Eingänge:

Das Arduino-Board besitzt 6 analoge Eingabepins (Pingruppe 2). Diese müssen in der `setup()`-Funktion nicht initialisiert werden. Stattdessen kann man mittels der Funktion: `int analogRead(pin: int)` direkt den Wert am Pin einlesen. Hinter jedem Pin steckt ein 10-Bit Analog-Digital-Wandler. Dies bedeutet, dass der Wandler eine eingelesene Spannung im Bereich von 0V bis xV in $2^{10} = 1024$ verschiedene Werte (also gibt es 1023 verschiedene Bereiche!) unterteilt und den eingelesenen Wert an den Prozessor zur weiteren Verarbeitung leitet. Der Wert xV, also die maximale einzulesende Spannung kann über die Funktion: `analogReference(type)` eingestellt werden. Hierbei gibt es mehrere Optionen:

Befehl	Erklärung
analogReference (DEFAULT) ;	Die maximale Spannung beträgt 5V. Diese Einstellung ist grundsätzlich gesetzt, so dass sie nicht unbedingt aufgerufen werden muss.
analogReference (INTERNAL1V1) ;	Die maximale Spannung ist 1,1V. Diese Option ist nicht bei allen Boards verfügbar.
analogReference (INTERNAL2V56) ;	Die maximale Spannung ist 2,56V. Diese Option ist nicht bei allen Boards verfügbar.
analogReference (EXTERNAL) ;	Die maximale Spannung ist die, die am Pin AREF extern angelegt wird. Sie muss zwischen 0V und 5V liegen.

Umrechnung des eingelesenen analogen Wert

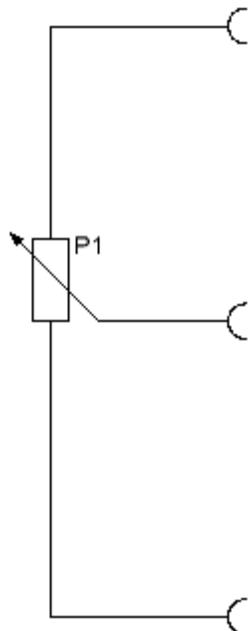
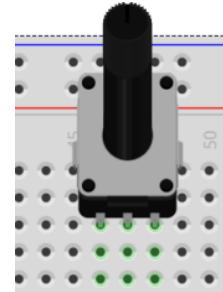
Angenommen, am Pin 0 wird der Wert 546 eingelesen, die maximale einzulesende Spannung ist 5V. Gesucht ist die Spannung, die am Pin 0 tatsächlich anliegt. Diese Spannung kann mit einem einfachen Dreisatz ausgerechnet werden:

$$\frac{5V}{1023} = \frac{x}{546} \mid \cdot 546 \rightarrow x = 5V \cdot \frac{546}{1023} = 2,67V$$

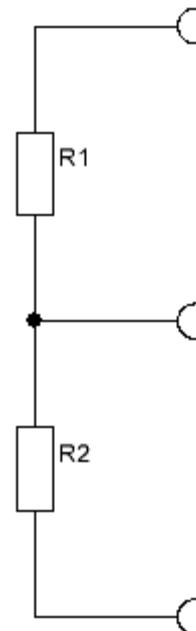
Potentiometer:

Das Potentiometer ist ein Bauteil mit drei Anschlüssen und stellt einen einstellbaren Widerstand dar.

Die interne Beschaltung ist dargestellt:



Potentiometer



Ersatzschaltbild

Erklärung Potentiometer:

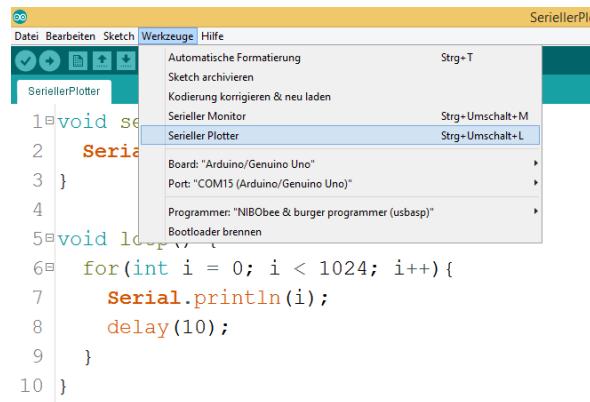
Das Potentiometer ist in zwei Widerstände R₁ und R₂ aufgeteilt, diese sind in Reihe geschaltet. Dreht man an dem Potentiometer, so verändert sich das Verhältnis der beiden Widerstände. Legt man nun an den beiden äußeren Anschlüssen des Potis eine Spannung U an, so kann man durch Drehen des Potentiometers den Spannungswert an dem mittleren Pin ändern. Die abzugreifende Spannung U_x berechnet sich wie folgt:

$$U_x = U \frac{R_2}{R_1 + R_2}$$

Bearbeiten Sie nun die Aufgaben 1 und 2 zum Einlesen analoger Werte.

Serieller Plotter:

Seit Version 1.6 der Entwicklungsumgebung gibt es im Menü Werkzeuge den Punkt Serieller Plotter:



Diese kleine Applikation erlaubt es, Daten, die vom Arduino an die serielle Schnittstelle des PCs geschickt werden., grafisch zu visualisieren.

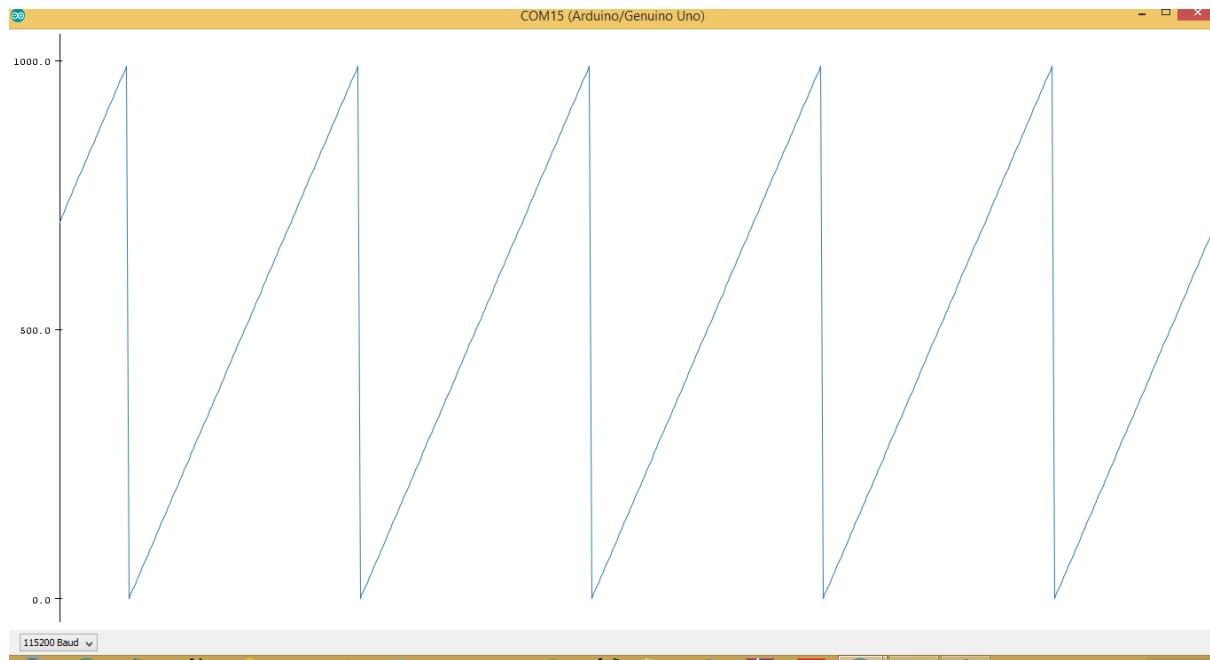
Beispiel:

Der folgende Quelltext schickt die Zahlen von 0 bis 990 in Zehnerschritten an die Schnittstelle:

```
void setup() {  
  
    Serial.begin(115200);  
  
}  
  
void loop() {  
  
    for(int i = 0; i < 1000; i+=10){  
  
        Serial.println(i);  
  
        delay(10);  
  
    }  
}
```

}

Das Ergebnis sieht wie folgt im seriellen Plotter aus:



Mit Hilfe des seriellen Plotters lassen sich also Anwendungen wie Oszilloskop oder Logic Analyzer realisieren.

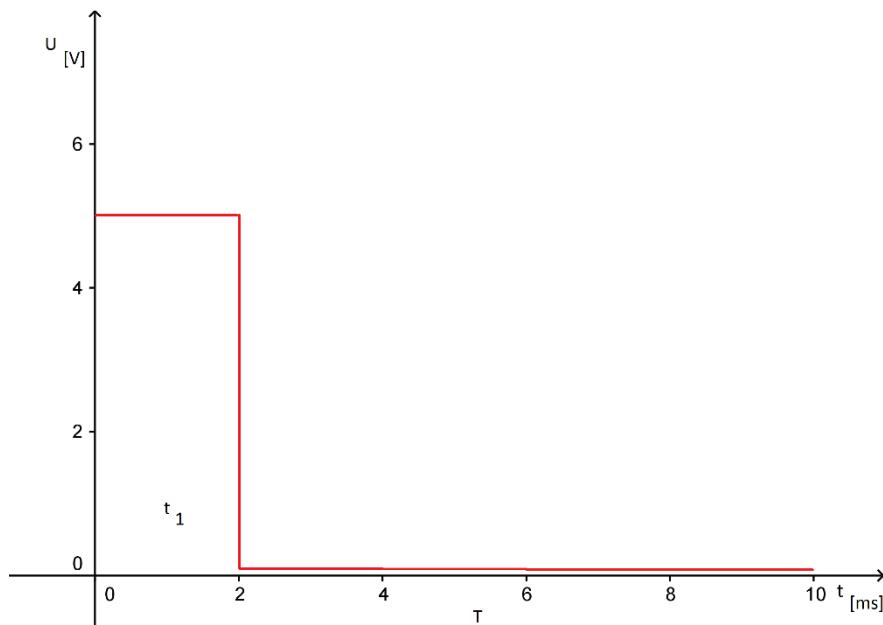
Analoge Werte ausgeben

Analoge Ausgabepins:

Einige der digitalen Pins haben das Tilde-Zeichen (~) vor der Pinnummer stehen. Mit diesen Pins kann man analoge Spannungen ausgeben. Dies wird allerdings nur durch die Puls-Weiten-Modulation simuliert:

Puls-Weiten-Modulation (PWM):

Bei der PWM wird ein digitaler Ausgang mit einer hohen Frequenz ein- und wieder ausgeschaltet. Das Verhältnis der Zeiten, an denen an dem Ausgang ein HIGH bzw. ein LOW-Signal anliegt, kann verändert werden. Misst man nun die Spannung an diesem Ausgang, so wird ein Mittelwert des HIGH-Anteils und des LOW-Anteils gebildet und ausgegeben. Dadurch wird auf dem Messgerät eine analoge Spannung angezeigt. Je länger der HIGH-Anteil ist, desto höher ist die Spannung und umgekehrt. Über diese technischen Feinheiten muss sich der reine Arduino-Anwender kaum Gedanken machen:



Das Verhältnis von $\frac{t_1}{T}$ wird als Tastgrad bezeichnet. Über den Tastgrad kann die ausgegebene analoge Spannung errechnet werden. Hat man beispielsweise eine Referenzspannung von 5V und einen Tastgrad von 0,2, so ergibt sich am analogen Ausgang eine Spannung von: $U_a = 0,2 \cdot 5V = 1V$

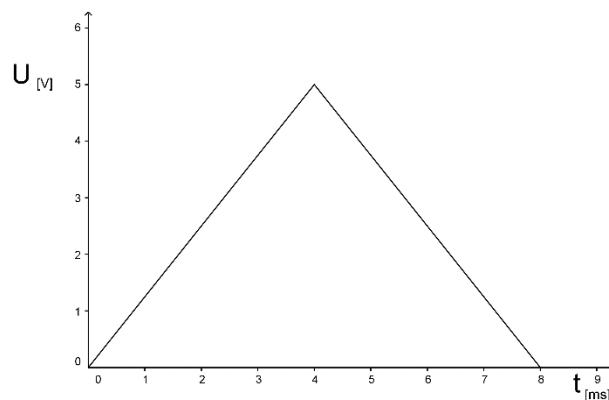
Ansteuern der analogen Ausgabepins:

Die Funktion `analogWrite(pin: int, wert: int)` gibt den Wert `wert` an den Pin `pin` aus. Der ausgegebene Wert liegt im Bereich 0 bis 255. Bei einer maximalen Ausgabespannung bedeutet dies 0V für den Wert 0 und 5V für den Wert 255. Über einen einfachen Dreisatz kann die Ausgabespannung berechnet werden.

Wichtig: Der verwendete Pin muss im Setup als OUTPUT definiert werden.

Dreiecksspannung:

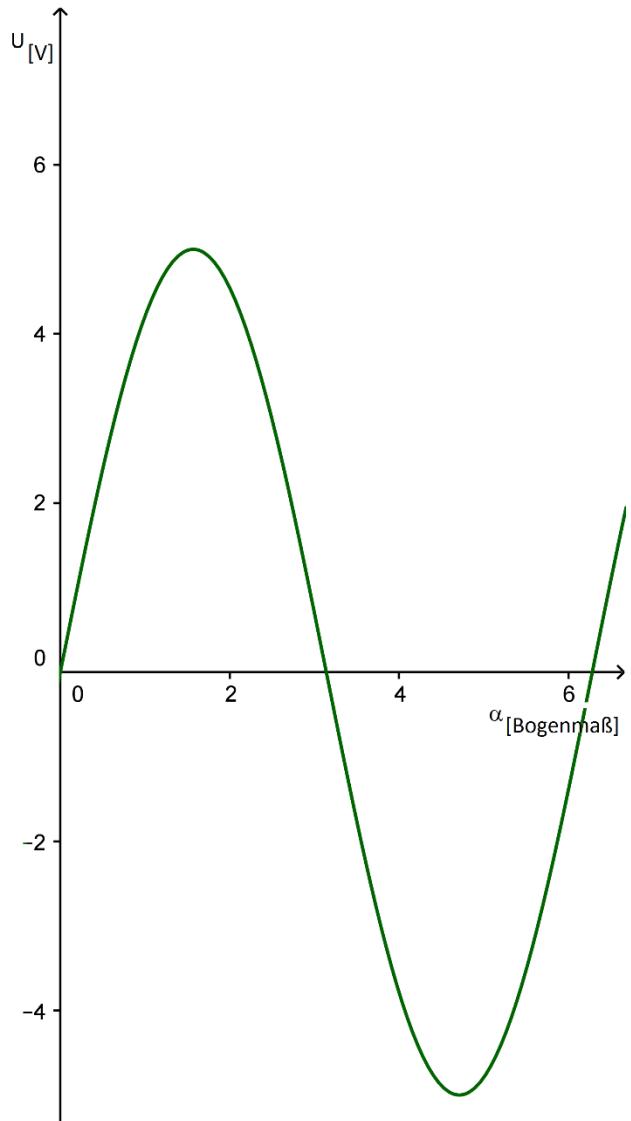
Eine Dreiecksspannung steigt linear an und fällt ebenso wieder ab.



Sinusspannung:

Eine sinusförmige Spannung sieht wie folgt aus:

Die Programmierfunktion `double sin(winkel:double)` gibt den Sinuswert des übergebenen Winkel zurück. Beachten Sie, dass der übergebene Wert im Bogenmaß verwendet wird.



Programmieren Teil 2

Schleifen

Kopfgesteuerte Schleife

Struktur der kopfgesteuerten Schleife:

```
while (Eintrittsbedingung) {  
    //Schleifenkörper  
}
```

Erklärung der kopfgesteuerten Schleife:

Die kopfgesteuerte Schleife wird erst betreten, wenn die Eintrittsbedingung erfüllt ist. Dann wird der Schleifenkörper ausgeführt und die Eintrittsbedingung erneut geprüft usw.

Fußgesteuerte Schleife

Struktur der fußgesteuerten Schleife:

```
do {  
    //Schleifenkörper  
} while (Austrittsbedingung);
```

Erklärung der fußgesteuerten Schleife:

Die fußgesteuerte Schleife wird immer betreten! Dann wird der Schleifenkörper ausgeführt und die Austrittsbedingung geprüft. Ist diese erfüllt, so wird die Schleife wiederholt. Man beachte, dass hinter der Austrittsbedingung ein Semikolon stehen muss!

Zählschleife

Struktur der Zählschleife:

```
for (Anfangsstatement; Eintrittsbedingung; Wiederholungsoperation) {  
    //Schleifenkörper  
}
```

Erklärung der Zählschleife:

Die Zählschleife ist dafür geeignet, um eine fest definierte Anzahl von Schleifendurchläufen abzuarbeiten. Gleichzeitig kann man eine Variable, die beispielsweise mit hochgezählt wird, für Ausgabe- oder Rechenoperationen verwenden.

Die Schleife beginnt mit einem Anfangsstatement. Meistens steht hier die Deklaration und Initialisierung einer Variablen, Bsp.: int i = 0;

Die Eintrittsbedingung funktioniert wie bei der kopfgesteuerten Schleife.

Die Wiederholungsoperation wird immer nach Abarbeitung des Schleifenkörpers einmal ausgeführt.

Beispiel:

```
for(int i = 0; i < 10; i++) {  
    //Schleifenkörper  
}
```

Erklärung:

Zu Beginn wird die Variable i mit dem Wert 0 belegt. Nach jedem Durchlauf des Schleifenkörpers wird die Variable i um den Wert 1 erhöht. Die Schleife läuft also 10x durch.

Funktionen

Funktionen verwendet man dann, wenn es Abläufe im Programm gibt, die öfter benötigt werden und nicht ständig neu im Quelltext vorkommen sollen. Eine Funktion hat immer einen Namen, eine Parameterliste von Daten, die man an die Funktion schickt und einen Datentyp als Rückgabewert.

Struktur von Funktionen

Beispiel:

Eine Funktion soll zwei Integerzahlen addieren und die Summe zurückgeben. Der Quelltext wäre der folgende:

```
int summiere (int wert1, int wert2) {  
    int summe;  
    summe = wert1 + wert2;  
    return summe;  
}
```

Zur Erklärung:

Die Funktion hat den Namen `summiere`. Ihr werden zwei Variablen vom Datentyp Integer übergeben. Die Variablennamen sind `wert1` und `wert2`. Die Funktion hat den Rückgabetyp `int`, also eine Integerzahl. In der Funktion werden die beiden übergebenen Werte addiert und der Variablen `summe` zugeordnet. Mittels `return` wird dann der Wert der Variablen `summe` zurückgegeben.

Rückgabewert	Funktionsname	Übergabeparameter
int, double, boolean, void ...	Repräsentativer Name	in runden Klammern: Parameter als Variablen mit Variabtentyp und Variablenname

Zugriff auf Funktionen

Funktionen haben Rückgabeparameter, die dann extern verwendet werden können.

Beispiel:

Die oben aufgeführte Funktion `summiere()` soll im Loop-Bereich verwendet werden. Hier gibt es zwei Möglichkeiten.

- 1) Der Rückgabewert wird gespeichert:

```
int summe = summiere (3, 4);  
Hier steht nun in der Variablen summe der Wert 7.
```

- 2) Der Rückgabewert wird direkt verwendet:

```
Serial.println(summiere (3, 4));  
An die serielle Schnittstelle wird der Wert 7 ausgegeben.
```

Überladen von Funktionen

Funktionsnamen können mehrfach verwendet werden. Möchte man die oben aufgeführte Funktion `summiere()` beispielsweise auch für Werte vom Datentyp `double` verwenden, so kann man die Funktion erneut kodieren, sie muss sich aber in den Übergabeparametern von der ursprünglichen Funktion unterscheiden. Beispiel:

```
double summiere (double wert1, double wert2) { . . .
```

Gültigkeitsbereich von Variablen

Variablen sind nur in speziellen Bereichen gültig. Wird beispielsweise im loop-Bereich die Integervariable int i deklariert und verwendet, so kennt der setup-Bereich diese Variable nicht! Auch sind innerhalb von Funktionen Variablen nur im Bereich von geschweiften Klammern gültig. Beispiel:

Der folgende Quelltext steht in der loop-Funktion:

```
do {  
    int i = 5;  
} while (i < 5);
```

Beim Kompilieren der Datei gibt es eine Fehlermeldung. Dies liegt daran, dass die Variable i innerhalb der geschweiften Klammern der Schleife gültig ist. Da der while-Bereich aber außerhalb liegt, ist i dort nicht bekannt. Um sich das Leben leicht zu machen (führt aber gegebenenfalls zu unübersichtlichen Quelltext), können Variablen zu Beginn der Datei außerhalb jeder Funktion deklariert werden. Sie sind dann in jeder Funktion überall bekannt. Man spricht hier von **globalen** Variablen. Die anderen Variablen nennt man **lokale** Variablen.

Bibliotheken

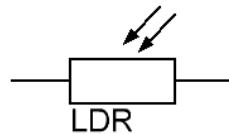
Viele Bibliotheken sind schon vorhanden. Schaut man sich in den Entwicklungsumgebungen bei den Beispielen um, so wird man für eine große Anzahl an technischen Anwendungen Programme finden. Um die fertigen Bibliotheken zu importieren muss am Anfang des Quelltextes über #include<> die entsprechende Bibliothek eingebunden werden. Die dort enthaltenen Funktionen können dann im eigentlichen Quelltext verwendet werden. Bsp:

```
#include <Servo.h>
```

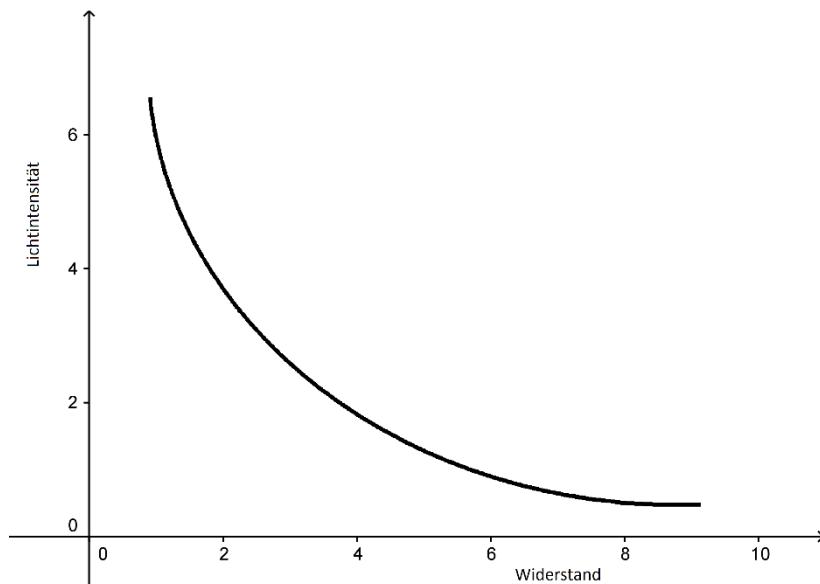
Sensoren

Fotowiderstand

Der lichtabhängige Widerstand (LDR, Light Dependent Resistor) verändert seinen Widerstandswert in Abhängigkeit des Lichteinfalls auf die schleifenähnliche Struktur oberhalb des Bauteils:



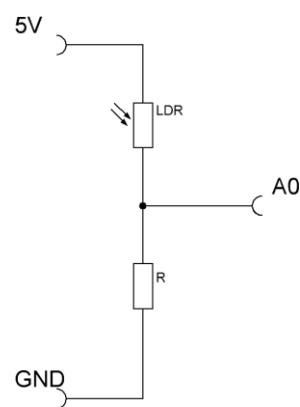
Das Bauteil hat einen negativen Lichtkoeffizienten. Dies bedeutet, dass der Widerstandswert des Bauteils sinkt, wenn mehr Licht einfällt und umgekehrt:



Auswerten der Helligkeit: Schaltung, Programm

Um den Widerstandswert des LDR auszuwerten, muss eine Schaltung erstellt werden, die die Wertänderung in eine messbare Spannung umsetzt. Hier scheint eine Reihenschaltung die richtige Wahl (zur Erinnerung: bei der Parallelschaltung ist die Spannung an allen Bauelementen gleich, bei der Reihenschaltung teilt sich die Spannung im Verhältnis der Bauteils auf!).

Schließt man den LDR wie abgebildet an, kann man an dem Pin A0 eine analoge Spannung messen, die einen Aufschluss auf den Helligkeitswert gibt.



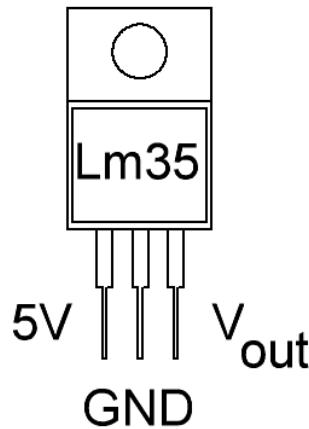
Bearbeiten Sie nun die Aufgabe 1 und 2 zum LDR.

Temperaturfühler

Im Starterkit ist ein fertiger Temperatursensor zu finden, der LM 35:



Bauteil LM 35



Anschlussbelegung

Der Sensor hat den Vorteil, dass er sich linear bei Temperaturänderungen verhält. Er arbeitet mit Betriebsspannungen von 4V bis 20V in einem Temperaturbereich von -55° bis 125°. Die Änderungen des Ausgangswertes beträgt: $\frac{10mV}{1^\circ}$. Eine Außenbeschaltung ist nicht nötig. Beachten Sie, dass der Sensor mit anderen Bauteilen (Transistoren) aus dem Starterkit leicht zu verwechseln ist. Der LM35 wurde mit einer weißen Markierung am Gehäuse zur Unterscheidung versehen.

Bearbeiten Sie nun die Aufgaben 1 und 2 zum Temperatursensor.

Tiltsensor



Der Tiltsensor ist nichts anderes, als ein Schalter. Als Kontakt dient eine Metallkugel im inneren des Gehäuses. Je nach Lage des Sensors schließt oder öffnet der Schalter.

Bearbeiten Sie nun die Aufgaben 1 bis 4 zum Tiltsensor.

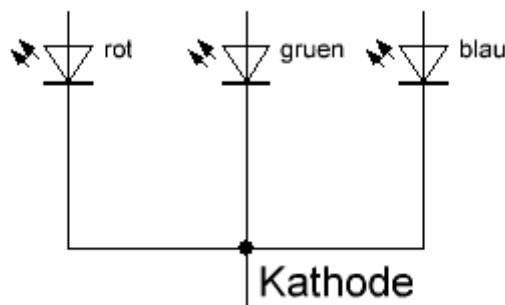
Aktoren

Mehrfarbige LED

Im Starterkit steht eine RGB-LED zur Verfügung. Diese sieht wie folgt aus:



Das längste Beinstehen stellt die gemeinsame Kathode dar, wird also an GND angeschlossen. Intern besteht dieses Bauteil aus drei verschiedenen LEDs mit unterschiedlichen Farben:



Da alle drei LEDs unabhängig voneinander angesteuert werden können, sind alle möglichen Mischfarben realisierbar. Beim Beschalten ist zu beachten, dass alle drei LEDs einen eigenen Vorwiderstand bekommen und jeweils auch ein eigenes Signal von den digitalen Ausgängen.

Bearbeiten Sie jetzt die Aufgaben zur RGB-LED.

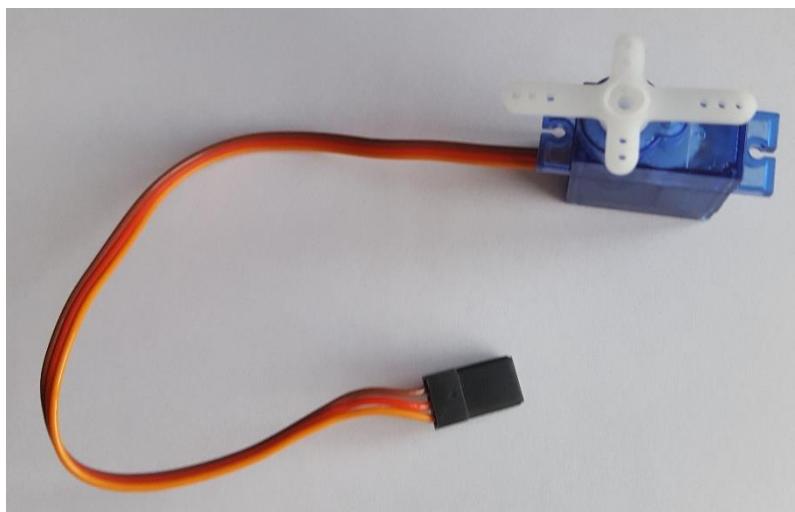
Summer

Der Piezosummer im Starterkit sorgt für Nerv törende Programme ☺. Er hat ein rotes und ein schwarzes Anschlusskabel. Das Schwarze liegt an GND, das Rote wird an einen Ausgang mit analoger Funktionalität angeschlossen. Die fertig gestellte Funktion `void tone(pin: int, frequenz: int)` liefert an diesem Ausgang ein Rechtecksignal in der entsprechenden Frequenz. Die Funktion `void noTone(pin: int)` beendet die Ausgabe. **Hinweis:** Der Summer sollte immer einen Ton mit einer Mindestfrequenz von 31Hz bekommen!

Bearbeiten Sie die Aufgabe 1 und 2 zum Summer.

Servo

Der Servo ist ein positionierbarer Motor. Die Steuerung erfolgt über die Pulsweitenmodulation.



Damit man die Pulsweitenmodulation nicht auch noch selbst programmieren muss, gibt es die Bibliothek `Servo.h`. Die Verwendung der Bibliothek ist wie folgt erklärt:

```
Servo myservo;          //Ein Objekt aus der Klasse Servo wird angelegt
myservo.attach(2);      //Der Datenanschluss an den Server erfolgt über
                       //Pin 2
myservo.write(winkel: int); //Über winkel (Werte von 0 bis 180) wird
                           //der Servo positioniert
```

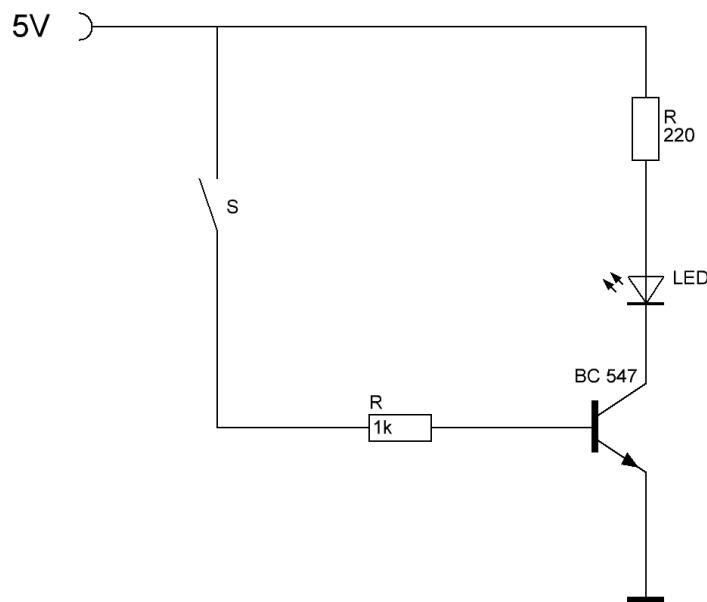
Bearbeiten Sie nun Aufgaben 1 und 2 zum Servo.

Transistor³

Der Transistor verstärkt Signale. Dies ist dann notwendig, wenn die Leistung der Ausgänge des Arduino nicht ausreicht, um beispielsweise Relais oder andere stärkere Verbraucher zu versorgen.



Der Transistor hat drei Anschlüsse, die mit Emitter, Kollektor und Basis bezeichnet werden. Von dem Transistor gibt es zwei verschiedenen Sorten, ein NPN-Typ und ein PNP-Typ. Angesteuert wird der Transistor über den Basisanschluss. Der Strom ist hier sehr klein und muss über einen Vorwiderstand ($1\text{k}\Omega$) abgesichert werden. Der Verbraucher wird dann zwischen Versorgungsspannung und Kollektor geschaltet. Nachfolgend wird eine Leuchtdiode über einen Transistor ein- und ausgeschaltet:



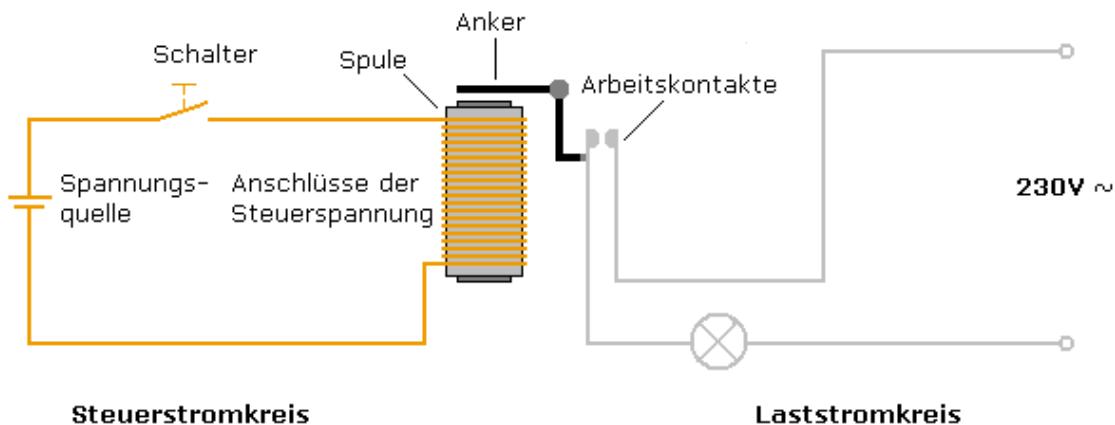
³ Die Erklärung des Transistors ist stark vereinfacht und nur dafür ausgelegt, den im Starterkit verfügbaren Transistor zu verwenden!

Bearbeiten Sie nun Aufgabe 1 und 2 zum Transistor.

Relais

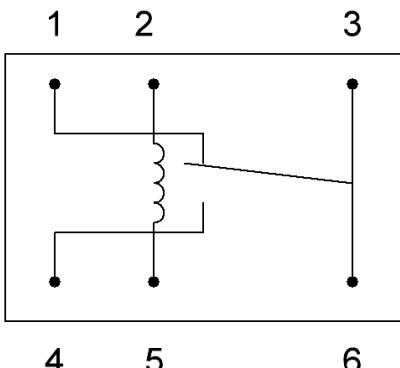
Das Relais dient dazu, größere Ströme zu schalten. Will man beispielsweise eine Glühlampe ein- und ausschalten, so kann dies nicht direkt über den Arduino erfolgen. Das Relais hilft:

Beim Relais wird ein Kontakt durch eine kleine Spannung bzw. durch einen kleinen Strom geschlossen. Dies ist beim Schließen durch ein "Klicken" auch zu hören. Dieses Schließen wird durch den Strom durch eine Spule und das dadurch entstehende Magnetfeld erreicht. Das Magnetfeld zieht den Kontakt an, wodurch dieser geschlossen wird (Prinzip Lautsprecher). Die nachfolgende Grafik verdeutlicht diese Vorgehensweise sowie die Schaltung.



Quelle: http://de.wikipedia.org/wiki/Relais#mediaviewer/File:Relais_Animation.gif

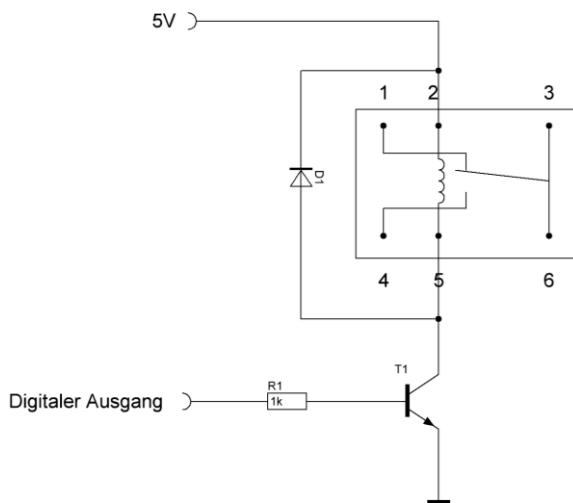
Beim Arduino Fritzing-Kit ist das Relais: FRS1H-S mitgeliefert. Aus dem Datenblatt entnimmt man die folgende Anschlussbelegung (Blick von unten):



Beschreibung:

Über Pin 2 und 5 wird das Relais geschaltet (Steuerkreis). Das Relais ist als Wechselschalter ausgelegt. Ohne angelegte Spannung sind Pin 1,3 und 6 kurzgeschlossen. Wird am Steuerkreis Spannung angelegt, so sind Pin 4, 3 und 6 kurzgeschlossen. Das Relais wird mit 5V gesteuert und kann Spannungen bis 125V und Ströme bis 1A schalten.

Zum Steuern des Relais wird ein Strom benötigt, der größer ist, als der Arduino mit einem einfachen digitalen Ausgang bereitstellen kann. Somit benötigt man eine Steuerschaltung mit Transistor:



Zur Erklärung:

Die Steuerspule wird einmal an die Betriebsspannung von 5V und an den Kollektor des Transistors angeschlossen. Parallel zur Spule wird eine Diode geschaltet (Freilaufdiode). Diese Diode sorgt dafür, dass beim Ausschalten des Transistors eine auftretende hohe Gegenspannung (Stichwort: Induktion) kurzgeschlossen wird und der Transistor nicht zerstört wird. Die Pins 1, 3, 4 und 6 können nun für den Laststromkreis verwendet werden.

Bearbeiten Sie nun die Aufgaben zum Relais.

Digitaltechnik

Digitaltechnische Grundschaltungen

UND / AND

Beim UND gilt die folgende Wertetabelle:

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Hierbei stellen A und B Eingänge dar, C ist der abhängige Ausgang. Der Tabelle entnimmt man, dass der Ausgang C nur dann gesetzt ist, wenn A und B gesetzt sind. Der Operator in der Programmierung ist: & &

ODER / OR

Die Wahrheitstabelle der ODER-Schaltung sieht wie folgt aus:

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Der Ausgang C ist also gesetzt, wenn mindestens einer der beiden Eingänge gesetzt ist. Der Operator in der Programmierung ist ||

NICHT / NOT

Folgende Wahrheitstabelle gilt beim NICHT:

A	C
0	1
1	0

Das NICHT negiert also nur den Eingang A.

Der Operator in der Programmierung ist !

Exklusiv-ODER

Beim Exklusiv-Oder wird der Ausgang gesetzt, wenn genau einer der beiden Eingänge gesetzt ist:

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Ein spezielles Schaltzeichen für die Programmierung gibt es nur für das bitweise Exklusiv-ODER.

Negierte Funktionen

Die beiden nachfolgenden digitalen Funktionen können in der Programmierung nicht direkt durch einen Operator dargestellt werden. Sie haben jedoch große Bedeutung in der Digitaltechnik, da durch sie sämtliche andere Funktionen realisiert werden können. Es ist also möglich, eine komplexe digitale Schaltung beispielsweise komplett mit NAND-Bausteinen zu realisieren, was die Kosten erheblich reduzieren kann.

NAND

Die Wahrheitstabelle entspricht einem negierten UND:

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NOR

Die Wahrheitstabelle entspricht einem negierten ODER:

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Komplexere digitale Schaltungen:

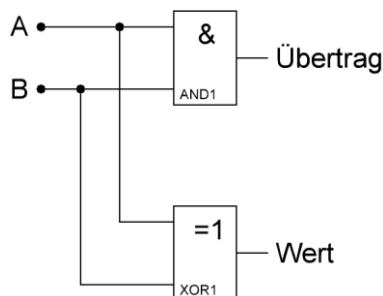
Halbaddierer

Der Halbaddierer addiert zwei einstellige binäre Zahlen. Er besitzt zwei Ausgänge: den einstelligen Wert der Addition, sowie den Wert des Übertrags.

Die Wertetabelle sieht wie folgt aus:

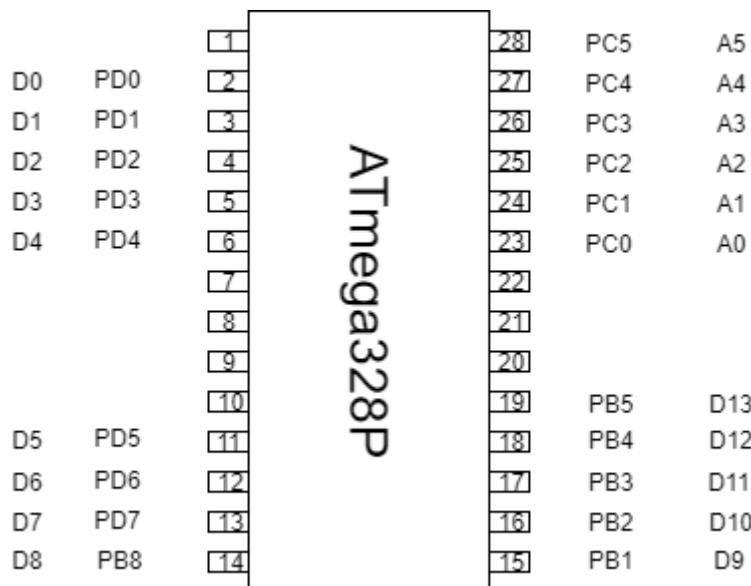
A	B	Wert des Übertrags	Einstelliger Wert der Addition
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Man sieht, der Wert des Übertrages kann durch eine UND-Schaltung, der einstellige Wert der Addition durch eine XOR-Schaltung realisiert werden. Die Schaltung des Halbaddierers sieht also wie folgt aus:



Register direkt ansprechen

Nachfolgend sehen Sie die Registerbelegung passend zu den Datenpins des Arduinos:



Selbstverständlich können auch beim Arduino die entsprechenden Register direkt angesprochen werden. Hierbei haben folgende Register eine Bedeutung (beispielhaft für das Register D!):

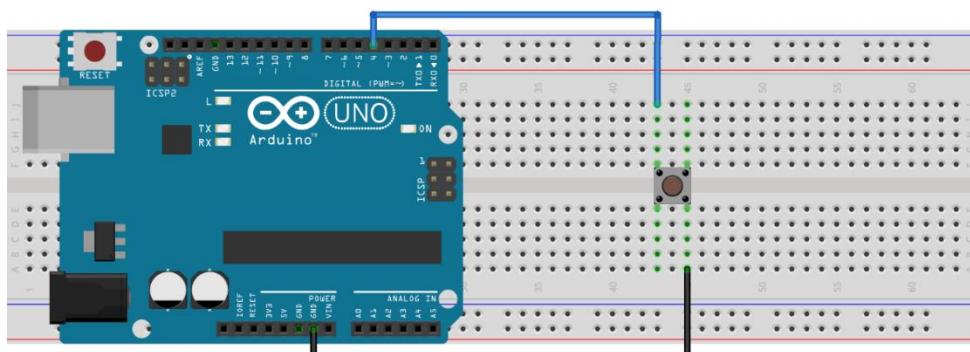
DDRD: Dieses Register setzt den Modus des Registers. Eine 1 an der entsprechenden Stelle bedeutet, dass das jeweilige Bit im Ausgabemodus ist. Eine 0 bedeutet, dass das jeweilige Bit des Registers im Eingabemodus ist. Im nachfolgendem Beispiel wird im Programm nur das Bit 5 des Registers B als Ausgangsbit definiert (Funktion `setup()`!).

PORTR: Dieses Register setzt den tatsächlichen Wert der Bits im Register B. Im nachfolgenden Programm wird das Bit 5 des Registers B abwechselnd von `HIGH` auf `LOW` und wieder zurückgesetzt. Das Register kann aber noch mehr: Wird ein Bit als Eingang definiert, und schreibt man dann im PORT-Register dieses Bit auf 1, so wird der interne Pullup-Widerstand aktiviert!

```
void setup() {
    DDRB = 0b00100000;
}

void loop() {
    PORTB = 0b00100000;
    delay(500);
    PORTB = 0b00000000;
    delay(500);
}
```

Im nachfolgenden Programm wird ein Taster an D4 ohne Pullup-Widerstand angeschlossen. Wird der Taster gedrückt, so wird die LED an Pin 13 eingeschaltet:



```
void setup() {  
    DDRB = 0b00100000;  
    DDRD = 0;  
    Serial.begin(9600);  
}  
  
void loop() {  
    PORTD = 0b00010000;  
    int wert = PIND;  
    wert = wert&16;  
    if(wert==16){  
        Serial.println("nicht gedrueckt");  
        PORTB = 0b00000000;  
    }else{  
        Serial.println("gedrueckt");  
        PORTB = 0b00100000;  
    }  
    delay(100);  
}
```

Interrupts

Es gibt zwei Möglichkeiten auf Ereignisse zu reagieren:

1. Polling

Polling bedeutet, dass man beispielsweise einen Pin ständig in einer Schleife abfragt und je nach Zustand im Programm reagiert. Diese Möglichkeit wurde bisher verwendet.

2. Interrupts

Der Arduino ist in der Lage, auf Ereignisse dann zu reagieren, wenn sie auftreten. Dies geschieht dadurch, dass man einem Eingangspin mitteilt, auf ein Signal zu warten und dann eine vorher definierte Funktion auszuführen. Der bis dahin laufende Programmcode wird unterbrochen (interrupt), die definierte Funktion wird ausgeführt und am Ende findet automatisch ein Rücksprung zum laufenden Programmpunkt statt. Die nachfolgende Grafik soll dies verdeutlichen:

Vier Funktionen sind für die Programmierung von Bedeutung:

Funktion	Bedeutung
interrupts()	Aktiviert die Interruptfunktionalität des Arduino. Diese Funktion ist standardmäßig aktiviert und muss nicht unbedingt aufgerufen werden. Erst nach Aufruf der nächsten Funktion noInterrupts() muss diese Funktion aufgerufen werden, damit der Arduino auf Interrupts reagieren kann.
noInterrupts()	Diese Funktion deaktiviert die Interrupts des Arduino. Die Funktionalität kann durch den Aufruf von interrupts() wieder hergestellt werden.
attachInterrupt()	Bindet einen Interrupt an einen Pin. Die Übergabeparameter werden unten dargestellt.
detachInterrupt()	Befreit den Pin wieder von seiner Interruptfunktionalität. Die Übergabeparameter werden unten dargestellt.

attachInterrupt(interruptnummer, funktion, modus)

Diese Funktion bindet einen Interrupt an einen Pin. Je nach Board sind diese Belegungen und Namen unterschiedlich. Hier werden nur die Möglichkeiten für das Arduino Uno Board dargestellt, andere Belegungen entnimmt man der API. Der Übergabeparameter `funktion` ist der Funktionname der Funktion, welche bei einem Interrupt ausgeführt werden soll. Man spricht von der Interruptserviceroutine (ISR).

Interruptbelegungen des Arduino Uno

Beim Arduino sind folgende Interrupts möglich:

Interruptnummer	Pin
0	2
1	3

Modus der Erkennung des Interrupts:

Folgende Modi sind möglich:

Modus	Beschreibung
LOW	Der Interrupt wird ausgelöst, wenn am jeweiligen Pin ein LOW-Signal anliegt.
CHANGE	Der Interrupt wird ausgelöst, wenn sich der Zustand am jeweiligen Pin ändert.
RISING	Der Interrupt wird ausgelöst, wenn am jeweiligen Pin eine aufsteigende Flanke erkannt wird.
FALLING	Der Interrupt wird ausgelöst, wenn am Interrupt eine absteigende Flanke erkannt wird.

Programmbeispiel:

Im nachfolgenden Programm wird in der Hauptfunktion eine Variable hochgezählt und deren Wert auf die serielle Schnittstelle ausgegeben. An Pin 2 ist über einen Pullupwiderstand ein Taster angeschlossen. Wird dieser Taster gedrückt oder losgelassen, so wird ein Interrupt ausgelöst, der die Funktion: `interruptfunction` aufruft.

```
int i = 0;
void setup()
{
    Serial.begin(9600);
    attachInterrupt(0, interruptfunction, CHANGE);
}

void loop()
{
    Serial.println(i);
    i++;
    delay(200);
}

void interruptfunction()
{
    Serial.println("Interrupt");
}
```

detachInterrupt(interruptnummer)

Die Funktion deaktiviert den Interrupt mit der übergebenen Interruptnummer (beim Arduino Uno 0 oder 1).

Bearbeiten Sie nun die Aufgaben zum Interrupt.

Fritzing

Fritzing ist eine Freeware, mittels der man virtuell Schaltungen zusammenbauen kann. Somit ist eine Visualisierung der Schaltung möglich. Die Software erstellt aus der Schaltung auch einen Schaltplan sowie ein Platinenlayout. Somit kann man seine, für den Arduino entwickelte Schaltungen, professionell fertigen lassen. Eine Simulation der Schaltung ist in Fritzing leider nicht möglich. Die Software kann unter: <http://fritzing.org/home/> herunter geladen werden.

Arduino Simulator

Einen Simulator für den Arduino gibt es unter: <http://www.virtronics.com.au/Simulator-for-Arduino.html>. Dieser ist nicht Freeware, allerdings recht kostengünstig und einfache Programme können simuliert werden. Einen weiteren (bisher nicht getesteten) Simulator gibt es für das Betriebssystem Linux unter: <http://web.simuino.com/downloads>

Informationen / Beispiele

Hinzufügen einer Bibliothek

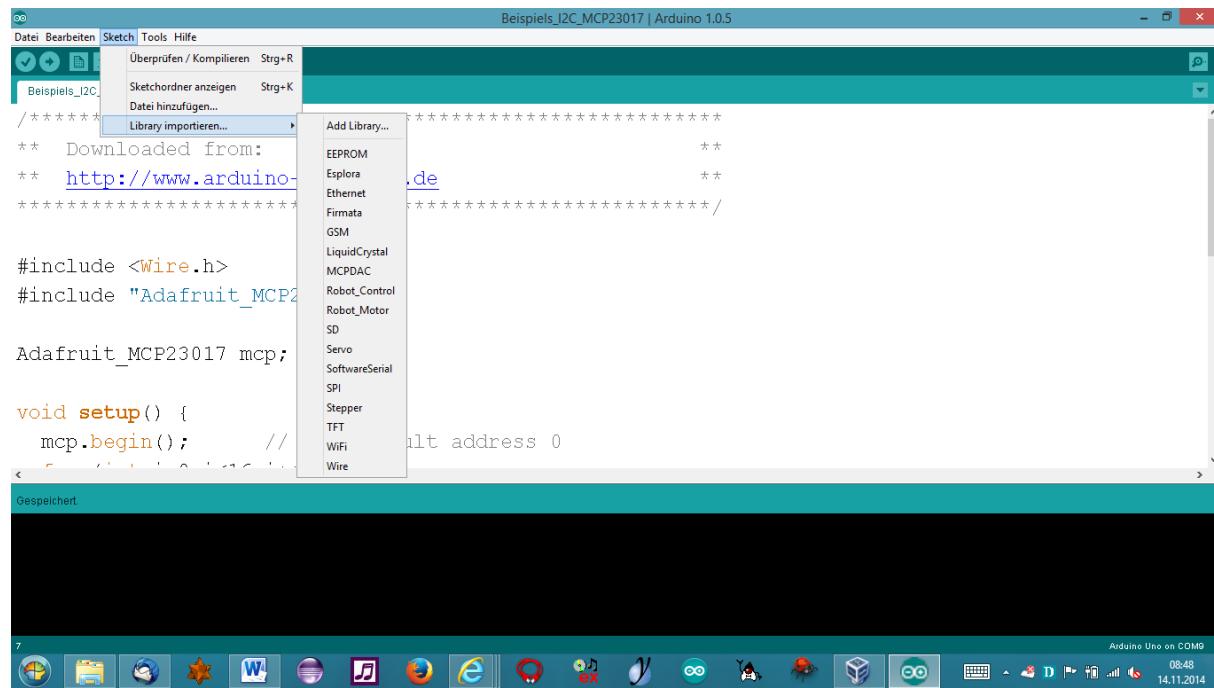
Bibliotheken nehmen einem viel Programmierarbeit ab. Eine Internetrecherche zeigt auf, wie viele Bibliotheken es für die unterschiedlichsten Bereiche gibt. Die einfachste Art, eine Bibliothek zu nutzen wird wie folgt beschrieben:

Laden Sie die Bibliothek herunter und entpacken Sie die Datei (Bibliotheken werden oft im .zip-Format angeboten).

Kopieren Sie das neu entstandene Verzeichnis in den Ordner: Dokumente / Arduino / libraries.

Fertig!

Im Programm können Sie nun unter dem Menüpunkt Sketch / Library importieren die Bibliothek zu Ihrem Programm hinzufügen (siehe Screenshot).

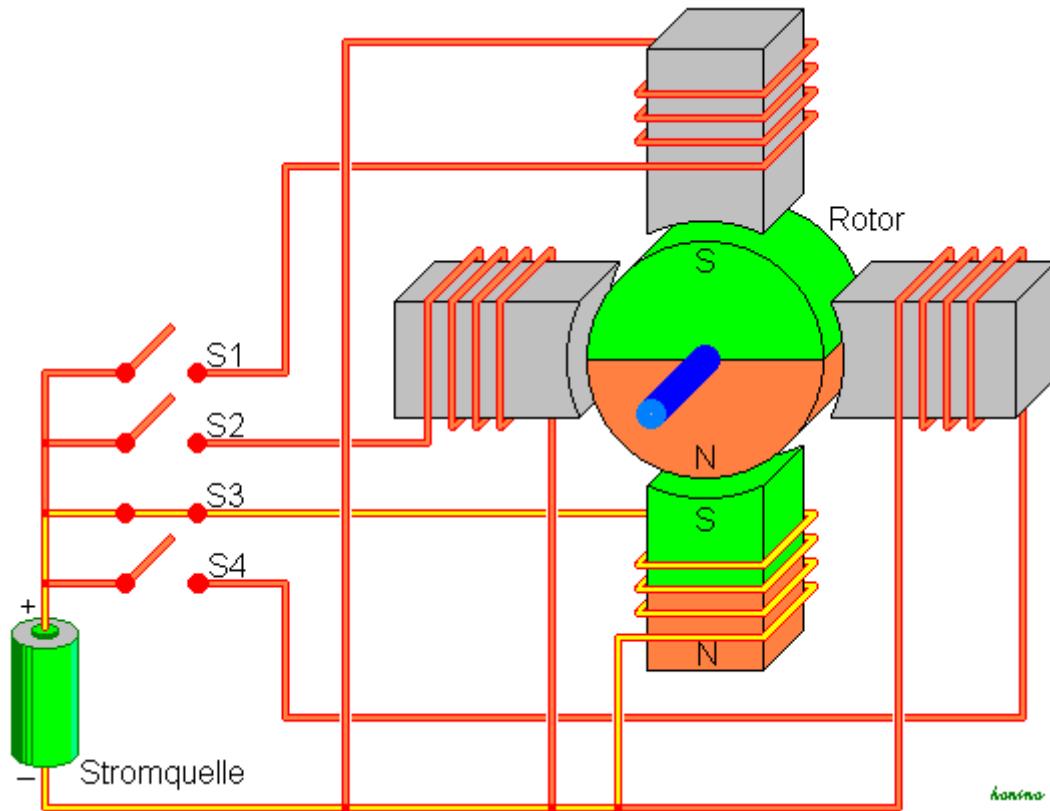


Der Vorteil dieser Vorgehensweise ist, dass Sie über Datei / Beispiele sofort auf die mit der Bibliothek mitgelieferten Beispiele zugreifen können.

Ansteuern eines Schrittmotors

Schrittmotoren eignen sich weniger als Antriebsmotoren. Sie werden als Stellmotoren eingesetzt, da man über die hohe Genauigkeit der Positionierung exakte Winkel anfahren kann. Somit eignet sich der Schrittmotor als mechanischer Antrieb für Roboter, PC-Laufwerke, CNC-Maschinen, Uhren etc.

Das Prinzip eines Schrittmotors



4

Der Rotor stellt einen Permanentmagneten dar. Die Statoren werden jeweils von einer Spule umwickelt. Legt man nun an die Spule eine Spannung an, so bildet sich in dem jeweiligen Stator ein Magnetfeld aus. Der Rotor richtet sich dann nach dem äußeren Magnetfeld wie eine Kompassnadel aus.

Durch eine geschickte Kombination von Ein- und Ausschaltvorgängen der Statorspulen lässt sich dann eine schrittweise Kreisbewegung des Rotors erzeugen, der Motor dreht sich. Durch eine zeitliche Veränderung der Ein- und Ausschaltvorgänge regelt man die Geschwindigkeit des Motors.

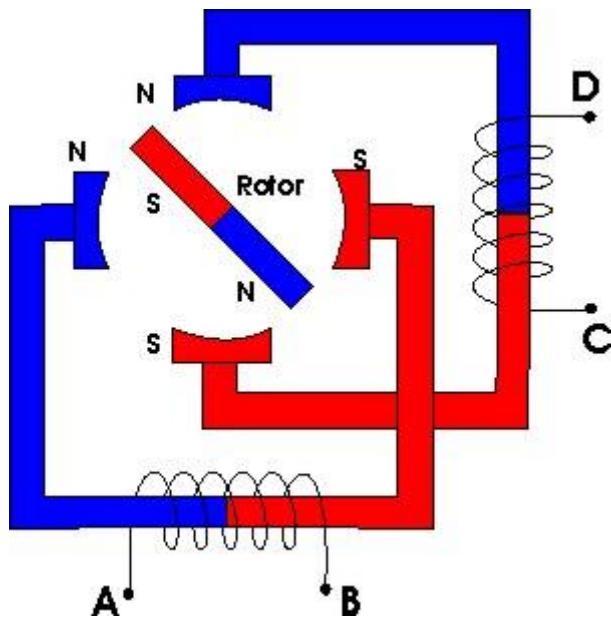
Arten von Schrittmotoren

Man unterscheidet unipolare und bipolare Schrittmotoren.

Bipolarer Schrittmotor

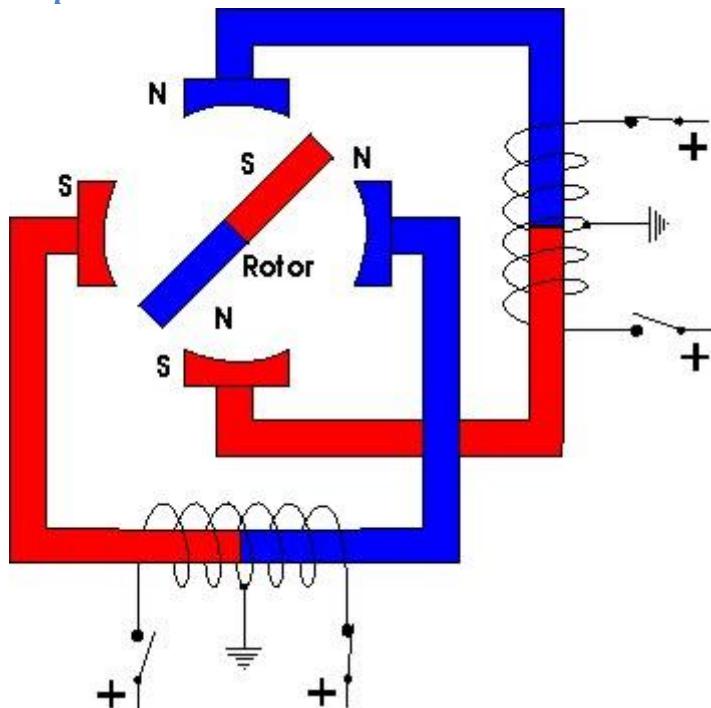
Beim bipolaren Schrittmotor gibt es nur zwei Statorspulen. Der Vorteil hier ist ein hohes Drehmoment. Nachteilig wirkt sich aus, dass die Polarität der einzelnen Spulen umgeschaltet werden muss. Dies bedeutet einen höheren Schaltungsaufwand zur Ansteuerung.

⁴ Grafik aus: <http://de.wikipedia.org/wiki/Schrittmotor>



5

Unipolarer Schrittmotor



6

Beim Unipolaren Schrittmotor haben die Statorspulen eine Mittelanzapfung. Damit ist eine leichtere Ansteuerungsschaltung möglich, da die einzelnen Spulen nicht umgepolt werden müssen.

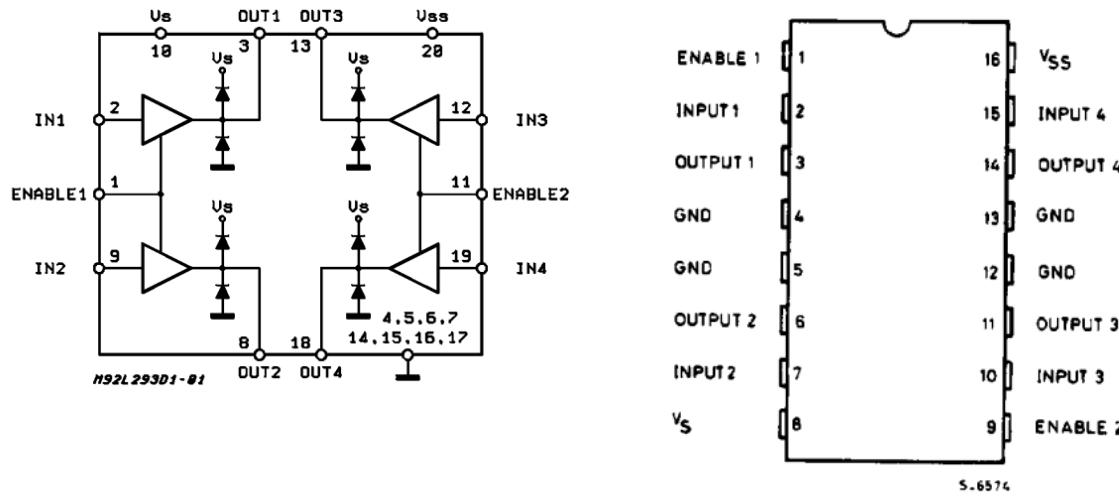
⁵ Grafik aus: http://www.mikrocontroller.net/mc-project/Pages/Robotik/Mechanik/mechanik_stepp_bipolar.jpg

⁶ Grafik aus: http://www.mikrocontroller.net/mc-project/Pages/Robotik/Mechanik/mechanik_stripper.jpg

Eine sehr schöne Visualisierung von diversen Schrittmotortypen und deren Ansteuerung findet man unter: <http://de.nanotec.com/support/tutorials/schrittmotor-und-blcd-motoren-animation/>

L293 D

Der Baustein realisiert einen vierfachen Motortreiber mit der Möglichkeit, über ein Enable-Signal die Geschwindigkeit beispielsweise auch eines Gleichstrommotors zu regeln. Aus dem Datenblatt entnimmt man die folgende Anschlussbelegung:



7

Der IC ist zweigeteilt. Jeweils die Pins 1 – 8 und 9 bis 16 bilden eine Einheit. An Pin 8 bzw. Pin 16 wird die Betriebsspannung des Motors bis zu 36V angeschlossen. Diese Spannung sorgt auch für den Betrieb des ICs. Die Pins 4,5 sowie 12 und 13 liegen auf Masse.

Ein HIGH-Signal an Pin 1 schaltet Block A aktiv, genauso wie ein HIGH-Signal an Pin 9 für Block B.

Die beiden Ausgänge von Block A werden über Pin 2 und 7 geschaltet, die von Block B über Pin 10 und 15. Hier können beispielsweise digitale Ausgänge des Arduino angeschlossen werden.

Aus dem Blockdiagramm entnimmt man, dass der LM293D Freilaufdioden schon integriert, einem direkten Motorbetrieb steht also nichts im Wege.

Der Vorteil dieses Bausteines ist, dass die geschaltete Spannung im Gegensatz zum ULN 2803 nicht negiert wird. Somit ist also auch ein Umschalten der Polarität der Wicklung eines Motors kein Problem und er eignet sich daher zum Ansteuern von Gleichstrommotoren, unipolaren Schrittmotoren und bipolaren Schrittmotoren.

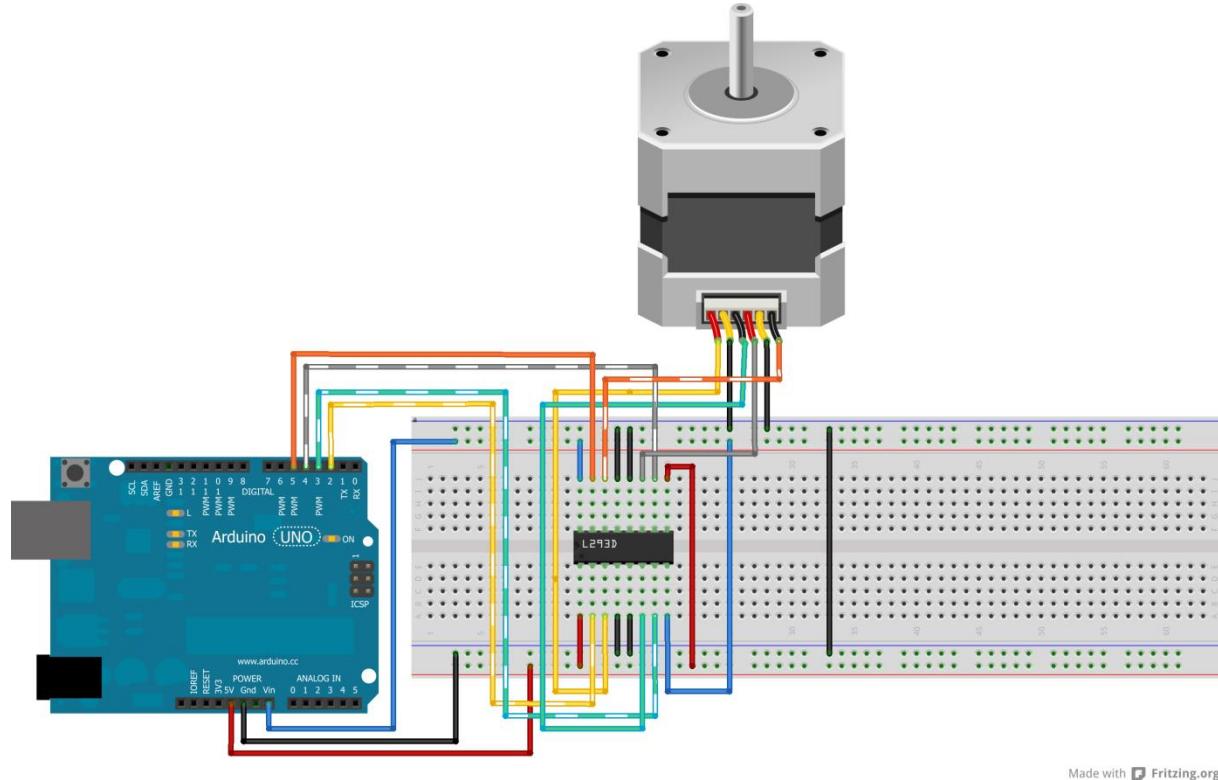
Achtung:

Der Baustein L293 (ohne D!) kann zwar einen höheren Strom schalten (bis 1,2A), implementiert aber keine Freilaufdioden!

⁷ Grafik aus: www.alldatasheet.com

Ansteuerung eines unipolaren Schrittmotors mit dem Arduino

Der nachfolgende Aufbau realisiert eine Schrittmotorsteuerung mit dem L293D und einem unipolaren Schrittmotor:



Zu beachten ist, dass der Schrittmotor mit einer externen Spannungsquelle versorgt ist, die an der Spannungsbuchse des Arduino angeschlossen ist (im Aufbau nicht zu sehen). Die externe Spannung kann am Pin V_{in} abgenommen werden.

Den folgenden Tabellen können die Ansteuerabfolge für Rechts- und für Linkslauf entnommen werden:

Rechtslauf:

Schritt	A	\bar{A}	B	\bar{B}
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Linkslauf:

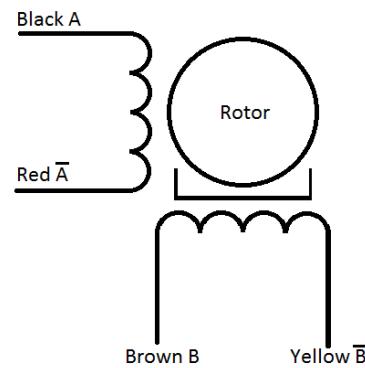
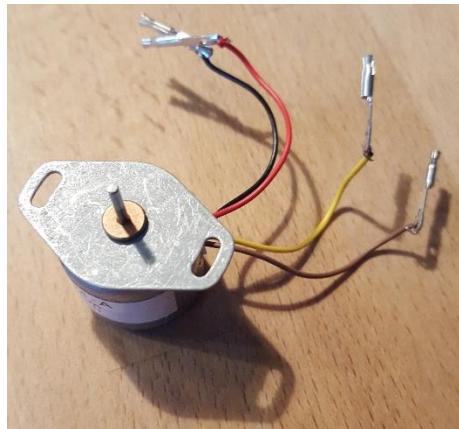
Schritt	A	\bar{A}	B	\bar{B}
1	1	1	0	0
2	1	0	0	1
3	0	0	1	1
4	0	1	1	0

Der Quellcode für eine Ansteuerung im Vollschrittbetrieb hierzu sieht wie folgt aus:

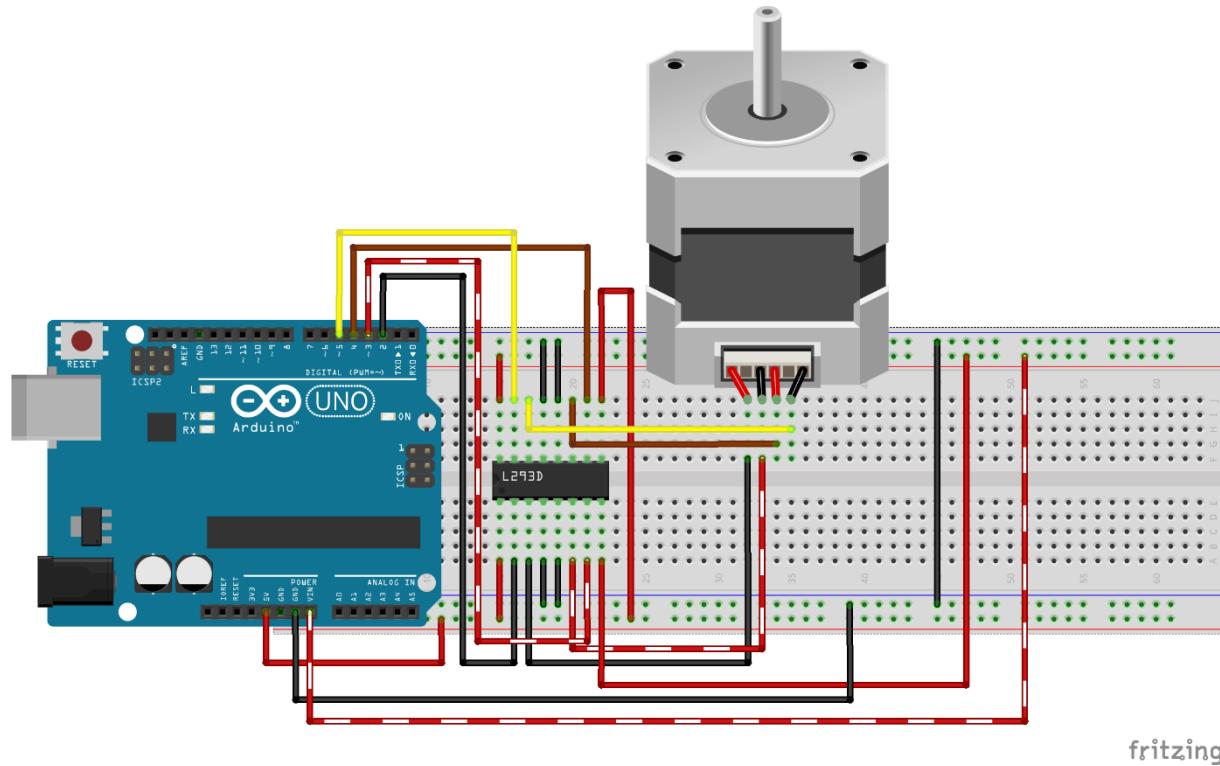
```
void setup() {  
    pinMode (2,OUTPUT);  
    pinMode (3,OUTPUT);  
    pinMode (4,OUTPUT);  
    pinMode (5,OUTPUT);  
}  
  
void loop(){  
    digitalWrite(2,HIGH);  
    digitalWrite(3,HIGH);  
    digitalWrite(4,LOW);  
    digitalWrite(5,LOW);  
    delay(5);  
  
    digitalWrite(2,LOW);  
    digitalWrite(3,HIGH);  
    digitalWrite(4,HIGH);  
    digitalWrite(5,LOW);  
    delay(5);  
  
    digitalWrite(2,LOW);  
    digitalWrite(3,LOW);  
    digitalWrite(4,HIGH);  
    digitalWrite(5,HIGH);  
    delay(5);  
  
    digitalWrite(2,HIGH);  
    digitalWrite(3,LOW);  
    digitalWrite(4,LOW);  
    digitalWrite(5,HIGH);  
    delay(5);  
}
```

Ansteuerung eines bipolaren Schrittmotors mit dem Arduino

Einen sehr günstigen Schrittmotor zu Versuchszwecken oder einfachen Applikationen erhält man bei Pollin. Die Bipolarität erkennt man an den vier Anschlüssen. Aus dem Datenblatt entnimmt man die Belegung:



Der Motor wird wie folgt mit dem Arduino und einem L293D aufgebaut:



fritzing

Der Quellcode des Vollschriftbetriebes:

```
int pinA = 2;
int pinNA = 3;
int pinB = 4;
int pinNB = 5;
int zeit = 50;

void setup() {
    pinMode(pinA, OUTPUT);
    
```

```

pinMode (pinNA, OUTPUT) ;
pinMode (pinB, OUTPUT) ;
pinMode (pinNB, OUTPUT) ;
}

void loop () {
    digitalWrite (pinA, HIGH) ;
    digitalWrite (pinNA, LOW) ;
    digitalWrite (pinB, HIGH) ;
    digitalWrite (pinNB, LOW) ;
    delay (zeit) ;

    digitalWrite (pinA, LOW) ;
    digitalWrite (pinNA, HIGH) ;
    digitalWrite (pinB, HIGH) ;
    digitalWrite (pinNB, LOW) ;
    delay (zeit) ;

    digitalWrite (pinA, LOW) ;
    digitalWrite (pinNA, HIGH) ;
    digitalWrite (pinB, LOW) ;
    digitalWrite (pinNB, HIGH) ;
    delay (zeit) ;

    digitalWrite (pinA, HIGH) ;
    digitalWrite (pinNA, LOW) ;
    digitalWrite (pinB, LOW) ;
    digitalWrite (pinNB, HIGH) ;
    delay (zeit) ;

}

}

```

Motorshields

Derzeit gibt es einige Motorshields für den Arduino auf dem Markt. Die meisten Shields beruhen entweder auf dem L293D oder dem L298N. Ersterer ist schon besprochen worden.

Shield mit dem L293D



Preislich liegt das Shield bei ca. 4€ und ist damit sehr günstig, um kleinere Projekte mit Motoren zu realisieren. Es gibt Anschlüsse für die Betriebsspannung und insgesamt vier Gleichstrommotoren bzw. zwei Schrittmotoren. Weiter können zwei Servos angeschlossen werden. Bei den günstigen Shields werden leider nur die analogen Eingänge aus dem Shield rausgeführt und entsprechende Steckbuchsen müssen noch aufgelötet werden.

Bearbeiten Sie nun die Aufgaben zum Schrittmotor.

Zeitmessung

Der Arduino stellt verschiedene Funktionen zur Messung von Zeit zur Verfügung:

Funktionsname	Funktion
millis()	Die Funktion liefert die Anzahl der Millisekunden zurück, die seit dem Start des Programms vergangen sind. Der Rückgabewert ist ein Wert vom Typ: unsigned long. Die Funktion kann maximal 50 Tage korrekte Werte liefern
micros()	Die Funktion liefert die Anzahl der Mikrosekunden zurück, die seit dem Start des Programms vergangen sind. Der Rückgabewert ist ein Wert vom Typ: unsigned long. Die Funktion kann maximal 90 Minuten korrekte Werte liefern.

Ruft man eine der beiden Funktionen zweimal auf und subtrahiert die gelieferten Werte, bekommt man die Zeit dazwischen als Ergebnis und kann mit dieser weiterrechnen. Im nachfolgenden Programm wird berechnet, wie lange der Ablauf einer 100ms-Verzögerung dauert.

Kommentiertes Beispielprogramm zur Zeitmessung:

```
//die benötigten Variablen
unsigned long millisZeit1;
unsigned long millisZeit2;
unsigned long mikrosZeit1;
unsigned long mikrosZeit2;
unsigned long millisZeit;
unsigned long mikrosZeit;

//Die serielle Schnittstelle wird zu Ausgabezwecken initialisiert
void setup(){
    Serial.begin(9600);
}

void loop(){
    //die Zeit seit dem Start des Programms wird gemessen
    millisZeit1 = millis();
    mikrosZeit1 = micros();
    delay(100);

    //die Zeit seit dem Start des Programms wird erneut gemessen
    millisZeit2 = millis();
    mikrosZeit2 = micros();
```

```
//die Zeitdifferenz wird ausgerechnet
millisZeit = millisZeit2 - millisZeit1;
mikrosZeit = mikrosZeit2 - mikrosZeit1;

//Ausgabe der Messergebnisse auf die serielle Schnittstelle
Serial.print("Millisekunden: ");
Serial.println(millisZeit);
Serial.print("Mikrosekunden: ");
Serial.println(mikrosZeit);
delay(1000);
}
```

Zeitmessung eines Impulses

Die Eingebaute Funktion `pulseIn()` ermöglicht es dem Anwender, die Länge eines Impulses zu ermitteln. Die Funktion bekommt die Pinnummer des Arduino sowie die Art des Impulses übergeben. Beispiel:

```
unsigned long zeit1 = pulseIn(3,HIGH);
```

Die Methode startet einen Timer dann, wenn am Pin 3 HIGH-Signal anliegt. Der Timer stoppt, wenn an Pin 3 LOW anliegt. Die Methode gibt dann die Anzahl der vergangenen Mikrosekunden zurück und speichert sie in einer Variable vom Typ `unsigned long`.

```
unsigned long zeit2 = pulseIn(4,LOW);
```

Die Methode startet den Timer bei LOW-Signal an Pin 4 und stoppt ihn bei HIGH Signal.

Entfernung messen mit Ultraschall

Der Sensor HC-SR04 (zwischen 1 und 2€ pro Stück) beinhaltet einen Ultraschallsender, Ultraschallempfänger und die nötige Elektronik.



8

Er wird mit 5V betrieben (Vcc und GND) und ist somit sehr gut für den Arduino geeignet. Über ein kurzes HIGH-Signal am Triggerpin (Trig) schickt der Sensor einen Ultraschallping raus. Am Pin Echo kann dann der Antwortping abgegriffen werden. Misst man die vergangene Zeit zwischen Senden

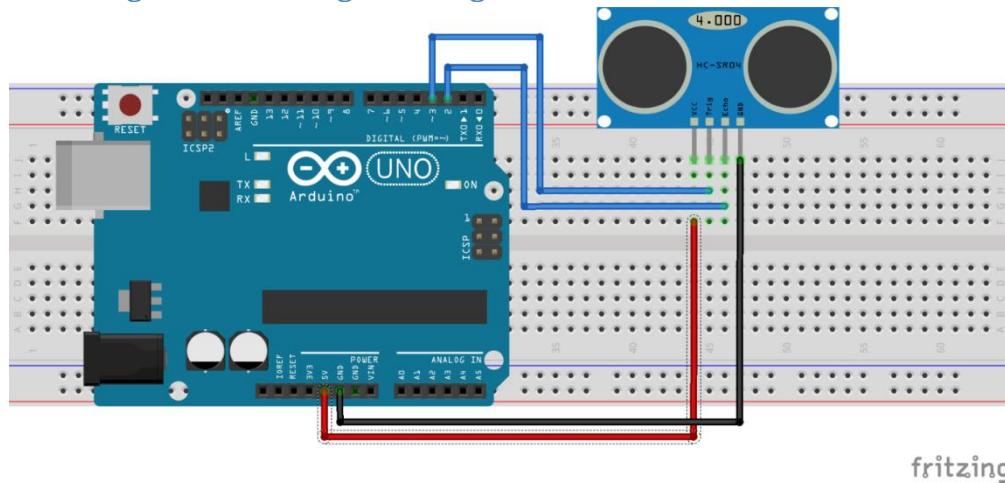
⁸ Bild aus: <https://www.aimagin.com/hc-sr04-ultrasonic-sensor.html>

und Empfangen kann man die Entfernung des Gegenstandes berechnen, an dem der Ping reflektiert ist. Bei einer Schallgeschwindigkeit von 340 m/s und der gemessenen Zeit t in Mikrosekunden kommt man auf folgende Gleichung zur Berechnung des Abstandes s in cm (man muss beachten, dass der Ping sowohl Hin- als auch Rückweg zurücklegt):

$$s = \frac{t}{29 \cdot 2}$$

Das folgende Programm mit der entsprechenden Schaltung setzt eine solche Entfernungsmessung um und schickt den gemessenen Abstand auf die serielle Schnittstelle.

Schaltung zur Entfernungsmessung:



Kommentierter Programmcode zur Entfernungsmessung:

```
//setzen der Pins
int TriggerPin = 3;
int EchoPin = 2;

//Konfigurieren der Ein- und Ausgabepins sowie der seriellen Schnittstelle
void setup()
{
    pinMode(TriggerPin, OUTPUT);
    pinMode(EchoPin, INPUT);
    Serial.begin(9600);
}

void loop()
{
    //Variable zur Messung der Zeit bzw. der Entfernung
    long zeit, cm;

    //Ein Ping wird gesendet
    digitalWrite(TriggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(TriggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(TriggerPin, LOW);

    // The same pin is used to read the signal from the PING)): a HIGH
    //Die Zeit bis zum Eintreffen des Pings wird gemessen und
    zeit = pulseIn(EchoPin, HIGH);

    // die Zeit wird in cm umgewandelt
    cm = mikrosekundenInZentimeter(zeit);
```

```
//Nur wenn der Wert < als 500cm ist:
if (cm <= 500)
{
    Serial.print(cm);
    Serial.println("cm");
}
else
{
    Serial.println("Messung ungenau! Entfernung groesser als 5 Meter !!!");
}
delay(100);
}

long mikrosekundenInZentimeter(long zeit)
{
    //Die Zeit muss halbiert werden (Hin- und Rückweg des Pins)
    //Bei einer Schallgeschwindigkeit von 340m/s ergibt sich ein Faktor von 29 für
    // cm/us
    return zeit / 29 / 2;
}
```

Bearbeiten Sie nun die Aufgaben zur Zeit- / Entfernungsmessung

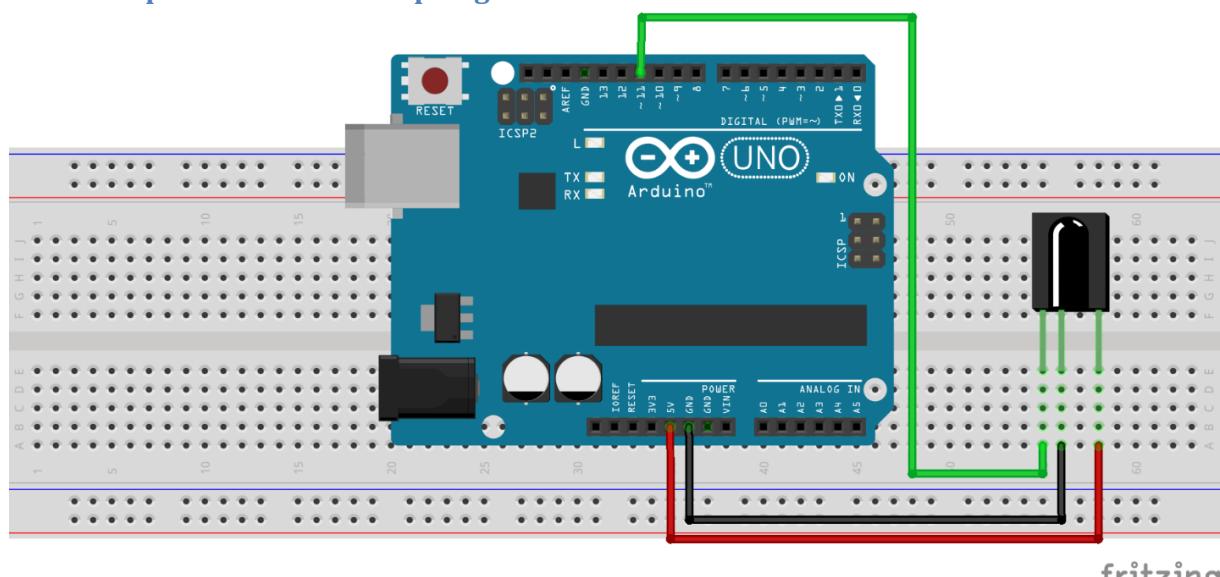
Infrarot

Infrarotsignale empfangen

Infrarotempfänger sind günstig zu erwerben. Sie eignen sich dazu, entweder Daten von Fernbedienungen auszuwerten, oder aber eigene Datenübertragungen per Infrarot zu realisieren. Die meisten Empfänger arbeiten auf einer Frequenz von 38kHz und haben einen Anschluss für Masse, Spannungsversorgung sowie einen Datenausgang. Achtung: Die Anschlüsse der Bauteile sind immer dem entsprechenden Datenblatt zu entnehmen!



Anschlussplan des Infrarotempfängers



Programmierung des Infrarotempfängers

Zur einfacheren Programmierung wird die Bibliothek IRremote verwendet. Wichtig ist bei der Verwendung der Bibliothek, dass die ursprüngliche IR-Bibliothek: RobotIRremote aus dem Bibliotheksverzeichnis der Entwicklungsumgebung (c:\Programme\Arduino\libraries) gelöscht wird!

Quellcode zum Auslesen einer Fernbedienung

```
#include <IRremote.h>
#include <IRremoteInt.h>

IRrecv irrecv(11);
decode_results results;

void setup() {
    Serial.begin(9600);
    irrecv.enableIRIn();

}

void loop() {
    if (irrecv.decode(&results)) {
        Serial.println(results.value, DEC);
        irrecv.resume();
    }
}
```

Bearbeiten Sie nun die Aufgaben zum IR-Empfänger

Ansteuern von LCD-Displays

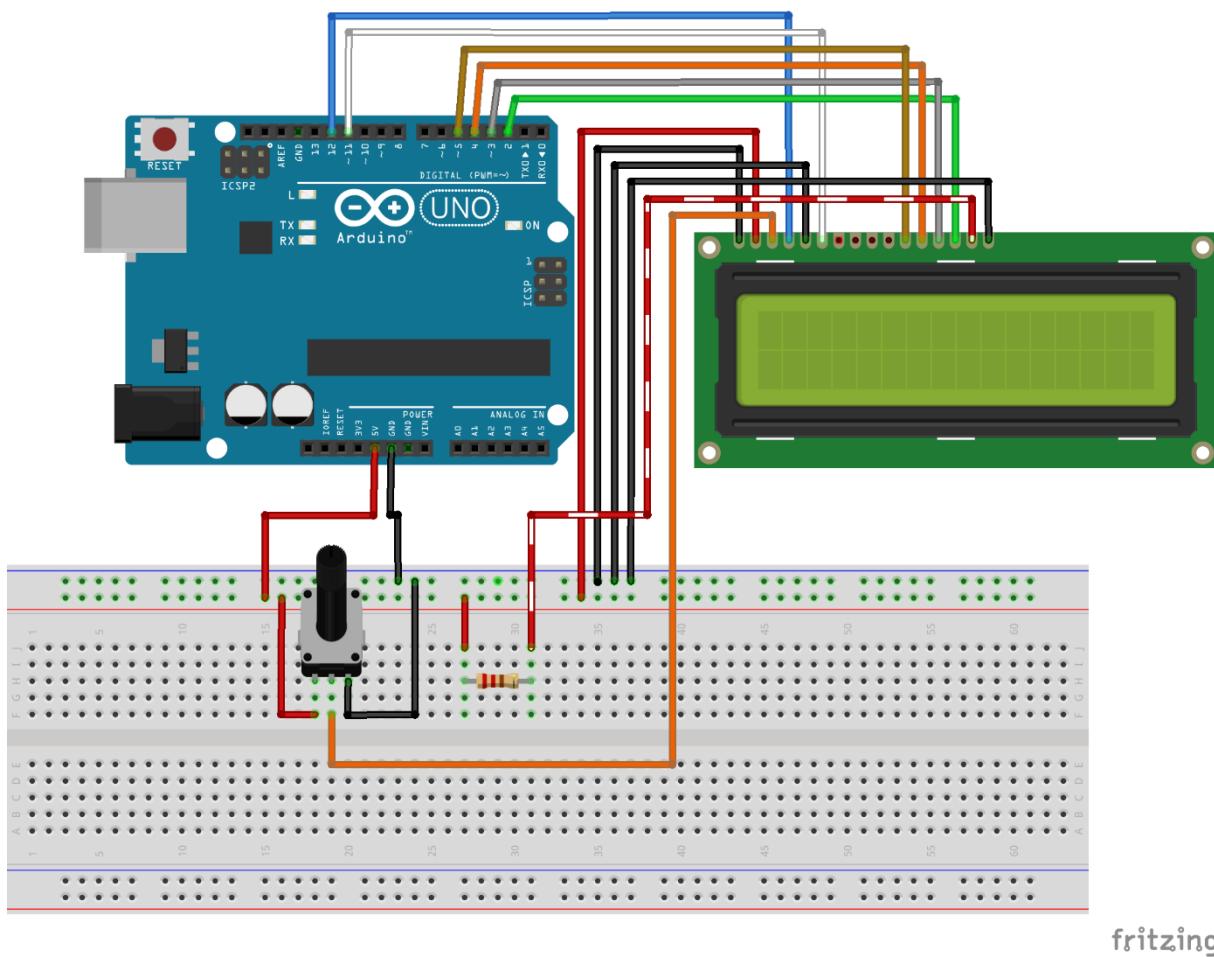
LCD-Displays können dazu dienen, Messdaten oder Texte sichtbar zu machen. Sie sind in verschiedenen Größen und Text- bzw. Hintergrundfarben recht günstig (6€) erhältlich. Beim Kauf der Displays sollte man darauf achten, dass diese den Standard **HD44780** von Hitachi erfüllen. Dies ist aber bei den meisten Displays der Fall. Ich empfehle für Experimentierzwecken die Anschlusspins des Displays über eine Sockelleiste einfacher zugänglich zu machen (Bezugsquellen im Anhang).

Die Displays können in den unterschiedlichsten Modi betrieben werden. Hier wird nur der 4-Bit-Modus verwendet. Für den Arduino wird schon eine fertige Bibliothek mitgeliefert, die man im eigenen Projekt nur importieren (`include`) muss. Die Befehle der Bibliothek sind nach dem kommentierten Beispielprogramm erklärt.

Anschlussplan und Beispielprogramm für den Betrieb eines LCD- Displays

Anschlussplan für das LCD-Display:

Erklärung: Die Pins des Displays zu den digitalen Ausgängen des Arduino dienen dem Datentransfer zur späteren Darstellung von Texten. Die Pins sind von links nach rechts von 1 bis 16 durchnummieriert. Pin 1 muss an Masse, Pin2 an 5V gelegt werden. Dies stellt die Betriebsspannung des Displays dar. An Pin 3 wird der Mittelabgriff des Potentiometers angeschlossen. Hiermit wird die Helligkeit der Hintergrundbeleuchtung des Displays eingestellt. Pin 5 wird ebenfalls an Masse angeschlossen. Dies ist der Read / Write – Anschluss des Displays. Die Read-Funktion wird nur dazu verwendet, um vom Display zu erfahren, ob es noch mit der Datenverarbeitung beschäftigt ist. Wählt man im Programm die Wartezeiten ausreichend groß, so kann auf diese Funktion verzichtet werden, was über ein LOW-Signal (Masse) umgesetzt wird.



Beispielprogramm für das LCD-Display:

```
// Importieren der entsprechenden Bibliothek
#include <LiquidCrystal.h>

// Ein Objekt wird erzeugt. Übergeben werden die Anschlusspins wie
// oben beschrieben
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
    // die Anzahl der Zeichen und der Zeilen wird angegeben
    lcd.begin(16, 2);
```

```
// ein Text wird auf das LCD-Display geschrieben
lcd.print("Hallo Welt");
}

void loop() {
    // Das Display wird ausgeschaltet und eine halbe Sekunde gewartet
    lcd.noDisplay();
    delay(500);
    // Das Display wird eingeschaltet und eine halbe Sekunde gewartet
    lcd.display();
    delay(500);
}
```

Befehlsübersicht der Bibliothek: `LiquidCrystal.h`

Befehl	Erklärung	Parameter
<code>LiquidCrystal(int rs, int enable, int d4, int d5, int d6, int d7)</code>	Erzeugt ein LCD-Objekt	rs: Pin 4 am Display, wählt das Befehls- oder Datenregister aus enable: Pin 6 am Display, Takt d4: Pin 11 am Display, Daten d5: Pin 12 am Display, Daten d6: Pin 13 am Display, Daten d7: Pin 14 am Display, Daten
<code>begin(int zeichen, int zeilen)</code>	Stellt die Größe des Displays ein	zeichen: Anzahl der Zeichen in einer Zeile zeilen: Anzahl der Zeilen des Displays
<code>clear()</code>	Löscht den dargestellten Text.	keine
<code>setCursor(int s, int z)</code>	Setzt den Cursor auf die angegebenen Position	s: Spalte z: Zeile
<code>print(String text)</code>	Schreibt den übergebenen Text an die aktuelle Cursorposition.	text: darzustellender Text
<code>cursor()</code>	Schaltet den Cursor ein.	keine
<code>noCursor()</code>	Schaltet den Cursor aus.	keine
<code>blink()</code>	Der Cursor blinkt.	keine
<code>noBlink()</code>	Der Cursor binkt nicht.	keine
<code>display()</code>	Das Display wird eingeschaltet.	keine
<code>noDisplay()</code>	Das Display wird ausgeschaltet.	keine

Bearbeiten Sie nun die Aufgaben zum LCD-Display.

SPI-Bus

Das Serial Peripheral Interface ist ein Bus-System, welches von der Firma Motorola entwickelt wurde. Über ein Master-Slave-Prinzip kommunizieren digitale Bausteine miteinander. SPI ist voll duplexfähig, Daten können also gleichzeitig gesendet und empfangen werden. Mittlerweile gibt es Bausteine aus allen möglichen Bereichen mit SPI Anschlussmöglichkeit

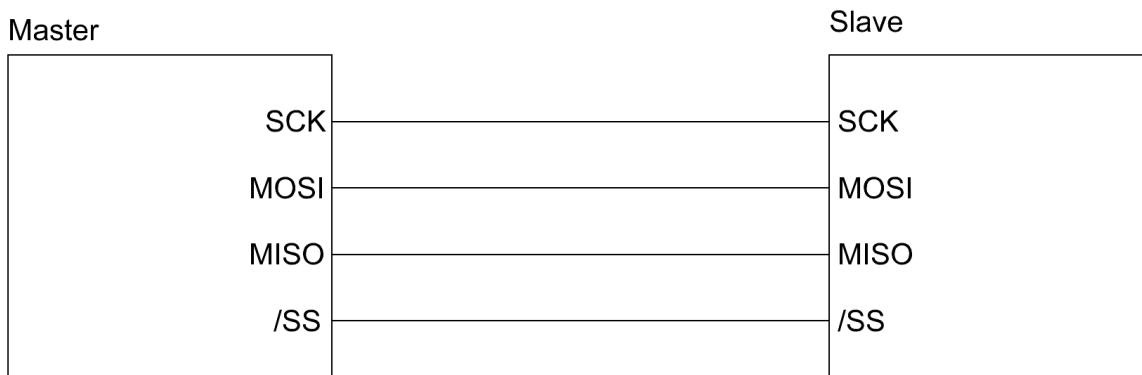
Der Bus besteht aus vier Leitungen:

MOSI: Master Out Slave In, Daten werden vom Master an den Slave gesendet.

MISO: Master In Slave Out, Daten werden vom Slave an den Master gesendet

SCK: Serial Clock, das Taktsignal

/SS: Slave Select, auch als CS: Chip Select bezeichnet, über ein LOW-Signal wird ein Slave ausgewählt.



Die Leitungen MOSI oder MISO müssen nicht unbedingt verwendet werden, wenn der anzusteuерnde Chip beispielsweise nur Daten empfängt. Dem Empfängerbaustein wird über ein LOW-Signal auf /SS angesprochen und ist nur in diesem Zustand bereit zu kommunizieren. Will man mehrere Bausteine ansprechen, so kann das nur über mehrere digitale Ausgänge des Arduino geschehen oder aber über eine speziell hierfür entwickelte Logikschaltung.

SPI unterstützt unterschiedliche Datenraten, 2Mbit/s, 1 Mbit/s, 0,5 Mbit/s und 0,0625Mbit/s sind derzeit die gängigsten Raten.

Die genaue Datenübertragung ist beim SPI nicht definiert. Es haben sich in der Praxis aber vier leicht unterschiedliche Modi durchgesetzt, die durch die Einstellung der Clock-Parity (CPOL) und der Clock-Phase (CPHA) definiert werden:

Modus 0: CPOL = 0 CPHA = 0

Modus 0: CPOL = 0 CPHA = 1

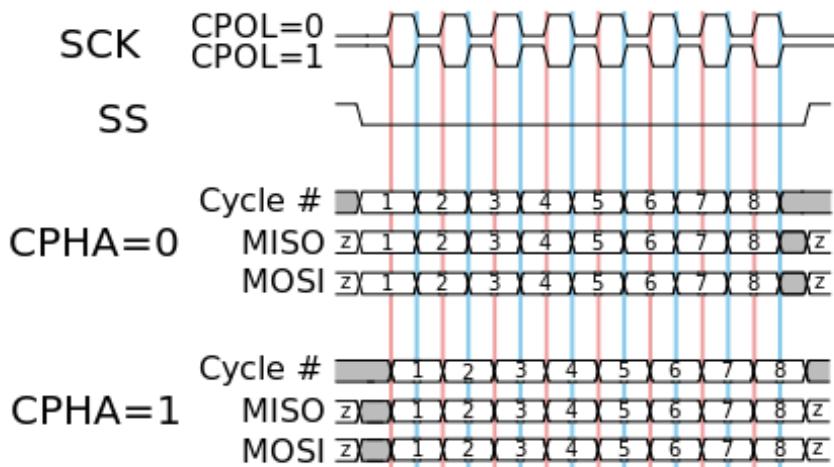
Modus 0: CPOL = 1 CPHA = 0

Modus 0: CPOL = 1 CPHA = 1

CPOL definiert, welcher Flankenwechsel nach Aktivierung eines ICs vom Clock-Signal zu Beginn anliegt. Bei CPOL = 0 liegt ein Flankenwechsel von LOW nach HIGH an. CPHA definiert, ob mit dem ersten Flankenwechsel (CPHA = 0) oder mit dem zweiten (CPHA = 1) begonnen wird, auf den Datenkanälen Daten zu übertragen.

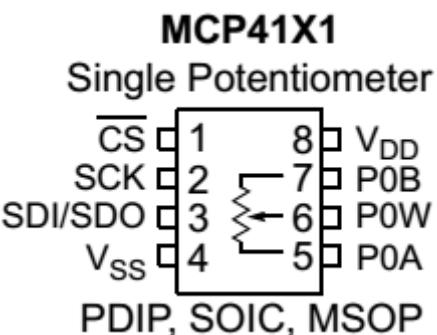
Welcher Modus verwendet wird, hängt vom angeschlossenen Baustein ab. Seinem Datenblatt können die entsprechenden Informationen entnommen werden.

Datenübertragung bei den verschiedenen Modi⁹:

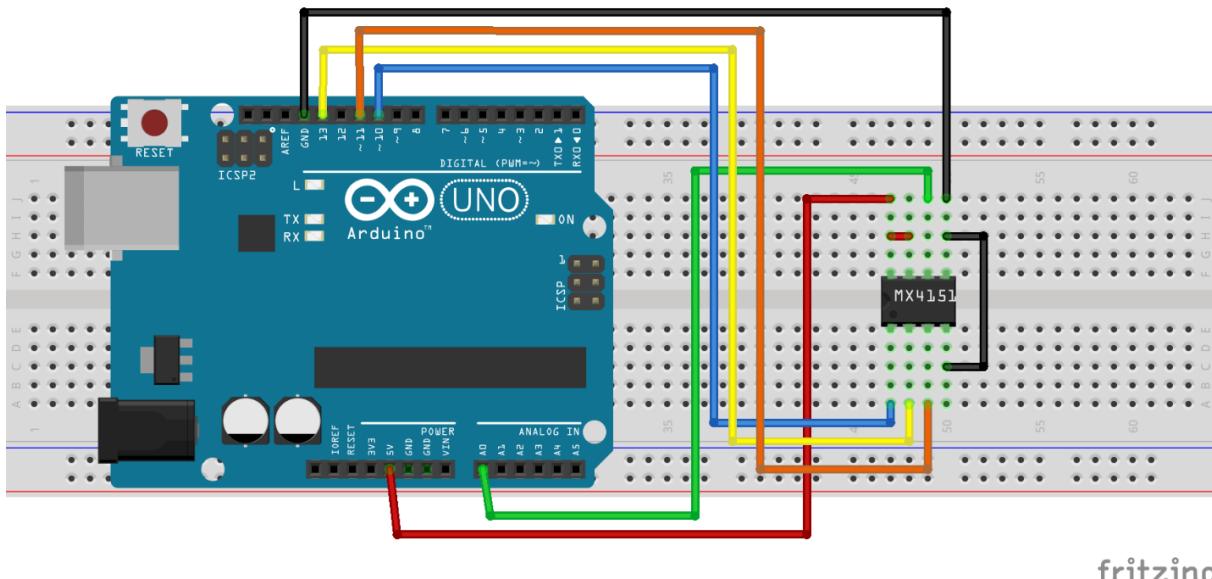


Programmierung eines digitalen Potentiometers MCP 4151

Der MCP 4151 stellt ein einfaches digital einstellbares Potentiometer mit einer Auflösung von 8Bit dar. Der einzustellende Widerstand kann per Register zwischen den Werten 5kΩ, 10kΩ, 50kΩ und 100kΩ ausgewählt werden.



⁹ Grafik aus: http://de.wikipedia.org/wiki/Serial_Peripheral_Interface



fritzing

Quellcode zur Ansteuerung des SPI-ICs

```
#include <SPI.h>

int chipSelect = 10;
int value = 0;

void setup() {
    Serial.begin(9600);
    SPI.begin();
}

void loop() {
    //chipSelect auf LOW --> Chip wird ausgewählt
    digitalWrite(chipSelect, LOW);
    //Chip bekommt Daten
    SPI.transfer(0);
    SPI.transfer(value);
    // chipSelect auf HIGH --> Datenübertragung beendet
    digitalWrite(chipSelect, HIGH);
    // Einlesen des analogen Eingangs
}
```

```

Serial.println(value);

int analog0 = analogRead(0);

Serial.println(analog0);

delay(500);

value++;

}

```

Bearbeiten Sie nun die Aufgaben zum SPI-Bus.

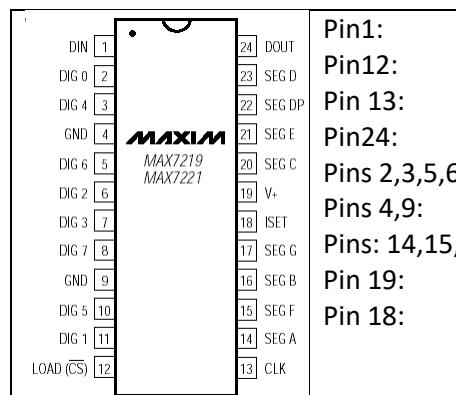
Ansteuern einer LED-Matrix oder von mehreren 7-Segment-Anzeigen

Die dreizehn digitalen Ein- / Ausgabepins des Arduino sind schnell erschöpft und reichen beispielsweise kaum aus, um eine 5x7 LED-Matrix anzusteuern. Sicherlich könnte man über eine Porterweiterung (siehe Abschnitt I²C) mehr Pins erzielen. Mit dem MAX 7219 gibt es einen Baustein, der über den SPI-Bus diese Aufgaben übernehmen kann:

MAX 7219

Dieser Baustein realisiert einen multipel einsetzbaren Display-Driver. An Außenbeschaltung ist nur ein einziger Widerstand notwendig. Die benötigten Vorwiderstände für LEDs sind schon integriert. Der Baustein eignet sich zum Ansteuern von LED-Matrizen, 7-Segment-Anzeigen, Bargraphs etc. Mehrere Bausteine sind kaskadierbar!

Die Anschlussbelegung ist wie folgt:

	Pin1: Pin12: Pin 13: Pin24: Pins 2,3,5,6,7,10,11: Pins 4,9: Pins: 14,15,16,17,20,21,22,23 Pin 19: Pin 18:	Dateneingang vom SPI-Bus Chip Select (SPI) Clock (SPI) Datenausgang (SPI oder Kaskadierung) Ausgänge zur Ansteuerung der Leds Masse Ausgänge zur Ansteuerung der Segmente Betriebsspannung (5V) Spannung zur Helligkeitssteuerung
---	--	---

Der Widerstand, über den die Betriebsspannung an I_{SET} (Pin 18) ist aus der folgenden Tabelle zu entnehmen:

I_{seg} (mA)	V_{LED} (V)				
	1,5	2,0	2,5	3,0	3,5
40	12,2 kΩ	11,8 kΩ	11,0 kΩ	10,6 kΩ	9,69 kΩ
30	17,8 kΩ	17,1 kΩ	15,8 kΩ	15,0 kΩ	14,0 kΩ
20	29,8 kΩ	28,0 kΩ	25,9 kΩ	24,5 kΩ	22,6 kΩ

10	66,7 kΩ	63,7 kΩ	59,3 kΩ	55,4 kΩ	51,2 kΩ
----	---------	---------	---------	---------	---------

Programmierung des MAX 7219

Die einzelnen Funktionen des Bausteins kann man dem Datenblatt entnehmen. Etwas einfacher geht es über die Einbindung der Bibliothek: LedControl.h

Die wichtigsten Befehle der Bibliothek:

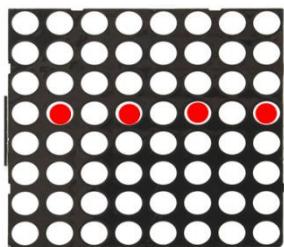
Befehl	Parameter	Funktion
LedControl lc=LedControl(int data,int clock,int cs,int num);	data: Datenpin (SPI) clock: Clockpin (SPI) cs: Chipselectpin (SPI) num: Anzahl der Bausteine	Erzeugt ein Objekt lc mittels dem man auf die Funktionen zugreifen kann.
shutdown(int device,boolean b);	device: Nummer des Bausteines (beginnend mit 0) b: Ruhemodus aktivieren (true) oder Baustein aufwecken (false)	Bestimmt, ob sich der Baustein im Ruhemodus befindet oder nicht. Zu Beginn ist jeder Baustein im Ruhemodus!
setIntensity(int num,int intensity);	num: Nummer des Bausteins intensity: Darstellungshelligkeit	Setzt die Darstellungshelligkeit der angeschlossenen LEDs
clearDisplay(int num);	num: Nummer des Bausteines	Alle LEDs sind aus.
setRow(int num,int digit,int row);	num: Nummer des Bausteines digit: 7-Segment oder Auswahl der Reihe einer Matrix row: Anzusteuernde LEDs im binären Format	Steuert die ausgewählten LEDs im ausgewählten Digit an.

Die Funktion setRow() bedarf einer genaueren Erklärung:

setRow(int num, int digit, int row) bei einer Matrix:

Reihe: 7								
Reihe: 6								
Reihe: 5								
Reihe: 4								
Reihe: 3								
Reihe: 2								
Reihe: 1								
Reihe: 0								
Wertigkeit:	128	64	32	16	8	4	2	1

Angenommen, es gibt nur einen Baustein und man möchte das folgende Bild erzeugen:



Dann ist der Befehl:

`setRow(0, 4, 85)`

erforderlich.

Die 85 setzt sich aus der Wertigkeit der LED zusammen: $85 = 1 + 4 + 16 + 64$.

Einfacher ist es im binären Format zu programmieren:

`setRow (0, 4, 0B01010101);`

Im nachfolgenden Programm wird ein Herz dargestellt, welches nach und nach immer heller wird.

Programm zur Absteuerung einer LED-Matrix

```
#include "LedControlMS.h"
```

```
LedControl lc=LedControl(12,11,10,1);
int intensity = 0;
void setup() {
/*
The MAX72XX is in power-saving mode on startup,
we have to do a wakeup call
*/
lc.shutdown(0,false);
/* Set the brightness to a medium values */
lc.setIntensity(0,15);
```

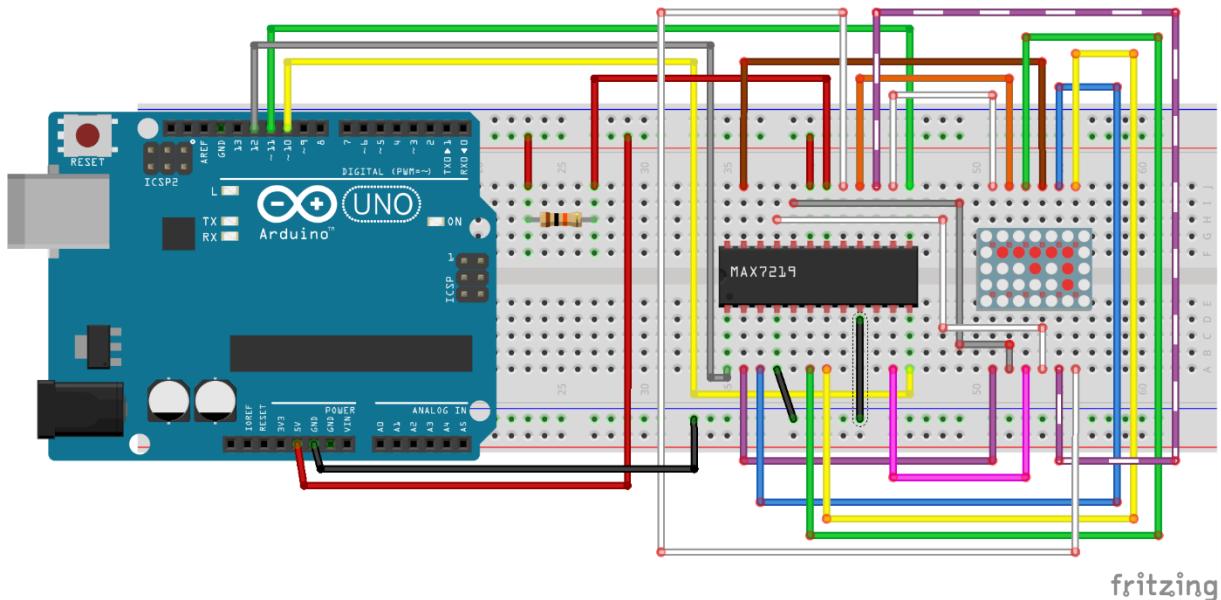
```

/* and clear the display */
lc.clearDisplay(0);
}

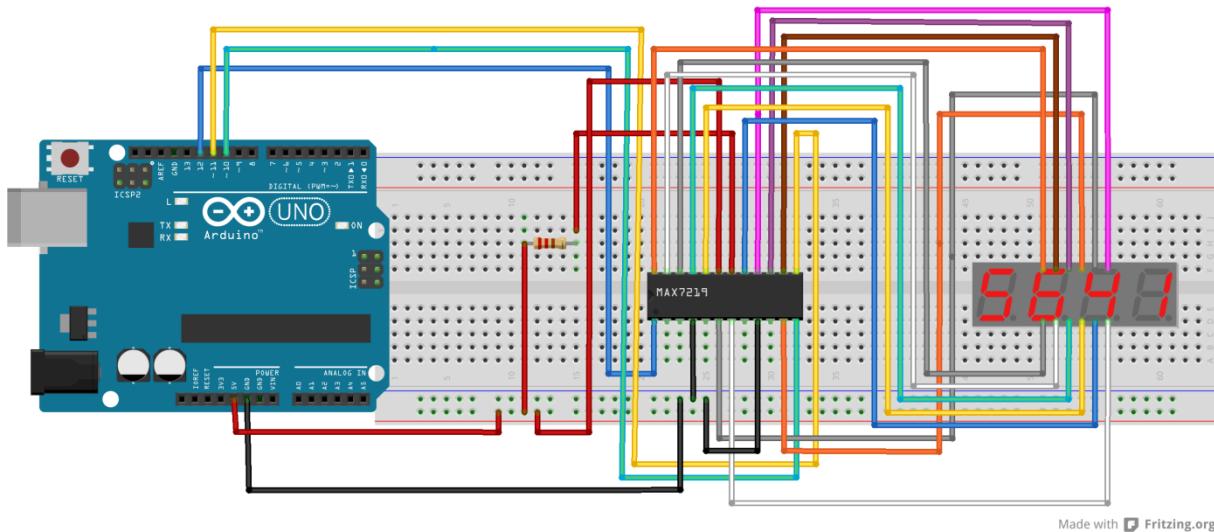
void loop() {
    lc.setIntensity(0,intensity);
    lc.setRow(0,0,0B1000);
    lc.setRow(0,1,0B10100);
    lc.setRow(0,2,0B100010);
    lc.setRow(0,3,0B101010);
    lc.setRow(0,4,0B10100);
    delay(500);
    lc.clearDisplay(0);
    delay(500);
    intensity++;
    if(intensity==16){
        intensity = 0;
    }
}

```

Schaltplan zur Ansteuerung einer 5x7-LED-Matrix mit dem MAX 7219



Eine Sieben-Segment-Anzeige



Bearbeiten Sie nun die Aufgaben zum MAX 7219.

Der I²C-Bus

Der I²C-Bus ist ein serieller Bus mit nur zwei notwendigen Signalleitungen: Daten und Takt. Die benötigte Betriebsspannung und Masse werden vom USB-Bus geliefert, so dass man keinerlei sonstige Spannungsquellen verwenden muss.

Der Bus benötigt einen Masterbaustein, der den Datenfluss kontrolliert. Diese Funktion kann vom Arduino übernommen werden. Der Datenausgang liegt beispielsweise beim Arduino an Pin A5, der Taktausgang an Pin A4 an.

Das Busprinzip:

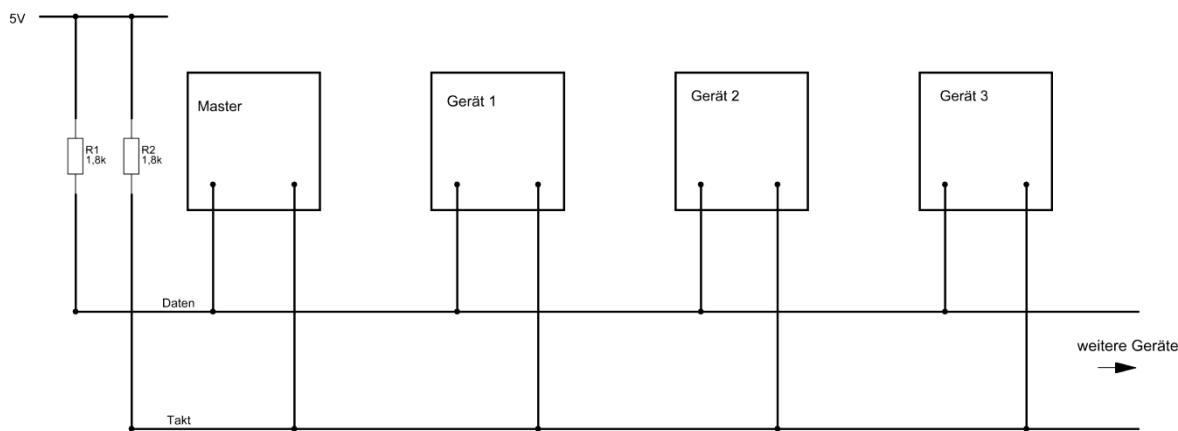


Abb.: Aufbau I²C-Bus

Wie man sieht, liegen die beiden Signalleitungen über jeweils einen $1,8\text{k}\Omega$ an 5V. Diese Pullup-Widerstände sind im Arduino schon integriert. Der Master sendet das Taktsignal,

auch dies erledigt der Arduino in den Frequenzen 50kHz, 100kHz und 400kHz (höhere Frequenzen sind ebenfalls möglich). IC-Bausteine wie der AT24C128 (EEprom), DS1803 (programmierbares Potentiometer), LM75 (Temperatursensor), PCF8570 (256 x 8Bit RAM), PCF8574A (8Bit IO-Erweiterung), PCF8591 (4-fach AD-Wandler, 1 DA-Wandler) und SD20 (Servocontroller) sind leicht anschließ- und programmierbar. Am Beispiel des PCF8591 werden der Aufbau und die Programmierung dargestellt. Will man andere ICs in den Bus integrieren muss man die Sendeabfolge auf dem Bus kennen:

Zu Beginn wird vom Master ein Adressbyte gesendet. Das Adressbyte setzt sich aus sieben Bit Adresse und einem Bit Read/Write zusammen. Bei den meisten ICs sind die vier höchsten Bits festgesetzt, die drei niedrigsten sind frei einstellbar, so dass von einem IC acht Stück verwendet werden können. Anschließend können zusätzliche Controllbytes gesendet werden. Dann kann man Daten lesen oder weiter an die ICs schicken.

Beispiel PCF8591:

Bit	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	R/W
Wert	1	0	0	1	x	x	x	0 schreiben 1 Lesen

Setzt man also alle wählbaren Bits auf Low und will zum IC schreiben muss das Byte 10010000 (Wert: 0x90) gesendet werden.

Anschließend werden ein oder mehrere Control Bytes gesendet. Diese definieren, wie der entsprechende Baustein arbeiten soll.

PCF8591

Der Baustein beinhaltet vier AD – Wandler sowie einen DA – Wandler jeweils in 8 Bit Auflösung. Die Betriebsspannung (Anschluss V_{DD} Pin 16) entspricht in unserem Fall der Referenzspannung (Anschluss V_{REF} Pin 14) von 5V und wird vom USB-Bus abgenommen. Die negative Betriebsspannung (VSS Pin 8) wird auf Masse gelegt.

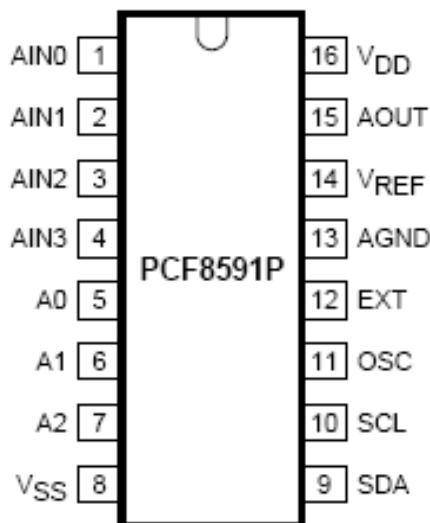


Abb.: Pinbelegung des PCF 8591

An Pin 1 bis 4 können die externen analogen Signale angelegt werden. Über Pin 5 bis 7 wird die Adresse des Bausteins eingestellt. Die einzelnen Pins werden entweder auf Masse oder an 5V gelegt. In der späteren Schaltung sollte die Möglichkeit geschaffen werden, die Adresse per Jumper einzustellen. Die Masse der analogen Eingänge kann rausgeführt werden, wird aber in diesem Anwendungsfall auf die Masse des USB-Busses gelegt. Mit Pin 12 wird eingestellt, ob man einen externen Oszillator verwenden will, liegt dieser Pin auf Masse, so kann man auch auf den Anschluss eines externen Taktes (Pin 11) verzichten. An Pin 15 kann der analoge Ausgang abgenommen werden.

Bleiben nur noch die beiden I²C-Bus Signalleitungen übrig. Die Datenleitung wird an Pin 9 und die Takteleitung an Pin 10 angelegt.

Die analogen Eingänge können in vier verschiedenen Modi arbeiten:

Modus 0:	Alle vier Eingänge arbeiten single ended, das bedeutet alle Eingänge messen separat die Spannung gegen Masse.
Modus 1:	Drei differentielle Eingänge sind vorhanden, wobei die Eingänge 0 bis 2 jeweils gegen Eingang 3 gemessen werden.
Modus 2:	Eingang 0 und Eingang 1 arbeiten single ended, Eingang 2 und Eingang 3 arbeiten gegeneinander differentiell.
Modus 3:	Jeweils Eingang 0, Eingang 1 und Eingang 2 und Eingang 3 arbeiten differentiell gegeneinander.

Der Modus und die Arbeitsweise des Bausteins werden mittels eines Controlbytes eingestellt welches sich wie folgt zusammensetzt:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	Enable DA	Mode	Mode		Autoincrement	AD-Channel	AD-Channel
0	x	x	x	0	x	x	x

Erklärung:

Bit 6: Wird dieses Bit auf HIGH gesetzt, so ist der DA-Wandler aktive, bei LOW nicht.

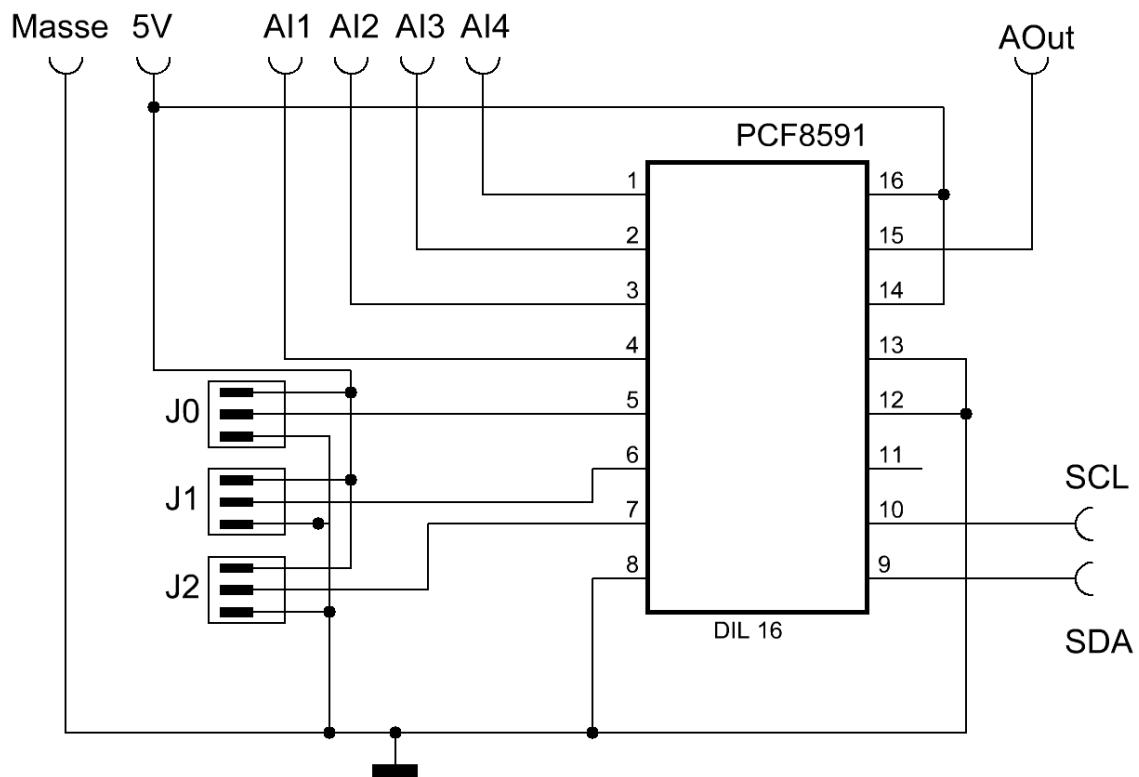
Bit 5 und 4: 00 wählt den Modus 0, 01 den Modus 1, 10 den Modus 2 und 11 den Modus 3.

Bit 2: Wird dieses Bit gesetzt, so wird bei jedem Lesezyklus der AD-Wandler gewechselt.

Bit 1 und 0: Der AD-Wandler wird ausgewählt: 00 wählt Kanal 0, 01 Kanal 1, 10 Kanal 2 und 11 Kanal 3.

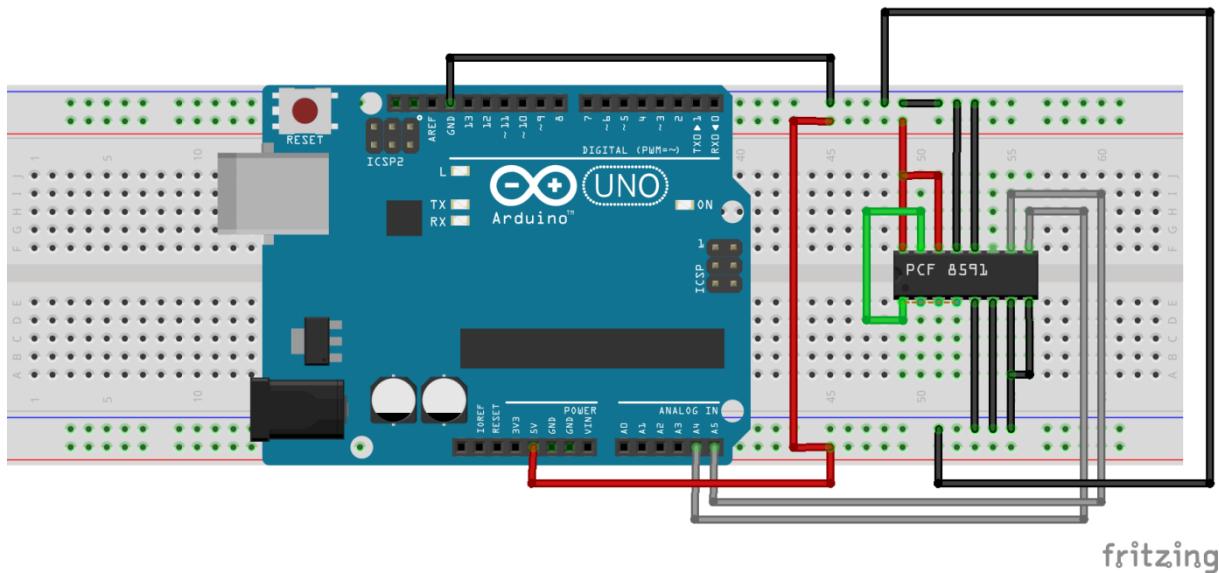
Beispiel:

Will man die vier Eingänge alle separat also im Modus 0 verwenden und den analogen Ausgang sowie den AD-Kanal 0 benutzen, so hat das Byte den Wert 01000000 also 0x40.



Schaltplan Analogmessung mit einem PCF 8591

Im Betrieb mit dem Arduino kann man einen PCF 8591 wie folgt verwenden:



Der Baustein wurde mit der Adresse 90x konfiguriert!

Die Programmierung des Bausteins funktioniert wie folgt¹⁰:

Für den I²C-Bus wird die Bibliothek Wire.h verwendet:

```
#include <Wire.h>
```

Die Adresse des Bausteins ist 90x. Im Arduino wird allerdings die Adresse nur ohne das Lese- / Schreibbit verwendet. Entweder man rechnet die Adresse um, oder man verwendet die Bitshiftoperation >>:

```
#define PCF8591 (0x90 >> 1)
//verschiebt den Wert 0x90 um ein Bit nach rechts und ordnet ihm der
//Konstante PCF8591 zu.
```

```
int da_wert=0;
```

```
int ad_wert;
```

Die nachfolgende Funktion gibt einen Wert auf dem DA-Wandler aus. Hierzu wird eine I²C-Verbindung mit dem Befehl `Wire.beginTransmission(adresse)` gestartet. Über den Befehl `Wire.write(byte)` wird dann das entsprechende Controllbyte an den IC geschickt und nachfolgend der gewünschte Wert ebenso. `Wire.endTransmission()` beendet die Kommunikation mit dem IC.

```
void setzeDAC(byte wert)
{
    Wire.beginTransmission(PCF8591);
    Wire.write(0x40);
    Wire.write(wert);
    Wire.endTransmission();
}
```

Die nachfolgende Funktion liest den Wert eines AD-Kanals aus. Die Funktion bekommt das entsprechende Controlbyte übergeben. Besonders ist hier, dass in der Kommunikation nur das Controllbyte gesendet wird. Anschließend wird über `Wire.requestFrom((int) PCF8591,2)` ein Empfang vom IC erwartet. Der Wert 2 bedeutet, dass 2 Bytes erwartet werden (siehe Datenblatt des PCF 8591). Liefert der IC einen Wert, so gibt die Funktion `Wire.available()` den Wert true zurück. Über `Wire.read()` wird dann ein Byte vom IC gelesen.

```
byte getADC(byte controll)
{
    Wire.beginTransmission(PCF8591);
```

¹⁰ Programm abgewandelt aus: http://www.horter.de/i2c/i2c-beispiele/arduino_1.html

```
Wire.send(config);
Wire.endTransmission();

Wire.requestFrom((int) PCF8591,2);
while (Wire.available())
{
    adc_value = Wire.receive();
    adc_value = Wire.receive();
}
return adc_value;
}
```

Die Setuproutine. Spannend ist hier nur der Befehl Wire.begin(). Der Initialisiert den I²C-Modus des Arduino.

```
void setup()
{
    pinMode(13, OUTPUT);
    Serial.begin(19200);
    Wire.begin();
}
```

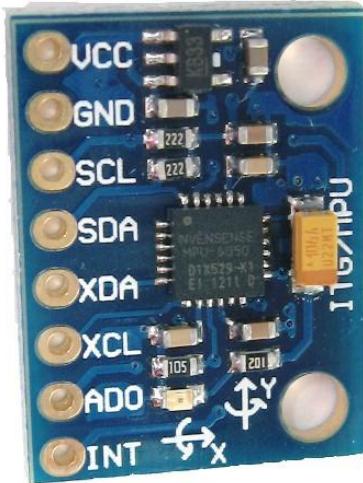
Eine mögliche loop():

```
void loop()
{
    setzeDAC(da_wert);           // DAC Wert setzen
    digitalWrite(13, 1);          // LED ein
    delay(10);
    ad_wert = getADC(0x40);      // ADC Wert von Kanal0 auslesen
    digitalWrite(13, 0);          // LED aus
    Serial.print(da_wert);        // DAC Wert ausgeben
    Serial.print("\t");
    Serial.println(ad_wert);       // ADC Wert ausgeben
    Serial.print("\t");
    da_wert++;
    delay(200);
}
```

Fassen Sie nun alle Programmteile zusammen, bauen Sie die Schaltung auf und testen Sie ihr Programm.

Bearbeiten Sie nun die Aufgaben zum I²C-Bus.

Beschleunigungsmesser und Gyroskop: MPU 6050 auf dem Modul GY-521



11

Ein weiterer sehr interessanter Sensor ist der MPU 6050. Dieser kann die Winkelgeschwindigkeit in x-, y- und z-Richtung messen. Gleichzeitig misst er auch die Beschleunigungswerte in allen drei Achsen. Er eignet sich also ideal als Grundlage für Projekte wie digitale Wasserwagen, Quadrokopter, Segways etc. Eine Temperaturmessung ist ebenfalls enthalten. Der Baustein ist in vielfältigen fertig bestückten und günstigen Modulen erhältlich. Hier wird ein GY-521 verwendet (zum Zeitpunkt für ca. 4€ zu beziehen).

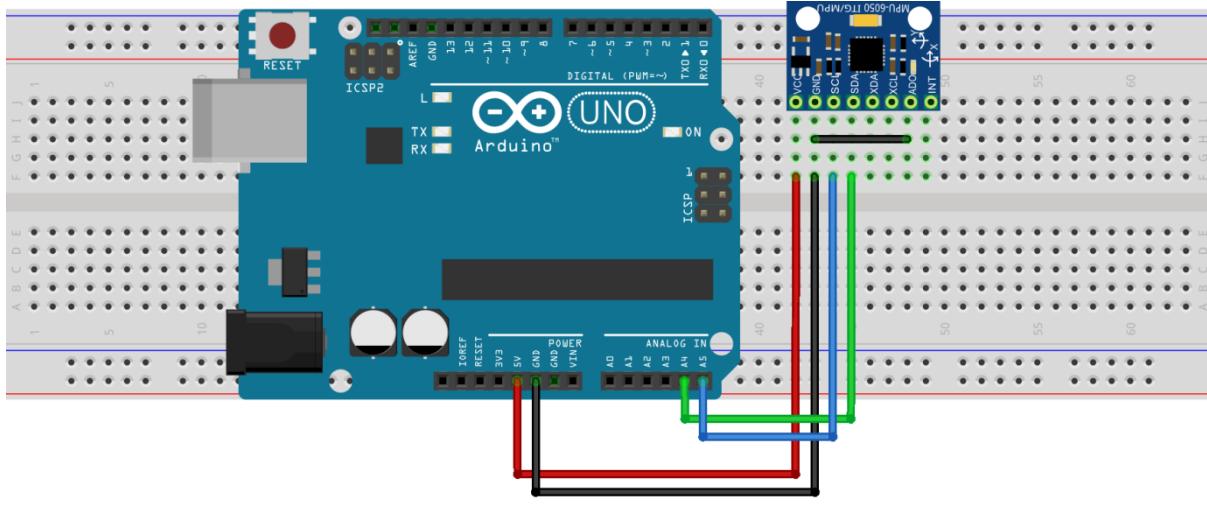
Das Modul bietet Anschlussmöglichkeiten für 5V (VCC), Masse (GND), I²C-Daten (SDA), I²C-Takt (SCL), Interruptausgang (INT) und ein Adresspin (AD0). Es können also maximal zwei dieser Module gleichzeitig verwendet werden. Die Adresse des Moduls ist 0x68 (AD0 auf Masse) oder 0x69 (AD0 auf 5V).

Weiterhin besitzt der MPU 6050 die Möglichkeit als I²C-Master zu dienen. Hierfür sind die Anschlüsse XDA (Daten) und XCL (Takt) vorgesehen.

Im nachfolgenden Beispiel wird die folgende Schaltungsanordnung verwendet.

¹¹ Bild aus: <http://playground.arduino.cc/Main/MPU-6050>

Beschaltung Arduino und MPU6050



fritzing

Das Programm zum Auslesen der Daten:

```
// MPU-6050 Short Example Sketch
// By Arduino User JohnChi
// August 17, 2014
// Public Domain
#include<Wire.h>
const int MPU=0x68; // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B); // PWR_MGMT_1 register
  Wire.write(0); // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  Serial.begin(9600);
}
void loop(){
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU,14,true); // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  Serial.print("AcX = "); Serial.print(AcX);
  Serial.print(" | AcY = "); Serial.print(AcY);
  Serial.print(" | AcZ = "); Serial.print(AcZ);
  Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);
  //equation for temperature in degrees C from datasheet
  Serial.print(" | GyX = "); Serial.print(GyX);
  Serial.print(" | GyY = "); Serial.print(GyY);
  Serial.print(" | GyZ = "); Serial.println(GyZ);
  delay(333);
}
```

Die Werte der einzelnen Register entnimmt man dem Datenblatt des MPU 6050. Wichtig ist, dass der MPU 6050 aufgeweckt wird.

Programmiertchnisch ist die folgende Zeile interessant:

```
AcX=Wire.read()<<8|Wire.read();
```

Zur Erklärung:

Der MPU6050 liefert pro Messwert jeweils zwei Byte, zuerst das höherwertige und dann das niedrige. Diese beiden Bytes müssen dann zu einem Wert zusammengesetzt werden:

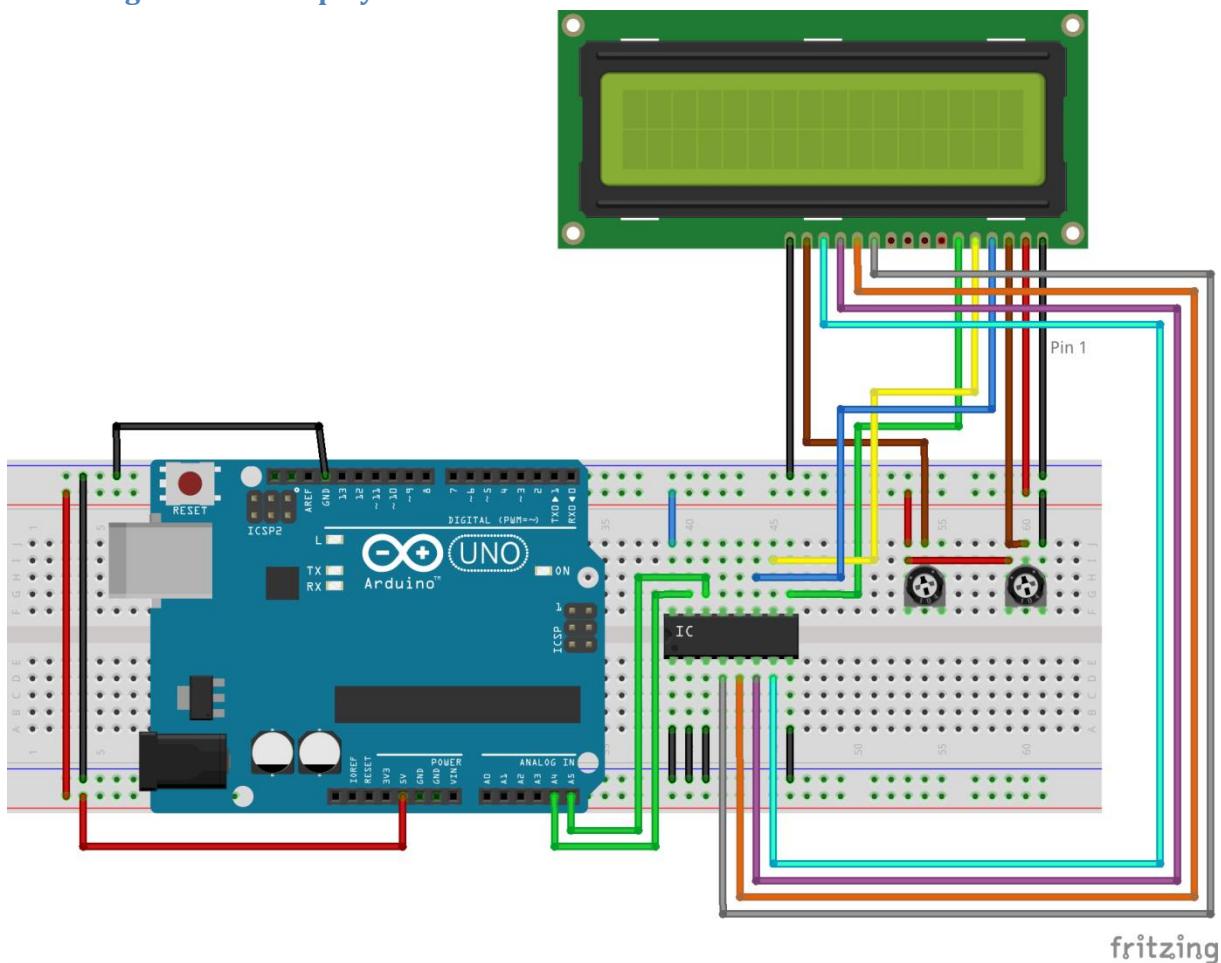
`Wire.read() <<8` liefert das erste Byte und verschiebt es um 8 Bit nach links. Anschließend wird über eine bitweise ODER-Verknüpfung (`|`) das untere Byte dem oberen hinzugefügt. Der endgültige Wert wird dann der Variablen `AcX` zugeordnet.

LCD-Display über I²C

Der Portbaustein PCF 8574AP liefert über den I²C-Bus eine 8-Bit-IO-Port. Er eignet sich damit sehr gut zum Ansteuern von LCD-Displays über den Bus, was IO-Ports auf dem Arduino einspart. Außerdem ist es so möglich, bis zu 8 LCD-Displays gleichzeitig zu verwenden. Aufpassen muss man bei der Adresse des ICs. Je nach Ausführung unterscheidet sich diese, ein Blick in das Datenblatt oder die Verwendung des Scanner-Programms löst dieses Problem. Im nachfolgenden Aufbau und Programm wurde ein PCF 8574AP verwendet, dessen Adresse mit 0x38 zu verwenden ist.

¹² Quellcode aus: <http://playground.arduino.cc/Main/MPU-6050#short>

Schaltung zum LCD-Display über I²C:



Im Programm wird die Bibliothek LiquidCrystal_I2C verwendet. Diese nimmt einem die I²C-Programmierung ab und stellt identische Befehle zur Bibliothek LiquidCrystal zur Verfügung. Der kommentierte Programmcode zum LCD-Display über I²C:

Programm für den Arduino mit einem I²C-LCD-Display

```
//Einbinden der benötigten Bibliotheken
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

//das Display hat 16 Zeichen und zwei Zeilen
//der IC hat die Adresse 0x38
LiquidCrystal_I2C lcd(0x38,16,2);

void setup()
{
    //das Display wird initialisiert
    lcd.init();
    //die Hintergrundbeleuchtung wird eingeschaltet. Dies hat nur bei
    //Displays mit eingebauter Hintergrundbeleuchtung einen Effekt.
    //Im Schaltplan wurde diese per Potentiometer eingestellt
    lcd.backlight();
```

```

}

void loop()
{
    //nach einer halben Sekunde wird der Cursor auf die erste Zeile
    //gesetzt und ein Text ausgegeben
    delay(500);
    lcd.setCursor(0,0);
    lcd.print("Hallo Display");
    //Der Cursor wird auf den Anfang der zweiten Zeile gesetzt
    lcd.setCursor(0,1);
    lcd.print("ueber I2C");
    delay(500);
    //nach einer halben Sekunde wird der Text auf dem Display gelöscht
    lcd.clear();
}

```

Wetterdaten

Luftdruck, Temperatur und Höhe mit dem bmp180 messen



13

Der BMP 180 ist ein I2C-Sensor, entwickelt von BOSCH. Er misst Temperatur aber vor allem den Luftdruck.

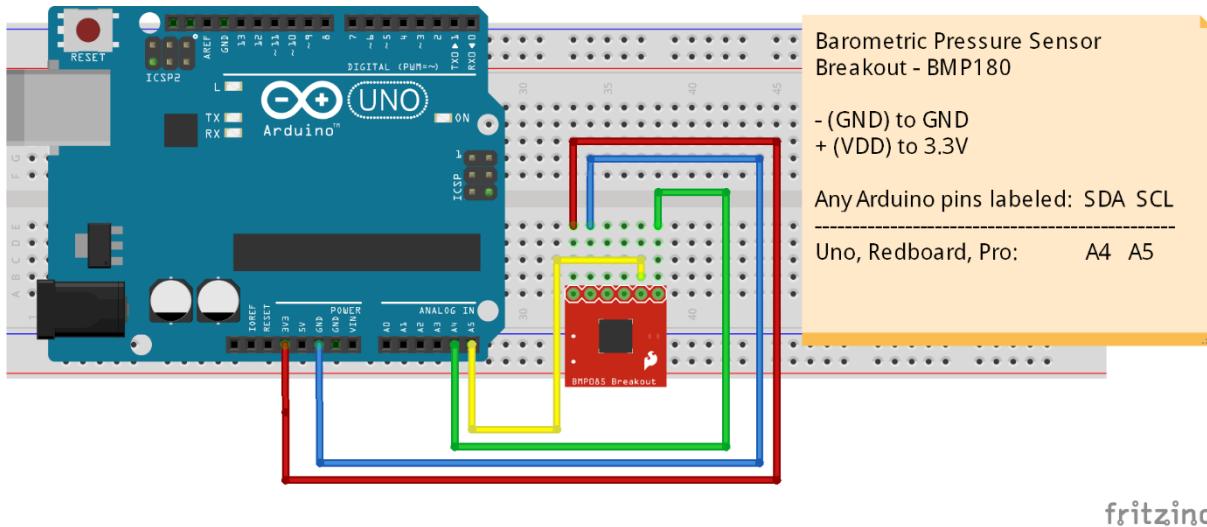
Über den I2C-Bus wird der Sensor angesprochen. Hierzu importiert man die Bibliothek SFW_BMP180. In den Beispielprogrammen der Bibliothek kann man sehen, wie man die Luftdruckdaten ermittelt und wie man aus den Luftdruckdaten die Höhe errechnen kann.

Wichtig:

Der BMP 180 arbeitet mit 3,3V. Daher ist die Versorgungsspannung von 3,3V am Arduino zu wählen. Eine Betriebsspannung von mehr als 4,3 V kann den Sensor zerstören.

Die Beschaltung des Sensors ist also die folgende:

¹³ Grafik aus: http://www.miniinthebox.com/de/bosch-bmp180-temperatur-luftdrucksensoren-modul_p1391704.html



Wichtig:

Die Spannungsversorgung des Bausteins erfolgt über den 3,3V – Anschluss des Arduino. Bitte nicht verwechseln. Der BMP 180 kann ansonsten zerstört werden.

Der Quellcode¹⁴:

```
#include <SFE_BMP180.h>
#include <Wire.h>

SFE_BMP180 pressure;

#define ALTITUDE 200.0

void setup()
{
    Serial.begin(9600);
    Serial.println("REBOOT");

    if (pressure.begin())
        Serial.println("BMP180 init success");
    else
    {
        Serial.println("BMP180 init fail\n\n");
        while(1); // Endlosschleife
    }
}

void loop()
{
    char status;
```

¹⁴ Der Quellcode entspricht weitestgehend der eines Beispielprogramms der verwendeten Bibliothek!

```
double T,P,p0,a;

Serial.println();
Serial.print("provided altitude: ");
Serial.print(ALTITUDE,0);
Serial.print(" meters, ");
Serial.print(ALTITUDE*3.28084,0);
Serial.println(" feet");

status = pressure.startTemperature();
if (status != 0)
{
    status = pressure.getTemperature(T);
    if (status != 0)
    {
        // Print out the measurement:
        Serial.print("temperature: ");
        Serial.print(T,2);
        Serial.print(" deg C, ");
        Serial.print((9.0/5.0)*T+32.0,2);
        Serial.println(" deg F");

        status = pressure.startPressure(3);
        if (status != 0)
        {

            status = pressure.getPressure(P,T);
            if (status != 0)
            {
                // Print out the measurement:
                Serial.print("absolute pressure: ");
                Serial.print(P,2);
                Serial.print(" mb, ");
                Serial.print(P*0.0295333727,2);
                Serial.println(" inHg");

                p0 = pressure.sealevel(P,ALTITUDE); // wir sind in Weilburg ☺
                Serial.print("relative (sea-level) pressure: ");
                Serial.print(p0,2);
                Serial.print(" mb, ");
                Serial.print(p0*0.0295333727,2);
                Serial.println(" inHg");

                a = pressure.altitude(P,p0);
                Serial.print("computed altitude: ");
                Serial.print(a,0);
                Serial.print(" meters, ");
                Serial.print(a*3.28084,0);
                Serial.println(" feet");
            }
            else Serial.println("error retrieving pressure measurement\n");
        }
        else Serial.println("error starting pressure measurement\n");
    }
}
```

```

        else Serial.println("error retrieving temperature measurement\n");
    }
else Serial.println("error starting temperature measurement\n");

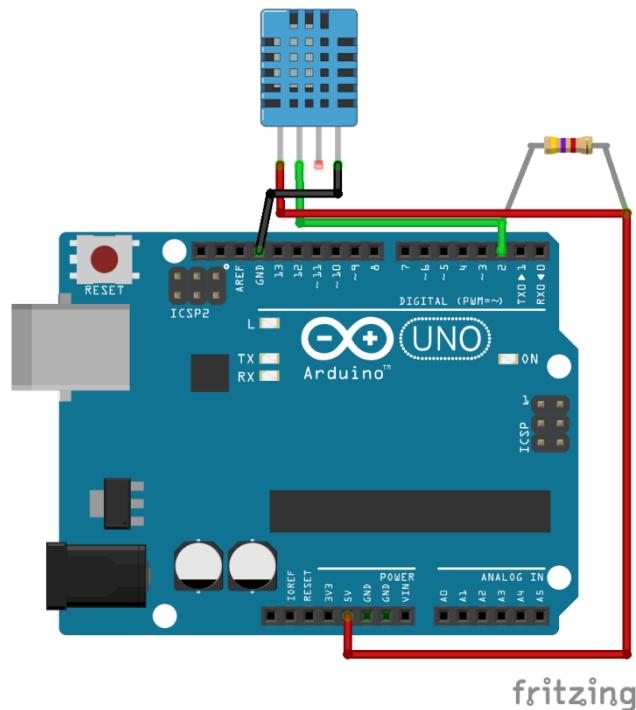
delay(5000); // Pause for 5 seconds.
}

```

Luftfeuchtigkeit mit einem DHT11 messen

Der DHT11 ist ein günstiger, wenn auch etwas ungenauer Sensor zum Ermitteln der Luftfeuchtigkeit.

Die Anschlussbelegung ist wie folgt:



Der Widerstand sollte größer gleich $4,7\text{k}\Omega$ betragen.

```

#include <DHT.h>

#define DHTPIN 2          // Datenpin des DHT 11

#define DHTTYPE DHT11     // DHT 11
// #define DHTTYPE DHT22     // DHT 22  (AM2302)
// #define DHTTYPE DHT21     // DHT 21  (AM2301)

```

```
DHT dht(DHTPIN, DHTTYPE);
```

```
void setup() {
```

```
Serial.begin(9600);
Serial.println("DHTxx test!");

dht.begin();
}

void loop() {
    delay(2000);

    float h = dht.readHumidity();
    float t = dht.readTemperature();
    float f = dht.readTemperature(true);

    if (isnan(h) || isnan(t) || isnan(f)) {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }

    float hi = dht.computeHeatIndex(f, h);

    Serial.print("Humidity: ");
    Serial.print(h);
    Serial.print(" %\t");
    Serial.print("Temperature: ");
    Serial.print(t);
    Serial.print(" *C ");
    Serial.print(f);
    Serial.print(" *F\t");
    Serial.print("Heat index: ");
    Serial.print(hi);
    Serial.println(" *F");
}
```

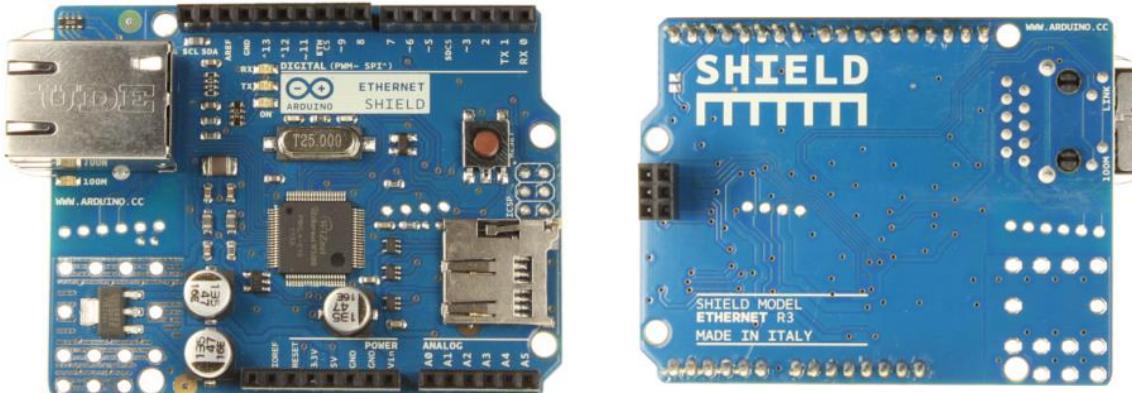
Shields

Fertige Shields sind Platinen mit aufgedruckten Schaltungen, die man "huckepack" auf den Arduino stecken kann. Die einzelnen Anschlussbuchsen des Arduino werden auf dem Shield wieder herausgeführt, um weiter mit den Anschlüsse arbeiten zu können, die das Shield nicht benötigt. Es ist also umgekehrt wichtig zu wissen, dass Shields Anschlüsse des Arduino benötigen. Hier helfen die entsprechenden Dokumentationen weiter.

Netzwerkshield

Das Ethernetshield dient dazu, den Arduino mit einem normalen Ethernet zu verbinden. Somit können Daten ins Netzwerk gesendet werden, oder aber Daten werden vom Netzwerk an den

Arduino geschickt und dort ausgewertet. Im Allgemeinen wird der Arduino als Server konfiguriert. Ein Client kann sich dann bei ihm über das Netzwerk anmelden und die entsprechenden Daten anfordern. Der Arduino kommuniziert mittels SPI mit dem Ethernetshield. Somit stehen die Pins 10, 11, 12 und 13 nicht zur Verfügung.



15

Die Programmierung sieht wie folgt aus:

Da das Ethernetshield über den SPI-Bus angesteuert wird, ist das Einbinden der Bibliotheken SPI.h nötig. Weiterhin wird die Bibliothek Ethernet.h eingebunden und stellt einfache Funktionen und Klassen zur Kommunikation über das Netzwerk zur Verfügung.

```
#include <SPI.h>
#include <Ethernet.h>
```

Anschließend wird die MAC-Adresse (Media Access Control) des Shields in einem Byte-Array angegeben. Die MAC-Adresse ist eine 6 Byte große Zahl, die für Netzwerkkomponenten einmalig sind (oder sein sollten!). Die MAC-Adresse des Shields ist auf diesem aufgedruckt oder aufgeklebt. Bei günstigen Ethernetshields ist oft keine MAC-Adresse aufgedruckt. Hier müssen dann einfach gültige Werte übergeben werden. Allerdings sind dann Funktionalitäten wie Filtern per MAC-Adresse bei Switches oder Routern nicht mehr möglich.

```
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0F, 0x49, 0x45 };
```

Als nächstes folgt in je einem weiteren Byte-Array die IP-Adresse, die Adresse des Gateways sowie die Netzmaske. Hierzu später mehr im Exkurs: Netzwerkadressen.

```
byte ip[] = { 192, 168, 1, 177 };
byte gateway[] = { 192, 168, 1, 255 };
byte subnet[] = { 255, 255, 255, 0 };
```

¹⁵ Quelle: <http://arduino.cc/en/Main/ArduinoEthernetShield>

Als nächstes wird ein Server erstellt. Dieser muss an einen Port gebunden werden, hier 10000. Der Begriff Port wird ebenso im Exkurs Netzwerkadressen erklärt.

```
EthernetServer server = EthernetServer(10000);
```

Im Setup wird die Kommunikation über das Netzwerk gestartet. Ebenso der Server. Dieser beginnt sofort damit, auf Anfragen von einem Client zu warten. Zu Debugzwecken wurde hier noch die serielle Schnittstelle initialisiert.

```
void setup()
{
    Ethernet.begin(mac, ip, gateway, subnet);
    Serial.begin(9600);
    server.begin();
}
```

Fragt ein Client am Server an, so liefert die Methode

```
EthernetClient client = server.available();
```

einen Wert für das Objekt client aus der Klasse EthernetClient zurück.

Die Methode

```
client.available();
```

gibt nun zurück, wie viele Bytes anliegen. Liegt nichts an, so liefert die Methode den Wert -1 zurück. In der Bedingung einer if-Entscheidung kann somit überprüft werden, ob Bytes vorliegen oder nicht.

Lieferte die Abfrage den Wert true, so kann über

```
char c = client.read();
ein Byte bzw. ein Character vom client gelesen werden.
```

Über den Befehl:

```
client.print(c);
```

wird der eingelesene Character wieder in das Ethernet zurückgeschrieben. Diese Variante gibt es auch als `println()` mit einem automatischen Zeilenumbruch.

```
client.stop();
```

stoppt die Kommunikation.

Quelltext eines Arduinos als Echo-Server auf Port 10000

```
#include <SPI.h>
#include <Ethernet.h>

// MAC-Adresse
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0F, 0x49, 0x45 };
// IP-Adresse
byte ip[] = { 10, 1, 0, 177 };
// Gateway
byte gateway[] = { 10, 1, 255, 254 };
// Subnetmaske
```

```
byte subnet[] = { 255, 255, 0, 0 };

// Erzeugen des Servers, Bindung an Port 10000
EthernetServer server = EthernetServer(10000);

void setup()
{
    // Starten des Servers im Ethernet
    Ethernet.begin(mac, ip, gateway, subnet);
    Serial.begin(9600);
    // Starten der Funktionalität des Servers
    server.begin();
}

void loop()
{
    //Wenn ein Client anklopft:
    EthernetClient client = server.available();
    if (client.available()) {
        // Bytes werden vom Client gelesen und wieder an ihn geschrieben
        char c = client.read();
        Serial.print(c);
        client.print(c);
        if(c=='q'){
            client.stop();
        }
    }
}
```

Weitere Ethernet-Befehle der Klasse EthernetClient:

```
char server[] = {"192.168.1.1"};
client.connect(server, 10000);
```

Erklärung: Der Arduino ist nun nicht Server sondern Client. Er kann sich somit beispielsweise bei einem Webserver anmelden und diesem seine Daten schicken. Übergeben wird die IP-Adresse oder eine URL sowie der Port.

```
boolean verbindung = client.connected();
```

Erklärung: Der Befehl überprüft, ob eine Ethernetverbindung vorhanden ist. Ist dies der Fall, wird true, ansonsten false zurückgegeben.

Themen

Bewegungsmelder PIR

Der Bewegungsmelder PIR ist denkbar einfach anzuschließen und zu programmieren. Der Sensor wird über eine Betriebsspannung von 5V versorgt. Am Pin OUT liegt ein HIGH-Signal an, wenn eine Bewegung wahrgenommen wird.



Der Bewegungsmelder von oben



und von unten

Ein Beispielprogramm, welches die LED an Pin 13 im Falle einer wahrgenommenen Bewegung leuchten lässt. Der Pin OUT ist am Pin 2 des Arduinos angeschlossen:

```
int led=13;
int bewegung=2;
int bewegungsstatus=0;

void setup()
{
    pinMode(led, OUTPUT);
    pinMode(bewegung, INPUT);
}

void loop()
{
    bewegungsstatus=digitalRead(bewegung);
    if (bewegungsstatus == HIGH)
    {
        digitalWrite(led, HIGH);
        delay(5000);
        digitalWrite(led, LOW);
    }
    else
    {
        digitalWrite(led, LOW);
    }
}
```

RFID (Technologie vereinfacht dargestellt!)

RFID (Radio Frequency Identification) ist eine Technologie, mittels derer man kleine Transponder per Funk auslesen aber auch beschreiben kann. Hierfür ist im Transmitter keine Spannungsquelle erforderlich. Die Spannungsversorgung funktioniert über die vom Empfangsgerät ausgesendeten Funkwellen. Diese werden im Transponder durch eine Antenne (meist nur wenige Wicklungen, die man sogar aufdrucken kann) aufgenommen und in Kapazitäten zwischengespeichert. Die gespeicherte Spannung aktiviert einen Chip, der seine Daten über die Antenne zurücksendet.

Zurzeit gibt es RFID-Empfänger und Chips in den unterschiedlichsten Varianten. Eine beim Arduino häufig eingesetzte Variante ist der RFID-DC522:

Der Baustein ist im Moment bei den einschlägigen Onlinehändlern für ca. 3€ zu erwerben. Meist sind auch noch zwei Transponder im Paket enthalten.

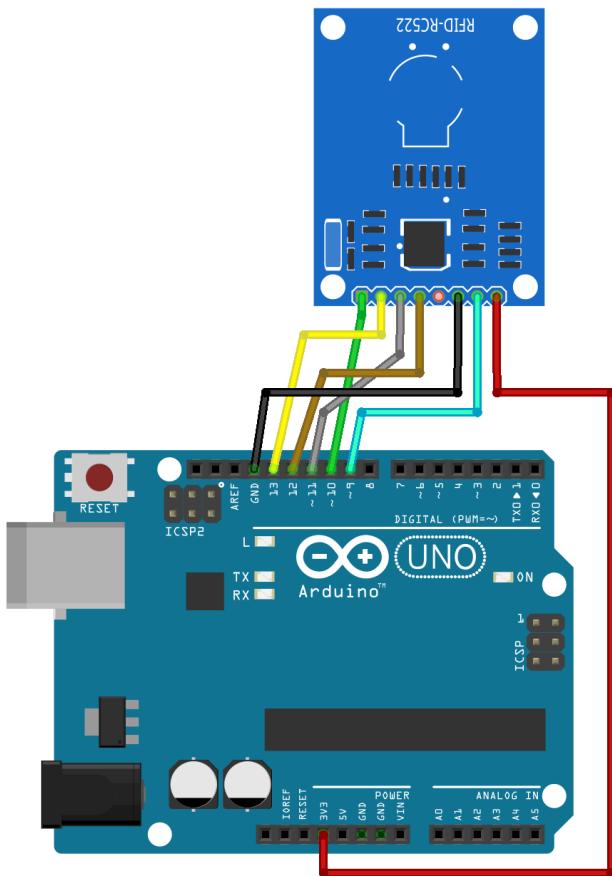


Die Kommunikation zu dem Lesegerät wird mittels SPI realisiert. Hierfür sind die folgenden Anschlüsse notwendig:

Arduino UNO	RFID-RC522
Pin 9 (Reset)	RST
Pin 10 (Chip Select)	SDA
Pin 11 (Master out Slave in)	MOSI
Pin 12 (Master in Slave out)	MISO
Pin 13 (SCK Clock)	SCK

Zusätzlich sind noch eine Masseverbindung und die Betriebsspannung nötig. Beachten Sie bitte, dass die Betriebsspannung 3,3V beträgt!!

Der Anschlussplan:



Die passende Bibliothek findet man beispielsweise bei GitHub im Internet. Die Bibliothek erfordert die schon installierte Bibliothek SPI.

Im nachfolgenden Quelltext wird einfacher RFID Transponder ausgelesen:

```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10 //Der ChipSelect-Pin wird gesetzt
#define RST_PIN 9 //Der Reset-Pin wird gesetzt
MFRC522 mfrc522(SS_PIN, RST_PIN); // Ein Objekt aus der Klasse MFRC522 wird
//erstellt

void setup() {
    Serial.begin(9600); // Serielle Schnittstelle initialisieren
    SPI.begin(); // SPI-Bus initialisieren
    mfrc522.PCD_Init(); // RFID-Leser initialisieren
    Serial.println("Karte wird gescannt...");
}

void loop() {
    // Es wird auf neue Transponder gewartet
    if ( ! mfrc522.PICC_IsNewCardPresent() ) {
        return;
    }

    // Wenn eine neuer Transponder entdeckt wurde wird er angesprochen und seine
    // Kommunikationsdaten auf die serielle Schnittstelle geschrieben
}
```

```
if ( ! mfrc522.PICC_ReadCardSerial() ) {
    return;
}

// Die gespeicherten Informationen der Karte werden auf die serielle
// Schnittstelle geschrieben
mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
```

Jeder Transponder besitzt eine eigene Unique Identification Number (uid). Diese kann man wie folgt auslesen:

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9

MFRC522 mfrc522(SS_PIN, RST_PIN);

void setup()
{
    Serial.begin(9600);
    SPI.begin();
    mfrc522.PCD_Init();
}

void loop()
{
    if ( ! mfrc522.PICC_IsNewCardPresent() )
    {
        return;
    }
    if ( ! mfrc522.PICC_ReadCardSerial() )
    {
        return;
    }

    Serial.print("Die ID des RFID-TAGS lautet:");

    for (byte i = 0; i < mfrc522.uid.size; i++) //Die uid steht in einem Array. Die
                                                //Anzahl der Elemente im Array wird über mfrc522.uid.size ermittelt
    {
        Serial.print(mfrc522.uid.uidByte[i], HEX); //Die einzelnen Elemente werden
                                                //hexadezimal auf die serielle Schnittstelle geschrieben
        Serial.print(" ");
    }
    Serial.println();
}
```

Beispiele zum Schreiben auf den Transponder finden sich in der Bibliothek. Hierzu sind allerdings beschreibbare Karten nötig.

Arduino im WLAN

Aktuell stehen mehrere Möglichkeiten zur Verfügung, den Arduino in ein WLAN zu integrieren. Die beiden gängigsten sind die Module basierend auf dem CC3000 und dem ESP8266:

CC3000



Pinbelegung:

CC3000	Arduino
IRQ	Pin 2
UB_EN	Pin 7
CS	Pin 10
MOSI	Pin 11
MISO	Pin 12
CLK	Pin 13
3V3	3,3 V
GND	Gnd

```
*****
WebClient.ino
CC3000 WebClient Test
Shawn Hymel @ SparkFun Electronics
March 1, 2014
https://github.com/sparkfun/SFE\_CC3000\_Library
```

Manually connects to a WiFi network and performs an HTTP GET request on a web page. Prints the contents of the page to

the serial console.

The security mode is defined by one of the following:
`WLAN_SEC_UNSEC`, `WLAN_SEC_WEP`, `WLAN_SEC_WPA`, `WLAN_SEC_WPA2`

Hardware Connections:

Uno Pin	CC3000 Board	Function
+5V	VCC or +5V	5V
GND	GND	GND
2	INT	Interrupt
7	EN	WiFi Enable
10	CS	SPI Chip Select
11	MOSI	SPI MOSI
12	MISO	SPI MISO
13	SCK	SPI Clock

Resources:

Include `SPI.h`, `SFE_CC3000.h`, and `SFE_CC3000_Client.h`

Development environment specifics:

Written in Arduino 1.0.5

Tested with Arduino UNO R3

This code is beerware; if you see me (or any other SparkFun employee) at the local, and you've found our code helpful, please buy us a round!

Distributed as-is; no warranty is given.

```
#include <SPI.h>
#include <SFE_CC3000.h>
#include <SFE_CC3000_Client.h>

// Pins
#define CC3000_INT      2 // Needs to be an interrupt pin (D2/D3)
#define CC3000_EN       7 // Can be any digital pin
#define CC3000_CS      10 // Preferred is pin 10 on Uno

// Connection info data lengths
#define IP_ADDR_LEN     4 // Length of IP address in bytes

// Constants
char ap_ssid[] = "FRITZ!Box 7490"; // SSID of network
char ap_password[] = "03426105451546069388"; // Password of network
unsigned int ap_security = WLAN_SEC_WPA2; // Security of network
unsigned int timeout = 30000; // Milliseconds
char server[] = "www.example.com"; // Remote host site

// Global Variables
SFE_CC3000 wifi = SFE_CC3000(CC3000_INT, CC3000_EN, CC3000_CS);
SFE_CC3000_Client client = SFE_CC3000_Client(wifi);

void setup() {

    ConnectionInfo connection_info;
    int i;
```

```
// Initialize Serial port
Serial.begin(115200);
Serial.println();
Serial.println("-----");
Serial.println("SparkFun CC3000 - WebClient");
Serial.println("-----");

// Initialize CC3000 (configure SPI communications)
if ( wifi.init() ) {
    Serial.println("CC3000 initialization complete");
} else {
    Serial.println("Something went wrong during CC3000 init!");
}

// Connect using DHCP
Serial.print("Connecting to SSID: ");
Serial.println(ap_ssid);
if(!wifi.connect(ap_ssid, ap_security, ap_password, timeout)) {
    Serial.println("Error: Could not connect to AP");
}

// Gather connection details and print IP address
if ( !wifi.getConnectionInfo(connection_info) ) {
    Serial.println("Error: Could not obtain connection details");
} else {
    Serial.print("IP Address: ");
    for (i = 0; i < IP_ADDR_LEN; i++) {
        Serial.print(connection_info.ip_address[i]);
        if ( i < IP_ADDR_LEN - 1 ) {
            Serial.print(".");
        }
    }
    Serial.println();
}

// Make a TCP connection to remote host
Serial.print("Performing HTTP GET of: ");
Serial.println(server);
if ( !client.connect(server, 80) ) {
    Serial.println("Error: Could not make a TCP connection");
}

// Make a HTTP GET request
client.println("GET /index.html HTTP/1.1");
client.print("Host: ");
client.println(server);
client.println("Connection: close");
client.println();
Serial.println();

void loop() {

    // If there are incoming bytes, print them
    if ( client.available() ) {
        char c = client.read();
        Serial.print(c);
    }
}
```

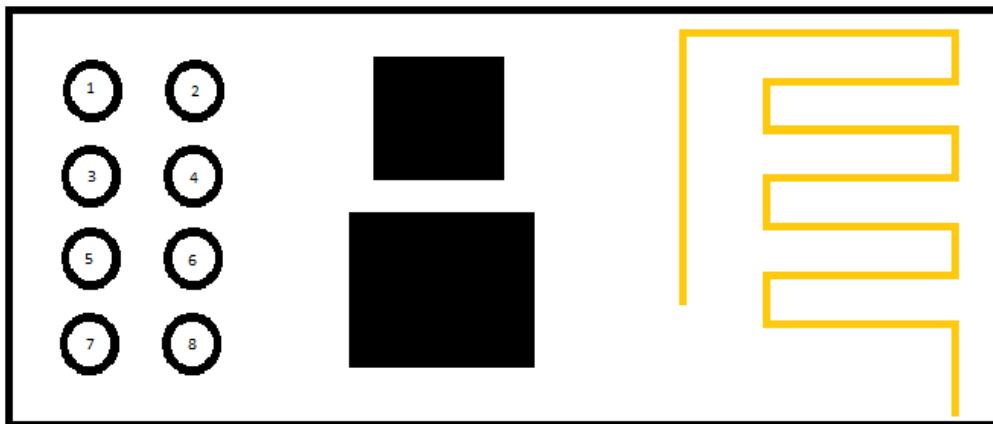
```
// If the server has disconnected, stop the client and wifi
if ( !client.connected() ) {
    Serial.println();

    // Close socket
    if ( !client.close() ) {
        Serial.println("Error: Could not close socket");
    }

    // Disconnect WiFi
    if ( !wifi.disconnect() ) {
        Serial.println("Error: Could not disconnect from network");
    }

    // Do nothing
    Serial.println("Finished WebClient test");
    while(true){
        delay(1000);
    }
}
}
```

ESP8266



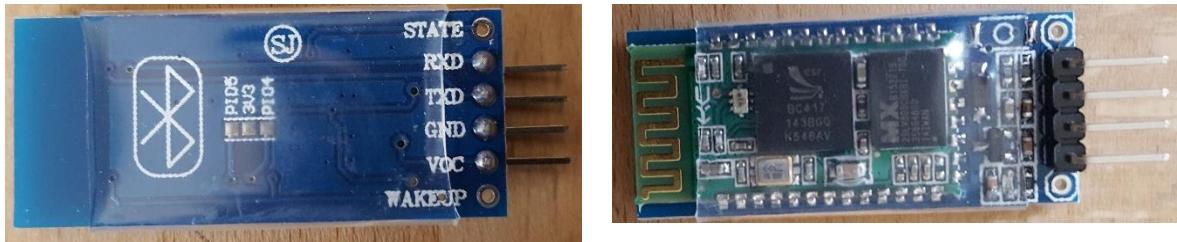
Pinbelegung:

Pin	Bedeutung
1	TXD: Serieller Sender
2	GND: Masse
3	CHPD: Chip ausschalten (LOW-aktiv, muss auf 3,3V gelegt werden)
4	GPIO 2: General Purpose Input Output 2
5	RST: Softwarereset
6	GPIO 0: General Purpose Input Output 0
7	VCC: Betriebsspannung 3,3 V mindestens 200mA
8	RXD: Serieller Empfänger

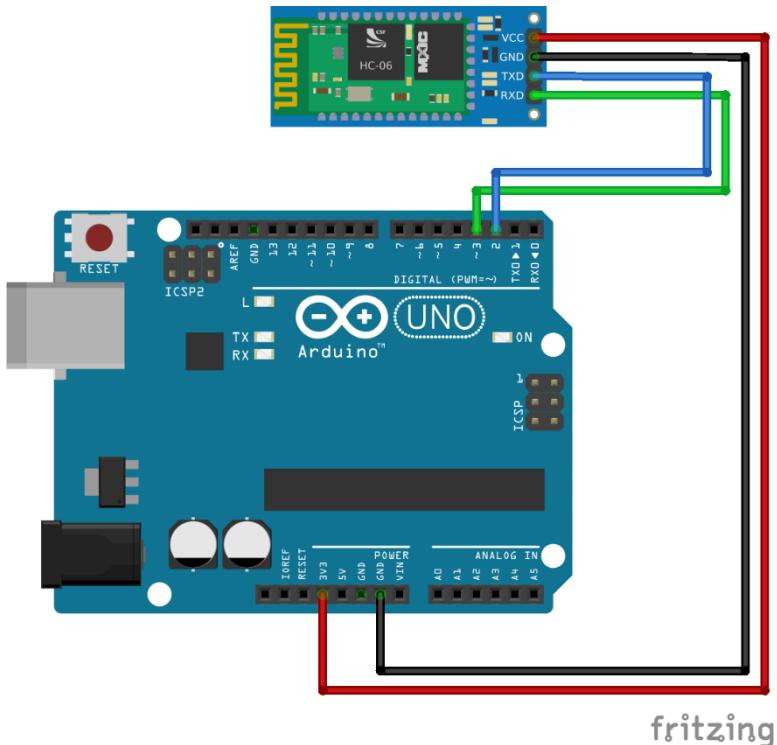
Arduino und Bluetooth

Das Modul HC-06 ist aktuell unter 3€ im Internet zu beziehen. Das Modul stellt ein eigenständiges Gerät dar. Dies bemerkt man dann, wenn man Betriebsspannung anlegt. Das Modul ist dann von anderen Bluetooth-Geräten zu sehen und eine Verbindung ist möglich. Die Kommunikation zwischen HC-06 und dem Arduino funktioniert über eine serielle Schnittstelle. Die Verwendung der Bibliothek SoftwareSerial ermöglicht es, die Schnittstelle vom PC zum Arduino weiter zu verwenden.

Das Modul HC-06:



Anschlussbelegung (Achtung: Das Modul arbeitet mit 3,3V!!):



fritzing

Der Quelltext des Programms zum einfachen Senden und Empfangen von Textnachrichten:

```
#define PIN_RECEIVE 2
#define PIN_SEND 3
#define PIN_STATUS 13
#define SPEED_BLUETOOTH 9600
#define SPEED_SERIAL 9600

#include <SoftwareSerial.h>

// Variablen
SoftwareSerial blueSerial(PIN_RECEIVE, PIN_SEND);
int zustand_status = HIGH;

void setup()
{
    Serial.begin(SPEED_SERIAL);
    Serial.println("Bluetoot Serial initialisiert");
    pinMode(PIN_STATUS, OUTPUT);

    // auf seriellen Port horchen
    blueSerial.begin(SPEED_BLUETOOTH);
    blueSerial.println("Bluetoot Serial Verbindung hergestellt");
}

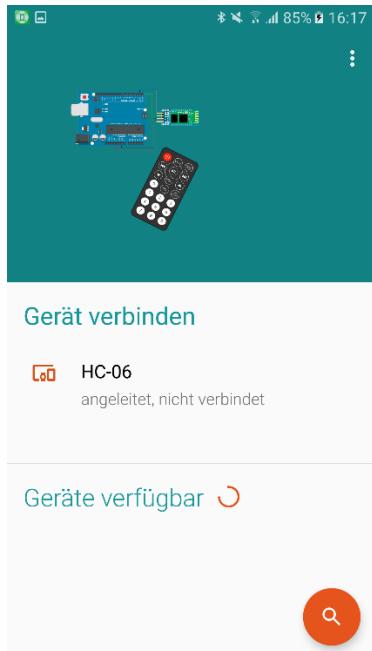
void loop()
{
    // Sendet alles, was eingegeben wird.
    if (Serial.available())
        blueSerial.write(Serial.read());

    // Gibt alles aus, was empfangen wird:
}
```

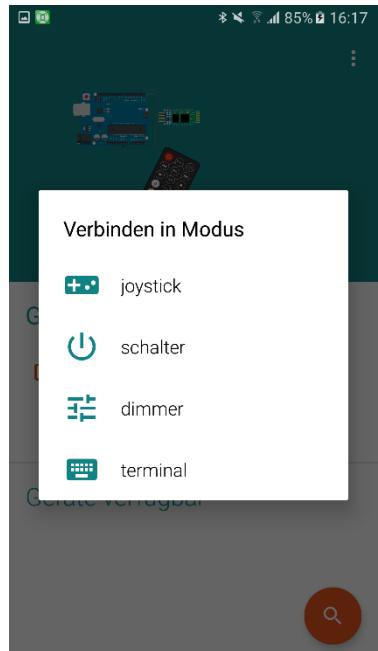
```
if (blueSerial.available()) {
    Serial.write(blueSerial.read());
    if(zustand_status == HIGH)
        zustand_status=LOW;
    else zustand_status=HIGH;
}
digitalWrite(PIN_STATUS,zustand_status);
}
```

Nun benötigt man nur noch eine App, um mit dem Modul zu kommunizieren. Ich habe die kostenfreie App: Arduino bluetooth controller verwendet.

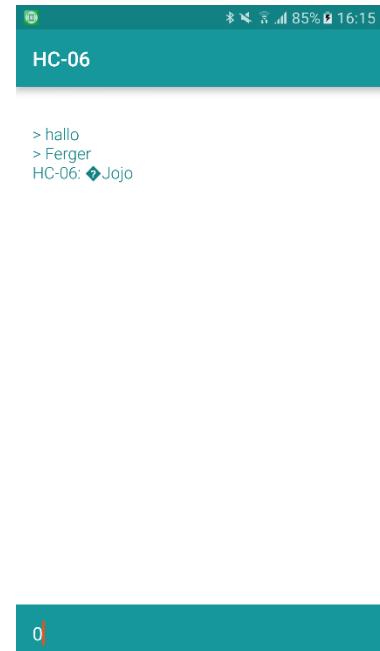
Nach der Installation kann die App gestartet und eine Verbindung zum Modul aufgebaut werden:



Nach der Verbindung (gelegentlich ist die Eingabe eines Codes erforderlich. Hier: 1234) kann man in der App auswählen, wie sie verwendet werden soll:



Die Auswahl „terminal“ erfolgt. Nun kann man mit dem Modul kommunizieren:



Umbenennen des Bluetoothmoduls

Im Unterricht sind mehrere Bluetoothmodule präsent. Daher ist es ratsam, die Module umzubenennen. Hierfür eignet sich ein USB-Serial-Adapter.

Schließen Sie das Modul über das entsprechende Kabel an den PC an. Eventuell ist eine Treiberinstallation nötig. Der PC richtet einen COM-Port ein.

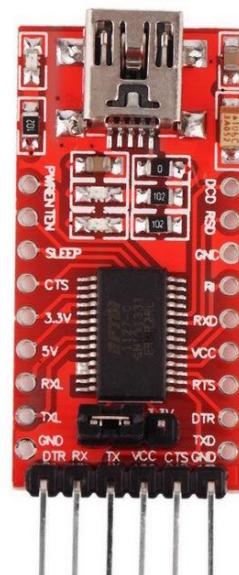
Das Bluetoothmodul ist wie folgt anzuschließen:

VCC → 3,3 V (Arduino verwenden!)

GND → GND

TX → RX

RX → TX



Starten Sie nun die Arduino Entwicklungsumgebung, stellen den seriellen Port des FTDI ein und starten den seriellen Monitor.

Stellen Sie die Baudrate auf 9600Baud ein (sofern nicht anders angegeben). Die Kontrollzeichen werden ausgeschaltet (kein Zeilenende).

Nun können Sie AT-Befehle absetzen:

AT+NAMExyz ändert den Namen des Moduls auf xyz.

AT+PIN5678 ändert den Pin des Moduls auf 5678.

Andere AT-Befehle kann man in der Dokumentation des Bluetoothmoduls nachlesen.

Ein Gerät bauen

Plant man ein eigenes Gerät zu bauen, so wird man kaum einen kompletten Arduino integrieren. Dies ist teuer und sperrig. Im Prinzip reicht es für das spätere Gerät, dass der Mikroprozessor seine Arbeit vollrichtet. Nachfolgend wird beschrieben, wie man dies mit und ohne externen Quarz realisiert.

Bootloader

Das Prinzip des Arduinos ist das folgende:

Auf dem Prozessor befindet sich der Bootloader, quasi ein kleines Betriebssystem.

Bekommt der Prozessor seine Betriebsspannung, bzw. hat er schon diese Spannung und der Reset wird ausgeführt, so wartet das Betriebssystem auf eine Reaktion seitens der seriellen Schnittstelle. Kommen von dort keine Signale, so springt das Betriebssystem an eine definierte Speicheradresse und führt den dortigen Proramcode aus.

Bekommt die serielle Schnittstelle Signale, so liest das Betriebssystem die entsprechenden Daten aus und schreibt sie an die definierte Speicheradresse. Anschließend wird dieses Programm dann ausgeführt.

Es sind also die folgenden Schritte nötig, um einen Prozessor „Arduino-fähig“ zu machen:

1. Treffen der Entscheidung, ob der Prozessor mit internem Takt oder mit externem Quarz betrieben werden soll.
2. Installation des entsprechenden Bootloaders auf den Mikroprozessor.
3. Aufspielen des eigentlichen Programms mittels einer seriellen Schnittstelle, hier ein USB-Seriell-Wandler.

1. Interner oder externen Takt

Der Mikroprozessor, der ATMEGA 328 P besitzt die Möglichkeit, einen internen Takt von 8MHz zu verwenden. Die Stabilität und Genauigkeit dieses Taktes ist allerdings nicht ausreichend, um hochpräzise zeitliche Abläufe zu realisieren. Für einfache Ablaufsteuerungen reicht es aber sicher aus.

Um den internen Takt zu aktivieren, müssen beim Beschreiben des Bootloaders sogenannte Fuses gesetzt werden. Dies sind einzelne Bits in den Registern des ATMEGA.

Um nicht in alte hexadezimale Programmierung zurück zu fallen, gibt es unter <https://www.arduino.cc/en/uploads/Tutorial/breadboard-1-6-x.zip> einen fertigen Bootloader für dieses Vorhaben.

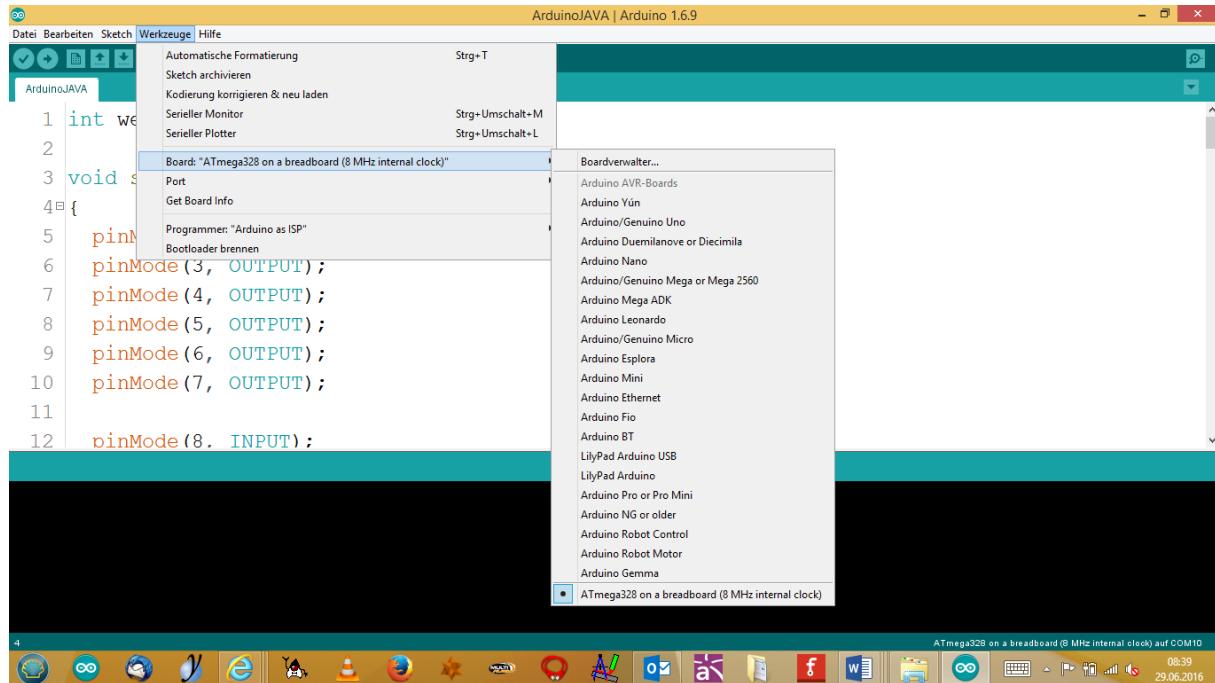
Laden Sie diese Datei herunter und entpacken das Archiv. Es entsteht ein Verzeichnis `breadboard`.

Erstellen Sie in Ihrem Arduino-Dokumentenordner ein neues Verzeichnis mit dem Namen: `hardware`.

Kopieren Sie das komplette Verzeichnis: `breadboard` in dieses Verzeichnis.

Starten Sie nun die Entwicklungsumgebung des Arduino.

Unter dem Menüpunkt Werkzeuge -> Boards sollte nun ein neuer Eintrag entstanden sein:

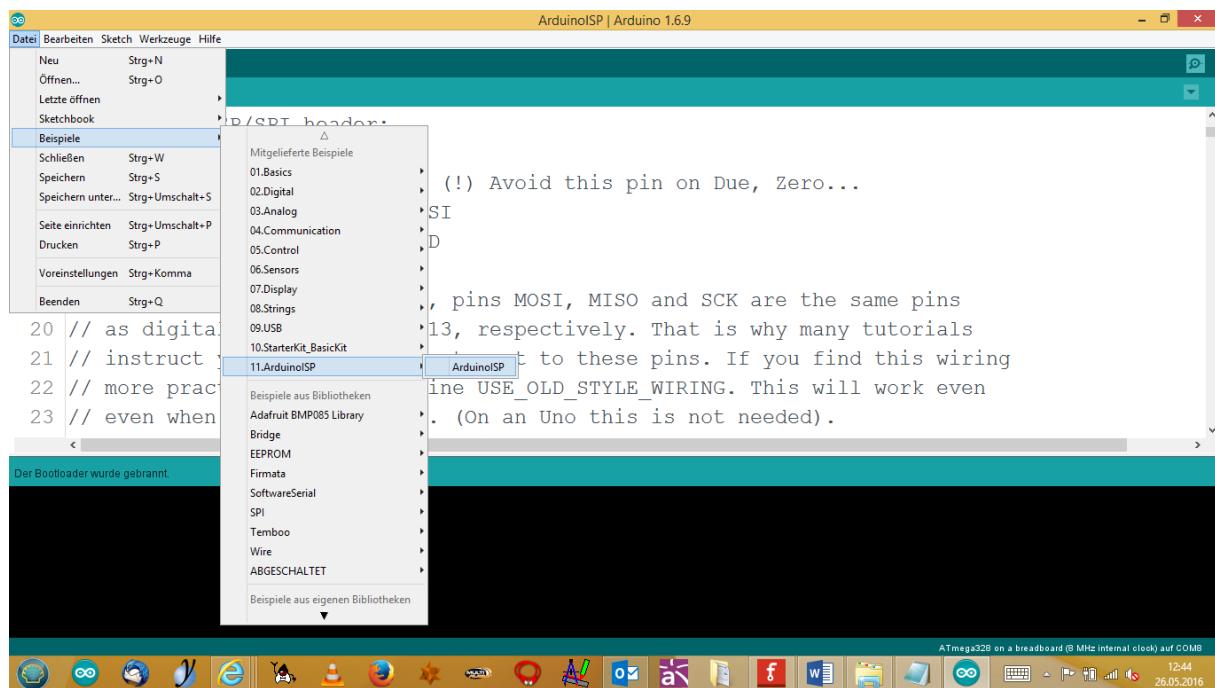


Nun muss dieser Bootloader auf den Prozessor aufgespielt werden. Hierzu können Sie einen ISP-Programmer verwenden. Es geht aber auch mittels eines Arduinosystems selbst:

Arduino als Programmer

Schließen Sie Ihren Arduino an den Rechner an.

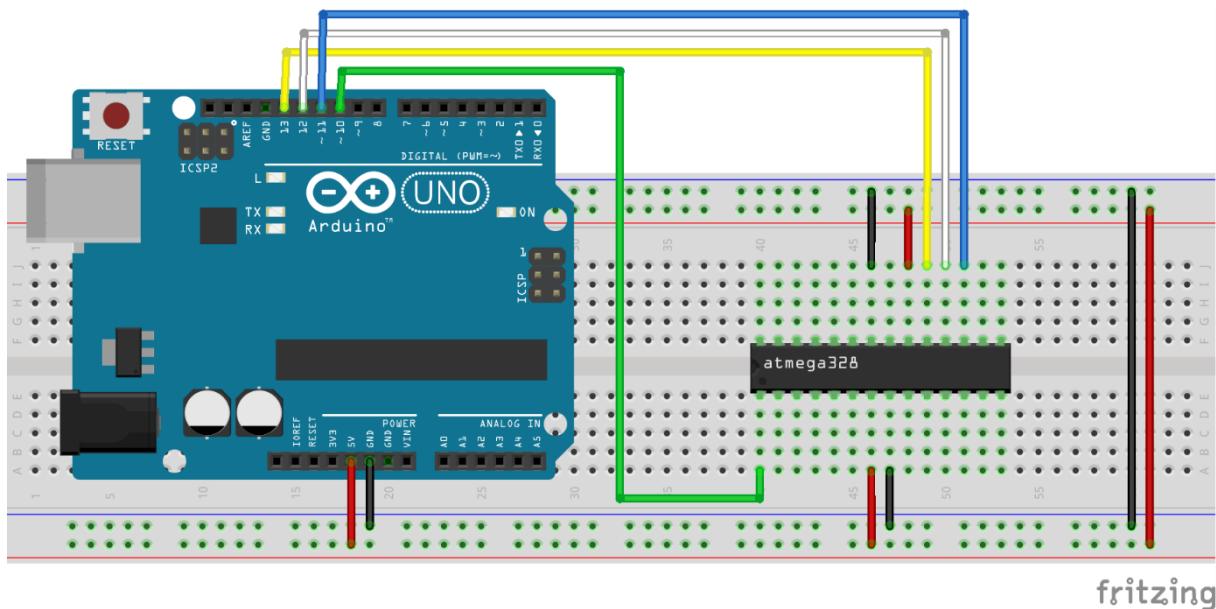
Starten Sie die Entwicklungsumgebung und wählen Sie den entsprechenden seriellen Port aus. Im Menüpunkt Beispiele finden Sie den Eintrag ArduinoISP und darunter einen Sketch.



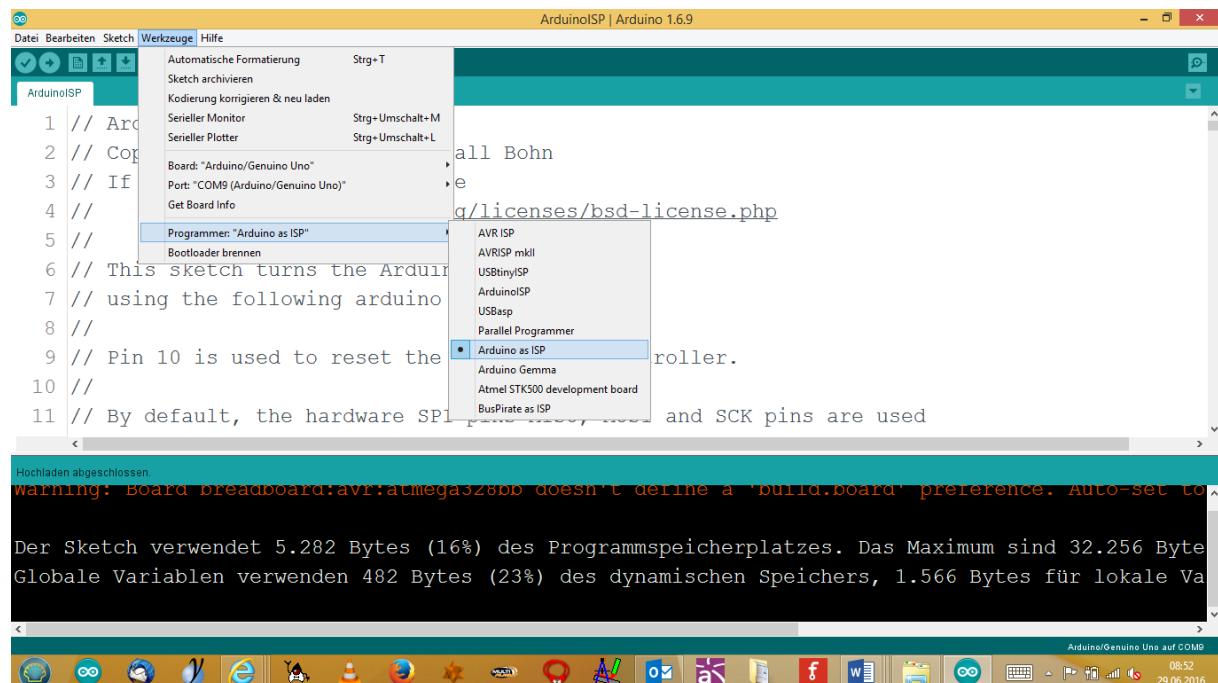
Öffnen Sie diesen und laden Sie ihn auf Ihren Arduino hoch. Der Arduino ist nun ein ISP-Programmer.

2. Installation des Bootloaders

Die Verdrahtung ist nachfolgend dargestellt. Hat man das Kapitel SPI im Aufbaukurs schon bearbeitet, so erkennt man, dass hier genau dieser Bus verwendet wird.



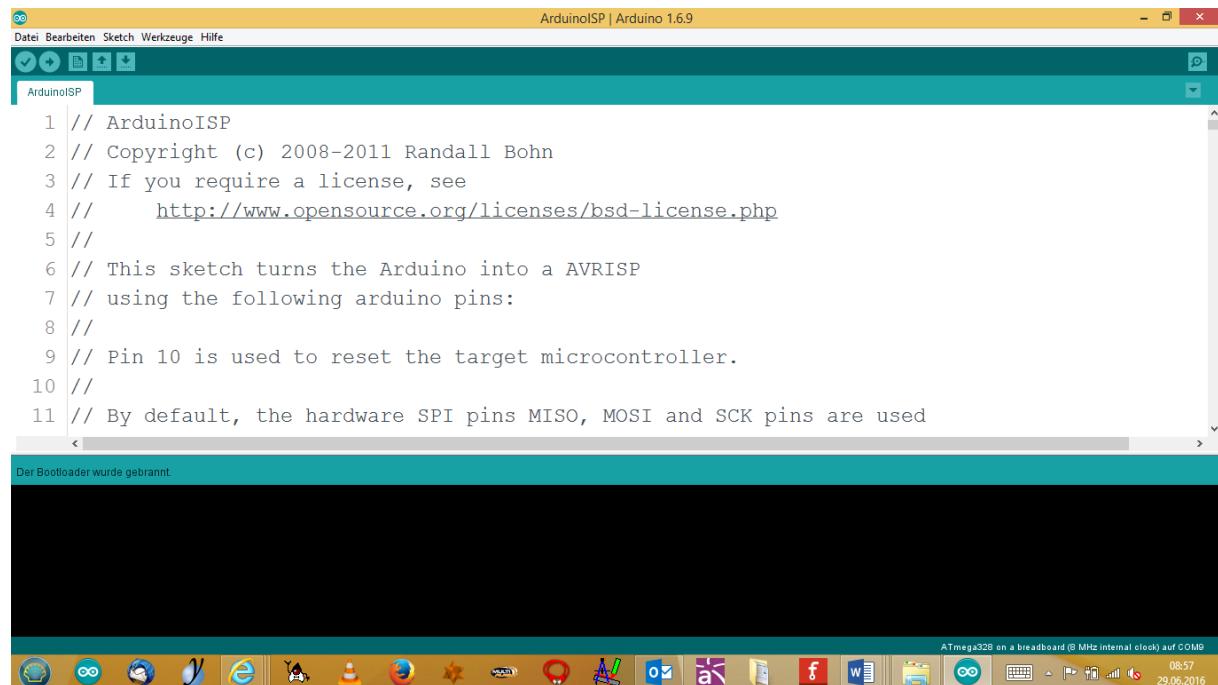
Nun wählt man im Menüpunkt Werkzeuge -> Programmer den „Arduino as ISP“ aus.



Im Menüpunkt Werkzeuge -> Board wird "AtMega 328 on an breadboard (8 MHz internal Clock)" ausgewählt.

Nun wird der Bootloader gebrannt. Klicken Sie auf Werkzeuge -> Bootloader brennen.

Hat alles funktioniert, so bekommen Sie in der Statuszeile die Meldung: „Der Bootloader wurde gebrannt“.



Änderung der Datei avrdude.conf

Hinweis:

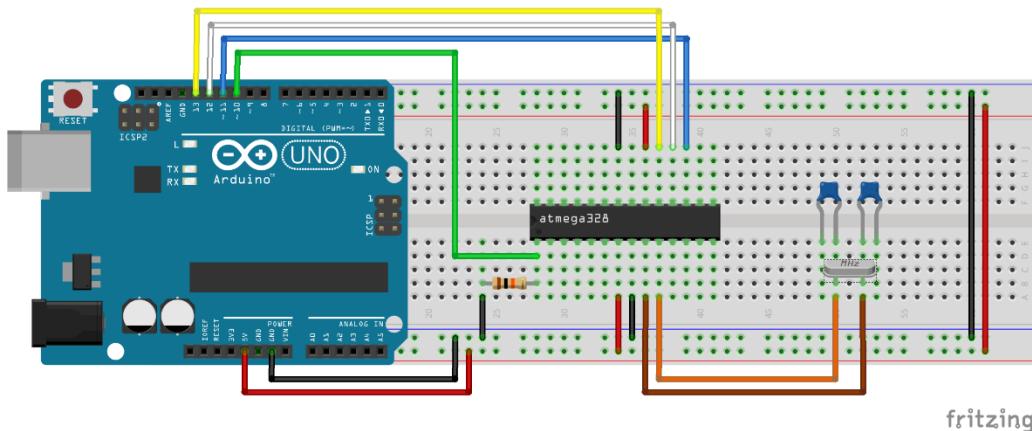
Es gibt unterschiedliche AtMega 328 Versionen (mit P und ohne P). Das P deutet darauf hin, dass der Mikroprozessor einige Powerfunktionen beinhaltet. Die genaue Version des Prozessors muss in einer Konfigurationsdatei angegeben sein. Diese Datei: avrdude.conf befindet sich im Ordner: C:\Program Files (x86)\Arduino\hardware\tools\avr\etc (oder aber in einem anderen Ordner je nach Installation).

Der korrekte Eintrag für einen AtMega 328 P lautet:

```
part parent "m328"
    id          = "m328p";
    desc        = "ATmega328P";
    signature   = 0x1e 0x95 0x0F;
    ocdrev     = 1;
;
```

AtMega 328 P mit externem Takt

Die Änderungen sind nur marginal. Es werden 2 Keramikkondensatoren mit 22pF und ein 16MHz Quarz benötigt. Die Verdrahtung ist wie folgend abgebildet:



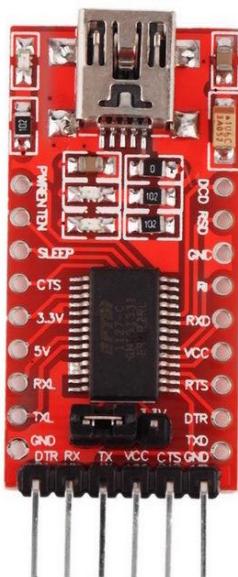
Beim Schreiben des Bootloaders ist im Menüpunkt Werkzeuge -> Board -> Arduino Uno zu wählen.

Der restliche Vorgang ist identisch.

3. Programmierung des Atmega mittels FTDI:

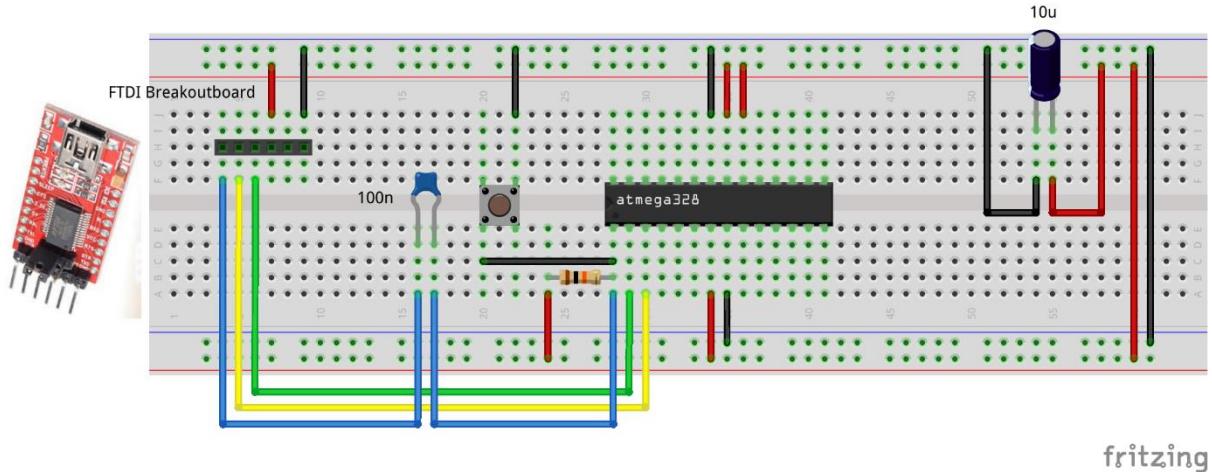
Da das Hochladen eines Sketches über eine serielle Schnittstelle erfolgt, ist es nötig, eine solche zu besitzen. Auf dem Arduino Board ist ein USB-zu-Seriell-Wandler integriert. Um aber ein eigenes Gerät zu entwickeln, benötigt man den Wandler nur für das Aufspielen des Sketches. Entsprechende Module bekommt man unter dem Stichwort FTDI bei den einschlägigen Händlern angeboten. Die Preise beginnen bei ca. 2,50 €.

Die Module sind alle ähnlich aufgebaut:



Meist wird das Modul über ein USB-Kabel mit Micro-Stecker an den PC angeschlossen. Die für die Programmierung wichtigsten Pins sind als Steckkontakte rausgeführt. Somit kann das Modul direkt auf dem Breadboard verwendet werden. Von Vorteil ist auch, dass die 5V Spannungsversorgung des Moduls beim Aufspielen eines Sketches verwendet werden kann. Es ist also keine externe Spannungsquelle nötig. Über einen Jumper kann auch eine 3,3V Spannungsversorgung ausgewählt werden, dies ist hier aber nicht nötig.

Der Aufbau ist nachfolgend dargestellt:



In der Arduino-Entwicklungsumgebung kann nun ein Sketch programmiert und hochgeladen werden.

Das eigene Gerät ist so gut wie fertig.

Im späteren Einsatz muss nun noch dafür gesorgt werden, dass der Mikroprozessor eine Spannungsversorgung bekommt. Hierfür eignen sich ICs wie der 7805 etc.

RGB-Sensor TCS 230

Der Sensor ist in niedrigen Preisbereichen erhältlich (und er arbeitet auch nicht sonderlich genau, genügt jedoch für Sortieraufgaben etc.). Er arbeitet mit einer 8x8-Matrix von Photodioden. Hiervon sind jeweils 16 mit einem Rot-, Grün bzw. Blau-Filter ausgestattet. Die letzten 16 Photodioden sind ohne Filter. Über eine 2bit-Kombination kann man die unterschiedlichen Dioden ansteuern. Das Ergebnis der Messung kann über eine Frequenz am Pin OUT ausgelesen werden. Diese Frequenz wird vorher über weitere 2 Bits voreingestellt bzw. skaliert.

Die Skalierung erfolgt nach der anschließenden Tabelle:



Abbildung 1:
http://img.dxcdn.com/productimage/sku_216448_3.jpg

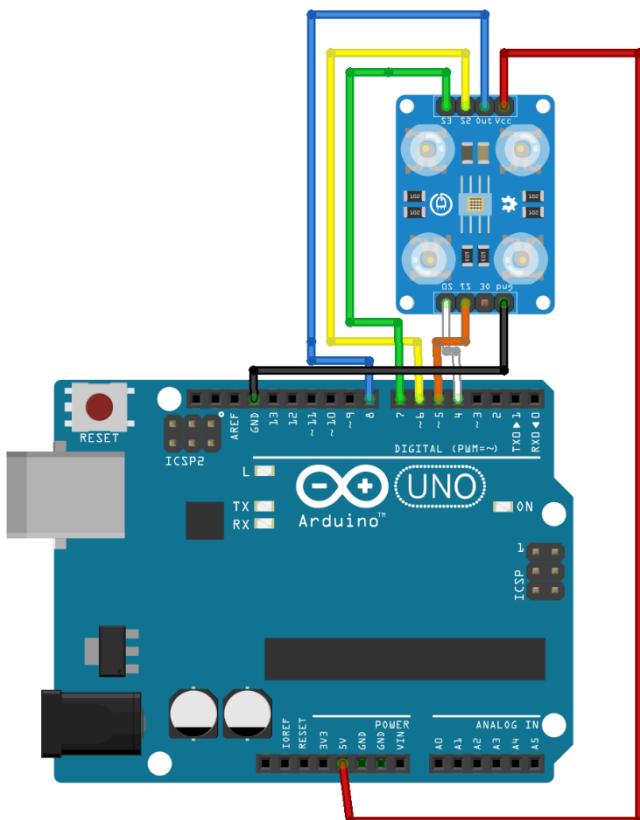
S0	S1	Frequenzskalierung
LOW	LOW	Sensor ausschalten
LOW	HIGH	2%
HIGH	LOW	20%
HIGH	HIGH	100%

Für den Arduino ist eine Frequenzskalierung von 20% ideal.

Die einzelnen Photodioden werden nach der Folgenden Tabelle ausgewählt:

S2	S3	Photodioden
LOW	LOW	rot
LOW	HIGH	blau
HIGH	LOW	unfiltriert
HIGH	HIGH	grün

Der Sensor wird wie folgend angeschlossen:



```
#define S0 4
#define S1 5
#define S2 6
#define S3 7
#define sensorOut 8

int frequenz = 0;

void setup() {
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(sensorOut, INPUT);

  // Frequenzskalierung auf 20%
  digitalWrite(S0,HIGH);
  digitalWrite(S1,LOW);
```

```
Serial.begin(9600);  
}  
  
void loop() {  
    // rote Dioden auswählen  
    digitalWrite(S2,LOW);  
    digitalWrite(S3,LOW);  
    // Zeitmessung  
    frequenz = pulseIn(sensorOut, LOW);  
    //Umsetzung des gemessenen Frequenzbereichs auf Werte von 0 bis 255  
    //(experimentell!!)  
    frequenz = map(frequenz, 75,190,255,0);  
  
    Serial.print("R= ");  
    Serial.print(frequenz);  
    Serial.print(" ");  
    delay(100);  
  
    // grüne Dioden auswählen  
    digitalWrite(S2,HIGH);  
    digitalWrite(S3,HIGH);  
    // Frequenz einlesen  
    frequenz = pulseIn(sensorOut, LOW);  
    //Remaping  
    frequenz = map(frequenz, 150,312,255,0);  
  
    Serial.print("G= ");  
    Serial.print(frequenz);  
    Serial.print(" ");  
    delay(100);  
  
    // blaue Dioden auslesen  
    digitalWrite(S2,LOW);  
    digitalWrite(S3,HIGH);  
    // Frequenz einlesen  
    frequenz = pulseIn(sensorOut, LOW);  
    //Remaping  
    frequenz = map(frequenz, 80,250,255,0);  
  
    Serial.print("B= ");  
    Serial.print(frequenz);  
    Serial.println(" ");  
    delay(500);  
}
```

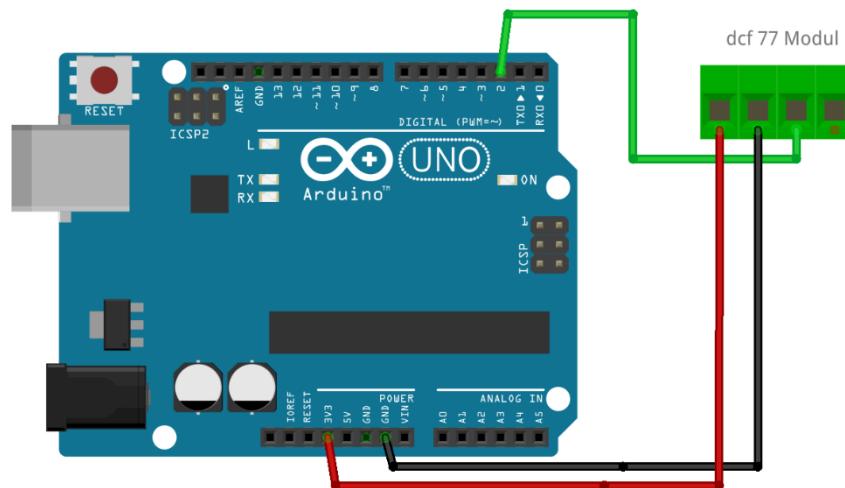
Hinweis:

Der Befehl `map(wert, vonNiedrig, vonHoch, zuNiedrig, zuHoch)` wandelt einen Wert aus dem Bereich `vonNiedrig` bis `vonHoch` in einen vom Bereich `zuNiedrig` bis `zuHoch` um. Dies bietet sich hier an, da der Sensor reziprok arbeitet: höhere Rotwerte liefern kleinere Wert am Ausgang usw. Die Wert, welche im `map`-Befehl ausgewählt sind, werden experimentell ermittelt.

Zeitermittlung mit dem DCF-77 – Protokoll

Das Modul, erhältlich beispielsweise bei Pollin, empfängt das Echtzeitsignal, welches von Mainflingen bei Frankfurt/Main über eine Frequenz von 77,5 kHz gesendet wird. Das Signal wird einmal pro Sekunde gesendet. Das genaue Protokoll ist vielfältig nachzulesen. Hier wird eine fertige Bibliothek verwendet.

Das Modul wird wie folgt verdrahtet:



Hinweis:

Beachten Sie bitte, dass das Modul mit 3,3V arbeitet. Eine höhere Spannung kann das Modul zerstören.

Im nachfolgenden Quelltext wird die Uhrzeit und das Datum ausgelesen und auf die serielle Schnittstelle geschrieben:

```
#include "DCF77.h"
#include "Time.h"

#define DCF_PIN 2           // Connection pin to DCF 77 device
#define DCF_INTERRUPT 0     // Interrupt number associated with pin

time_t time;
// Non-inverted input on pin DCF_PIN
DCF77 DCF = DCF77(DCF_PIN, DCF_INTERRUPT, true);

void setup() {
  Serial.begin(9600);
  DCF.Start();
  Serial.println("Waiting for DCF77 time ...");
}
Serial.println("It will take at least 2
minutes before a first time update.");
}

void loop() {
  delay(1000);
}
```



```

time_t DCFtime = DCF.getTime(); // Check if new DCF77 time is available
if (DCFtime!=0)
{
    Serial.println("Time is updated");
    setTime(DCFtime);
}
digitalClockDisplay();
}

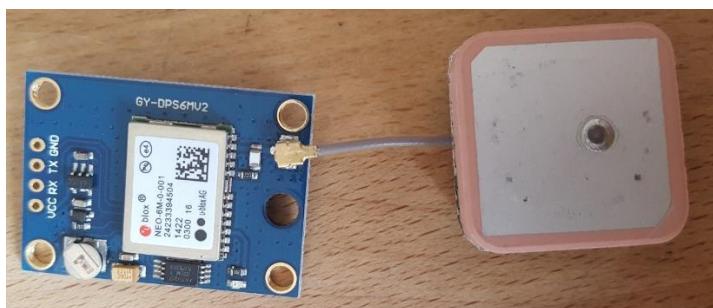
void digitalClockDisplay(){
    // digital clock display of the time
    Serial.print(hour());
    printDigits(minute());
    printDigits(second());
    Serial.print(" ");
    Serial.print(day());
    Serial.print(" ");
    Serial.print(month());
    Serial.print(" ");
    Serial.print(year());
    Serial.println();
}

void printDigits(int digits){
    // utility function for digital clock display: prints preceding colon and leading
0
    Serial.print(":");
    if(digits < 10)
        Serial.print('0');
    Serial.print(digits);
}

```

GPS

Das GPS-Modul Neo-6m ist momentan für ca. 10€ erhältlich. Es wird über eine serielle Schnittstelle angesprochen. Da die Bibliothek `SoftwareSerial` eine solche simuliert, kann die eigentliche serielle Schnittstelle des Arduino für die Kommunikation mit dem PC weiterverwendet werden. Zur Auswertung der GPS-Daten dient die Bibliothek `TinyGPS++`.



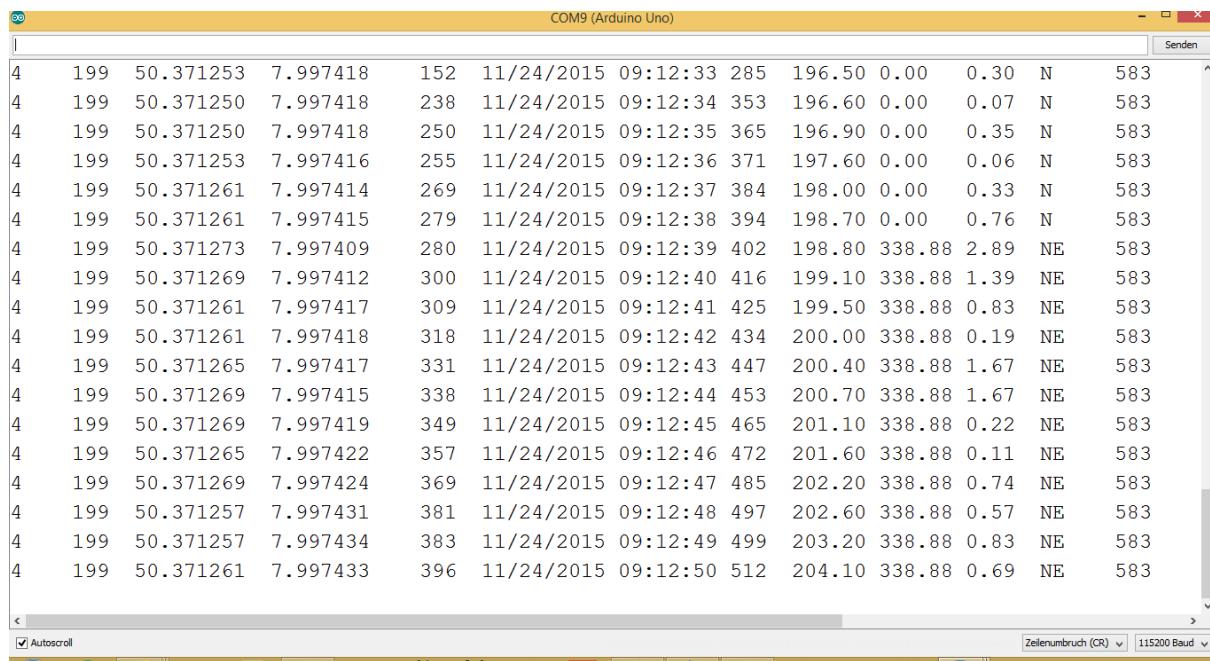
Das GPS-Modul wird wie folgt angeschlossen:

Arduino	GPS
5V	VCC

GND	GND
Pin 3	RX
Pin 4	TX

Andere Beschaltungen sind nicht notwendig.

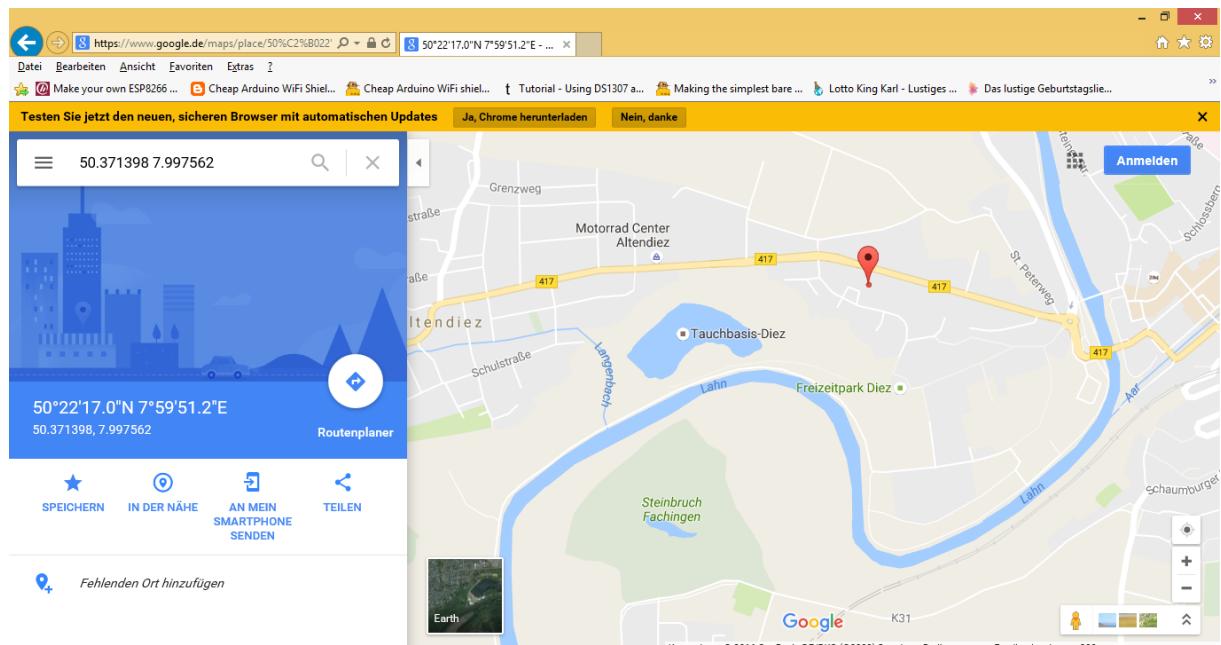
Im Beispielprogramm werden dann die Satellitendaten ausgelesen und auf den seriellen Monitor geschrieben. Die Bibliothek wird hier nicht näher erläutert, die einzelnen Befehle können aus dem Beispielprogramm entnommen werden. Es sind mindestens vier empfangbare Satelliten erforderlich. Die nachfolgende Ausgabe wurde in Diez aufgenommen:



```
COM9 (Arduino Uno)
[Senden]
4 199 50.371253 7.997418 152 11/24/2015 09:12:33 285 196.50 0.00 0.30 N 583
4 199 50.371250 7.997418 238 11/24/2015 09:12:34 353 196.60 0.00 0.07 N 583
4 199 50.371250 7.997418 250 11/24/2015 09:12:35 365 196.90 0.00 0.35 N 583
4 199 50.371253 7.997416 255 11/24/2015 09:12:36 371 197.60 0.00 0.06 N 583
4 199 50.371261 7.997414 269 11/24/2015 09:12:37 384 198.00 0.00 0.33 N 583
4 199 50.371261 7.997415 279 11/24/2015 09:12:38 394 198.70 0.00 0.76 N 583
4 199 50.371273 7.997409 280 11/24/2015 09:12:39 402 198.80 338.88 2.89 NE 583
4 199 50.371269 7.997412 300 11/24/2015 09:12:40 416 199.10 338.88 1.39 NE 583
4 199 50.371261 7.997417 309 11/24/2015 09:12:41 425 199.50 338.88 0.83 NE 583
4 199 50.371261 7.997418 318 11/24/2015 09:12:42 434 200.00 338.88 0.19 NE 583
4 199 50.371265 7.997417 331 11/24/2015 09:12:43 447 200.40 338.88 1.67 NE 583
4 199 50.371269 7.997415 338 11/24/2015 09:12:44 453 200.70 338.88 1.67 NE 583
4 199 50.371269 7.997419 349 11/24/2015 09:12:45 465 201.10 338.88 0.22 NE 583
4 199 50.371265 7.997422 357 11/24/2015 09:12:46 472 201.60 338.88 0.11 NE 583
4 199 50.371269 7.997424 369 11/24/2015 09:12:47 485 202.20 338.88 0.74 NE 583
4 199 50.371257 7.997431 381 11/24/2015 09:12:48 497 202.60 338.88 0.57 NE 583
4 199 50.371257 7.997434 383 11/24/2015 09:12:49 499 203.20 338.88 0.83 NE 583
4 199 50.371261 7.997433 396 11/24/2015 09:12:50 512 204.10 338.88 0.69 NE 583
```

Autoscroll Zeilenumbruch (CR) 115200 Baud

Gibt man die Koordinaten bei Google ein, kann man den Standort ermitteln:



Stimmt exakt ☺

Wichtig:

Stellen Sie die Übertragungsrate zum GPS-Modul (nicht die zum seriellen Monitor) auf 9600 Baud ein.
Hiermit haben sich gute Ergebnisse erzielen können.

LCD-Keypad-Shield

Das LCD-Keypad-Shield ist mittlerweile für ca. 6€ bei den einschlägigen Internethändlern zu erwerben. Es bietet den Vorteil, dass sowohl LCD-Display (2 x 16) sowie 5 Taster direkt auf den Arduino aufgesteckt werden können. Eine weitere Verdrahtung erübrigts sich. Nachteil des Shields ist es, dass die nicht verwendeten Pins des Arduino nicht über Steckbuchsen herausgeführt werden. Ein weiterer Nachteil ist, dass die fünf Taster gleichzeitig über einen analogen Eingang (A0) eingelesen werden. Die Taster sind so beschaltet, dass sie jeweils eine Reihe von Widerständen, angeschlossen an der Versorgungsspannung, kurzschließen. Dies bedeutet auch, dass es nicht möglich ist, eine Kombination von Tasten auszulesen. Weiterhin schließen einzelne Taster andere kurz. Drückt man beispielsweise den Taster „RIGHT“, so können alle anderen Taster gedrückt werden, es ändert nichts am Ergebnis (siehe Schaltplan unter dem Quelltext).



Quelltext¹⁶:

```
#include <LiquidCrystal.h>           //Sample using LiquidCrystal library

LiquidCrystal lcd(8, 9, 4, 5, 6, 7); // select the pins used on the LCD panel

// define some values used by the panel and buttons
#define btnRIGHT 0
#define btnUP    1
#define btnDOWN  2
#define btnLEFT  3
#define btnSELECT 4
```

¹⁶ http://www.alhin.de/arduino/arduino_code/LCD_KEYPAD_TEST0.ino

```
#define btnNONE      5
int lcd_key        = 0;
int adc_key_in    = 0;

int read_LCD_buttons()                                //function for detection of pressed
keypad button
{
    adc_key_in = analogRead(0);                      // read the analog value from the sensor
    if (adc_key_in > 1000) return btnNONE;           // We make this the 1st option for speed
reasons since it will be the most likely result
    if (adc_key_in < 50)   return btnRIGHT;
    if (adc_key_in < 195)  return btnUP;
    if (adc_key_in < 380)  return btnDOWN;
    if (adc_key_in < 555)  return btnLEFT;
    if (adc_key_in < 790)  return btnSELECT;
    return btnNONE; // when all others fail, return this...
}

void setup()
{
    lcd.begin(16, 2);                               // start the LCD library
    lcd.setCursor(0,0);                            // set cursor position at start
}

void loop()
{
    lcd.clear();

    lcd.setCursor(0,0);
    lcd.print(analogRead(0));

    lcd.setCursor(7,0);
    lcd_key = read_LCD_buttons(); // read the buttons function
    switch (lcd_key)                  // depending on which button was pushed, we perform
an action
    { case btnRIGHT:
        { lcd.print("RIGHT ");
          break;
        }
    case btnLEFT:
        { lcd.print("LEFT   ");
          break;
        }
    case btnUP:
        { lcd.print("UP     ");
          break;
        }
    case btnDOWN:
        { lcd.print("DOWN   ");
          break;
        }
    case btnSELECT:
        { lcd.print("SELECT");
          break;
        }
    case btnNONE:
        {

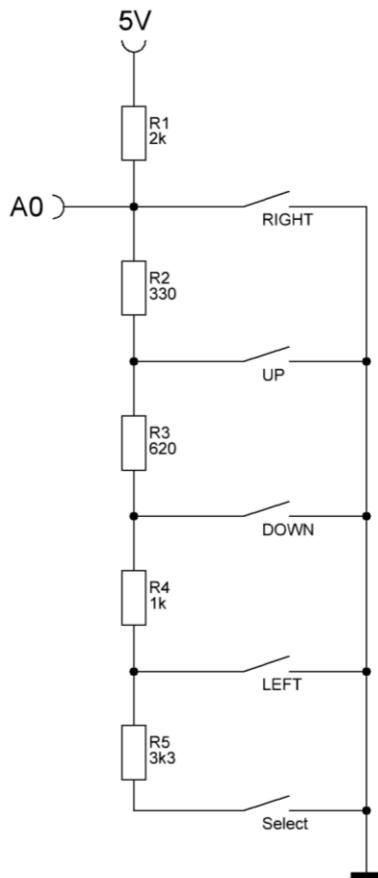
    
```

```

    { lcd.print("NONE  ");
      break;
    }
}
delay(40); //wait 40ms
}

```

Schaltplan des LCD-Keypad-Shields, Belegung der Taster:



LED&Keys

Das fertige Modul beinhaltet 8 7-Segmentanzeigen sowie 8 Taster. Die Steuerung erfolgt über den Chip TM 1638. Dies ist ein LED-Driver, der insgesamt eine Matrix von 10 x 8 LEDs ansteuern kann, oder aber 8 Multi-Led-Module. Weiter kann er an 3 Eingängen eine Tastenmatrix einlesen, somit ist also die zusätzliche Abfrage von 8 Tastern möglich (oder aber alle Kombinationen von 3 Tastern). Das Modul ist derzeit für teilweise unter 2€ zu erwerben und eignet sich daher als günstiges Anzeige- oder Eingabemodul für Projekte.

Aufgabensammlung

Aufgaben: Einfache Ein- / Ausgabe

Aufgabe 1 Einfache Ein- / Ausgabe

Erstellen Sie ein Programm mit folgender Funktionalität:

Eine LED wird über einen Widerstand an einem der Ausgänge angeschlossen. Die LED soll nach dem Start des Programms mit einer Frequenz von 10 Hertz blinken.

Aufgabe 2 Einfache Ein- / Ausgabe

Modifizieren Sie Ihr Programm Blink so, dass die LED immer schneller blinkt.

Aufgabe 3 Einfache Ein- / Ausgabe

Erstellen Sie ein Programm mit folgender Funktionalität:

Simulieren Sie mit drei LEDs eine Verkehrsampel mit den Phasen: Rot, Rot Gelb, Grün, Gelb Rot.

Aufgabe 4 Einfache Ein- / Ausgabe

Erstellen Sie ein Programm mit folgender Funktionalität:

Wird ein Taster (angeschlossen an Pin 8) gedrückt, so leuchtet eine LED (angeschlossen an Pin 2).

Wird der Taster losgelassen, erlischt die LED.

Aufgabe 5 Einfache Ein- / Ausgabe

Erstellen Sie ein Programm mit folgender Funktionalität:

Wird ein Taster gedrückt, so blinkt eine LED fünfmal mit einer Frequenz von 2 Hertz.

Aufgabe 6 Einfache Ein- / Ausgabe

Erstellen Sie ein Programm mit folgender Funktionalität:

Wird ein Taster gedrückt, so blinkt eine rote LED. Wird der Taster nicht gedrückt, so blinkt eine grüne LED. Als Ausgänge fungieren die Pins 11 und 12. Pin 2 ist als Eingang zu konfigurieren.

Aufgabe 7 Einfache Ein- / Ausgabe

Erstellen Sie ein Programm mit folgender Funktionalität:

Simulieren Sie eine Verkehrsampel mit Fußgängerampel. Wird ein Taster gedrückt, soll die Verkehrsampel zur Phase Rot wechseln und die Fußgängerampel zur Phase Grün. Nach 5 Sekunden dürfen die Autofahrer weiterfahren.

Aufgabe 8 Einfache Ein- / Ausgabe

Erstellen Sie ein Programm mit folgender Funktionalität:

Der Zustand einer LED (an / aus) soll sich per Druck auf einen Taster ändern.

Aufgabe 9 Einfache Ein- / Ausgabe

Ändern Sie Aufgabe 8 so ab, dass das Programm mit einer Flankenänderung arbeitet. Nur bei einer positiven Flanke soll sich der Zustand der LED ändern.

Aufgabe 10 Einfache Ein- / Ausgabe

Erstellen Sie ein Programm, welches die Anzahl der Tastendrucke in einer Variablen hochzählt. Diese Anzahl soll einmal auf die serielle Schnittstelle ausgegeben werden. Gleichzeitig sollen 4 LEDs die Anzahl binär anzeigen!

Aufgaben Kontrollausgaben

Aufgabe 1 zu Kontrollausgaben

Geben Sie einmal den Text "Hallo Arduino" auf die serielle Schnittstelle aus und lassen Sie sich die Ausgabe vom seriellen Monitor anzeigen.

Hinweis:

Der folgende Quellcode stellt eine Endlosschleife ohne Funktion dar:

```
while(1) {}
```

Aufgabe 2 zu Kontrollausgaben

Geben Sie das 10er-Einmaleins auf die serielle Schnittstelle in folgender Form aus:

$1 \times 10 = 10$

$2 \times 10 = 20$

usw.

$10 \times 10 = 100$

Nach der Ausgabe des letzten Ausdrucks soll keine weitere Ausgabe mehr erfolgen.

Aufgabe 3 zu Kontrollausgaben

Ändern Sie Aufgabe 7 zu einfache Ein- / Ausgaben so ab, dass auf der seriellen Schnittstelle der jeweilige Zustand von Fahrer (Fahren, Achtung, Stopp) und Fußgänger (Gehen, Stopp) angezeigt wird.

Aufgaben Einlesen analoger Werte

Aufgabe 1 zum Einlesen analoger Werte:

Schließen Sie an den Arduino (Pin A0) den Mittelabgriff eines Potentiometers an. Legen Sie den oberen Abgriff an die Spannung 5V und den unteren Abgriff an GND. Erstellen Sie ein Programm, welches den an Pin A0 eingelesenen Wert ständig an die serielle Schnittstelle ausgibt. Kontrollieren Sie Ihr Ergebnis über den seriellen Monitor.

Aufgabe 2 zum Einlesen analoger Werte:

Wandeln Sie das Programm aus Aufgabe 1 so ab, dass der Wert der eingelesenen Spannung an die serielle Schnittstelle ausgegeben wird. Bedenken Sie, dass dieser Wert ein Kommawert sein kann!

Aufgabe 3 zum Einlesen analoger Werte:

Ein Bargraph soll realisiert werden. Ein Bargraph ist eine Anzeige, die per farbige LEDs den analogen Wert einer Spannung angibt. Beispielsweise werden solche Anzeigen bei Soundanlagen oder Mischpulten verwendet. Der Bargraph besteht aus 6 LEDs (2x grün, 2 x gelb, 2 x rot). Gemessen wird

eine Spannung am analogen Eingang 0. Die Spannung soll über ein Potentiometer abgeändert werden können. Je größer die gemessene Spannung ist, desto mehr LEDs (von grün nach rot) leuchten.

Aufgaben zu den analogen Ausgabepins:

Aufgabe 1 zu den analogen Ausgabepins:

Geben Sie eine Spannung von ca. 2,5V an dem Pin 3 aus. Lesen Sie diese Spannung wieder an dem analogen Eingabepin A0 ein und geben Sie den eingelesenen Wert an die serielle Schnittstelle aus. Kontrollieren Sie Ihr Ergebnis mittels des seriellen Monitors.

Aufgabe 2 zu den analogen Ausgabepins (Dimmer):

Lesen Sie den Wert am Mittelabgriff eines Potis an Pin A0 ein. Steuern Sie mit diesem Wert die Helligkeit einer LED. Die LED soll über Pin 3 mit Spannung versorgt werden. Denken Sie an den Vorwiderstand!

Aufgabe 3 zu den analogen Ausgabepins (Signalgenerator I):

Am analogen Ausgabepin 3 soll eine definierte Spannung ausgegeben werden. Wird ein Taster einmal gedrückt, so soll am Ausgabepin eine Dreiecksspannung ausgegeben werden. Wird der Taster erneut gedrückt, so soll eine Sinusspannung ausgegeben werden. Der Taster schaltet also immer wieder zwischen den beiden Spannungsarten um. Das Ergebnis kann mittels eines Oszilloskops visualisiert werden.

Aufgabe 4 zu den analogen Ausgabepins (Signalgenerator II):

Ändern Sie Aufgabe 3 so ab, dass Sie zusätzlich mit einem Potentiometer die Frequenz der Ausgabespannung verändern können.

Aufgaben zum LDR

Aufgabe 1 zum LDR

Erstellen Sie Schaltung und Programm, so dass Sie den an einem LDR gemessenen Spannungswert an die serielle Schnittstelle ausgeben. Verwenden Sie die im Skript dargestellte Reichenschaltung. Als Widerstand bringt ein 220Ω-Widerstand brauchbare Ergebnisse. Testen Sie Ihre Schaltung auch mit anderen Widerstandswerten.

Aufgabe 2 zum LDR (Nachtlicht)

Eine LED soll in Abhängigkeit der Außenhelligkeit leuchten. Ist es außen dunkel, so leuchtet die LED stark, ist es hell, so leuchtet die LED nicht.

Aufgaben zum Temperatursensor

Aufgabe 1 zum Temperatursensor

Erstellen Sie Schaltung und Programm, so dass Sie den Spannungswert am Datenausgang eines LM35 einlesen an die serielle Schnittstelle ausgeben. Verwenden Sie hierbei als Versorgungsspannung die interne analoge Referenzspannung am Pin AREF. Probieren Sie mehrere Referenzspannungen für AREF aus (mittels der Funktion analogReferences() konnte man diese Spannung umschalten → siehe oben). Ermitteln Sie für eine Referenzspannung den Datenausgangswert des LM35 bei Raumtemperatur (angenommen werden 20°).

Aufgabe 2 zum Temperatursensor

Erstellen Sie ein Programm Thermometer, welches die aktuell gemessene Temperatur als String an die serielle Schnittstelle ausgibt.

Aufgaben zum Tiltsensor

Aufgabe 1 zum Tiltsensor

Erstellen Sie Schaltung und Programm, so dass eine LED leuchtet, sobald der Sensor schließt.

Aufgabe 2 zum Tiltsensor

Erstellen Sie ein Programm, so dass eine rote LED beim Schließen des Sensors und eine grüne LED beim Öffnen des Sensors leuchtet.

Aufgabe 3 zum Tiltsensor (Pedometer)

Erstellen Sie ein Programm, welches die Schließ- und Öffnungsvorgänge des Sensors zählt und diese Zahl an die serielle Schnittstelle ausgibt.

Aufgabe 4 zum Tiltsensor (Pedometer)

Modifizieren Sie Schaltung und Programm aus Aufgabe 3 so um, dass die Anzahl als binäre Zahl mit LEDs (max. 6 LEDs) ausgegeben wird.

Aufgabe zu Aktoren

Aufgaben zur RGB-LED

Aufgabe 1 zur RGB-LED

Erstellen Sie ein Programm und eine Schaltung, welches die RGB-LED in allen "binären" Farben leuchten lässt. Mit "binären" Farben ist gemeint, dass jede LED immer mit HIGH- oder LOW-Signal des jeweiligen digitalen Ausgangs angesteuert wird, und alle Kombinationen der drei Ausgänge vorkommen sollen (also insgesamt 7 verschiedene Farben + LED-Aus).

Aufgabe 2 zur RGB-LED

Steuern Sie nun die drei LEDs analog an. Verändern Sie den analogen Ausgangswert und beobachten Sie, welche Farbverläufe entstehen. Verwenden Sie hierzu eine selbstgeschriebene Funktion: `void setRGBLED(rot: int, gruen: int, blau: int)`.

Aufgaben zum Summer

Aufgabe 1 zur Summer

Erstellen Sie ein Programm mit folgender Funktionalität: Wird ein Taster gedrückt, so erzeugt der Summer einen Ton. Wird der Taster wieder losgelassen, so verstummt der Ton.

Aufgabe 2 zur Summer

Erstellen Sie ein Programm, welches eine komplette Tonleiter hoch und wieder hinunter abspielt. Anschließend soll kein Ton mehr produziert werden. Hinweise, die folgende Tabelle hilft Ihnen bei der Umsetzung von Tönen und Frequenzen:

Ton	C'	C#'	D'	D#'	E'	F'	F#'	G'	G#'	A'	A#'	H'	C"
-----	----	-----	----	-----	----	----	-----	----	-----	----	-----	----	----

Frequenz	262	277	294	311	330	349	370	392	415	440	466	494	523
----------	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Aufgaben zum Servo

Aufgabe 1 zur Servo

Erstellen Sie ein Programm, welches einen Servo von links nach rechts und wieder zurück fahren lässt.

Aufgabe 2 zur Servo

Modifizieren Sie das Programm / die Schaltung aus Aufgabe 1 so, dass Sie die Position des Servos mittels eines Potentiometers steuern.

Aufgaben zum Transistor

Aufgabe 1 zum Transistor

Modifizieren Sie die Schaltung (über einen Transistor wird eine Diode geschaltet), so, dass der Arduino die Funktion des Schalters übernimmt. Das Programm soll die Diode mit einer Frequenz von 1Hz blinken lassen.

Aufgabe 2 zum Transistor

Ersetzen Sie in der Schaltung aus Aufgabe 1 den Basisvorwiderstand durch ein Potentiometer und beschreiben Sie, was sich bei Einstellungen am Poti ändert.

Aufgaben zum Relais

Aufgabe 1 zum Relais

Entwickeln Sie eine Schaltung / ein Programm mit dem Sie mittels eines Tasters ein Relais ein- und ausschalten können. Sofern Sie eine Last wie beispielsweise einen Gleichstrommotor o.ä. verwenden, nutzen Sie als Versorgungsspannung der Last ein externes Netzteil.

Aufgaben zur Digitaltechnik

Aufgabe 1 zur Digitaltechnik

Erstellen Sie eine Schaltung und ein Sketch mit folgender Funktionalität:

Der Motor einer Blechstanze wird nur dann betätigt, wenn mit beiden Händen gleichzeitig jeweils ein Taster gedrückt wird. Der Motor fährt in seine Ausgangslage automatisch wieder zurück und kann erst wieder neu gestartet werden, wenn die Taster beide losgelassen werden und wieder neu zusammen gedrückt werden. Zur Darstellung des Motors soll ein Servo verwendet werden. Dieser dreht bei Betätigung auf 90°, wartet eine Sekunde und fährt dann wieder in seine Ausgangsposition zurück.

Aufgabe 2 zur Digitaltechnik

Simulieren Sie eine Wechselschaltung mittels eines Arduino-Sketchs. Zwei Taster schalten eine LED ein bzw. aus. Die Taster sollen hierbei beide die LED ausschalten, wenn diese vorher geleuchtet hat und beide die LED einschalten, sofern diese vorher ausgeschaltet war. Beachten Sie, dass die Taster entprellt werden müssen!

Aufgabe 3 zur Digitaltechnik

Vier Verbraucher dürfen zusammen eine Leistungsaufnahme von 10000W nicht überschreiten. Verbraucher 1 hat eine Leistungsaufnahme von 5kW, Verbraucher 2 3kW, Verbraucher 3 4kW und Verbraucher 4 2kW. Wird die Gesamtleistung von 10000W genau erreicht, soll eine gelbe LED leuchten. Bei einer höheren Leistungsaufnahme soll eine rote LED leuchten. Realisieren Sie Schaltung und Programm zu dieser Aufgabenstellung. Die Verbraucher werden durch Taster dargestellt.

Aufgabe 4 zur Digitaltechnik

Ein Halbaddierer soll simuliert werden. Als Eingänge dienen zwei Taster. Als Ausgang dienen die serielle Schnittstelle und zwei LEDs. Die Ausgabe an die serielle Schnittstelle soll wie folgt lauten (a,b,x,y sind die entsprechenden Werte): Eingang A: a Eingang B: b Einstelliger Wert der Addition: x Uebertrag: y. Die rote LED stellt den Übertrag dar, eine grüne den einstelligen Wert der Addition.

Aufgaben zu Interrupts

Aufgabe 1 zu Interrupts

Ändern Sie das Programm aus dem Kapitel Interrupts so ab, dass er Interrupt nur bei aufsteigender / abfallender Flanke ausgelöst wird.

Aufgabe 2 zu Interrupts

Erstellen Sie Programm / Schaltung mit folgender Funktionalität:

Eine Verkehrsampel (rot, gelb grün) läuft ständig korrekt ab. Wird der Taster einer Fußgängerampel gedrückt, so wechselt die Verkehrsampel korrekt auf rot, die Fußgängerampel (rot, grün) lässt den Fußgänger passieren und die Verkehrsampel nimmt anschließend ihren Betrieb wieder auf.

Aufgabe 3 zu Interrupts

Erstellen Sie Programm / Schaltung mit folgender Funktionalität:

Ein Programm läuft in einer Endlosschleife. Wird Interrupt 0 per Taster ausgelöst, so leuchtet eine LED an Pin 13. Interrupt 1 wird scharf gemacht, Interrupt 0 wird abgeschaltet. Wird anschließend Interrupt 1 ausgelöst, so wird die LED wieder ausgeschaltet, Interrupt 1 wird abgeschaltet und Interrupt 0 wird scharf gemacht.

Aufgabe 4 zu Interrupts

Erstellen Sie Programm / Schaltung "Treppenhauslicht" mit folgender Funktionalität:

Im Treppenhaus gibt es 2 Leuchten. Durch Druck auf einen von zwei Tastern werden die Leuchten für 10 Sekunden aktiviert. In dieser Zeit ist ein Tastendruck auf beiden Tastern ohne Funktion. Nach den 10 Sekunden erlöschen die Leuchten wieder.

Aufgaben zum Schrittmotor

Aufgabe 1 zum Schrittmotor

Recherchieren Sie im Internet die Schrittabfolge eines bipolaren Motors im Halbschrittbetrieb. Bringen Sie Ihren Motor im Halbschrittbetrieb zum Laufen.

Aufgabe 2 zum Schrittmotor

Ergänzen Sie Aufgabe 1 so, dass Sie die Geschwindigkeit des Motors mit einem Potentiometer ändern können.

Aufgabe 3 zum Schrittmotor

Ändern Sie Aufgabe 2 so ab, dass Sie über ein Potentiometer nicht nur die Geschwindigkeit, sondern auch die Laufrichtung ändern können. Steht das Potentiometer in der Mitte, so hält der Motor an. Wird das Potentiometer dann nach rechts gedreht, so läuft der Motor rechts herum, das Potentiometer erhöht durch weitere Rechtsdrehung die Geschwindigkeit. Wird das Poti nach Mittelstellung nach links gedreht, läuft der Motor links herum. Ebenso soll hier die Geschwindigkeit reguliert werden können.

Aufgaben zur Zeit- / Entfernungsmessung

Aufgabe 1 zur Zeit - / Entfernungsmessung

Bauen Sie einen Radarwarner. Nähert sich ein Gegenstand näher als 10cm an den Sensor, so soll ein pulsierender Piepton ertönen. Liegt der Abstand näher als 5cm, so soll ein Dauerton hierauf hinweisen.

Aufgaben zum IR-Empfänger

Aufgabe 1 zum IR-Empfänger

Nehmern Sie die Signale einer IR-Fernbedienung auf und geben Sie deren Codes auf der seriellen Schnittstelle aus.

Aufgabe 2 zum IR-Empfänger

Steuern Sie mit einer Fernbedienung den Zustand dreier unterschiedlicher LEDs.

Aufgabe 3 zum IR-Empfänger

Steuern Sie mit einer Fernbedienung Laufrichtung und Geschwindigkeit eines Schrittmotors.

Aufgaben zum LCD-Display

Aufgabe 1 zum LCD-Display

Bringen Sie das Beispielprogramm zum Laufen. Experimentieren Sie mit den im Text erklärten Befehlen.

Aufgabe 2 zum LCD-Display

Erstellen Sie ein Arduino-Programm, welches auf dem LCD-Display den Text "Hallo" von links nach rechts laufen lässt.

Aufgabe 3 zum LCD-Display

Modifizieren Sie das Programm aus Aufgabe 2 so, dass der Text über die Zeilen hinaus und wieder von vorne beginnend läuft.

Aufgabe 4 zum LCD-Display

Stellen Sie den Spannungswert, der am analogen Eingang A0 gemessen wird, mit der entsprechenden Einheit auf dem LCD-Display dar. Die gemessene Spannung soll über ein Potentiometer einstellbar sein.

Aufgaben zum SPI-Bus

Aufgabe 1 zum SPI-Bus

Steuern Sie mit dem MCP die Helligkeit einer LED.

Aufgaben zum MAX 7219

Aufgabe 1 zum MAX 7219

Lassen Sie auf einer 5x7-Matrix den Text: Arduino ablaufen.

Aufgabe 2 zum MAX 7219

Stellen Sie auf der vierfach 7-Segmentanzeige einen Spannungswert dar. Dieser soll über ein Potentiometer über einen Analogen Eingang abgenommen werden.

Aufgaben zum I²C-Bus

Aufgabe 1 zum I²C-Bus

Erstellen Sie Schaltung und Programm, so dass Sie den Mittelabgriff eines Potentiometers am Kanal 0 des PCF 8591 auslesen und an die serielle Schnittstelle ausgeben.

Aufgabe 2 zum I²C-Bus

Auf der Lernplattform finden Sie das Datenblatt des MCP 23017 (16 Bit Porterweiterung). Erstellen Sie Funktionen zum Ausgeben eines Bytes auf dem IC und zum Einlesen eines Bytes. Testen Sie das Einlesen, indem Sie per Jumperkabel 0 und 5V auf die Eingänge geben bzw. die Spannungen an den Ausgängen des Bausteines messen.

Aufgabe 3 zum I²C-Bus

Ändern Sie das Programm zum MPU 6050 so ab, dass Winkelgeschwindigkeit (°/s), Temperatur (° Celsius) und Beschleunigung (g) in den entsprechenden Einheiten auf der seriellen Schnittstelle ausgegeben werden.

Aufgabe 4 zum I²C-Bus

Bauen Sie mit dem MCP 23017 und einem ULN 2803 ein IO-Interface auf. Es sollen 8 LEDs als Ausgang angesprochen werden können. Die übrigen 8 Bits des MCPs sollen als Eingang fungieren.

Aufgaben zum Ethernetsield

Aufgabe 1 zum Ethernetsield

Erstellen Sie ein Programm auf dem Arduino, welches diesen als Steuerverserver funktionieren lässt. Bekommt der Arduino vom Client den Befehl: ,t‘, so wird eine LED an Pin 7 (bitte an Vorwiderstand

denken!) angeschaltet. Kommt der Befehl „f“, so wird die LED wieder ausgeschaltet. Bei „q“ wird die Ethernetverbindung unterbrochen.

Aufgaben zum Bewegungsmelder PIR:

Aufgabe 1 zum Bewegungsmelder

Der Bewegungsmelder soll bei einer Wahrnehmung einen Piezo ansteuern.

Aufgaben zu RFID:

Aufgabe 1 zu RFID

Schreiben Sie ein Programm, mit dem Sie eine LED über einen RFID-Transponder steuern können.

Aufgabe 2 zu RFID

Schreiben Sie ein Programm, welches einen Servo über zwei RFID-Transponder jeweils um 90° dreht (Transponder 1 → +90°, Transponder 2 → -90°).

Aufgaben zum RGB-Sensor TCS320:

Aufgabe 1 zu RFID

In Abhängigkeit der gemessenen Farbe soll jeweils eine andere angeschlossene LED leuchten.

Aufgabe 2 zu RFID

Bauen Sie mittels des Sensors und einer RGB-LED ein Chamäleon.

Aufgaben zum DCF-77:

Aufgabe 1 zu DCF-77

Bauen Sie einen Wecker, der einen Piezolautsprecher zu einer bestimmten Uhrzeit piepsen lässt.

Aufgabe 2 zu DCF-77

Bauen Sie einen neuen Pausengong für Ihre Schule

Aufgaben zum GPS-Modul:

Aufgabe 1 zum GPS-Modul

Bauen Sie ein Gerät, welches Sie durch eine rote LED erkennen lässt, dass Sie in Ihrer Schule sind. Sind Sie wieder zu Hause, so soll eine grüne LED leuchten.

Aufgabe 2 zu GPS-Modul

Bauen Sie ein Gerät, welches die GPS-Koordinaten auf ein LCD-Display ausgibt.

Aufgaben zum LCD-Keypad-Shield:

Aufgabe 1 zum LCD-Keypad-Shield

Erstellen Sie ein Programm, welches zufällig eine der fünf Tastennamen auf dem Bildschirm ausgibt. Der Anwender muss dann die entsprechende Taste so schnell wie möglich drücken. Die Reaktionszeit wird gemessen und auf dem Display ausgegeben.

Hinweis: Die Funktion `random(x, y)` erzeugt eine Zufallszahl im Bereich von x bis y-1!

Projektideen

Kapazitätsmessung eines Kondensators

Widerstandsmessung

LED-Tester

Quellen:

Quelle 2: <http://creativecoding.uni-bayreuth.de/assets/Uploads/Bauteile/LEDs.png>

